CCCG 2021

Proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG 2021)

August 10-12, 2021
Dalhousie University
Halifax, Nova Scotia
Canada

Logo designed by Meng He

Preface

This volume contains the proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG 2021), which took place on August 10–12, 2021. It was originally schedule to be held on campus at Dalhousie University, in Halifax, Nova Scotia, Canada. However, due to the COVID-19 pandemic, it was organized as a virtual event.

We are grateful to the CCCG 2021 Program Committee members and external reviewers, for their effort and hard work on reviewing all submissions. Each submission was reviewed by at least three members of the program committee. The committee decided to accept 43 papers out of 56 papers submitted. We thank the authors of all submitted papers and all those who have registered to attend the conference. We especially thank the invited speakers: Dr. Pankaj K. Agarwal (the Memorial Lecture for Paul Erdős) and Dr. Elisabetta Matsumoto (the Joint Memorial Lecture for Hurtado and Toussaint). In addition, we are grateful for the support and assistance provided by the CCCG 2021 Local Organizing Committee.

We acknowledge the generous support from our sponsors: Elsevier, the Faculty of Computer Science at Dalhousie University, the Atlantic Association for Research in the Mathematical Sciences (AARMS), the Fields Institute for Research in Mathematical Sciences, and the Pacific Institute for the Mathematical Sciences (PIMS).

Meng He
Don Sheehy
CCCG 2021 Program Committee Co-Chairs

Sponsored by

ELSEVIER

DALHOUSIE UNIVERSITY
FACULTY OF COMPUTER SCIENCE

AARMS

FIELDS

PIMS

Invited Speakers

Pankaj K. Agarwal	Duke University
Elisabetta Matsumoto	Georgia Institute of Technology

Program Committee

Oswin Aichholzer	Graz University of Technology
Hugo Akitaya	Tufts University
Binay Bhattacharya	Simon Fraser University
Prosenjit Bose	Carleton University
David Bremner	University of New Brunswick
Hsien-Chih Chang	Dartmouth College
Jean-Lou De Carufel	University of Ottawa
David Eppstein	University of California, Irvine
Ruy Fabila-Monroy	Cinvestav
Brittany Terese Fasy	Montana State University
Marina Gavrilova	University of Calgary
Meng He (co-chair)	Dalhousie University
Akitoshi Kawamura	Kyoto University
Jason Morrison	University of Manitoba
Sharath Raghvendra	Virginia Tech
Suneeta Ramaswami	Rutgers University
Maria Saumell	The Czech Academy of Sciences & Czech Technical University in Prague
Don Sheehy (co-chair)	North Carolina State University
Michiel Smid	Carleton University
Bettina Speckmann	Eindhoven University of Technology
Katharine Turner	Australian National University
Yusu Wang	University of California, San Diego

Additional Reviewers

Carlos Alegría, Tetsuya Araki, Ahmad Biniaz, Mirela Damian, Ivan Tadeu Ferreira Antunes Filho, Fabrizio Frati, Younan Gao, Kirk Gardner, Andrei Gonczi, Joachim Gudmundsson, Benjamin Holmgren, Ramesh Jallu, Amirhossein Khameneh, Linda Kleist, Fabian Klute, Jesus Leanos, Christian Lindorfer, Jayson Lynch, Brad McCoy, David L. Millman, Debajyoti Mondal, Klara Mundilova, Soeren Nickel, Joachim Orthaber, Irene Parada, Daniel Perz, Pablo Pérez-Lantero, Edgardo Roldán-Pensado, Tadashi Sakuma, Anna Schenfisch, Jordan Schupbach, Siddharth Sheth, Rodrigo Silveira, Willem Sonke, Arthur van Goethem, Inmaculada Ventura Molina, Kevin Verbeek, Haitao Wang, Alexandra Weinberger, Aaron Williams

Local Organizers

Nathaniel Brown	Travis Gagie	Younan Gao
Meng He (chair)	Zhen Liu	Michael St Denis

(All at Dalhousie University)

Table of Contents

Wednesday, August 11

Near-Delaunay Metrics

Nathan van Beusekom* Kevin Buchin* Hidde Koerts*
Wouter Meulemans* Benjamin Rodatz† Bettina Speckmann*

Abstract

We study metrics that assess how close a triangulation is to being a Delaunay triangulation, for use in contexts where a good triangulation is desired but constraints (e.g., maximum degree) prevent the use of the Delaunay triangulation itself. Our near-Delaunay metrics derive from common Delaunay properties and satisfy a basic set of design criteria, such as being invariant under similarity transformations. We compare the metrics, showing that each can make different judgments as to which triangulation is closer to Delaunay. We also present a preliminary experiment, showing how optimizing for these metrics under different constraints gives similar, but not necessarily identical results, on random and constructed small point sets.

1 Introduction

Delaunay triangulations are a common construct in computational geometry: practically any class on computational geometry teaches about Delaunay triangulations of point sets and their duality to the Voronoi diagram. They are efficiently computable, and have many desirable properties, for instance, lexicographically maximizing the minimum angle of the corners of the resulting triangles [20], having bounded dilation [4, 11], minimizing the maximum circumcircle [5], maximizing the minimum enclosed circle [5, 18] of its triangles. As such, they form the basis of many algorithms that require some triangulation of a point set.

However, there are various scenarios imaginable where the Delaunay triangulation is not immediately applicable due to constraints on the desired triangulation, such as a limited vertex degree or a set of edges that needs to be included. In such cases, we may want to find a triangulation that is as close as possible to the Delaunay triangulation while adhering to the constraints. This, however, requires a way to measure how *near-Delaunay* a triangulation is. Another scenario in which such a measure would be useful is when a triangulation is already given – for example, the triangulation of a terrain. In such a setting, we could use the measure to assess the quality of the triangulation.

An example of a triangulation that aims to be as close as possible to the Delaunay triangulation given a set of edges that needs to be included is the Constrained Delaunay Triangulation (CDT) [3, 13]. It provides an alternative definition of when an edge or triangle may be part of the triangulation, such that it is "as close as possible" to being Delaunay for the given constraints. However, it does not help in assessing how close a triangulation is to being the actual Delaunay triangulation, nor does it generalize to other forms of constraints.

In this work, we consider various ways to measure how close a triangulation is to being Delaunay. The problem of studying the properties of triangulations that are close to the Delaunay triangulation was proposed at CCCG 2017 by O'Rourke [16]. In this context, two measures were already proposed [16, 14], which we discuss further in Section 2. The aim of our work, is to explore a broader range of measures together with algorithms to compute them and with an analytical and experimental comparison. We identify several criteria for near-Delaunay metrics:

C1 The Delaunay triangulation should obtain the perfect score. We do not want to distinguish how nice Delaunay triangulations are of different point sets – because this would be a factor of the point set itself. A non-Delaunay triangulation should always score less than perfect, such that any non-Delaunay triangulation is considered less Delaunay.

C2 The measure should behave continuously for slight perturbations of the point set. Triangles that "severely" violate properties of a Delaunay triangulation should score worse than those with "slight" violations.

C3 The measure should be invariant under similarity transformations (translations, rotations, scaling and reflections). Though this criteria follows from the first for the Delaunay triangulation (which is invariant under similarity transformations), it is desirable for this to also hold for non-Delaunay triangulations, such that triangulations of different point sets can still be reasonably compared.

*Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands; n.a.c.v.beusekom@tue.nl, k.a.buchin@tue.nl, h.o.koerts@student.tue.nl, w.meulemans@tue.nl, b.speckmann@tue.nl.

†Department of Computer Science, University of Oxford; benjamin.rodatz@cs.ox.ac.uk.

C4 The measure should be decomposable, that is, evaluated separately on different elements of the triangulation. Though not strictly necessary, this allows various forms of aggregation (worst situation, average situation, etc.).

We observe, that the combination of criterion *C3* and *C4* suggests that the metric should also be "locally invariant". That is, even within the same triangulation, a decomposition element that is subject to the same constraints to another element up to similarity transformations should score the same. That is, the metric should not naturally award higher (or lower) scores to elements that are larger; instead small and large elements should contribute equally to the overall metric. In other words, a triangulation should not be considered near-Delaunay, simply because the deviations from the Delaunay property are only at very small elements.

We propose and compare several *near-Delaunay metrics*. These metrics (as shown in Table 1) satisfy our four criteria. They differ in the properties of the Delaunay triangulation they aim to capture, as well as how they decompose the given triangulation. For this, we use decompositions into: (1) quadrilaterals – a quadrilateral here is an edge and its two incident triangles, ignoring all other points; (2) edges – an edge is considered in context of all other points; (3) triangles in context of all other points. We show that these measures behave differently, capturing different aspects of how near-Delaunay a triangulation is. Finally, we briefly compare how our measures relate to the CDT.

Related work. One of the well-known results in computational geometry is that any triangulation can be transformed into the Delaunay triangulation using Lawson flips [12]. A natural consideration would thus be to measure the number of flips necessary to transform a triangulation into a Delaunay triangulation. Though it satisfies *C1* and *C3*, it satisfies neither *C2* – it is a discrete measure and it is not immediately based on the violations of a Delaunay property – nor *C4*. An additional complication is that computing such flip distances is generally hard [17].

Another natural consideration is to measure the number of points in any circumcircle of a triangle in the triangulation. This leads to the notion of higher-order Delaunay triangulations [7]. We do not consider them in this paper, since a small perturbation can greatly change the number of points in a circumcircle. That is, the resulting measure would not adhere to *C2*. But when this criterion is not needed, higher-order Delaunay triangulations are well suited to obtain triangulations close to the Delaunay triangulation that at the same time are optimized for another criterion. Van Kreveld et al. [22] discuss optimizing over first-order Delaunay triangulations for various criteria like minimizing the maximum degree.

Computing a triangulation that is as close as possible to the Delaunay triangulation (given certain constraints) can be seen as optimization problem. There are many papers studying optimal triangulations under various criteria. Bern et al. [2] show how to efficiently compute triangulations under criteria like maximizing the minimum height or minimizing the maximum eccentricity. Unfortunately for other criteria, computing optimal triangulations is more difficult. For instance computing the minimum-weight triangulation is NP-hard [15]. While the complexity of finding the minimum-dilation triangulation is open [8], many related problems on minimizing dilation are NP-hard [6]. Similarly, the complexity of finding the minimum-degree triangulation seems open, while it is NP-hard if as a constraint certain edges have to be included [9, 10].

The fact that for many optimization criteria efficient algorithms are not known, also limits the size of the point sets that we can include in our experiments, in which we for instance want to compute near-Delaunay triangulations with additional constraints like a degree bound or a bound on the weight of the triangulation. We here resort to enumerating triangulations, however, the total number of triangulation for a given point set is exponential [1, 19].

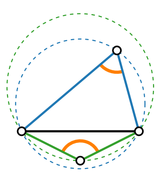
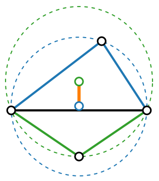
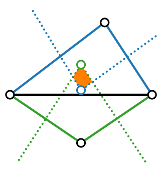
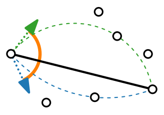
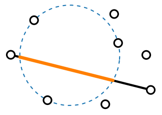
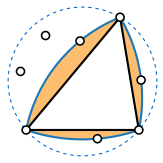
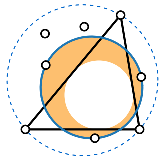
2 Near-Delaunay metrics

We define seven near-Delaunay metrics as shown in Table 1, which all satisfy our four criteria. Throughout, we assume that we are to measure a triangulation T on a point set P. We further assume, for sake of simplicity, that P is in general position: no three points are on a line and no four points are cocircular. We organize them below into three categories, depending on their form of decomposition (quadrilateral, edge and triangle). We always describe the measure just for a single decomposition element, silently assuming some form of aggregation such as taking their sum or extremal value.

2.1 Quadrilateral-based metrics

We start by introducing intuitive metrics that evaluate quadrilaterals. A quadrilateral consists of a non-convex-hull edge and the two triangles that are incident to it, defined by two vertices. Particularly, note that any point in P that is not one of the four defining vertices does not influence the metric on this particular quadrilateral – this contrasts the edge-based and triangle-based metrics. Thus, to compute the metrics it is sufficient to be given only the quadrilateral. Consequently, the metrics we introduce for quadrilaterals can be computed in constant time per quadrilateral and in linear time overall. Note that for each quadrilateral-based metric, lower scores mean closer to Delaunay (contrasting our other two forms of decomposition).

Table 1: Overview of metrics. Listed with each is the form of decomposition, the Delaunay property it is based on, and the running time for computing the measure for a given triangulation.

Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
						
Quadrilateral	Quadrilateral	Quadrilateral	Edge	Edge	Triangle	Triangle
Max-min angle	Voronoi Dual (edges)	Voronoi Dual (faces)	Empty Circle	Empty Circle	Empty Circumcircle	Empty Circumcircle
$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

Throughout, we denote by (u, v) the defining edge of the quadrilateral, and with p and q the *opposing* vertices of the two incident triangles $\triangle uvp$ and $\triangle uvq$. We use c_p and c_q to denote the center of the circumcircles of these triangles. A quadrilateral is *locally Delaunay* if the edge (u, v) is part of the Delaunay triangulation of $\{u, v, p, q\}$ – in other words, if the circumcircle of the one triangle does not contain the other vertex.

Opposing Angles. At CCCG 2017, O'Rourke suggested that a triangulation T is a near-Delaunay triangulation if the opposite angles α and β of a quadrilateral sum to at most $\pi+\varepsilon$ for $\varepsilon \geq 0$ [16] (see Fig. 1). If $\varepsilon = 0$, then T is Delaunay. Hence, it is natural to consider the smallest ε for a triangulation as a metric for how close a triangulation is to a Delaunay triangulation. We can readily interpret ε on a per-quadrilateral basis for a metric in our context; O'Rourke's suggestion is simply to take their maximum as the overall metric.

The intuition behind the metric is, when the sum of two opposing angles is larger than π, the empty circumcircle property is violated. When the sum is close to π, a slight movement in the points can restore the empty circumcircle property. However as the sum grows larger, the points generally need to move further to restore the property, unless p or q is very close to u or v, in which case a small movement is sufficient. Thus, we can use the sum of two opposing angles to evaluate how far from Delaunay a quadrilateral is.

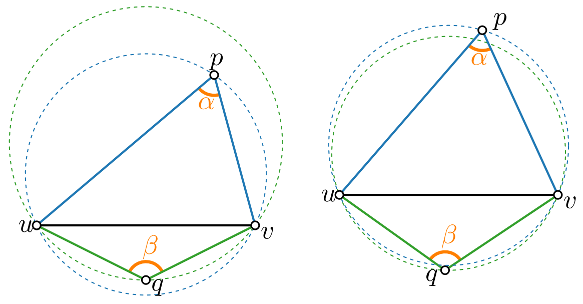


Figure 1: Opposing Angles metric. (left) Sum of opposing angles is less than π for quadrilateral that is not locally Delaunay. (right) It is at least π for quadrilaterals that are locally Delaunay.

Dual Edge Ratio. Additionally, Mitchell [16] suggested "measuring the signed distance between circumcenters of triangles sharing an edge; for Delaunay triangulations this is simply the dual edge length and nonnegative, but for non-Delaunay triangulations the circumcenters can be in the wrong order and hence have a negative distance between them. So one could look at the ratio of the dual edge signed-length to the primal edge length (for 2D triangulations) as a continuous measure of how close it is to non-Delaunay." [16] – which are also known as Hodge-optimized triangulations [14].

We adapt this metric to evaluate quadrilaterals: we measure only the negative distance part of the suggestion, to satisfy criterion *C1* and not distinguish between Delaunay triangles. The "wrong order" referred to above matched to the quadrilateral being not locally Delaunay. We thus define the Dual Edge Ratio as

$$\begin{cases} 0, & \text{if the quadrilateral is locally Delaunay.} \\ \frac{d(c_p,c_q)}{d(u,v)}, & \text{otherwise.} \end{cases}$$

Intuitively, if the quadrilateral is not locally Delaunay, ratio of the distance between c_p and c_q and the length of (u, v) roughly corresponds to how skinny the triangles are and thus how far from Delaunay they are as well (see Fig. 2). Clearly, the Delaunay triangulation scores zero on all its quadrilaterals. Any quadrilateral that is not locally Delaunay, and hence does not locally describe the dual of the Voronoi diagram, will score greater than zero. Any non-Delaunay triangulation must have at least one such quadrilateral.

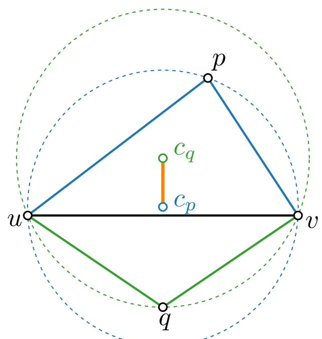


Figure 2: Dual Edge Ratio metric. The dual edge (orange) for a quadrilateral that is not locally Delaunay. We measure its length relative to the length of (u, v).

Dual Area Overlap. Similar to the previous metric, we may use the duality to Voronoi diagrams but in a different manner. Rather than looking at the distance, we may also consider an area-based metric. Intuitively, we consider the incorrect area of overlap between the "local Voronoi cells" that we may construct from the triangles of the quadrilateral.

The local Voronoi cell of p (and q analogously) is defined by the bisectors of u and v with p, which cross in c_p. If the quadrilateral is locally Delaunay, then the cells of p and q are disjoint. However, if the quadrilateral is not locally Delaunay, then they must overlap. The area of this overlap divided by the squared length of (u, v) is our Dual Area Overlap metric (see Fig. 3). We normalize using the squared edge length here, to ensure scale invariance, criterion *C3*.

The intuition behind this metric is similar to the intuition from the Dual Edge Ratio metric. The area of the overlap of the two Voronoi regions implies how far the quadrilateral is from having a proper non-overlapping Voronoi dual. A larger area means that points will have to move further to reach a non-overlapping dual, while a smaller area implies that a small movement in the points can already achieve this.

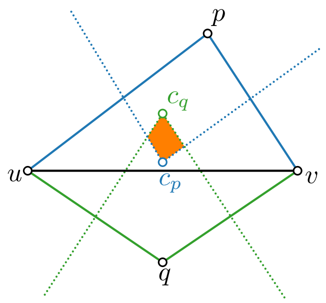


Figure 3: Dual Area Overlap metric. The local Voronoi cells of p and q (dotted) for a quadrilateral that is not locally Delaunay. We measure the overlap (orange).

2.2 Edge-based metrics

We now turn to edge-based metrics, that evaluate each edge (u, v) of T in context of all other points in P. That is, it penalizes edges, even if the defined quadrilateral is locally Delaunay. We present two new metrics below, both of which are based on the same principle: as an edge of the Delaunay triangulation must be the chord of a circle that does not strictly contain any other vertices of P, we consider how much we much deform a circle to find such an empty deformed circle instead. The difference between our two metrics is how they perform this deformation. Note that in both cases, higher scores mean closer to Delaunay, contrasting quadrilateral-based metrics.

Lens. When an empty circle exists, it can be seen as two circulars arcs, one on each side of the edge. The arc in the one halfplane with respect to the line spanned by (u, v) excludes from its interior all vertices of P in that same halfplane. With our Lens metric, we reverse this idea to deform our circle into a lens; the "sharpness" of this lens is then our metric.

Specifically, consider all points $P' \subseteq P$ that lie on one side of the line spanned by edge (u, v). The circular arc from u to v through some point of P' that is minimal in terms of segment area (or equivalently, arc length or central angle) is the largest arc possible on this side of the edge that does not contain any point of P' in its segment area. Let a and a' denote the two circular arcs obtained this way. We consider the "interior" angle α between the tangent directions of a and a' at u as our metric (see Fig. 4). If $\alpha \geq \pi$, the edge is a Delaunay edge and we cap the metric to π to satisfy criterion *C1*; for any non-Delaunay edge, $\alpha < \pi$.

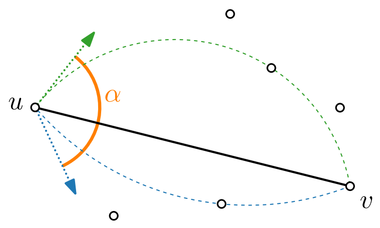


Figure 4: Lens metric. We find the largest empty arc on both sides of (u, v), and measure the angle α between their tangent directions at u.

We can easily compute the metric for a triangulation in quadratic time: for each edge, find the point that gives the smallest arc on both sides; then compute the angle at which the arcs touch.

Shrunk Circle. With our second metric, we consider a different type of deformation: scaling. That is, we aim to find a smaller empty circle that relates to the edge we wish to measure. For a non-Delaunay edge, such a circle cannot have (u, v) as a chord, but it may still over-

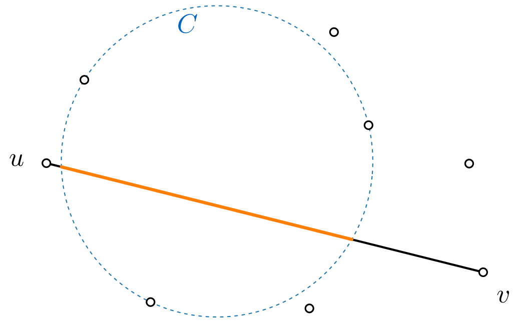


Figure 5: Shrunk Circle metric. We find the empty circle C that covers edge (u, v) most (orange).

lap the edge. As a Delaunay edge would be overlapped fully by an empty circle, we define our Shrunk Circle metric as the maximal fraction of the edge that can be overlapped by an empty circle C (see Fig. 5)). Note that, whereas the Lens metric considers both halfplanes independently, this is not the case here.

To compute this measure for an edge (u, v) of triangulation T on point set P, we observe the following: if an empty circle does not touch at least two points of P, we can readily grow it to a circle C' that does touch two points and strictly encompasses the previous circle and thus the overlap with (u, v) does not decrease. In other words, we need to consider only maximal circles with centers on the Voronoi diagram P. The lemma below argues that the overlap along one edge of the Voronoi diagram is convex and thus we need to test only its endpoints. In fact, we need to test only its vertices since unbounded edges occur only for pairs of points on the convex hull and thus the overlap of such circles with (u, v) are determined only by the circle's part inside the convex hull – which is maximized at bounded side of the unbounded edge. We can thus first compute the Delaunay triangulation for P, and then for every edge (u, v) of T, test the circumcircles of the Delaunay triangulation explicitly. This readily gives an quadratic-time algorithm for the overall metric.

Lemma 1 *Let e be an edge of the Voronoi diagram of P. The overlap of maximal circles along e with edge (u, v) is a convex function.*

Proof. Let p and q denote the points defining e. As the problem is invariant under translation, rotation and scaling, assume without loss of generality that $p = (0, 1)$ and $q = (0, -1)$; this implies that e is along the horizontal axis. Maximal circles C' are thus fully defined by their center $(m, 0)$. Let $\ell\colon y = ax + b$ denote the line spanned by (u, v). We first consider the overlap of C' with ℓ as a function of m.

The two intersection points of $\delta C'$ with ℓ are obtained by solving $|(x, ax + b) - (m, 0)|_2 = |(c, 0) - p|_2$, which simplifies to $(a^2 + 1)x^2 + 2(ab - m)x + (b^2 - 1) = 0$. The difference in x-coordinates between the two solutions to this quadratic equation are given by $\sqrt{D}/(a^2+1)$, where $D = 4(ab-m)^2 - 4(a^2+1)(b^2-1)$ is the discriminant; the amount over overlap with ℓ is thus $a\sqrt{D}/(a^2+1)$. As D is a convex quadratic function in m, so is $a\sqrt{D}/(a^2+1)$. To note, D is negative if the circle does not overlap ℓ, in which case the overlap is trivially zero: technically, the overlap is thus a function $a\sqrt{\max\{0, D\}}/(a^2 + 1)$. This proves that the overlap with ℓ is a convex function.

Since we are only interested in the values of m along e, that is, for which C' is empty, the overlap with ℓ is either fully within (u, v) or fully outside. Since the intersections behave continuously, the overlap with (u, v) thus behaves convex as well. $\square$

2.3 Triangle-based metrics

For our triangle-based metrics, we measure a triangle $\triangle uvw$ of T in context of all other points in P. Analogous to our edge-based metrics, we deform the circumcircle C of $\triangle uvw$ (which is empty for a Delaunay triangle) to find a suitable empty deformed circle. In contrast to the edge-based metrics, we now have a single fixed circle C which guides (and constrains) our metric. Specifically, we restrict our deformations to be contained in the circumcircle. As with edge-based metrics, higher scores mean closer to Delaunay.

Triangular Lens. Similarly to the Lens metric, when a empty circumcircle exists, it can be considered as three circular arcs on the outside of the triangle. We again replace each arc by the largest arc that is contained in the arc of the circumcircle and that contains no other points of P.

For our metric, we measure the fraction of the area in C but outside $\triangle uvw$ that is covered by the constructed lens. Let a_{uv}, a_{vw} and a_{wu} denote the segment areas of the three arcs constructed (see Fig. 6). Interpreting C

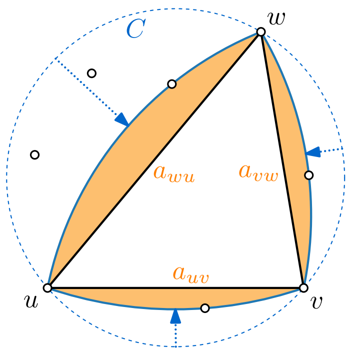


Figure 6: Triangular Lens metric. We compute the largest arc inside C that does not contain any points in its segment area for each edge of the triangle. Note that the three arcs are determined independently.

and $\triangle uvw$ as their enclosed areas, the score is then

$$\frac{a_{uv} + a_{vw} + a_{wu}}{C - \triangle uvw}.$$

We subtract the triangle to be able to assess and to meaningfully aggregate both skinny and fat triangles; this also means that each triangle's score lies in $(0, 1]$, where a score of 0 is only achievable in the limit (a point converging on each edge).

We easily compute the metric for a triangulation in quadratic time: for each triangle, find for each edge the smallest arc by testing all other vertices; then compute the segment areas and compute the resulting fraction.

Shrunk Circumcircle. As with our edge-based measures, the Triangular Lens metric deforms independently in the three segment areas defined by the edges. With our Shrunk Circumcircle metric, we consider a variant that considers all points simultaneously instead.

Specifically, we aim to find an empty circle C' that is contained in the circumcircle C and intersects all three sides of $\triangle uvw$. The score we associate with this circle C' is $\frac{C'-I}{C-I}$, where I is the area of the inscribed circle and we identify C' and C with the areas of these circles as well (see Fig. 7 (left)).

Note that we subtract the areas (numbers), we do not use set subtraction (difference of the shapes) and their resulting area; in our figures, we use examples where the inscribed circle is contained in the largest empty circle such that these two variants are the same, to easily visualize the score, but this is not necessarily the case.

The Shrunk Circumcircle metric is the circle C' with the highest associated score. For any triangle, I and C are constant, and thus it is simply the largest circle satisfying the two constraints. We subtract I as a lower bound on the largest empty circle (since the triangle itself must be empty), and divide by $C - I$ to normalize the the score to the range $(0, 1]$ independent of triangle shape and make the metric scale invariant.

We constrain C' to lie within C, such that the area we count is always an actual circle: if C' was to be allowed to grow outside C, it would either count area outside of the area that the Delaunay triangulation "considers", or the region we use the area of is not a circle in itself.

We constrain C' to intersect all three sides to make the score meaningful, even for very skinny triangles: otherwise, the largest empty circle may simply be fully in one segment, and even yield a relatively high score, though points are very close to the triangle edges (see Fig. 7 (right)).

To compute the metric for a triangle $\triangle uvw$ in context of P, we first argue about the properties of largest C'. Circle C' must either touch two points strictly inside C, or touch one such point and the boundary of C – otherwise, we can grow the circle into one that strictly encloses C' while still adhering to the constraints. As such, its center lies on the edges of the *local Voronoi diagram* of circle C and the points of P inside C (see Fig. 8). Curved segments for circle centers equidistant to the boundary of C and one of the points inside C, and straight segments defined by two points inside C. These curved segments are elliptical: the distance from a point on this curve to the center of C and to the contained point of P sums up to the radius of C. By construction, segments incident to the outerface of the diagram are curved segments; interior segments are straight.

In case the entire ellipse is part of the local Voronoi diagram (and hence in fact the only curve), we may consider it an elliptical arc, starting and ending at the furthest point from its defining point. Then, both along a straight segment as well as along a curved segment of the local Voronoi diagram, the radius of C' behaves unimodally: the function has a single (local and global) minimum at the closest point of the segment to the defining point(s). Along a segment, the maximal circle is hence either one of a small set of *critical placements* (if they exist): the endpoints of the segment, the fur-

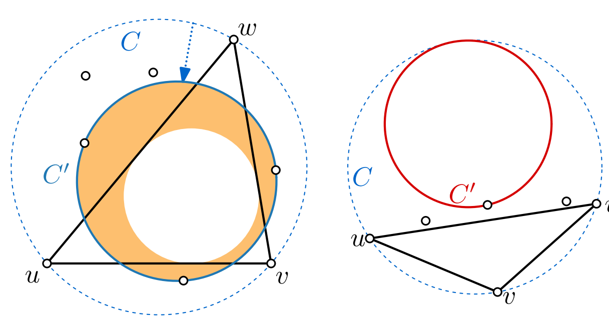


Figure 7: Shrunk Circumcircle metric. (left) We look for the empty circle C' in C of maximal radius. The score (orange) is its area minus the inscribed circle. (right) C' must intersect all three sides, for the measure to be meaningful when points are close to the triangle.

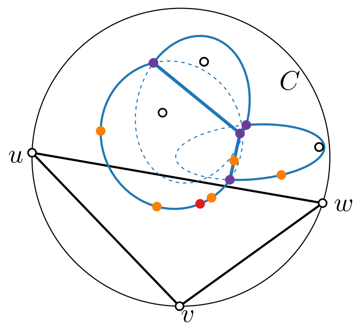


Figure 8: The local Voronoi diagram (blue) consists of elliptical and straight segments. The critical placements: endpoints of straight segments (purple), the furthest point on an ellipse (red), and circle touching a triangle edge (orange).

thest point of the ellipse, or one of the $O(1)$ placements where the defined circle touches a triangle edge. We further observe that the straight segments are a subset of the (full) Voronoi diagram of P.

With the insights above, we can define an algorithm to compute the Shrunk Circumcircle metric for all triangles in T in quadratic time. First, we compute the (full) Voronoi diagram of P. Then, for each triangle, we compute the metric as follows, in linear time. If there are no points inside the circumcircle, we have a Delaunay triangle and the metric is 1. If the circumcircle contains one point, then the local Voronoi diagram is a single ellipse and we test the critical placements. If the circumcircle contains more than one point, we find all straight segments local Voronoi diagram by traversing the full Voronoi diagram, testing the critical placements. For any straight segment found as such, we first shorten it to ensure that it does not define circles extending outside C. Then we test its critical placements. If the segment was shortened, we know that its defining points also define a segment of the local Voronoi diagram. For these segments, we also test the critical placements. By "testing" in the above, we mean testing whether the defined circle intersects all three sides (it is contained in C by construction), and if so, see if its radius is larger than any circle found so far. As we test $O(1)$ critical cases per segment, computing the metric takes linear time per triangle (after computing the full Voronoi diagram in $O(P \log P)$ time once) and thus quadratic time for the entire triangulation.

3 Comparing metrics

In the previous section, we defined seven near-Delaunay metrics. These capture in different ways how close to Delaunay a triangulation is. Though future work may endeavor to establish a standard here, it is not a-priori clear which measure is "the best": this likely depends on context, that is, the purpose of evaluating a triangulation. The question we ask here, is whether these metrics actually capture different facets of being "near-Delaunay". Specifically, given two metrics, do they always evaluate the same triangulation to be closer to Delaunay, for any given pair of triangulations?

Considering a single decomposition element to answer this question, there is clear distinction between the forms of decomposition: quadrilateral-based metrics do not take other points into account, contrasting the other two forms; edge-based metrics use angles and lengths, whereas triangle-based metrics use area ratios instead. We thus focus here on comparisons between metrics using the same form of decomposition. We study how the metrics differ from each other, and what properties they value. Specifically, for each comparison of metrics μ and μ', we show that there are triangulations T and T', such that $\mu(T) = \mu(T')$ and $\mu'(T) > \mu'(T')$. Such an example answers the above question of making the same judgments negatively.

Quadrilateral-based metrics. Our three quadrilateral-based metrics use only four points in measuring one element. Yet, we show that each metric evaluates a different facet of being near-Delaunay. Note that a (convex) quadrilateral is immediately a triangulation on this point set.

The Opposing Angles metric does not consider how the two summed angles are distributed over the two triangles. Even a very flat triangle (angle approaching π) can be offset a very tall triangle (angle approaching 0). Yet, these angles behave very differently from the distances between the circumcenters and thus the Dual Edge Ratio and Dual Area Overlap. Thus, we can construct two quadrilaterals (see Fig. 9): one has very similar triangles, while the other has very different triangles, but both have the same edge (u, v). Whereas the Opposing Angles metric scores the same on both, we see readily that the Dual Edge Ratio and Dual Area Overlap score differently.

Comparing the Dual Edge Ratio to the Dual Area Overlap, we see that the former is based purely on the distance between the circumcenters, whereas the latter depends also on the shape of the triangles. This allows

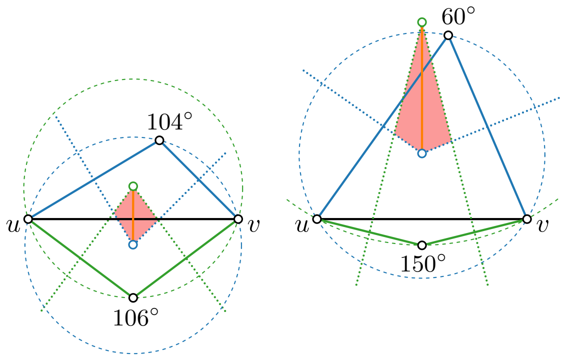


Figure 9: Two quadrilaterals with equal (sum of) Opposing Angles, but different Dual Edge Ratio (orange) and Dual Area Overlap (red).

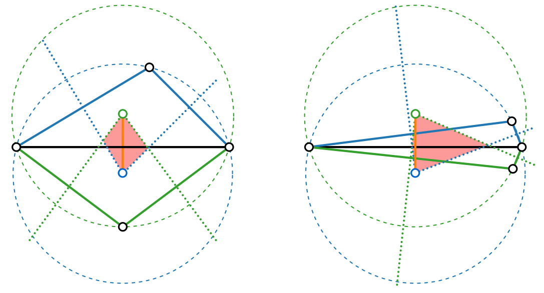


Figure 10: Two quadrilaterals with equal Dual Edge Ratio (orange) and different Dual Area Overlap (Red).

us to construct another two quadrilaterals (see Fig. 10): we move the opposing points along their defined circumcircles so as to not move the circumcenters, while changing the area of overlap.

Edge-based metrics. We have two metrics here: Lens and Shrunk Circle. As already mentioned in the previous sections, these differ by how they treat points on the different sides of the edge. Whereas the Lens measure treats these independently, the Shrunk Circle measure requires an integrated consideration of all points. This allows us to construct two triangulations again, using only four points in convex position (see Fig. 11): we keep the Lens measure constant, by moving one of the vertices over the defining arc, thus keeping the tangent at u constant as well; in contrast, the Shrunk Circle metric in the first example uses a circle that encompasses a large part of the defining arc of the Lens – by placing the point there, we can force the circle to shrink further and cover less of the edge (u, v).

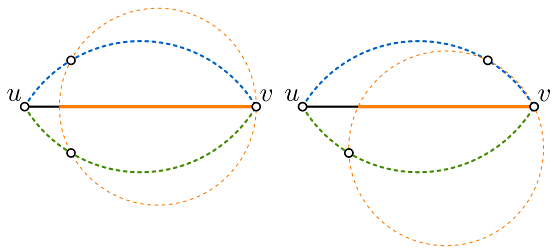


Figure 11: Two edges with equal Lens (blue, green) different Shrunk Circle (orange).

Triangle-based metrics. Finally, we consider the two triangle-based metrics. They behave somewhat similarly with respect to each other as the edge-based metrics do: whereas the Triangular Lens works with independent arcs per edge, the Shrunk Circumcircle uses a single circle that must overlap each of the three edges. We can thus follow the same principle to show two triangles in a point set that score equally on the Triangular Lens, but differently on the Shrunk Circumcircle, by moving the points along the arcs of the former to force the defining circle of the latter to shrink (see Fig. 12).

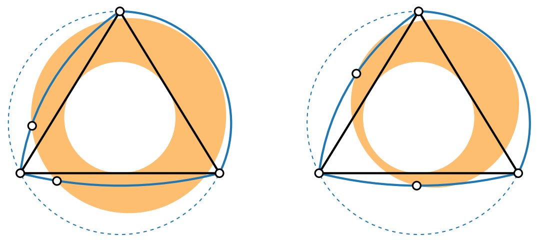


Figure 12: Two triangles with equal Triangular Lens (blue) and different Shrunk Circumcircle (orange).

4 Experiments

Here we explore what the most Delaunay-like triangulation of a point set looks like, for each of our metrics, given different constraints that can force a triangulation to be non-Delaunay. For a point set, we try all triangulations that adhere to the given constraints to find the *optimized* triangulation that scores best according to each metric. We use small sets of only 10 points, to ensure that this is feasible.

Constraints. We consider four types of constraints: constrained edges, a lower bound or upper bound on the total edge length, and a maximum degree.

For constrained edges, we are given a set of edges that must be included in the triangulation. The constrained edges are handpicked edges, which we consider to be "interesting". Most importantly the constrained edges are not in the Delaunay triangulation, are not chord of the convex hull, and lie somewhat close to another point. This is the same constraint as for the CDT and hence we may also compare how this structure compares to our result.

The length of the triangulation is the sum over the lengths of its edges. As the Delaunay triangulation neither minimizes nor maximizes the length, we can use an upper bound (maximum length) or a lower bound (minimum length), to constrain the triangulation to be non-Delaunay. Specifically, we use either 1.2 times the length of the Delaunay triangulation as a lower bound, or 0.8 times the length of the Delaunay triangulation as an upper bound. However, since the Delaunay triangulation is inherently short for many point sets, such triangulations often do not exist. We hence also create a special point set where the Delaunay triangulation is longer than most other triangulations, to see the differences between metrics.

Bounding the maximum degree means we consider only triangulations for which all vertices have degree at most a given constant. We use maximum degree 5 in our experiment. This number is sometimes exceeded by the Delaunay triangulation, but still allows for different triangulations of the same point set.

Aggregation. For our optimized triangulations, we have to evaluate each metric on the entire triangulation. So far, we have left the method of aggregation out of our considerations. For the purpose of our experiment, we consider two methods: using the sum and using bottleneck values.

The sum is a natural way to aggregate the values of a triangulation T, such that all decomposition elements (quadrilaterals, edges or triangles) have an impact on the triangulation. For quadrilateral-based metrics, we minimize the sum; otherwise, we maximize it.

Using bottleneck values means we focus on the worst-case element (maximum for quadrilateral-based metrics, minimum for our other metrics). The goal of this approach is to let the worst value be the deciding factor. A problem with this method is that there can be two different triangulations that score the same, as they both include the same bottleneck. Hence, we compare triangulations lexicographically: this means that the elements are sorted (increasingly for quadrilateral-based, decreasingly for other metrics) and the first element in which the triangulations differ, determines which triangulation is closer to Delaunay.

Results. Tables 3 through 10 in the full version [21] show our experimental results; Table 2 contains the most relevant excerpts. For the constrained edges, we show the CDT for comparison, using red edges to indicate the constraints. For the other constraints, we show the Delaunay triangulation for comparison. In both cases, the optimized triangulations use green markings to show edges that are different from the comparison.

Table 3 in [21] shows optimized triangulations with constrained edges using sum aggregation. We observe that most metrics are always similar to the CDT for such random point sets. The only differences are for Dual Edge Ratio and Dual Area Overlap: each time Dual Edge Ratio is different from the CDT, Dual Area Overlap is also different, though not necessarily vice versa. One exception is for Shrunk Circle. This suggests to us that, at least in small random cases, our metrics capture near-Delaunay quite well, as the CDT is an established way of getting a Delaunay-like triangulation, for these constraints. Using bottleneck aggregation (Table 4 in [21]), we observe that the quadrilateral metrics often behave similarly and are regularly different from the CDT. With one exception, the other optimized triangulations are again all identical to the CDT.

Tables 5 and 6 in [21] show the optimized triangulations with a minimum-length constraint. We can compare the different optimizations to empirically evaluate the behavior of the metrics. We observe that in the sum aggregation quadrilateral-based metrics generally behave differently than the other metrics. However, this distinction is less clear using the bottleneck aggregation. Furthermore, with bottleneck aggregation the Shrunk Circle metric often produces a unique triangulation. Note that this is the same case that was excepting from the general trend for constrained edges.

Table 7 and 8 in [21] show the optimized triangulation with a maximum-length constraint. If no such triangulation exists, we simply show the Delaunay triangulation. In every point set except the one specifically created for this constraint, there was no such triangulation. For this case, we observe that the Dual Edge Ratio and Dual Area overlap produce different triangulations from the other metrics. In the bottleneck variant, we also see that the Lens measure procedures another distinct triangulation.

Table 9 and 10 in [21] shows the optimization with a maximum-degree constraint. In only few of the random point sets, the maximum degree exceeds the constraint. Most metrics flip the same edge, the notable exception being the Shrunk Circumcircle metric which flips two different edges for one case (shown). The two other point sets shown were specifically created for this case. We see considerable differences here between metrics, as well as between sum and bottleneck aggregation.

With the wheel example here, we may perhaps see the one case where there is a somewhat clear case of a "visually nice" optimized triangulation. Specifically, a somewhat regular pattern emerges for Lens and Shrunk Circle using sum aggregation as well as all metrics except for the Dual Edge Ratio and Dual Area Overlap using bottleneck aggregation – in the other cases we see very skinny triangles occurring. Whether this form of being near-Delaunay, however, is the most useful remains possibly context dependent. This hints at bottleneck aggregation perhaps being more useful – indeed, it matches the traditional lexicographic optimization of the minimum angle that the Delaunay triangulation achieves.

5 Discussion

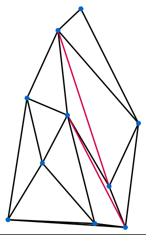
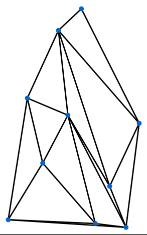
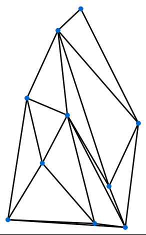
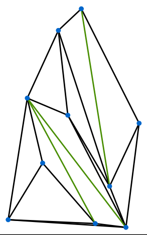
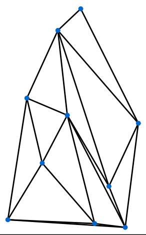
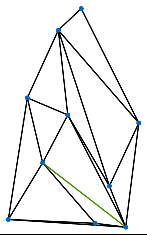
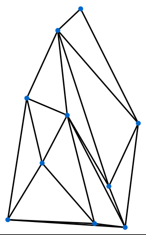
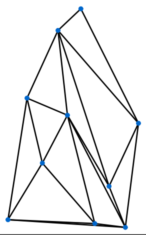
With a suite of metrics, we now have a common framework to think about situations where we need a good (Delaunay-like) triangulation to compute with, but constraints such as bounded degree prevent us from actually using the Delaunay triangulation itself. We have shown how these metrics differ among themselves as well as how they result in different triangulations when considering optimizing under the constraint of including given edges (like the Constrained Delaunay Triangulation).

We leave to future work to establish efficient algorithms to compute the best triangulation (given one of the metrics) given a set of constraints. It may further be interesting to investigate how humans (or computational geometers) assess the quality of a triangulation, how close it is to being Delaunay, and how this relates to the metrics provided here. This may uncover that at least to match intuition, we may need combinations of metrics, or possibly a different metric altogether.

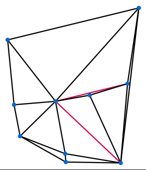
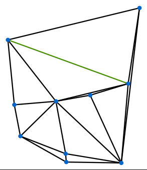
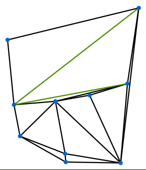
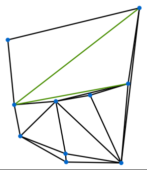
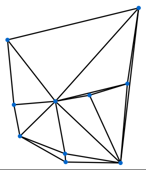
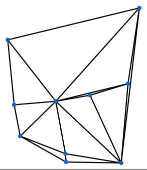
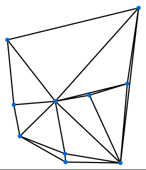
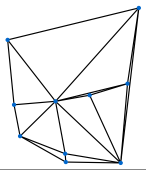
A possible avenue for further metrics is to consider the empty ellipse as the generalization of the empty circle, trying to optimize for the ellipse's aspect ratio. This seems mostly relevant for edge-based or quadrilateral-based metrics, since triangles do not necessarily allow for empty ellipses passing through their corners. But for a single edge or quadrilateral, a general ellipse seems to provide too much freedom, allowing for somewhat arbitrary-seeming results. Restricting one of the axes of the ellipse to be parallel to the defining edge may offer a solution; yet, this seems to cause a counterintuitive relation between the two sides of the edge.

Table 2: Excerpt with the most relevant results, summarizing the appendix in the full version [21].

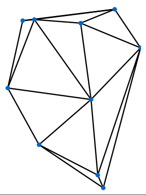
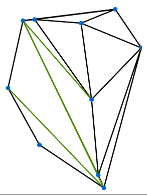
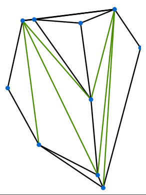
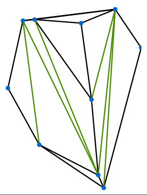
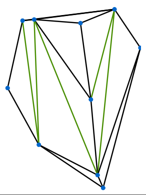
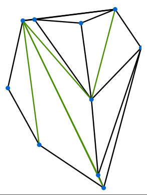
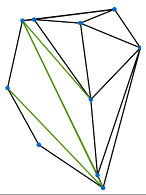
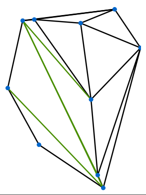
Optimizing with constrained edges (Sum aggregation)

CDT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

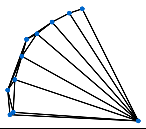
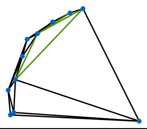
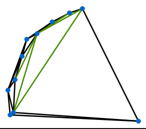
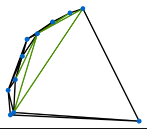
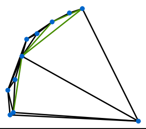
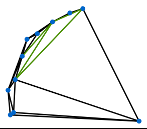
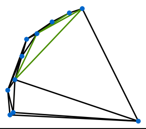
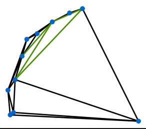
Optimizing with constrained edges (Bottleneck aggregation)

CDT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

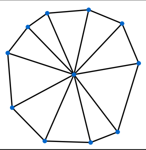
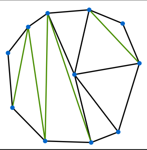
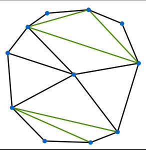
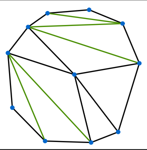
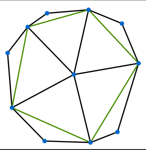
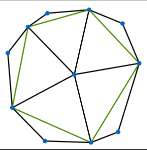
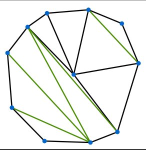
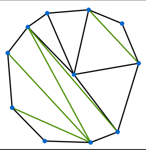
Optimizing with minimum length 1.2 Delaunay length (Sum aggregation)

DT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

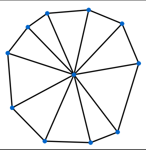
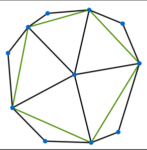
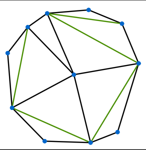
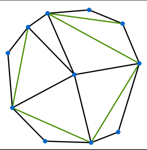
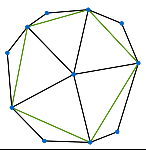
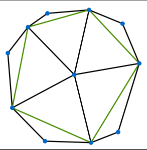
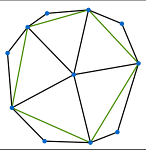
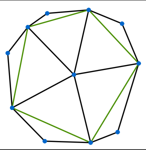
Optimizing with maximum length 0.8 Delaunay length (Bottleneck aggregation)

DT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

Optimizing with maximum degree 5 (Sum aggregation)

DT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

Optimizing with maximum degree 5 (Bottleneck aggregation)

DT	Opposing Angles	Dual Edge Ratio	Dual Area overlap	Lens	Shrunk Circle	Triangular Lens	Shrunk Circumcircle
							

References

[1] O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004.

[2] M. Bern, H. Edelsbrunner, D. Eppstein, S. Mitchell, and T. S. Tan. Edge insertion for optimal triangulations. *Discrete & Computational Geometry*, 10(1):47–65, 1993.

[3] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[4] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.

[5] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1063–1075, 1989.

[6] P. Giannopoulos, R. Klein, C. Knauer, M. Kutz, and D. Marx. Computing geometric minimum-dilation graphs is NP-hard. *International Journal of Computational Geometry & Applications*, 20(02):147–173, 2010.

[7] J. Gudmundsson, M. Hammar, and M. van Kreveld. Higher order Delaunay triangulations. *Computational Geometry*, 23(1):85–98, 2002.

[8] J. Gudmundsson and C. Knauer. Dilation and detours in geometric networks. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

[9] K. Jansen. One strike against the min-max degree triangulation problem. *Computational Geometry*, 3(2):107–120, 1993.

[10] G. Kant and H. L. Bodlaender. Triangulating planar graphs while minimizing the maximum degree. *Information and Computation*, 135(1):1–14, 1997.

[11] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992.

[12] C. L. Lawson. Software for C1 surface interpolation. In *Mathematical software*, pages 161–194. Elsevier, 1977.

[13] D.-T. Lee and A. K. Lin. Generalized delaunay triangulation for planar graphs. *Discrete & Computational Geometry*, 1(3):201–217, 1986.

[14] P. Mullen, P. Memari, F. de Goes, and M. Desbrun. HOT: Hodge-optimized triangulations. In *Proc. ACM SIGGRAPH 2011*, Article 103, pages 1–12, 2011.

[15] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM (JACM)*, 55(2):1–29, 2008.

[16] J. O'Rourke. Open problems from CCCG 2017, 2018.

[17] A. Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry*, 47(5):589–604, 2014.

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

[19] M. Sharir and A. Sheffer. Counting triangulations of planar point sets. *The Electronic Journal of Combinatorics*, 18(P70):1, 2011.

[20] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1978.

[21] N. van Beusekom, K. Buchin, H. Koerts, W. Meulemans, B. Rodatz, and B. Speckmann. Near-Delaunay Metrics, 2021. `https://arxiv.org/abs/2106.11621`.

[22] M. van Kreveld, M. Löffler, and R. I. Silveira. Optimization for first order Delaunay triangulations. *Computational Geometry*, 43(4):377–394, 2010.

Shortcut Hulls: Vertex-restricted Outer Simplifications of Polygons

Annika Bonerath[*] Jan-Henrik Haunert[†] Joseph S. B. Mitchell[‡] Benjamin Niedermann[§]

Abstract

Let P be a crossing-free polygon and $\mathcal{C}$ a set of shortcuts, where each shortcut is a directed straight-line segment connecting two vertices of P. A shortcut hull of P is another crossing-free polygon that encloses P and whose oriented boundary is composed of elements from $\mathcal{C}$. Shortcut hulls find their application in geo-related problems such as the simplification of contour lines. We aim at a shortcut hull that linearly balances the enclosed area and perimeter. If no holes in the shortcut hull are allowed, the problem admits a straight-forward solution via shortest paths. For the more challenging case that the shortcut hull may contain holes, we present a polynomial-time algorithm that is based on computing a constrained, weighted triangulation of the input polygon's exterior. We use this problem as a starting point for investigating further variants, e.g., restricting the number of edges or bends. We demonstrate that shortcut hulls can be used for drawing the rough extent of point sets as well as for the schematization of polygons.

1 Introduction

The simplification of polygons finds a great number of applications in geo-related problems. For example in map generalization it is used to obtain abstract representations of area features such as lakes, buildings, or contour lines. A common technique, which originally stems from polyline simplification, is to restrict the resulting polygon Q of a polygon P to the vertices of P, which is also called a *vertex-restricted simplification* [21, 25, 39]. In that case Q consists of straight edges[1] that are *shortcuts* between vertices of P. In the classic problem definition of line and area simplification the result Q may cross edges of P.

In this paper, we consider the vertex-restricted crossing-free simplification of a polygon P considering only shortcuts that lie in the exterior of P or are part of the boundary of P. In contrast to other work, we consider the shortcuts as input for our problem and do not require special properties, e.g., that they are crossing-

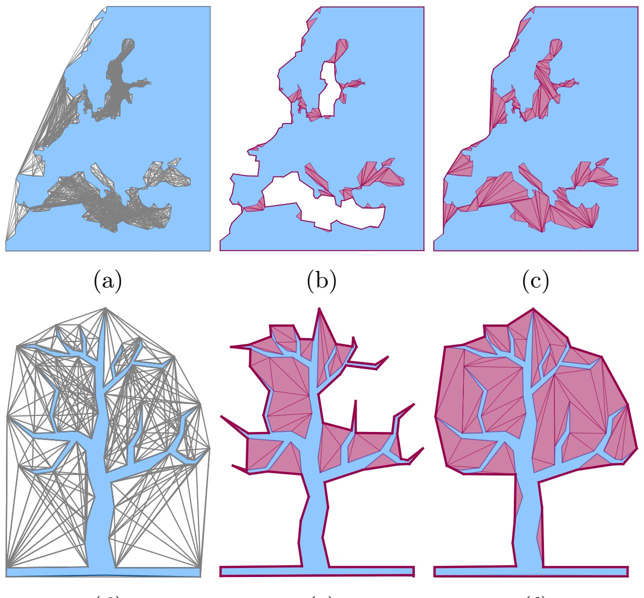
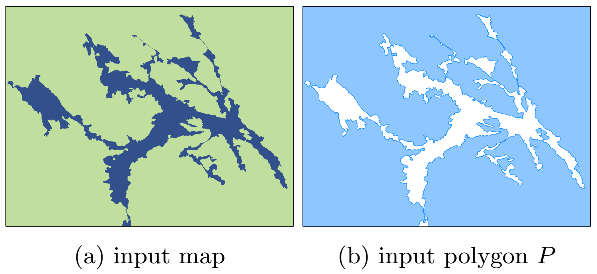


Figure 1: 1st column: Input polygon (blue) with a set $\mathcal{C}$ of all possible shortcuts (gray). 2nd–3rd columns: Optimal $\mathcal{C}$-hulls (blue and red area) for different λ.

free, or that they comprise all possible shortcuts. The result of the simplification is a *shortcut hull* Q of P possibly having holes. We emphasize that the edges of a shortcut hull do not cross each other. Figure 1 shows polygons (blue area) with all possible shortcuts and different choices of shortcut hulls (blue and red area). Such hulls find their application when it is important that the simplification contains the polygon. Figure 2 shows the simplification of a network of lakes. We emphasize that the lakes are connected to the exterior of the green polygon at the bottom side. In that use case, it can be desirable that the water area is only decreased to sustain the area of the land occupied by important map features. The degree of the simplification of Q can be measured by its perimeter and enclosed area. While a small perimeter indicates a strong simplification of P, a small area gives evidence that Q adheres to P. In the extreme case Q is either the convex hull of P minimizing the possible perimeter, or Q coincides with P minimizing the enclosed area. We present algorithms that construct shortcut hulls of P that linearly balance these two contrary criteria by a parameter $\lambda \in [0, 1]$, which specifies the degree of simplification. With increasing

*University of Bonn, `bonerath@igg.uni-bonn.de`
†University of Bonn, `haunert@igg.uni-bonn.de`
‡Stony Brook University `joseph.mitchell@stonybrook.edu`
§University of Bonn, `niedermann@igg.uni-bonn.de`
[1]Throughout this paper, we use the term *edge* instead of *straight-line segment*.



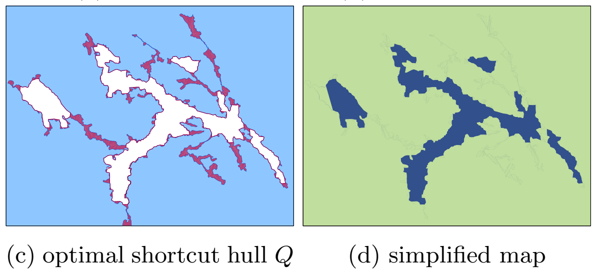


Figure 2: Simplification of a network of lakes in Sweden.

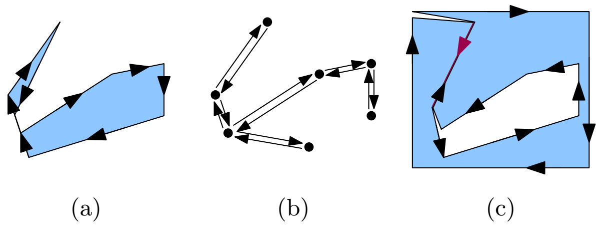


Figure 3: Weakly-simple polygons. (a)–(b) Valid input polygon as the exterior is a connected region. (c) Invalid input polygon as the exterior consists of two regions.

λ the enclosed area is increased, while the perimeter is decreased. We show that for the case that Q must not have holes we can reduce the problem to finding a cost-minimal path in a directed acyclic graph that is based on the given set of possible shortcuts. However, especially for the application in geovisualization, where it is about the simplification of spatial structures, we deem the support of holes in the simplification as an essential key feature. For example, in Figure 2d the connections between the lakes are not displayed anymore as they are very narrow, while it is desirable to still show the large lakes. We therefore investigate the case of shortcut hulls with holes in greater detail.

Input Polygon. As input we expect a clockwise-oriented polygon P that is *weakly-simple*, which means that we allow vertices to lie in the interior of edges as well as edges that point in opposite directions to lie on top of each other; see Figure 3. In particular, the edges of P do not cross each other. Such polygons are more general than simple polygons and can be used to describe more complex geometric objects such as the faces of a graph embedded into the plane; see Figure 4

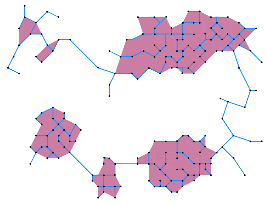


Figure 4: Shortcut hull of a minimum spanning tree.

for minimum spanning tree. For the input polygon P we further require that its exterior is one connected region; we say that the exterior of P is *connected*; see Figure 3. Hence, both a simple polygon and the outer face of the plane embedding of a planar graph are possible inputs. Finally, we emphasize that P may have holes. We can handle every hole separately assuming that we have inserted a narrow channel in P connecting it with the exterior of P; consider the lakes in Figure 2. We can force the algorithm to fill the artificially introduced channel with the interior of Q.

Formal Problem Definition. We are given a weakly-simple polygon P with connected exterior and a set $\mathcal{C}$ of directed edges in the exterior of P such that the endpoints of the edges in $\mathcal{C}$ are vertices of P; see Figure 5a. We call the elements in $\mathcal{C}$ *shortcuts*. A $\mathcal{C}$*-hull* is a weakly-simple polygon whose oriented boundary consists only of directed edges from $\mathcal{C}$, whose exterior is connected, and that contains P. We allow $\mathcal{C}$-hulls to have holes. We observe that such holes can only lie in the exterior of P. We are interested in a $\mathcal{C}$-hull Q that linearly balances the perimeter and enclosed area of Q. Formally, we define the *cost* of a $\mathcal{C}$-hull Q as

$$c(Q) = \lambda \cdot c_{\mathrm{P}}(Q) + (1-\lambda) \cdot c_{\mathrm{A}}(Q), \qquad (1)$$

where $\lambda \in [0,1]$ is a given constant balancing the perimeter $c_{\mathrm{P}}(Q)$ and the area $c_{\mathrm{A}}(Q)$ of Q. Further, Q is *optimal* if for every $\mathcal{C}$-hull Q' of P it holds $c(Q) \leq c(Q')$.

ShortcutHull.
given: weakly-simple polygon P with n vertices and connected exterior, set $\mathcal{C}$ of shortcuts of P, and $\lambda \in [0,1]$
find: optimal $\mathcal{C}$-hull Q of P (if it exists)

Further, we observe that it holds $|\mathcal{C}| \in O(n^2)$ as the edges of $\mathcal{C}$ have their endpoints on the boundary of P.

Our Contribution. We first discuss how to construct an optimal $\mathcal{C}$-hull in $O(|\mathcal{C}|)$ time for the case that it must not have holes (Section 3). Afterwards, we turn our focus to $\mathcal{C}$-hulls that may have holes (Sections 4–6). In particular, we show that finding an optimal $\mathcal{C}$-hull Q of P is closely related to finding a triangulation T of the exterior of P such that each triangle $\Delta \in T$ either

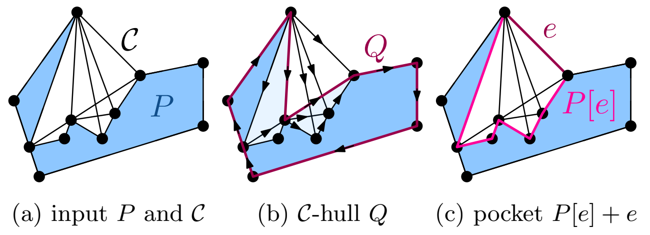


Figure 5: The input, a solution, and a subinstance for an instance of the problem.

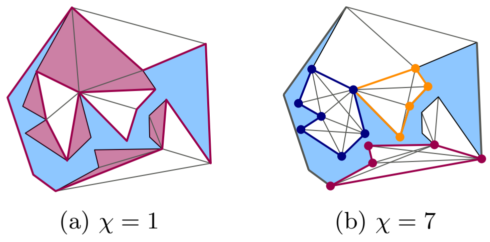


Figure 6: Two examples of set C with different spatial complexities χ. (a) $\mathcal{C}$-triangulation and $\mathcal{C}$-hull. (b) connected components of the crossing graph.

belongs to the interior or exterior of Q; see Figure 6a. We present an algorithm that solves ShortcutHull in $O(n^2)$ time if we forbid holes and in $O(n^3)$ time in the general case. Moreover, in the case that the edges of $\mathcal{C}$ do not cross each other, it runs in $O(n)$ time. More generally, we analyse the running time based on the structure of $\mathcal{C}$. Let S be the region between P and the convex hull of P. Let G be the *crossing graph* of $\mathcal{C}$, i.e., each node of G corresponds to an edge in $\mathcal{C}$ and two nodes of G are adjacent if the corresponding edges in $\mathcal{C}$ cross each other. The *spatial complexity* of $\mathcal{C}$ is the smallest number $\chi \in \mathbb{N}$ for which every connected component of G can be enclosed by a polygon with χ edges that lies in the exterior of P and only consists of vertices from P; see Figure 6. We show that the purposed algorithm runs in $O(\chi^3 + n\chi)$ time. We emphasize that $\chi \in O(n)$. Moreover, we present two variants of $\mathcal{C}$-hulls that restrict the number of permitted edges or bends. We further discuss relations of shortcut hulls with respect to problems from application in cartography and computational geometry (Section 7).

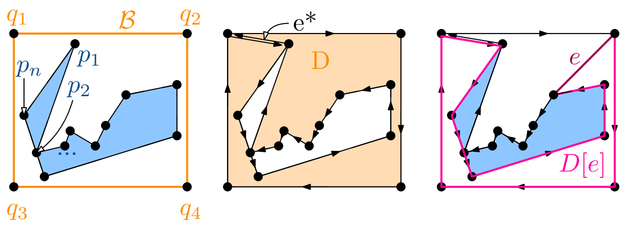
2 Related Work

In the following, we consider two major research fields that are closely related to our work. At first, the field of representing geometric objects by less complex and possibly schematized geometric objects and, secondly, the field of constrained and weighted triangulations. Application fields for the representation of geometric objects by less-complex and possibly schematized objects are found, for example, in cartography: administrative borders [8, 12, 27, 52], building footprints [29, 54], and metro maps [31, 43, 55]. In particular, we want to point out the generalization of isobathymetric lines in sea charts where the simplified line should lie on the downhill side of the original line to avoid the elimination of shallows [56]. In this context, it is important to find a good balance between the preservation of the information and the legibility of the visualization [13]. Considering a polygon as input geometry, a basic technique for simplification and schematization is the convex hull [6, 18, 26, 45]. An approach for rectilinear input polygons are tight rectilinear hulls [10]. Multiple other approaches for polygonal hulls of polygons exist—some of them can be solved in polynomial time [29], while others are shown to be NP-hard [30]. A closely related field is the topologically correct simplification and schematization of polygonal subdivisions [12, 24, 38, 40, 53]. For the case that multiple geometric objects are the input of the problem, there exist several techniques for combining the aggregation and representation by a more simple geometry. In the case that the input is a set of polygons, a common technique is to use a partition of the plane, such as a triangulation, as basis [17, 32, 36, 37, 46, 50]. In the case that the input is a set of points, we aim at representing this by a polygonal hull. Many approaches such as α-shapes [23] and χ-shapes [22] use a triangulation as their basis. Another approach is based on shortest-paths [19]. Note that there also exists work on combining the aggregation of point sets resulting in schematized polygons [11, 54]. For considering polylines as input there exists work on computing an enclosing simple polygon based on the Delaunay triangulation [3]. The schematization of polylines is also closely related to our approach. On the one hand, there is the schematization of a polyline inside a polygon or between obstacles [2, 35, 41, 49]. Alternatively, there also exists work on the simplification of a polyline based on a Delaunay triangulation [3, 4, 5]. For the general simplification of polylines we also refer to the Douglas-Peucker algorithm, which is most widely applied in cartography [20], and similar approaches [1, 42, 44].

Triangulating a polygon is widely studied in computational geometry. Triangulation of a simple polygon can be done in worst-case linear time [14]. A polygon with h holes, having in total n vertices, can be triangulated in $O(n \log n)$ time [28] or even $O(n + h \log^{1+\varepsilon} h)$ time [7]. Our approach is particularly related to minimum-weight triangulations [47] and constrained triangulations [15, 16, 33, 34, 48].

3 Computing Optimal Shortcut Hulls without Holes

Let $G_{\mathcal{C}}$ be the graph induced by the edges in $\mathcal{C}$. We call $G_{\mathcal{C}}$ the *geometric graph* of $\mathcal{C}$. If we do not allow the shortcut hull to have holes, we can compute an op-



(a) $\mathcal{B}$ (b) D (c) pocket of e

Figure 7: Containing box $\mathcal{B}$ and sliced donut D of P.

timal $\mathcal{C}$-hull Q based on a cost-minimal path in $G_\mathcal{C}$; see Figure 5b. For each edge e let $P[e]$ be the polyline of P that is enclosed by e. We call the polygon describing the area enclosed by e and $P[e]$ the *pocket* of e; see Figure 5c. We direct e of $G_\mathcal{C}$ such that it starts at the starting point of $P[e]$ and ends at the endpoint of $P[e]$. For each edge e we introduce costs that rate the length $c_\mathrm{P}(e)$ of e as well as the area $c_\mathrm{A}(P[e])$ of the pocket of e with respect to λ, i.e. $c(e) = \lambda \cdot c_L(e) + (1-\lambda) \cdot c_\mathrm{A}(P[e])$.

Observation 1 *The vertices of the convex hull of P are part of the boundary of any shortcut hull of P.*

Due to Observation 1, any $\mathcal{C}$-hull of P contains the topmost vertex v of P. Hence, $G_\mathcal{C}$ does not contain any edge e that contains v in its pocket and when removing v from $G_\mathcal{C}$ we obtain a directed acyclic graph. We use this property to prove that a cost-minimal path in $G_\mathcal{C}$ corresponds to an optimal $\mathcal{C}$-hull.

Theorem 1 *The problem* SHORTCUTHULL *without holes can be solved in $O(|\mathcal{C}|)$ time. In particular, in the case that the edges in $\mathcal{C}$ do not cross each other it can be solved in $O(n)$ time and $O(n^2)$ time otherwise.*

The proof of Theorem 1 is deferred to the full version [9]. If we allow Q to have holes, we cannot rate the costs for the area of a pocket in advance.

4 Structural Results for Shortcut Hulls with Holes

In this section, we present structural results for SHORTCUTHULL, which we utilize for an algorithm in Section 5. We allow the shortcut hull to have holes.

4.1 Basic Concepts

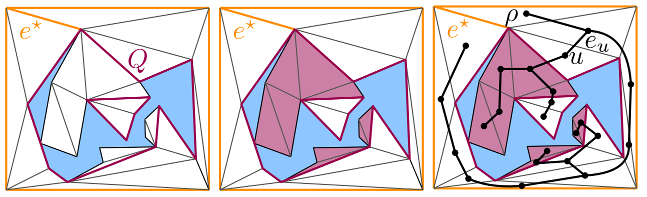
Let P be a weakly-simple polygon with connected exterior. Let $p_1, \dots, p_n$ be the vertices of P; see Figure 7a. We assume that the topmost vertex of P is uniquely defined; we always can rotate P such that this is the case. We denote that vertex by p_1 and assume that P is clockwise oriented. Further, let $\mathcal{C}$ be a set of shortcuts of P and $\lambda \in [0,1]$; see Figure 5a. Due to Observation 1, any $\mathcal{C}$-hull of P contains p_1.

First we introduce concepts for the description of the structural results and the algorithm. Let $\mathcal{B}$ be an axis-aligned rectangle such that it is slightly larger than the bounding box of P; see Figure 7a. Let $q_1, \dots, q_4$ be the vertices of $\mathcal{B}$ in clockwise order such that q_1 is the top-left corner of $\mathcal{B}$. We require that the diagonal edges q_1q_3 and q_2q_4 intersect P, which is always possible. We call $\mathcal{B}$ a *containing box* of P. Let D be the polygon $q_1 \dots q_4 q_1 p_1 p_n \dots p_1 q_1$. We call D a *sliced donut* of P; see Figure 7b. We observe that D is a weakly-simply polygon whose interior is one connected region. Further, we call $e^\star = p_1q_1$ the *cut edge* of D. For an edge e in the interior of D connecting two vertices of D let $D[e]$ be the polyline of D that connects the same vertices such that $e^\star$ is not contained; see Figure 7c. Let $D[e] + e$ be the polygon that we obtain by concatenating $D[e]$ and e such that $e^\star$ lies in the exterior of $D[e] + e$. Note that if $e \in \mathcal{C}$ then $D[e] = P[e]$. We call $D[e] + e$ the *pocket* e. In particular, we define D to be the pocket of $e^\star$.

Observation 2 *The edges of a $\mathcal{C}$-hull of P are contained in the sliced donut D.*

In the following, we define a set $\mathcal{C}^+$ of edges in D with $\mathcal{C} \subseteq \mathcal{C}^+$ that we use for constructing triangulations of D, which encode the shortcut hulls. Generally, a *triangulation* of a polygon H is a superset of the edges of H such that they partition the interior of H into triangles. Further, for a given set E of edges an E-*triangulation* of H is a triangulation of H that only consists of edges from E. Moreover, we say that a set E of edges is *part of* a triangulation T if E is a subset of the edges of T. Conversely, we also say that T *contains* E if E is part of T. Note that the edges of H are part of any E-triangulation of H.

We call a set $\mathcal{C}^+$ of edges with $\mathcal{C} \subseteq \mathcal{C}^+$ an *enrichment* of the shortcuts $\mathcal{C}$ and the sliced donut D if (1) every edge of $\mathcal{C}^+$ is contained in D, (2) every edge of $\mathcal{C}^+$ starts and ends at vertices of D, and (3) for every set $\mathcal{C}' \subseteq \mathcal{C}$ of pair-wisely non-crossing edges there is a $\mathcal{C}^+$-triangulation T of D such that $\mathcal{C}'$ is part of T. First, we observe that $\mathcal{C}^+$ is well-defined as every edge in $\mathcal{C}$ satisfies the first two properties. Further, by definition for any $\mathcal{C}$-hull Q there is a $\mathcal{C}^+$-triangulation T of D that contains Q. Hence, as an intermediate step our algorithm for computing an optimal $\mathcal{C}$-hull Q creates an enrichment of $\mathcal{C}$ and D, and then constructs a $\mathcal{C}^+$-triangulation that contains Q. In Section 4.2 we discuss the structural correspondences between $\mathcal{C}^+$-triangulations of D and (optimal) $\mathcal{C}$-hulls. In Section 4.3 we then show how to construct $\mathcal{C}^+$. For example a simple approach for an enrichment of $\mathcal{C}$ is the set of all possible shortcuts in D. We observe that any enrichment $\mathcal{C}^+$ of $\mathcal{C}$ has $O(n^2)$ edges. In general, the size of $\mathcal{C}^+$ can be described by the spatial complexity of $\mathcal{C}$, which impacts the running time of our algorithm (Section 5).



(a) Q is part of T (b) labels (c) dual graph $G^\star$

Figure 8: $\mathcal{C}^+$-triangulation T. (b) The red triangles are active, while all other triangles are inactive. (c) The restricted dual graph $G^\star$ of T forms a tree with root ρ.

4.2 From $\mathcal{C}^+$-Triangulations to $\mathcal{C}$-Hulls

In this section, we assume that we are given an enrichment $\mathcal{C}^+$ for the set of shortcuts $\mathcal{C}$ and a sliced donut D. Let T be a $\mathcal{C}^+$-triangulation of D; see Figure 8.

Observation 3 *For each enrichment $\mathcal{C}^+$ of $\mathcal{C}$ and each $\mathcal{C}$-hull Q there exists a $\mathcal{C}^+$-triangulation T of the sliced donut D such that Q is part of T.*

Let T be a $\mathcal{C}^+$-triangulation of D such that the $\mathcal{C}$-hull Q is part of T; see Figure 8a. We can partition the set of triangles of T in those that are contained in the interior of Q and those that are contained in the exterior of Q. We call the former ones *active* and the latter ones *inactive*; see Figure 8b. Further, we call an edge e of T a *separator* if (1) it is part of P and adjacent to an inactive triangle, or (2) it is adjacent to both an active and an inactive triangle. Conversely, let $\ell\colon T \to \{0,1\}$ be a labeling of T that assigns to each triangle Δ of T whether it is active ($\ell(\Delta) = 1$) or inactive ($\ell(\Delta) = 0$). We call the pair $\mathbf{T} = (T, \ell)$ a *labeled* $\mathcal{C}^+$-triangulation. From Observation 3 we obtain the next observation.

Observation 4 *For each enrichment $\mathcal{C}^+$ of $\mathcal{C}$ and each $\mathcal{C}$-hull Q there exists a labeled $\mathcal{C}^+$-triangulation such that its separators stem from $\mathcal{C}$ and form Q.*

Let $\mathbf{T} = (T, \ell)$ be a labeled $\mathcal{C}^+$-triangulation of the interior of a polygon H. We denote the set of separators of $\mathbf{T}$ by $S_\mathbf{T}$. We define

$$c_\mathrm{P}(S_\mathbf{T}) = \sum_{e \in S_\mathbf{T}} c_\mathrm{P}(e) \text{ and } c_\mathrm{A}(T) = \sum_{\substack{\Delta \in T,\\ \ell(\Delta)=1}} c_\mathrm{A}(\Delta),$$

where $c_\mathrm{P}(e)$ denotes the length of e and $c_\mathrm{A}(\Delta)$ denotes the area of Δ. The *costs* of $\mathbf{T}$ are then defined as

$$c(\mathbf{T}) = \lambda \cdot c_\mathrm{P}(S_\mathbf{T}) + (1-\lambda) \cdot c_\mathrm{A}(T).$$

For any $e \in \mathcal{C}^+ \setminus \mathcal{C}$ we define $c_\mathrm{P}(e) = \infty$. Thus, we have $c(\mathbf{T}) < \infty$ if and only if $S_\mathbf{T} \subseteq \mathcal{C}$. We call a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of H *optimal* if there is no other labeled $\mathcal{C}^+$-triangulation $\mathbf{T}'$ of H with $c(\mathbf{T}') < c(\mathbf{T})$.

Next, we show that a labeled $\mathcal{C}^+$-triangulation $\mathbf{T} = (T, \ell)$ that is optimal can be recursively constructed based on optimal sub-triangulations. Let $G^\star$ be the restricted dual graph of T, i.e., for each triangle $G^\star$ has a node and two nodes are adjacent iff the corresponding triangles are adjacent in T; see Figure 8c.

Lemma 1 *The restricted dual graph $G^\star$ of a $\mathcal{C}^+$-triangulation T of D is a binary tree.*

Proof. As each edge of T starts and ends at the boundary of D, each edge of T splits D into two disjoint regions. Hence, $G^\star$ is a tree. Further, since each node of $G^\star$ corresponds to a triangle of T, each node of $G^\star$ has at most two child nodes. □

We call $G^\star$ a *decomposition tree* of D. Let ρ be the node of $G^\star$ that corresponds to the triangle of T that is adjacent to the cut edge $e^\star$ of D; as $e^\star$ is a boundary edge of D, this triangle is uniquely defined. We assume that ρ is the root of $G^\star$; see Figure 8c. Let $G^\star_u$ be an arbitrary sub-tree of $G^\star$ that is rooted at a node u of $G^\star$. Further, let e_u be the edge of the triangle Δ_u of u that is not adjacent to the triangles of the child nodes of u; we call e_u the *base edge* of Δ_u. The triangles of the nodes of $G^\star_u$ form a $\mathcal{C}^+$-triangulation T_u of the pocket $A_u = D[e_u] + e_u$ of e_u. Thus, $G^\star_u$ is a decomposition tree of A_u. A labeled $\mathcal{C}^+$-sub-triangulation $\mathbf{T}_u = (T_u, \ell_u)$ consists of the $\mathcal{C}^+$-triangulation T_u of A_u with $T_u \subseteq T$ and the labeling ℓ_u with $\ell_u(\Delta) = \ell(\Delta)$ for every $\Delta \in T_u$.

Lemma 2 *Let $\mathbf{T}$ be a labeled $\mathcal{C}^+$-triangulation of D that is optimal. Let $\mathbf{T}_u = (T_u, \ell_u)$ be the labeled $\mathcal{C}^+$-sub-triangulation of $\mathbf{T}$ rooted at the node u and let $\mathbf{T}'_u = (T'_u, \ell'_u)$ be an arbitrary labeled $\mathcal{C}^+$-triangulation of the same region. We denote the triangles of $\mathbf{T}_u$ and $\mathbf{T}'_u$ adjacent to e_u by Δ_u and Δ'_u, respectively.*

If Δ_u and Δ'_u have the same labels, i.e., $\ell_u(\Delta_u) = \ell'_u(\Delta'_u)$, then $c(\mathbf{T}_u) \leq c(\mathbf{T}'_u)$.

The proof is deferred to the full version [9]. We use Lemma 2 for a dynamic programming approach that yields a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of D that is optimal.

Lemma 3 *Let $\mathbf{T}$ be a labeled $\mathcal{C}^+$-triangulation of D that is optimal and has cost $c(\mathbf{T}) < \infty$. The separators of $\mathbf{T}$ form an optimal $\mathcal{C}$-hull of P.*

Proof. We show the following two claims, which proves the lemma. (1) For every $\mathcal{C}$-hull Q of P there is a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of D such that the separators of $\mathbf{T}$ form Q and $c(\mathbf{T}) = c(Q)$. (2) For every labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of D with $c(\mathbf{T}) < \infty$ the separators of $\mathbf{T}$ form a $\mathcal{C}$-hull Q with $c(\mathbf{T}) = c(Q)$.

Claim 1. Let Q be a $\mathcal{C}$-hull of P. By the definition of $\mathcal{C}^+$ there is a $\mathcal{C}^+$-triangulation T of D such that Q is part of T. We define the labeling ℓ such that $\ell(\Delta) = 1$

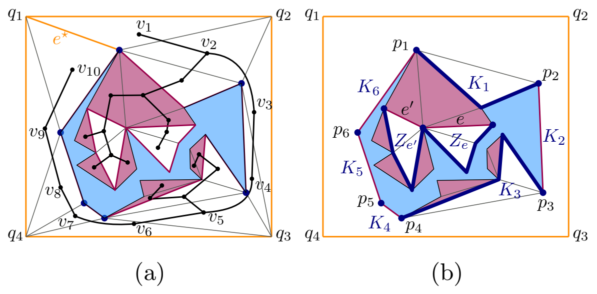


Figure 9: Proof of Lemma 3. (a) The triangles incident to the vertices q_1, q_2, q_3 and q_4 form a path in the dual graph of the labeled triangulation $\mathbf{T}$. (b) The vertices $p_1, \ldots, p_5$ form a $\mathcal{C}^+$-hull of P containing all active triangles (red) of $\mathbf{T}$.

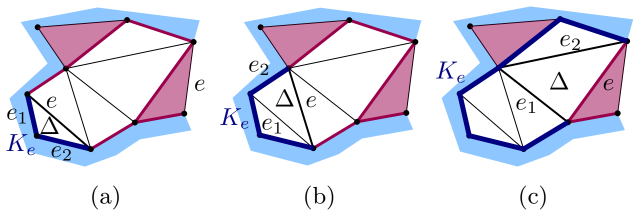


Figure 10: Inductive construction of the boundary path K_e of an edge e that is a base edge of an inactive triangle Δ. (a) Base case. (b) e_1 is a base edge of an inactive triangle, and e_2 is a separator. (c) Both e_1 and e_2 are base edges of inactive triangles.

for every triangle $\Delta \in T$ that is contained in the interior of Q and $\ell(\Delta) = 0$ for every other triangle $\Delta \in T$. Hence, the separators of the labeled $\mathcal{C}^+$-triangulation $\mathbf{T} = (T, \ell)$ are the edges of Q. Further, by the construction of $\mathbf{T}$ we have $c(\mathbf{T}) = c(Q)$. This proves Claim 1.

Claim 2. Let $\mathbf{T} = (T, \ell)$ be a $\mathcal{C}^+$-triangulation of D with $c(\mathbf{T}) < \infty$ and let $S_\mathbf{T}$ be the separators of $\mathbf{T}$. By the definition of the costs of $\mathbf{T}$ we have $S_\mathbf{T} \subseteq \mathcal{C}$. Moreover, as T is a triangulation, the edges in $S_\mathbf{T}$ do not cross each other. We show that the edges in $S_\mathbf{T}$ form a $\mathcal{C}$-hull Q with $c(Q) = c(\mathbf{T})$. Let $G^\star$ be the dual graph of T. As the diagonal edges of the containing box $\mathcal{B}$ intersect P, each triangle of T that is incident to one of the vertices of $\mathcal{B}$ is also incident to a vertex of P; see Figure 9a. The vertices of the triangles incident to the vertices of $\mathcal{B}$ form a path $v_1, \ldots, v_k$ in $G^\star$ such v_1 is the root of $G^\star$ and v_k is a leaf. We denote the triangles represented by this path by $\Delta_1, \ldots, \Delta_k$, respectively.

Let $p_1, \ldots, p_l$ be the vertices of P in the order as they are incident to the triangles $\Delta_1, \ldots, \Delta_k$ in clockwise order; see Figure 9a. We define $p_{l+1} = p_1$. The vertices $p_1, \ldots, p_l$ form a weakly-simple polygon Q' that contains P; if P crossed Q', this would contradict that the vertices are incident to the disjoint triangles $\Delta_1, \ldots, \Delta_k$. We observe that Q' is a $\mathcal{C}^+$-hull of P without holes. Let $T' \subseteq T$ be the set of triangles that are contained in Q' and let E' be the edges of these triangles. We first show that for each edge $e \in E'$ that is a base edge of an inactive triangle in $\mathbf{T}$ there is a path K_e in the pocket of e such that (1) K_e only consists of edges from $S_\mathbf{T}$, (2) K_e connects the endpoints of e, and (3) the polygon $K_e + e$ only contains inactive triangles of $\mathbf{T}$. We call K_e the *boundary path* of e; see Figure 10. Later, we use these boundary paths to assemble Q.

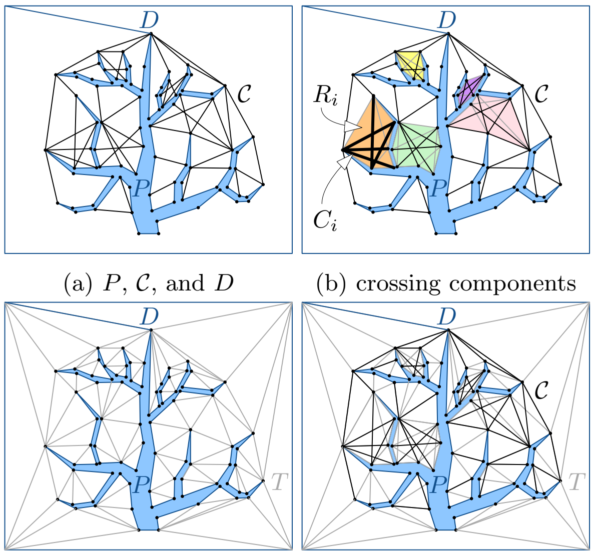
Let Δ be the inactive triangle of which e is the base edge and let e_1 and e_2 be the other two edges of Δ. We do an induction over the number of triangles of $\mathbf{T}$ that are contained in the pocket of e. If the pocket of e only contains Δ, both edges e_1 and e_2 are edges of P; see Figure 10a. Hence, by definition they are separators. We define K_e as the path $e_1 + e_2$, which satisfies the three requirements above. So assume that the pocket of e contains more than one triangle; see Figure 10b–c. If e_1 is not a separator, then it is the base edge of an inactive triangle. Hence, by induction there is a path K_{e_1} that satisfies the requirements above. If e_1 is a separator, we define $K_{e_1} = e_1$. In the same way we define a path K_{e_2} for the edge e_2. The concatenation $K_{e_1} + K_{e_2}$ forms a path that satisfies the requirements above, which proves the existence of the boundary path for an edge $e \in E'$.

We now describe the construction of the boundary of Q. For a pair p_i, p_{i+1} with $1 \leq i < l$ the adjacent triangle incident to one of the vertices of $\mathcal{B}$ is inactive. Let $K_i = p_ip_{i+1}$ if p_ip_{i+1} is a separator. Otherwise, p_ip_{i+1} is the base edge of an inactive triangle in $\mathbf{T}$. Thus, it has a boundary path $K_{p_ip_{i+1}}$ and we define K_i as K_e. The concatenation $K_1 + \cdots + K_l$ forms the boundary B of a weakly-simple polygon Q that encloses P; see Figure 9b. By construction it consists of edges from $\mathcal{C}$.

Finally, we show how to construct the holes of Q. Let $e \in S_\mathbf{T}$ be a separator that is contained in the interior of B and that is a base edge of an inactive triangle; see e and e' in Figure 9b. The polygon Z_e that consists of e and the boundary path K_e only contains inactive triangles of $\mathbf{T}$ and is entirely contained in B. Further, for any pair e and e' of such separators in the interior of B the interiors of the polygons Z_e and $Z_{e'}$ are disjoint. Hence, we set these polygons to be the holes of Q. Thus, we obtain a $\mathcal{C}$-hull Q of P with holes such that the inactive triangles of $\mathbf{T}$ lie in the exterior of Q, while all active triangles lie in the interior of Q. This implies that $c(Q) = c(\mathbf{T})$, which concludes the proof of Claim 2. □

4.3 From $\mathcal{C}$ to $\mathcal{C}^+$

Solving ShortcutHull relies on the considered enrichment $\mathcal{C}^+$. For an edge $e \in \mathcal{C}^+$ let δ_e be the number of triangles that can be formed by e and two other edges from $\mathcal{C}^+$, and let $\delta(\mathcal{C}^+)$ be the maximum δ_e over all edges e in $\mathcal{C}^+$. In Section 5 we show that the problem can be solved in $O(|\mathcal{C}^+| \cdot \delta(\mathcal{C}^+))$ time.



(a) P, $\mathcal{C}$, and D (b) crossing components (c) edges E_T (d) $\mathcal{C}^+$

Figure 11: Obtaining the enrichment $\mathcal{C}^+$ from $\mathcal{C}$.

A simple choice for $\mathcal{C}^+$ is the set of all edges that lie in D and connect vertices of D. It is an enrichment of $\mathcal{C}$ as it contains any choice of $\mathcal{C}$ and any triangulation of D that is based on the vertices of D is a subset of $\mathcal{C}^+$.

Observation 5 *There is an enrichment $\mathcal{C}^+$ of $\mathcal{C}$ with $|\mathcal{C}^+| \in O(n^2)$ and $\delta(\mathcal{C}^+) \in O(n)$.*

If $\mathcal{C}$ has no crossings, we can do much better. We first observe that the edges of any triangulation T of the sliced donut D are an enrichment of $\mathcal{C}$ and D if $\mathcal{C}$ is a subset of these edges. Hence, we can define an enrichment as the set of edges of a triangulation T of D such that the edges of $\mathcal{C}$ are part of T; for this purpose we can for example utilize constrained Delaunay triangulations, but also other triangulations are possible.

Observation 6 *If the edges in $\mathcal{C}$ do not cross, $\mathcal{C}$ has an enrichment $\mathcal{C}^+$ with $|\mathcal{C}^+| \in O(n)$ and $\delta(\mathcal{C}^+) \in O(1)$.*

In the following we generalize both constructions of $\mathcal{C}^+$ and relate $|\mathcal{C}^+|$ and $\delta(\mathcal{C}^+)$ to the number n of vertices of P and the spatial complexity χ of $\mathcal{C}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_h$ be subsets of $\mathcal{C}$ such that two edges $e \in \mathcal{C}_i$ and $e' \in \mathcal{C}_j$ with $1 \leq i, j \leq h$ cross each other if and only if $i = j$; see Figure 11. We call $\mathcal{C}_i$ a *crossing component* of $\mathcal{C}$. Let R_i be the polygon in D with fewest edges, that is defined by vertices of P and contains C_i. We call R_i the *region* of C_i. Let $\mathcal{C}^+$ be the set of edges that contains (i) all edges of $\mathcal{C}$, (ii) the edges E_T of a constrained triangulation for the interior of D, and (iii) for each $1 \leq i \leq h$ the set E_{R_i} of all possible shortcuts of region R_i such that these start and end at vertices of R_i and are contained in D. Hence, an enrichment is of size $O(\chi^2 + n)$ as each region R_i has at most χ vertices.

Theorem 2 *There is an enrichment $\mathcal{C}^+$ of $\mathcal{C}$ with $|\mathcal{C}^+| \in O(\chi^2 + n)$ and $\delta(\mathcal{C}^+) \in O(\chi)$.*

Proof. Let $\mathcal{C}^+$ be the set of edges that contains all edges of $\mathcal{C}$, E_T, and $E_{R_1}, \ldots, E_{R_h}$. We show that $\mathcal{C}^+$ is an enrichment, by proving that for each set $\mathcal{C}' \subseteq \mathcal{C}$ of pair-wisely non-crossing edges there is a $\mathcal{C}^+$-triangulation T of D such that $\mathcal{C}'$ is part of T.

Observe that the regions $R_1, \ldots, R_h$ of crossing components induce a partition $\mathbf{R}$ of D that contains $R_1, \ldots, R_h$ and regions $R'_1, \ldots, R'_g$ partitioning $D \setminus \bigcup_{i=1}^{h} R_i$. Since an edge $e \in \mathcal{C}^+$ cannot cross the boundary of two regions $R, R' \in \mathbf{R}$, the triangulation of each region $R \in \mathbf{R}$ can be constructed independently.

Let E be the edges of $\mathcal{C}'$ that are contained in region $R \in \mathbf{R}$. If R is a region of a crossing component, $\mathcal{C}^+$ contains all shortcuts in this region. Since the edges of E are crossing-free, there exists a $\mathcal{C}^+$-triangulation of R that is constrained to E. Thus, the edges of E are part of a $\mathcal{C}^+$-triangulation of R. If R is not a region of a crossing component, the enrichment $\mathcal{C}^+$ contains the edges of a triangulation of D constrained to all edges of $\mathcal{C}$ that are contained in R. Since $E \subseteq \mathcal{C}$, this triangulation contains all edges of E. By joining the $\mathcal{C}^+$-triangulations for each region of the partition, we obtain a $\mathcal{C}^+$-triangulation of D such that $\mathcal{C}'$ is part of it. $\square$

5 Computing Optimal Shortcut Hulls with Holes

The core of our algorithm is a dynamic programming approach that recursively builds the decomposition tree of T as well as the labeling ℓ using the sliced donut D of the input polygon P and the input set of shortcut $\mathcal{C}$ as guidance utilizing Lemma 2. The algorithm consists of the following steps.

1. Create a containing box $\mathcal{B}$ and the sliced donut D of P and $\mathcal{B}$. Let $e^\star$ be the cut edge of D.
2. Create an enrichment $\mathcal{C}^+$ of $\mathcal{C}$ and D.
3. Create the geometric graph $G_{\mathcal{C}^+}$ based on $\mathcal{C}^+$. Let $\mathcal{T}$ be the set of triangles in $G_{\mathcal{C}^+}$.
4. Determine for each edge e of $G_{\mathcal{C}^+}$ the set $T_e \subseteq T$ of all triangles (e, e_1, e_2) in $G_{\mathcal{C}^+}$ such that e_1 and e_2 lie in the pocket of e.
5. Create two tables A and I such that they have an entry for each edge e of $G_{\mathcal{C}^+}$.
   - A[e]: minimal cost of a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of the pocket $D[e]{+}e$ s.t. the triangle adjacent to e is active.
   - I[e]: minimal cost of a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of the pocket $D[e]{+}e$ s.t. the triangle adjacent to e is inactive.
6. Starting at I[$e^\star$] apply a backtracking procedure to create a $\mathcal{C}^+$-triangulation $\mathbf{T}$ of D that is optimal. Return $\mathbf{T}$ and the corresponding optimal $\mathcal{C}$-hull Q of $\mathbf{T}$ (see proof of Lemma 3 for construction of Q).

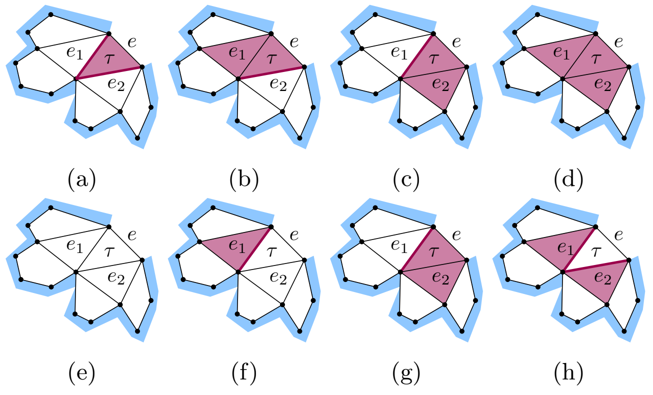


Figure 12: The possible cases for the (a)–(d) active (red) and (e)–(h) inactive cost of a triangle Δ.

We now explain Step 5 and Step 6 in greater detail.

Step 5. We compute the table entries of A and I in increasing order of the areas of the edges' pockets. Let e be the currently considered edge of $G_{\mathcal{C}^+}$. For a triangle $\Delta = (e, e_1, e_2) \in \mathcal{T}_e$ of e we define its *active cost* x_Δ as

$$x_\Delta = \sum_{i \in \{1,2\}} \min\{\mathrm{A}[e_i], I[e_i] + \lambda \cdot c_{\mathrm{P}}(e_i)\}.$$

Hence, x_Δ is the cost of a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}_e$ of the pocket $D[e]+e$ such that Δ is active and the subtriangulations of $\mathbf{T}_e$ restricted to the pockets $D[e_1]+e_1$ and $D[e_2]+e_2$ are optimal, respectively; see Figure 12 for the four possible cases.

$$\mathrm{A}[e] = \begin{cases} \infty & e \notin \mathcal{C} \\ \beta \cdot c_{\mathrm{A}}(e) & e \in \mathcal{C},\ \mathcal{T}_e = \emptyset, \\ \min\{x_\Delta \mid \Delta \in \mathcal{T}_e\} + \beta \cdot c_{\mathrm{A}}(e) & e \in \mathcal{C},\ \mathcal{T}_e \neq \emptyset, \end{cases}$$

where $\beta = (1-\lambda)$. Analogously, we define for Δ its *inactive cost* y_Δ as

$$y_\Delta = \sum_{i \in \{1,2\}} \min\{\mathrm{A}[e_i] + \lambda \cdot c_{\mathrm{P}}(e_i), I[e_i]\}.$$

Hence, y_Δ is the cost of a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}_e$ of the pocket $D[e]+e$ such that Δ is inactive and the subtriangulations of $\mathbf{T}_e$ restricted to the pockets $D[e_1]+e_1$ and $D[e_2]+e_2$ are optimal, respectively. We compute the entry I[e] as follows.

$$\mathrm{I}[e] = \begin{cases} \infty & e \in \mathcal{C} \text{ and } \mathcal{T}_e = \emptyset, \\ \min\{y_\Delta \mid \Delta \in \mathcal{T}_e\} & \text{otherwise.} \end{cases}$$

By the definition of the tables A and I and Lemma 2 it directly follows, that I[$e^\star$] is the cost of a labeled $\mathcal{C}^+$-triangulation of D that is optimal. In particular, by Lemma 3 the entry I[$e^\star$] is the cost of an optimal $\mathcal{C}$-hull.

Step 6. When filling both tables, we further store for each entry A[e] the triangle $(e, e_1, e_2) \in \mathcal{T}_e$ with minimum active cost. In particular, for the edge e_i (with $i \in \{1,2\}$) we store a pointer to the entry A[e_i] if A[e_i] < I[e_i] + $\lambda \cdot c_{\mathrm{P}}(e_i)$ and a pointer to the entry I[e_i] otherwise. Similarly, we store for each entry I[e] the triangle $(e, e_1, e_2) \in T_e$ with minimum inactive cost. In particular, for the edge e_i (with $i \in \{1,2\}$) we store a pointer to the entry I[e_i] if I[e_i] < A[e_i] + $\lambda \cdot c_{\mathrm{P}}(e_i)$ and a pointer to the entry A[e_i] otherwise. Starting at the entry I[$e^\star$], we follow the pointers and collect for each encountered entry its triangle —if such a triangle does not exist, we terminate the traversal. If the entry belongs to A we label Δ active and if it belongs to I, we label Δ inactive. The set T of collected triangles forms a labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of D that is optimal. By Lemma 3 the separators of $\mathbf{T}$ form an optimal $\mathcal{C}$-hull.

Running Time. The first step clearly runs in $O(n)$ time. By Theorem 2 there is an enrichment $\mathcal{C}^+$ of $\mathcal{C}$ and D that has size $O(\chi^2+n)$. It can be easily constructed in $O(\chi^3+\chi n)$ time, which dominates the running times of Step 2, Step 3 and Step 4. Further, for each edge e of $G_{\mathcal{C}^+}$ the set $\mathcal{T}_e$ contains $\delta(\mathcal{C}^+)$ triangles. Hence, filling the tables A and I takes $O(|\mathcal{C}^+| \cdot \delta(\mathcal{C}^+))$ time. Hence, by Theorem 2 we obtain $O(\chi^3+\chi n)$ running time. The backtracking takes the same time.

Theorem 3 ShortcutHull *can be solved in $O(\chi^3 + n\chi)$ time. In particular, it is solvable in $O(n^3)$ time in general and in $O(n)$ time if the edges in $\mathcal{C}$ do not cross.*

6 Edge and Bend Restricted Shortcut Hulls

In this section, we discuss two variants of ShortcutHull in which we restrict the number of edges and bends of the computed shortcut hull. These restrictions are particularly interesting for the simplification of geometric objects as they additionally allow us to easily control the complexity of the simplification.

6.1 Restricted $\mathcal{C}$-Hull: Number of Edges

Next, we show how to find a $\mathcal{C}$-hull Q that balances its enclosed area and perimeter under the restriction that it consists of at most k edges. We say that Q is optimal *restricted to at most k edges*, if there is no other $\mathcal{C}$-hull Q' with at most k edges and $c(Q') < c(Q)$.

k-EdgeShortcutHull.

given: weakly-simple polygon P with n vertices and connected exterior, set $\mathcal{C}$ of shortcuts of P, $\lambda \in [0,1]$, and $k \in \mathbb{N}$

find: optimal $\mathcal{C}$-hull Q of P (if it exists) restricted to at most k edges.

To solve k-EdgeShortcutHull we adapt Step 5 of the algorithm presented in Section 5. We extend the tables

A and I by an additional dimension of size k modelling the budget of edges that we have left for the particular instance. For a shortcut $e \in \mathcal{C}^+$ and a *budget* b we interpret the table entries as follows.

- $\mathrm{A}[e][b]$: cost of labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of the pocket of e s.t. $\mathbf{T}$ is optimal, the triangle adjacent to e is active and $\mathbf{T}$ contains at most b separators.
- $\mathrm{I}[e][b]$: cost of labeled $\mathcal{C}^+$-triangulation $\mathbf{T}$ of the pocket of e s.t. $\mathbf{T}$ is optimal, the triangle adjacent to e is inactive and $\mathbf{T}$ contains at most b separators.

Let e be the currently considered edge of $G_{\mathcal{C}^+}$ when filling the tables. For a triangle $\Delta = (e, e_1, e_2) \in \mathcal{T}_e$ of e its active and inactive costs depend on the given budgets b_1 and b_2 with $1 \leq b_1, b_2 \leq k$ that we intend to use for the sub-instances attached to e_1 and e_2.

$$x_{\Delta,b_1,b_2} = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i][b_i], I[e_i][b_i - 1] + \lambda \cdot c_{\mathrm{P}}(e_i)\}$$

$$y_{\Delta,b_1,b_2} = \sum_{i\in\{1,2\}} \min\{\mathrm{A}[e_i][b_i - 1] + \lambda \cdot c_{\mathrm{P}}(e_i), I[e_i][b_i]\}$$

Hence, for the case that $e \in \mathcal{C}$ and $\mathcal{T}_e \neq \emptyset$ we define

$$\mathrm{A}[e][b] = \min\{x_{\Delta,b_1,b_2} \mid \Delta \in \mathcal{T}_e,\, b_1 + b_2 = b\} + \beta \cdot c_{\mathrm{A}}(e),$$

where $\beta = (1 - \lambda)$. There are b possible choices of b_1 and b_2 that satisfy $b_1 + b_2 = b$. Thus, we can compute $A[e][b]$ in $O(b)$ time. For the remaining cases we define

$$\mathrm{A}[e][b] = \begin{cases} \infty & e \notin \mathcal{C} \\ \beta \cdot c_{\mathrm{A}}(e) & e \in \mathcal{C},\, \mathcal{T}_e = \emptyset, \end{cases}$$

which can be computed in $O(1)$ time. Moreover, for the case that $e \notin \mathcal{C}$ or $\mathcal{T}_e \neq \emptyset$ we define

$$\mathrm{I}[e][b] = \min\{y_{\Delta,b_1,b_2} \mid \Delta \in \mathcal{T}_e, b_1 + b_2 = b\}.$$

For the same reasons as before we can compute $\mathrm{I}[e][b]$ in $O(1)$ time. For $e \in \mathcal{C}$ or $\mathcal{T}_e \neq \emptyset$ we define $\mathrm{I}[e][b] = \infty$. Finally, to cover border cases we set $\mathrm{A}[e][0] = \infty$ and $\mathrm{I}[e][0] = \infty$. Altogether, the entry $\mathrm{I}[e^\star][k]$ contains the cost of an optimal $\mathcal{C}$-hull that is restricted to k edges. Apart from minor changes in Step 6 the other parts of the algorithm remain unchanged.

Running time. Compared to the algorithm of Section 5 the running time of computing a single entry increases by a factor of $O(k)$. Further, there are $O(k)$ times more entries to be computed, which yields that the running time increases by a factor of $O(k^2)$.

Theorem 4 *The problem k-EdgeShortcutHull can be solved in $O(k^2(\chi^3 + n\chi))$ time. In particular, it can be solved in $O(k^2n^3)$ time in general and in $O(k^2n)$ time if the edges in $\mathcal{C}$ do not cross.*

6.2 Restricted $\mathcal{C}$-Hull: Number of Bends

A slightly stronger constraint than restricting the number of edges is restricting the number of bends of a $\mathcal{C}$-hull. Formally, we call two consecutive edges of a simply-weakly polygon a *bend* if the enclosed angle is not 180°. We say that Q is *optimal restricted to at most k bends* if there is no other $\mathcal{C}$-hull Q' with at most k bends and $c(Q') < c(Q)$.

k-BendShortcutHull.
given: weakly-simple polygon P with n vertices and connected exterior, set $\mathcal{C}$ of shortcuts of P, $\lambda \in [0, 1]$, and $k \in \mathbb{N}$
find: optimal $\mathcal{C}$-hull Q of P (if it exists) that is restricted to at most k bends.

If the vertices of P are in general position, i.e., no three vertices lie on a common line, a $\mathcal{C}$-hull Q of P is optimal restricted to at most k bends if and only if it is optimal restricted to k edges. Hence, in that case we can solve k-BendShortcutHull using the algorithm presented in Section 6.1. In applications, the case that the vertices of P are not in general position, occurs likely when the input polygon is, e.g., a schematic polygon or a polygon whose vertices lie on a grid. In that case, we add an edge p_1p_h to $\mathcal{C}$ for each sequence $p_1, \ldots, p_h$ of at least three vertices of P that lie on a common line; we add p_1p_h only if it lies in the exterior of P. The newly obtained set $\mathcal{C}'$ has $O(n^2)$ edges. Hence, compared to $\mathcal{C}$ it possibly has an increased spatial complexity with $\chi \in O(n)$. From Theorem 4 we obtain the next result.

Theorem 5 *The problem k-BendShortcutHull can be solved in $O(k^2 \cdot n^3)$ time.*

7 Relations to other Geometric Problems

We have implemented the algorithm presented in Section 5. For example, computing a shortcut hull for the instance shown in Figure 2 one run of the dynamic programming approach (Step 5) took 400ms on average. This suggests that despite its cubic worst-case running time our algorithm is efficient enough for real-world applications. However, more experiments are needed to substantiate this finding.

Balancing the Costs of Area and Perimeter In Figure 1 we display a series of optimal $\mathcal{C}$-hulls[2]. We use the same polygon and the set of all possible shortcuts as input while increasing the parameter λ of the cost function. To find relevant values of λ we implemented a systematic search in the range $[0, 1]$. It uses the simple observation that with monotonically increasing λ the amount of area enclosed by an optimal shortcut hull increases monotonically. More in detail, we compute

[2]Figure 1b: $\lambda = 0.906$; Figure 1c: $\lambda = 0.995$; Figure 1e: $\lambda = 0.914$; Figure 1f: $\lambda = 0.975$

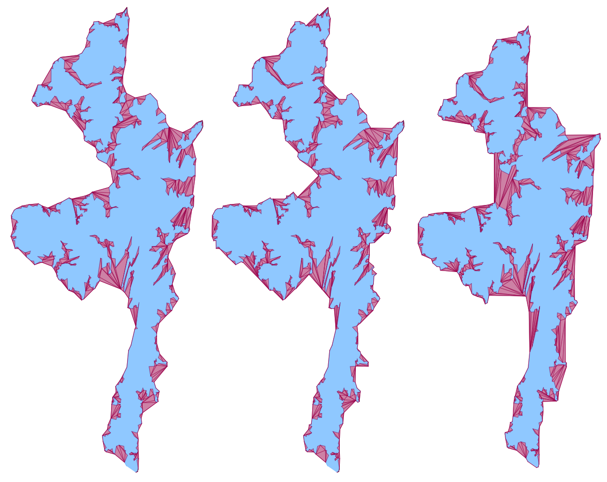


Figure 13: Simplification (a) and schematization (b)–(c) of the main island of Shetland.

the optimal shortcut hull for $\lambda = 0$ and $\lambda = 1$. If the area cost c_A of these shortcut hulls differ, we recursively consider the intervals $[0, 0.5]$ and $[0.5, 1]$ for the choice of λ similar to a binary search. Otherwise, we stop the search.

As presented in Equation 1, we consider costs for the area and perimeter in ShortcutHull. The second column of Figure 1 shows a result for a small value of λ, i.e., the costs for the area are weighted higher. As expected the resulting optimal $\mathcal{C}$-hull is rather close to the input polygon. In contrast, the last column of Figure 1 shows the optimal $\mathcal{C}$-hull for a larger λ-value. We particularly obtain holes that represent large areas enclosed by the polygon, while small gaps are filled.

Simplification and Schematization of Simple Polygons In the following, we discuss how our approach relates to typical measures for simplification and schematization. These are the number of edges, the number of bends [20] or the perimeter [51], which are implemented by shortcut hulls; e.g., Figure 13a shows the simplification of the border of the main island of Shetland by a $\mathcal{C}$-hull as defined in ShortcutHull. The schematization of a polygon is frequently implemented as a hard constraint with respect to a given set O of edge orientations. For schematizing a polygon with $\mathcal{C}$-hulls, we outline two possibilities: a non-strict and a strict schematization. For the non-strict schematization, we adapt the cost function of the shortcuts such that edges with an orientation similar to an orientation of O are cheap while the others are expensive; see Figure 13b for O consisting of horizontal, vertical, and diagonal orientations and Figure 13c for O consisting of the horizontal and vertical orientations. The strict schematization restricts the set $\mathcal{C}$ of shortcuts, such that each edges' orientation is from O. For example, one can define $\mathcal{C}$ based on an underlying grid that only uses orientations from O. We then need to take special care about the connectivity of $\mathcal{C}$, e.g., by also having all edges of the input polygon in $\mathcal{C}$.

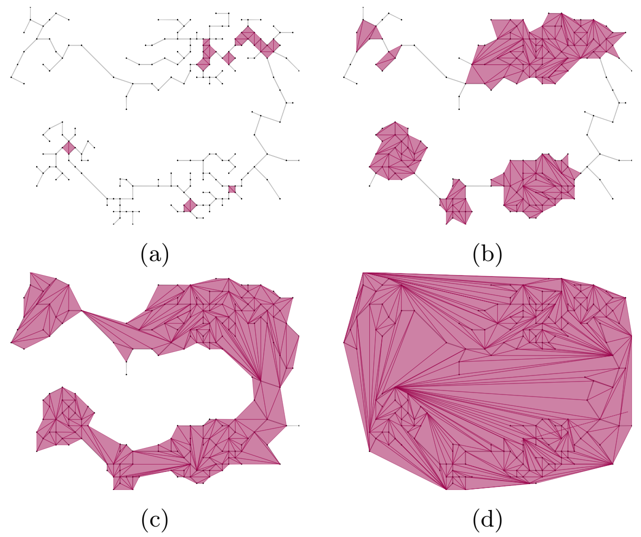


Figure 14: Optimal $\mathcal{C}$-hulls for increasing values of λ for a point set using a minimum spanning tree as basis.

Aggregation of Multiple Objects and Clustering We can adapt $\mathcal{C}$-hulls for multiple geometric objects, e.g. a point set. We suggest to use a geometric graph that contains all vertices of the input geometries, all edges of the input geometries and is connected as input for problem ShortcutHull, e.g., a minimum spanning tree of the point set; see Fig 14. With increasing λ-value the regions of the shortcut hull first enclosed are areas with high density. By removing all edges of Q that are not adjacent to the interior of Q, we possibly receive multiple polygons which each can be interpreted as a cluster.

8 Conclusion

We introduced a simplification technique for polygons that yields shortcut hulls, i.e., crossing-free polygons that are described by shortcuts and that enclose the input polygon. In contrast to other work, we consider the shortcuts as input. We introduced a cost function of a shortcut hull that is a linear combination of the covered area and the perimeter. Computing optimal shortcut hulls without holes takes $O(n^2)$ time. For the case that we permit holes we presented an algorithm based on dynamic programming that runs in $O(n^3)$ time. If the input shortcuts do not cross it runs in $O(n)$ time.

We plan on considering (i) the bends as part of the cost function, (ii) more general shortcuts, e.g. allowing one bend per shortcut, and (iii) optimal spanning trees for the case of multiple input geometries.

Acknowledgements This work has partially been funded by the German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 - PhenoRob, and by NSF (Mitchell, CCF-2007275).

References

[1] M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Discret. Comput. Geom.*, 43(3):497–515, 2010.

[2] J. Adegeest, M. H. Overmars, and J. Snoeyink. Minimum-link c-oriented paths: Single-source queries. *Int. J. Comput. Geom. Appl.*, 4(1):39–51, 1994.

[3] T. Ai, S. Ke, M. Yang, and J. Li. Envelope generation and simplification of polylines using Delaunay triangulation. *Int. J. Geogr. Inf. Sci.*, 31(2):297–319, 2017.

[4] T. Ai, Y. Liu, and J. Chen. The hierarchical watershed partitioning and data simplification of river network. In *Progress in spatial data handling*, pages 617–632. Springer, 2006.

[5] T. Ai, Q. Zhou, X. Zhang, Y. Huang, and M. Zhou. A simplification of ria coastline with geomorphologic characteristics preserved. *Marine Geodesy*, 37(2):167–186, 2014.

[6] C. Alegría, D. Orden, C. Seara, and J. Urrutia. Efficient computation of minimum-area rectilinear convex hull under rotation and generalizations. *J. Glob. Optim.*, 79(3):687–714, 2021.

[7] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint jordan chains. *Int. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[8] T. Barkowsky, L. J. Latecki, and K. Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, volume 1849 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2000.

[9] A. Bonerath, J.-H. Haunert, J. S. B. Mitchell, and B. Niedermann. Shortcut hulls: Vertex-restricted outer simplifications of polygons. *CoRR*, abs/2106.13620, 2021.

[10] A. Bonerath, J.-H. Haunert, and B. Niedermann. Tight Rectilinear Hulls of Simple Polygons. In *Proc. of the 36th European Workshop on Computational Geometry, EuroCG 2020*.

[11] A. Bonerath, B. Niedermann, and J. Haunert. Retrieving α-shapes and schematic polygonal approximations for sets of points within queried temporal ranges. In *Proc. of 27th Int. Conf. on Advances in Geographic Information Systems, SIGSPATIAL 2019*, pages 249–258. ACM, 2019.

[12] K. Buchin, W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving simplification and schematization of polygonal subdivisions. *ACM Trans. Spatial Algorithms Syst.*, 2(1):2:1–2:36, 2016.

[13] D. Burghardt, S. Schmid, and J. Stoter. Investigations on cartographic constraint formalisation. In *10th ICA Workshop on Generalization and Multiple Representation*, volume 19, page 2, 2007.

[14] B. Chazelle. Triangulating a simple polygon in linear time. *Discret. Comput. Geom.*, 6:485–524, 1991.

[15] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[16] F. Y. L. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained voronoi diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28(2):471–486, 1998.

[17] J. Damen, M. van Kreveld, and B. Spaan. High quality building generalization by extending the morphological operators. In *11th ICA Workshop on Generalization and Multiple Representation*, pages 1–12, 2008.

[18] J. J. Daymude, R. Gmyr, K. Hinnenthal, I. Kostitsyna, C. Scheideler, and A. W. Richa. Convex hull formation for programmable matter. In *Proc. of 21st Int. Conf. on Distributed Computing and Networking, ICDCN 2020*, pages 2:1–2:10. ACM, 2020.

[19] M. de Berg, W. Meulemans, and B. Speckmann. Delineating imprecise regions via shortest-path graphs. In *Proc. of 19th Int. Conf. on Advances in Geographic Information Systems, SIGSPATIAL 2011*, pages 271–280. ACM, 2011.

[20] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: Int. J. for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[21] A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013.

[22] M. Duckham, L. Kulik, M. F. Worboys, and A. Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognit.*, 41(10):3224–3236, 2008.

[23] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Trans. Information Theory*, 29(4):551–558, 1983.

[24] R. Estkowski and J. S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proc. of 17th Symp. on Computational Geometry, SOCG 2001*, pages 40–49. ACM, 2001.

[25] A. Filtser and O. Filtser. Static and streaming data structures for fréchet distance queries. In *Proc. of Symp. on Discrete Algorithms, SODA 2021*, pages 1150–1170. SIAM, 2021.

[26] E. Fink and D. Wood. *Restricted-Orientation Convexity*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[27] J. García and J. Fdez-Valdivia. Boundary simplification in cartography preserving the characteristics of the shape features. *Computers & Geosciences*, 20(3):349–368, 1994.

[28] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inf. Process. Lett.*, 7(4):175–179, 1978.

[29] J. Haunert and A. Wolff. Optimal and topologically safe simplification of building footprints. In *Proc. of 18th Int. Symp. on Advances in Geographic Information Systems, SIGSPATIAL 2010*, pages 192–201. ACM, 2010.

[30] J.-H. Haunert, A. Wolff, et al. Optimal simplification of building ground plans. In *Proc. of 21st ISPRS Congress*, pages 372–378, 2008.

[31] B. Jacobsen, M. Wallinger, S. G. Kobourov, and M. Nöllenburg. Metrosets: Visualizing sets as metro maps. *IEEE Trans. Vis. Comput. Graph.*, 27(2):1257–1267, 2021.

[32] C. B. Jones, G. L. Bundy, and M. J. Ware. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22(4):317–331, 1995.

[33] T. C. Kao and D. M. Mount. Incremental construction and dynamic maintenance of constrained Delaunay triangulations. In *Proc. of 4th Canadian Conf. on Computational Geometry, CCCG 1992*, pages 170–175, 1992.

[34] D.-T. Lee and A. K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete & Computational Geometry*, 1(3):201–217, 1986.

[35] D. T. Lee, C. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. *Discret. Appl. Math.*, 70(3):185–215, 1996.

[36] C. Li, Y. Yin, X. Liu, and P. Wu. An automated processing method for agglomeration areas. *ISPRS Int. J. Geo Inf.*, 7(6):204, 2018.

[37] J. Li and T. Ai. A triangulated spatial model for detection of spatial characteristics of GIS data. In *Proc. of Int. Conf. on Progress in Informatics and Computing, PIC 2010*, volume 1, pages 155–159. IEEE, 2010.

[38] T. Mendel. Area-preserving subdivision simplification with topology constraints: Exactly and in practice. In *Proc. of 20th Workshop on Algorithm Engineering and Experiments, ALENEX 2018*, pages 117–128. SIAM, 2018.

[39] W. Meulemans. *Similarity measures and algorithms for cartographic schematization.* PhD thesis, Mathematics and Computer Science, 2014.

[40] W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving subdivision schematization. In *Proc. of 6th Int. Conf. on Geographic Information Science, GIScience 2010*, volume 6292 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2010.

[41] J. S. B. Mitchell, V. Polishchuk, and M. Sysikaski. Minimum-link paths revisited. *Comput. Geom.*, 47(6):651–667, 2014.

[42] G. Neyer. Line simplification with restricted orientations. In *Proc. of 6th Workshop on Algorithms and Data Structures, WADS '99*, volume 1663 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 1999.

[43] M. Nöllenburg. A survey on automated metro map layout methods. In *Schematic Mapping Workshop 2014*, 2014.

[44] J. L. G. Pallero. Robust line simplification on the plane. *Comput. Geosci.*, 61:152–159, 2013.

[45] G. J. E. Rawlins and D. Wood. Optimal computation of finitely oriented convex hulls. *Inf. Comput.*, 72(2):150–166, 1987.

[46] A. Sayidov and R. Weibel. Generalization of geological maps: Aggregation and typification of polygon groups. 2019.

[47] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. of 16th Symp. on Foundations of Computer Science, FOCS 1975*, pages 151–162. IEEE Computer Society, 1975.

[48] J. R. Shewchuk and B. C. Brown. Fast segment insertion and incremental construction of constrained Delaunay triangulations. *Comput. Geom.*, 48(8):554–574, 2015.

[49] B. Speckmann and K. Verbeek. Homotopic c-oriented routing with few links and thick edges. *Comput. Geom.*, 67:11–28, 2018.

[50] S. Steiniger, D. Burghardt, and R. Weibel. Recognition of island structures for map generalization. In *Proc. of 14th Int. Symp. on Geographic Information Systems, SIGSPATIAL 2006*, pages 67–74. ACM, 2006.

[51] E. R. Tufte. The visual display of quantitative information. *The Journal for Healthcare Quality (JHQ)*, 7(3):15, 1985.

[52] T. C. van Dijk, A. van Goethem, J. Haunert, W. Meulemans, and B. Speckmann. Map schematization with circular arcs. In *Proc. of 8th Int. Conf. on Geographic Information Science, GIScience 2014*, volume 8728 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.

[53] A. van Goethem, W. Meulemans, B. Speckmann, and J. Wood. Exploring curved schematization of territorial outlines. *IEEE Trans. Vis. Comput. Graph.*, 21(8):889–902, 2015.

[54] M. van Kreveld, T. van Lankveld, and M. de Rie. (α, δ)-sleeves for reconstruction of rectilinear building facets. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 231–247. Springer, 2013.

[55] H. Wu, B. Niedermann, S. Takahashi, M. J. Roberts, and M. Nöllenburg. A survey on transit map layout - from design, machine, and human perspectives. volume 39, pages 619–646, 2020.

[56] X. Zhang and E. Guilbert. A multi-agent system approach for feature-driven generalization of isobathymetric line. In *Advances in Cartography and GIScience. Volume 1*, pages 477–495. Springer, 2011.

Stochastic Analysis of Empty-Region Graphs

Olivier Devillers[*] Charles Duménil[*]

Abstract

Given a set of points X, an empty-region graph is a graph in which $p, q \in X$ are neighbors if some region defined by (p, q) does not contain any point of X. We provide expected analyses of the degree of a point and the possibility of having *far* neighbors in such a graph when X is a planar Poisson point process. Namely the expected degree of a point in the empty axis-aligned-ellipse graph for a Poisson point process of intensity λ in the unit square is $\Theta(\ln \lambda)$. It is $\Theta(\ln \beta)$ if the ellipses are constrained to have an aspect ratio between 1 and $\beta > 1$, and $\Theta(\beta)$ when the aspect ratio is constrained but ellipses are not axis-aligned.

1 Introduction

We start by defining the notion of empty-region graph [2]:

Definition 1 *For each pair $(p, q) \in \mathbb{R}^d \times \mathbb{R}^d$, let $\mathcal{R}(p, q)$ be a family of regions. Consider a locally finite point set $X \subset \mathbb{R}^d$. We denote by $\mathcal{G}^{\emptyset}_{\mathcal{R}}(X)$ the graph on X in which p is a neighbor of q if and only if there exists an empty region in $\mathcal{R}(p, q)$.*

This notion unifies the classical Delaunay triangulation [3] where $\mathcal{R}(p, q)$ is the set of disks whose boundaries contains p and q, the Gabriel graph [6] where $\mathcal{R}(p, q)$ is reduced to the disk of diameter pq, the β-skeleton [7, 1], the empty-ellipse graph [4], the nearest neighbor-graph, the Θ-graphs, and the Yao graphs [9].

In this paper, we will assume that X is a Poisson point process in the plane and compute quantities like the expected degree of a point $p \in X$ in $\mathcal{G}^{\emptyset}_{\mathcal{R}}(X)$ or the probability that p has neighbors further than some threshold. Computing such quantities when $R(p, q)$ is a singleton, as for the Gabriel graph, is much easier than when it is a bigger set, as for Delaunay triangulation. To this aim, it is interesting to try to get upper and lower bounds by comparing empty-region graphs. This idea was already used by Devroye, Lemaire and Moreau [5] to bounds the size of the Delaunay triangulation by the sizes of the Gabriel graph and the half-moon graph.

[*]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France. `FirstName.LastName@inria.fr`. This research has been funded by grant ANR-17-CE40-0017 of the French National Research Agency (Aspag)

In this paper we formalize the process through two lemmas: the Combination lemma and the Partition lemma, and we illustrate these tools with empty-ellipse graphs. In a forthcoming paper we apply these results to equations of higher degree appearing when parameterizing 3D surfaces.

2 First Example: Delaunay and Gabriel Graphs

2.1 Delaunay Triangulation

The Delaunay triangulation is the empty-region graph where $\mathcal{R}(p, q) = \{D(p, q, r); r \in \mathbb{R}^2\}$ and $D(p, q, r)$ is the open disk with p, q, and r on its boundary.

Although the expected degree of a random point in any kind of triangulation is well known to be 6 using Euler formula, we prove it using stochastic tools to illustrate the complexity of such a computation:

Theorem 2 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$ and p a point of $\mathbb{R}^2$. The expected degree $\mathbb{E}[\deg(p, \mathrm{Del})]$ of p in the Delaunay triangulation $\mathrm{Del}(X \cup \{p\})$ is 6.*

Proof. Without loss of generality, we assume that p is at the origin. Let $D(p, q, r)$ denote the open disk with p, q, and r on its boundary. The number of neighbors of p in $\mathrm{Del}(X \cup \{p\})$ is the number of distinct sets $\{q, r\}$ in X^2 with $q \neq r$ such that $D(p, q, r)$ does not contain any point of X. It is given by the random value: $\deg(p, \mathrm{Del}) = \frac{1}{2} \sum_{q \in X} \sum_{r \in X \setminus \{q\}} \mathbb{1}_{[D(p,q,r) \cap X = \emptyset]}$, where the factor $\frac{1}{2}$ corrected the double counting of each set $\{q, r\}$ in the sum. We compute the expectation of this formula:

$$\begin{aligned}\mathbb{E}[\deg(p, \mathrm{Del})] &= \mathbb{E}\left[\frac{1}{2} \sum_{q \in X} \sum_{r \in X \setminus \{q\}} \mathbb{1}_{[D(p,q,r) \cap X = \emptyset]}\right] \\ &= \frac{1}{2} \int_{\mathbb{R}^2} \int_{\mathbb{R}^2} \lambda^2 \, \mathbb{P}[D(p, q, r) \cap X = \emptyset] \, \mathrm{d}r \mathrm{d}q \\ &\qquad \text{by Slivnyak-Mecke Theorem [8]} \\ &= \frac{1}{2} \int_{\mathbb{R}^2} \int_{\mathbb{R}^2} \lambda^2 e^{-\lambda |D(p,q,r)|} \mathrm{d}r \mathrm{d}q \\ &\qquad \text{by definition of Poisson point process.}\end{aligned}$$

The computation of this integral is a bit technical and is given in appendix. It involves a Blaschke-Petkantschin like variables substitution to turn the cartesian coordinates of q and r in the coordinate of the center of $D(p,q,r)$ and two angles to place q and r on the boundary of $D(p,q,r)$.

It finally turns out that the value of this integral is 6, as anticipated. $\square$

2.2 Gabriel Graph and Half-Moon Graph

We now turn our interest to cases where $\mathcal{R}(p,q)$ is a singleton. We consider the three following possibilities: $\mathcal{R}(p,q) = \{\text{Gab}(p,q)\}$ the disk of diameter pq, $\mathcal{R}(p,q) = \{hm_r(p,q)\}$ the half-disk of diameter pq to the right of pq, and $\mathcal{R}(p,q) = \{hm_\ell(p,q)\}$ the half-disk of diameter pq to the left of pq. Then $\mathcal{G}^{\emptyset}_{\{\text{Gab}\}}$ is the Gabriel graph, $\mathcal{G}^{\emptyset}_{\{hm_r\}}$ is the right half-moon graph, and $\mathcal{G}^{\emptyset}_{\{hm_\ell\}}$ is the left half-moon graph. The half-moon graph is $\mathcal{G}^{\emptyset}_{\{hm_r\}} \cup \mathcal{G}^{\emptyset}_{\{hm_\ell\}}$.

Lemma 3 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$ and p a point of $\mathbb{R}^2$. The expected degree $\mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{\text{Gab}\}}\right)\right]$ of the origin p in the Gabriel graph $\mathcal{G}^{\emptyset}_{\{\text{Gab}\}}(X)$ is 4.*

Proof.

$$\begin{aligned}
\mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{\text{Gab}\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[\text{Gab}(p,q)\cap X=\emptyset]}\right] \\
&= \int_{q\in\mathbb{R}^2} \lambda\, \mathbb{P}\left[\text{Gab}(p,q)\cap X=\emptyset\right] \mathrm{d}q \\
&= \int_{\mathbb{R}^2} \lambda e^{-\lambda|\text{Gab}(p,q)|}\mathrm{d}q \\
&= \int_{\mathbb{R}^+}\int_0^{2\pi} \lambda e^{-\lambda\frac{\pi\rho^2}{4}}\rho\,\mathrm{d}\theta\mathrm{d}\rho \;= 4. \quad \square
\end{aligned}$$

Lemma 4 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$ and p a point of $\mathbb{R}^2$. Then*

$$\mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{hm_r\}}\right)\right] = \mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{hm_\ell\}}\right)\right] = 8.$$

Proof. By symmetry, we only do the computation for $\mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{hm_r\}}\right)\right]$.

$$\begin{aligned}
\mathbb{E}\left[\deg\left(p,\mathcal{G}^{\emptyset}_{\{hm_r\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[hm_r(p,q)\cap X=\emptyset]}\right] \\
&= \int_{\mathbb{R}^2} \lambda e^{-\lambda|hm_r(p,q)|}\mathrm{d}q \\
&= \int_{\mathbb{R}^+}\int_0^{2\pi} \lambda e^{-\lambda\frac{\pi\rho^2}{8}}\rho\,\mathrm{d}\theta\mathrm{d}\rho \;= 8. \square
\end{aligned}$$

As one can see, the fact that $\mathcal{R}(p,q)$ is a singleton made the computation much simpler than in the case of the Delaunay triangulation.

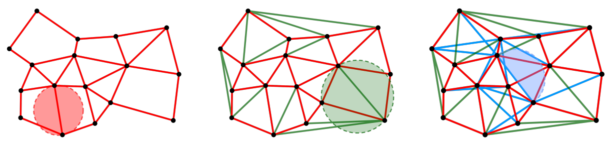


Figure 1: The Gabriel graph, on the left, is included in the Delaunay triangulation, in the middle, itself included in the half-moon graph, on the right.

2.3 Graph Relations

The following relations between the graphs are straightforward since any disk with p and q on its boundary contains either $hm_r(p,q)$ or $hm_\ell(p,q)$ (see Figure 1):

$$\begin{aligned}
&\mathcal{G}^{\emptyset}_{\{\text{Gab}\}} = \mathcal{G}^{\emptyset}_{\{hm_r\}} \cap \mathcal{G}^{\emptyset}_{\{hm_\ell\}}, \\
&\mathcal{G}^{\emptyset}_{\{\text{Gab}\}} \subset \text{Del} \subset \mathcal{G}^{\emptyset}_{\{hm_r\}} \cup \mathcal{G}^{\emptyset}_{\{hm_\ell\}} = \mathcal{G}^{\emptyset}_{\{hm_r,hm_\ell\}}.
\end{aligned}$$

From this, we deduce

$$\deg\left(p,\mathcal{G}^{\emptyset}_{\{\text{Gab}\}}\right) \le \deg(p,\text{Del}) \le \deg\left(p,\mathcal{G}^{\emptyset}_{\{hm_r,hm_\ell\}}\right)$$
$$4 \le \deg(p,\text{Del}) \le 8+8-4 = 12.$$

This result is weaker than the exact bound of Theorem 2 but the computations are much simpler. It also illustrates that, given regions of similar areas, the degree remains of equal order of magnitude. In that case, for two points p and q, $\text{Gab}(p,q)$ and $hm_r(p,q)$ or $hm_\ell(p,q)$ have both an area quadratic in the distance between p and q, and this induces a constant expected degree.

3 General Method

We propose a general method that both formalizes and generalizes the half-moon method to link the degree in general empty-region graphs to the degree in empty-region graphs defined by singletons. We formalize the following facts: *(i)* the Delaunay disks can be parameterized by their center on the bisector of pq, *(ii)* this bisector can be partitioned in two rays at the midle of pq, and *(iii)* each half-moon is contained in all disks centered on one of the rays.

In a more general setting, the general idea is *(i)* to identify a parameter space in $\mathbb{R}^k$ defining the regions, *(ii)* to partition this space in convex domains, and *(iii)* have inclusion relations for regions at the vertices of the partition.

The following lemma is instrumental for proving that if a set of region depends on k parameters and if the k-tuple of parameters belongs to a convex polyhedron P of $\mathbb{R}^k$ then, if we want to prove that all regions parameterized by P contain a given region, it is enough to prove this inclusion for the regions parameterized by the vertices of P. If P is not bounded, we can extend the lemma to limit points at infinity: for a point c going to infinity along some ray of $\mathbb{R}^k$ the region r_c has a limit. The result also holds using this limit regions. We will show below as a didactic example how this lemma can be applied on Delaunay disks.

Lemma 5 (Combination Lemma) *Let $c \in \mathbb{R}^k$ and $E_c : \mathbb{R}^d \to \mathbb{R}$ such that for any $x \in \mathbb{R}^d$, $c \mapsto E_c(x)$ is an affine function, and let r_c be the region $\{x \in \mathbb{R}^d,\ E_c(x) < 0\}$. Let P be a subset of $\mathbb{R}^k$, if $c \in P$, then $\bigcap_{v\in\mathcal{X}(P)} \mathrm{r}_v \subset \mathrm{r}_c$, where $\mathcal{X}(P)$ denotes the extreme points of the convex hull of P.*

Proof. Consider two points $a, b \in P \subset \mathbb{R}^k$. Let $x \in \mathrm{r}_a \cap \mathrm{r}_b$ and, for $t \in [0,1]$, $c_t = (1-t)a + tb$ be a point on $[ab]$. The function $f : t \mapsto E_{c_t}(x)$ verifies $f(0) = E_a(x) < 0$ and $f(1) = E_b(x) < 0$. Since f is affine, for any $t \in [0,1]$, $E_{c_t}(x) = f(t) = (1-t)f(0) + tf(1) < 0$, so $x \in \mathrm{r}_{c_t}$. Thus $\mathrm{r}_a \cap \mathrm{r}_b \subset \mathrm{r}_{c_t}$ for any c_t on the edge $[ab]$. The extension from an edge $[ab]$ to the convex hull of P follows directly from the its convexity. $\square$

We now show, as an example, that any Delaunay disk contains one of the two half-moons using the Combination lemma:

Corollary 6 *Let p, q two points in the Euclidean plane and D a disk with p and q on its boundary, then $hm_r(p,q) \subset D$ or $hm_\ell(p,q) \subset D$*

Proof. We choose the coordinate system so that p is the origin and $q = (x_q, y_q)$ with $y_q \neq 0$. A disk D with p and q on its boundary can be parameterized by the inequality $E_c(x,y) : x^2 - 2xx_c + y^2 - 2yy_c < 0$ where c verifies $y_c = \frac{x_q^2 - 2x_q x_c + y_q^2}{2y_q}$. Since this is actually the equation of the bisector line of $[pq]$, the centers $c = (x_c, y_c)$ are the actual geometric centers of the disks. That provides a 1-dimensional family of disks parameterized by x_c. In that parameterization, $x_c \mapsto E_c(x,y)$ is an affine function.

Then we can consider the center c_{Gab} of the Gabriel disk and the center c_r at the infinity of the bisector line to the right of $\overrightarrow{pq}$; their associated regions are the Gabriel disk $\mathrm{Gab}(p,q)$ and the half-plane $hp_r(p,q)$ to the right of $\overrightarrow{pq}$. Since the ray $[c_{\mathrm{Gab}}, c_r)$ is convex, we can apply the Combination lemma with $(k,d) = (1,2)$ to ensure that any disk whose center belongs to $[c_{\mathrm{Gab}}, c_r)$ contains $hm_r(p,q)$; indeed $hm_r(p,q) = \mathrm{Gab}(p,q) \cap hp_r(p,q)$. We apply the same reasoning for $hm_\ell(p,q)$ to conclude that if a disk has p and q on its boundary, it contains either $hm_r(p,q)$ or $hm_\ell(p,q)$ depending on the position of its center on the bisector line. $\square$

After the Combination lemma, the second ingredient of our demonstration scheme is the Partition lemma:

Lemma 7 (Partition Lemma) *Let $\mathcal{G}^{\emptyset}_{\mathcal{R}}$ be an empty-region graph with $\mathcal{R}(p,q) = \{r_c; c \in P \subset \mathbb{R}^k\}$ a set of regions parameterized by c. Let $(P_i)_{1\le i\le n}$ be a convex subdivision of P, the parameter space. Let $\mathcal{R}^*_i(p,q) = \{r^*_i(p,q)\}$ be n singletons. If $\forall c \in P_i;\ r^*_i(p,q) \subset r_c$ then $\mathcal{G}^{\emptyset}_{\mathcal{R}}$ is a subgraph of $\cup_{1\le i\le n} \mathcal{G}^{\emptyset}_{\mathcal{R}^*_i}$ and*

$$\deg\left(p, \mathcal{G}^{\emptyset}_{\mathcal{R}}\right) \le \sum_{1\le i\le n} \deg\left(p, \mathcal{G}^{\emptyset}_{\mathcal{R}^*_i}\right).$$

Proof. If pq is an edge of $\mathcal{G}^{\emptyset}_{\mathcal{R}}(X)$, according to Definition 1, there exists $c \in P$ such that $r_c(p,q) \cap X = \emptyset$. Using the convex subdivision, there is some j such that $c \in P_j$ and $r^*_j(p,q) \subset r_c(p,q)$ by the hypothesis in the lemma. Thus $r^*_j(p,q) \cap X = \emptyset$ and pq is also an edge of $\mathcal{G}^{\emptyset}_{\mathcal{R}^*_j}$. $\square$

Using these two lemmas, the general idea of the method we apply to compute an upper bound on the degree of a point in a given empty-region graph of a Poisson point process can be outlined as follows: *(i)* find a good affine parameterization of the regions to be able to apply the Combination lemma, *(ii)* find a good partition of the parameter space to be able to apply the Partition lemma, and *(iii)* analyze the size of the relevant empty-singleton-region graphs.

4 Empty Axis-Aligned Ellipse Graphs

In this section, we analyze empty-region graphs where the regions are axis-aligned ellipses. By "axis-aligned", we mean that their axes of symmetry are parallel to the x and y axes. We then call aspect ratio, the ratio of the lengths of the vertical axis to the horizontal axis of the ellipse.

4.1 Some Features of Axis-Aligned Ellipses

We give some explanations on the expression of ellipses we consider, and some properties that will be used thereafter. In $\mathbb{R}^2$, we consider an axis-aligned ellipse with the origin p on its boundary. We denote the ellipse r since it is seen as a region. Such an ellipse has three degrees of freedom, that can be set by considering a positive number α and a point $c = (x_c, y_c)$, so that r can be defined by the inequality:

$$\mathrm{r} : \alpha^2 x^2 - 2xx_c + y^2 - 2yy_c < 0.$$

In that parameterization, c is the affine parameter of r, and α its aspect ratio. To ensure that the boundary of the ellipse passes through a second point q, the three parameters x_c, y_c and α must satisfy: $\alpha^2 x_q^2 - 2x_q x_c + y_q^2 - 2y_q y_c = 0$. Expressing α in terms of c and q, we define

$$E_c(x,y) := \alpha^2 x_q^2 x^2 - 2xx_c + y^2 - 2yy_c, \qquad (1)$$
$$\text{with } \alpha^2 = \frac{2x_q x_c - y_q^2 + 2y_q y_c}{x_q^2}.$$

The inequality $E_c(x,y) < 0$ is an affine parameterization of $\mathrm{r}_c(p,q)$, the only axis-aligned ellipse passing through p and q with c for parameter. We stress that c is not the usual geometric center of the ellipse.

In some proofs, we bring the expression back to its canonical form namely $\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$, in which the ellipse has aspect ratio $\frac{b}{a}$ and area πab.

Proposition 8 *For a given $q \in \mathbb{R}^2$, the parameters c of the ellipses $\mathrm{r}_c(p,q)$ with same aspect ratio lie on a line perpendicular to (pq).*

Proof. The aspect ratio is given by the coefficient of x^2 in Equation (1). The set of points $c = (x_c, y_c)$ that yields to a constant aspect ratio defines a line parallel to $\mathcal{L} : xx_q + yy_q = 0$, by multiplying by $\frac{x_q^2}{2}$ and omitting the constant terms in the expression of α^2. This line is perpendicular to (pq). □

Proposition 9 *For a given $q \in \mathbb{R}^2$ and for $\alpha \in \mathbb{R}^+$, consider the ellipse $\mathrm{r}_c(p,q)$ parameterized by $c = (\alpha^2 \frac{x_q}{2}, \frac{y_q}{2})$.*

The geometric center of $\mathrm{r}_c(p,q)$ is the midpoint of $[pq]$, and its area is $\frac{\pi}{4}\left(\alpha x_q^2 + \frac{y_q^2}{\alpha}\right)$.

Proof. Transforming the equation $E_c(x,y) < 0$ of $\mathrm{r}_c(p,q)$ we get, its canonical form:

$$\frac{4\alpha^2}{\alpha^2 x_q^2 + y_q^2}(x - \frac{x_q}{2})^2 + \frac{4}{\alpha^2 x_q^2 + y_q^2}(y - \frac{y_q}{2})^2 - 1 < 0.$$

We identify, with that expression, that $\mathrm{r}_c(p,q)$ is the translated copy of an ellipse of center p, and area $\frac{\pi}{4}\left(\alpha x_q^2 + \frac{y_q^2}{\alpha}\right)$ by the vector $\frac{1}{2}\overrightarrow{pq}$. Details can be found in appendix. □

4.2 Unbounded Aspect Ratio: Right-Triangle Graph

In this section, we prove, using our framework, a logarithmic bound for the empty axis-aligned ellipse graph of a Poisson point process in a bounded domain. A similar result was proven for a uniform distribution instead of a Poisson distribution [4].

For two points p and q in $\mathbb{R}^2$, we consider the family $\mathit{Ell}(p,q)$ of all axis-aligned ellipses with p and q on their boundaries. Assuming that p is the origin, we show that the expected degree of p in the associated empty-region graph $\mathcal{G}^{\emptyset}_{\mathit{Ell}}(X)$ is $\Theta(\ln\lambda)$ when X is a Poisson process of intensity λ. In order to identify an upper bound, we consider the graph $\mathcal{G}^{\emptyset}_{\{\Delta_r,\Delta_\ell\}}$ where $\Delta_r(p,q)$ (resp. $\Delta_\ell(p,q)$) denotes the axis-aligned right triangle with hypotenuse $[pq]$ on the right (resp. left) side of $\overrightarrow{pq}$.

Lemma 10 *$\mathcal{G}^{\emptyset}_{\{\Delta_r,\Delta_\ell\}}$ is a super-graph of $\mathcal{G}^{\emptyset}_{\mathit{Ell}}$.*

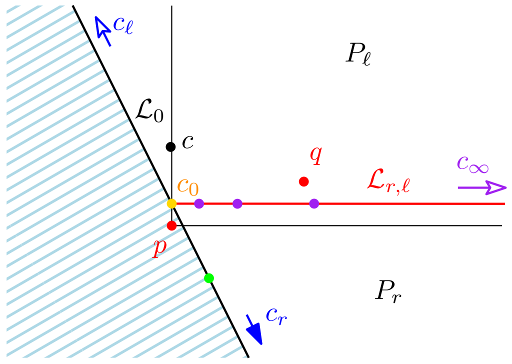


Figure 2: Partition of space of parameters into $\{P_\ell, P_r\}$.

Proof. For each region $\mathrm{r}_c(p,q) \in \mathit{Ell}(p,q)$, we consider the parameterization $\mathrm{r}_c(p,q) : E_c(x,y) < 0$ where E_c is defined in Equation (1).

The space $P \subset \mathbb{R}^2$ where c lives is delimited by the inequality: $2x_q x_c - y_q^2 + 2y_q y_c > 0$ that is the half-plane whose boundary is the line $\mathcal{L}_0$ perpendicular to (pq) passing through $c_0 = (0, \frac{y_q}{2})$ and that does not contain p (if c is not in this half-plane, the equation for α^2 has no positive solutions). With a small abuse of notation, we define two points at infinity at the two extremities of $\mathcal{L}_0$: c_r to the right of $\overrightarrow{pq}$ and c_ℓ to its left. (see Figure 2). On the boundary $\mathcal{L}_0$ of P, elliptic regions degenerate into parabolas. At point c_0 it degenerates to the horizontal strip $\mathrm{r}_{c_0} = \{|y - \frac{y_q}{2}| < \frac{|y_q|}{2}\}$ that can be seen as an ellipse with aspect ratio 0 (see Figure 3). At the limit when going at infinity along $\mathcal{L}_0$ the parabola degenerates in two half-planes: r_{c_r} and r_{c_ℓ} bounded by (pq).

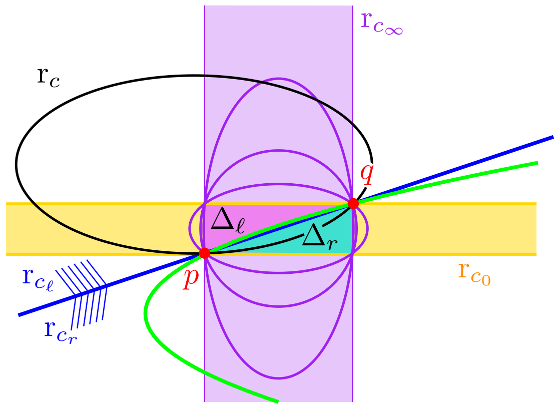


Figure 3: Ellipses corresponding to points in Figure 2. They all contain either Δ_r or Δ_ℓ. Regions whose parameter are on $\mathcal{L}_{r,\ell}$ are in purple, they range from r_{c_0} (in yellow), to $\mathrm{r}c_\infty$. Any region whose parameter is in P_ℓ, like r_c, contains Δ_ℓ.

Then, we consider the ray $\mathcal{L}_{r,\ell} : y = \frac{y_q}{2} \cap P$, starting at c_0, named after the fact that it will distinguish right and left regions. Let c_∞ be the point at infinity on this ray. When the parameter c is equal to c_∞, the ellipse degenerates into the vertical strip $r_{c_\infty} = \{|x - \frac{x_q}{2}| < \frac{|x_q|}{2}\}$, seen as a vertical ellipse with infinite aspect ratio. For c on $\mathcal{L}_{r,\ell}$, ellipses are centered on the midpoint of $[pq]$ and ranges from the horizontal strip r_{c_0} to the vertical strip r_{c_∞}.

By the Combination lemma, if $c \in P_r$ then

$$\Delta_r = \mathrm{r}_{c_0} \cap \mathrm{r}_{c_r} \cap \mathrm{r}_{c_\infty} \subset \mathrm{r}_c$$

and if $c \in P_\ell$ then

$$\Delta_\ell = \mathrm{r}_{c_0} \cap \mathrm{r}_{c_\ell} \cap \mathrm{r}_{c_\infty} \subset \mathrm{r}_c.$$

This ensures, by the Partition lemma, that

$$\mathcal{G}^{\emptyset}_{Ell} \subset \mathcal{G}^{\emptyset}_{\{\Delta_r,\Delta_\ell\}} = \mathcal{G}^{\emptyset}_{\{\Delta_r\}}(X) \cup \mathcal{G}^{\emptyset}_{\{\Delta_\ell\}}(X). \qquad \square$$

Now, we bound from above the expected degree of p in $\mathcal{G}^{\emptyset}_{\{\Delta_r\}}(X)$ or $\mathcal{G}^{\emptyset}_{\{\Delta_\ell\}}(X)$ when X is a Poisson point process with intensity λ.

The area of both triangles Δ_r and Δ_ℓ is $\frac{|x_q y_q|}{2}$. Unfortunately, for any positive λ, $\int_{\mathbb{R}}\int_{\mathbb{R}} e^{-\lambda|xy|}dydx$ does not converge. In that case, we assume that X is distributed in a rectangle $\mathcal{D} = [-L, L] \times [-l, l]$ for positive L and l.

Lemma 11 *Let X be a Poisson point process with intensity λ in $\mathcal{D} = [-L, L] \times [-l, l]$. The expected degree $\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{\Delta_r\}}\right)\right]$ of the origin p in $\mathcal{G}^{\emptyset}_{\{\Delta_r\}}(X)$ is $\Theta(\ln\lambda + \ln L + \ln l)$.*

Proof. Let t be a positive number such that $tLl > 1$, we start by bounding from above the following integral:

$$\begin{aligned}
I_{L,l}(t) &= \int_0^L \int_0^l e^{-txy}\mathrm{d}y\mathrm{d}x \\
&= \int_0^L \int_0^{lx} \frac{e^{-tu}}{x}\mathrm{d}u\mathrm{d}x && \text{with } u = xy \\
&= \int_0^L \frac{1 - e^{-tlx}}{tx}\mathrm{d}x \\
&= \frac{1}{t}\int_0^{tLl} \frac{1 - e^{-v}}{v}\mathrm{d}v && \text{with } v = tlx \\
&= \frac{1}{t}\Big(\int_0^1 \frac{1 - e^{-v}}{v}\mathrm{d}v + \int_1^{tLl} \frac{1 - e^{-v}}{v}\mathrm{d}v\Big) \\
&\le \frac{1}{t}\left(\int_0^1 \mathrm{d}v + \int_1^{tLl} \frac{1}{v}\mathrm{d}v\right) && \text{since } 1-e^{-v} \le \min(1, v) \\
&= \frac{1}{t}\left(1 + \ln(tLl)\right).
\end{aligned}$$

And bounding from below:

$$\begin{aligned}
I_{L,l}(t) &= \frac{1}{t}\int_0^{tLl} \frac{1 - e^{-v}}{v}dv \\
&\ge \frac{1}{t}\int_0^{tLl} \frac{1}{v+1}dv && \text{since } \frac{1 - e^{-v}}{v} \ge \frac{1}{v+1} \text{ if } v \ge 0 \\
&= \frac{1}{t}\left(\ln(tLl + 1)\right) && > \frac{\ln(tLl)}{t}.
\end{aligned}$$

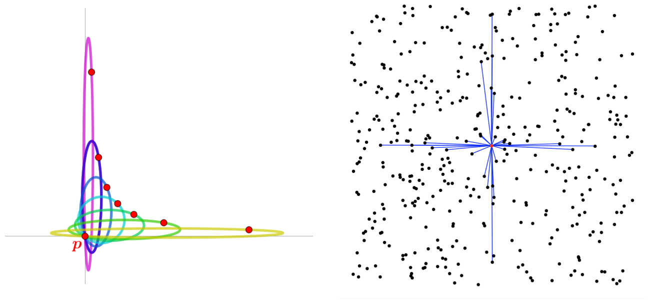


Figure 4: Left: Some *xy-ell* ellipses whose color depends of the aspect ratio. Right: An instance of $\mathcal{G}^{\emptyset}_{\{xy\text{-}ell\}}$ where p is the central (red) point. For any point q, $xy\text{-}ell(p, q)$ has the smallest area among ellipses passing through p and q so that a far point keeps chance to be a neighbor of p as long as it is close to an axis.

Then we can give an upper bound on the expected degree:

$$\begin{aligned}
\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{\Delta_r\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[\Delta_r(q)\cap X=\emptyset]}\right] \\
&= \int_{\mathcal{D}} \lambda(\mathbb{P}\left[\Delta_r(q) \cap X = \emptyset\right])\mathrm{d}q \\
&= \int_{-L}^{L}\int_{-l}^{l} \lambda e^{-\lambda|\Delta_r(q)|}\mathrm{d}y\mathrm{d}x \\
&= 4\lambda\int_0^L\int_0^l e^{-\lambda\frac{xy}{2}}\mathrm{d}y\mathrm{d}x \\
&= 4\lambda I_{L,l}(\tfrac{\lambda}{2}) \quad = \Theta(\ln(\lambda Ll)). \qquad \square
\end{aligned}$$

We can compare with the Delaunay triangulation, the Gabriel and the half-moon graphs. In those cases, the empty region had an area quadratic in the distance pq and the expected degree was constant. In this new case, the "xy" area provides a logarithmic degree.

Lemmas 10 and 11 give an upper bound on the degree of a point in the empty-axis aligned-ellipse graph. To get a lower bound, we will exhibit a subgraph of $\mathcal{G}^{\emptyset}_{Ell}$. In order to get a tight bound, the area of the chosen region must be $\Theta(x_q y_q)$.

To find such an ellipse, that we name *xy-ell(p,q)*, we use Proposition 9, with $\alpha = \frac{y_q}{x_q}$. We define *xy-ell(p,q)* to be the ellipse parameterized by $c_{xy} = \left(\frac{y_q^2}{2x_q}, \frac{y_q}{2}\right)$ (see Figure 4), then according to Proposition 9, the area of this ellipse is $\frac{\pi}{2}x_q y_q$.

Lemma 12 *$\mathcal{G}^{\emptyset}_{\{xy\text{-}ell\}}$ is a subgraph of $\mathcal{G}^{\emptyset}_{Ell}$.*

Proof. Straightforward because $xy\text{-}ell \in Ell$. $\square$

Lemma 13 *Let X be a Poisson point process with intensity λ in $\mathcal{D} = [-L, L] \times [-l, l]$. The expected degree*

$\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{xy\text{-}ell\}}\right)\right]$ *of the origin p in $\mathcal{G}^{\emptyset}_{\{xy\text{-}ell\}}(X)$ is* $\Theta(\ln\lambda + \ln L + \ln l)$.

Proof. As said above, for any $q \in \mathbb{R}^2$, the area of $xy\text{-}ell(p,q)$ is $\frac{\pi}{2}x_q y_q$.

Then we can express the expected degree:

$$\begin{aligned}\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{xy\text{-}ell\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{\{xy\text{-}ell(p,q)\cap X=\emptyset\}}\right] \\ &= \int_{\mathcal{D}} \lambda e^{-\lambda|xy\text{-}ell(p,q)|}\mathrm{d}p \\ &= 4\lambda \int_0^L \int_0^l e^{-\lambda\pi\frac{xy}{2}}\mathrm{d}y\mathrm{d}x \\ &= \Theta(\ln(\lambda L l)).\end{aligned}$$ □

The above lemmas allow to conclude:

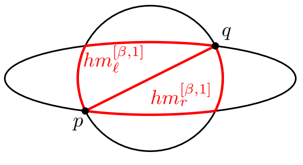
Theorem 14 *Let X be a Poisson point process with intensity λ in $\mathcal{D} = [-L, L] \times [-l, l]$ and p the origin:*

$$\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{Ell}\right)\right] = \Theta(\ln\lambda + \ln L + \ln l).$$

4.3 Bounded Aspect Ratio: Rhombus Graph

In the previous part, we proved that when the aspect ratio is not bounded, neither is the expected degree. One can wonder what happens when the aspect ratio ranges between two finite numbers. For two points p and q in $\mathbb{R}^2$ and a number $\beta \in (0, 1)$, we consider the family $Ell^{[\beta,1]}(p, q)$ of horizontal elliptic regions with p and q on their boundary and whose aspect ratio ranges between β and 1. An important fact to be considered is that, when the aspect ratio is not bounded, a point q far from p could be a neighbor of p as long as it is close enough to the axis, since in that case, ellipses passing through p and q may have a small area, and that leads to a logarithmic bound. When the aspect ratio is bounded, all ellipses preserve an area $\Omega(x_q^2+y_q^2)$, so that we expect a constant bound on the expected degree. In this section we will prove that it is actually the case and detail how this constant depends on β.

In order to apply the same method as above, we search for simple geometrical regions that fit inside the whole family $Ell^{[\beta,1]}(p, q)$. A good choice is the following: as before, we consider the intersection of ellipses that are centered on the midpoint of $[pq]$, and we cut the intersection along (pq). The remaining regions $hm_r^{[\beta,1]}(p, q)$ and $hm_\ell^{[\beta,1]}(p, q)$ look like two axis-aligned right triangles with rounded sides for almost all q (see above figure).



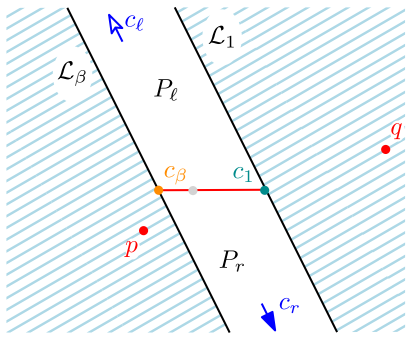


Figure 5: Partition of space of parameters into $\{P_\ell, P_r\}$ in the bounded aspect ratio case.

Lemma 15 *The graph $\mathcal{G}^{\emptyset}_{\{hm_r^{[\beta,1]},\ hm_\ell^{[\beta,1]}\}}$ is a supergraph of $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}}$.*

Proof. The proof is very similar to the one of Lemma 10, so we just spell out the important points. For each $\mathrm{r}_c(p, q) \in Ell^{[\beta,1]}(p, q)$, we consider the parameterization $\mathrm{r}_c(p, q) : E_c(x, y) < 0$ defined in Equation (1) with $\beta \le \alpha \le 1$.

The space $P \subset \mathbb{R}^2$ where c lives is delimited by the inequality: $\beta^2 \le \frac{2x_qx_c - y_q^2 + 2y_qy_c}{x_q^2} \le 1$ that is the strip perpendicular to (pq) whose boundary are the lines $\mathcal{L}_\beta$ and $\mathcal{L}_1$, where $\mathcal{L}_\alpha = \{(x, y), \alpha^2x_q^2 = 2x_qx - y_q^2 + 2y_qy\}$. We consider the segment defined by $y = \frac{y_q}{2}$ inside P and its extremities c_β on $\mathcal{L}_\beta$ and c_1 on $\mathcal{L}_1$. We partition P into P_r and P_ℓ where P_r is the part of P on the right of $[c_\beta, c_1)$, and P_ℓ the part on its left (see Figure 5).

c_β and c_1 have for regions the ellipses r_{c_β} and r_{c_1} with respectively β and 1 for aspect ratio. Furthermore, any parameter c_r in P at infinity on the right of $\overrightarrow{pq}$ has its region that degenerates into the half-plane bounded by (pq) on the right side of $\overrightarrow{pq}$ (and the same holds for c_ℓ and the left side).

By the Combination lemma, if $c \in P_r$ then $hm_r^{[\beta,1]} := \mathrm{r}_{c_1} \cap \mathrm{r}_{c_\beta} \cap \mathrm{r}_{c_r} \subset \mathrm{r}_c$ and if $c \in P_\ell$ then $hm_\ell^{[\beta,1]} := \mathrm{r}_{c_1} \cap \mathrm{r}_{c_\beta} \cap \mathrm{r}_{c_\ell} \subset \mathrm{r}_c$ (see Figure 6). By the Partition lemma, an edge of $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}}$ is an edge of $\mathcal{G}^{\emptyset}_{hm_r^{[\beta,1]}}$ or $\mathcal{G}^{\emptyset}_{hm_\ell^{[\beta,1]}}$. □

The problem now is that it may be complicated to compute an integral involving the area of $hm_r^{[\beta,1]}(p, q)$ or $hm_\ell^{[\beta,1]}(p, q)$. To solve this issue, we consider a strictly smaller region. We could have used the axis-aligned right triangles Δ_r and Δ_ℓ but their areas do not respect the order of magnitude (as illustrated by Figure 7). A more suitable region is what we call the half-rhombus. We define the rhombus $Rh^\beta(p, q)$ as the one whose vertices are the horizontal extreme points of $\mathrm{r}_{c_1}(p, q)$ and the vertical extreme points of $\mathrm{r}_{c_\beta}(p, q)$. Then we separate it into two halves $Rh_r^\beta(p, q)$ and $Rh_\ell^\beta(p, q)$, delimited by (pq). By convexity, it is clear that $Rh_r^\beta(p, q) \subset$

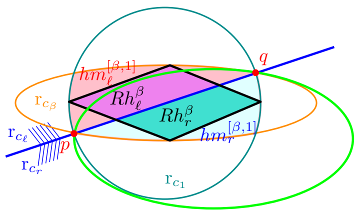


Figure 6: The green ellipse with parameter to the right of $\overrightarrow{pq}$ contains Rh_r^β.

$hm_r^{[\beta,1]}(p,q)$ and $Rh_\ell^\beta(p,q) \subset hm_\ell^{[\beta,1]}(p,q)$ (see Figure 6). Finally, we can say that $\mathcal{G}^\emptyset_{\{Rh_r^\beta\}}$ is a super-graph of $\mathcal{G}^\emptyset_{\{hm_r^{[\beta,1]}\}}$ and that $\mathcal{G}^\emptyset_{\{Rh_\ell^\beta\}}$ is a super-graph of $\mathcal{G}^\emptyset_{\{hm_\ell^{[\beta,1]}\}}$.

Before proceeding to the computation of the expected degree, we introduce a lemma that provides properties on the involved integral.

Lemma 16 *Let $t>0$, $\beta \in (0,1)$ and*

$$I_\beta(t) = \int_{\mathbb{R}}\int_{\mathbb{R}} e^{-t\sqrt{(x^2+y^2)(\beta^2x^2+y^2)}}\mathrm{d}y\mathrm{d}x,$$

$$I_\beta(t) = \frac{1}{t}I_\beta(1) \le \frac{\pi}{t}\left(1+\ln(\tfrac{1}{\beta})\right).$$

Proof. The computations are in appendix. □

Lemma 17 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$, and $\beta \in (0,1)$.*

$$\mathbb{E}\left[\deg\left(p, \mathcal{G}^\emptyset_{\{Rh_r^\beta\}}\right)\right] = O(\ln\frac{1}{\beta}).$$

Proof. We first compute the area of the rhombus $Rh^\beta(p,q)$. We identify its width and height as being respectively $\sqrt{x_q^2+y_q^2}$ and $\sqrt{\beta^2x_q^2+y_q^2}$ so that the value of its area is given by $\frac{1}{2}\sqrt{\left(x_q^2+y_q^2\right)\left(\beta^2x_q^2+y_q^2\right)}$.

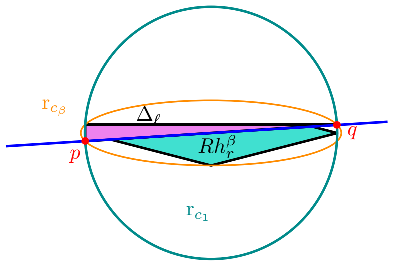
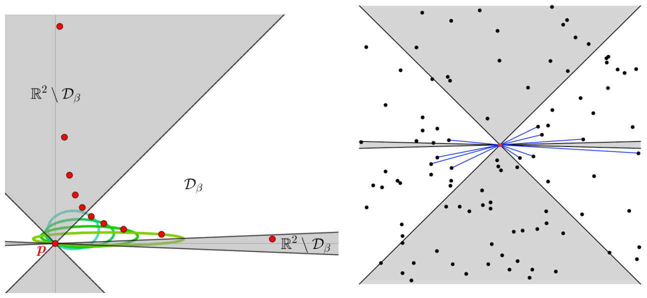


Figure 7: The area of Δ_r and Rh_r^β can have different order of magnitude.



$\mathbb{R}^2 \setminus \mathcal{D}_\beta$
$\mathcal{D}_\beta$
p
$\mathbb{R}^2 \setminus \mathcal{D}_\beta$

Figure 8: Some β-*ell* ellipses for points in $\mathcal{D}_\beta$ and an instance of $\mathcal{G}^\emptyset_{\{\beta\text{-}ell\}}$ where p is the red point. A far point reduces strongly its probability to be a neighbor of p because it cannot anymore be close to the axes.

Then we can compute the expected degree of p in $\mathcal{G}^\emptyset_{\{Rh_r^\beta\}}(X)$:

$$\begin{aligned}
\mathbb{E}\left[\deg\left(p,\mathcal{G}^\emptyset_{\{Rh_r^\beta\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[Rh_r^\beta(p,q)\cap X=\emptyset]}\right] \\
&= \int_{\mathbb{R}^2} \lambda\,\mathbb{P}\left[Rh_r^\beta(p,q)\cap X=\emptyset\right] \\
&= \int_{\mathbb{R}^2} \lambda e^{-\lambda\frac{1}{2}|Rh^\beta(p,q)|}\mathrm{d}q \\
&= \int_{\mathbb{R}}\int_{\mathbb{R}} \lambda e^{-\frac{\lambda}{4}\sqrt{\left(x_q^2+y_q^2\right)\left(\beta^2x_q^2+y_q^2\right)}}\mathrm{d}y\mathrm{d}x \\
&= \lambda I_\beta\left(\tfrac{\lambda}{4}\right) \\
&\le 4\pi\left(1-\ln(\beta)\right) \quad \text{by Lemma 16.}
\end{aligned}$$ □

We obtain a tight lower bound, when β goes to 0, by identifying, for each q, a particular region, named β-*ell*(p,q), such that β-*ell*(p,q) is or contains an element of $Ell^{[\beta,1]}(p,q)$. To achieve this, we partition the plane into two parts (see Figure 8):

1. if $q \in \mathcal{D}_\beta := \{(x,y), \beta|x| < |y| < |x|\}$, then, as in Lemma 12, we define β-*ell*(p,q) = *xy-ell*(p,q),
2. otherwise β-*ell*$(p,q) = \mathbb{R}^2$, that is another way to say that q is not a neighbor of p.

Lemma 18 *$\mathcal{G}^\emptyset_{\{\beta\text{-}ell\}}$ is a subgraph of $\mathcal{G}^\emptyset_{Ell^{[\beta,1]}}$.*

Proof. If $q \in \mathcal{D}_\beta$, we have to prove that β-*ell*(p,q), *i.e.* *xy-ell*(p,q), is in $Ell^{[\beta,1]}$. This is true because the aspect ratio of if *xy-ell*(p,q) is $|\frac{y_q}{x_q}|$, and verifies $\beta < |\frac{y_q}{x_q}| < 1$ if $\beta|x_q| < |y_q| < |x_q|$.

Otherwise, it is clear that β-*ell*(p,q), *i.e.* $\mathbb{R}^2$, is larger than any other ellipse from $Ell^{[\beta,1]}$. □

Lemma 19 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$. The expected degree*

$\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{\beta\text{-}ell\}}\right)\right]$ *of the origin* p *in* $\mathcal{G}^{\emptyset}_{\{\beta\text{-}ell\}}(X)$ *is* $\Omega(\ln \frac{1}{\beta})$.

Proof. β-$ell(p,q)$ is actually chosen to simplify the computation. Recall that $\mathcal{D}_\beta$ is the domain $\{(x,y), \beta|x| < |y| < |x|\}$;

$$\begin{aligned}
\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{\beta\text{-}ell\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[\beta\text{-}ell(p,q)\cap X=\emptyset]}\right] \\
&= \int_{\mathbb{R}^2} \lambda\, \mathbb{P}\left[\beta\text{-}ell(p,q)\cap X=\emptyset\right] \mathrm{d}p \\
&= \int_{\mathcal{D}_\beta} \lambda\, \mathbb{P}\left[xy\text{-}ell(p,q)\cap X=\emptyset\right] \mathrm{d}p + \int_{\mathbb{R}^2\setminus\mathcal{D}_\beta} 0\ \mathrm{d}p \\
&= \int_{\mathcal{D}_\beta} \lambda e^{-\lambda|xy\text{-}ell(p,q)|} \mathrm{d}p \\
&= 4\int_0^\infty \int_{\beta x}^{x} \lambda e^{-\lambda\pi\frac{xy}{2}} \mathrm{d}y\mathrm{d}x \\
&= 4\int_{\tan^{-1}(\beta)}^{\frac{\pi}{4}} \int_0^\infty \lambda\rho e^{-\lambda\pi\frac{\rho^2\cos(\theta)\sin(\theta)}{2}} \mathrm{d}\rho\mathrm{d}\theta \\
&= 4\int_{\tan^{-1}(\beta)}^{\frac{\pi}{4}} \lambda\frac{1}{\lambda\pi\cos(\theta)\sin(\theta)} \mathrm{d}\theta \\
&= \frac{4}{\pi}\left(\ln(\tan(\tfrac{\pi}{4})) - \ln(\beta)\right) \quad \text{since } \frac{\mathrm{d}}{\mathrm{d}\theta}\ln(\tan(\theta)) = \frac{1}{\cos(\theta)\sin(\theta)} \\
&= \frac{4}{\pi}\ln(\tfrac{1}{\beta}).
\end{aligned}$$

□

We can finally conclude using Lemmas 15 to 19:

Theorem 20 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$. The expected degree* $\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}}\right)\right]$ *of the origin p in* $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}}(X\cup\{p\})$ *is* $\Theta\left(\ln(\frac{1}{\beta})\right)$.

If $\beta > 1$ we have by symmetry,

$$\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{Ell^{[1,\beta]}}\right)\right] = \mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{Ell^{[\frac{1}{\beta},1]}}\right)\right] = \Theta(\ln\beta).$$

4.4 Non-Axis-Aligned Ellipses

We turn our interest to the case of non-axis-aligned ellipses. We consider the graph in which two points p and q are neighbors if there exists an empty ellipse passing through p and q whose aspect ratio is between β and 1, for $\beta\in[0,1]$. For two points p and q, we define the family $Ell^{[\beta,1]}_*(p,q)$ of all ellipses with such aspect ratios passing through p and q, and $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}_*}$, the corresponding empty region graph.

The case where $\beta = 1$ corresponds to the Delaunay triangulation, and the case where $\beta = 0$ corresponds to the complete graph, since we can consider that a segment between two points is an ellipse with aspect ratio 0 and random points are in general position. Thus we assume that $\beta\in(0,1)$.

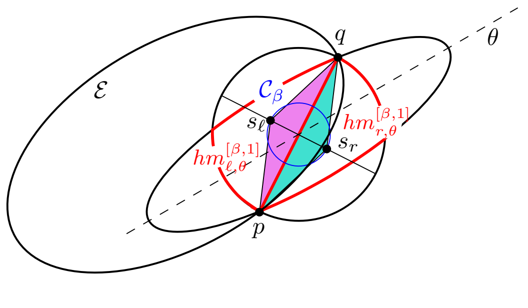


Figure 9: Non-axis-aligned ellipses.

Consider two points, p at the origin and q, and an ellipse $\mathcal{E}$ passing through p and q. Since $\mathcal{E}$ is not anymore axis-aligned but has its great axis in some direction θ, we can consider the regions $hm^{[\beta,1]}_{r,\theta}$ and $hm^{[\beta,1]}_{\ell,\theta}$ as in the previous sections but parameterized by direction θ. Clearly the circle $\mathcal{C}_\beta$ centered at the midpoint of pq and of diameter $\beta|pq|$ is inside $hm^{[\beta,1]}_{r,\theta}\cup hm^{[\beta,1]}_{\ell,\theta}$ (see Figure 9). Consider the isosceles triangles pqs_r and pqs_ℓ such that $s_\ell, s_r\in\mathcal{C}_\beta$ with s_r on the right of $\overrightarrow{pq}$ and s_ℓ on its left. Then $pqs_r\subset hm^{[\beta,1]}_{r,\theta}$ and $pqs_\ell\subset hm^{[\beta,1]}_{\ell,\theta}$.

Since this is true for any ellipse, we can assume that any ellipse whose aspect ratio is between β and 1 and passing through p and q contains either pqs_r or pqs_ℓ. Notice that these triangles are independent of the direction θ. So we can apply the Partition lemma to yield that $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}_*}$ is a subgraph of $\mathcal{G}^{\emptyset}_{\{pqs_r,pqs_\ell\}}$.

Now we consider a Poisson point process X of intensity λ, and we compute an upper bound on the expected degree of p in $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}_*}(X\cup\{p\})$.

$$\begin{aligned}
&\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{pqs_r,pqs_\ell\}}\right)\right] \\
&\le 2\,\mathbb{E}\left[\sum_{q\in X}\mathbb{1}_{[pqs_r\cap X=\emptyset]}\right] \\
&= 2\int_{\mathbb{R}^2}\lambda\mathbb{P}\left[pqs_r\cap X=\emptyset\right]\mathrm{d}q \\
&= 2\int_{\mathbb{R}^2}\lambda e^{-\lambda|pqs_r|}\mathrm{d}q \\
&= 2\int_0^{2\pi}\int_0^\infty \lambda e^{-\frac{\lambda}{8}\beta\rho^2}\rho\,\mathrm{d}\rho\mathrm{d}\theta \quad = \frac{16\pi}{\beta}.
\end{aligned}$$

On the other hand, among the ellipses passing through p and q, we can choose the ellipse $\mathcal{E}_{\beta*}$ whose great axis is $[p,q]$ and has aspect ratio β to obtain a subgraph of $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}_*}$.

The expected degree of p in this graph is

$$\begin{aligned}\mathbb{E}\left[\deg\left(p, \mathcal{G}^{\emptyset}_{\{\mathcal{E}_{\beta *}\}}\right)\right] &= \mathbb{E}\left[\sum_{q\in X} \mathbb{1}_{[\mathcal{E}_{\beta *}\cap X=\emptyset]}\right] \\ &= \int_{\mathbb{R}^2} \lambda\,\mathbb{P}\left[\mathcal{E}_{\beta *}\cap X=\emptyset\right] \mathrm{d}q \\ &= \int_{\mathbb{R}^2} \lambda e^{-\lambda|\mathcal{E}_{\beta *}|}\mathrm{d}q \\ &= \int_0^{2\pi}\int_0^{\infty} \lambda e^{-\lambda\frac{\pi}{4}\beta\rho^2}\rho\,\mathrm{d}\rho\mathrm{d}\theta \quad = \frac{4}{\beta}\end{aligned}$$

We deduce the following theorem:

Theorem 21 *Let X be a Poisson point process in $\mathbb{R}^2$. The expected degree of the origin p in $\mathcal{G}^{\emptyset}_{Ell_*^{[\beta,1]}}(X\cup\{p\})$ is $\Theta\left(\frac{1}{\beta}\right)$.*

5 Probability of Existence of Far Neighbors

At some point, for a given graph $\mathcal{G}$ and a positive number t, we may be interested in computing the probability for p to have a neighbor in $\mathcal{G}$ at a distance greater than t.

As before, for illustration on a simple case, we start by the Delaunay triangulation:

Lemma 22 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$, p a point of $\mathbb{R}^2$, and t a positive number. The probability that p has some Delaunay neighbor at a distance greater than t is smaller than $8e^{-\lambda\frac{\sqrt{2}}{8}t^2}$.*

Proof. If q is a Delaunay neighbor of p, Let σ be an empty disk whose boundary passes through p and q. If q is at distance greater than t from p, then the diameter of σ is obviously also greater than t, so its homothet σ' toward p that has exactly diameter t is included in σ and by consequence empty.

Consider the triangle with vertices p, $(\frac{\sqrt{2}}{2}t, 0)$, and $(\frac{1}{2}t, \frac{1}{2}t)$ and its seven adjacent copies around p (see Figure 10). We name them τ_i for $i \in \{1,\ldots,8\}$. Their area is $|\tau_1| = \frac{\sqrt{2}}{8}t^2$.

One can notice that, at least one triangle is included in σ': the one whose angular sector from p contains the center of σ'.

So we get:

$$\begin{aligned}&\mathbb{P}\left[\exists q\in X;\ [pq]\in \mathrm{Del}(X\cup\{p\}) \mid |pq|>t\right] \\ \leq\ & \mathbb{P}\left[\exists i\in[1,\ldots,8], \tau_i\cap X=\emptyset\right] \\ \leq\ & \sum_{i=1\ldots,8} \mathbb{P}\left[\tau_i\cap X=\emptyset\right] \\ =\ & 8\,\mathbb{P}\left[\tau_1\cap X=\emptyset\right] \quad = \quad 8e^{-\lambda\frac{\sqrt{2}}{8}t^2}. \qquad \square\end{aligned}$$

We establish in the next lemma a similar bound for the empty axis-aligned ellipse graph with bounded aspect ratio in $[\beta, 1]$. We are mainly interested in the behavior of the probability when β is small, thus we assume $\beta < \frac{1}{2}$.

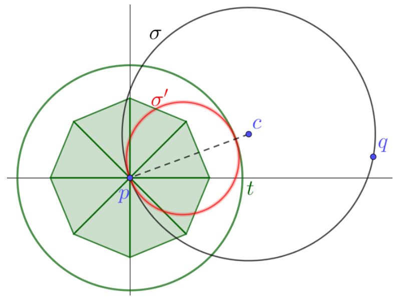


Figure 10: If $|pq| > t$, any disk passing through p and q contains one of the 8 triangles.

Lemma 23 *Let X be a Poisson point process with intensity λ in $\mathbb{R}^2$, p a point of $\mathbb{R}^2$, t and β two positive numbers with $\beta < \frac{1}{2}$. The probability that p has some neighbor in $\mathcal{G}^{\emptyset}_{Ell^{[\beta,1]}}(X)$ at a distance greater than t is smaller than $4\left(e^{-\lambda\frac{\sqrt{2}}{16}\beta t^2} + e^{-\lambda\frac{\sqrt{2}}{16}\beta^{\frac{3}{2}}t^2}\right)$.*

Proof. The proof idea is similar to the previous one, except that we apply a homothety on the empty ellipse σ until its image σ' fits inside the axis-aligned square inscribed in the circle of radius t (see Figure 11).

We consider eight triangles $(tr_i)_{1\leq i\leq 8}$, that have the property that for any ellipse σ, σ' contains one of them.

To this aim we define the four points

$$\begin{aligned} v_1 &= (\tfrac{1}{2}t, 0), & v_2 &= (\tfrac{\sqrt{2}}{4}t, \tfrac{\sqrt{2}}{4}\beta t), \\ v_3 &= (\tfrac{\sqrt{2\beta}}{4}t, \tfrac{\sqrt{2}}{4}\beta t), & v_4 &= (0, \tfrac{1}{2}\beta t). \end{aligned}$$

The triangles tr_1 and tr_2 are respectively pv_1v_2 and pv_3v_4. Their respective areas are $\frac{\sqrt{2}}{16}\beta t^2$ and $\frac{\sqrt{2}}{16}\beta^{\frac{3}{2}}t^2$. We will show that any ellipse tangent to the square in the upper right quadrant contains tr_1 or tr_2. We complete the set of triangles by their symmetrical copies with respect to the x-axis, to the y-axis and to the point p, and name them according to the trigonometric order

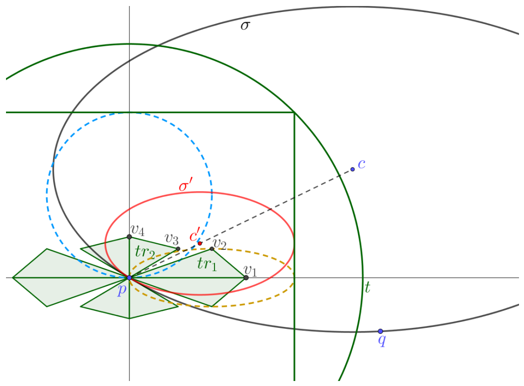


Figure 11: If $|pq| > t$, any ellipse passing through p and q contains one of the 8 triangles.

from tr_1 to tr_8 to cover the ellipses tangent to other parts of the square.

Without loss of generality, we assume that the center c' of σ' is in the upper right quadrant. In such a case, the right most point of σ' has abcissa $\frac{\sqrt{2}}{2}t$, its left most point has negative abcissa, and its center verifies $0 \leq x_{c'} \leq \frac{\sqrt{2}}{4}t$.

As long as $x_{c'} \geq \frac{1}{4}t$, using the symmetry of the ellipse with respect to its vertical axis, v_1 is between p and the symmetric of p, and thus is inside σ_0. We prove that such ellipses, with $x_{c'} \geq \frac{1}{4}t$, also contain v_2. Actually v_2 is chosen as the highest point of the thinnest ellipse of center $c' = (\frac{\sqrt{2}}{4}t, 0)$ (in yellow on Figure 11), with aspect ratio β. Since the abscissa of v_2 is between p and v_1, moving the center c' upward or to the left or increasing β imply that v_2 remains inside σ'. So as long as $x_{c'} \geq \frac{1}{4}t$, the triangle tr_1 is inside σ'.

Suppose now that $x_{c'} \leq \frac{1}{4}t$. The ellipse σ', if its aspect ratio is α, has equation:

$$\alpha^2 x^2 - 2\alpha^2 x x_{c'} + y^2 - 2y y_{c'} \leq 0.$$

For a fixed α, the lowest possible center is reached when $x_{c'} = \frac{1}{4}t$ and since σ' is tangent to the right side of the square at $(\frac{\sqrt{2}}{2}t, y_{c'})$, by substitution we have:

$$\tfrac{1}{2}\alpha^2 t^2 - \sqrt{2}\alpha^2 t \tfrac{1}{4} t - y_{c'}^2 = 0.$$

Thus $y_{c'}$ is minimized for $\alpha = \beta$, and so:

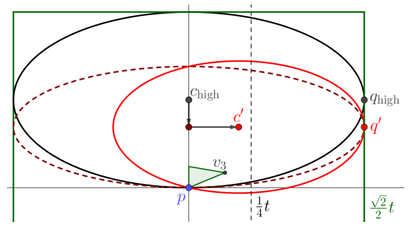
$$y_{c'} = \frac{1}{2}\sqrt{2 - \sqrt{2}}\,\beta t \simeq 0.383\beta t.$$

We can deduce, by symmetry with respect to the horizontal axis of σ', that all those ellipses contain the segment between p and $(0, \sqrt{2-\sqrt{2}}\,\beta t)$, including v_4.

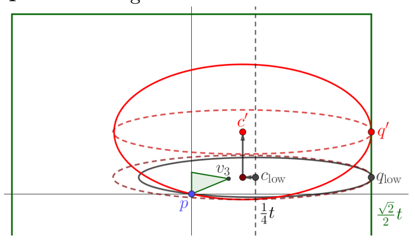
To prove that $v_3 \in \sigma'$, we make a distinction between the side of tangency of σ'. We call contact point of an ellipse, the point of the ellipse in which it is tangent to the square, for σ' we name it q'. Suppose first that σ' is tangent to the right side of the square. We consider the two extreme ellipses σ_{high} and σ_{low}, with highest and lowest contact points q_{high} and q_{low}, at respectively $\frac{\sqrt{2}}{4}t$ and $\frac{1}{2}\sqrt{2-\sqrt{2}}\beta t$ for ordinate. They both contain v_3:

$$\begin{aligned}
&v_3 \in \sigma_{\text{low}} :\\
&\beta^2\left(\tfrac{\sqrt{2\beta}}{4}t\right)^2 - \beta^2\left(\tfrac{\sqrt{2\beta}}{4}t\right)\tfrac{t}{4} + \left(\tfrac{\sqrt{2}}{4}\beta t\right)^2 - 2\left(\tfrac{\sqrt{2}}{4}\beta t\right)\tfrac{\sqrt{2-\sqrt{2}}}{2}\beta t\\
&= \beta^2 t^2\left(\tfrac{\beta}{8} - \tfrac{\sqrt{2\beta}}{16} + \tfrac{1}{8} - \tfrac{\sqrt{4-2\sqrt{2}}}{8}\right) \leq 0 \qquad \text{for } \beta \leq 0.6\\
&v_3 \in \sigma_{\text{high}} :\\
&\left(\tfrac{\sqrt{2\beta}}{4}t\right)^2 - 0 + \left(\tfrac{\sqrt{2}}{4}\beta t\right)^2 - 2\left(\tfrac{\sqrt{2}}{4}\beta t\right)\tfrac{\sqrt{2}}{4}t\\
&= \beta t^2\left(\tfrac{1}{8} + \tfrac{\beta}{8} - \tfrac{2}{8}\right) \leq 0 \qquad \text{since } \beta \leq 1
\end{aligned}$$

We call bottom part of the ellipse, the counterclockwise arc from p to the contact point, and top part the following arc from the contact point to the intersection with the y-axis.



We show that the bottom part of σ' is below the bottom part of σ_{high}. We apply a vertical affine transformation that flattens σ_{high} until its contact point becomes q'. The new ellipse clearly has its bottom part lower since the transformation lowered every point. Then we shift horizontally the center into c', maintaining the points p and q'. Since that makes the aspect ratio grow, here again we lowered the bottom part. So the bottom part of σ' is below the bottom part of σ_{high}.



On the other hand we apply a homothetic transformation on σ_{low} centered on its contact point such that the length of the horizontal axis is the same as the length as σ', followed by a vertical translation until the contact point coincides with q', finally completed by a vertical affine transformation that makes it reach the correct aspect ratio, that is greater. All these transformations make the upper part of the ellipse go upward. We deduce that any ellipse tangent to the right side of the square and whose center has abscissa smaller than $\frac{1}{4}t$ contains tr_2.

Then we can go to ellipses tangent to the top side of the square. The proof is quite identical so we do not develop it but keep in mind that the important point is that v_3 belongs to circle centered at $(0, \frac{\sqrt{2}}{4}t)$ because v_3 lies on the parabola $y = \frac{2\sqrt{2}}{t}x^2$, that is above the circle for $y < \frac{\sqrt{2}}{4}t$.

Above arguments proved that any ellipse whose center is in the upper right corner of the triangle contains either (pv_1v_2) or (pv_3v_4). By extension, we deduce that any ellipse contains at least one of the 8 triangles tr_i.

So we get:

$$\begin{aligned}
&\mathbb{P}\left[[pq] \in \mathcal{G}^{\emptyset}_{Ell[\beta,1]}(X) \mid |pq| > t\right]\\
&\quad\leq \quad \mathbb{P}\left[\exists i \in [1,\ldots,8], tr_i \cap X = \emptyset\right]\\
&\quad= \quad 4\left(\mathbb{P}\left[tr_1 \cap X = \emptyset\right] + \mathbb{P}\left[tr_2 \cap X = \emptyset\right]\right)\\
&\quad= \quad 4\left(e^{-\lambda\frac{\sqrt{2}}{16}\beta t^2} + e^{-\lambda\frac{\sqrt{2}}{16}\beta^{\frac{3}{2}}t^2}\right)\\
&\quad= \quad \Theta(e^{-\lambda\sqrt{2}\beta^{\frac{3}{2}}\left(\frac{t}{4}\right)^2}) \qquad \square
\end{aligned}$$

Acknowledgements

The authors thank Sylvain Lazard for its relevant suggestions.

References

[1] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical models and image processing*, 60(2):125–135, 1998. `doi:10.1006/gmip.1998.0465`.

[2] J. Cardinal, S. Collette, and S. Langerman. Empty region graphs. *Computational geometry*, 42(3):183–195, 2009. `doi:10.1016/j.comgeo.2008.09.003`.

[3] B. Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[4] O. Devillers, J. Erickson, and X. Goaoc. Empty-ellipse graphs. In *19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 1249–1256, San Francisco, United States, 2008. URL: `https://hal.inria.fr/inria-00176204`.

[5] L. Devroye, C. Lemaire, and J.-M. Moreau. Expected time analysis for delaunay point location. *Computational geometry*, 29(2):61–89, 2004. `doi:10.1016/j.comgeo.2004.02.002`.

[6] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Biology*, 18(3):259–278, 09 1969. `doi:10.2307/2412323`.

[7] D. G. Kirkpatrick and J. D. Radke. A framework for computational morphology. In *Machine Intelligence and Pattern Recognition*, volume 2, pages 217–248. Elsevier, 1985.

[8] R. Schneider and W. Weil. *Stochastic and Integral Geometry*. Probability and Its Applications. Springer, 2008.

[9] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. `doi:10.1137/0211059`.

Appendix

Proof. Integral for Theorem 2
We have to compute

$$\mathbb{E}\left[\deg\left(p, \mathrm{Del}\right)\right] = \frac{1}{2}\int_{\mathbb{R}^2}\int_{\mathbb{R}^2} \lambda^2 e^{-\lambda|D(p,q,r)|}\mathrm{d}r\mathrm{d}q.$$

We use a Blaschke-Petkantschin like variables substitution [8, Theorem 7.2.7] from $\mathbb{R}^4$ to $\mathbb{R}^+ \times [0, 2\pi)^3$, to express the parameterization of q and r into $(\rho, \varphi, \theta_q, \theta_r)$ where (ρ, φ) denotes the polar coordinates of the center c of the circle circumscribing p, q, and r, and θ_q and θ_r denote the angles of the points q and r from c to the horizontal line (see Figure 12).

$$x_q = \rho(\cos\varphi + \cos\theta_q),\ \ y_q = \rho(\sin\varphi + \sin\theta_q),$$
$$x_r = \rho(\cos\varphi + \cos\theta_r),\ \ y_\rho = r(\sin\varphi + \sin\theta_r).$$

The Jacobian matrix J of the transformation can be written:

$$J(\rho,\varphi,\theta_q,\theta_r) = \begin{pmatrix} \cos\varphi+\cos\theta_q & -\rho\sin\varphi & -\rho\sin\theta_q & 0 \\ \sin\varphi+\sin\theta_q & \rho\cos\varphi & \rho\cos\theta_q & 0 \\ \cos\varphi+\cos\theta_r & -\rho\sin\varphi & 0 & -\rho\sin\theta_r \\ \sin\varphi+\sin\theta_r & \rho\cos\varphi & 0 & \rho\cos\theta_r \end{pmatrix},$$

and has the following determinant:

$$\begin{aligned}
&det\left(J(\rho,\varphi,\theta_q,\theta_r)\right) \\
&= \begin{vmatrix} \cos\varphi+\cos\theta_q & -\rho\sin\varphi & -\rho\sin\theta_q & 0 \\ \sin\varphi+\sin\theta_q & \rho\cos\varphi & \rho\cos\theta_q & 0 \\ \cos\varphi+\cos\theta_r & -\rho\sin\varphi & 0 & -\rho\sin\theta_r \\ \sin\varphi+\sin\theta_r & \rho\cos\varphi & 0 & \rho\cos\theta_r \end{vmatrix} \\
&= \quad (\cos\varphi+\cos\theta_q)\begin{vmatrix} \rho\cos\varphi & \rho\cos\theta_q & 0 \\ -\rho\sin\varphi & 0 & -\rho\sin\theta_r \\ \rho\cos\varphi & 0 & \rho\cos\theta_r \end{vmatrix} \\
&\quad - (\sin\varphi+\sin\theta_q)\begin{vmatrix} -\rho\sin\varphi & -\rho\sin\theta_q & 0 \\ -\rho\sin\varphi & 0 & -\rho\sin\theta_r \\ \rho\cos\varphi & 0 & \rho\cos\theta_r \end{vmatrix} \\
&\quad + (\cos\varphi+\cos\theta_r)\begin{vmatrix} -\rho\sin\varphi & -\rho\sin\theta_q & 0 \\ \rho\cos\varphi & \rho\cos\theta_q & 0 \\ \rho\cos\varphi & 0 & \rho\cos\theta_r \end{vmatrix} \\
&\quad - (\sin\varphi+\sin\theta_r)\begin{vmatrix} -\rho\sin\varphi & -\rho\sin\theta_q & 0 \\ \rho\cos\varphi & \rho\cos\theta_q & 0 \\ -\rho\sin\varphi & 0 & -\rho\sin\theta_r \end{vmatrix}
\end{aligned}$$

We develop from the coefficient that is the only not zero in a column,

$$\begin{aligned}
&= \quad (\cos\varphi+\cos\theta_q)\left(-\rho\cos\theta_q\right)\left(-\rho^2\sin\varphi\cos\theta_r+\rho^2\cos\varphi\sin\theta_r\right) \\
&\quad - (\sin\varphi+\sin\theta_q)\left(\rho\sin\theta_q\right)\left(-\rho^2\sin\varphi\cos\theta_r+\rho^2\cos\varphi\sin\theta_r\right) \\
&\quad + (\cos\varphi+\cos\theta_r)\left(\rho\cos\theta_r\right)\left(-\rho^2\sin\varphi\cos\theta_q+\rho^2\cos\varphi\sin\theta_q\right) \\
&\quad - (\sin\varphi+\sin\theta_r)\left(-\rho\sin\theta_r\right)\left(-\rho^2\sin\varphi\cos\theta_q+\rho^2\cos\varphi\sin\theta_q\right) \\
&= \rho^3\Big(\left(-\cos\varphi\cos\theta_q-\cos^2\theta_q\right)\left(-\sin\varphi\cos\theta_r+\cos\varphi\sin\theta_r\right) \\
&\quad - \left(\sin\varphi\sin\theta_q+\sin^2\theta_q\right)\left(-\sin\varphi\cos\theta_r+\cos\varphi\sin\theta_r\right) \\
&\quad + \left(\cos\varphi\cos\theta_r+\cos^2\theta_r\right)\left(-\sin\varphi\cos\theta_q+\cos\varphi\sin\theta_q\right) \\
&\quad - \left(-\sin\varphi\sin\theta_r-\sin^2\theta_r\right)\left(-\sin\varphi\cos\theta_q+\cos\varphi\sin\theta_q\right)\Big).
\end{aligned}$$

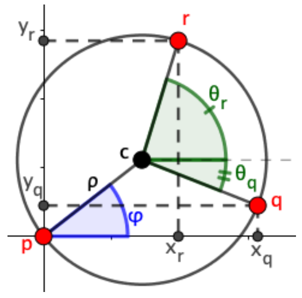


Figure 12: The Blaschke-Petkantchin variables substitution converts the Cartesian coordinates of q and r into polar coordinates related to the circle circumscribing p, q and r.

We factorize by the right factor,

$$\begin{aligned}
&= \rho^3\big((-\cos\varphi\cos\theta_q - \cos^2\theta_q - \sin\varphi\sin\theta_q - \sin^2\theta_q)\\
&\qquad\cdot(-\sin\varphi\cos\theta_r + \cos\varphi\sin\theta_r)\\
&\quad + (\cos\varphi\cos\theta_r + \cos^2\theta_r + \sin\varphi\sin\theta_r + \sin^2\theta_r)\\
&\qquad\cdot(-\sin\varphi\cos\theta_q + \cos\varphi\sin\theta_q)\big)\\
&= \rho^3\big((-\cos\varphi\cos\theta_q - \sin\varphi\sin\theta_q - 1)(-\sin\varphi\cos\theta_r + \cos\varphi\sin\theta_r)\\
&\quad + (\cos\varphi\cos\theta_r + \sin\varphi\sin\theta_r + 1)(-\sin\varphi\cos\theta_q + \cos\varphi\sin\theta_q)\big).
\end{aligned}$$

We distribute the 1, develop, and many terms cancel each other,

$$\begin{aligned}
&= \rho^3\big(\sin\theta_q\cos\theta_r - \cos\theta_q\sin\theta_r + \sin\varphi\cos\theta_r\\
&\qquad - \cos\varphi\sin\theta_r - \sin\varphi\cos\theta_q + \cos\varphi\sin\theta_q\big).
\end{aligned}$$

Finally we apply the formulae: $\cos a\sin b - \cos b\sin a = \sin(a-b)$, on the three well-chosen pairs of terms,

$$\begin{aligned}
&= \rho^3(\sin(\theta_q-\theta_r) + \sin(\theta_q-\varphi) + \sin(\varphi-\theta_r))\\
&= \rho^3(\sin(\pi-(\theta_q-\theta_r)) + \sin(\theta_q-\varphi) + \sin(\varphi-\theta_r))\\
&= 4\rho^3\sin\left(\tfrac{\pi-(\theta_q-\theta_r)}{2}\right)\sin\left(\tfrac{\theta_q-\varphi}{2}\right)\sin\left(\tfrac{\varphi-\theta_r}{2}\right),
\end{aligned}$$

where the last line derives from the formula: $\sin a + \sin b + \sin c = 4\sin\frac{a}{2}\sin\frac{b}{2}\sin\frac{c}{2}$ when $a+b+c=\pi$. So that we get:

$$\begin{aligned}
&\mathbb{E}\left[\deg(p,\mathrm{Del})\right]\\
&= \tfrac{1}{2}\int_{\mathbb{R}}\int_0^{2\pi}\int_0^{2\pi}\int_0^{2\pi}\lambda^2 e^{-\lambda\pi\rho^2}|\det(J(\rho,\varphi,\theta_q,\theta_r))|\,\mathrm{d}\theta_r\mathrm{d}\theta_q\mathrm{d}\varphi\mathrm{d}\rho\\
&= \int_{\mathbb{R}} 2\rho^3\lambda^2 e^{-\lambda\pi\rho^2}\mathrm{d}\rho\\
&\quad\times\int_0^{2\pi}\int_0^{2\pi}\int_0^{2\pi}\left|\sin\left(\tfrac{\pi-(\theta_q-\theta_r)}{2}\right)\sin\left(\tfrac{\theta_q-\varphi}{2}\right)\sin\left(\tfrac{\varphi-\theta_r}{2}\right)\right|\mathrm{d}\theta_r\mathrm{d}\theta_q\mathrm{d}\varphi
\end{aligned}$$

simplified by the translation $(\theta_q,\theta_r)\mapsto(\theta_q-\pi-\varphi,\theta_r-\pi-\varphi)$ applied in the $(2\pi,2\pi)$-periodic function $(\theta_q,\theta_r)\mapsto \sin\left(\frac{\pi-(\theta_q-\theta_r)}{2}\right)\sin\left(\frac{\theta_q-\varphi}{2}\right)\sin\left(\frac{\varphi-\theta_r}{2}\right)$,

$$\begin{aligned}
&= \frac{1}{\pi^2}\times\int_0^{2\pi}\mathrm{d}\varphi\times\int_0^{2\pi}\int_0^{2\pi}\left|\sin\left(\tfrac{\theta_q-\theta_r}{2}\right)\right|\sin\tfrac{\theta_q}{2}\sin\tfrac{\theta_r}{2}\mathrm{d}\theta_r\mathrm{d}\theta_q\\
&= \frac{1}{\pi^2}\times 2\pi\times 3\pi = 6. \qquad\square
\end{aligned}$$

Proposition 9 *For a given $q\in\mathbb{R}^2$ and for $\alpha\in\mathbb{R}^+$, consider the ellipse $\mathrm{r}_c(p,q)$ parameterized by $c = (\alpha^2\frac{x_q}{2}, \frac{y_q}{2})$.*

The geometric center of $\mathrm{r}_c(p,q)$ is the midpoint of $[pq]$, and its area is $\frac{\pi}{4}\left(\alpha x_q^2 + \frac{y_q^2}{\alpha}\right)$.

Proof. of Proposition 9 We note $E_c(x,y) := \alpha^2x^2 - 2xx_c + y^2 - 2yy_c$ with $\alpha^2 = \frac{2x_qx_c - y_q^2 + 2y_qy_c}{x_q^2}$. If $y_c = \frac{y_q}{2}$, then $\alpha^2x_q^2 - 2x_qx_c = 0$, and so $x_c = \alpha^2\frac{x_q}{2}$.

$$\begin{aligned}
E_c(x,y) &= \alpha^2x^2 - 2xx_c + y^2 - 2yy_c\\
&= \alpha^2(x^2 - xx_q) + y^2 - yy_q\\
&= \alpha^2(x-\frac{x_q}{2})^2 + (y-\frac{y_q}{2})^2 - \alpha^2\frac{x_q^2}{4} - \frac{y_q^2}{4}.
\end{aligned}$$

Dividing by $\frac{\alpha^2x_q^2+y_q^2}{4}$, another equation of $\mathrm{r}_c(p,q)$ is:

$$\frac{4\alpha^2}{\alpha^2x_q^2+y_q^2}(x-\frac{x_q}{2})^2 + \frac{4}{\alpha^2x_q^2+y_q^2}(y-\frac{y_q}{2})^2 - 1 < 0.$$

We identify, with that expression, that $\mathrm{r}_c(p,q)$ is the translation by the vector $\frac{1}{2}\overrightarrow{pq}$ of the ellipse defined by:

$$\frac{4\alpha^2}{\alpha^2x_q^2+y_q^2}x^2 + \frac{4}{\alpha^2x_q^2+y_q^2}y^2 - 1 = 0,$$

whose center is p, and area is $\frac{\pi}{4}\left(\alpha x_q^2 + \frac{y_q^2}{\alpha}\right)$. $\square$

Lemma 16 *Let $t>0$, $\beta\in(0,1)$ and*

$$I_\beta(t) = \int_{\mathbb{R}}\int_{\mathbb{R}} e^{-t\sqrt{(x^2+y^2)(\beta^2x^2+y^2)}}\mathrm{d}y\mathrm{d}x,$$

$$I_\beta(t) = \frac{1}{t}I_\beta(1) \le \frac{\pi}{t}\left(1+\ln(\tfrac{1}{\beta})\right).$$

Proof. of Lemma 16
We apply, in the integral, the variables substitution: $(x,y) = (\frac{1}{\sqrt{t}}X, \frac{1}{\sqrt{t}}Y)$ with Jacobian determinant $\frac{1}{t}$.

$$\begin{aligned}
I_\beta(t) &= \int_{\mathbb{R}}\int_{\mathbb{R}} e^{-t\sqrt{(x^2+y^2)(\beta^2x^2+y^2)}}\mathrm{d}y\mathrm{d}x\\
&= \int_{\mathbb{R}}\int_{\mathbb{R}} \frac{1}{t}e^{-\sqrt{(X^2+Y^2)(\beta^2X^2+Y^2)}}\mathrm{d}Y\mathrm{d}X\\
&= \frac{1}{t}I_\beta(1).
\end{aligned}$$

Then we compute an upper bound:

$$\begin{aligned}
I_\beta(1) &= \int_{\mathbb{R}}\int_{\mathbb{R}} e^{-\sqrt{(x^2+y^2)(\beta^2x^2+y^2)}}\mathrm{d}y\mathrm{d}x\\
&= 4\int_0^\infty\int_0^\infty e^{-\sqrt{(x^2+y^2)(\beta^2x^2+y^2)}}\mathrm{d}y\mathrm{d}x\\
&= 4\int_0^{\frac{\pi}{2}}\int_0^\infty re^{-r^2\sqrt{\beta^2\cos^2\theta+\sin^2\theta}}\mathrm{d}r\mathrm{d}\theta\\
&= 2\int_0^{\frac{\pi}{2}}\left(\beta^2\cos^2\theta+\sin^2\theta\right)^{-\frac{1}{2}}\mathrm{d}\theta.
\end{aligned}$$

On $[0,\frac{\pi}{2}]$, $\left(\beta^2\cos^2\theta+\sin^2\theta\right)^{-\frac{1}{2}}$ is smaller than both $\frac{1}{\beta}$ and $\frac{\pi}{2\theta}$; on the one hand, because $\left(\beta^2\cos^2\theta+\sin^2\theta\right)^{-\frac{1}{2}}$ decreases from $\frac{1}{\beta}$ to 1, on the other hand, because $\left(\beta^2\cos^2\theta+\sin^2\theta\right)^{\frac{1}{2}} \ge \sin\theta \ge \frac{2}{\pi}\theta$, so that:

$$\begin{aligned}
I_\beta(1) &\le 2\int_0^{\frac{\pi}{2}}\min\left(\tfrac{1}{\beta},\tfrac{\pi}{2\theta}\right)\mathrm{d}\theta\\
&= 2\left(\int_0^{\beta\frac{\pi}{2}}\tfrac{1}{\beta}\mathrm{d}\theta + \int_{\beta\frac{\pi}{2}}^{\frac{\pi}{2}}\tfrac{\pi}{2\theta}\mathrm{d}\theta\right)\\
&= \pi\left(1-\ln(\beta)\right). \qquad\square
\end{aligned}$$

Rearranging a Sequence of Points onto a Line*

Taehoon Ahn† Jongmin Choi† Chaeyoon Chung† Hee-Kap Ahn‡ Sang Won Bae§
Sang Duk Yoon¶

Abstract

Given a sequence of n weighted points $\langle p_1, p_2, \ldots, p_n\rangle$ in the plane, we consider the problem of finding a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than their weight difference, and the maximum distance between p_i and q_i over all i is minimized. We present efficient algorithms that compute an optimal rearrangement for three variants of the problem under the Euclidean metric. When the line is fully specified or partially specified by only its orientation, our algorithms take near-linear time. When we need to find a target line, onto which the input sequence can be rearranged with the optimal rearrangement cost, we present an $O(n^3 \operatorname{polylog} n)$-time algorithm.

1 Introduction

Consider an object moving in the plane and its trajectory data which can be represented by a sequence of pairs, each consisting of a time stamp and the coordinates of the object at the time. One popular problem concerning such trajectories is to determine whether the object follows a path of a certain shape. The quality of the trajectory with respect to the path can be measured by their similarity, that is, how closely the trajectory follows the path in increasing order of time stamps. Therefore, in a good trajectory, every trajectory point can be translated to a point in the path such that the translation distance is small and two consecutive trajectory points are translated to points close to each other along the path. Formally, we define this problem for linear paths as follows.

Weighted point-to-line rearrangement. Given a sequence of n points $\langle p_1, p_2, \ldots, p_n\rangle$ in the plane and their weights w_i for p_i with $w_1 \leq w_2 \leq \cdots \leq w_n$, find a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than $w_{i+1} - w_i$, and the maximum distance between p_i and q_i over all i is minimized. We call such a rearrangement an *optimal* rearrangement of the point sequence, and the maximum distance of an optimal rearrangement the *optimal* rearrangement cost.

Observe that the constraint on the distance of two consecutive points implies that any two points q_i and q_j $(i \leq j)$ are at distance no more than $w_j - w_i$. See Figure 1 for an illustration of rearranging five points onto a line ℓ.

We consider three variants of the problem: (1) the rearrangement line is given, (2) only the orientation of the rearrangement line is given, or (3) the rearrangement line is not specified at all. For the variants (2) and (3), we need to find a best line, onto which the input sequence can be rearranged with the optimal rearrangement cost, and realize such a rearrangement.

For ease of presentation, we will discuss a special case in which $w_i = i$ for every index i. We first describe our algorithms for this special case and then show how to extend to the general weighted problem without increasing time complexities. The special case is equivalent to the following unweighted problem.

(Unweighted) point-to-line rearrangement. Given a sequence of n points $\langle p_1, p_2, \ldots, p_n\rangle$ in the plane, find a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than 1, and the maximum distance between p_i and q_i over all i is minimized.

Again, the constraint on the distance between two consecutive points implies that any two points q_i and q_j $(i \leq j)$ are at distance no more than $j - i$.

*T.Ahn, J.Choi, C.Chung, and H.-K.Ahn were supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)). S.W.Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07042755). S.D.Yoon was supported by "Cooperative Research Program for Agriculture Science & Technology Development (Project No. PJ015269032021)" Rural Development Administration, Republic of Korea.

†Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. {sloth, icothos, chaeyoon17}@postech.ac.kr

‡Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Pohang, Korea. heekap@postech.ac.kr

§Department of Computer Science, Kyonggi University, Suwon, Korea. swbae@kgu.ac.kr

¶Department of Service and Design Engineering, Sungshin Women's University, Seoul, Korea. sangduk.yoon@sungshin.ac.kr

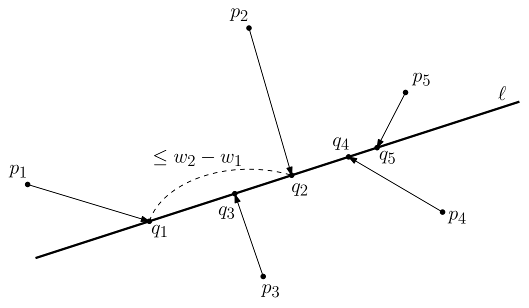


Figure 1: A rearrangement $\langle q_1, ..., q_5\rangle$ of five points $\langle p_1, ..., p_5\rangle$ onto ℓ. Any two points q_i and q_j ($i \leq j$) are at distance no more than $w_j - w_i$. The cost of this rearrangement is the maximum distance between p_i and q_i over all $i = 1, \dots, 5$.

Related work. There has been a fair amount of work on rearranging points with respect to certain objectives. One of those problems the most related to ours asks to find the *center line* that minimizes the maximum distance from the input points to the line. This can be solved in $O(n \log n)$ time by computing the center line of a minimum-width slab for the points from the convex hull [11, 12]. Notice that the center line problem implicitly assumes rearrangements of input points obtained by their orthogonal projections onto a line, and hence the center line problem is equivalent to our rearrangement problem where the proximity constraint of consecutive points is relaxed.

Similarly, a circle that aggregates a point set can be found from the minimum-width annulus for the points [9]. There is a deterministic $O(n^{8/5+\varepsilon})$-time algorithm [2] and an expected $O(n^{3/2+\varepsilon})$-time algorithm [1] for computing the minimum-width annulus for n points in the plane. When the radius r of the aggregating circle is fixed, there is an $O(n \log n)$-time algorithm [8, 10] that finds the minimum-width annulus with median radius r.

For a sequence of n points in the increasing order of x-coordinates in the plane, there has been a series of work to find an x-monotone curve with minimum error under various settings on the curve. When the curve is a polyline, the segmented least squares algorithm finds the polyline that minimizes a combination of the total squared errors and the number of segments in the polyline in $O(n^2)$ time [4].

When the weights of the input points are the same, we have $w_{i+1} - w_i = 0$ for all i, and any rearrangement $\langle q_1, \dots, q_n\rangle$ has the same point for all q_i's. Thus, the optimal rearrangement is achieved by the point $q \in \ell$ such that the smallest disk centered at q and enclosing the input points has minimum radius among all enclosing disks centered at points of ℓ. There is an $O(n)$-time algorithm [14] for finding the center, and an $O(n \log n)$-time algorithm for k centers restricted to a line [18].

Our results. We present efficient algorithms for finding an optimal rearrangement among the rearrangements of the sequence S of n points onto a line under the Euclidean metric. For the case that the rearrangement line ℓ is given, we observe that the cost of the optimal rearrangement of S onto ℓ is determined by at most two points, and present a simple $O(n^2)$-time algorithm. Then we improve the running time to near-linear by applying several optimization techniques. We present an expected $O(n)$-time algorithm using randomized optimization [6] and a deterministic $O(n \log \log n)$-time algorithm using parametric search [7] with some additional preprocessing.

For the case that only the orientation of the rearrangement line is given, we compute an optimal rearrangement line ℓ among all lines of the orientation and an optimal rearrangement onto ℓ using a set of $O(n)$ convex functions, each representing the optimal rearrangement cost of a contiguous subsequence of S. The upper envelope of those functions coincides with the function of the optimal rearrangement cost of S. By applying convex programming, our algorithm computes an optimal rearrangement in $O(n \log n)$ time. For the case that the rearrangement line is not specified at all, we present an $O(n^3 \operatorname{polylog} n)$-time deterministic algorithm that finds an optimal rearrangement line ℓ and an optimal rearrangement onto ℓ. The detailed algorithms for the case that the rearrangement line is not specified can be found in the full version of this paper.

2 Preliminaries

Let $S = \langle p_1, \dots, p_n\rangle$ denote the input sequence of points, and $w_1, \dots, w_n$ be their weights with $w_1 \leq \cdots \leq w_n$. Throughout the paper, we mainly discuss the unweighted problem, so we assume $w_i = i$ for each $i = 1, \dots, n$, unless stated otherwise. For any $1 \leq i \leq j \leq n$, we denote by S_{ij} the contiguous subsequence of S from p_i to p_j, that is, $S_{ij} = \langle p_i, \dots, p_j\rangle$. Let $\| \cdot \|$ denote the Euclidean norm on the plane so that we use $\|p - q\|$ to denote the distance between two points p and q. A sequence $\langle q_i, \dots, q_j\rangle$ of points is a *rearrangement* of S_{ij} onto a line ℓ if q_k lies on ℓ and $\|q_{k'} - q_k\| \leq w_{k'} - w_k$ for every k and k' with $i \leq k \leq k' \leq j$. Its *cost* is defined to be $\max_{i \leq k \leq j} \|q_k - p_k\|$.

An *optimal* rearrangement of S onto a line ℓ is a rearrangement with the minimum cost among all rearrangements of S onto ℓ. An optimal rearrangement of S onto a set of lines is a rearrangement with the minimum cost among all optimal rearrangement of S over the lines in the set. We use $\delta^*(\ell)$ to denote the cost of an optimal rearrangement of S onto ℓ. We may simply write δ^* if it is understood from the context.

For a real number $r \geq 0$, we denote by $I(r)$ the segment of length $2r$ joining two points $(-r, 0)$ and $(r, 0)$

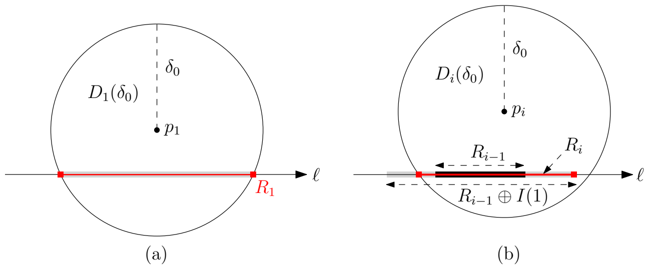


Figure 2: (a) R_1 is defined as the intersection of $D_1(\delta_0)$ and ℓ. (b) R_i is defined as the intersection of $D_i(\delta_0)$ and $R_{i-1} \oplus I(1)$.

on the x-axis. We will often consider the Minkowski sum of a compact set X in the plane and the segment $I(r)$, denoted by $X \oplus I(r)$.

3 Rearrangement onto a Fixed Line

In this section, we consider the problem when the rearrangement line is given as an input, so we are given a sequence $S = \langle p_1, \ldots, p_n \rangle$ and a line ℓ, and want to find an optimal rearrangement of S onto ℓ. Without loss of generality, assume that ℓ is the x-axis. For any real number r, we abuse the notation so that r also denotes the point $(r, 0)$ on the x-axis ℓ if there is no confusion from the context.

We first present an $O(n)$-time decision algorithm determining for a given real value $\delta_0 \geq 0$, whether there exists a rearrangement of S with cost at most δ_0. We define a *feasible range* R_i for each point p_i of S, representing the range in the x-axis in which q_i of a rearrangement $\langle q_1, \ldots, q_i \rangle$ of S_{1i} onto ℓ with cost at most δ_0 can be placed. The decision algorithm computes feasible ranges as follows: $R_1 = \ell \cap D_1(\delta_0)$ and $R_i = (R_{i-1} \oplus I(1)) \cap D_i(\delta_0)$ for $1 < i \leq n$, where $D_i(\delta_0) := \{q \mid \|q - p_i\| \leq \delta_0\}$ denotes the disk with center p_i and radius δ_0. See Figure 2 for an illustration of the ranges.

Lemma 1 *There is a rearrangement of S_{1i} with cost at most δ_0 and p_i rearranged to q_i if and only if $q_i \in R_i$.*

Proof. We first prove the if part by induction. If $i = 1$, it is trivial. For any $i > 1$, we pick a point $q_i \in R_i$. Then we have $(q_i \oplus I(1)) \cap R_{i-1} \neq \emptyset$ as $q_i \in R_{i-1} \oplus I(1)$. Pick a point q_{i-1} in $(q_i \oplus I(1)) \cap R_{i-1}$. Then, by the induction hypothesis, there is a rearrangement $\langle q_1, \ldots, q_{i-1} \rangle$ of $S_{1(i-1)}$ with cost at most δ_0. Since $\|q_i - q_{i-1}\| \leq 1$, $\langle q_1, \ldots, q_{i-1}, q_i \rangle$ is a rearrangement of S_{1i} with cost at most δ_0.

We now prove the only if part by induction. It is trivial for $i = 1$. For any $i > 1$, let $\langle q_1, \ldots, q_i \rangle$ be a rearrangement of S_{1i} with cost at most δ_0. We have $q_i \in R_{i-1} \oplus I(1)$ because $\|q_{i-1} - q_i\| \leq 1$, and $q_{i-1} \in R_{i-1}$ by the induction hypothesis. We also have $q_i \in \ell \cap D_i(\delta_0)$ because the cost of the rearrangement is at most δ_0. Therefore, $q_i \in R_i$. $\square$

By Lemma 1, the decision problem can be answered by checking whether $R_n \neq \emptyset$ (`yes`) or $R_n = \emptyset$ (`no`). Since we can compute R_1 in $O(1)$ time, and R_i in $O(1)$ time once we have R_{i-1}, we can compute R_n in $O(n)$ time. If $R_n \neq \emptyset$, we can compute a rearrangement $\langle q_1, \ldots, q_n \rangle$ of S in $O(n)$ time, by choosing q_n from R_n, and then choosing q_i from $(q_{i+1} \oplus I(1)) \cap R_i$ repeatedly for i from $n-1$ to 1.

Lemma 2 *Given a point sequence S of n points, a line ℓ, and a real value δ_0, we can decide whether there exists a rearrangement of S onto ℓ with cost at most δ_0 in $O(n)$ time. If such rearrangement exists, we can compute a rearrangement of S with cost at most δ_0 in $O(n)$ time.*

We present some characterizations of an optimal rearrangement of S. We first show that there are at most two points of S which determine the cost $\delta^* = \delta^*(\ell)$ of an optimal rearrangement in the following lemma.

Lemma 3 *There exists an optimal rearrangement $\langle q_1, \ldots, q_n \rangle$ of S onto ℓ satisfying one of the followings.*

(1) There is a point p_j in S such that $\|p_j - q_j\| = \delta^*$ and q_j is the orthogonal projection of p_j onto ℓ.

(2) There are two points p_i and p_j $(i < j)$ in S such that $\|p_i - q_i\| = \|p_j - q_j\| = \delta^*$, $\|q_i - q_j\| = j - i$, and both q_i and q_j lie in between the orthogonal projections of p_i and p_j onto ℓ.

Proof. Among the feasible ranges of the points of S for δ^*, there must be a feasible range that is a single point. Otherwise, there is a real value $\epsilon > 0$ such that $R_n \neq \emptyset$ with cost $(\delta^* - \epsilon)$, which contradicts the optimality of δ^*.

If a feasible range R_j is a single point, then ℓ is tangent to $D_j(\delta^*)$, or $D_j(\delta^*)$ intersects $R_{j-1} \oplus I(1)$ only at an endpoint. The former case implies that q_j is the orthogonal projection of p_j onto ℓ with $\|p_j - q_j\| = \delta^*$, and thus we have case (1). For the latter case, observe that an endpoint of $R_{j-1} \oplus I(1)$ is an endpoint of the intersection of $D_i(\delta^*) \oplus I(j-i)$ and ℓ for some i with $1 \leq i < j$. Therefore, the common intersection of $D_i(\delta^*) \oplus I(j-i)$, ℓ, and $D_j(\delta^*)$ is just a single point. Then, $\|p_i - q_i\| = \|p_j - q_j\| = \delta^*$, $\|q_j - q_i\| = j - i$, and both q_i and q_j lie in between the orthogonal projections of p_i and p_j onto ℓ. Thus, we have case (2). $\square$

For an optimal rearrangement, we call the points of S that satisfy cases (1) or (2) of Lemma 3 the *determinators* of the rearrangement. We now define a value δ_{ij} for every two indices $1 \leq i \leq j \leq n$. Let q_i and q_j be the points on ℓ minimizing $\max\{\|p_i - q_i\|, \|p_j - q_j\|\}$ with

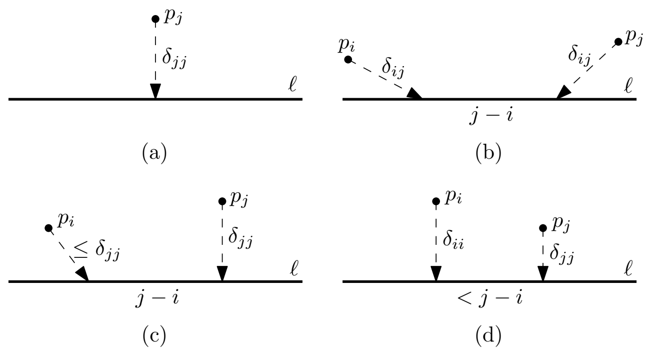


Figure 3: (a) δ_{jj} is the length of the projection from p_j to ℓ. (b) When the difference of x-coordinates between two points p_i and p_j $(i<j)$ is bigger than $j-i$, δ_{ij} is defined by two points in between the orthogonal projections of p_i and p_j with distance $j-i$. (c) One of the points that define δ_{ij} can be the orthogonal projection of p_j, so that $\delta_{ij}=\delta_{jj}$. (d) When the difference of x-coordinates between two points p_i and p_j is smaller than or equal to $j-i$, $\delta_{ij}=\max\{\delta_{ii},\delta_{jj}\}$.

$\|q_i-q_j\|\leq j-i$. Then $\delta_{ij}=\max\{\|p_i-q_i\|,\|p_j-q_j\|\}$. (Figure 3). Note that δ_{jj} is the length of the orthogonal projection of p_j to ℓ. Then δ_{ij} denotes the minimum cost required by two points p_i and p_j of S such that there is a rearrangement of S.

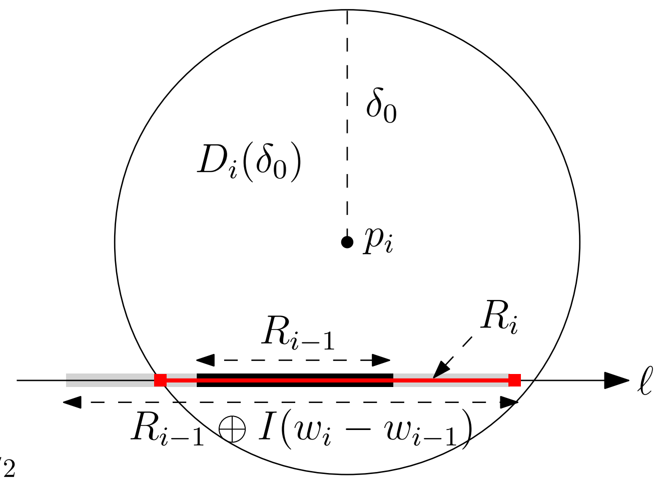
Lemma 4 $\delta^*=\max_{i,j}\delta_{ij}$.

Proof. By Lemma 3, there is an optimal rearrangement with determinators. If the optimal rearrangement belongs to case (1) of Lemma 3, then $\delta^*=\delta_{jj}$ for the determinator p_j (Figure 3(a)). If it belongs to case (2) of Lemma 3, then $\delta^*=\delta_{ij}$ for the determinators p_i and p_j. Therefore, $\delta^*\leq\max_{i,j}\delta_{ij}$ holds (Figure 3(b)).

If $\delta^*<\delta_{ii}$, there is no point $q\in\ell$ such that $\|p_i-q\|\leq\delta^*$, which is a contradiction. Assume that there are two indices i,j $(i<j)$ such that $\delta^*<\delta_{ij}$. There is an optimal rearrangement $Q^*=\langle q_1^*,\dots,q_n^*\rangle$ with cost δ^*. However, $\|q_i^*-q_j^*\|>j-i$ holds by the assumption, which contradicts that Q^* is a rearrangement. Therefore, $\max_{i,j}\delta_{ij}\leq\delta^*$ holds. □

3.1 Randomized algorithm

This problem can be solved in $O(n)$ expected time using the randomized optimization technique by Chan [6] as follows. We consider the weighted version of the problem in which a sequence $S=\langle p_1,\dots,p_n\rangle$ of n weighted points is given, the weight of p_i denoted by w_i, satisfying $w_i\leq w_j$ for every pair of indices i,j with $i<j$. The objective is to find a rearrangement $Q=\langle q_1,\dots,q_n\rangle$ of S onto ℓ such that $\|q_i-q_j\|\leq w_j-w_i$ for every pair of indices i,j with $i\leq j$, and the rearrangement cost $\max_i\|p_i-q_i\|$ is minimized.



/2/2

Figure 4: The definition of R_i for $i>1$ is replaced.

Observe that Lemmas 2, 3, and 4 also hold for this weighted version, by replacing the definition of R_i with $R_i=(R_{i-1}\oplus I(w_i-w_{i-1}))\cap D_i(\delta)$ (R_1 remains the same) and by replacing the condition $\|q_i-q_j\|=j-i$ in case (2) of Lemma 3 with $\|q_i-q_j\|=w_j-w_i$ (Figure 4). Thus, we have the following corollary.

Corollary 5 *For a sequence S of n weighted points, a line ℓ, and a real value δ_0, we can decide whether there exists a rearrangement of S onto ℓ with cost at most δ_0 in $O(n)$ time.*

Let S_1, S_2 and S_3 be subsequences of S with length at most $\lceil n/3\rceil$ and $S_1;S_2;S_3=S$ where $A;B$ is the concatenation of two sequences A and B such that the elements of B comes after the last element of A in the concatenation. Then $\delta(S)=\max\{\delta(S_1;S_2),\delta(S_1;S_3),\delta(S_2;S_3)\}$ by Lemma 3, where $\delta(A)$ denotes the optimal rearrangement cost of a sequence A. Therefore, by applying the randomized optimization technique by Chan, we obtain a randomized algorithm to compute δ^*, which takes the time linear to the running time of the decision algorithm, $O(n)$. By Lemma 2, we can compute a rearrangement with cost δ^* using $O(n)$ additional time.

Theorem 6 *Given a sequence S of n weighted points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in expected $O(n)$ time.*

3.2 Deterministic algorithm

By Lemma 4, we get an $O(n^2)$-time deterministic algorithm to compute δ^* that computes δ_{ij} for every pair of indices i,j with $i\leq j$ and returns the maximum value among them. We present a more efficient deterministic algorithm for the problem. We first extend the definition of R_i to define a function representing the range within which q_i can be placed with respect to the position of q_1 and the cost δ. We present sub-linear time sequential and parallel decision algorithms using those functions after preprocessing. By applying parametric search, we obtain an $O(n \log\log n)$-time algorithm to compute the optimal rearrangement cost δ^*.

Recall that R_i denotes the feasible range for p_i of S with a fixed cost. For the deterministic algorithm, we consider R_i as a function of the cost variable δ and a real value r, and thus we use $R_i(\delta, r)$ to denote the function. For a fixed cost δ_0 and a fixed value r_0, $R_i(\delta_0, r_0)$ represents the range in ℓ on which q_i of a rearrangement $\langle q_1 = r_0, \ldots, q_i\rangle$ of S_{1i} with cost at most δ_0 can be placed. Our algorithm takes S as input and computes $R_n(\delta, r)$ in the rearrangements of S for all cost values δ and real values r.

Characterization of $R_n(\delta, r)$ for a fixed cost δ_0. To characterize $R_i(\delta, r)$, we set δ to a fixed value δ_0, and use $R_i(r)$ to denote $R_i(\delta_0, r)$. Observe that $R_i(r)$ is an interval on ℓ and its two boundary points can be described by functions B_i and T_i defined on r such that $R_i(r) = [B_i(r), T_i(r)]$. In case that $R_i(r) = \emptyset$, B_i and T_i are not defined for r. We use dom_i to denote the range of r for which $R_i(r) \neq \emptyset$, and thus B_i and T_i are defined for $r \in \mathsf{dom}_i$.

Observe that $R_i(r)$ can be defined inductively as R_i in the beginning of Section 3. We have $R_1(r) = [r, r]$ which is defined for $r \in \mathsf{dom}_1 = \ell \cap D_1(\delta_0)$, and $R_i(r) = (R_{i-1}(r) \oplus I(1)) \cap D_i(\delta_0)$ for $1 < i \leq n$.

Lemma 7 *There is a rearrangement $\langle q_1 = r, \ldots, q_n\rangle$ of S onto ℓ with cost at most δ_0 if and only if $q_n \in R_n(r)$.*

Proof. If S consists of one point, the lemma holds by the definition of $R_n(r)$. For S consisting of more than one point, we prove the lemma by induction.

Assume $q_n \in R_n(r)$. Then $R_{n-1}(r) \neq \emptyset$, and this implies $r \in \mathsf{dom}_{n-1}$. Also, by the definition of $R_n(r)$, $\|p_n - q_n\| \leq \delta_0$ and there exists a point $q' \in R_{n-1}(r)$ with $\|q_n - q'\| \leq 1$. By the induction hypothesis, we get a rearrangement $\langle q_1 = r, \ldots, q_{n-1}\rangle$ of $S_{1(n-1)}$ with cost at most δ_0. Then we obtain the rearrangement $\langle q_1 = r, \ldots, q_{n-1}, q_n\rangle$ of S with cost at most δ_0.

Let $Q = \langle q_1 = r, \ldots, q_{n-1}, q_n\rangle$ be a rearrangement of S onto ℓ with cost at most δ_0. By the induction hypothesis, we have $r \in \mathsf{dom}_{n-1}$ and $q_{n-1} \in R_{n-1}(r)$. Since we have $q_n \in (q_{n-1} \oplus I(1))$ and $q_n \in D_n(\delta_0)$, $q_n \in R_n(r)$ holds. $\square$

Let b_i and t_i be the two boundary points of $\ell \cap D_i(\delta_0)$ with $b_i \leq t_i$. If $\ell \cap D_i(\delta_0) = \emptyset$, b_i and t_i are not defined. If ℓ is tangent to $D_i(\delta_0)$, $b_i = t_i$. We can check in $O(n)$ time whether b_i and t_i are defined for every i with $1 \leq i \leq n$. Since there is a rearrangement with cost δ_0 only if b_i and t_i are defined for every i, we assume that they are defined for every i in the remainder of this section. In the following lemmas, we show that $B_n(r)$, $T_n(r)$, and dom_n can be expressed using b_i's and t_i's.

Lemma 8 *For $r \in \mathsf{dom}_n$, $B_n(r) = \max_{1\leq i\leq n}\{r - n + 1, b_i + i - n\}$ and $T_n(r) = \min_{1\leq i\leq n}\{r + n - 1, t_i - i + n\}$.*

Proof. We prove the claim by induction. When $n = 1$, $[B_1(r), T_1(r)] = [r, r] = [r - 1 + 1, r + 1 - 1]$. Since $b_1 \leq r \leq t_1$ for any $r \in \mathsf{dom}_1$, the statement holds. For an index $j > 1$, $\mathsf{dom}_j \subseteq \mathsf{dom}_{j-1}$ holds since $R_j(r) \neq \emptyset$ only if $R_{j-1}(r) \neq \emptyset$. Therefore, for $r \in \mathsf{dom}_j$, $r \in \mathsf{dom}_{j-1}$. Then $[B_j(r), T_j(r)] = [B_{j-1}(r) - 1, T_{j-1}(r) + 1] \cap [b_j, t_j]$. Using the induction hypothesis, we can show that $B_j(r) = \max\{B_{j-1}(r) - 1, b_j\} = \max_{1\leq i\leq j}\{r - j + 1, b_i + i - j\}$ holds. We can show the claim for $T_j(r)$ similarly. $\square$

Lemma 9 *If $\mathsf{dom}_n \neq \emptyset$, $\mathsf{dom}_n = \bigcap_{1\leq i\leq n}[b_i - i + 1, t_i + i - 1]$.*

Proof. Let R_j^{rev} denote the range in ℓ on which q_j of a rearrangement $\langle q_j, \ldots, q_n\rangle$ of S_{jn} with cost at most δ_0 can be placed. By Lemma 7, $r \in \mathsf{dom}_n$ if and only if there is a rearrangement $\langle q_1 = r, \ldots, q_n\rangle$ of S onto ℓ with cost at most δ_0. Therefore, $R_1^{\mathsf{rev}} = \mathsf{dom}_n$. We show that $R_j^{\mathsf{rev}} = \bigcap_{j\leq i\leq n}[b_i - i + j, t_i + i - j]$ by induction on j from n to 1.

As the base case, $R_n^{\mathsf{rev}} = \ell \cap D_n(\delta_0) = [b_n, t_n]$. For any $j < n$, let $R_j^{\mathsf{rev}} \neq \emptyset$. As $r \in R_j^{\mathsf{rev}}$ if and only if $r \in (R_{j+1}^{\mathsf{rev}} \oplus I(1))$ and $r \in D_j(\delta_0)$, $R_{j+1}^{\mathsf{rev}} \neq \emptyset$. Then by the induction hypothesis, we have the following equation.

$$R_{j+1}^{\mathsf{rev}} = \bigcap_{j+1\leq i\leq n}[b_i - i + (j+1), t_i + i - (j+1)]$$

Since $R_j^{\mathsf{rev}} = (R_{j+1}^{\mathsf{rev}} \oplus I(1)) \cap [b_j, t_j]$, the claim holds. Therefore, we conclude $\mathsf{dom}_n = R_1^{\mathsf{rev}} = \bigcap_{1\leq i\leq n}[b_i - i + 1, t_i + i - 1]$ if $\mathsf{dom}_n = R_1^{\mathsf{rev}} \neq \emptyset$. $\square$

Observe that $\mathsf{dom}_n = \emptyset$ if $R_j^{\mathsf{rev}} = \emptyset$ for some j, even when $\bigcap_{1\leq i\leq n}[b_i - i + 1, t_i + i - 1] \neq \emptyset$. Therefore, we have to check whether $\mathsf{dom}_n = \emptyset$. By Lemma 8, each of B_n and T_n consists of at most two segments as $\max_{1\leq i\leq n}\{b_i - n + i\}$ and $\min_{1\leq i\leq n}\{t_i + n - i\}$ remain unchanged for different r values. See Figure 5.

Observation 1 *$B_n(r)$ consists of at most two segments, one with slope 0 followed by one with slope 1. $T_n(r)$ consists of at most two segments one with slope 1 followed by one with slope 0.*

Observe that $R_n(r) = [B_n(r), T_n(r)]$ is determined by at most four points of S with indices (a) $\arg\max_i\{b_i + i\}$ and (b) $\arg\min_i\{t_i - i\}$ from Lemma 8, and (c) $\arg\max_i\{b_i - i\}$ and (d) $\arg\min_i\{t_i + i\}$ from Lemma 9. If $\mathsf{dom}_n \neq \emptyset$, $\mathsf{dom}_n = [b_{(\mathrm{c})} - (\mathrm{c}) + 1, t_{(\mathrm{d})} + (\mathrm{d}) - 1]$. For a real value $r \in \mathsf{dom}_n$, $B_n(r) = \max\{r - n + 1, b_{(\mathrm{a})} + (\mathrm{a}) -$

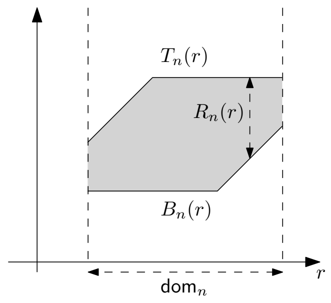


Figure 5: $R_n(r) = [B_n(r), T_n(r)]$, where B_n and T_n defined in dom_n have constant complexity.

$n\}$ and $T_n(r) = \max\{r + n - 1, t_{(\mathrm{b})} - (\mathrm{b}) + n\}$. We call the set of those four points the *combinatorial structure* of $R_n(r)$ for δ_0.

Computing $R_n(\delta, r) = [B_n(\delta, r), T_n(\delta, r)]$**.** Let $B_{ij}(r)$ and $T_{ij}(r)$ denote the two boundary points of the feasible range $R_{ij}(r) = [B_{ij}(r), T_{ij}(r)]$ of p_j with respect to the subsequence S_{ij} with p_i rearranged to r. We can compute $B_n(r)$ and $T_n(r)$ using the feasible ranges $[B_{1k}(r), T_{1k}(r)]$ and $[B_{kn}(r), T_{kn}(r)]$ of two subsequences S_{1k} and S_{kn} of S for any k with $1 < k < n$ in $O(1)$ time.

Lemma 10 *We can compute $B_n(r)$ and $T_n(r)$ of S in $O(1)$ time once we have $B_{1k}(r)$, $T_{1k}(r)$, $B_{kn}(r)$, and $T_{kn}(r)$ for any k with $1 < k < n$.*

Proof. We first check whether $\mathsf{dom}_n = \emptyset$. Let $\mathsf{dom}_{1k} = [r_1, r_2]$, which is the domain of functions $B_{1k}(r)$ and $T_{1k}(r)$. Note that $B_{1k}(r)$ and $T_{1k}(r)$ are monotonically increasing functions by Observation 1. Then the maximal range that q_k of a rearrangement $\langle q_1, \dots, q_k\rangle$ of S_{1k} with cost at most δ_0 can be placed is $[B_{1k}(r_1), T_{1k}(r_2)]$. There is a rearrangement of S if and only if $[B_{1k}(r_1), T_{1k}(r_2)] \cap \mathsf{dom}_{kn} \neq \emptyset$, where dom_{kn} is the domain of $B_{kn}(r)$ and $T_{kn}(r)$. We can check whether $\mathsf{dom}_n = [B_{1k}(r_1), T_{1k}(r_2)] \cap \mathsf{dom}_{kn} = \emptyset$ in $O(1)$ time. If $\mathsf{dom}_n \neq \emptyset$, we can find the combinatorial structure of $R_n(r)$ from the combinatorial structures of $R_{1k}(r)$ and $R_{kn}(r)$ by $O(1)$ comparisions. After finding the combinatorial structure, we can compute $B_n(r)$ and $T_n(r)$ of S in $O(1)$ time. $\square$

Now we treat δ also as a variable of the feasible range R_i and its boundary points B_i and T_i so that $q_i \in R_i(\delta, r) = [B_i(\delta, r), T_i(\delta, r)]$ holds if and only if there exists a rearrangement $\langle q_1 = r, \dots, q_i\rangle$ of S_{1i} with cost at most δ. We also use δ as a variable of two boundary points $b_i(\delta)$ and $t_i(\delta)$ of $\ell \cap D_i(\delta)$ with $b_i(\delta) \le t_i(\delta)$. We compute $R_n(\delta, r)$ by storing all different combinatorial structures over δ in increasing order of δ.

Lemma 11 *The combinatorial structure of $R_n(\delta, r)$ changes $O(n)$ times over δ.*

Proof. We prove that $\arg\max_i\{b_i(\delta) + i\}$ changes at most $O(n)$ times for δ increasing from 0 to ∞. For any two indices i and j, $b_i(\delta) + i = b_j(\delta) + j$ holds for at most one δ value. Therefore, if $\arg\max_i\{b_i(\delta) + i\}$ changes from j to j' at δ', $j = \arg\max_i\{b_i(\delta) + i\}$ does not hold for $\delta > \delta'$. This implies that each index becomes $\arg\max_i\{b_i(\delta)+i\}$ for at most one interval, which bounds the number of changes to $O(n)$ in total. We can bound the numbers of changes of $\arg\min_i\{t_i(\delta) - i\}$, $\arg\max_i\{b_i(\delta) - i\}$, and $\arg\min_i\{t_i(\delta) + i\}$ in the same way. $\square$

Let $B_{ij}(\delta, r)$ and $T_{ij}(\delta, r)$ denote the two boundary points of the feasible range $R_{ij}(\delta, r) = [B_{ij}(\delta, r), T_{ij}(\delta, r)]$ of p_j with respect to S_{ij} with p_i rearranged to r. We can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ using the feasible ranges $[B_{1k}(\delta, r), T_{1k}(\delta, r)]$ and $[B_{kn}(\delta, r), T_{kn}(\delta, r)]$ of two subsequences S_{1k} and S_{kn} of S for any k with $1 < k < n$ in $O(n)$ time.

Lemma 12 *We can compute $R_n(\delta, r)$ of S in $O(n)$ time once we have $R_{1k}(\delta, r)$, and $R_{kn}(\delta, r)$ for any k with $1 < k < n$.*

Proof. The functions $R_{1k}(\delta, r)$ and $R_{kn}(\delta, r)$ consist of $O(k)$ and $O(n - k)$ combinatorial structures, respectively, by Lemma 11. As they are stored in the increasing order of δ, we simply merge the functions to compute $R_n(\delta, r)$ in increasing order of δ. For each range where the combinatorial structures of $R_{1k}(\delta, r)$ and $R_{kn}(\delta, r)$ remain the same, we can compute $R_n(\delta, r)$ in $O(1)$ time by Lemma 10. As there are $O(n)$ such ranges, the algorithm takes $O(n)$ time in total. $\square$

By Lemma 12, we can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ of S in $O(n \log n)$ time applying divide and conquer.

Lemma 13 *We can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ in $O(n \log n)$ time.*

Optimization algorithm using parametric search. By Lemma 13, we have an $O(n \log n)$-time algorithm for computing an optimal rearrangement as follows: Compute $R_n(\delta, r) = [B_n(\delta, r), T_n(\delta, r)]$ in $O(n \log n)$ time, and find δ^* which is the minimum value of δ satisfying $\mathsf{dom}_n \neq \emptyset$. Using $O(n)$ additional time, the algorithm computes an optimal rearrangement by Lemma 2. Here, we present an algorithm using Cole's parametric search [7] to further improve the running time to $O(n \log\log n)$ time.

Before applying parametric search, our algorithm preprocesses the feasible ranges of subsequences. The algorithm subdivides S into $\lceil n/t\rceil$ subsequences, each consisting of at most $t + 1$ points for some parameter t, which will be chosen later. Every two consecutive subsequences share a point and S is entirely covered by the subsequences. For each subsequence, the algorithm computes the feasible range over all δ values. This procedure takes $O(t \log t)$ time for each subsequence, and thus it takes $O(n \log t)$ time in total.

After the preprocessing, the algorithm determines (sequentially) whether $\delta \geq \delta^*$ as follows. It finds the combinatorial structure of the feasible range of each subsequence with respect to δ using binary search. This takes $O(\log t)$ time for each subsequence, and thus it takes $O((n \log t)/t)$ time in total. Then the algorithm merges the feasible ranges in the sequential order to determine whether $\mathsf{dom}_n = \emptyset$ for δ. This takes $O(1)$ time for merging two feasible ranges in the order, and thus it takes $O(n/t)$ time in total, which is dominated by $O((n \log t)/t)$ time.

With the preprocessing, we present a parallel decision algorithm using $O(n/t)$ processors. The algorithm finds the combinatorial structure of the feasible range with respect to δ by assigning a processor for each subsequence. Then for the remaining steps, the algorithm merges two consecutive feasible ranges using a processor. After $O(\log n)$ merge steps, the algorithm computes $R_n(\delta, r)$. This procedure takes $O(1)$ time for each of $O(\log n)$ steps, after finding the combinatorial structure of feasible range for each subsequence in $O(\log t)$ time. Thus, it takes $O(\log n)$ time in total. As each process of the parallel algorithm depends on at most two other processes, we can apply Cole's parametric search [7].

By applying parametric search, the algorithm can compute the optimal rearrangement cost δ^* in $O(PT_P + T_S(T_P + \log P)) = O((n \log n \log t)/t)$ time after $O(n \log t)$-time preprocessing, where $P = O(n/t)$ is the number of processors needed for parallel decision, T_P is the running time of our parallel decision algorithm, and T_S is the running time of our sequential decision algorithm. Setting $t = \log n$, we obtain an $O(n \log \log n)$-time algorithm. After finding δ^* in $O(n \log \log n)$ time, we can find an optimal rearrangement of S using $O(n)$ additional time by Lemma 2.

Theorem 14 *For a sequence S of n points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in $O(n \log \log n)$ time.*

3.3 Weighted version

In this section, we show that the deterministic algorithm of Section 3.2 can be extended to the weighted version without increasing the time complexity. Observe that Lemma 7 also holds for the weighted version, by replacing the definition of $R_i(r) = [B_i(r), T_i(r)]$ for fixed δ_0 with $R_i(r) = (R_{i-1}(r) \oplus I(w_i - w_{i-1})) \cap D_i(\delta_0)$ ($R_1(r)$ remains the same). Then we can show that $R_i(r)$ is determined by at most four points of S with indices (a) $\arg\max_i\{b_i + w_i\}$, (b) $\arg\min_i\{t_i - w_i\}$, (c) $\arg\max_i\{b_i - w_i\}$, and (d) $\arg\min_i\{t_i + w_i\}$ as shown in Lemmas 8 and 9. Observe that the points determining $R_i(\delta, r)$ change $O(n)$ times over δ as in Lemma 11, we obtain $O(n \log \log n)$-time algorithm using parametric search.

Theorem 15 *For a sequence S of n weighted points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in $O(n \log \log n)$ time.*

4 Rearrangement onto a Line with Fixed Orientation

Recall that $\delta^*(\ell)$ denotes the cost of an optimal rearrangement of S onto a line ℓ. Given a sequence $S = \langle p_1, \ldots, p_n \rangle$ of n points and an orientation $\vec{c}$, we find a line ℓ such that $\delta^*(\ell)$ is minimum among all lines parallel to $\vec{c}$ in the plane, and compute an optimal rearrangement of S onto ℓ.

Without loss of generality, we assume that the orientation is horizontal, and thus our target line is horizontal. For a real value h, we use $\ell(h)$ to denote a horizontal line with $y = h$.

We compute an optimal rearrangement using a function that represents the optimal rearrangement cost of S onto all horizontal lines. In doing so, we first compute a convex function for a contiguous subsequence of S that partially describes the optimal rearrangement cost of the subsequence. We do this for $O(n)$ contiguous subsequences of S so that the upper envelope of those functions coincides with the function of the optimal rearrangement cost of S. By applying convex programming on the upper envelopes of disjoint subsets of functions, we can find a horizontal line ℓ such that $\delta^*(\ell)$ is minimum among all horizontal lines, and get the optimal rearrangement cost of S onto ℓ.

We abuse the function δ^* so that for a real value h, $\delta^*(h) = \delta^*(\ell(h))$ denotes the optimal rearrangement cost of S onto $\ell(h)$, as defined in Section 3. Then our goal is to minimize $\delta^*(h)$ over all $h \in \mathbb{R}$. Let h^* denote a real value such that $\delta^*(h^*) = \min_h \delta^*(h)$, and let $\delta^* := \delta^*(h^*)$ denote the optimal cost over all horizontal lines. Note that once we know h^*, we can find an optimal rearrangement of S onto $\ell(h^*)$ with cost δ^* in $O(n)$ time by Lemma 2.

Let us define two distance functions, $\delta_{ij}(h)$ and $\sigma_{ij}(h)$, for two indices i, j with $1 \leq i \leq j \leq n$ with respect to a horizontal line $\ell(h)$ for a real value h as follows.

Recall the definition of δ_{ij} from Section 3. We define a function $\delta_{ij}(h)$ of a real value h with respect to $\ell(h)$ for every two indices $i \leq j$ similarly. Let q_i and q_j be the points on $\ell(h)$ minimizing $\max\{\|p_i - q_i\|, \|p_j - q_j\|\}$ with $\|q_i - q_j\| \leq j - i$. Then $\delta_{ij}(h) = \max\{\|p_i - q_i\|, \|p_j - q_j\|\}$. By Lemma 4, $\delta^*(h) = \max_{i,j} \delta_{ij}(h)$ for any fixed h. Let $d(p, s) = \min_{q \in s} \|p - q\|$ denote the distance from a point p to a line segment s. For a point $p_i \in S$ and an index

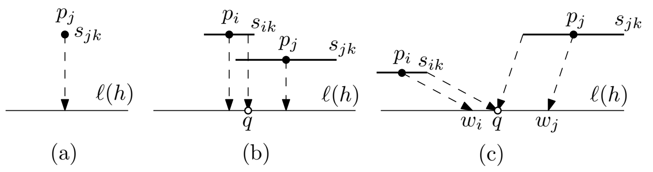


Figure 6: (a) If $i = j$, $\delta_{ij}(h)$ is the distance from p_j to $\ell(h)$. (b) If $|x(p_i) - x(p_j)| \le j - i$, $\max\{d(q, s_{ik}), d(q, s_{jk})\} = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ for any point $q \in \ell(h)$ such that the vertical line through q intersects both s_{ik} and s_{jk}. (c) If $|x(p_i) - x(p_j)| > j - i$, we can find a point q on $\ell(h)$ such that $\delta_{ij}(h) = \max\{d(q, s_{ik}), d(q, s_{jk})\}$.

k, let $s_{ik} = p_i \oplus I(|i - k|)$ denote the horizontal line segment of length $2 \cdot |i - k|$ with midpoint p_i. In case $i = k$, s_{ik} is a degenerate line segment with length 0.

Lemma 16 $\delta_{ij}(h) = \min_{q\in\ell(h)} \max\{d(q, s_{ik}), d(q, s_{jk})\}$ *for any index k with $i \le k \le j$,*

Proof. If $i = j$, the lemma holds obviously as both sides denote the distance from p_j to $\ell(h)$ (Figure 6(a)).

Consider the case that $i < j$. If $|x(p_i) - x(p_j)| \le j - i$, we have $\delta_{ij}(h) = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ by definition. Moreover, there is a point $q \in \ell(h)$ such that the vertical line through q intersects both s_{ik} and s_{jk}. For any such point q, we have $\max\{d(q, s_{ik}), d(q, s_{jk})\} = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ (Figure 6(b)).

If $|x(p_i) - x(p_j)| > j - i$, there are two points $u_i, u_j \in \ell(h)$ such that $\|u_i - u_j\| = j - i$ and $\delta_{ij}(h) = \max\{\|p_i - u_i\|, \|p_j - u_j\|\}$ by the definition of $\delta_{ij}(h)$. Let q be the point on $\ell(h)$ such that $\|q - u_i\| = k - i$ and $\|q - u_j\| = j - k$. Then $\|p_i - u_i\| = d(q, s_{ik})$ and $\|p_j - u_j\| = d(q, s_{jk})$, and for any point $r \in \ell(h)$, either $d(r, s_{ik}) \ge d(q, s_{ik})$ or $d(r, s_{jk}) \ge d(q, s_{jk})$. Therefore, we conclude that $\delta_{ij}(h) = \max\{d(q, s_{ik}), d(q, s_{jk})\}$ (Figure 6(c)). □

We now define another function $\sigma_{ij}(h)$ for two indices i, j with $1 \le i \le j \le n$ as follows. Let $\mathsf{far}(p, A) = \max_{l\in A} d(p, l)$ for a point $p \in \mathbb{R}^2$ and a set A of horizontal line segments. Let $\mathsf{L}_{ab} = \{s_{ta} \mid a \le t \le b\}$ and $\mathsf{R}_{ab} = \{s_{tb} \mid a \le t \le b\}$ be sets of segments.

$$\sigma_{ij}(h) = \min_{q\in\ell(h)} \mathsf{far}(q, \mathsf{R}_{ik} \cup \mathsf{L}_{kj}), \text{ for } k = \lceil (i + j)/2 \rceil.$$

We now show that $\sigma_{ij}(h)$ is a convex function consisting of $O(j - i)$ pieces of quadratic functions. Also, we show that there are $O(n)$ pairs of indices such that the upper envelope of $\sigma_{ij}(h)$'s for the pairs coincides with $\delta^*(h)$. The *farthest-site Voronoi diagram* for line segments in A, denoted by $\mathsf{Vor}(A)$ decomposes the plane into regions such that every point p in the same region has the same *farthest* line segment l among the line segments in A, that is, $d(p, l) = \mathsf{far}(p, A)$ for every point p in the region. By computing $\mathsf{Vor}(A)$ for a given set A, we identify a full description of $\mathsf{far}(p, A)$. It is known by Aurenhammer et al. [3] that $\mathsf{Vor}(A)$ consists of $O(|A|)$ vertices, edges, and cells. Using this property, we prove the following lemma.

Lemma 17 *$\sigma_{ij}(h)$ is a convex function consisting of $O(j - i)$ pieces of quadratic functions.*

Proof. Let $A = \mathsf{R}_{ik} \cup \mathsf{L}_{kj}$ with $k = \lceil (i + j)/2 \rceil$. Since $d(q, l)$ is a convex function of $q \in \mathbb{R}^2$ for any fixed line segment $l \in A$, $F(q) := \mathsf{far}(q, A)$ is also convex. Therefore, $\sigma_{ij}(h) = \min_{q\in\ell(h)} F(q)$ is convex. For a value $h \in \mathbb{R}$, let $F_h(q) := F|_{\ell(h)}(q)$ be a function of $q \in \ell(h)$, and $Q(h)$ be the set of points in $\ell(h)$ minimizing F_h. Then $\delta_{ij}(h) = F_h(q) = \mathsf{far}(q, A)$ for any $q \in Q(h)$. Since F_h is convex, $Q(h)$ forms a line segment on $\ell(h)$, possibly being degenerate to a point.

Let $l \in A$ be a farthest segment from $Q(h)$, which is a farthest segment from every point $q \in Q(h)$. Then $\sigma_{ij}(h) = d(q, l)$ for any $q \in Q(h)$. Note that there can be more than two farthest segments from $Q(h)$. We analyze $\sigma_{ij}(h)$ when the number of the farthest segments from $Q(h)$ is (1) one, (2) two, or (3) more than two.

(1) Let l_1 be the farthest segment of $Q(h)$. Then there is a point $q \in Q(h)$ lying in the interior of the cell of l_1 in $\mathsf{Vor}(A)$. Therefore, $\sigma_{ij}(h) = d(q, l_1)$ is the difference of y-coordinate of q and l_1, which is a linear function of h (Figure 7(a)). Note that l_1 is a segment with either the largest or the smallest y-coordinate value among the segments in A.

(2) Let l_2 and l_2 be the farthest segments of $Q(h)$. Then there must be a Voronoi edge defined by l_2 and l_3 in $\mathsf{Vor}(A)$, and $Q(h)$ is the intersection of $\ell(h)$ and the Voronoi edge. If the Voronoi edge is a parabolic curve, we can find a point $q \in \ell(h)$ that lies in the interior of the cell of l_2 or l_3 with $F(q) = \sigma_{ij}(h)$ as l_2 and l_3 are horizontal segments which makes one of two segments not farthest segment from $Q(h)$. Therefore, the Voronoi edge must be a line segment, which is the bisector of the endpoints of the segments. Thus, $\delta_{ij}(h)$ is a quadratic function of h (Figure 7(b)).

(3) Let l_1, l_2, and l_3 be three farthest segments of $Q(h)$. Then there must be a Voronoi vertex defined by the segments in $\mathsf{Vor}(A)$ and $Q(h)$ is the Voronoi vertex (Figure 7(c)).

There are at most two (unbounded) intervals of h with case (1). For the case (2), the farthest segments from $Q(h)$ do not change while increasing h until $\ell(h)$ hits a Voronoi vertex of $\mathsf{Vor}(A)$ to make the case (3). As there are $O(j - i)$ vertices of $\mathsf{Vor}(A)$, $\sigma_{ij}(h)$ consists of $O(j - i)$ partial quadratic functions. □

By Lemma 17, the algorithm can compute the function $\sigma_{ij}(h)$ from $\mathsf{Vor}(A)$ in $O(j - i)$ time as follows. The

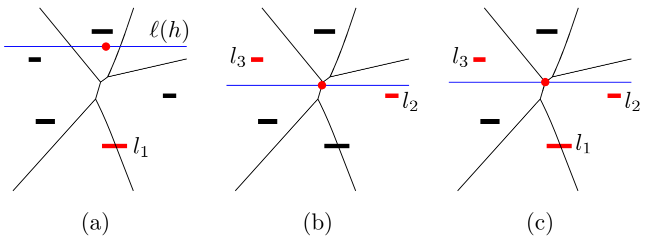


Figure 7: $q \in Q(h)$ is marked as a red dot. For each q, red segments are the farthest segments of A from q. (a) When there is one farthest segment, $\sigma_{ij}(h)$ is a linear function. (b) When there are two farthest segments, $\sigma_{ij}(h)$ is a quadratic function. (c) If there are more then two farthest segments, q lies on the Voronoi vertex of $\mathsf{Vor}(A)$.

algorithm first identify $\mathsf{far}(p, A)$ from $\mathsf{Vor}(A)$ in $O(j-i)$ time. Then it computes $\sigma_{ij}(h)$ from $\mathsf{far}(p, A)$ by keeping track of $Q(h)$ and the farthest segments of $Q(h)$ while increasing h from $-\infty$ to ∞. As the farthest segments of $Q(h)$ changes $O(j-i)$ times and each change can be identified from a Voronoi vertex of $\mathsf{Vor}(A)$ in $O(1)$ time, the algorithm takes $O(j-i)$ time in total.

Since $\mathsf{far}(q, A)$ is induced by $\mathsf{Vor}(A)$, $\sigma_{ij}(h)$ is the radius of the smallest disk intersecting every segment in $\mathsf{R}_{ik} \cup \mathsf{L}_{kj}$ whose center is restricted to lie on $\ell(h)$. If the center is determined by a single line segment s_{ak}, then $\sigma_{ij}(h) = \delta_{aa}(h)$. If the center is determined by two line segments, s_{ak} and s_{bk} with $a \le k \le b$, $\sigma_{ij}(h) = \delta_{ab}(h)$. We prove these relations between $\sigma_{ij}(h)$ and $\delta_{ab}(h)$ in Lemmas 18 and 19.

Lemma 18 *For any value h, there is a pair (i', j') with $i \le i' \le j' \le j$ such that $\sigma_{ij}(h) \le \delta_{i'j'}(h)$.*

Proof. There is a segment s_{ak} with $i \le a \le j$ such that $\min_{q\in\ell(h)} d(q, s_{ak}) = \sigma_{ij}(h)$, or there are two segments s_{ak}, s_{bk} with $i \le a < b \le j$ such that $\min_{q\in\ell(h)} \max\{d(q, s_{ak}), d(q, s_{bk})\} = \sigma_{ij}(h)$.

In the first case, $\sigma_{ij}(h) = \delta_{aa}(h)$. In the second case, let q^* be a point on $\ell(h)$ such that $\sigma_{ij}(h) = \max\{d(q^*, s_{ak}), d(q^*, s_{bk})\}$. Then $\sigma_{ij}(h) = \delta_{ab}(h)$ for $a \le k$ and $k \le b$. When $b < k$, $s_{ab} \subset s_{ak}$ and $s_{bb} \subset s_{bk}$. Therefore, $\sigma_{ij}(h) = \max\{d(q^*, s_{ak}), d(q^*, s_{bk})\} \le \max\{d(q^*, s_{ab}), d(q^*, s_{bb})\} = \delta_{ab}(h)$. We can show that $\sigma_{ij}(h) \le \delta_{ab}(h)$ for $k < a$ similarly. □

Lemma 19 *For any index pair (i', j') with $i \le i' \le \lceil (i+j)/2 \rceil \le j' \le j$, we have $\delta_{i'j'}(h) \le \sigma_{ij}(h)$.*

Proof. For any fixed index a with $i \le a \le j$ and $k = \lceil (i+j)/2 \rceil$, $d(q, s_{ak})$ for $q \in \ell(h)$ is convex. Therefore, $\sigma_{ij}(h) = \max_{i\le a\le b\le j} \min_{q\in\ell(h)} \mathsf{far}(q, \{s_{ak}, s_{bk}\})$. As $\min_{q\in\ell(h)} \mathsf{far}(q, \{s_{ak}, s_{bk}\}) = \delta_{ab}(h)$ for $i \le a \le k$ and $k \le b \le j$, the inequality holds. □

Data structures and algorithm. Let $\mathcal{T}$ be a binary tree such that the root corresponds to $S = S_{1n}$, each internal node v corresponds to a subsequence S_{ab} of S for some $a \le b$ with its left and right children corresponding to S_{ac} and S_{cb}, respectively, for $c = \lceil (a+b)/2 \rceil$. Each leaf node of $\mathcal{T}$ corresponds to a subsequence consisting of a single point, so $\mathcal{T}$ has $O(n)$ nodes and its height is $O(\log n)$. At each node v of $\mathcal{T}$ corresponding to S_{ab}, we store $\delta_v(h) = \sigma_{ab}(h)$. Then we show the following lemma.

Lemma 20 *For any value h, $\delta^*(h) = \max_{v\in\mathcal{T}} \delta_v(h)$.*

Proof. For a node v of $\mathcal{T}$, $\delta_v(h) \le \max_{1\le i\le j\le n} \delta_{ij}(h) = \delta^*(h)$ by Lemma 18. So we have $\max_{v\in\mathcal{T}} \delta_v(h) \le \delta^*(h)$.

We now show $\max_{v\in\mathcal{T}} \delta_v(h) \ge \delta^*(h)$. For any two indices i, j with $1 \le i \le j \le n$, let $v \in \mathcal{T}$ be the lowest node of $\mathcal{T}$ such that its corresponding subsequence S_{ab} contains both p_i and p_j, that is, $a \le i \le j \le b$. Note that $\delta_v(h) = \sigma_{ab}(h)$. Further, by our construction, $a \le i \le \lceil (a+b)/2 \rceil \le j \le b$. Then, by Lemma 19, $\delta_{ij}(h) \le \delta_v(h)$. Hence, $\delta^*(h) = \max_{i,j} \delta_{ij}(h) \le \max_v \delta_v(h)$. □

The algorithm computes $\delta_v(h)$ for nodes $v \in \mathcal{T}$ in bottom-up fashion. For each node v corresponding to S_{ab}, the algorithm stores $\mathsf{Vor}(\mathsf{R}_{ab})$ and $\mathsf{Vor}(\mathsf{L}_{ab})$. If v is a leaf node, the algorithm computes $\delta_v(h)$ in $O(1)$ time. If v is an internal node, the algorithm computes $\delta_v(h)$ in time linear to the length of the corresponding subsequence of v by using $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{L}_{cb})$ with $c = \lceil (a+b)/2 \rceil$ stored in the child nodes of v. The details on computing $\delta_v(h)$ for an internal node v are in the following.

Since every cell in $\mathsf{Vor}(A)$ is unbounded, there is a cyclic order of the cells of $\mathsf{Vor}(A)$ along a closed curve at infinity. The algorithm by Aurenhammer et al. [3] computes $\mathsf{Vor}(A)$ in two stages: finds out the cyclic order of the cells in $O(|A| \log |A|)$ time, and then constructs the diagram based on the order in additional $O(|A| \log |A|)$ time. Later, Khramtcova and Papadopoulou [13] showed that the second stage can be done in $O(|A|)$ time. From the two results, we have the following lemma.

Lemma 21 *Given $\mathsf{Vor}(A)$ and $\mathsf{Vor}(B)$ for two sets A and B of $O(n)$ line segments in total, we can compute $\mathsf{Vor}(A \cup B)$ in $O(n)$ time.*

Proof. We can compute the cyclic order of the cells of $\mathsf{Vor}(A)$ and the cyclic order of the cells of $\mathsf{Vor}(B)$ in $O(n)$ time. Then we can merge them into the cyclic order of cells of $\mathsf{Vor}(A \cup B)$ in $O(n)$ time using the algorithm by Aurenhammer et al. [3]. Finally, we can compute $\mathsf{Vor}(A \cup B)$ based on the order in $O(n)$ time [13]. □

Now we present an $O(n)$ time algorithm to compute $\mathsf{Vor}(\mathsf{R}_{ab})$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{R}_{cb})$ for $c = \lceil (a+b)/2 \rceil$. We first compute $\mathsf{Vor}(\mathsf{R}_{ac} \oplus I(b-c))$.

Lemma 22 *For a set A of horizontal line segments, the cyclic order of the cells of $\mathsf{Vor}(A)$ and the cyclic order of the cells of $\mathsf{Vor}(A \oplus I(r))$ are the same for any real value $r > 0$.*

Proof. For a direction $\vec{d}$, we use $C(\vec{d})$ to denote the cell of a farthest-site Voronoi diagram such that for any point p, there is a point q on the ray of direction $\vec{d}$ emanated from p such that $q \in C(\vec{d})$. For $\mathsf{Vor}(A)$, $C(\vec{d})$ is the cell of site $l \in A$ if and only if there exists an open half plane with inner normal vector $\vec{d}$ that intersects all the line segments of $A \setminus \{l\}$ but not l. Let h be an open half plane that intersects every line segment of $A \setminus \{l\}$. Let h' be the translate of h along the x-axis towards $\vec{d}$ by r. Then h' intersects every line segment of $(A \setminus \{l\}) \oplus I(r)$ but not $l \oplus I(r)$. Therefore, for any fixed direction $\vec{d}$, $\mathsf{Vor}(A)$ and $\mathsf{Vor}(A \oplus I(r))$ have the same site (line segment) defining $C(\vec{d})$ in their diagrams. □

Therefore, the algorithm can compute $\mathsf{Vor}(\mathsf{R}_{ac} \oplus I(b-c))$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ in linear time by Lemma 22. As $\mathsf{R}_{ab} = (\mathsf{R}_{ac} \oplus I(b-c)) \cup \mathsf{R}_{cb}$, the algorithm can also compute $\mathsf{Vor}(\mathsf{R}_{ab})$ in linear time by Lemma 21. Computing $\mathsf{Vor}(\mathsf{L}_{ab})$ can be done in linear time as well.

For an internal node $v \in \mathcal{T}$, the algorithm computes $\mathsf{Vor}(\mathsf{R}_{ac} \cup \mathsf{L}_{cb})$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{L}_{cb})$ in linear time by Lemma 21. Then the algorithm computes $\delta_v(h)$ from $\mathsf{Vor}(\mathsf{R}_{ac} \cup \mathsf{L}_{cb})$ in linear time by Lemma 17. Therefore, we can conclude the following lemma.

Lemma 23 *We can compute an explicit description of function $\delta_v(h)$ for all nodes v of $\mathcal{T}$ in $O(n \log n)$ time.*

Convex programming with $\delta_v(h)$'s. Recall that $\delta_v(h)$ for each node v is convex by Lemma 17. To find the lowest point on the function $\delta^*(h)$, the algorithm computes h^* and $\delta^*(h^*)$ using convex programming by taking $\delta_v(h)$ as a constraint for each v and $f(h) = h$ as the objective function. Chan [5] showed that the convex programming can be done by $O(k \log k)$ primitive operations, where k is the number of constraints and the primitive operations are (1) to find a point that optimizes the objective function while satisfying two constraints, or (2) to find intersections between a constraint and a line. Our problem consists of $O(n)$ constraints ($\delta_v(h)$'s) and the primitive operation takes $O(\log n)$ time as there are $O(n)$ nodes in $\mathcal{T}$ and $\delta_v(h)$ consists of $O(n)$ partial functions. Therefore, we obtain an $O(n \log^2 n)$-time algorithm to compute h^* and $\delta^*(h^*)$. However, as the complexity of $\delta_v(h)$ varies through the nodes of $\mathcal{T}$, we can reduce the number of constraints without increasing the time complexity of primitive operation as follows.

Improving time complexity. For a node u with corresponding subsequence of length L with $\lceil \log n \rceil \leq L < 2\lceil \log n \rceil$, we compute the upper envelope of $\delta_v(h)$ for nodes v in the subtree with root u. Observe that the upper envelope consists of $O(\log n \log\log n \cdot 2^{\alpha(n)})$ partial functions, where $\alpha(\cdot)$ is the inverse Ackermann function [16]. The bound comes from the fact that every partial function of $\delta_v(h)$ is a quadratic function by Lemma 17, so that any two partial functions intersect at most twice. Thus we can compute the upper envelope in $O(\log n \log^2 \log n \cdot 2^{\alpha(n)})$ time. There are $O(n/\log n)$ such nodes, and thus computing the upper envelopes for all such nodes takes $O(n \log^2 \log n \cdot 2^{\alpha(n)})$ time. Thus, we have $O(n/\log n)$ convex constraints, each consisting of $O(n)$ partial functions. Therefore, convex programming can be done in $O(n \log n)$ time. Since all $\delta_v(h)$'s can be computed in $O(n \log n)$ time by Lemma 23, h^* and $\delta^*(h^*)$ can be computed in $O(n \log n)$ time. Then the algorithm computes an optimal rearrangement of S onto $\ell(h^*)$ in $O(n)$ time by Lemma 2.

Theorem 24 *For a sequence S of n points and an orientation, we can compute an optimal rearrangement of S onto a line with the orientation in $O(n \log n)$ time.*

4.1 Weighted version

In this section, we show that the algorithm above can be extended to weighted version without increasing the time complexity. By changing the definition of s_{ik} to $p_i \oplus I(w_k - w_i)$, Lemmas 17 and 20 hold for the weighted version. Therefore, by Lemmas 21 and 22, we can compute $\delta_v(h)$ for each node v in $\mathcal{T}$ in time linear to the length of the corresponding subsequence in bottom-up fashion. By computing $\delta_v(h)$'s for nodes v in $\mathcal{T}$, we obtain the following theorem.

Theorem 25 *For a sequence S of n weighted points and an orientation, we can compute an optimal rearrangement of S onto a line with the orientation in $O(n \log n)$ time.*

5 Rearrangement into an Arbitrary Line

Given a sequence of $S = \langle p_1, \ldots, p_n \rangle$ of n points in the plane, we find a line ℓ such that the optimal rearrangement cost $\delta^*(\ell)$ of S onto ℓ is minimum among all lines in the plane, and compute an optimal rearrangement of S onto ℓ.

The detailed algorithms can be found in the full version of this paper.

Theorem 26 *For a sequence S of n points, we can compute an optimal rearrangement of S onto any line in $O(n^3 \operatorname{polylog} n)$ time.*

Theorem 27 *For a sequence S of n weighted points, we can compute an optimal rearrangement of S onto any line in $O(n^3 \operatorname{polylog} n)$ time.*

6 Conclusion

We consider the problem of finding an optimal rearrangement of sequence of n points onto a line. We present an expected $O(n)$-time algorithm and deterministic $O(n \log \log n)$-time algorithm for a given line. When the rearrangement line is given only by its orientation or not specfied at all, we present $O(n \log n)$-time and $O(n^3 \operatorname{polylog} n)$-time algorithm, respectively.

Our algorithms are described for solving the rearrangement problems under the Euclidean metric, but they can be applied to the cases that the underlying metric is L_p or has the distance function of constant complexity.

There are few work to study. One is to improve the deterministic time complexity to linear for a given rearrangement line, or to show a tight time bound on the problem. Another one is to study the optimal rearrangement problem for more general objects.

References

[1] P. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Computational Geometry*, 16:317–337, 1996.

[2] P. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *Journal of Algorithms*, 17(3):292–318, 1994.

[3] F. Aurenhammer, R. Drysdale, and H. Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006.

[4] R. Bellman and R. Roth. Curve fitting by segmented straight lines. *Journal of the American Statistical Association*, 64(327):1079–1084, 1969.

[5] T. M. Chan. Deterministic algorithms for 2-d convex programming and 3-d online linear programming. *Journal of Algorithms*, 27(1):147–166, 1998.

[6] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22:547–567, 1999.

[7] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, Jan. 1987.

[8] M. de Berg, P. Bose, D. Bremner, S. Ramaswami, and G. Wilfong. Computing constrained minimum-width annuli of point sets. *Computer-Aided Design*, 30(4):267–275, 1998. Computational Geometry and Computer-Aided Design and Manufacturing.

[9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[10] C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 1997, page 121–130, USA, 1997. Society for Industrial and Applied Mathematics.

[11] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[12] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

[13] E. Khramtcova and E. Papadopoulou. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In *Proceedings of the 26th International Symposium on Algorithms and Computation*, ISAAC 2015, pages 404–414, 2015.

[14] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1982.

[15] F. P. Preparata. New parallel-sorting schemes. *IEEE Transactions on Computers*, C-27(7):669–673, 1978.

[16] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, USA, 2010.

[17] L. G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[18] H. Wang and J. Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry & Applications*, 26(03n04):185–210, 2016.

Mapping Points to the Grid with Bounded Hausdorff Distance

Maarten Löffler[*] Jérôme Urhausen[*]

Abstract

We consider the problem of representing a set of m points using disjoint pixels on a grid with bounded Hausdorff distance. We prove that optimizing the problem is NP-complete. Additionally, we present a constant factor approximation algorithm with running time in $O(m^2 \log \delta^* / \log m)$, where δ^* is the Hausdorff distance in an optimal solution, as well as a slower algorithm with a constant additive error.

1 Introduction

The field of *digital geometry* concerns itself with the representation of geometric objects using pixels on a grid while preserving geometric properties. Examples are mapping convex regions to a similar-looking orthoconvex set of pixels or mapping lines to chains of pixels that still only intersect at most once. Digital geometry finds application in image processing and storage. For a survey, see Klette and Rosenfeld [15, 16].

More recently, error bounds under the Hausdorff distance have been studied. Chun et al. [7] investigate the problem of digitizing rays originating in the origin to digital rays such that certain properties are satisfied. They show that rays can be represented on the $n \times n$ grid in a consistent manner with Hausdorff distance $O(\log n)$. This bound is tight in the worst case. By ignoring one of the consistency conditions, the distance bound improves to $O(1)$. Their research is extended by Christ et al. [5] to line segments (not necessarily starting in the origin), who obtain the logarithmic distance bound in this case as well. A possible extension to curved rays was developed by Chun et al. [6]. Other results with a digital geometry flavor within the algorithms community are those on snap rounding [8, 11, 14], integer hulls [1, 13], and discrete schematization [17].

The present submission is inspired by two recent papers: *Mapping Polygons to the Grid with Small Hausdorff and Fréchet Distance*, by Bouts, Kostitsyna, van Kreveld, Meulemans, Sonke and Verbeek [3] and *Mapping Multiple Regions to the Grid with Bounded Hausdorff Distance* by van der Hoog, van de Kerkhof, van Kreveld, Löffler, Staals, Urhausen and Vermeulen [20]. Intuitively the problem discussed in these papers is: for a given set of regions find a set of pixels from the unit grid that best represents the input (see Figure 1).

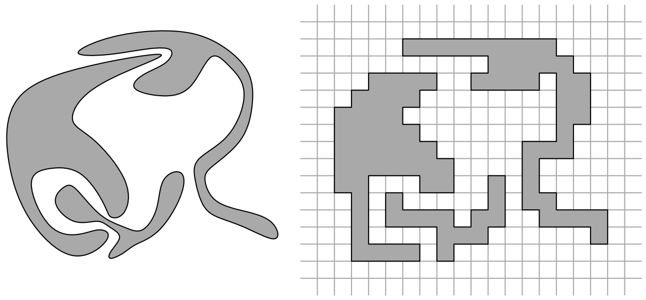


Figure 1: An example of a valid mapping of regions to grid polygons.

In the following we use the notation $H'(R, P)$ for the maximum between the Hausdorff distance between the sets R and P and the Hausdorff distance between their boundaries ∂R and ∂P. Amongst others, Bouts et al. [3] show that for a given connected region R, one can find a simply connected grid polygon P in the unit grid such that the Hausdorff distance $H'(R, P)$ is at most a constant. Note that this result holds, no matter the resolution of R. On the other hand they show that, given a region R, it is NP-hard to find the grid polygon P that minimizes the Hausdorff distance $H'(R, P)$.

Van der Hoog et al. [20] extend this concept to multiple regions. Whereas the result from [3] was extended to two regions, for three or more regions there is no constant upper bound on the Hausdorff distance between the regions and any simply connected grid polygons. Nonetheless, they show that, if the regions are m β-fat convex regions, one can construct a set of disjoint grid polygons within Hausdorff distance H' at most $O(\sqrt{m})$, for β constant. This is tight in the worst case. Note that points are β-fat convex regions, for any β. Their last result is that for m convex regions, one can construct a set of orthoconvex disjoint grid polygons within Hausdorff distance of $O(m)$, which is again tight in the worst case.

Computation. Previous work focuses on the *existence* of solutions with bounded Hausdorff distance, but not on their *efficient computation.* When considering the question of efficiency, we are faced with some additional modeling questions. The two main ones are:

[*]Department of Information and Computing Sciences, Utrecht University, {m.loffler|j.e.urhausen}@uu.nl. This work is partially supported by the Dutch Research Council (NWO) under project number 612.001.651, 614.001.504 and 628.011.005.

1. How do we locate input features (vertices or edges) on the grid?
2. How do we compactly represent sets of pixels that correspond to output regions?

Regarding question (1), we note that traditionally, geometric algorithms are analysed in the *Real RAM* computation model. In this model, one may work with arbitrary real numbers, but certain natural operations are not available; in particular the *floor* operation is known to be problematic [2]. In the context of digital geometry, where we have a natural underlying grid, and the whole objective is to map objects to a grid, such a restriction seems not entirely reasonable. In this work, we will move away from the Real RAM model and assume that the input coordinates are all polynomial in the input size—in other words, the *bit complexity* is logarithmic—and that the floor function is available. (Note that under our bit complexity assumption, if desired, the floor function can also be implemented in logarithmic time on a Real RAM.)

Regarding question (2), we note that when mapping large regions (in size, not in description complexity) to a grid, an explicit representation of the output listing precisely which pixels are and which are not part of a set would necessarily be large as well, and may be unrelated to the input complexity. Alternatively, one could compactly represent output regions by providing only the coordinates of *vertices* and interpolating boundary edges onto the grid. Given the complexity of mapping lines to the grid, however, it is not entirely clear how to do this in a consistent way. In this work, we make a first step towards understanding the computational aspect of the question by focusing on *point* regions. For a single point region and constant Hausdorff distance, the output is necessarily a set of only a constant number of pixels, and thus we avoid the issue.

To summarize, in the present submission, we make the following assumptions:

- The input is a set $\mathcal{R}$ of m points in $\mathbb{R}^2$, with a polynomial upper bound on the coordinate sizes; that is, for every point $R \in \mathcal{R}$ we have $-f(m) < x_R < f(m)$ and $-f(m) < y_R < f(m)$ for some polynomial function f.
- We have access to a *floor* operation, which can provide us with the integer part of any real number in the range $[-f(m), f(m)]$.

Related Work. Testing whether there exists a set of pixels within Hausdorff distance δ from $\mathcal{R}$ can be seen as a problem where we are given a set of squares of (L_1)-radius δ, and are asked to place a set of unit squares such that each square touches an input square. The problem of placing unit squares in the neighbourhood of points can be viewed as dual to the problem of placing points in squares: if we shrink the (L_1)-radius of the squares-to-be-placed by $\frac{1}{2}$ and we grow the δ-neighbourhoods of the points also by $\frac{1}{2}$, a valid solution where points are placed at integer coordinates at distance at least 2 from each other corresponds exactly to a valid solution to our problem.

The problem of placing points in squares such that the points are not too close to each other has been introduced under the name of *distant representatives* [10] and was later also studied in the context of data *imprecision* [18]. Fiala et al. [10] prove that the problem is NP-hard (both for disk and square regions), and Cabello [4] proposes a constant-factor approximation algorithm.

We note that our problem is essentially different, since for us, valid placements are restricted to a *discrete* set of points (the unit grid). Neither the hardness proof nor the algorithmic result carry over directly to this discrete setting.

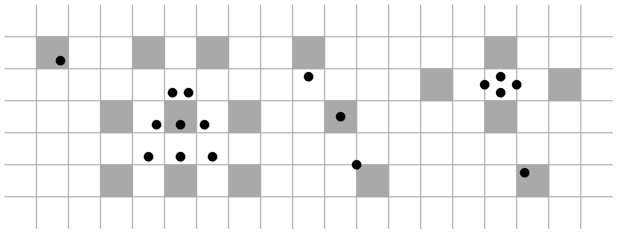


Figure 2: An example of a valid mapping of points to pixels on the grid.

Contribution. In the present submission, we study the computational question of mapping point sets to disjoint pixels on a unit grid with small Hausdorff distance, as visualized in Figure 2. Van der Hoog et al. [20] observed that in general the solution constructed with their algorithms might yield a "visually unfortunate" output. Formally, the algorithm might yield a solution with high Hausdorff distance, even in the case where the optimal solution has constant Hausdorff distance. In Section 2, we show that finding the solution with minimal Hausdorff distance to a given set of regions is NP-complete, even if the regions are just points. Then, in Section 3, we present an approximation algorithm for points that produces a solution with Hausdorff distance at most $2\sqrt{2}(\lceil\delta^*\rceil + 1) \leq 6\sqrt{2}\delta^*$ and has a running time of $O(m^2 \log \delta^* / \log m)$, where δ^* is the maximal Hausdorff distance in an optimal solution. Finally, we present a second algorithm which produces a solution with Hausdorff distance at most $\lceil\delta^*\rceil + \sqrt{2}$ in $O(\delta^{*4} m^2 / \log m)$ time.

Notation and Definitions. We denote by Γ the (infinite) unit grid in two dimensions, whose unit squares are referred to as *pixels*. The *(symmetric) Hausdorff distance* between two sets $A, B \subset \mathbb{R}^2$ is defined as $H(A,B) = \max\{\max_{a\in A}(\min_{b\in B}(|ab|)), \max_{b\in B}(\min_{a\in A}(|ab|))\}$, where $|ab|$ is the distance between the points a and b.

Let $\mathcal{R} = \{R_1, R_2, \dots R_m\}$ be a set of m points in the plane. In this paper, we treat the problem on how to assign a pixel $P_i \in \Gamma$ to each point $R_i \in \mathcal{R}$ such that different pixels do not meet in any edge or vertex of the grid. This is consistent with the problem definition from [20]. We call the set $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ of such pixels a *valid mapping* for $\mathcal{R}$. Our goal is to find a valid mapping $\mathcal{P}$ that minimizes $\max_{i\in\{1,\dots,m\}}\{H(R_i, P_i)\}$. See Figure 2 for an example.

Note that in contrast to [3] and [20], we disregard the Hausdorff distance between the boundaries $H(\partial R_i, \partial P_i)$ because we have $H(\partial R_i, \partial P_i) = H(R_i, P_i)$, for convex R_i and P_i.

2 NP-completeness

Bouts et al. [3] proved that for a single simply connected region R it is NP-complete to test if there is a grid-polygon within Hausdorff distance 1/2. We extend this result to multiple point-regions. Formally, we show that for a set of points $\mathcal{R}$ it is NP-complete to test if there is a valid mapping within Hausdorff distance $\sqrt{2}$. Our proof is inspired both by the construction of Bouts et al. [3] and the proof by Fiala et al. [10] for a similar problem in a continuous setting.

We first prove containment in NP. For each point there are at most 9 options to place the corresponding pixel. An oracle can guess the correct placement and then just has to test that no two pixels share a common grid vertex.

We now show that the problem is NP-hard. We reduce from the NP-complete problem monotone rectilinear planar 3-Sat [9].

Rectilinear monotone planar 3-SAT. Input: a 3-Sat formula with only all positive or all negated variables per clause, embedded as a graph with rectilinear, non-crossing edges. The set of vertices consists of variable-, split- and clause-vertices; variable-vertices are drawn on a horizontal line that no edge crosses; clause-vertices for positive (negative) clauses are drawn above (below) this line; clauses are connected with the variables they contain with an edge or a path of edges and split-vertices. Output: "Yes" if there exists a satisfying assignment for the variables, "No" otherwise.

Such a 3-Sat formula embedded as a graph is illustrated in Figure 3. Without loss of generality we can assume that the embedded graph has the following additional properties: edges have at most one bend, each variable-vertex v has degree at most 2 and the incident edges are vertical at v, split-vertices w have degree 3 and only one incident edge is horizontal at w, and for each variable a, all split-vertices corresponding to a are vertically aligned with the variable-vertex v_a of a.

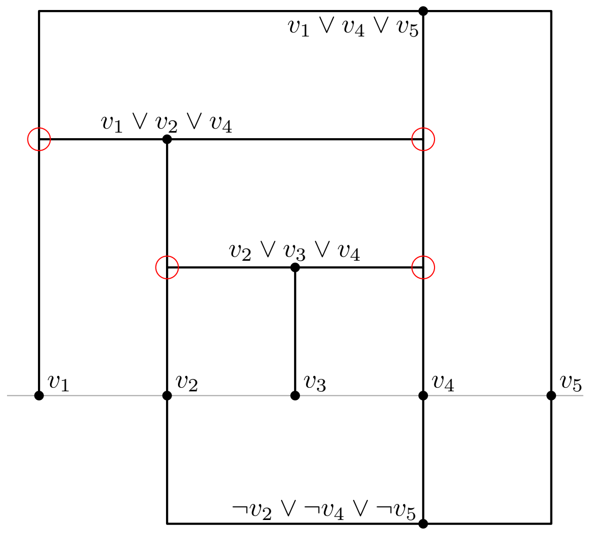
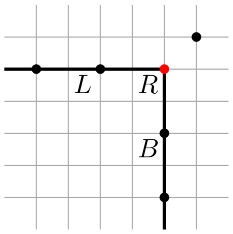


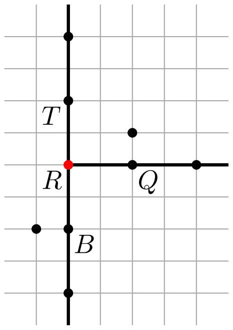
Figure 3: An example of the embedded formula $(v_2 \vee v_3 \vee v_4) \wedge (v_1 \vee v_2 \vee v_4) \wedge (v_1 \vee v_4 \vee v_5) \wedge (\neg v_2 \vee \neg v_4 \vee \neg v_5)$. The split vertices are highlighted in red.

For a given monotone rectilinear planar 3-Sat instance that is embedded as described above, let $\mathcal{G}$ be a drawing of the embedding. Without loss of generality we assume that $\mathcal{G}$ is drawn on the unit grid and that the horizontal line containing all variables is the x-axis. We scale $\mathcal{G}$ such that each vertex is on an even grid vertex $(2x, 2y)$ and such that the distance between any two vertices, between any vertex and any bend, and between any two non-incident edges is at least 8.

Construction. We create a set of points $\mathcal{R}$. We only place points on grid vertices. In the end, we ask the question whether one can place pixels within Hausdorff distance $\sqrt{2}$ from the points of $\mathcal{R}$, that is, we ask the question if for each point in $\mathcal{R}$, we can choose one of the four adjacent pixels so that no two chosen pixels of different points share a common vertex. We say a point R_i has a *top-left* (*top*, *top-right*, ...) *pixel* if P_i is to the top-left (top, top-right, ...) of R_i.

These following two observations are the main tools for the construction of the gadgets, depicted in Figure 4. When two horizontally aligned points are at distance 1, the leftmost (rightmost) point has a left (right) pixel. For two horizontally aligned points at distance 2, if the leftmost (rightmost) point has a right (left) pixel, the rightmost (leftmost) point has a right (left) pixel, too. This is symmetric for vertically aligned points.





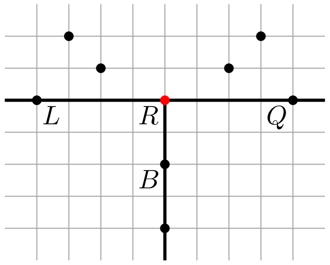


Figure 4: The bend, split and clause gadgets. The point R is highlighted in each gadget.

Variable. We first place a point $R \in \mathcal{R}$ on each even grid vertex that is intersected by the drawing. For each variable a in G, there is a point $R_a = (2x, 0)$ in $\mathcal{R}$ where the variable-vertex v_a is drawn. We call that point R_a the *indicator* of a. Intuitively, if R_a has a bottom (top) pixel, a is true (false). If the point $(2x, 2)$ has a bottom (top) pixel we say it has a pixel *toward the variable* (*away from the variable*). Symmetrically, if the point $(2x, -2)$ has a top (bottom) pixel we say it has a pixel *toward the variable* (*away from the variable*). This concept propagates throughout the points corresponding to a.

Bend. Let $R = (2x, 2y) \in \mathcal{R}$ be a point at the corner of a bend of an edge e. We assume e connects a vertex to the left of R with a vertex below R. The other cases are symmetric. Thus, the points $L = (2x - 2, 2y)$ and $B = (2x, 2y - 2)$ are in $\mathcal{R}$. We add another point $(2x + 1, 2y + 1)$ to $\mathcal{R}$. Now, if L has a right pixel, B has a bottom pixel and if B has a top pixel, L has a left pixel.

Split. Let $R = (2x, 2y) \in \mathcal{R}$ be a point at a split-vertex. We assume that the horizontal edge incident to R is to its right and that the split vertex is above the x-axis and therefore its corresponding variable-vertex. Thus, the points $T = (2x, 2y + 2)$, $Q = (2x + 2, 2y)$ and $B = (2x, 2y - 2)$ are in $\mathcal{R}$. The other cases are symmetric. We add the points $(2x + 2, 2y + 1)$ and $(2x - 1, 2y - 2)$ to $\mathcal{R}$. If T has a bottom pixel or if Q has a left pixel, B has a bottom pixel. That is, if a point on the top or right edges has a pixel toward the variable, the points on the bottom edge also have pixels toward the variable.

Clause. Let $R = (2x, 2y) \in \mathcal{R}$ be a point at a clause-vertex. We call R the *clause-point*. We assume the three edges connect to the left, right and bottom of R respectively, else the situation is symmetric. As the distance between two gadgets is least 8, the points $(2x-2, 2y)$, $L = (2x - 4, 2y)$, $(2x + 2, 2y)$, $Q = (2x + 4, 2y)$ and $B = (2x, 2y - 2)$ are in $\mathcal{R}$. We move the points $(2x - 2, 2y)$ and $(2x + 2, 2y)$ to $(2x - 2, 2y + 1)$ and $(2x + 2, 2y + 1)$ and add two points $(2x - 3, 2y + 2)$ and $(2x + 3, 2y + 2)$ to $\mathcal{R}$. It follows that if the clause-point R has a top-left (top-right, bottom) pixel, L (Q, B) has a left (right, bottom) pixel. That means that the points on at least one of the incident edges have pixels toward the variable.

Put together the points $\mathcal{R}$ and a valid mapping $\mathcal{P}$ are shown in Figure 5.

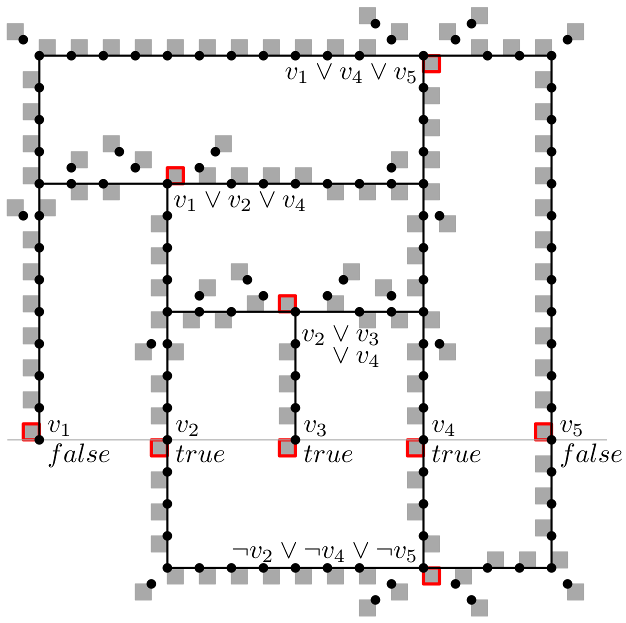


Figure 5: The complete construction of the NP-hardness reduction. The pixels corresponding to indicators or to clause-points are highlighted.

Proof of correctness. Let A be an assignment of variables to $\{true, false\}$ such that the 3-Sat formula is satisfied. For each variable a, if a is true, first, we give the indicator R_a a bottom pixel. Second, we give each point $R \in \mathcal{R}$ that is on an edge corresponding to a a pixel toward (away from) the variable, if p is above (below) the x-axis. For each variable assigned the value false we do the inverse. As in each positive (negative) clause there is a variable that is assigned to true (false), each clause-point can place its pixel in one of the four adjacent spots. So overall there is a valid mapping such that the Hausdorff distance between a point and its corresponding pixel is at most $\sqrt{2}$.

Inversely, let there be a set of pixels such that the Hausdorff distance between any point and its corresponding pixel is at most $\sqrt{2}$. For each variable a, if the indicator R_a has a bottom (top) pixel, we set a to true (false). We now prove that in each positive (negative) clause there is a variable that is assigned to true (false). Let c be a positive clause such that the incident edges are on the left, right and bottom of the clause point R of c. The other cases are symmetric. If R has a top-left (top-right, bottom) pixel, we know that the points on the left (right, bottom) edge have pixels toward the variable. Let e be an edge incident to R whose points have pixels toward the variable. If e connects to a split-vertex w, the points on the vertical edge e' incident at the bottom at w also have pixels toward the variable. We then set $e = e'$. This repeats until we have an edge e that connects to the variable-vertex of a. It follows that the indicator R_a has a bottom pixel and a has been assigned the value true. The theorem follows.

Theorem 1 *If $\mathcal{R}$ is a set of m points, it is NP-complete to decide whether there exists a valid mapping such that for each point $R_i \in \mathcal{R}$ with corresponding pixel P_i, we have $H(R_i, P_i) \leq \sqrt{2}$.*

3 Approximation Algorithms

We now turn our attention to approximation. We start by making some observations. Clearly, the optimal Hausdorff distance δ^* for any instance is at least $\frac{1}{2}\sqrt{2}$, since the distance is taken between a point and (at least one) unit square. Therefore, a constant *additive* approximation in this case automatically translates to a constant *multiplicative* approximation. We also note that, due to the discrete nature of the output, we cannot hope to do better than a constant factor approximation.

To illustrate the complexity of the problem, in Section 3.1 we first discuss some natural ideas which do *not* lead to a working approximation. Then, in Section 3.2, we then present an algorithm that achieves a Hausdorff distance of at most $2\sqrt{2}\lceil\delta^*\rceil + 2\sqrt{2}$. In Section 3.3 we show how to improve the approximation to $\lceil\delta^*\rceil + \sqrt{2}$, at the cost of a slower runtime.

3.1 A First Attempt

As discussed in the introduction, Cabello [4] presents a constant factor approximation algorithm for placing n points into respective discs or squares. A first approach could be to dualize our problem and directly run their algorithm to place a set of points—however, we would have no guarantee the points are placed at integer coordinates. We would have to snap the points to the grid before translating them back to squares. The Hausdorff distance itself would only increase by at most $\frac{1}{2}\sqrt{2}$ in this way, which would still result in a constant-factor approximation, albeit with a slightly higher constant. However, the snapping procedure could also result in touching or even overlapping pixels, so the solution would not necessarily be valid.

Another approach would be to find a subdivision of the grid into cells such that for each cell all the points contained in it can be assigned separate pixels in that cell. If the minimal size of the cells depends only on the Hausdorff distance between the points and an optimal valid mapping, this approach could lead to an approximation algorithm. Formally, let Γ_k be a coarsening of the grid Γ whose cells have $k \times k$ pixels. We call these cells *superpixels*. The following lemma proves that this approach does not work either.

Lemma 2 *There is a set of points $\mathcal{R}$ and a point $R \in \mathcal{R}$, such that (1) there is a valid mapping $\mathcal{P}$ within Hausdorff distance at most 3; (2) for any superpixel with side length s that contains R and at most $\lfloor \frac{s}{2} \rfloor^2$ points from $\mathcal{R}$, we have $s \in \Omega(|\mathcal{R}|)$.*

Proof. A pixel is a set $[i, i+1] \times [j, j+1]$, for integers i, j. We define the set of points $\mathcal{R} = \{R = (\frac{1}{4}, \frac{1}{4})\} \cup \{(2i + \frac{1}{2}, 2j + \frac{1}{2}), (2i + \frac{1}{2}, -2j - \frac{1}{2}), (-2i - \frac{1}{2}, -2j - \frac{1}{2}), (-2i - \frac{1}{2}, 2j + \frac{1}{2}) \mid i, j \in \{0, \ldots, n\}\}$, as shown in Figure 6. Let the four endpoints of a superpixel S containing R be (a, b), $(-c, b)$, $(-c, -d)$ and $(a, -d)$, with $a, b \geq 1; c, d \geq 0$. The side length of the superpixel is $s = a + c = b + d$. If $s \leq n$, the superpixel S contains $1 + \lceil a/2 \rceil \lceil b/2 \rceil + \lceil c/2 \rceil \lceil b/2 \rceil + \lceil c/2 \rceil \lceil d/2 \rceil + \lceil a/2 \rceil \lceil d/2 \rceil > \lfloor s/2 \rfloor^2$ points. □

3.2 A Constant Factor Approximation Algorithm

We present an algorithm that, for a given set of points $\mathcal{R}$ in $\mathbb{R}^2$ determines a valid mapping $\mathcal{P}$, such that the Hausdorff distance between a point and its corresponding pixel is at most $2\sqrt{2}(\lceil\delta^*\rceil + 1)$, for δ^* being the minimal possible Hausdorff distance between $\mathcal{R}$ and any valid mapping $\mathcal{P}$.

For a coarsening Γ_k of the grid Γ, let $\mathcal{S}_k$ be the set of superpixels that either contain a point in $\mathcal{R}$ or are adjacent to a superpixel that does.

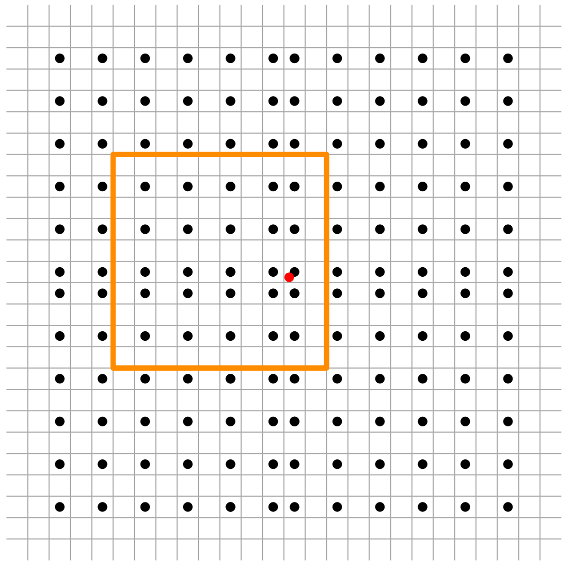


Figure 6: Any superpixel with side length i containing the red point R contains more than $\lfloor i/2 \rfloor^2$ points. For example, the yellow superpixel has side length 10 and contains 26 points.

Observation 1 *Let $x \geq \delta^*$ and $x \in \mathbb{N}$ even. Then we know that for each point $R_i \in \mathcal{R}$ in a superpixel $S \in \mathcal{S}_x$, the pixel P_i is in S or in one of the 8 superpixels adjacent to S. Additionally, each superpixel contains at most $(x/2)^2$ pixels that are disjoint.*

Building on that idea, we create the following test $f(\cdot)$. For an even number $i \in \mathbb{N}$, we want $f(i) =$ true if for each coarsening Γ_i, there exists an assignment $g : \mathcal{R} \to \mathcal{S}_i$ of points to superpixels such that:

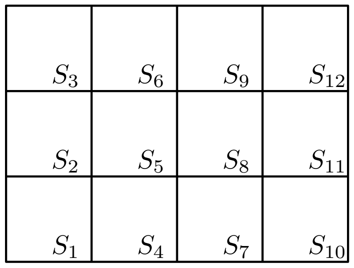
1. $\forall R \in \mathcal{R}$, $g(R)$ is the superpixel containing R or one of the eight adjacent ones;

2. $\forall S \in \mathcal{S}_i$, there are at most $(i/2)^2$ points R with $g(R) = S$.

We call such an assignment g a *correct assignment*. Otherwise, if for each coarsening Γ_i no correct assignment can be found, we want $f(i) =$ false. If a correct assignment g can only be found for some coarsenings, $f(i)$ can either be true or false. We define $f(0) =$ false.

Binary Search. We use exponential and binary search to find an even number $i \in \mathbb{N}$ with $f(i) =$ true and $f(i-2) =$ false. We start at $i = 2$. We iterate calculating $f(i)$: if $f(i) =$ false, we double i and continue, else we stop. Then we binary search normally. Note that from Observation 1 we get $f(i) = true$, for $i \geq \delta^*$. Therefore, this binary search algorithm results in a number $I \leq \lceil \delta^* \rceil + 1$ and has a running time of $O(F \times \log \delta^*)$, where F the the time to run the test $f(\cdot)$.

Test. The calculation of $f(i)$ proceeds as follows. We use a flow algorithm [12] to determine if a correct assignment g exists. We choose a coarsening Γ_i and create a directed acyclic graph $G = (V, E)$ as illustrated in Figure 7. We set $V = \{s, t\} \cup \{S_{in}, S_{out} \mid S \in \mathcal{S}_i\}$ as the set of vertices. We define $(a, b, c) \in E$ as the edge between the vertices $a \in V$ and $b \in V$ with capacity $c \in \mathbb{N} \cup \{\infty\}$. We set $E = \{(s, S_{in}, |S \cap \mathcal{R}|) \mid S \in \mathcal{S}_i\} \cup \{(S_{in}, S'_{out}, \infty) \mid S, S' \in \mathcal{S}_i \wedge (S = S' \vee S \text{ adjacent to } S')\} \cup \{(S_{out}, t, (i/2)^2) \mid S \in \mathcal{S}_i\}$ as the set of edges.



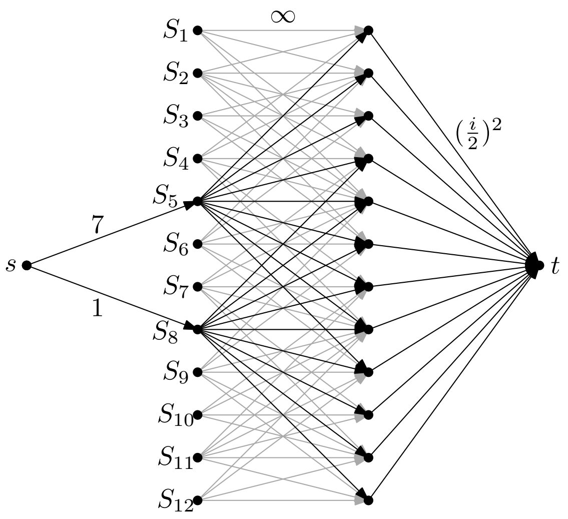


Figure 7: A set $\mathcal{R}$ of 8 points and the corresponding graph G produced by the algorithm in Section 3.2. The edges are annotated with their respective capacities. Edges with capacity 0 are omitted; edges starting at $(S_i)_{in}$ are grayed out if S_i is empty.

We then calculate a maximal flow from s to t. We can assume that the flow in each edge is a natural number. As $|E| \in O(|V|)$, the flow algorithm runs in $O(m^2/\log m)$ time [19] because $|V| \in O(m = |\mathcal{R}|)$. If G admits a flow from s to t with flow rate $|\mathcal{R}|$, the flow induces a correct assignment g as follows: we repeat the following for every superpixel $S \in \mathcal{S}_i$. Let $\{S_1, \ldots, S_9\} = \{S' \mid S' \in \mathcal{S}_i \wedge (S = S' \vee S \text{ adjacent to } S')\}$ be the set containing S and the superpixels adjacent to S. Let $U_1 \cup \cdots \cup U_9 = S \cap \mathcal{R}$ be any partition of the points in S, where, for $j, k \in \{1, \ldots, 9\}$, we have $j \neq k \implies U_j \cap U_k = \emptyset$ and $|U_j|$ is the flow from S_{in} to $(S_j)_{out}$. For each $j \in \{1, \ldots, 9\}$ and each point $R \in U_j$, we set $g(R) = S_j$. This results in a correct assignment g.

Thus, if G admits a flow from s to t with flow rate $|\mathcal{R}|$, we set $f(i) = \text{true}$. Otherwise $f(i) = \text{false}$. This test performs as requested and has a running time of $O(m^2/\log m)$.

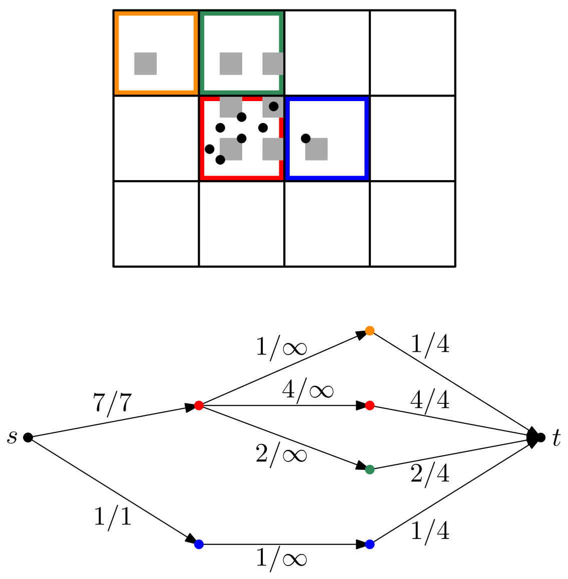


Figure 8: A set $\mathcal{R}$ of 8 points and the graph G with a maximal s-t-flow where edges with flow 0 are omitted, produced by the algorithm in Section 3.2. The superpixels and their respective vertices in G are color-coded. The pixels $\mathcal{P}$ induced by the flow are show in gray.

Placing the Pixels. Let I be the even number that is the result from the binary search, that is, $f(I) = \text{true}$ and $f(I-2) = \text{false}$. Let now Γ_I be a coarsening and let g be a correct assignment, calculated by the test $f(I)$. We place the pixels $\mathcal{P}$ as shown in Figure 8: for each superpixel $S \in \mathcal{S}_I$, let $\{R_1^S, R_2^S, \ldots\} = \{R \in \mathcal{R} \mid g(R) = S\}$ be the points assigned to S. We set the pixel corresponding to R_j^S as $\left(2\left(j \bmod \frac{I}{2}\right), 2\left\lceil \frac{2j}{I} \right\rceil\right)_S$, where $(1,1)_S$ ($(I,1)_S$, $(I,I)_S$) is the pixel on the bottom-left (bottom-right, top-right) of S. That way no two pixels in $\mathcal{P}$ touch and for each point R its corresponding pixel is in the superpixel assigned to R. It follows that the Hausdorff distance between a point and its corresponding pixel is at most $2\sqrt{2}I$. Since I is an even number, $I \leq \lceil \delta^* \rceil + 1$, and $\delta^* \geq \sqrt{(2)}/2$, we get:

Theorem 3 *Given a set of m points $\mathcal{R}$, we can determine a valid mapping $\mathcal{P}$ such that for each point R_i with corresponding pixel P_i, we have $H(R_i, P_i) \leq 2\sqrt{2}(\lceil \delta^* \rceil + 1) \leq 6\sqrt{2}\delta^*$ in $O(m^2 \log \delta^* / \log m)$ time.*

3.3 An Algorithm with Constant Additive Error

Contrary to the constant factor algorithm presented in the previous section, we now present an approximation algorithm with *only* a constant additive error: that is, for a given set $\mathcal{R}$, we determine a valid mapping $\mathcal{P}$, with $H(R_i, P_i) \leq \delta^* + c$ for each $R_i \in \mathcal{R}$, where c is a constant and δ^* is the minimal possible Hausdorff distance between $\mathcal{R}$ and any valid mapping $\mathcal{P}$. The algorithm uses similar ideas to the algorithm in Section 3.2. Let Γ_2 be a coarsening, that is each superpixel only contains 4 pixels forming a square. It follows that when placing a pixel in each superpixel, the pixels are disjoint. We define $H^*(R, S) = \min_{\text{pixel } P \in S} H(R, P)$ as the Hausdorff distance between the point $R \in \mathcal{R}$ and its closest pixel in the superpixel S. For $i \in \mathbb{N}$, let $\mathcal{S}_i$ be the set of superpixels S, such that there is a point $R \in \mathcal{R}$ with $H^*(R, S) \leq i$. Note that here, the size of the set $\mathcal{S}_i$ is dependent on i^2.

Observation 2 *Let $x \geq \delta^*$ and $x \in \mathbb{N}$. For a given valid mapping $\mathcal{P}$ that minimizes the Hausdorff distance, for each point $R_i \in \mathcal{R}$, the pixel $P_i \in \mathcal{P}$ is in a superpixel $S \in \mathcal{S}_x$, with $H^*(R_i, S) \leq x$. Additionally, each superpixel contains at most one pixel.*

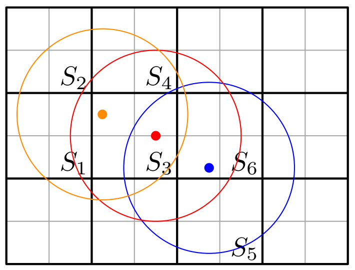
The observation leads us to create the following test $f(\cdot)$: we want $f(i) = \text{true}$, if there exists an assignment $g : \mathcal{R} \to \mathcal{S}_i$ of points to superpixels such that:

1. $\forall R \in \mathcal{R}$, $H^*(R, g(R)) \leq i$;
2. $\forall S \in \mathcal{S}_i$, there is at most one R with $g(R) = S$.

We again call such an assignment g a *correct assignment*. Otherwise, we want $f(i) = \text{false}$. Note that $f(0) = \text{false}$.

Binary Search. Similarly to Section 3.2, we use exponential and binary search to find a number $i \in \mathbb{N}$ with $f(i) = \text{true}$ and $f(i-1) = \text{false}$. We start at $i = 1$. We iterate calculating $f(i)$: if $f(i) = \text{false}$, we double i and continue, else we stop. Then we binary search normally. Due to Observation 2, we have $f(i) = \textit{true}$, for $i \geq \delta^*$. This binary search algorithm results in a number $I \leq \lceil \delta^* \rceil$ and has a running time of $O(F \times \log \delta^*)$, where F the the time to run the test $f(\cdot)$.

Test. The calculation of $f(i)$ proceeds very similarly to Section 3.2. We use a flow algorithm [12] to determine if a correct assignment g exists. We create a directed acyclic graph $G = (V, E)$, as illustrated in Figure 9. We set $V = \{s, t\} \cup \mathcal{R} \cup \mathcal{S}_i$ as the set of vertices. We define $(a, b) \in E$ as the edge between the vertices $a \in V$ and $b \in V$ with capacity 1. We set $E = \{(s, R) \mid R \in \mathcal{R}\} \cup \{(R, S) \mid R \in \mathcal{R}, S \in \mathcal{S}_i \wedge H^*(R, S) \leq i\} \cup \{(S, t) \mid S \in \mathcal{S}_i\}$ as the set of edges.



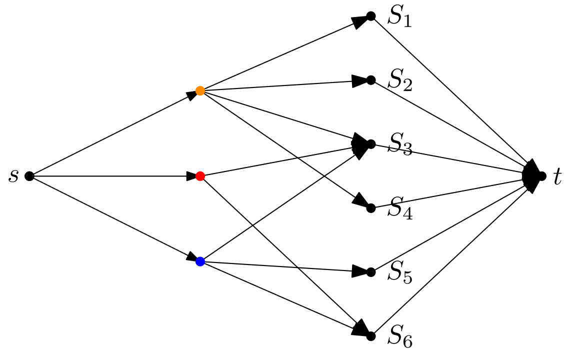


Figure 9: A small set $\mathcal{R}$ of 3 points and the graph $G = (V, E)$ produced by the algorithm in Section 3.3 for $i = 2$. All edges have a capcity of 1. The points in $\mathcal{R}$ and their respective vertices in G are color-coded.

We then calculate a maximal flow from s to t. We can assume that the flow in each edge is a natural number. As $|V|, |E| \in O(i^2 m)$, the flow algorithm runs in $O(i^4 m^2/(\log i \log m))$ time [19]. If G admits a flow from s to t with flow rate $|\mathcal{R}|$, the flow induces a correct assignment g as follows: for each point $R \in \mathcal{R}$, there is exactly one superpixel $S \in \mathcal{S}_i$ where the edge (R, S) has flow 1. We set $g(R) = S$. This results in a correct assignment g.

Thus, if G admits a flow from s to t with flow rate $|\mathcal{R}|$, we set $f(i) =$ true. Otherwise $f(i) =$ false. This test $f(i)$ performs as requested and has a running time of $O(i^4 m^2/(\log i \log m))$.

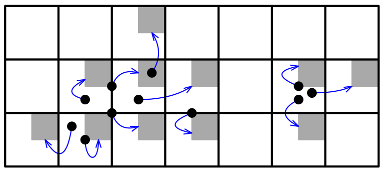


Figure 10: A set $\mathcal{R}$ of 11 points and the valid mapping $\mathcal{P}$ respecting a correct assignment g produced by the algorithm in Section 3.3.

Placing the Pixels. Let I be the number that is the result from the binary search, that is, $f(I) =$ true and $f(I-1) =$ false. Let g be a correct assignment, calculated by the test $f(I)$. We place the pixels $\mathcal{P}$ as shown in Figure 10: for each superpixel $S \in \mathcal{S}_I$, if there is a point R assigned to S, we place the pixel corresponding to R in the top-right pixel of S. That way no two pixels in $\mathcal{P}$ touch and for each point R its corresponding pixel is in the superpixel assigned to R. It follows that the Hausdorff distance between a point and its corresponding pixel is at most $I + \sqrt{2}$. Since $I \leq \lceil \delta^* \rceil$, we have:

Theorem 4 *Given a set of m points $\mathcal{R}$, we can determine a valid mapping $\mathcal{P}$ such that for each point R_i with corresponding pixel P_i, we have $H(R_i, P_i) \leq \lceil \delta^* \rceil + \sqrt{2}$ in $O({\delta^*}^4 m^2/\log m)$ time.*

4 Conclusion

Overall we extended the previously known results on mapping regions to the grid with bounded Hausdorff distance. Where Bouts et al. [3] showed that, for a given set of regions, it is NP-hard to find the set of grid polygons that minimizes the Hausdorff distance H', even if for just one region, we show that this is hard, even if all regions are points. On the other hand, where van der Hoog [20] focused on worst-case tight algorithms, we present the first approximation algorithm for mapping regions to the grid.

An interesting open question is whether the concepts presented in this paper can be extended to an approximation algorithm that maps convex regions to the grid.

References

[1] E. Althaus, F. Eisenbrand, S. Funke, and K. Mehlhorn. Point containment in the integer hull of a polyhedron. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 929–933, 2004.

[2] L. Babai, B. Just, and F. Meyer auf der Heide. On the limits of computations with the floor function. *Inform. Comput.*, 78(2):99–108, 1988.

[3] Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke, and K. Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *Proceedings 24th Annual European Symposium on Algorithms*, pages 22:1–22:16, 2016.

[4] S. Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007.

[5] T. Christ, D. Pálvölgyi, and M. Stojaković. Consistent digital line segments. *Discrete & Computational Geometry*, 47(4):691–710, 2012.

[6] J. Chun, K. Kikuchi, and T. Tokuyama. Consistent digital curved rays. In *Abstracts 34th European Workshop on Computational Geometry*, 2019.

[7] J. Chun, M. Korman, M. Nöllenburg, and T. Tokuyama. Consistent digital rays. *Discrete & Computational Geometry*, 42(3):359–378, 2009.

[8] M. de Berg, D. Halperin, and M. Overmars. An intersection-sensitive algorithm for snap rounding. *Computational Geometry*, 36(3):159–165, 2007.

[9] M. De Berg and A. Khosravi. Optimal binary space partitions for segments in the plane. *International Journal of Computational Geometry & Applications*, 22(03):187–205, 2012.

[10] J. Fiala, J. Kratochvil, and A. Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145:306–316, 2005.

[11] M. T. Goodrich, L. J. Guibas, J. Hershberger, and P. J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings 13th Annual Symposium on Computational Geometry*, pages 284–293, 1997.

[12] T. Harris and F. Ross. Fundamentals of a method for evaluating rail net capacities. Technical report, RAND Corporation, Santa Monica, California, U.S., 1955.

[13] W. Harvey. Computing two-dimensional integer hulls. *SIAM Journal on Computing*, 28(6):2285–2299, 1999.

[14] J. Hershberger. Stable snap rounding. *Computational Geometry*, 46(4):403–416, 2013.

[15] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric methods for digital picture analysis.* Elsevier, 2004.

[16] R. Klette and A. Rosenfeld. Digital straightness - a review. *Discrete Applied Mathematics*, 139(1-3):197–230, 2004.

[17] M. Löffler and W. Meulemans. Discretized approaches to schematization. In *Proceedings 29th Canadian Conference on Computational Geometry*, 2017.

[18] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry: Theory and Applications*, 43(4):419–433, 2010.

[19] J. B. Orlin. Max flows in o(nm) time, or better. In *Symposium on Theory of Computing (STOC)*, pages 765–774, 2013.

[20] I. van der Hoog, M. van de Kerkhof, M. van Kreveld, M. Löffler, F. Staals, J. Urhausen, and J. L. Vermeulen. Mapping multiple regions to the grid with bounded hausdorff distance. In *Algorithms and Data Structures Symposium (WADS)*, 2021. To Appear.

Angles of Arc-Polygons and Lombardi Drawings of Cacti

David Eppstein* Daniel Frishberg* Martha C. Osegueda*

Abstract

We characterize the triples of interior angles that are possible in non-self-crossing triangles with circular-arc sides, and we prove that a given cyclic sequence of angles can be realized by a non-self-crossing polygon with circular-arc sides whenever all angles are $\leq \pi$. As a consequence of these results, we prove that every cactus has a planar Lombardi drawing (a drawing with edges depicted as circular arcs, meeting at equal angles at each vertex) for its natural embedding in which every cycle of the cactus is a face of the drawing. However, there exist planar embeddings of cacti that do not have planar Lombardi drawings.

1 Introduction

Artist Mark Lombardi drew beautiful diagrams of international political and financial conspiracies, often using curved edges and circular layouts [22]. His name is commemorated in Lombardi drawing, a style of graph drawing in which the edges are drawn as circular arcs that meet at equal angles at each vertex [15, 16]. Many kinds of graph are now known to have Lombardi drawings, including for instance all 2-degenerate graphs, the graphs that can be formed from a single edge by repeatedly adding a new vertex incident to at most two previous vertices [15]. Beyond their aesthetic quality, these drawings can be considerably more compact than straight-line drawings of the same graphs [16], and they have found application in the realization of soap bubble foams [18] and in the visualization of knots and links [25].

In this style of drawing, it is of interest, when possible, to avoid any crossings or intersections of edges other than at shared endpoints, forming a planar drawing [14]. Obviously, this requires that the graph to be drawn be planar, but not every planar graph has a planar Lombardi drawing [15]. The planar graphs known to have planar Lombardi drawings include the trees [16], the subcubic planar graphs [18], the 4-regular polyhedral graphs [18], the Halin graphs [15], and the outerpaths [14]. However, examples of planar graphs with no planar Lombardi drawing have been found for graph classes including the 4-regular graphs [18], planar 3-trees [14], and planar bipartite graphs [19]. In addition, for planar graphs with a fixed choice of embedding, a planar Lombardi drawing might not exist even when the given graph is series-parallel [19].

For some of the simplest classes of planar graphs (notably, the outerplanar graphs) the existence of planar Lombardi drawings has remained unknown. In this work, we tackle an even simpler class of graphs, the cacti. Intuitively, a cactus is a tree of cycles; it can be defined as a graph in which each edge belongs to at most one cycle. These graphs have a natural class of planar embeddings in which their cycles form faces of the embedding, with the rest of the graph always drawn outside of the cycle. In this paper, we show that these embeddings of cacti always have Lombardi drawings. However, we find examples of other embeddings of cacti (including one as simple as a triangle with four leaf vertices attached to each triangle vertex) that have no planar Lombardi drawing.

The cycles of any planar Lombardi drawing form simple closed curves in the plane, with sides composed of circular arcs; we call these shapes arc-polygons. The constraints of a Lombardi drawing translate into constraints on the vertex angles of these arc-polygons, and naturally raise the question of which systems of angles can be realized by an arc-polygon and what other geometric properties their realizations have. For instance, the analysis of arc-quadrilaterals with equal angles at all vertices, and the key property of these arc-quadrilaterals that their vertices are all cocircular, figured heavily into our previous work on Lombardi drawings of 4-regular graphs [18], planar bipartite graphs, and embedded series-parallel graphs [19]. Here, we focus on arc-triangles, the simplest (and therefore most highly constrained) shapes needed for the faces of Lombardi drawings of cacti. We completely characterize the triples of angles that can be realized by simple arc-triangles; this characterization forms the basis of our proof that some embedded cacti have no planar Lombardi drawing. We also use the same characterization to prove a natural sufficient condition for the existence of a simple arc-polygon with specified interior angles: such an arc-polygon exists whenever all of the specified angles are at most π. Larger angles than this are not needed for the Lombardi drawings of the natural embeddings of cacti, from which it follows that all cacti have planar Lombardi drawings for their natural embeddings.

*Department of Computer Science, University of California, Irvine, {eppstein,dfrishbe,mosegued}@uci.edu

1.1 New results

The main results that we prove in this work are the following.

- We completely characterize the triples of angles that can be realized as the internal angles of arc-triangles, in terms of a system of linear inequalities on the angles (Theorem 6).
- We prove that every cyclic sequence of three or more angles, all of which are in the range $[0, \pi]$, except for the triple $(0, 0, \pi)$, can be realized as the internal angles of an arc-polygon (Theorem 11).
- We prove that every cactus graph has a planar Lombardi drawing in its natural planar embedding, the embedding in which every cycle is a face (Theorem 17).
- We find an embedded cactus graph that does not have a planar Lombardi drawing for that embedding (Theorem 18).

1.2 Related work

As well as in Lombardi drawing, circular arcs have been incorporated in other ways into graph drawing; see for instance [3, 8, 10, 17, 30]. Force-directed methods can often achieve good but not perfect angular resolution for arc-based graph drawings [5, 11, 12], and the effectiveness of curved edges in graph drawing has been tested through user studies [13, 29, 34].

The cactus graphs whose drawings we study here are an old and well-studied class of graphs [21, 23]. Cactus graphs are one of the most basic minor-closed families of graphs, defined (as simple graphs) by the absence of a diamond graph minor or (as multigraphs) by the absence of a 3-edge dipole graph minor. In graph drawing, cactus graphs play a key role in the proof that polyhedral graphs have greedy embeddings [26], in the visualization of minimum cuts [6], and in the approximation of maximum planar subgraphs [7]. The question of whether cactus graphs have Lombardi drawings was explicitly posed by our previous work on the non-existence of Lombardi drawings for certain bipartite planar graphs [19]; this question was the main motivation for our present work.

Beyond our work's contributions to graph drawing, we believe that it contributes to the fundamental study of arc-polygons. These are a natural and important class of two-dimensional shapes whose long history of study can be traced back to the work of Archimedes and Pappus on the arbelos, an arc-triangle formed by three semicircles on the same side of a line, to the use of the Reuleaux triangle in Gothic architecture and by Leonardo da Vinci for map projection and fortress floor plans, and to the work of 18th-century mathematician Roger Boscovich on shapes that have a center through which every line

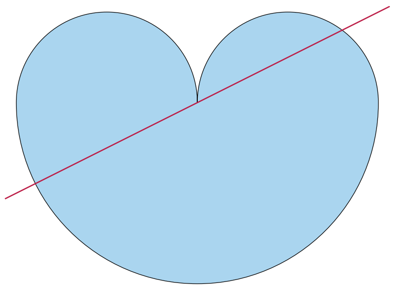


Figure 1: Roger Boscovich's arc-triangle formed from three semicircles has the property that every line through its cusp partitions the perimeter into two equal lengths.

bisects the perimeter (Fig. 1) [4]. It has been stated that "more than 90% of machined parts" have arc-polygon shapes, because of their ease of manufacturing [9], and arc-polygons have been used to model the shapes for irregular parts to be packed into and cut from metal sheets [28]. Arc-polygons can accurately approximate arbitrary smooth curves [20, 27], and their approximations of non-smooth curves form a useful stepping stone to the Riemann mapping theorem [24]. Aichholzer et al. [2] argue that for approximating irregular shapes in this way, arc-polygons have significant advantages over straight-sided polygons in allowing more concise and accurate representations while still allowing efficient computations of basic geometric primitives such as medial axes. For additional work on the computational geometry of these shapes see [1, 32, 33].

2 Preliminaries

2.1 Arc-polygons

Definition 1. *We define an* arc-polygon *to be a cyclic sequence that alternates between points and closed circular arcs in the Euclidean plane, with each arc appearing in the sequence consecutively with its two endpoints. (Here, we allow line segments to count as a degenerate special case of circular arcs.) The points of the arc-polygon are called its* vertices *and the arcs are called its* edges*. In particular, an* arc-triangle *is an arc-polygon with three vertices and three edges. An arc-polygon is* simple *if the only points of intersection between pairs of its edges are shared vertices consecutive with both edges. The union of the arcs in a simple arc-polygon forms a Jordan curve, which separates the plane into two components, the* interior *and* exterior *of the arc-polygon. We define the* interior angle *of a vertex of a simple arc-polygon to be the angle between tangent lines to its two arcs at that vertex, spanning the interior of the arc-polygon in a (possibly empty) neighborhood of the vertex.*

Unlike classical straight-sided polygons, it is also possible to form arc-polygons with only two vertices and two *sides. As in [19], we call these* bigons*.*

A bigon with two different arcs as sides is automatically simple: two different circles can cross in at most two points, and these two crossing points are used up by the vertices of the bigon, so no more crossings are possible. The two interior angles of a bigon must be equal, and can be any angle in the open interval $(0, 2\pi)$. It will be convenient to consider an arc-polygon with two identical arcs to be a kind of degenerate bigon, with interior angle 0.

2.2 Lombardi drawings

Definition 2. *We define a* Lombardi drawing *of a given graph to be a mapping from the vertices of the graph to points in the plane, and from the edges to circular arcs or straight line segments, with the following two properties:*

- *For each edge, the two endpoints of the edge in the graph are mapped to the two endpoints of its arc in the plane.*
- *For each vertex v, the incident edges form arcs that are equally spaced around the point representing v, forming angles of $2\pi/\deg(v)$ between each consecutive pair of arcs.*

A Lombardi drawing is planar *if the only points of intersection between pairs of its arcs are shared vertices.*

2.3 Möbius transformations

By the *extended plane* we mean the Euclidean plane augmented with a single point at infinity, denoted ∞. An *inversion* of the extended plane, with respect to a circle C, maps each point p to another point p', so that p and p' both belong to a single ray from the center of C, with the product of their distances from the center equal to the squared radius of C. The center of C is mapped to ∞, and vice versa. We consider a reflection of the plane to be a degenerate case of an inversion, and we consider the line of reflection to be a degenerate case of a circle with infinite radius. A *Möbius transformation* is any functional composition of inversions. These transformations map circles (or degenerate circles) to other circles (or degenerate circles), and preserve the angles between any two curves. For an introduction to these transformations, and the geometry of circles under these transformations, see e.g. [31].

Because Möbius transformations map circular arcs (or straight line segments) to curves of the same type, and preserve angles, they map simple arc-polygons to other simple arc-polygons and Lombardi drawings to other Lombardi drawings. It is possible for a Möbius transformation to map the interior of a simple arc-polygon to its exterior and vice versa, but otherwise these transformations leave the angles of arc-polygons unchanged.

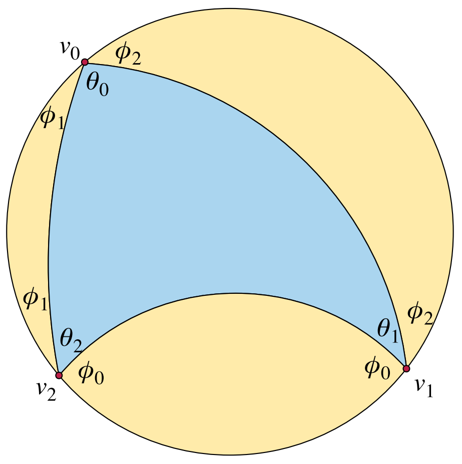


Figure 2: A simple arc-triangle (blue), the circle through its three vertices (yellow), and the three bigons between the arc-triangle and the circle, labeled with vertices v_i, arc-triangle angles θ_i, and bigon angles ϕ_i.

As previous work on Lombardi drawing has already shown [18], these properties make Möbius transformations very convenient as a tool for bringing pieces of arc-polygons and of Lombardi drawings into a position where they can be glued together.

3 Arc-triangles

Consider an arbitrary simple arc-triangle with vertices v_i and interior angles θ_i (for $i \in \{0, 1, 2\}$), for instance the one in Fig. 1 (for which the interior angles are $2\pi, \pi, \pi$) or the one in Fig. 2. Any three points are contained either in a unique circle or a straight line, but for the following definitions we need to know which side of the circle is its inside and which its outside, so we assume (by perturbing the triangle by a Möbius transformation, if necessary) that the three vertices are not collinear, and are clockwise as v_0, v_1, and v_2 on the circle C through them. We assume also that these three vertices have the same clockwise ordering on the arc-triangle, as shown in Fig. 2, meaning that when traveling along the arcs from v_0 to v_1, from v_1 to v_2, and from v_2 to v_0, the polygon is consistently on the right side of each arc; if necessary, this can be achieved by an inversion with respect to C.

Let θ_i be the interior angle of the arc-triangle at vertex v_i, as labeled on the figure. Each arc of the triangle is separated from C by a bigon, and (in the case that the arc is contained in C) we let ϕ_i denote the angle of the bigon opposite vertex v_i. It is also possible for an arc to lie on C, defining a degenerate bigon with $\phi_i = 0$. If an arc of the arc-triangle lies outside C, it still defines a bigon, but in this case we define ϕ_i to be a negative number, the negation of the interior angle of the bigon.

Observation 3. *For angles θ_i and ϕ_i defined as above from a simple arc-triangle, and for $i \in \{0,1,2\}$,*

$$\theta_i + \phi_{(i-1) \bmod 3} + \phi_{(i+1) \bmod 3} = \pi.$$

Proof. The three angles on the left hand side are the angles at v_i measured clockwise from the arc of C clockwise of v_i to the side of the arc-triangle clockwise of v_i, from this side of the arc-triangle to the other side, and from the other side of the arc-triangle to the arc of C counter-counterclockwise of v_i. Therefore, their sum is the total angle at v_i between the two arcs of C, which is just π. □

Corollary 4. *Let $\psi = (\pi - \sum \theta_i)/2$. Then $\phi_i = \psi + \theta_i$.*

Proof. Simple algebra verifies that this is the solution to the system of three linear equations in three unknowns given by Observation 3. □

Lemma 5. *For three given angles θ_i with $0 \le \theta_i \le 2\pi$, at most one of the angles ϕ_i calculated from the formula of Corollary 4 can fail to satisfy $-\pi \le \phi_i \le \pi$. If $\phi_i < -\pi$ then θ_i must be the only angle of the three given angles that is less than π; if $\phi_i > \pi$ then θ_i must be the only angle of the three given angles that is greater than π.*

Proof. Consider the three angles θ_i and the angle π, listed in sorted order. Each ϕ_i is obtained by adding two of these angles, subtracting the other two, and dividing the total by two; in terms of the sorted ordering of the angles, we can represent this (up to the sign of ϕ_i) as one of the three sign patterns $++--$, $+-+-$, or $+--+$. However, only the first of these can produce a value of ϕ_i that is out of range. The other two sign patterns describe the difference of the smallest two angles, plus or minus the difference of the largest two angles. Because these two differences span different ranges of angles, they can sum to at most 2π (and must sum to at least -2π). Therefore, for these other two sign patterns, division by two produces a value in the range between $-\pi$ and π.

For the sign pattern $++--$, π cannot be the maximum or minimum of the three angles, because if it were then each difference of angles would be at most π, and so would the average of two differences. So when this sign pattern leads to a value of ϕ_i that is out of range, there is a unique angle θ_i with the same sign in the sign pattern, implying the second part of the lemma. □

The definitions above of θ_i and ϕ_i are only for simple arc-triangles, but one can also plug in other choices of θ_i, compute ϕ_i from them, and examine the arc-triangles that result, which may not be simple. Examples of what can go wrong are depicted in Fig. 3: the left arc-triangle of the figure has a vertex on the opposite side arc, and the right arc-triangle has two crossing pairs of arcs.

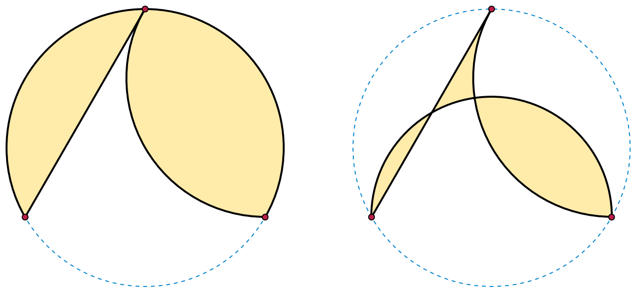


Figure 3: For some triples of angles θ_i, using Corollary 4 to compute angles ϕ_i and then drawing arcs with these angles may not produce a simple arc-triangle. Left: $\theta_i = 0, 4\pi/3, 5\pi/3$ (at the top, right, and left vertices, respectively) and $\phi_i = -\pi, \pi/3, 2\pi/3$. The arc with angle $\phi_i = -\pi$ overlays the opposite vertex. Right: $\theta_i = 0, 3\pi/2, 11\pi/6$, $\phi_i = -7\pi/6, \pi/3, 2\pi/3$. The arc with angle $\phi_i = -7\pi/6$ crosses the other two arcs.

Theorem 6. *Three given angles θ_i with $0 \le \theta_i \le 2\pi$ can be realized as the interior angles of a simple arc-triangle if and only if the angles ϕ_i, as calculated by Corollary 4, satisfy the inequalities $-\pi < \phi_i < \pi$.*

Proof. By Corollary 4, a realization must have the form shown in Fig. 2: three circular arcs making angles of ϕ_i to the circumcircle of the three vertices. The location of these vertices on the circumcircle, and the location of the circumcircle, are not important, as any three points may be transformed into any other three points by a Möbius transformation. The important question for the realizability of these angles by an arc-triangle is whether these three arcs form a simple arc-triangle Δ, or whether they have undesired crossings.

First, let us disprove the existence of a realization when the angles ϕ_i do not satisfy the inequalities.

Consider the case that some $\phi_i = \pm\pi$. In this case, the arc with angle ϕ_i, opposite vertex v_i, coincides with an arc of the circumcircle passing through vertex v_i. Since a simple arc-triangle is not allowed to have one of its sides passing through the opposite vertex, the angles in this case cannot come from a simple arc-triangle.

Next, instead suppose that there is an angle ϕ_i that is out of range: $\phi_i < -\pi$ or $\phi_i > \pi$. By Lemma 5, it must be the only angle with this property. In the case that $\phi_i < -\pi$, consider continuously increasing ϕ_i (and decreasing the two given angles that are not θ_i to match) until it is greater than $-\pi$. By the second part of Lemma 5, this decrease in the given angles cannot cause them to become negative. At the end of the sequence, the angles meet the conditions of the previous case of the theorem, producing a simple arc-triangle, but at the point in the sequence where ϕ_i becomes equal to $-\pi$, and the arc with angle ϕ_i crosses over vertex v_i, two crossings with the other two arcs are removed.

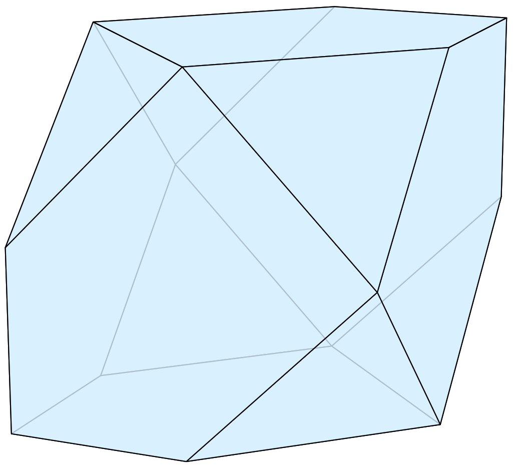


Figure 4: Polyhedral visualization of the subset of triples of angles that can be realized by simple arc-triangles.

There is no combinatorial change to the configuration of arcs and their crossings anywhere else in the sequence, so these two crossings must have been present in the configuration with the given angle ϕ_i. In the case when $\phi_i > \pi$, instead continuously decrease ϕ_i until it is less than π; the remaining argument is the same.

Finally, let us prove that angles ϕ_i satisfying the inequalities must be simple. Suppose all ϕ_i obey $-\pi < \phi_i < \pi$. Then it is impossible for the two arcs at v_i to cross. If the values of ϕ corresponding to these two arcs have opposite signs, then one is inside the circumcircle and the other outside, and if one has sign zero, then it lies on an arc of the circumcircle inaccessible to the other. If both arcs have values of ϕ with the same sign, then they leave the circumcircle at v_i in a relative ordering, determined by their angle at v_i, that is consistent with the relative ordering at which they reach the circumcircle again at the other two vertices, determined by the positions of those two vertices. Since circular arcs can meet only twice, and these two arcs meet once at v_i, any additional crossing would swap their relative positions, so they cannot cross. □

The linear inequalities $0 \leq \theta_i \leq 2\pi$ can be thought of as defining a cube in a three-dimensional space having the angles θ_i as Cartesian coordinates. For this interpretation of these numbers, the additional linear inequalities $-\pi < \phi_i < \pi$ cut off six of the eight corners of the cube (the corners where not all coordinate values are equal), replacing them by equilateral triangle faces whose vertices are midpoints of the cube edges. The result of this truncation is a feasible region of triples of angles that can be realized by simple arc-triangles, bounded by six pentagonal faces (the truncated faces of the cube) and six triangular faces (the inequalities on ϕ_i). It is depicted in Fig. 4. The result in Lemma 5 that only one of the inequalities on ϕ_i can be exceeded corresponds geometrically to the fact that the triangular faces meet only in vertices, not in edges. At the vertices where two triangular faces meet, two of these inequalities on ϕ_i are exactly met but neither is exceeded.

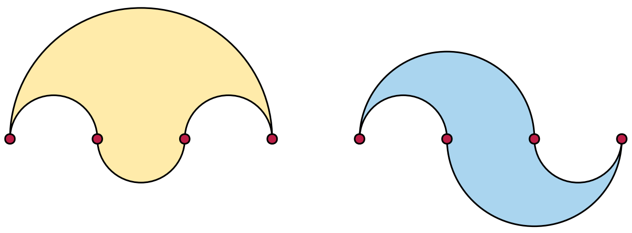


Figure 5: Arc-quadrilaterals with interior angles $(0, 0, \pi, \pi)$ (left, yellow) and $(0, \pi, 0, \pi)$ (right, blue).

We remark that it is straightforward to implement this construction of a simple arc-triangle for given angles and given triangle vertices, by computing the circumcircle of the vertices and the angle made by the triangle arcs to this circumcircle.

4 Arc-polygons

We do not have a complete characterization of the angle sequences of higher-order simple arc-polygons, as we do for arc-triangles. Instead, we prove a sufficient condition, which will be enough for our application to Lombardi drawing: with a single exception, an angle sequence can be realized by a simple arc-polygon whenever all angles are $\leq \pi$. For triangles, this follows from our previous characterization:

Lemma 7. *If a triple of given angles θ_i all lie in the range $0 \leq \theta_i \leq \pi$ and is not a permutation of $(0, 0, \pi)$ then there exists a simple arc-triangle having these interior angles.*

Proof. For these angles, the values of ϕ_i calculated according to Corollary 4 range from a minimum of $-\pi/2$ (when $\theta_i = 0$ and both other given angles are π) to a maximum of π (when $\theta_i = \pi$ and both other given angles are 0). Because the angles $(0, 0, \pi)$ are the only combination achieving this maximum, all other triples of angles result in $-\pi/2 \leq \phi_i < \pi$. The result follows from Theorem 6. □

We also need the following two special cases:

Observation 8. *The sequences $(0, 0, \pi, \pi)$ and $(0, \pi, 0, \pi)$ can be realized as the interior angles of arc-quadrilaterals.*

Proof. See Fig. 5, which realizes these sequences using four vertices equally spaced along a line, with semicircular arcs. □

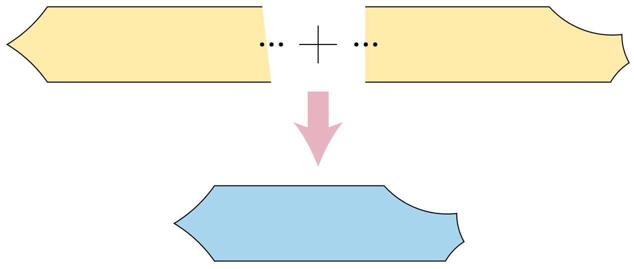


Figure 6: Gluing together two simple arc-polygons with cusps at infinity, realizing angle sequences $\sigma 0$ and $\tau 0$, to produce a single arc-polygon realizing angle sequence $\sigma\tau$.

For higher-order polygons, we will prove the existence of a simple realization by induction on the order of the polygon, gluing together arc-polygons with fewer vertices at *cusps*, vertices of interior angle zero.

Definition 9. *If σ and τ denote sequences of angles, we let $\sigma\tau$ denote their concatenation, and we let $\sigma 0$ and $\tau 0$ denote the sequences of angles obtained from σ and τ respectively by including one more angle equal to zero.*

With this notation, the following lemma describes the process of gluing together two arc-polygons.

Lemma 10. *Let σ and τ both be sequences of angles such that the augmented sequences $\sigma 0$ and $\tau 0$ are realizable as the sequences of interior angles of simple arc-polygons. Then the concatenation $\sigma\tau$ is also realizable by a simple arc-polygon.*

Proof. Let S and T be simple arc-polygons realizing σ_0 and τ_0 respectively. Apply inversions separately to S and T, centered respectively on the cusps of S and T, producing arc-polygons S' and T'. Both S' and T' have their cusp transformed to lie at ∞, and the circular arcs incident to these cusps transformed into infinite rays. All other vertices and edges of S' and T' remain finite. The condition that the angle at the cusp be zero in S and T corresponds, in S' and T', to the condition that these two rays be parallel. By adjusting the radii of the inversions we can additionally ensure that in both S' and T' the rays lie on lines at distance one from each other. By rotating S' and T' so that their pairs of rays both lie on the same two lines, in opposite directions, and translating them far enough apart along these two parallel lines, we may glue them together into a single simple arc-polygon, as shown in Fig. 6, giving a realization of $\sigma\tau$. $\square$

Theorem 11. *A finite sequence of three or more angles θ_i can be realized as the interior angles of a simple arc-polygon whenever all angles satisfy $0 \le \theta_i \le \pi$ and the sequence is not a permutation of $(0, 0, \pi)$.*

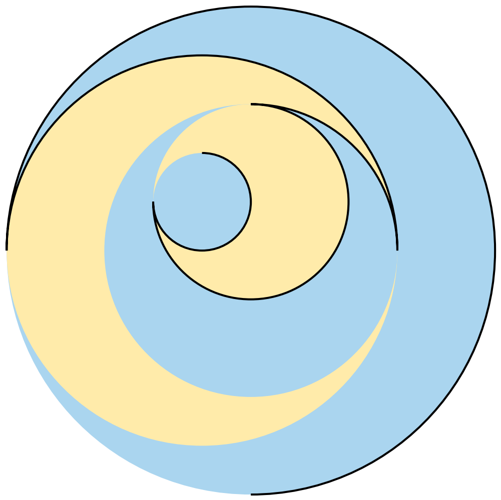


Figure 7: Interior angles that alternate between 0 and 2π force the arcs of a simple arc-polygon to be confined to a sequence of nested circles.

Proof. We use induction on the number of angles, with Lemma 7 as a base case for sequences of exactly three angles and Observation 8 for the two four-angle-sequences $(0, 0, \pi, \pi)$ and $(0, \pi, 0, \pi)$ and their cyclic permutations.

For any other sequence of more than three angles that includes π as one of its angles, we form a shorter sequence by removing the π, realize the shorter sequence as the interior angles of a simple arc-polygon by the induction hypothesis, and then reinsert the vertex with angle π anywhere along the circular arc between its two neighbors in the resulting arc-polygon.

For a sequence of more than three angles that does not include π as an angle, partition the sequence into a concatenation $\sigma\tau$ where both σ and τ contain at least two angles. By the induction hypothesis, both $\sigma 0$ and $\tau 0$ are realizable, so by Lemma 10 their concatenation $\sigma\tau$ is also realizable. $\square$

It is tempting to guess that all sequences of four or more angles are realizable by simple arc-polygons, but this is not true, as the following example demonstrates.

Observation 12. *In a simple arc-polygon, if the two interior angles at the vertices of a given arc A are 0 and 2π, then the two adjacent arcs lie on opposite sides of the circle through A.*

Proof. The arcs are tangent to A, and two circles that are tangent cannot cross. The choice of angles ensures that, near the point of tangency, the arcs are on opposite sides of the circle, and because they cannot cross this circle they remain on opposite sides for their entire length. $\square$

Corollary 13. *It is not possible for a simple arc-polygon with an even number of sides to have angles that alternate between 0 and 2π around the polygon.*

Proof. The alternation would cause the sides to belong to a sequence of nested circles (Fig. 7), from which it would be impossible to loop back around from the innermost to the outermost circle. □

5 Lombardi drawing

5.1 Existence

As stated in the introduction, a *cactus* is a graph in which each edge belongs to at most one cycle. In the natural embedding of a cactus, each cycle forms a face, with the other edges that are incident to each cycle vertex placed outside the face. At a vertex of degree $d \geq 2$, the interior angle of the cycle will be $2\pi/d$. Because all angles are nonzero and at most π, the sequence of interior angles meets the conditions of Theorem 11, allowing the face to be drawn as a simple arc-polygon. Each edge belongs to at most one cycle, and the edges that do not belong to cycles are even easier to draw (on their own) as line segments. To construct a Lombardi drawing for the entire graph, we glue these pieces of drawings together using Möbius transformations, similar to the gluing process used to construct arc-polygons in Lemma 10.

Definition 14. *If G is an embedded graph, and H is a subgraph of G we define a* partial Lombardi drawing *of H with respect to G to be a drawing of the vertices and edges of H as points and circular arcs, in the manner of a Lombardi drawing, with pairs of circular arcs that meet at a vertex having the angle that they would be required to have in a Lombardi drawing of G. We define a partial Lombardi drawing to be* planar *in the same way as for a full Lombardi drawing: the only intersection points of arcs are shared endpoints.*

The *biconnected components* of a cactus are its cycles and its edges that do not belong to cycles. Its *articulation points* are the vertices that belong to more than one biconnected component. The reasoning above proves the following observation:

Observation 15. *Each biconnected component of a cactus has a planar partial Lombardi drawing with respect to the natural embedding of the cactus.*

Lemma 16. *Let H_i be a collection of subgraphs of a given graph G that all share a common vertex v, with no other pairwise intersections, let G be given a planar embedding in which the edges of each subgraph H_i appear consecutively at v, and let each H_i have a planar partial Lombardi drawing with respect to G. Then the union $\cup H_i$ also has a planar partial Lombardi drawing with respect to G.*

Proof. Perform an inversion centered at v on each drawing of H_i, producing a drawing where v is at ∞ and its incident edges have become rays, radiating outward from the finite part of the drawing, with relative angles equal to the angles they should have at v in a Lombardi drawing of G.

Let v have degree d in G, and draw a system of d rays meeting at the origin in the plane, separated from each other by angles of $2\pi/d$. Assign consecutive subsets of rays to each subgraph H_i, according to its number of edges incident to v. Additionally assign to each subgraph H_i a wedge of the plane, with the apex as the origin, extending beyond these assigned rays to an additional angle of π/d on both sides. This assignment produces open wedges for each H_i, within which (after a suitable rotation) each may be placed; we place each H_i so that its rays are parallel to the rays it is assigned, but we do not require these rays to coincide. Because the wedges are open, they are disjoint from each other and from the origin.

Once all of the drawings of subgraphs have been placed in this way, another inversion centered at the origin returns the drawing to a state in which all vertices and arcs are finite. All of the angles within each drawing of each subgraph H_i have been preserved, and the rays to ∞ invert to circular arcs meeting at v at the desired angles. □

Theorem 17. *Every cactus has a planar Lombardi drawing for its natural embedding.*

Proof. We construct planar Lombardi drawings for each biconnected component of the cactus by Observation 15, and glue these partial drawings into partial drawings for larger subgraphs by applying Lemma 16 at each articulation point of the cactus, until the entire drawing has been glued together. Each step preserves planarity, so the resulting Lombardi drawing is planar. □

5.2 Nonexistence

Theorem 18. *The embedded cactus depicted in Fig. 8, consisting of a 3-cycle with four pendant vertices at each 3-cycle vertex, embedded so that the pendant vertices are outside the 3-cycle at two 3-cycle vertices and inside at one, has no planar Lombardi drawing.*

Proof. A planar Lombardi drawing of this embedded graph would draw its 3-cycle as a simple arc-triangle with angles θ_i of $\pi/3$, $\pi/3$, and $5\pi/3$. A calculation following Corollary 4 shows that these angles produce corresponding angles ϕ_i of $-\pi/3$, $-\pi/3$, and π, respectively. Since one of the angles ϕ_i is π, Theorem 6 shows that this simple arc-triangle cannot exist. □

Additional pendant vertices only make the angles farther from realizability, producing an infinite family of examples without planar Lombardi drawings.

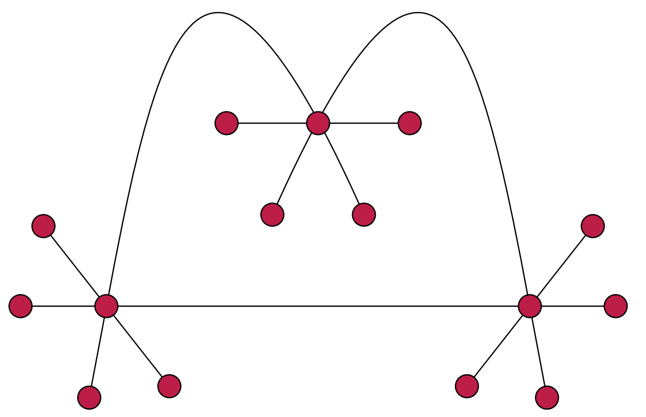


Figure 8: An embedded cactus that has no planar Lombardi drawing.

6 Conclusions

We have completely characterized the triples of internal angles of simple arc-triangles, and proven that for higher-order simple arc-polygons (with one exception) all sequences of angles that are at most π are realizable. We have used these results to find planar Lombardi drawings for the natural planar embeddings of all cacti, and to prove that certain other embeddings of cacti do not have planar Lombardi drawings. In particular, this shows that it is possible for an outerplanar graph, embedded in a non-outerplanar way, to fail to have a planar Lombardi drawing.

This work leaves the following questions open for future research:

- Can we characterize more completely the sequences of angles that can be realized by simple arc-polygons? (Corollary 13 shows that this question has a nontrivial answer.) What is the computational complexity of finding such a realization?
- Similarly, what is the computational complexity of determining whether an embedded cactus has a planar Lombardi drawing?
- In previous work, for trees with fixed embeddings, we were able to prove the existence of Lombardi drawings whose area is a polynomial multiple of the minimum separation between vertices, compared to the exponential area requirement for straight-line drawings with uniformly spaced edges at each vertex [16]. What are the area requirements for Lombardi drawing of cacti?
- As already discussed in our previous work on Lombardi drawing, do all outerplanar graphs have planar Lombardi drawings? Do they have planar Lombardi drawings for each outerplanar embedding? What is the computational complexity of finding these drawings?

References

[1] O. Aichholzer, W. Aigner, F. Aurenhammer, K. Čech Dobiásová, B. Jüttler, and G. Rote. Triangulations with circular arcs. *J. Graph Algorithms Appl.* 19(1):43–65, 2015, doi:10.7155/jgaa.00346, MR3321736.

[2] O. Aichholzer, F. Aurenhammer, T. Hackl, B. Jüttler, M. Rabl, and Z. Šíŕ. Computational and structural advantages of circular boundary representation. *Internat. J. Comput. Geom. Appl.* 21(1):47–69, 2011, doi:10.1142/S0218195911003548, MR2777029.

[3] P. Angelini, D. Eppstein, F. Frati, M. Kaufmann, S. Lazard, T. Mchedlidze, M. Teillaud, and A. Wolff. Universal point sets for drawing planar graphs with circular arcs. *J. Graph Algorithms Appl.* 18(3):313–324, 2014, doi:10.7155/jgaa.00324, MR3213192.

[4] T. Banchoff and P. Giblin. On the geometry of piecewise circular curves. *Amer. Math. Monthly* 101(5):403–416, 1994, doi:10.2307/2974900, MR1272938.

[5] M. J. Bannister, D. Eppstein, M. T. Goodrich, and L. Trott. Force-directed graph drawing using social gravity and scaling. *Proc. 20th Internat. Symp. Graph Drawing (GD 2012)*, pp. 414–425. Springer, Lect. Notes Comput. Sci. 7704, 2012, doi:10.1007/978-3-642-36763-2_37.

[6] U. Brandes, S. Cornelsen, C. Fieß, and D. Wagner. How to draw the minimum cuts of a planar graph. *Comput. Geom.* 29(2):117–133, 2004, doi:10.1016/j.comgeo.2004.01.008, MR2082210.

[7] G. Călinescu, C. G. Fernandes, U. Finkler, and H. Karloff. A better approximation algorithm for finding planar subgraphs. *J. Algorithms* 27(2):269–302, 1998, doi:10.1006/jagm.1997.0920, MR1622397.

[8] J. Cardinal, M. Hoffmann, V. Kusters, C. D. Tóth, and M. Wettstein. Arc diagrams, flip distances, and Hamiltonian triangulations. *Comput. Geom.* 68:206–225, 2018, doi:10.1016/j.comgeo.2017.06.001, MR3715053.

[9] J.-M. Chen, J. A. Ventura, and C.-H. Wu. Segmentation of planar curves into circular arcs and line segments. *Image Vis. Comput.* 14(1):71–83, 1996, doi:10.1016/0262-8856(95)01042-4.

[10] C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. *Discrete Comput. Geom.* 25(3):405–418, 2001, doi:10.1007/s004540010080, MR1815440.

[11] R. Chernobelskiy, K. I. Cunningham, M. T. Goodrich, S. G. Kobourov, and L. Trott. Force-directed Lombardi-style graph drawing. *Proc. 19th Internat. Symp. Graph Drawing (GD 2011)*, pp. 320–331. Springer, Lect. Notes Comput. Sci. 7034, 2011, doi:10.1007/978-3-642-25878-7_31.

[12] W. Dong, X. Fu, G. Xu, and Y. Huang. An improved force-directed graph layout algorithm based on aesthetic criteria. *Comput. Vis. Sci.* 16(3):139–149, 2013, doi:10.1007/s00791-014-0228-5.

[13] P. M. Dudas. *The Impact of Visual Aesthetics on the Utility, Affordance, and Readability of Network Graphs*. Ph.D. thesis, University of Pittsburgh, 2016, https://d-scholarship.pitt.edu/26607/.

[14] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, M. Löffler, and M. Nöllenburg. Planar and poly-arc Lombardi drawings. *J. Comput. Geom.* 9(1):328–355, 2018, doi:10.20382/jocg.v9i1a11, MR3855883.

[15] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg. Lombardi drawings of graphs. *J. Graph Algorithms Appl.* 16(1):85–108, 2012, doi:10.7155/jgaa.00251, MR2872431.

[16] C. A. Duncan, D. Eppstein, M. T. Goodrich, S. G. Kobourov, and M. Nöllenburg. Drawing trees with perfect angular resolution and polynomial area. *Discrete Comput. Geom.* 49(2):157–182, 2013, doi:10.1007/s00454-012-9472-y, MR3017904.

[17] A. Efrat, C. Erten, and S. G. Kobourov. Fixed-location circular arc drawing of planar graphs. *J. Graph Algorithms Appl.* 11(1):145–164, 2007, doi:10.7155/jgaa.00140, MR2354167.

[18] D. Eppstein. A Möbius-invariant power diagram and its applications to soap bubbles and planar Lombardi drawing. *Discrete Comput. Geom.* 52(3):515–550, 2014, doi:10.1007/s00454-014-9627-0, MR3257673.

[19] D. Eppstein. Bipartite and series-parallel graphs without planar Lombardi drawings. *Proc. 31st Canad. Conf. Comput. Geom. (CCCG 2019)*, pp. 17–22, 2019.

[20] L. Fejes Tóth. Approximation by polygons and polyhedra. *Bull. Amer. Math. Soc.* 54:431–438, 1948, doi:10.1090/S0002-9904-1948-09022-X, MR24640.

[21] F. Harary and G. E. Uhlenbeck. On the number of Husimi trees, I. *Proc. Natl. Acad. Sci. U.S.A.* 39(4):315–322, 1953, doi:10.1073/pnas.39.4.315, MR0053893.

[22] R. C. Hobbs and J. Richards. *Mark Lombardi: Global Networks*. Independent Curators International, 2003.

[23] K. Husimi. Note on Mayers' theory of cluster integrals. *J. Chem. Phys.* 18(5):682–684, 1950, doi:10.1063/1.1747725, MR0038903.

[24] O. D. Kellogg. The second edition of the Hurwitz–Courant Funktionentheorie. *Bull. Amer. Math. Soc.* 32(3):288–292, 1926, doi:10.1090/S0002-9904-1926-04215-4, MR1561208.

[25] P. Kindermann, S. G. Kobourov, M. Löffler, M. Nöllenburg, A. Schulz, and B. Vogtenhuber. Lombardi drawings of knots and links. *J. Comput. Geom.* 10(1):444–476, 2019, doi:10.20382/jocg.v10i1a15, MR4039890.

[26] T. Leighton and A. Moitra. Some results on greedy embeddings in metric spaces. *Discrete Comput. Geom.* 44(3):686–705, 2010, doi:10.1007/s00454-009-9227-6, MR2679063.

[27] D. S. Meek and D. J. Walton. Approximating smooth planar curves by arc splines. *J. Comput. Appl. Math.* 59(2):221–231, 1995, doi:10.1016/0377-0427(94)00029-Z, MR1346015.

[28] S. Plankovskyy, Y. Tsegelnyk, O. Shypul, A. Pankratov, and T. Romanova. Cutting irregular objects from the rectangular metal sheet. *Integrated Computer Technologies in Mechanical Engineering*, pp. 150–157. Springer, Adv. Intell. Syst. Comput. 1113, 2020, doi:10.1007/978-3-030-37618-5_14.

[29] H. C. Purchase, J. Hamer, M. Nöllenburg, and S. G. Kobourov. On the Usability of Lombardi Graph Drawings. *Proc. 20th Internat. Symp. Graph Drawing (GD 2012)*, pp. 451–462. Springer, Lect. Notes Comput. Sci. 7704, 2012, doi:10.1007/978-3-642-36763-2_40.

[30] A. Schulz. Drawing graphs with few arcs. *J. Graph Algorithms Appl.* 19(1):393–412, 2015, doi:10.7155/jgaa.00366, MR3395765.

[31] H. Schwerdtfeger. *Geometry of Complex Numbers*. Dover, 1979.

[32] Z.-J. Wang, X. Lin, M.-E. Fang, B. Yao, Y. Peng, H. Guan, and M. Guo. Re2L: an efficient output-sensitive algorithm for computing Boolean operations on circular-arc polygons and its applications. *Comput.-Aided Des.* 83:1–14, 2017, doi:10.1016/j.cad.2016.07.004, MR3577906.

[33] B. Weiß, B. Jüttler, and F. Aurenhammer. Mitered offsets and straight skeletons for circular arc polygons. *Eur. Worksh. Comput. Geom. (EuroCG 2018)*, pp. 52:1–52:6, 2018, https://conference.imp.fu-berlin.de/eurocg18/download/paper_52.pdf.

[34] K. Xu, C. Rooney, P. J. Passmore, D. Ham, and P. H. Nguyen. A user study on curved edges in graph visualization. *IEEE Trans. Vis. Comput. Graph.* 18(12):2449–2456, 2012, doi:10.1109/TVCG.2012.189.

Practical Methods for the Embroidery Problem

Michelle Tran*

Abstract

We consider the problem of optimizing thread usage while embroidering a given pattern denoted by a Euclidean graph G. A valid embroidery of a pattern is described as a closed tour with alternating front and back edges, each with an associated weight (which, due to the Euclidean embedding, is equivalent to the distance between vertices). We explore various options for calculating exact and approximate solutions, and analyze the feasibility of each in practice.

1 Introduction

Embroidery is a fiber art where the artist creates 2D designs on fabric. As a purely aesthetic craft, this “image” is created on one side of the fabric, while the other side is solely for utility. As such, stitches have a ‘front’ and ‘back’ to them: the design on the side of the fabric that is meant to be seen will always be given in the design, but the back of the fabric can have any number of paths, depending on how the artist chose to transition between stitches. There are no restrictions on the movement on the back side of the canvas, so choosing how to transition between stitches is what causes the difference in thread usage.

This paper focuses on finding practical ways of solving the problem of efficient thread usage: we discuss both exact and approximate approaches to the optimization problem, where we minimize the amount of thread used for a given pattern. In technical terms, the embroidery problem is a graph traversal optimization problem if we assume the constraints of forming a cycle (i.e. the end of the stitching should return to the starting point so that a knot can be tied) and the thread is not cut at any point. Given a Euclidean graph $G(V, E)$, where V is a set of vertices representing Euclidean coordinates and E is a set of edges with weights corresponding to the Euclidean distance between vertices, we seek to find the minimum weight of the optimal tour T that alternates between front edges (equivalent to set E) and back edges (the set of edges in the complete graph on vertices V). The tour must traverse each edge in E exactly once, while there are no restrictions on how many times back edges can be traversed.

The body of literature for this specific problem is limited: Arkin *et al.* has produced the only paper about the general embroidery problem [4], though Biedl, Horton, and Lopez-Ortiz have written a paper about a more specific version of embroidery known as cross stitching [5]. Both papers conclude that even in a restricted form (like the case of cross stitching), the embroidery optimization problem is NP-hard, and thus they provide approximation algorithms. Graphs of arbitrary makeup are shown to have a 2-approximation, though more specific configurations have better algorithms: in fact, connected designs can be solved in polynomial time, while independent segments (i.e. edges in the design do not share endpoints) have a 1.5-approximation [4]. In essence, both papers demonstrate that the difficulty of this problem is substantial through theoretical results.

There are much more popular problems that are similar to the embroidery optimization problem: namely, the Rural Postman Problem (RPP) (where the main distinction is that the embroidery problem requires alternating between front and back thread usage, and thus the tour cannot traverse two required edges consecutively) and the Euclidean Traveling Salesperson Problem (TSP) (which ensures all vertices, rather than edges, are traversed). Both have been studied extensively, especially TSP. There exist many established algorithms for TSP, most notably Christofides’ 3/2-approximation algorithm for metric TSP [6] (though Karlin, Klein, and Gharan recently discovered a very slightly improved version of Christofides’ algorithm in September of 2020 [8]). Dantzig *et al.* also created an established approach to approximating TSP using linear programming [7]. These methods are well tested theoretically and empirically.

Due to the lack of study, the embroidery problem does not have any empirical data behind its approximation algorithms, nor does it have practical implementations. This paper seeks to change that by providing practical implementations and benchmarking by using TSP solvers and approximation algorithms.

2 Technical Specifications & Procedure

All measurements were taken on a MacBook Pro 2017 with a 3.1 GHz Dual-Core Intel Core i5 processor and 8 GB of memory. The scripts are written in Python3 (specifically, Python 3.6.5 at the time of measurement).

*Computer Science Undergraduate Program, University of Colorado Boulder, michelle.h.tran@colorado.edu

For benchmarking, random instances of the Embroidery Problem were generated with n ”stitches”, i.e. tuples of two coordinates on the Euclidean plane, on a 5x5 grid (dimensions were selected due to the limitations of the experimental hardware and to prevent excessive sparseness for the average Embroidery instance). For each n, 100 trials were taken, and the resulting figures plot the mean of each set of 100 computation time measurements. Graphs for individual methods (*Figures 1, 4, 5,* and *6*) were all done on their own set of random Embroidery Problem instances. For *Figure 7*, each method was applied to the same Embroidery Problem instance for more direct comparison.

All data from this paper can be found here, under the *fair_comparison* branch.

3 Complexity Classification

As mentioned by Arkin *et al.*, Euclidean TSP can be reduced to the embroidery problem, and thus we can classify embroidery as NP-Hard. However, it remains an open question as to whether the embroidery problem is NP-Complete: is embroidery in NP?

A major feature of the Embroidery Problem is that the sum of the length of the thread consists of many sums of square roots (due to the distance formula). However, the summation of square roots is widely regarded as an open problem, seeing as comparing a decimal value with irrational numbers can be computationally difficult due to cases where the sum is very close, but not quite, equal to the given quantity [1]. For a problem to be in NP, it must be a decision problem – the natural decision problem for the Embroidery Problem bounds the total thread usage by some k. This decision problem requires us to compare thread usage (which will be a sum of square roots) and an integer k, meaning that verifying embroidery problem instances likely cannot be done in polynomial time.

This is further evident by the reduction to Euclidean TSP [4] – Euclidean TSP is likely not in NP due to this same exact issue. While some researchers claim that Euclidean TSP is NP-Complete [9], they explicitly state as an assumption that there exists a large enough scale factor that each weight can be multiplied by to get integer weight values. However, this ignores irrational numbers, and the square root of any non-square integer is irrational; so, for the sake of accuracy, we will not make that assumption. Therefore, Euclidean TSP is likely not in NP, thus it is unlikely for the embroidery problem to be in NP as well.

This distinction is important as it eliminates the possibility of using reductions to NP-Complete problems that have rigorously improved practical implementations, such as the Constraint Satisfiability Problem (SAT).

4 TSP Solvers

The embroidery problem is quite similar to the popular optimization problem, Traveling Salesperson (TSP). Due to its relevance in practical applications, many TSP solvers have been written and refined over the years to become extremely efficient for an NP-Hard problem. Thus, it could be advantageous to reduce embroidery to TSP, and then leverage these state-of-the-art solvers.

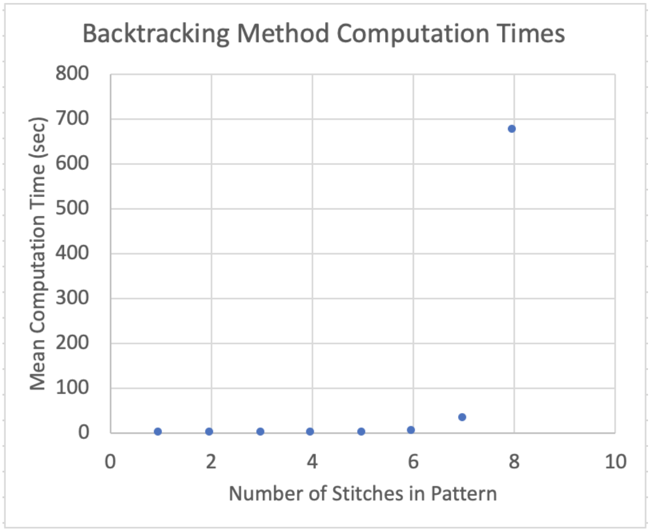
4.1 Point of Comparison: Backtracking Algorithm

The brute force algorithm for embroidery is quite simple in concept: we can use a backtracking algorithm to check every possible traversal of the embroidery pattern, all the while ensuring that the tour is alternating between front and back edges. We use a boolean flag to track when we are on the front side or the back side of the fabric: while on the front side, we use the adjacency matrix of the vertex to determine which edge to take next, and keep track of which edges have been traversed in a visited list. For back edges, we simply try all other vertices in the graph (seeing as the graph of back edges in an embroidery instance is a complete graph). The implementation for this algorithm can be found here.

Table 1: Mean brute-force computation times

# Stitches	Mean	# Stitches	Mean
1	3.65e-05	5	1.64e-01
2	2.52e-04	6	2.28+00
3	1.86e-03	7	3.30e+01
4	1.46e-02	8	6.74e+02

Figure 1: Runtimes for the brute-force algorithm



Whether on the front or the back side of the fabric, each node in the search tree space will branch at most

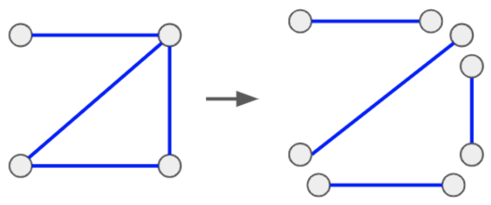
$n-1$ times, n being the number of vertices in the graph. Thus, as expected for a brute-force backtracking algorithm, its time complexity is exponential: $O(n^n)$. Empirically, we can see by the benchmarking in *Table 1* that despite the small search space, the backtracking algorithm was extremely slow.

4.2 Reduction to TSP

To utilize the power of TSP solvers, we must first reduce embroidery to TSP ($EMBROIDERY \leq_p TSP$). Given an input graph to the embroidery problem G(V, E, B), where V is the set of vertices at Euclidean locations in space, E is the set of front edges in the stitching, and B is the set of back edges that can be used in the stitching, we can reduce to an instance of TSP.

There are a few non-TSP properties of the embroidery problem that we must maintain: namely, enabling some vertices to be repeated (which is technically allowed for the TSP problem in general, but the TSP solvers used in this paper do not allow this), the alternating nature of the stitching, and that all front stitches are traversed precisely once during the tour. The idea is that we duplicate all vertices in V by the degree of the front edges for each vertex to create $G_d(V_d, E)$ (where the d subscript indicates we are using the set of duplicated vertices), as shown in *Figure 2*. We then copy all of the vertices in V_d to construct a graph G'_d with only the back edges (i.e. the complete graph of all nodes in V_d, except nodes that correspond to the same $v \in V$ will not be connected), where $G'_d(V'_d, B_d)$.

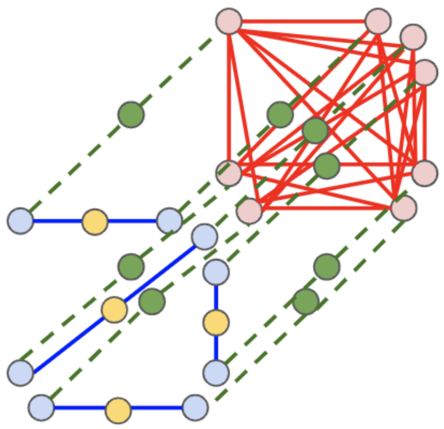
Figure 2: Example of duplicating vertices by degree of front edges



We then connect these two graphs $G_d(V_d, E)$ to $G'_d(V'_d, B_d)$ using two edges of weight 1 and a gadget node that goes between each vertex and its copy. For example, say we are connecting vertex $v \in V_d$ to $v' \in V'_d$: use gadget vertex u and two 1-weight edges $e_1(v, u)$ and $e_2(u, v')$ to connect the vertices. With the gadget vertex, optimal TSP tours are now guaranteed to alternate between front and back edges: practically, this is due to the fact that the TSP solvers used in this paper do not repeat vertices. Theoretically, optimal tours are guaranteed to alternate due to the triangle inequality (consecutive back edges can be eliminated without increasing the length of the tour). These gadget vertices and edges are the green components shown in *Figure 3*. It is important to note that the edges for this gadget should be weighted zero in concept so that alternating between sides of the fabric doesn't have a cost associated with it. However, zero weight edges are not compatible with the TSP solvers used in this paper. Thus, we simply subtract the excess weight after this reduction is run through a TSP solver, which will always be $2 \cdot |V_d|$, since there exists two 1-weight edges for every vertex in G_d.

As for ensuring that we traverse each front edge, we can once again use gadget vertices: for each front edge in the graph, insert a gadget node u "on" an edge $e(v_i, v_j)$, as in, remove e, and connect vertices v_i and v_j with a similar gadget to before, where the edges are $e_1(v_1, u)$ and $e_2(u, v_2)$. However, this time, the weights of e_1 and e_2 must sum to the weight of e – it does not matter what exactly the weight of each value is, just that both are non-zero. These gadgets are represented in *Figure 3* as the yellow vertices in G_d. This reduction is implemented here as the function *multi_node_reduction()*.

Figure 3: Final reduction on the embroidery instance from Fig. 2.



This reduction is in polynomial time. Duplicating vertices requires iterating through each vertex in V and getting the number of neighbors per vertex, which takes $O(|V|)$ time. Creating each gadget is done in constant time since there are precisely two edges and one vertex per gadget, and these gadgets only need to be added in a polynomially bounded number of locations: one for each vertex in $G_d(|V_d|)$, and one between every front edge (the maximum size of the front edges E is the complete graph on the original vertices V, which is would be of size $\frac{|V|\cdot(|V|-1)}{2}$). Creating the back edges for G'_d requires creating a nearly complete graph on V'_d, which will be at most $\frac{|V_d|\cdot(|V_d|-1)}{2}$) edges. The runtime for this reduction is thus $|V|+|V_d|+\frac{|V|\cdot(|V|-1)}{2}+\frac{|V_d|\cdot(|V_d|-1)}{2} = |V|+|V|^2+$

$\frac{|V|\cdot(|V|-1)}{2} + \frac{|V|^2\cdot(|V|^2-1)}{2} = O(|V|^4)$, since V_d is at most $|V|^2$.

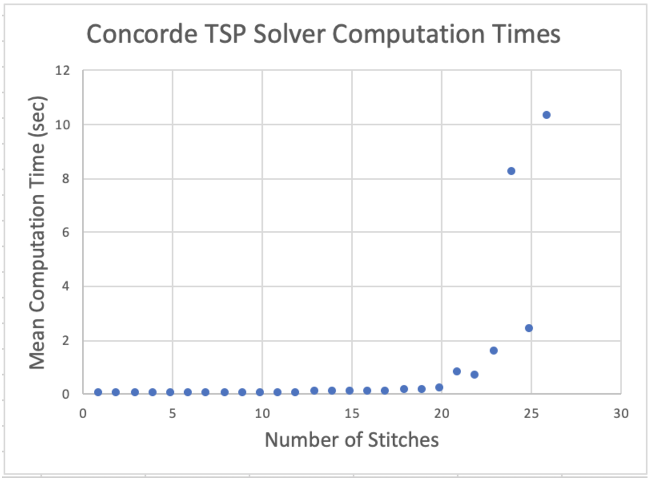
4.3 Concorde TSP Solver

Now that we have a TSP reduction from the embroidery problem, we can leverage the immense progress made by many theorists by using state of the art TSP solvers. The Concorde TSP solver from the University of Waterloo is regarded as the most accurate and efficient TSP solver to date [3]. Using PyConcorde (a Python wrapper for the Concorde solver and the linear program solver QSopt [11]), we can now evaluate the TSP reduction. See *Table 2* for benchmarking.

Table 2: Concorde TSP Solver computation times

# Stitches	Mean	# Stitches	Mean
1	0.0137	14	0.0461
2	0.0167	15	0.0556
3	0.0173	16	0.0835
4	0.0170	17	0.0881
5	0.0166	18	0.1036
6	0.0169	19	0.1261
7	0.0181	20	0.1857
8	0.0208	21	0.8137
9	0.0249	22	0.6732
10	0.0252	23	1.5560
11	0.0296	24	8.2427
12	0.0377	25	2.4030
13	0.0445	26	10.3159

Figure 4: Runtimes for the Concorde TSP solver



While the efficiency of the computation times for Concorde was impressive (it took only a little longer than 10 seconds on 26-stitch instances, which is over three times the maximum number of stitches for the brute-force backtracking algorithm), the results are approximations: the format of the input to PyConcorde required integer edge weights for non-Euclidean instances, so before passing the input to Concorde, the ceiling for each edge weight value was taken. Thus, the resulting tour length is at most $|G_R(E)|$ longer than necessary (where G_R is the final graph produced by the reduction).

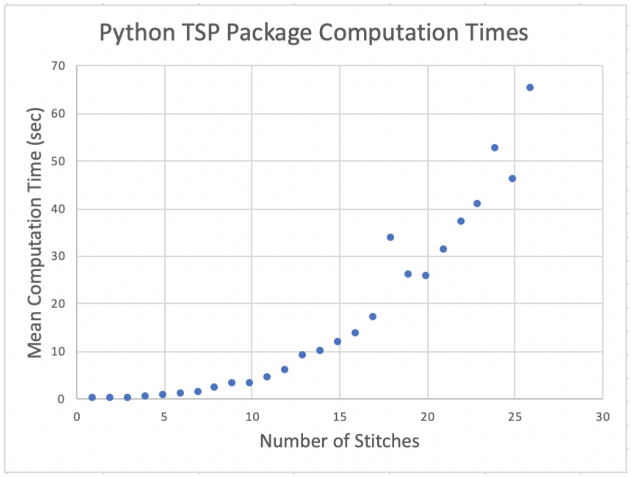
4.4 Python *tsp* Package

While Concorde is a great option due to its incredible efficiency, to get more precise results, we can use other popular TSP solvers. The *tsp* package is a Python implementation of the same branch and cut linear programming strategy [11].

Table 3: tsp *package computation times*

# Stitches	Mean	# Stitches	Mean
1	0.05	14	9.82
2	0.11	15	11.92
3	0.22	16	13.77
4	0.41	17	17.18
5	0.67	18	33.56
6	0.97	19	25.87
7	1.39	20	25.58
8	2.17	21	31.29
9	3.02	22	37.03
10	3.28	23	40.93
11	4.30	24	52.44
12	5.86	25	46.18
13	8.96	26	65.20

Figure 5: Runtimes for the tsp *package*



While it is slower than Concorde, we get more accurate tour lengths, with a decent runtime that is still substantially faster than the backtracking algorithm. As shown in *Table 3*, for 26 stitches, the *tsp* package solved embroidery-reduced instances in a little over a minute on average.

5 Approximation Algorithms

Embroidery is not a massively precise art; it's usually fine for thread to be longer than absolutely necessary, since cutting thread is easy (however, gracefully extending thread is not). This makes approximation algorithms very useful: for embroidery, it is more important that an algorithm can efficiently provide a reasonable upper bound of how much thread is needed, rather than slowly producing the exact required length.

5.1 Arkin *et al.* 2-Approximation

As mentioned, Arkin *et al.* provides a 2-approximation algorithm for graphs of arbitrary makeup [4]. The algorithm has two cases, depending on which of the two possible front edge configurations a given pattern has: either all front stitches are connected, or they are not.

For a connected component, we find the minimum set of back edges such that for each vertex in G(V, E, B), $deg_E(v) = deg_B(v)$, which is equivalent to the b-matching problem on the complete graph where $b(v) = deg_E(v)$. Anstee has a polynomial time algorithm that solves the b-matching problem exactly [2], however; Anstee's algorithm is arduous to implement. Instead, we can approximate the necessary length of thread for connected components, by simply taking the same edges in the back graph as the front graph. This maintains the property that all vertices satisfy $deg_E(v) = deg_B(v)$, and the degree of each vertex is even, so there must exist an Euler tour with no consecutive front edges. Furthermore, this algorithm is a 2-approximation: the optimal stitching for a connected component contains at least the front edges, so $OPT \geq |E|$. Thus, the approximation $T_{approx} \leq 2 \cdot |E| \leq 2 \cdot OPT$.

In the case that the front edges do not form a connected graph, we cannot use the b-matching algorithm (or our approximation) for all instances, as the optimal solution might not connect each of the stitching components (which would violate the idea of using a single thread). Thus, to approximate disconnected components, we take the minimum spanning tree (MST) between all of the connected components in G. We then initialize the back edges B to the set of edges containing two copies of each edge in the MST and a single copy of the front edges E. We can now approximate the length of the tour, given by: $T_{approx} = E \cup B$. Thus the length of T_{approx} is $|T_{approx}| = 2|E| + 2|MST|$.

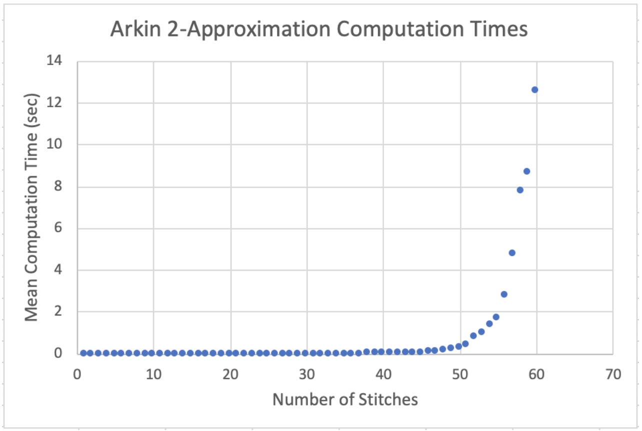
Now, we can modify T_{approx} to decrease its length. $G(V, T_{approx})$ must be connected (due to the MSTs in T_{approx}) as well as have even degrees for each $deg_T(v)$ by our construction, and since for all v in V, $deg_T(v) \geq 2 \cdot deg_F(v)$, there must exist an Euler tour with no consecutive front edges. However, that doesn't mean there won't be consecutive back edges: to avoid consecutive back edges, we can use the triangle inequality to eliminate them without increasing $|T_{approx}|$. As in, say T_{approx} contains two edges in B $e_1(v_a, v_b)$ and $e_2(v_b, v_c)$, we can simply use the edge $e_3(v_a, v_c)$, which must exist since B is a complete graph. Note that shortcutting doesn't disconnect the tour, since we know that all front edges are used.

This shows that this modified T_{approx} is a 2-approximation for the optimal solution, as the optimal solution would have to be at a minimum, the weight of the front edges plus the MST (to ensure the graph is connected with the minimum distance possible). As in, $OPT \geq |E| + |MST|$. Thus, we can conclude that $T_{approx} \leq 2(|E| + |MST|) \leq 2OPT$.

Table 4: Arkin et al. *2-approx. computation times*

#	Mean	#	Mean	#	Mean
1	4e-05	21	0.00113	41	0.01969
2	8e-05	22	0.00108	42	0.02488
3	0.00023	23	0.00112	43	0.04107
4	0.00018	24	0.00142	44	0.03571
5	0.00026	25	0.0016	45	0.0694
6	0.00036	26	0.00166	46	0.1005
7	0.00037	27	0.00184	47	0.116
8	0.00048	28	0.00188	48	0.17339
9	0.00054	29	0.00207	49	0.24419
10	0.00067	30	0.00235	50	0.32175
11	0.00069	31	0.00357	51	0.4262
12	0.00081	32	0.00405	52	0.8143
13	0.0009	33	0.00346	53	0.98213
14	0.00104	34	0.00457	54	1.41139
15	0.00087	35	0.00445	55	1.72012
16	0.00101	36	0.00611	56	2.78791
17	0.00086	37	0.00845	57	4.74467
18	0.001	38	0.0101	58	7.80175
19	0.00101	39	0.01352	59	8.6776
20	0.00119	40	0.01394	60	12.59123

Figure 6: Runtimes for Arkin et al.*'s approximation*



The implementation for this polynomial time approximation algorithm can be found here. See *Table 4* for benchmarking.

The runtime of this algorithm is in polynomial time. We know this because there are only two routes the 2-approximation can take: the connected components or disconnected components approach, and both run in polynomial time. First consider the connected components: this is done after we identify the connected components in G, and then find there is only one. Identifying connected components in the front graph of G can be done using Depth First Search (DFS) on each vertex in G (if it has not been visited). DFS has a runtime of $O(|V| + |E|)$, and running this on each unvisited vertex in the graph will result in a worst case runtime of $|V| \cdot O(|V| + |E|) = O(|V|^2 + |V| \cdot |E|)$. Once a single component is identified, we can run the b-matching algorithm provided by Anstee, which has a polynomial runtime of $O((|V|log|V|)(|E|+|V|log|V|))+O(|V|^2|E|)$ [2]. For this implementation however, we approximate in $O(|E|)$ time, as we simply find the sum of the lengths of the front edges, and multiply by two.

The runtime of the multiple connected components is also polynomial: once again, we must identify connected components (which as per the explanation above, has a runtime of $O(|V|^2+|V|\cdot|E|)$). However, we now have to find the MST of the connected components. Prim's algorithm has a runtime of $O(|E|\cdot log|V|)$, which is polynomial on input G. All we have left is to find an alternating Eulerian Circuit in T_{approx}, which can be found in linear time in terms of the edges $O(|E(T_{approx})|)$. Shortcutting between consecutive back edges in the Euler tour can also be done in linear time post-tour selection. As such, the runtime of Arkin *et al.*'s 2-approximation for multiple connected components is also polynomial.

6 Results

The embroidery problem is very difficult to solve exactly, and given the need for efficiency and the leniency for precision, approximation algorithms are the most practical solutions. Arkin *et al.* provides a fairly lightweight and fast algorithm that can be produce solutions with little infrastructure (using the approximation, rather than b-matching, for connected components).

However, when more precision is required, reducing the embroidery problem to TSP provides more precise solutions (as shown in *Figure 8*, where the backtracking algorithm serves as a source of truth for for the precise amount of necessary thread), though it will take more time.

Figure 7: Runtime distributions for all four methods on the same embroidery instances, where measurements for each method ceased once the mean computation time exceeded 10 seconds.

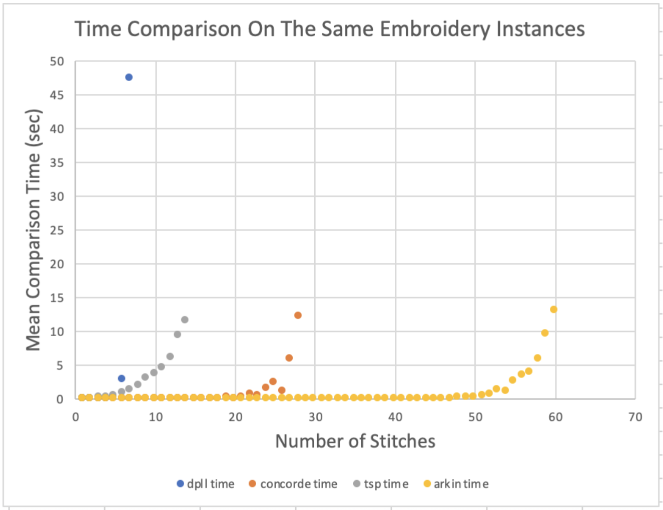
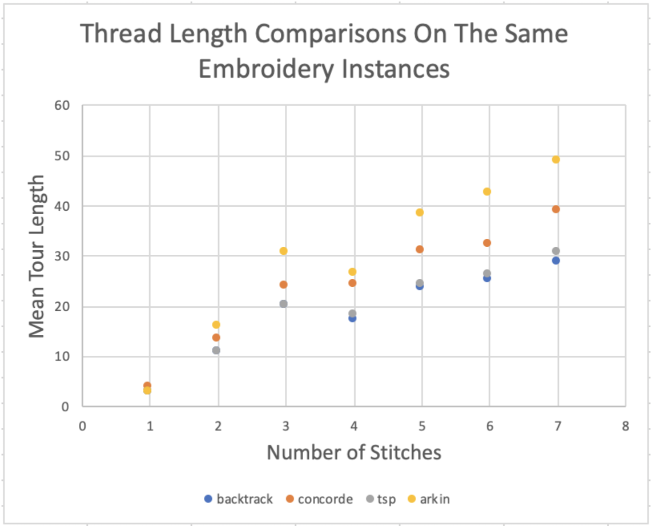


Figure 8: Mean tour lengths for all four methods on the same embroidery instances.



7 Limitations & Further Study

The range of data recorded in this paper is narrow. For one, this paper takes on the broad issue of arbitrary embroidery instances, as opposed to more specialized instances (i.e. segmented patterns as described by Arkin *et al.* [4], cross-stitching, etc). Furthermore, these experimental results were taken on limited hardware, thus restricting the embroidery instances to a small board (5x5 units is a very impractical size for cross stitching patterns). It may be useful to measure the computation times of more useful sizes of the embroidery problem, given enough time and resources.

In addition to limited data, the algorithms chosen for this paper are not necessarily exhaustive. One method that may be worthwhile to explore is Christofides' 3/2-approximation, which boasts a polynomial runtime and greater accuracy than any other approximation algorithm for TSP in P (excepting the recent developments in 2020). While it was not used in this paper because of concerns that reducing from the embroidery problem to TSP produces non-metric instances of TSP by necessity, it may be worthwhile to explore viable methods of transforming the embroidery TSP instance into a metric TSP instance.

8 Acknowledgements

The author would like to thank her former professor Josh Grochow for teaching the class that provided her with the concepts and frameworks to approach this problem, as well as CCCG's anonymous reviewers for their time and insightful feedback.

References

[1] E. Allender, P. Burgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the Complexity of Numerical Analysis. *SIAM J. Comput.*, 38(5), 1987–2006.

[2] R. P. Anstee. A polynomial algorithm for b-matchings: an alternative approach. *Information Processing Letters*, 24(3):153–157, 1987.

[3] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde TSP solver. 2006.

[4] E. M. Arkin, G. W. Hart, J. Kim, I. Kostitsyna, J. S. B. Mitchell, G. R. Sabhnani, and S. S. Skiena. The Embroidery Problem. *Proceedings of the 20th Annual Canadian Conference on Computational Geometry*, Montreal, Canada, August 13-15, 2008.

[5] T. Biedl, J. D. Horton, and A. Lopez-Ortiz. Cross-stitching using little thread. *In 17th Canadian Conference of Computational Geometry*, pages 196-199, 2005.

[6] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Tech. Report*, GSIA, Carnegie Mellon Univ., 1976.

[7] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a Large-Scale Traveling-Salesman Problem. *Operations Research 2*, pages 393–410, 1954.

[8] A. Karlin, N. Klein, S. O. Gharan. A (Slightly) Improved Approximation Algorithm for Metric TSP. arXiv preprint arXiv:2007.01409, 2020.

[9] C. H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[10] S. Tsutomu. tsp. 2020. https://pypi.org/project/tsp [Online; accessed Nov. 2020]

[11] J. Vankerschaver. PyConcorde. 2018. https://github.com/jvkersch/pyconcorde [Online; accessed Nov. 2020]

Another Small but Long Step for Crossing Numbers: $\mathrm{cr}(13) = 225$ and $\mathrm{cr}(14) = 315$

Oswin Aichholzer*

Abstract

In this paper we consider the crossing number of simple drawings of complete graphs. Following the iterative enumeration approach developed in [3] we report on a heavily computer assisted proof that the crossing number of the complete graph K_{13} is $\mathrm{cr}(13) = 225$. This implies that $\mathrm{cr}(14) = 315$.

1 Introduction

In this paper we report on the long-term application of methods developed in the work of Ábrego at al [3] to the computation of the crossing number of the complete graph K_{13}, namely to show that $\mathrm{cr}(13) = 225$ and $\mathrm{cr}(14) = 315$. The programs we used have been developed and implemented in the course of [3] and were later refined for full efficiency on high performance clusters. The computational power needed to perform all calculations sums up to the equivalent of over 1000 years on a single CPU.

Our approach is based on the representation of weak isomorphism classes of simple drawings of the complete graph by rotation systems. In the following subsections we will provide the necessary background on simple drawings, crossing numbers, and rotation systems.

In Section 2 we briefly describe the iterative enumeration approach developed in [3] and apply it then to the specific case of K_{13} in Section 3. We close with a short discussion on possible extensions and alternative approaches in Section 4.

1.1 Basics

In a *simple drawing* of a graph in the plane the vertices are represented by points and the edges are represented by non-self-intersecting Jordan arcs connecting two points which represent vertices. These arcs must not contain any other point representing a vertex in their interior, two edges meet at most once, either in a proper crossing (no tangency) or a common endpoint, and no three edges intersect in a common crossing. Such drawings are sometimes also called *good drawings* or *simple topological graphs*.

*Institute of Software Technology, Graz University of Technology, Austria, `oaich@ist.tugraz.at`

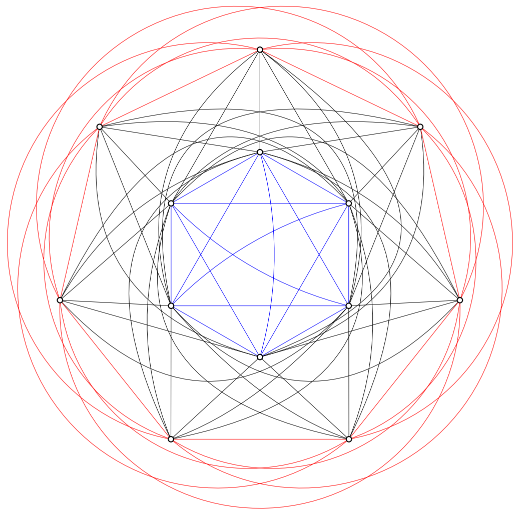


Figure 1: Crossing optimal cylindrical drawing of K_{13} with 225 crossings.

In this work we are especially interested in simple drawings of the complete graph K_n on n vertices. To simplify the presentation we will thus limit the following definitions to this setting. The *rotation* of a vertex v in a simple drawing is the clockwise cyclic order of edges incident to v, represented as a sequence of the opposite vertices of all edges incident to v. The *rotation system* of a simple drawing is the set of rotations of all its vertices. Two such rotation systems are equivalent if one can be obtained from the other by relabeling the vertices and optionally inverting all rotations. Rotation systems obtained from simple drawings are called *realizable* (by this simple drawing).

Simple drawings with the same pairs of crossing edges form the so-called weak isomorphism classes. Rotation systems can be used to represent these classes. This is based on the following two results: (1) The rotation system of a simple drawing of the complete graph determines the pairs of crossing edges [18]. (2) The set of crossing pairs of edges determines the equivalence class of the rotation system of a simple drawing of the complete graph [9].

Consequently we do not need to generate all drawings of K_n to find the minimal possible number of crossings, but can restrict our considerations to different rotation systems. This way we have a simple construct to han-

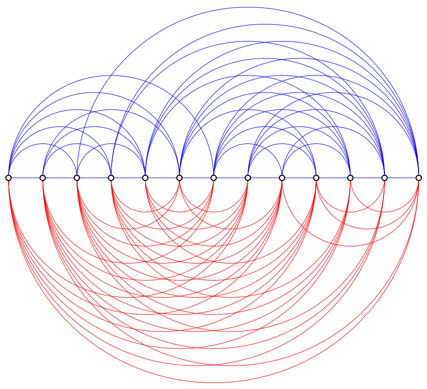


Figure 2: Crossing optimal 2-page book drawing of K_{13} with 225 crossings.

dle the combinatorial task and can save an exponential factor, as there might be an exponential number of drawings for a given rotation system.

1.2 Crossings of the Complete Graph

There are many different definitions of the crossing number of a drawing of a graph, see for example [17] and [20]. We use the following setting. For every intersection point of (at least two) edges in a drawing (which is not a common vertex) we count one crossing for every pair of edges that intersects there. For this definition it is well known and easy to see that for any drawing of a graph that minimizes the number of crossings there is a simple drawing with the same number of crossings. In other words, if we are interested in minimizing the number of crossings, it is sufficient to consider simple drawings.

The crossing number of a graph is defined as the minimum number of crossings over all (simple) drawings of this graph. For the complete graph K_n on n vertices we denote the crossing number of K_n by $\mathrm{cr}(n)$.

For the complete graph the Harary-Hill conjecture states that the number of crossings in any drawing of K_n is at least $H(n) = \frac{1}{4}\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{n-2}{2}\rfloor\lfloor\frac{n-3}{2}\rfloor$, that is, $\mathrm{cr}(n) \geq H(n)$. There exist several types of drawings which obtain this number of crossings, for example cylindrical drawings and 2-page book drawings, and consequently we know that $\mathrm{cr}(n) \leq H(n)$. Figures 1 and 2 show these drawings for $n = 13$ points with $H(13) = 225$ crossings. See the nice survey of Beineke and Wilson [8] for the history of this conjecture and the related constructions.

For many classes of simple drawings the Harary-Hill conjecture is known to be true. This has been shown for 2-page book drawings [2], monotone drawings [4], and more generally for so-called shellable [5] and bishellable [1] drawings, where these classes also include for example cylindrical drawings.

For general simple drawings so far the Harary-Hill conjecture has been confirmed for $n \leq 12$. Already in 1972 Guy [10] showed that it holds for $n \leq 10$. For $n \leq 8$ the results can be obtained with reasonable effort by hand, but for $n \geq 9$ the proof used an algorithmic approach. Consequently in 2015 McQuillan and Richter provided a computer-free proof for $\mathrm{cr}(9) = 36$ [15]. Using computers, Pan and Richter [19] proved that the crossing number of K_{11} is $\mathrm{cr}(11) = H(11) = 100$, which implies that $\mathrm{cr}(12) = H(12) = 150$.

The later step follows from the following folklore observation. To obtain a relation between $\mathrm{cr}(n)$ and $\mathrm{cr}(n-1)$ observe that every $n-1$ subset of K_n has at least $\mathrm{cr}(n-1)$ crossings. If we sum over all n such subsets, every crossing is counted $n-4$ times. This gives the relation

$$\mathrm{cr}(n) \geq \frac{n}{n-4}\,\mathrm{cr}(n-1) \qquad (1)$$

Plugging this into $H(n)$ it follows that if $\mathrm{cr}(n) \geq H(n)$ is true for some odd n, then also $\mathrm{cr}(n+1) \geq H(n+1)$ is true. This step does not work from even to odd, but for K_{13} at least it follows that $\mathrm{cr}(13) \geq 217 \geq \frac{13}{9}150$.

Kleitman [11, Section 5] proved that any two simple drawings of K_n, where n is odd, have crossing numbers that have the same parity[1]. We will refer to this as the parity property of drawings of K_n for n odd.

From this parity property it follows that the crossing number of any simple drawing of K_{13} must be odd, as it must have the same parity as $\binom{13}{4} = 715$, which is the number of crossings in a straight line drawing of 13 points in convex position. This implies $\mathrm{cr}(13) \in \{217, 219, 221, 223, 225\}$, and as a first step it was shown that no drawing with only 217 crossings exists, that is, $\mathrm{cr}(13) \geq 219$ [14].

The method of extending rotation systems – see the next section for details – has already successfully been used in [3] to show that $\mathrm{cr}(13) \geq 223$. In our work we close the gap and report that no drawing of K_{13} with 223 crossing exists, that is, $\mathrm{cr}(13) = H(13) = 225$. Using (1) this implies that $\mathrm{cr}(14) = H(14) = 315$.

Theorem 1 *The crossing number of K_{13} is $\mathrm{cr}(13) = H(13) = 225$ and the crossing number of K_{14} is $\mathrm{cr}(14) = H(14) = 315$.*

2 Extending Rotation Systems

In [3] it is described how all different realizable rotation systems of given cardinality n can be generated. This is equivalent to generate all weak isomorphism classes of simple drawings with n vertices. The basic idea is to extend a rotation system of size $n-1$ by adding an

[1]Note that actually Kleitman shows this result for $K_{m,n}$ where m and n are both odd, but the argument for K_n is precisely the same.

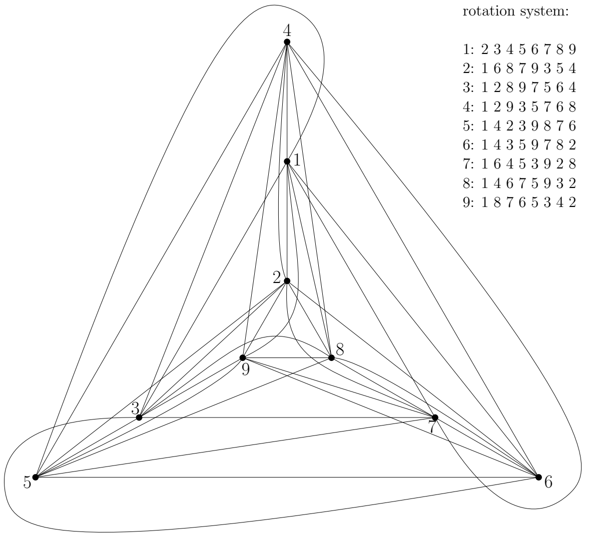
additional vertex and all its incident edges in all possible ways. With rotation systems this can essentially be done by inserting the label of the new vertex in all possible positions between the labels of the existing rotation around each vertex. In this way the rotation around the new vertex gets determined, and can be used to check if all 4-tuples and 5-tuples in the extension are valid; see [3] for full details.

Kynčl [13] has designed a polynomial time algorithm to check simple realizability of abstract topological graphs. These are graphs where in addition to the graph the set of pairs of its edges which cross is given. From the discussion in the introduction of this paper it follows, that (abstract) rotation systems are realizable if and only if all its K_4 and K_5 subgraphs are realizable. So for a given (abstract) rotation system, it can be trivially checked in $O(n^5)$ time if it can be realized as a simple drawing or not. Note that this is in sharp contrast to the geometric case. There drawings for complete graphs can only be extended in an abstract way as described for example in [6]. The main reason is that realizing abstract order types is a notoriously hard problem which is know to be $\exists\mathbb{R}$-hard [16].

For us this implies that for each generated rotation system we check that every 4- and 5-tuple is realizable. For every 4-tuple this is actually guaranteed by the way the extension takes place, as for a given 4-tuple the order around one of the vertices is fixed by the rotation of the other three vertices. For all 5-tuples this test can be done in $O(n^4)$ time, as only those $O(n^4)$ 5-tuples which include the new vertex need to be checked. Here we assume that the rotation system we extend is realizable, which is granted by induction and the data base we start with: In [3] all 7 198 391 729 different realizable rotation systems of 9 vertices have been generated. This will serve as the base of our iterative approach.

Note that the number of weak isomorphism classes of K_n is in $2^{n^2\alpha(n)^{O(1)}}$ [12] and $2^{\Omega(n^2)}$ [18]. This prevents a computation of all realizable rotation systems for $n > 9$. Thus our approach will only generate rotation systems which are relevant for us, as explained below.

A natural idea to speed up the extension of rotation systems would be to show that every crossing minimal drawing of K_n contains a subdrawing of K_{n-1} which is also crossing minimal. If this so-called subset property would be true, then it would be sufficient to only extend rotation systems for crossing minimal drawings. In the next section (cf. Table 1) we will see that for the extension from 12 vertices to 13 vertices we need to consider 56 043 781 rotation systems. On the other hand for $n = 12$ there are only 2 592 crossing minimal sets. So this subset-property would reduce the effort to less than $\frac{1}{20000}$ of the current work load. Actually, if the Harary-Hill conjecture holds then it follows from (1) that for n even for any crossing minimal drawing of K_n every subdrawing of K_{n-1} is also crossing minimal.



rotation system:

1: 2 3 4 5 6 7 8 9
2: 1 6 8 7 9 3 5 4
3: 1 2 8 9 7 5 6 4
4: 1 2 9 3 5 7 6 8
5: 1 4 2 3 9 8 7 6
6: 1 4 3 5 9 7 8 2
7: 1 6 4 5 3 9 2 8
8: 1 4 6 7 5 9 3 2
9: 1 8 7 6 5 3 4 2

Figure 3: Crossing optimal drawing for K_9 with 36 crossings where every subdrawing of a K_8 has 20 crossings.

Unfortunately this subset property is not true for n odd. An example for $n = 9$ can be seen in Figure 3. There every subdrawing of K_8 has 20 crossings, whereas an optimal drawing of K_8 has only 18 crossings. Let us remark that out of 421 crossing minimal rotation systems for K_9 there are only 4 which do not contain a subset of size 8 with 18 or 19 crossings. And the depicted example is the only one where every subset of size 5 has only 1 or 3 crossings, but it does not contain a subdrawing of K_5 with 5 crossings. Intuitively this means that every subdrawing is not too bad, and crossings are evenly distributed. But consequently there is also no really good subdrawing, that is, no subdrawing of K_8 with the minimum possible number of crossings.

3 The computation for $\mathrm{cr}(13) = 225$

There exist drawings of K_{13} with 225 crossings (see e.g. Figures 1 and 2) and we know from the computations in [3] that $\mathrm{cr}(13) \geq 223$. Moreover, by the above mentioned parity property of Kleitman [11] we therefore know that $\mathrm{cr}(13) \in \{223, 225\}$. To prove our main result $\mathrm{cr}(13) = 225$ it is thus sufficient to show that no drawing of K_{13} with 223 crossings exist. For this task we use the iterative approach described in the previous section, starting with realizable rotation systems of size 9 obtained in [3].

From (1) it follows that a drawing D with n vertices and $\mathrm{cr}(D)$ crossings must have a subdrawing D' of $n-1$ vertices with at most $\mathrm{cr}(D') \leq \lfloor \frac{n-4}{n}\,\mathrm{cr}(D) \rfloor$ crossings. If we want to obtain a rotation system which provides a drawing of K_{13} with 223 crossings, we know

cr(9)	#	cr(10)	#	cr(11)	#	cr(12)	#	cr(13)
36	421	60	37					
$\hookrightarrow$		61	516					
$\hookrightarrow$		62	4 407					
$\hookrightarrow$		63	22 007	100	403 079	150	2 592	217
$\hookrightarrow$		$\hookrightarrow$		$\hookrightarrow$		151	73 014	219
38	4 790	64	75 159	102	6 678 654	153	8 137 376	221
$\hookrightarrow$		$\hookrightarrow$		$\hookrightarrow$		154	46 850 304	223
≤ 38	5 211	≤ 64	102 126	≤ 102	7 081 733	≤ 154	56 043 781	< 225

Table 1: The road from $n = 9$ to $\mathrm{cr}(13) < 225$: To obtain all rotation systems for a given size and fixed crossing number, we first need to generate all rotation systems with the parameters given in the table to the left of it (smaller cardinality) and in the same line or above it (number of crossings according to $\mathrm{cr}(n-1) \leq \lfloor \frac{n-4}{n} \mathrm{cr}(n) \rfloor$). The $\hookrightarrow$ indicates that for a given line it is sufficient to consider the next crossing number above.

that it contains a sub-drawing of K_{12} with at most $154 = \lfloor \frac{13-4}{13} 223 \rfloor$ crossings. It is thus sufficient to only extend rotation systems of size 12 with 150 to 154 crossings. This step can be iterated. So to generate all relevant rotation systems of size 12, we have to extend all rotation systems of size 11 with at most $102 = \lfloor \frac{12-4}{12} 154 \rfloor$ crossings. By the parity property we get that these drawings have either 100 or 102 crossings. Repeating this process, Table 1 gives the complete picture of all steps from 9 to 13 vertices.

As a warm up example consider the task of showing that $\mathrm{cr}(13) > 217$. To this end it is sufficient to start with all 421 rotation systems of size 9 with 36 crossings. In the next three extension steps we only keep generated rotation systems with a limited number of crossings, that is, for $n = 10, 11, 12$ sets with ≤ 63, 100, and 150 crossings, respectively. In the last step we have to extend the 2592 rotation systems of size 12, and the result is that no rotation system of size 13 with 217 crossings exists. In total there are 433 059 sets which are extended in this process. The whole computation for this task needed 29.25 days on 12 CPUs in parallel, which is equivalent to approx. 0.96 years on a single CPU. This has to be compared to the time reported in [14] where the computation took over 500 hours on 256 CPUs in parallel, equivalent to approx. 14.6 years on a single CPU. This shows that using non isomorphic rotation systems instead of non isomorphic drawings reduces the computational effort significantly.

To show that $\mathrm{cr}(13) > 223$ it can bee seen from Table 1 that we start by extending 5 211 rotation systems of size 9. This has to be seen in contrast to the 7 198 391 729 realizable rotation systems that exist for this cardinality. So we need to extend only about 0.0000724% of the possible rotation systems, which explains why this partial extension is feasible, while a complete generation of all rotation systems for size 10 seems to be unrealistic within a reasonable time bound.

The computations started in 2013 [3] and have been running till the beginning of 2021 on various computers in the background, using their idle time. Table 2 summarizes the time needed for the extension from 12 vertices to 13 vertices. The most intensive part – the extension from $n = 12$ with 154 crossings – ended in September 2020. No rotation system of 13 vertices with 223 crossings has been generated. The last computation we started was to extend from $n = 12$ with 153 crossings, and this was done in February 2021. Again, no rotation systems for K_{13} with 223 crossings have been found, showing that no simple drawings of K_{13} with 223 crossings exist. In total the extension from $n = 12$ to $n = 13$ needed the equivalent of over 1000 years on a single CPU. The computations for smaller cardinality took significantly less time, and have partially been performed before this project. As various computer systems were used, the needed time has to be compared with care. The following list gives a rough approximation: $n = 9 \rightarrow n = 10$: 11 minutes; $n = 10 \rightarrow n = 11$: 12.5 days; $n = 11 \rightarrow n = 12$: 15.6 years. All times are again given as the equivalent on a single CPU.

To compare the last step for the extension $n = 12 \rightarrow n = 13$ with geometric drawings, that is, drawings were edges are straight line segments, it is worth to mention that from Table 2 it can be seen for $n = 12$ that there are 8 137 376 different rotation systems with 153 crossings. For the geometric case the minimum number of crossings of K_{12} is 153, and there is only one unique order type for this. That means that the remaining 8 137 375 rotation systems, as well as all with 150 or 151 crossings, can not be stretched (that is, can not be drawn with straight lines).

As already mentioned, from (1) it follows that the crossing number for odd cardinality implies the crossing number for the next larger even cardinality. Consequently we get $\mathrm{cr}(14) \geq \frac{14}{10} 225 = 315$. As there exist simple drawings of K_{14} which have 315 crossings, we in addition obtain $\mathrm{cr}(14) = 315$.

A natural question is whether we could compute all rotation systems for $n = 13$ with 225 crossings. To this end we would need all rotation systems for $n = 12$ with

No. of crossings	No. of input rot. systems	time in days	time in years
150	2 592	22.7	0.1
151	73 014	564.4	1.5
152	980 495	6 805.7	18.6
153	8 137 376	53 284.8	146.0
154	46 850 304	320 493.4	878.1
$\leq$ 154	56 043 781	381 171.0	1 044.3

Table 2: Computation time needed for extension from $n = 12$ to $n = 13$. Given is the equivalent if computations would have been performed on a single CPU.

Number of crossings	Number of all rotation systems	Number of convex rotation systems
150	2 592	2 592
151	73 014	73 014
152	980 495	914 882
153	8 137 376	6 592 493
154	46 850 304	30 355 238
155	?	94 110 607
156	?	203 940 546
157	?	327 333 179
158	?	446 426 769
$\leq$ 158	?	1 109 749 320

Table 3: General and convex realizable rotation systems for $n = 12$ with up to 158 crossings.

up to $155 = \lfloor \frac{9}{13} 225 \rfloor$ crossings. To obtain these rotation systems it would still be sufficient for $n = 11$ to consider only sets with at most 102 crossings. But given the time needed to compute $\text{cr}(13) > 223$ this is currently out of reach. So far we extended all rotations systems for $n = 12$ with 150 and 151 crossings, and partially those with 152 crossings. In this way we already obtained over 114 million different realizable rotation systems for $n = 13$ with 225 crossings.

4 Conclusion

Based on exhaustive extension of rotation systems we showed that $\text{cr}(13) = 225$ which implies that $\text{cr}(14) = 315$. But how about using this approach to determine cr(15), and thus also cr(16)? This seems to be illusive. Even computing all crossing optimal rotation systems for $n = 13$ is currently out of reach, as argued above.

Arroyo et al [7, Question 6.6] conjectured that crossing minimal drawings of K_n are always convex drawings. These are drawings where for every three-cycle in the drawing there is a side, such that the edge connecting any two vertices on that side is also entirely on this side; see [7] for a formal definition. Essentially that means that out of the 5 possible simple drawings of K_5 only three types are used, namely the three types which are also realizable as a straight line drawing. With our computations we checked this conjecture to be true for up to size $n = 12$, which also implies that all the crossing minimal drawings given in this paper belong to that class. So how about checking the crossing number conjecture under the assumption that the convexity conjecture is true? To this end we computed all convex sets of size 12 with small crossing numbers, more precisely with at most 158 crossings, as this would be the bound needed to extend the rotation systems to check cr(15). Table 3 shows how many such rotation systems for $n = 12$ exist. Unfortunately it turned out that most sets with few crossings are in fact convex, so the necessary work load still is way beyond being reasonable even under the assumption of the convexity conjecture.

A downside of our approach is that a positive result in the sense that a rotation system with a certain crossing number exists, can easily be checked by providing a drawing realizing it. If no rotation system is generated, this can only be checked by repeating the computations with an alternative implementation. But the needed time makes this approach unrealistic. Moreover, the induction base we used, that is, all realizable rotation systems of size 9, needs about 430 GB of disc space. So providing this for download is also not realistic. However, we are happy to provide intermediate files on request, and also the whole data base for $n = 9$, e.g. via different media.

Considering these limitations, we expect that our work will stimulate alternative approaches to show that $\text{cr}(13) = 225$ in a less time consuming way.

Acknowledgments. We would like to acknowledge the use of HPC resources provided by the ZID of Graz University of Technology. We are especially grateful to Mario Lang and Simon Kainz for their continuous support with these resources. Moreover we want to thank all coauthors of [3] for many fruitful discussions on this long-term project and the anonymous referees for helpful remarks to improve the presentation of this work. Research partially supported by FWF grant W1230.

References

[1] Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Dan McQuillan, Bojan Mohar, Petra Mutzel, Pedro Ramos, R. Bruce Richter, and Birgit Vogtenhuber. Bishellable drawings of K_n. *SIAM Journal on Discrete Mathematics*, 32(4):2482–2492, 2018. https://doi.org/10.1137/17M1147974

[2] Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Pedro Ramos, and Gelasio Salazar. The 2-page crossing number of K_n. *Discrete & Computational Geometry*, 49(4):747–777, 2013.

[3] Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Thomas Hackl, Jürgen Pam-

mer, Alexander Pilz, Pedro Ramos, Gelasio Salazar, and Birgit Vogtenhuber. All Good Drawings of Small Complete Graphs. In *Proc. 31st European Workshop on Computational Geometry EuroCG '15*, pages 57–60, Ljubljana, Slovenia, 2015.

[4] Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Pedro Ramos, and Gelasio Salazar. More on the crossing number of K_n: Monotone drawings. *Electronic Notes in Discrete Mathematics*, 44:411–414, 2013. Special issue dedicated to LAGOS2013. http://dx.doi.org/10.1016/j.endm.2013.10.064

[5] Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Pedro Ramos, and Gelasio Salazar. Shellable drawings and the cylindrical crossing number of K_n. *Discrete and Computational Geometry*, 52:743–753, 2014. https://doi.org/10.1007/s00454-014-9635-0

[6] Oswin Aichholzer and Hannes Krasser. Abstract Order Type Extension and New Results on the Rectilinear Crossing Number. *Computational Geometry: Theory and Applications, Special Issue on the 21st European Workshop on Computational Geometry*, 36(1):2–15, 2006.

[7] Alan Arroyo, Dan McQuillan, R. Bruce Richter, Gelasio Salazar. Convex drawings of the complete graph: topology meets geometry. *arXiv preprint arXiv:1712.06380, 2017*

[8] Lowell Beineke and Robin Wilson. The early history of the brick factory problem. *The Mathematical Intelligencer*, 32(2):41–48, 10 2010. https://doi.org/10.1007/s00283-009-9120-4

[9] Emeric Gioan. Complete graph drawings up to triangle mutations. In *Graph-Theoretic Concepts in Computer Science*, pages 139–150, 2005.

[10] Richard K. Guy. Crossing numbers of graphs. In Y. Alavi, D. R. Lick, and A. T. White, editors, *Graph Theory and Applications*, pages 111–124, Berlin, Heidelberg, 1972. Springer Berlin Heidelberg.

[11] Daniel J. Kleitman. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory*, 9(4):315 – 323, 1970. URL: https://doi.org/10.1016/S0021-9800(70)80087-4

[12] Jan Kynčl. Improved enumeration of simple topological graphs. *Discrete & Computational Geometry*, 50(3):727–770, 2013. http://dx.doi.org/10.1007/s00454-013-9535-8

[13] Jan Kynčl. Simple realizability of complete abstract topological graphs simplified. *Discrete & Computational Geometry*, 64:1–27, 2020. https://doi.org/10.1007/s00454-020-00204-0

[14] Dan McQuillan, Shengjun Pan, and R. Bruce Richter. On the crossing number of K_{13}. *Journal of Combinatorial Theory, Series B*, 115(C):224–235, November 2015. https://doi.org/10.1016/j.jctb.2015.06.002

[15] Dan Mcquillan and R. Bruce Richter. On the crossing number of K_n without computer assistance. *Journal of Graph Theory*, 82:387–432, 10 2015. https://doi.org/10.1002/jgt.21908

[16] Nikolai Mnëv. On manifolds of combinatorial types of projective configurations and convex polyhedra. In *Soviet Math. Doklady*, volume 32, pages 335–337, 1985.

[17] János Pach and Géza Tóth. Which crossing number is it anyway? *Journal of Combinatorial Theory, Series B*, 80(2):225–246, 2000. http://dx.doi.org/10.1006/jctb.2000.1978,

[18] János Pach and Géza Tóth. How many ways can one draw a graph? *Combinatorica*, 26(5):559–576, 2006. http://dx.doi.org/10.1007/s00493-006-0032-z

[19] Shengjun Pan and R. Bruce Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56(2):128–134, 2007. http://dx.doi.org/10.1002/jgt.20249,

[20] Marcus Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics*, 20, 04 2013. https://doi.org/10.37236/2713

Generalized LR-Drawings of Trees

Therese Biedl[*] Giuseppe Liotta[†] Jayson Lynch [*] Fabrizio Montecchiani[†]

Abstract

The *LR-drawing-method* is a method of drawing an ordered rooted binary tree based on drawing one root-to-leaf path on a vertical line and attaching recursively obtained drawings of the subtrees on the left and right. In this paper, we study how to generalize this drawing-method to trees of higher arity. We first prove that (with some careful modifications) the proof of existence of a special root-to-leaf path transfers to trees of higher arity. Then we use such paths to obtain *generalized* LR-drawings of trees of arbitrary arity.

1 Introduction

Tree-drawing is a very popular topic in the graph drawing literature. Nearly all tree-drawing methods required that the drawing is *planar* (has no crossing), but there are many other variations, depending on whether we demand that the drawing is *straight-line* (as opposed to permitting bends in edges) and/or *order-preserving* (a given order of edges at each node is maintained). For a rooted tree, one also distinguishes by whether the drawing must be *(strictly) upward* (nodes are (strictly) above their descendants). In all our drawings, we assume that nodes (and also bends, if there are any) are placed at *grid-points*, i.e., points with integer coordinates. The main objective is to obtain drawings of small *area*, measured via the number of grid-points in the minimum enclosing bounding box of the drawing. Sometimes one also considers the *width* and *height* of the drawing, measured by the number of columns (respectively rows) that intersect the drawing. We refer to a survey by Di Battista and Frati [6] for many results up to 2014, and to Chan's recent paper [3] for some development since.

In 2002, Chan [2] published a tree-drawing paper that became the inspiration for many follow-up papers. In particular, he studied rooted trees and only considered *ideal drawings*, i.e., drawings that respect all four of the above constraints (planar, straight-line, order-preserving and strictly upward). His area-results were superseded by later improvements [1, 3, 9], but the techniques introduced in [2] are still widely useful; see e.g. a recent paper by Frati, Patrignani and Roselli [8] that uses Chan's recursive approaches to obtain small straight-line drawings of outer-planar graphs.

Background and related results. One of the methods proposed by Chan [2] is the one that creates *LR-drawings*. The idea is to take a root-to-leaf path π, drawing it vertically, and attach the left and right subtrees of the path just below their parent, using a recursively obtained drawing for the subtree. See Figure 1, where the nodes of π are white (as in all other figures). These drawings are defined only for binary trees.

To describe this precisely, we need a few definitions. Let T be a rooted binary tree that comes with a fixed order of children at each node (we call this an *ordered rooted binary tree*). A *root-to-leaf* path $\pi = \langle v_1, v_2, \ldots, v_\ell \rangle$ is a path in T where v_1 is the root and v_ℓ is a leaf. A *left* subtree of π is a subtree rooted at some child c of a vertex $v_i \in \pi$ for which c comes before the path-child in the order at v_i. We call this a *left subtree at* v_i when needing to specify the node of π. A *right* subtree is defined symmetrically.

The *LR-drawing-method* consists of picking a root-to-leaf path π, drawing it vertically, and attaching (recursively obtained) drawings of the subtrees of $T \setminus V(\pi)$ on the left and right side so that the order is maintained. Since T is binary, there is only one such subtree at each $v \in \pi$; we place its drawing in the rows just below v (after lengthening edges of π as needed) and one unit to the left/right of path π.

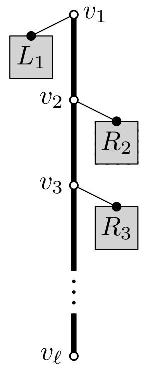


Figure 1: An LR-drawing where $\pi = \langle v_1, v_2, \ldots v_\ell \rangle$ and L_i and R_i are various subtrees.

Let $W(n)$ be the maximum (over all binary trees T with n nodes) of the width of the drawing, and note

[*]David R. Cheriton School of Computer Science, University of Waterloo, {biedl,jayson.lynch}@uwaterloo.ca

[†]Department of Engineering, University of Perugia, {giuseppe.liotta,fabrizio.montecchiani}@unipg.it

that it observes the following recursion:

$$W(n) \leq 1 + \min_{\pi} \left(\max_{\alpha} W(|\alpha|) + \max_{\beta} W(|\beta|) \right) \quad (1)$$

where α and β are any left and right subtree, respectively. We are therefore interested in picking a path π that has useful bounds on the size of α and β. Chan showed that there exists a path such that $|\alpha|+|\beta| \leq n/2$, for any left and right subtrees α and β. He then improved this to the following:

Lemma 1 *[2] Let* $p = 0.48$. *Given any binary ordered rooted tree T of size n, there exists a root-to-leaf path π such that for any left subtree α and any right subtree β of π, $|\alpha|^p + |\beta|^p \leq (1-\delta)n^p$ for some constant $\delta > 0$.*

Lemma 1, together with Eq. (1), and an inductive argument, implies the existence of LR-drawings of width $O(n^{0.48})$ (and the height is n, as it is for all LR-drawings constructed as described above). Later on, Frati et al. [8] showed that for some binary trees, a width of $\Omega(n^{0.418})$ is required in any LR-drawing, and this was improved further to $\Omega(n^{0.429})$ by Chan and Huang [4]. The latter paper also gave another construction-method that does not follow the above method exactly, instead the chosen path π may have some non-vertical edges while the vertically drawn path continues in some subtree of π. In this way, they can obtain drawings of width $O(n^{0.437})$.

As should be clear from the above lower bounds, LR-drawings are not the best tool for small-area ideal tree-drawings, since other papers can achieve width $O(\log n)$ (or better if the pathwidth is small) [1, 9] while keeping the height at n. But LR-drawings have a number of other appealing features:

- Drawings of disjoint rooted subtrees are "vertically separated", i.e., if T_v and T_w are two disjoint rooted subtrees, then there exists some horizontal line that separates the drawings of T_v and T_w. (This can be seen by studying the construction at the lowest common ancestor of the two trees.) As such, the drawings are perhaps easier to understand than drawings created with other methods (such as [9]) that delay the drawing of a subtree until further down, leading to 'interleaved' drawings of subtrees.

- LR-drawings (and in particular Lemma 1) has been used for a number of graph drawing results, including for octagonal drawings, orthogonal drawings, and drawings of outer-planar graphs [5, 7, 10].

- Last but not least, "the question on LR-drawings is still interesting and natural, as it is fundamentally about combinatorics of trees, or more specifically, decompositions of trees via path separators" [4].

Contribution. Our interest in LR-drawings originally came from the need to generalize Lemma 1 to rooted trees with higher *arity*, i.e., maximum number of children at a node. (As we will detail in a separate, forthcoming, paper, such a lemma for ternary trees can be used to obtain drawings of outer-1-planar graphs with small area.) In the process, we discovered that all the results and applications of LR-drawings seem to be only concerned with *binary trees*. It is not even clear what exactly an LR-drawing should be for trees of higher arity, and no area-bounds are known. Our results in this paper are as follows:

- We first show (in Lemma 2 in Section 2) that Lemma 1 holds for trees of arbitrary arity.

  There does not seem to be an easy way to derive this result from Chan's result, since it is not clear how we could modify a rooted tree T into a binary tree without either increasing the number of nodes or missing a subtree that may be too big. For this reason, we re-prove the result from scratch. The proof is similar in structure to the one by Chan, but we need to be much more careful in defining the inequalities that hold if we can extend the path to a subtree.

- We then discuss in Section 3 what the appropriate generalization of LR-drawings to trees of higher arity should be. We also give a simple construction that shows that ideal LR-drawings of area $O(n^2)$ always exist.

- In Section 3.1 and 3.2, we then give constructions for generalized LR-drawings that are directly based on Lemma 2 and therefore achieve $O(n^{0.48})$ width. Unfortunately, neither construction gives ideal drawings: the first one has one bend per edge, and the second one is not upward. Both constructions can be modified to achieve ideal drawings, but at the expense of increasing the height (possibly more than polynomially).

- In Section 3.3, we give a construction for generalized LR-drawings that are ideal drawings. The price to pay is that the construction is more complicated, and the height (which was linear in the previous constructions) increases to $O(n^{1.48})$, meaning that the area is only just barely sub-quadratic, namely $O(n^{1.96})$.

We end in Section 4 with open questions.

2 Choosing a path

In this section, we show that Lemma 1 can be generalized to any ordered rooted tree, regardless of its arity.

Lemma 2 *Let $p = 0.48$. Given any ordered rooted tree T of size n, there exists a root-to-leaf path π in T such that for any left subtree α and any right subtree β of π, $|\alpha|^p + |\beta|^p \leq (1-\delta)n^p$ for some constant $\delta > 0$.*

Proof. We will iteratively expand path $\pi = \langle v_1, \ldots, v_i \rangle$ to get closer to a leaf, and let α_i, β_i be the largest left/right subtree of this path (not considering the subtrees at v_i). Initially set v_1 to be the root. We maintain the invariant that $|\alpha_i|^p + |\beta_i|^p \leq (1-\delta)n^p$ for every i; this holds vacuously initially.

Now assume that path π up to v_i for some $i \geq 1$ has been chosen. Let $S_i^{(1)}, \ldots, S_i^{(d_i)}$ be the subtrees at v_i, enumerated from left to right. Call such a subtree $S_i^{(k)}$ *feasible* if we could use its root to extend π while maintaining the invariant. Thus $S_i^{(k)}$ is feasible if

$$\max\left\{|\alpha_i|, |S_i^{(1)}|, \ldots, |S_i^{(k-1)}|\right\}^p + \max\left\{|\beta_i|, |S_i^{(k+1)}|, \ldots, |S_i^{(d_i)}|\right\}^p \leq (1-\delta)n^p.$$

For future reference we note that $S_i^{(k)}$ is *infeasible* if one of the following three *violations* occurs:

1. $|\alpha_i|^p + |S_i^{(\ell)}|^p > (1-\delta)n^p$ for some $\ell > k$.
2. $|S_i^{(h)}|^p + |\beta_i|^p > (1-\delta)n^p$ for some $h < k$.
3. $|S_i^{(h)}|^p + |S_i^{(\ell)}|^p > (1-\delta)n^p$ for some $h < k < \ell$.

Case 1: Exactly one subtree $S_i^{(k)}$ is feasible. Then we set v_{i+1} to be the root of $S_i^{(k)}$. The invariant holds by the definition of "feasible".

Case 2: At least two subtrees $S_i^{(k)}$, $S_i^{(\ell)}$ (with $k < \ell$) are feasible. We terminate the construction as follows.

Consider first the subcase where $|S_i^{(k)}| \leq |S_i^{(\ell)}|$. Set path π to be the concatenation of $\langle v_1, \ldots, v_i \rangle$ with the leftmost path in $S_i^{(k)}$ down to a leaf. A left subtree of this path has size at most $\max\{|\alpha_i|, \max_{h<k} |S_i^{(h)}|\}$. A right subtree of this path up to v_i has size at most $\max\{|\beta_i|, \max_{h>k} |S_i^{(h)}|\}$. A right subtree of this path below v_i is a subtree of $S_i^{(k)}$ (and hence no bigger than $|S_i^{(k)}| \leq |S_i^{(\ell)}| \leq \max_{h>k} |S_i^{(h)}|$ by assumption). Since $S_i^{(k)}$ is feasible the invariant holds.

The other subcase, $|S_i^{(\ell)}| \leq |S_i^{(k)}|$ can be handled in a symmetric fashion by extending instead with the rightmost path in $S_i^{(\ell)}$.

Claim 1 *One of the above two cases always applies.*

Proof. Assume not. To show that this leads to a contradiction, we find some subtrees (or collections of subtrees) for which we can lower-bound the size. This part is significantly more complicated than in Chan's proof because there are now multiple ways in which a subtree might not be feasible, and we must choose our subtrees correspondingly. Consider Figure 2 for an illustration of the following definitions.

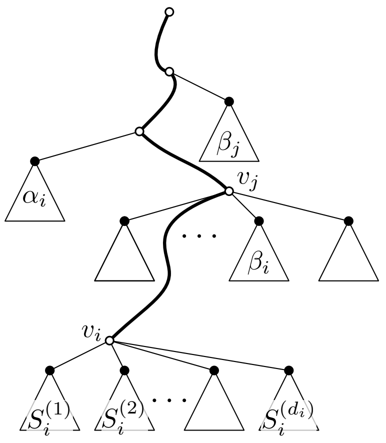


Figure 2: The situation up to symmetry.

Suppose that the parent of β_i's root, which we denote by v_j, is no higher than the parent of α_i's root; the other case is symmetric. We first derive one inequality from v_j. We have $j < i$ by definition of β_i. Because we did not terminate the path when extending at v_j, Case 2 did not apply at v_j. Therefore the subtree $S_j^{(k)}$ of v_j that contains v_i was the *only* feasible subtree at v_j.

Consider Figure 3. Tree β_i is a right subtree at v_j, say it was $S_j^{(\ell)}$ with $\ell > k$. Let $\mathcal{L}_j$ be the set of nodes that belong to subtrees $S_j^{(1)}, \ldots, S_j^{(\ell-1)}$. We know that $S_j^{(\ell)}$ was infeasible, and study the three possible violations:
(1) $|\alpha_j|^p + |S_j^{(\ell')}|^p > (1-\delta)n^p$ for some $\ell' > \ell$. But this is impossible since $S_j^{(k)}$ is feasible and $k < \ell$.
(2) $|S_j^{(k')}|^p + |\beta_j|^p > (1-\delta)n^p$ for some $k' < \ell$ (possibly $k = k'$). In this case, set $B_j := \beta_j$.
(3) $|S_j^{(k')}|^p + |S_j^{(\ell')}|^p > (1-\delta)n^p$ for some $k' < \ell < \ell'$. In this case, set $B_j := S_j^{(\ell')}$. Note that either way B_j is disjoint from $\beta_i = S_j^{(\ell)}$ and a right subtree of one of $v_1, \ldots, v_j$, so also disjoint from $\mathcal{L}_j$. The nodes in the subtree $S_j^{(k')}$ that caused the above violation for $S_j^{(\ell)}$ belong to $\mathcal{L}_j$, and therefore

$$|\mathcal{L}_j|^p + |B_j|^p > (1-\delta)n^p \tag{2}$$

Now we define two subtrees L_i, R_i at v_i. Since the leftmost subtree $S_i^{(1)}$ at v_i is infeasible, but extending into it would not add left subtrees to the path, the infeasibility must be caused by violation (1), i.e., there exists some $k > 1$ such that $|S_i^{(k)}|^p + |\alpha_i|^p > (1-\delta)n^p$. Set R_i to be this subtree $S_i^{(k)}$, choosing the largest possible

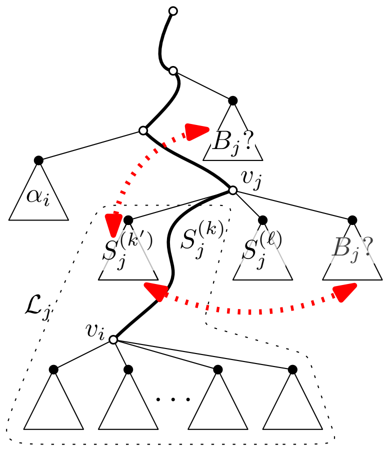


Figure 3: Close-up on v_j. Red arrows indicates pairs of subtrees that violate feasibility.

index k. We have

$$|R_i|^p + |\alpha_i|^p > (1-\delta)n^p. \qquad (3)$$

Symmetrically, since the rightmost subtree $S_i^{(d_i)}$ at v_i is infeasible, there must be some $h < d_i$ such that $|S_i^{(h)}|^p + |\beta_i|^p > (1-\delta)n^p$. Set L_i to be this subtree $S_i^{(h)}$, choosing the smallest possible index h. We have

$$|L_i|^p + |\beta_i|^p > (1-\delta)n^p. \qquad (4)$$

Note that it is possible $L_i = R_i$ and that both are subsets of $\mathcal{L}_j$.

Case A: $L_i \neq R_i$. See Figure 4. In this case the contradiction is obtained exactly as done by Chan, except by substituting the trees/forests that we have chosen above suitably. Recall that Hölder's inequality states that for $p < 1$ we have

$$\sum_a x_a y_a \leq \left(\sum_a x_a^{1/(1-p)}\right)^{1-p} \left(\sum_a y_a^{1/p}\right)^p$$

(in some of our applications below $x_a = 1$ for all a). We can derive a contradiction for the value $p = 0.48$ (with a sufficiently small δ) by combining Eqs. (2-4) as follows:

$$\begin{aligned}
&2.5(1-\delta)n^p \\
<&|\alpha_i|^p + |\beta_i|^p + |L_i|^p + |R_i|^p + 0.5|\mathcal{L}_j|^p + 0.5|B_j|^p \\
\leq&|\alpha_i|^p + |\beta_i|^p + 2^{1-p}(|L_i|+|R_i|)^p + 0.5|\mathcal{L}_j|^p + 0.5|B_j|^p \\
\leq&|\alpha_i|^p + |\beta_i|^p + (2^{1-p}+0.5)|\mathcal{L}_j|^p + 0.5|B_j|^p \\
\leq&\big(1+1+(2^{1-p}+0.5)^{1/(1-p)} + 0.5^{1/(1-p)}\big)^{1-p} \\
&\qquad\cdot(|\alpha_i| + |\beta_i| + |\mathcal{L}_j| + |B_j|)^p \\
<&2.499n^p
\end{aligned}$$

since $\alpha_i, \beta_i, \mathcal{L}_i$ and B_j are all disjoint.

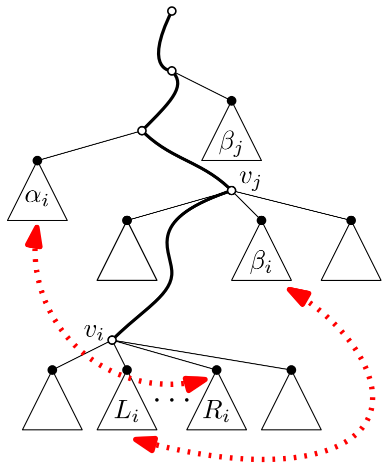


Figure 4: Close-up on v_i, Case (A).

Case B: $L_i = R_i$. In this case we derive one further inequality, see Figure 5. Since $R_i = S_i^{(k)}$ is not feasible, there must exist a violation, and we consider its three possible forms. (1) $|S_i^{(\ell)}|^p + |\alpha_i|^p > (1-\delta)n^p$ for some $\ell > k$. But this is impossible since R_i was chosen as the rightmost such violation. (2) $|S_i^{(h)}|^p + |\beta_i|^p > (1-\delta)n^p$ for some $h < k$. But this is impossible since $R_i = L_i$ was chosen as the leftmost such violation. (3) $|S_i^{(h)}|^p + |S_i^{(\ell)}|^p > (1-\delta)n^p$ for some $h < k < \ell$. In this case we define $\hat{L}_i := S_i^{(h)}$ and $\hat{R}_i := S_i^{(\ell)}$. Summarizing, $\mathcal{L}_j$ contains the three mutually distinct subtrees $\hat{L}_i, L_i{=}R_i, \hat{R}_i$ and

$$|\hat{L}_i|^p + |\hat{R}_i|^p > (1-\delta)n^p. \qquad (5)$$

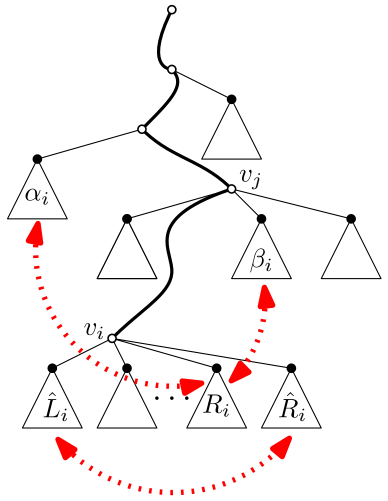


Figure 5: Close-up on v_i, Case (B).

Using again Hölder's inequality we obtain the desired contradiction: by combining Eqs. (2)-(5) (and $L_i = R_i$)

as follows:

$$\begin{aligned} & 3.5(1-\delta)n^p \\ < \; & |\alpha_i|^p + |\beta_i|^p + 2|R_i|^p + |\hat{L}_i|^p + |\hat{R}_i|^p \\ & \quad +0.5|\mathcal{L}_j|^p + 0.5|B_j|^p \\ \leq \; & |\alpha_i|^p + |\beta_i|^p + (2^{1/(1-p)}+2)^{1-p}(|R_i| + |\hat{L}_i| + |\hat{R}_i|)^p \\ & \quad +0.5|\mathcal{L}_j|^p + 0.5|B_j|^p \\ \leq \; & |\alpha_i|^p + |\beta_i|^p + ((2^{1/(1-p)}+2)^{1-p} + 0.5)|\mathcal{L}_j|^p \\ & \quad +0.5|B_j|^p \\ \leq \; & (1 + 1 + ((2^{1/(1-p)}+2)^{1-p}+0.5)^{1/(1-p)} \\ & \quad +0.5^{1/(1-p)})^{1-p} \cdot (|\alpha_i| + |\beta_i| + |\mathcal{L}_j| + |B_j|)^p \\ < \; & 3.396n^p. \end{aligned}$$

So the claim holds. □

So Case 1 or Case 2 always applies, and we can continue to expand the path until we terminate the procedure at a leaf or in some application of Case 2. □

3 Generalized LR-drawings

LR-drawings for binary trees were defined via two particular construction operations. In contrast, we want to define here *generalized LR-drawings* via the properties that the drawings must satisfy. Let T be an ordered rooted tree, and consider an order-preserving planar drawing Γ of T. We call Γ a *generalized LR-drawing* (or *GLR-drawing* for short) if it (and all induced drawings of rooted subtrees) satisfies the following two conditions:

(P1) (*vertical path*): There exists a root-to-leaf path π $\langle v_1, \dots, v_\ell \rangle$ that is drawn vertically aligned, with v_i above v_{i+1} for $1 \leq 1 < \ell$.

(P2) (*path-separation*): The column that contains π separates the drawings of the left and right subtrees, i.e., for any left or right subtree T', the drawing Γ' of T' induced by Γ does not use the column containing π.

Clearly any LR-drawing for binary trees satisfies these two conditions, so this is indeed a generalization. As one can verify by inspecting the proofs, these are the *only* two conditions needed for the lower-bound arguments in [8] and [4]. We hence have:

Corollary 3 (based on [4]) *For every positive n there exists an n-node ordered rooted tree T such that any GLR-drawing of T has width $\Omega(n^{0.429})$.*

We note that many existing algorithms for ideal drawings of trees (see e.g. [1, 2, 9]) satisfy the condition on drawing some path π vertically. The real restriction on GLR-drawings is that the vertical path separates the left and right subtrees. The algorithms in [1,2,9] all reuse the column of the vertically-drawn path for some large subtree that has been "pushed down".

There are many more properties that are satisfied by the LR-drawings for binary trees (and so arguably one could have included them in the definition of GLR-drawings, though for maximal flexibility we chose not to do that). All LR-drawings of binary trees, and also all drawings that we will create, satisfy the following three properties:

(P3) (*vertical separation of subtrees*). For any node v of T, let T_v be the subtree rooted at v. There exists a horizontal strip that contains all nodes of T_v and that does not contain any other node of Γ.

(P4) (*grouping of subtrees at v_i*). For any node $v_i \in \pi$, there exists a horizontal strip that contains v_i and all nodes of all left and right subtrees at v_i and that does not contain any other node of Γ.

(P5) (*grouping of left/right subtrees at v_i*). For any node $v_i \in \pi$, there exists a horizontal strip that contains all nodes of all left subtrees at v_i and does not contain any other nodes of Γ. There also exists a horizontal strip that contains all nodes of all right subtrees at v_i and does not contain any other nodes of Γ.

Finally, there are three more properties that the LR-drawings of binary trees satisfy, but some of our constructions do not (and as we will argue, we cannot hope to satisfy them and have sub-quadratic area).

(P6) The drawing is straight-line.

(P7) The drawing is strictly upward.

(P8) (*minimum-distance*) Bounding boxes of subtree-drawings have the minimum possible distance to the path and to each other. Formally, we use $B(T')$ to denote the bounding box of the drawing of rooted subtree T'. We require that if T' is a left subtree of π, then the right side of $B(T')$ lies one unit left of π, and symmetrically for right subtrees. We also require minimal vertical distances between bounding boxes, most easily expressed by demanding that every row contains a node.

We first show that all eight conditions given above can be satisfied simultaneously if we allow for quadratic area. The construction is the "standard construction" [3] and hence nearly trivial; we repeat the details for completeness' sake.

Lemma 4 *Any n-node ordered rooted tree has a GLR-drawing that additionally satisfies conditions (P3)-(P8) and has area $O(n^2)$. Furthermore, the root is placed in the top-left corner of the bounding box.*

Proof. If T consists of a single node, draw such node as an arbitrary point in the plane. Otherwise let $R^{(1)}, \ldots, R^{(d)}$ be the subtrees rooted at the children of the root v_T of T, enumerated from right to left. Recursively compute a drawing for each such subtree and combine them as follows. The drawing of $R^{(1)}$ is placed such that the top side of its bounding box $B(R^{(1)})$ is one unit below v_T, while its left side is one unit to the right of v_T. The drawing of $R^{(i)}$, $1 < i < d$, is placed such that the top side of $B(R^{(i)})$ is one unit below the bottom side of $B(R^{(i-1)})$, while its left side is again one unit to the right of v_T. The drawing of $R^{(d)}$ is drawn such that its root is vertically aligned with v_T and the top side of $B(R^{(d)})$ is one unit below the bottom side of $B(R^{(d-1)})$. (This corresponds to choosing π as the leftmost path of T.) It is easy to verify that each edge can be drawn as a straight-line segment, and that the resulting drawing is a strictly-upward GLR-drawing that satisfies all conditions. Moreover, the construction guarantees that both the width and the height are at most n. □

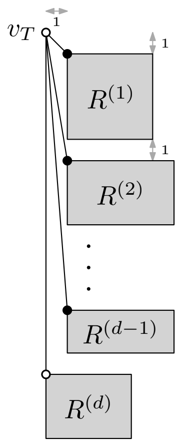


Figure 6: Construction of an ideal GLR-drawing of a d-ary tree in $O(n^2)$ area.

Unfortunately, it turns out that we cannot hope for smaller area if we want to satisfy all conditions.

Lemma 5 *For every positive integer k there exists an ordered rooted tree T with $n = 6k - 1$ nodes and arity 4 such that any GLR-drawing of T that also satisfies conditions (P3)-(P8) for all subtrees has area $\Omega(n^2)$.*

Proof. Tree T consists of root v_0 with four children $\ell_1, v_1, v_1', \ell_1'$ (in left-to-right order). Here ℓ_1, ℓ_1' are leaves while v_1 and v_1' are each root of a tree of height k. Specifically, for $1 \leq i < k$, nodes v_i and v_i' each have three children (in order): one leaf ℓ_{i+1} resp. ℓ_{i+1}', node v_{i+1} resp. v_{i+1}', and another leaf. See Figure 7 for the case when $k = 4$. Fix an arbitrary GLR-drawing of T that satisfies criteria (P3)-(P8). Up to symmetry, we

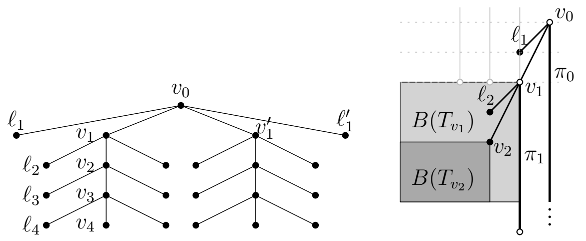


Figure 7: Construction of a family of trees that require quadratic area in any GLR-drawing that satisfies all conditions (P3)-(P8).

may assume that the vertically drawn path π_0 uses v_1' or ℓ_1', so T_{v_1} is a left subtree of π_0.

We will now show by induction that for $i = 1, \ldots, k$ tree T_{v_i} is a left subtree of the path π_{i-1} that is drawn vertically in $T_{v_{i-1}}$. This holds for $i = 1$ by the above. Assume T_{v_i} is a left subtree of π_{i-1} for some $i \geq 1$. By the minimum-distance condition, and since T_{ℓ_i} only consists of ℓ_i, the location of ℓ_i is one unit left of path π_{i-1}. Now consider the bounding box $B(T_{v_i})$ of the drawing of T_{v_i}. Again by the minimum-distance condition, its right side must be one unit left of path π_{i-1}. Since left subtrees at v_{i-1} are grouped, distances are minimum and the drawing is ordered, the top side of $B(T_{v_i})$ must be one unit below ℓ_i. Since the drawing is strictly-upward, node v_i is in the top row of $B(T_{v_i})$. Since (v_{i-1}, v_i) is drawn with a straight-line segment, this requires v_i to be on the top right corner of $B(T_{v_i})$, otherwise the drawing would not be order-preserving at v_{i-1} or (v_{i-1}, v_i) would overlap ℓ_i. So we now know that T_{v_i} is drawn with its root in the top right corner. It follows that π_i must use the right child of v_i, i.e., goes to a leaf while $T_{v_{i+1}}$ is a left subtree of π_i. This finishes the induction step.

So each v_i is in a left subtree of the path π_{i-1}, and therefore one unit further left that v_{i-1} (which lies on π_{i-1}). Therefore the drawing has width at least $k \in \Omega(n)$. We also know that the drawing has height at least $k \in \Omega(n)$ since path $v_0, v_1, \ldots, v_k$ is drawn strictly upward. So the area is $\Omega(n^2)$. □

So in our constructions, we relax some of the conditions (P6)-(P8), and show that then we can achieve subquadratic-area GLR-drawings.

3.1 Upward 1-bend GLR-drawings

Let T be an ordered rooted tree, and let $\pi = \langle v_1, v_2, \ldots, v_\ell \rangle$ be a root-to-leaf path of T. We give a simple recursive construction to compute a strictly-upward generalized LR-drawing of T by using at most one bend per edge. Refer to Figure 8 for an illustration.

Assume v_1 is placed at an arbitrary point of the plane. For ease of notation, let $L^{(1)}, \ldots, L^{(l_1)}$ be the left sub-

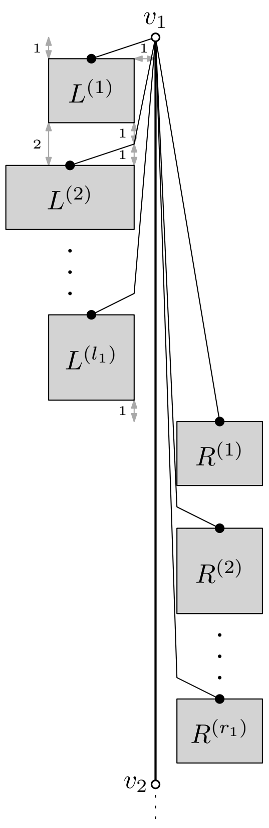


Figure 8: Construction for 1-bend generalized LR-drawings.

trees rooted at v_1, enumerated from left to right. Recursively compute a drawing for each $L^{(i)}$, $1 \leq i \leq l_1$, and denote by $B(L^{(i)})$ the corresponding bounding box. Place the drawing of $L^{(1)}$ such that the top side of $B(L^{(1)})$ is one unit below v_1 and so that the right side of $B(L^{(1)})$ is one unit to the left of v_1. Similarly, for each $1 < i \leq l_1$, place the drawing of $L^{(i)}$ such that that the top side of $B(L^{(i)})$ is two units below the bottom side of $B(L^{(i-1)})$ and so that the right side of $B(L^{(i)})$ is one unit to the left of v_1. Let $R^{(1)}, \ldots, R^{(r_1)}$ be the right subtrees at v_1, enumerated from right to left. Apply a symmetric construction as for the left subtrees and move them down such that the top side of $B(R^{(1)})$ is placed one unit below the bottom side of $B(L^{(l_1)})$. Now place v_2 vertically aligned with v_1 and one unit below the bottom side of $B(R^{(r_1)})$. Concerning the edges, observe that the edge connecting v_1 to the root of $L^{(1)}$ can be drawn with a straight-line segment without crossings, whereas the other edges that connect v_1 to the root of each subtree $L^{(i)}$, with $i > 1$, can instead be drawn with precisely one bend placed one unit above the top side of $B(L^{(i)})$ and one unit to the left of v_1. The edges that connect v_1 to the roots of the subtrees $R^{(i)}$ are drawn symmetrically. By repeating the construction for each v_i, with $1 < i \leq \ell$, we conclude the drawing.

Every row contains a node or a bend, so the height is $O(n)$. The width obeys Eq. (1). When path π is chosen as prescribed by Lemma 2, Chan [2] proved that Eq. (1) solves to $O(n^{0.48})$. The next lemma follows.

Lemma 6 *Any ordered rooted tree of size n admits a strictly-upward GLR-drawing with at most one bend per edge, whose width is $O(n^{0.48})$ and whose height is $O(n)$.*

The above GLR-drawings satisfy (P3-P5) and (P7), but not (P5) or (P8) due to the bends and the space needed for them. They can be vertically stretched so to become straight-line and hence an ideal GLR-drawing. Namely, for each edge (u, v) drawn with one bend, it suffices to insert sufficiently many rows above the bend point so to guarantee a direct line of sight between u and v. This is always possible because each bend point is such that no other node or bend is placed with the same y-coordinate. While the above transformation does not change the width of the drawing, it may produce a height that is not polynomial in n. Also, it does not satisfy (P8).

3.2 Non-upward straight-line LR-drawings

In this section, we show how we can avoid using bends. Thus we create a GLR-drawing that is straight-line, and in fact satisfies all of (P3)-(P8) except that it is not upward. The crucial idea is to give two drawing-algorithms to create different types of GLR-drawings.

- In a type-I drawing, the root is located in the top row (with no restriction on the column).
- In a type-II drawing, the root is located in the leftmost or rightmost column, with no node above it in the same column. We will use type-IIℓ and type-IIr to specify whether the root is left or right.

(A somewhat similar idea for tree-drawing, in a very different model, was used in [5].)

Lemma 7 *Let $p = 0.48$. Given any ordered rooted tree T of size n, there exist*

- *a straight-line GLR-drawing of type I that has width at most $cn^p - 1$ (for some constant $c > 0$),*
- *a straight-line GLR-drawing of type IIℓ that has width at most cn^p (for the same constant c), and*
- *a straight-line GLR-drawing of type IIr that has width at most cn^p (for the same constant c).*

Furthermore, all drawings have height n.

Proof. If T consists of a single node then the claim holds trivially. Otherwise, pick a path $\pi = \langle v_1, \ldots, v_\ell \rangle$ with Lemma 2.

We first explain how to create type-I drawings, which is very similar to Section 3.1 except that we use type-II drawings to avoid bends. Assume v_1 is placed at an arbitrary point of the plane. Let $L_1^{(1)}, \dots, L_1^{(l_1)}$ and $R_1^{(1)}, \dots, R_1^{(r_1)}$ be the left and right subtrees at v_1, enumerated as in Section 3.1. Recursively compute drawings as follows:

- a type-I drawing for $L_1^{(1)}$ and $R_1^{(1)}$,
- a type-IIr drawing for each $L_1^{(i)}$, $2 \le i \le l_1$, and
- a type-IIℓ drawing for each $R_1^{(i)}$, $2 \le i \le r_1$.

Place the drawings as in Section 3.1, except leave only one unit vertical distance between the bounding boxes. As before, the edges from v_1 to the roots of $L_1^{(1)}$ and $R_1^{(1)}$ can be drawn straight-line without crossing. The edges to all other children can now also be drawn straight-line since those children are in an adjacent column to v_1.

Any left subtree α uses at most $c|\alpha|^p$ columns and any right subtree β uses at most $c|\beta|^p$ columns by induction, so by Lemma 2 the width is at most

$$c|\alpha|^p + c|\beta|^p + 1 \le c(1-\delta)n^p + 1 \le cn^p - 1$$

for the constant $\delta > 0$ from Lemma 2 and assuming c is sufficiently large.

Now we turn towards type-II drawings and only explain how to create a type-IIℓ drawing; the other type is symmetric. Let $\pi = \langle v_1, \dots, v_\ell \rangle$, and let $k \ge 1$ be the minimal index such that v_k has a left subtree. (If there is no such v_k then the type-I drawing is in fact a type-IIℓ drawing.) We draw $v_1, \dots, v_{k-1}$ as we did for type-I drawings; since they do not have left subtrees this places $v_1, \dots, v_{k-1}$ in the leftmost column. At v_k, we proceed as follows:

- As in Section 3.1, let $L_k^{(1)}, \dots, L_k^{(l_k)}$ and $R_k^{(1)}, \dots, R_k^{(r_k)}$ be the left and right subtrees at v_k.
- We use a type-IIℓ drawing for $R_k^{(1)}, \dots, R_k^{(r_k)}$, and denote by $B(R_k^{(i)})$ the corresponding bounding box. Place the drawing of $R_k^{(1)}$ such that the top left corner of $B(R_k^{(1)})$ is one unit below the bottom left corner of the bottommost subtree of v_{k-1}. (If $k = 1$, then place $R_k^{(1)}$ arbitrarily.) For $i = 2, \dots, r_k$, place the drawing of $R_k^{(i)}$ such that the top left corner of $B(R_k^{(i)})$ is one unit below the bottom left corner of $B(R_k^{(i-1)})$.
- Place v_k in the next row below. If $k > 1$, place v_k vertically below v_{k-1}. If $k = 1$ and $r_1 > 0$, place v_k such that it is one unit to the left of $B(R_k^{(1)})$. If $k = 1$ and there was no right subtree, then place v_k arbitrarily.

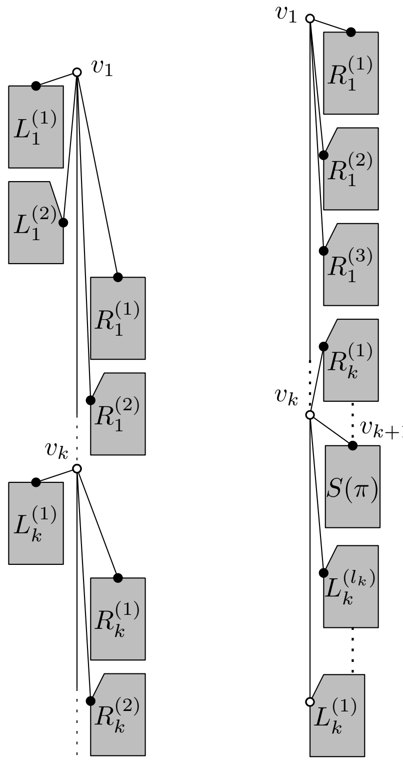


Figure 9: Constructions of straight-line drawings with linear height. (Left) Type-I drawings. (Right) Type-IIℓ drawings. A 'cut-off' corner indicates a Type-II drawing where the column above the root must be empty.

- Let $S(\pi)$ be the subtree rooted at v_{k+1} (thus containing the rest of π). Use a type-I drawing for $S(\pi)$, and place it in the rows below v_k, with the left side of $B(S(\pi))$ one unit to the right of v_k.
- We use a type-IIℓ drawing for $L_k^{(1)}, \dots, L_k^{(l_k)}$. We know that $l_k > 0$ by choice of k. If $l_k > 1$, then place $L_k^{(l_k)}$ such that the top left corner of $B(L_k^{(l_k)})$ is one unit below the bottom left corner of $B(S(\pi))$. For i from $l_k - 1$ down to 2, place the drawing of $L_k^{(i)}$ such that the top left corner of $B(L_k^{(i)})$ is one unit below the bottom left corner of $B(L_k^{(i+1)})$. Finally, place the drawing of $L_k^{(1)}$ in the next rows such that the top left corner of $B(L_k^{(1)})$ is exactly below v_k.

Thus, as in [4], the vertically drawn path is *not* the path π that we started out with, instead it is $v_1, \dots, v_k$ plus the vertically-drawn path of $L^{(1)}$. But still we obtain a GLR-drawing. To prove that this drawing has the appropriate width, we need an observation.

Claim 2 *Let T' be a left or right subtree of path π chosen with Lemma 2. Then T' has size at most $(1-\delta)^{1/p} n$, for the constant $\delta > 0$ from Lemma 2.*

Proof. This follows directly from the bound in the lemma since necessarily $|T'|^p \leq (1-\delta)n^p$. □

Therefore, at any subtree other than $S(\pi)$, the width is by induction at most $1 + c(1-\delta)n^p \leq cn^p$ for sufficiently large c. At subtree $S(\pi)$, the recursively obtained type-I drawing has width at most $cn^p - 1$, and so the width again is at most cn^p as desired.

In all cases, we never insert empty rows between drawings, so every row contains exactly one node and the height is n as desired. □

As in Section 3.1, we can stretch the drawing vertically to make it upward, by moving all subtrees at v_k downward and leaving a sufficiently large gap between $B(R^{(r_k)})$ and $B(S(\pi)))$ so that edge (v_k, v_{k+1}) can be routed straight-line. (Details are left to the reader.) Again the height may not be polynomial and condition (P8) no longer holds.

3.3 Upward straight-line LR-drawings

As shown in the previous sections, we can achieve width $O(n^{0.48})$, but the drawings are either not straight-line, or not upward, or have large (possibly super-polynomial) height. In this section, we show that with a different construction, we can bound the height to be $O(n^{1.48})$ in a straight-line, upward GLR-drawing of width $O(n^{0.48})$. The area hence is $O(n^{1.96})$, just barely under the trivial $O(n^2)$ bound.

The idea for this is to follow a *different* approach of Chan ('Method 4' [2]) for tree-drawing; here we occasionally double the height used for some subtrees, but this happens rarely enough that overall the height can still be bounded. Chan's Method 4 does not produce GLR-drawings (he lets the largest subtree re-use the column of the vertical path) but we can modify the approach at the cost of increasing the width by one unit. To compensate for this we use a type-I drawing for the largest subtree, so that the overall width does not increase too much.

So again we have drawing-types. One of them is exactly the type-I drawing used in the previous section. The other one, which we call type-III drawing, has the root located in the top left or top right corner; we use type-IIIℓ and type-IIIr to specify whether the root is left or right.

Lemma 8 *Let $p = 0.48$. Given any ordered rooted tree T of size n, there exist*

- *a straight-line upward GLR-drawing of type I that has width at most $cn^p - 1$ (for some constant $c > 0$),*
- *a straight-line upward GLR-drawing of type IIIℓ that has width at most cn^p (for the same constant c), and*
- *a straight-line upward GLR-drawing of type IIIr that has width at most cn^p (for the same constant c).*

Furthermore, all drawings have height at most $2n^{1+p}$.

Proof. If T consists of a single node then the claim holds trivially. Otherwise, pick a path π with Lemma 2, so that $|\alpha|^p + |\beta|^p \leq (1-\delta)n^p$ for any left and right subtrees α, β of the path. The creation of a type-I drawing is exactly as in the previous section, except that we use type-III drawings in place of type-II drawings so that we have an upward drawing. Using $H(\cdot)$ to denote the height, we have $H(n) \leq \sum_{i=1}^{d} H(n_i) + 1$ where d is the number of subtrees and n_i is the size of the ith subtree. Since $n_i \leq n$ and $\sum_i n_i = n - 1$ we have

$$\begin{aligned} & \sum_{i=1}^{d} H(n_i) + 1 \\ \leq \quad & \sum_{i=1}^{d} 2n_i n^p + 2n^p \\ \leq \quad & 2n^p(\sum_{i=1}^{d} n_i + 1) = 2n^{1+p}. \end{aligned}$$

To construct type-III drawings, we proceed much as in Chan's Method 4. Fix $A = n/2^{1/p} > 0.23n$. (The value of A is different from Chan's, but its use is nearly the same.) We will completely disregard path π and instead pick one subtree of the root based on A. To simplify notations, let $S_1, \ldots, S_d$ be the subtrees at the root, enumerated from left to right. We only explain how to construct a type-IIIℓ drawing; constructing type-IIIr drawings is symmetric. We have two cases:

Case 1: Either $d \leq 2$ or the subtrees $S_2, \ldots, S_{d-1}$ all have size at most $n - A$. In this case, recursively construct a type-IIIℓ drawing for $S_1, \ldots, S_{d-1}$ and a type-I drawing for S_d. Combine these drawings with the standard method that was used already in Lemma 4, see also Figure 10. Clearly this is a planar order-preserving straight-line upward drawing, and its height is $1 + \sum_i H(n_i) \leq 2n^{1+p}$ with the same analysis as for type-I drawings. The width $W(n)$ is at most $1 + (cn^p - 1) = cn^p$ at S_d, and at most $W(n-1) \leq cn^p$ at S_1. At any other subtree S_i, the size is at most $n - A < 0.77n$, and the width is at most $1 + W(0.77n) \leq 1 + c(0.77)^p n^p \leq cn^p$, assuming c is sufficiently large.

Case 2: $d \geq 3$ and at least one subtree S_k with $1 < k < d$ has size $n - A$ or more. In this case, use a type-I drawing for S_k and S_d, and a type-IIIℓ drawing for all other subtrees.

We explain how to build the drawing in Figure 10 bottom-up. Place the drawing of S_1 arbitrarily. Place

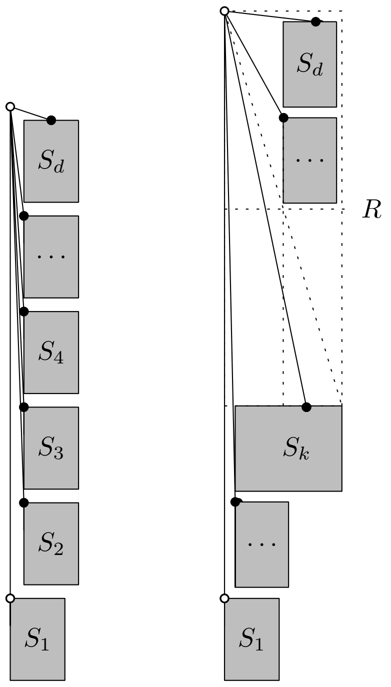


Figure 10: Constructions of type-IIIℓ straight-line upward drawings with subquadratic area. (Left) Case 1. (Right) Case 2.

$S_2, \dots, S_k$ on top of this, with one unit between their bounding boxes, such that the left sides of their bounding boxes are one unit to the right of the root of S_1. Now place an imaginary $W' \times H'$ rectangle R with its left side aligned with the left side of $B(S_1)$ and its bottom side coinciding with the top side of $B(S_k)$. Here $W' = \max\{|S_k|^p, \max_{i>k}\{2W(|S_i|)\}$, while $H' = 3 + 2\sum_{i>k} H(|S_i|)$. In particular, the top right quadrant of R is big enough to accommodate the drawings of $S_{k-1}, \dots, S_1$, plus one and a half rows. We place the root at the top left corner of R. We place the drawings of $S_{k-1}, \dots, S_1$ (in this order) below the root and in the top right quadrant of R, with the left sides of their bounding boxes aligned and unit vertical distance between boxes. Note that this does *not* use the centerpoint of R (which is not a grid point due to the odd height of R). The root of S_k is somewhere along the bottom of R, hence the edge from it to the root of T does not enter the top right quadrant of R and the drawing is planar. Clearly it is a GLR-drawing and also strictly upward. In fact, conditions (P3)-(P7) are all satisfied (but (P8) is not).

We first analyze the width. At S_k, the width is at most $1 + (cn^p - 1) \le cn^p$ by induction. At any other subtree S_i, the size is $n_i \le A = n/2^{1/p}$ and the width is at most

$$\max\{1 + W(|S_i|), 2W(|S_i|)\} \le 2 \cdot c \left(\tfrac{n}{2^{1/p}}\right)^p = cn^p$$

as desired. As for the height, S_k contributes at most $2n_k^{1+p}$ rows and any other subtree S_i contributes at the most $2 \cdot 2n_i^{1+p}$ rows. We need two further rows for R (the bottom row was already counted). Since $n_i \le A = n/2^{1/p}$ for $i \ne k$ and $\sum_i n_i = n - 1$, the height is at most

$$2 + 2n^p n_k + \sum_{i \ne k} 4\left(\tfrac{n}{2^{1/p}}\right)^p n_i \le 2n^p + 2n^p \sum_{i=1}^{d} n_i = 2n^{1+p}$$

as desired. □

4 Remarks

In this paper, we studied how to generalize the concept of LR-drawings that was previously designed for binary trees [2, 4, 8] to trees of higher arity. To this end, we first generalized a lemma by Chan about paths for which $|\alpha|^p + |\beta|^p \le (1-\delta)n^p$ for constant $p = 0.48$, $\delta > 0$ and any left and right subtree α, β. Then we explained how to use this path to construct generalized LR-drawings of width $O(n^{0.48})$ and subquadratic area, both with and without the restriction on the drawing being straight-line and/or upward. We conclude the paper by listing some open problems:

- The most natural open problem is to close the gap on the width of GLR-drawings. Frati et al. showed that width $\Omega(n^{0.418})$ is sometimes required [8], and Chan and Huang improved this to $\Omega(n^{0.428})$ [4]. These lower bounds were for binary trees; could they perhaps be strengthened if we allow higher arity? Using ternary trees, one can immediately reduce the size of the lower-bound tree T_h of [4, 8], by $2^h - 1$ (contract every second edge of path π) without affecting the validity of the lower-bound proof. Unfortunately, this improves the lower bound only by a lower-order term.

- We showed that in a GLR-drawing where all additional conditions (P3)-(P8) are satisfied, the width must be $\Omega(n)$. Is there an intermediate lower bound that shows up when requiring other subsets of these properties? We are especially curious about removing condition (P5) ('grouping of left/right subtrees'), which seems very artificial but is crucially required in the proof of Lemma 5.

- Chan and Huang improved the width of LR-drawings of binary trees [4]. The main idea is that rather than drawing one chosen path π as a straight-line, they add an 'i-twist' to the drawing of path π, using 2^i non-vertical edges for π while the corresponding other subtrees at these edges use vertical lines. With this, they can achieve an LR-drawing of width $O(n^{0.438})$ (and even smaller with further improvements).

It would be interesting to see whether this approach could be generalized to trees of higher arity. We cannot generalize the algorithm directly, because the subtrees that use vertical lines are defined via a size-property. In trees of higher arity these subtrees may well have common parents on π, making it impossible to use distinct vertical lines for them, as is necessary in the construction.

- Our construction of ideal GLR-drawings achieves subquadratic area, but barely. Can the area be improved?

References

[1] T. Biedl. Ideal drawings of rooted trees with approximately optimal width. *J. Graph Algorithms and Appl.*, 21(4):631–648, 2017.

[2] T. M. Chan. A near-linear area bound for drawing binary trees. *Algorithmica*, 34(1):1–13, 2002.

[3] T. M. Chan. Tree drawings revisited. *Discret. Comput. Geom.*, 63(4):799–820, 2020.

[4] T. M. Chan and Z. Huang. Improved upper and lower bounds for LR drawings of binary trees. In D. Auber and P. Valtr, editors, *GD 2020*, volume 12590 of *LNCS*, pages 71–84. Springer, 2020.

[5] G. Di Battista and F. Frati. Small area drawings of outerplanar graphs. *Algorithmica*, 54(1):25–53, 2009.

[6] G. Di Battista and F. Frati. A survey on small-area planar graph drawing, 2014. CoRR report 1410.1006.

[7] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. In *GD 2007*, volume 4875 of *LNCS*, pages 76–87. Springer, 2007.

[8] F. Frati, M. Patrignani, and V. Roselli. LR-drawings of ordered rooted binary trees and near-linear area drawings of outerplanar graphs. *J. Comput. Syst. Sci.*, 107:28–53, 2020.

[9] A. Garg and A. Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. *Int. J. Comput. Geometry Appl.*, 13(6):487–505, 2003.

[10] A. Garg and A. Rusu. Area-efficient planar straight-line drawings of outerplanar graphs. *Discret. Appl. Math.*, 155(9):1116–1140, 2007.

Massively Winning Configurations in the Convex Grabbing Game on the Plane

Martin Dvorak[*] Sara Nicholson[†]

Abstract

The convex grabbing game is a game where two players, Alice and Bob, alternate taking extremal points from the convex hull of a point set on the plane. Rational weights are given to the points. The goal of each player is to maximize the total weight over all points that they obtain. We restrict the setting to the case of binary weights. We show a construction of an arbitrarily large odd-sized point set that allows Bob to obtain almost 3/4 of the total weight. This construction answers a question asked by Matsumoto, Nakamigawa, and Sakuma in [Graphs and Combinatorics, 36/1 (2020)]. We also present an arbitrarily large even-sized point set where Bob can obtain the entirety of the total weight. Finally, we discuss conjectures about optimum moves in the convex grabbing game for both players in general.

1 Introduction

The graph grabbing game, which was first presented by Winkler [7], is a game where two players alternate removing non-cut vertices from a vertex-weighted graph. This game has been studied [1, 4, 6] and led to variants including the convex grabbing game by Matsumoto, Nakamigawa, and Sakuma [3], which we discuss here.

A cake $\mathcal{C}$ is determined by a set of points, which we call *cherries*, that lie in general position in the Euclidean plane. Each cherry c is given a weight $\mathrm{w}(c) \in \mathbb{Q}$. There are two players in this game: *Alice* and *Bob*. They alternate selecting cherries from the set of remaining cherries $C \subseteq \mathcal{C}$, with Alice going first. They can only select extremal points of the convex hull of C, defined by $\mathrm{Ex}(C) = \{c \in C \mid c \notin \mathrm{conv}(C \setminus \{c\})\}$, and the selected cherry is removed from the set C. The game is over when all cherries are taken.

If $|\mathcal{C}|$ is even, we say that $\mathcal{C}$ is an even-sized cake. Similarly, if $|\mathcal{C}|$ is odd, we say that $\mathcal{C}$ is an odd-sized cake.

[*]Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University; and Institute of Science and Technology, Austria; martin.dvorak@matfyz.cz

[†]Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University; s.nicholson@mail.utoronto.ca Supported by grant no. 21-32817S of the Czech Science Foundation (GAČR).

We denote the sequence of moves by Alice as:

$$(a_1, a_2, \dots, a_{\lceil \frac{|\mathcal{C}|}{2} \rceil})$$

We denote the sequence of moves by Bob as:

$$(b_1, b_2, \dots, b_{\lfloor \frac{|\mathcal{C}|}{2} \rfloor})$$

This results in a gameplay

$$\mathbf{q} = (a_1, b_1, a_2, b_2, \dots, a_{\frac{|\mathcal{C}|}{2}}, b_{\frac{|\mathcal{C}|}{2}})$$

on an even-sized cake $\mathcal{C}$, or

$$\mathbf{q} = (a_1, b_1, a_2, b_2, \dots, a_{\lfloor \frac{|\mathcal{C}|}{2} \rfloor}, b_{\lfloor \frac{|\mathcal{C}|}{2} \rfloor}, a_{\lceil \frac{|\mathcal{C}|}{2} \rceil})$$

on an odd-sized cake $\mathcal{C}$. In the end, each player obtains a total score equal to the sum of the weights of the cherries they selected. In particular, we define the total gain of Alice as:

$$\mathrm{A}(\mathbf{q}) = \sum_{i \in \{1,2,\dots,\lceil \frac{|\mathcal{C}|}{2} \rceil\}} \mathrm{w}(a_i)$$

The objective of Alice is to maximize $\mathrm{A}(\mathbf{q})$. The objective of Bob is to minimize $\mathrm{A}(\mathbf{q})$. We also define the complement:

$$\mathrm{B}(\mathbf{q}) = \sum_{i \in \{1,2,\dots,\lfloor \frac{|\mathcal{C}|}{2} \rfloor\}} \mathrm{w}(b_i)$$

We observe that $\mathrm{A}(\mathbf{q}) + \mathrm{B}(\mathbf{q})$ is invariant of $\mathbf{q}$; it is constant for a given cake $\mathcal{C}$. Alice wants to minimize $\mathrm{B}(\mathbf{q})$ and, naturally, Bob wants to maximize $\mathrm{B}(\mathbf{q})$. We will work with $\mathrm{B}(\mathbf{q})$ a lot because we will focus on maximizing Bob's results — which looks like a harder task, at least at first glance.

Finally, we define the *minimax result* on the cake $\mathcal{C}$, denoted by $\mathcal{M}(\mathcal{C})$, as the total gain of Bob if both players play optimally throughout the whole game.

$$\mathcal{M}(\mathcal{C}) = \min_{a_1 \in \mathrm{Ex}(\mathcal{C})} \left(\max_{\substack{b_1 \in \mathrm{Ex}(\\ \mathcal{C} \setminus \{a_1\})}} \left(\min_{\substack{a_2 \in \mathrm{Ex}(\\ \mathcal{C} \setminus \{a_1, b_1\})}} \left(\max_{\substack{b_2 \in \mathrm{Ex}(\\ \mathcal{C} \setminus \{a_1, b_1, a_2\})}} \left(\min_{\substack{a_3 \in \mathrm{Ex}(\\ \mathcal{C} \setminus \{a_1, b_1, a_2, b_2\})}} \left(\dots \left(\mathrm{B}((a_1, b_1, a_2, b_2, a_3, \dots)) \right) \dots \right) \right) \right) \right) \right)$$

We focus on a restricted version of this game where only $\{0,1\}$ weights are considered. Any cherry $c \in \mathcal{C}$ where $\mathrm{w}(c) = 1$ is called *red*; and we define the set of red cherries $\mathrm{R}(\mathcal{C}) \coloneqq \{c_r \in \mathcal{C} \mid \mathrm{w}(c_r) = 1\}$. In a similar manner, any cherry $c \in \mathcal{C}$ where $\mathrm{w}(c) = 0$ is called *green*; and we define their set $\mathrm{G}(\mathcal{C}) \coloneqq \{c_g \in \mathcal{C} \mid \mathrm{w}(c_g) = 0\}$. We thereby have $\mathrm{R}(\mathcal{C}) \cup \mathrm{G}(\mathcal{C}) = \mathcal{C}$ and $\mathrm{R}(\mathcal{C}) \cap \mathrm{G}(\mathcal{C}) = \emptyset$.

For any $C \subseteq \mathcal{C}$, we define values $\mathrm{r}(C) \coloneqq |\,\mathrm{R}(C)|$ and $\mathrm{g}(C) \coloneqq |\,\mathrm{G}(C)|$. Note that that $\mathrm{r}(C) + \mathrm{g}(C) = |C|$ and that $\mathrm{r}(C) = \sum_{c \in C} \mathrm{w}(c)$.

Matsumoto, Nakamigawa, and Sakuma [3] posed the question of finding the maximum possible value for $\mathcal{M}(\mathcal{C}) - (\mathrm{r}(\mathcal{C}) - \mathcal{M}(\mathcal{C}))$ on an odd-sized cake, that is: How much can Bob win by? In Section 3, we present, for any natural number z, a construction of an odd-sized cake $\mathcal{C}$ such that $\mathrm{r}(\mathcal{C}) = 4z + 2$; and we provide a tactic for Bob which guarantees $\mathcal{M}(\mathcal{C}) \geq \frac{3}{4}\,\mathrm{r}(\mathcal{C}) - \frac{1}{2}$. Therefore, Bob can win by an arbitrarily large margin.

In Section 4, we show that there exists an even-sized cake $\mathcal{D}$ where $\mathrm{r}(\mathcal{D}) = y$ and $\mathcal{M}(\mathcal{D}) = y$ for every $y \in \mathbb{N}$; that is, Bob can obtain all red cherries.

2 Order types

In this section, we provide a combinatorial point of view on the convex grabbing game. The following definition has been adapted from [5].

Definition 1 *Given a tuple (p, q, r) of three distinct cherries, we define their* orientation ∇pqr *as $+1$ if the sequence (p, q, r) traverses the triplet $\{p, q, r\}$ in a counterclockwise direction, and as -1 if this direction is clockwise.*

Consider two cakes, P and Q, where $|P| = |Q|$. We say that a bijection $\pi : P \to Q$ is order-preserving *if $\mathrm{w}(c) = \mathrm{w}(\pi(c))$ for each cherry $c \in P$ and there exists a sign $\sigma \in \{-1, +1\}$ such that $\nabla\pi(p)\pi(q)\pi(r) = \sigma \cdot \nabla pqr$ for all sequences (p, q, r) of three distinct cherries in P.*

If such a bijection exists, we say that P and Q are order-equivalent.

Proposition 2 *If $\mathcal{C}$ and $\mathcal{D}$ are order-equivalent cakes, then $\mathcal{M}(\mathcal{C}) = \mathcal{M}(\mathcal{D})$.*

The proof is in the full version [2] of this paper.

3 Sun configuration

We present a family of odd-sized cakes which we call the sun configuration, and we show that, from any cake $\mathcal{C}$ in the family, Bob will obtain at least $\frac{3}{4}\,\mathrm{r}(\mathcal{C}) - \frac{1}{2}$ red cherries given that he follows a certain tactic.

Definition 3 *We define a* beam $\mathcal{Y}$ *as four cherries in the order [green, red, green, red] lying on an arc (see Figure 1).*

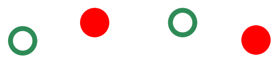


Figure 1: Beam $(\mathcal{Y})$.

Definition 4 *Let $k > 2$ be an odd integer. We define the sun as a cake $\mathcal{S}_k$ with k beams and an additional green cherry ζ in the centre (see Figure 2 for an example $\mathcal{S}_5$) such that:*

- *The sun is rotationally symmetric with the k beams evenly spaced around the centre ζ.*
- *Each beam is far enough from the centre such that, with the removal of any proper subset Y of the cherries on the beam, the outermost cherry on the beam will always be in $\mathrm{Ex}(\mathcal{S}_k \setminus Y)$.*
- *Consider any beam $\mathcal{Y}_i$ (see Figure 3). A line drawn through any two cherries of $\mathcal{Y}_i$ does not cut through any other beam and it keeps $\frac{k-1}{2}$ beams from $\mathcal{S}_k \setminus \mathcal{Y}_i$ on each side. Additionally, if we consider $\mathcal{Y}_i \cup \{\zeta\}$, then they are all in convex position.*

We have constructed a sun $\mathcal{S}_k$ where $\mathrm{r}(\mathcal{S}_k) = 2k$ and $\mathrm{g}(\mathcal{S}_k) = 2k + 1$.

Definition 5 *In the convex grabbing game on a sun, we say that a player follows the* Careful greedy tactic *if the player chooses a move according to these instructions:*

```
Is there an extremal red cherry?
  YES ⟶ Is there an extremal red cherry on a beam
  that also contains a red cherry that is not extremal?
    YES ⟶ Take extremal red cherry from this beam.
    NO ⟶ Is there a beam that contains a single
    red cherry and this red cherry is extremal?
      YES ⟶ Take this extremal red cherry.
      NO ⟶ Take any extremal red cherry.
  NO ⟶ Is there any beam with at least one green
  cherry and no red cherries on it?
    YES ⟶ Take a green cherry from this beam.
    (since at least one cherry from each remaining
    beam is extremal in any moment)
    NO ⟶ FAIL!
```

Theorem 6 *From the sun $\mathcal{S}_k$, Bob will get at least $\frac{3k-1}{2} = \frac{3}{4}\,\mathrm{r}(\mathcal{S}_k) - \frac{1}{2}$ red cherries by using the Careful greedy tactic, no matter how Alice plays. As a result, we obtain the desired property $\mathcal{M}(\mathcal{S}_k) \geq \frac{3}{4}\,\mathrm{r}(\mathcal{S}_k) - \frac{1}{2}$.*

We approach the proof as follows. We let Bob follow the Careful greedy tactic in all his moves. Alice can do anything.

We always describe the game state by the set of remaining cherries $C \subseteq \mathcal{S}_k$ and we create a lower bound for how many red cherries Bob is guaranteed to obtain from this moment until all cherries are taken. We

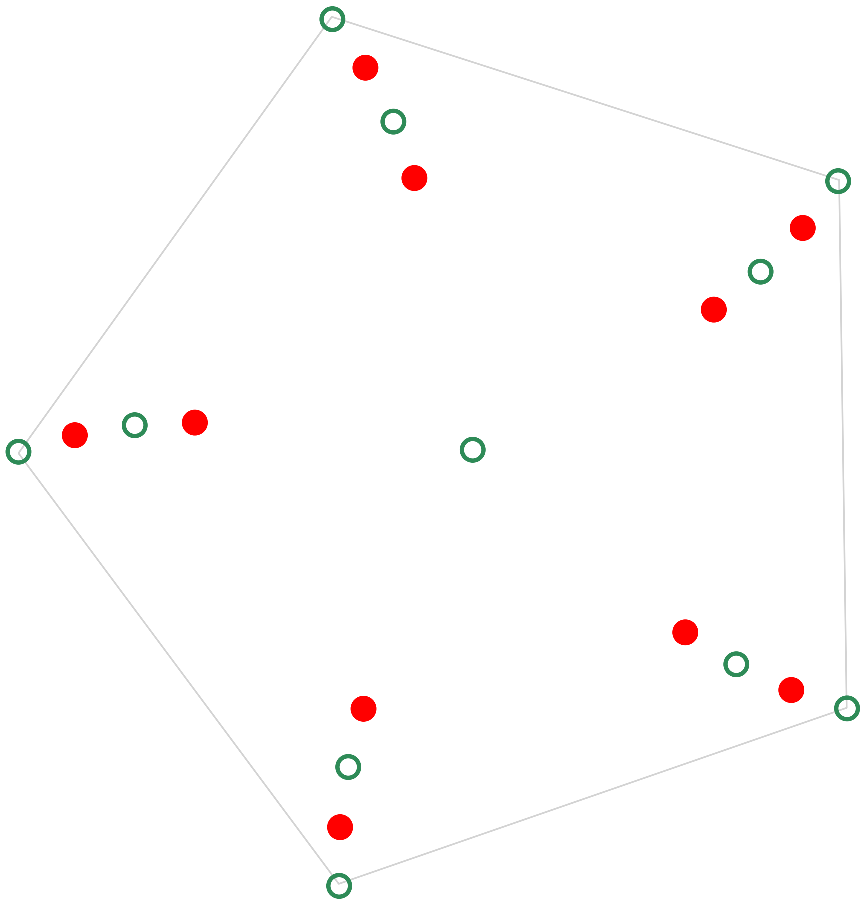


Figure 2: Sun ($\mathcal{S}_5$).

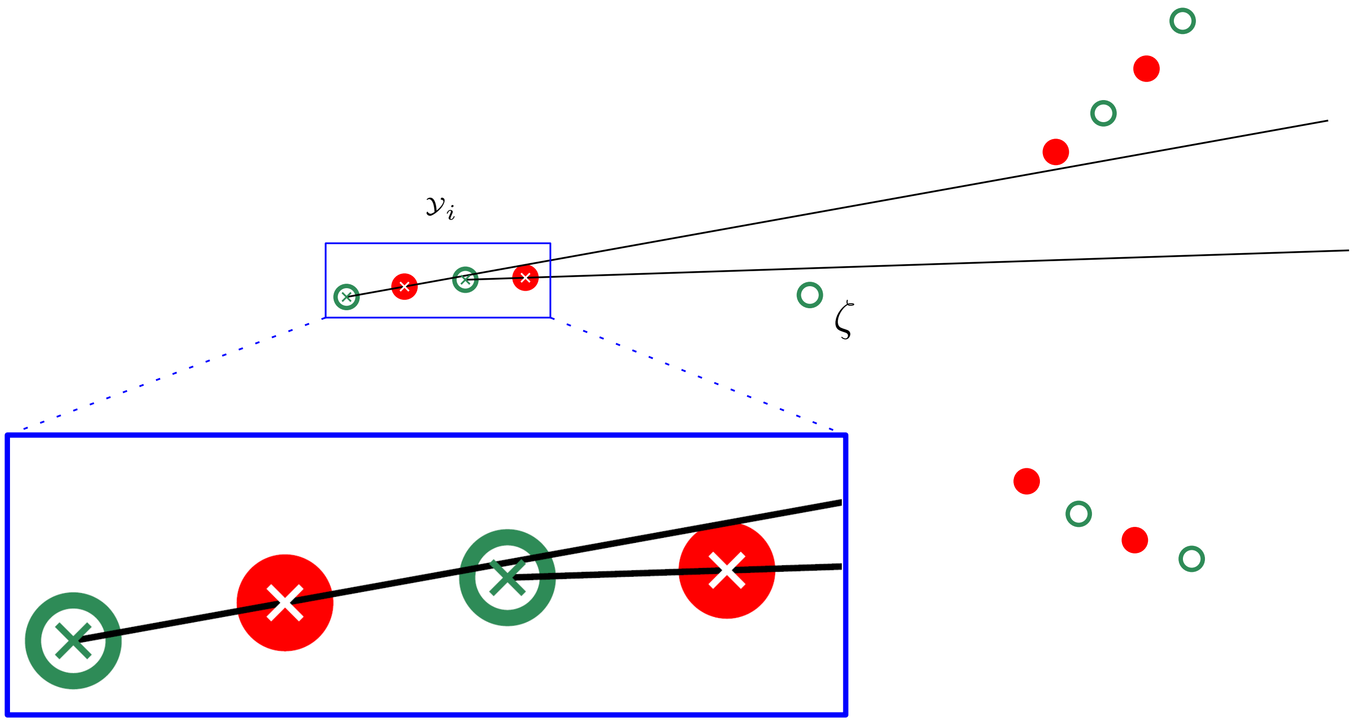


Figure 3: Highlighting details of the sun.

characterize the set C by the existence of a certain line (see Definition 7) and by quantities $\mathrm{r}(C)$, $\mathrm{s}(C)$, and $\mathrm{t}(C)$ (see Definition 12).

We start our proof with Lemma 13 in order to calculate what will happen in the second phase of the game. Lemma 14 then analyzes the first phase of the game while using the result of Lemma 13 in order to obtain the sum of Bob's score over both phases. In the end, we prove Theorem 6 as a straightforward corollary of Lemma 14.

Definition 7 *In a moment of a gameplay on the sun $\mathcal{S}_k$, denote $C \subseteq \mathcal{S}_k$ as the set of remaining cherries. For all lines that pass through ζ, we define the set $\mathcal{U}_C$ as the set of all closed half-planes defined by these lines. If $C \subseteq U \in \mathcal{U}_C$, then U is called a* bounding half-plane *for C.*

Lemma 8 *If Bob has been following the Careful greedy tactic from the beginning of the game on the sun $\mathcal{S}_k$, then, for $C \subseteq \mathcal{S}_k$ in a moment of gameplay when it is Alice's turn and a bounding half-plane does not yet exist, we have:*

1. *$\zeta \notin \mathrm{Ex}(C)$*
2. *Each beam $\mathcal{Y}$ is either fully remaining (that is $\mathcal{Y} \subseteq C$), or fully removed (that is $\mathcal{Y} \cap C = \emptyset$), or exactly the two innermost cherries (one green, one red) remain.*

Proof.

Item (1): From the hyperplane separation theorem; if ζ were an extremal point of C, there would be a bounding half-plane going directly through ζ.

Item (2): We proceed by induction on the number of taken cherries. Base case: $\mathcal{S}_k$ satisfies the properties since each beam is fully remaining. Assume that this holds up till some set $C_0 \subseteq \mathcal{S}_k$, and it is Alice's turn. Induction step: From C_0 Alice can only take a green cherry from some beam $\mathcal{Y}$. The beam $\mathcal{Y}$ is either fully remaining, or has exactly the two innermost cherries remaining by the induction hypothesis. After Alice takes the green cherry, a red cherry will be revealed on beam y. This will be the only red cherry available, and so by following the Careful greedy tactic, Bob will take this cherry. Therefore beam $\mathcal{Y}$ will end up either fully removed, or with the two innermost cherries remaining. These two moves will give $C_1 \subset C_0$ where $|C_1| = |C_0| - 2$ which either will maintain all properties or a bounding half-plane will have emerged. ☐

Lemma 9 *If Bob always follows the Careful greedy tactic, he will never reach FAIL.*

Proof. Before a bounding half-plane emerges, this is clear from Lemma 8.

After a bounding half-plane emerges, leaving $C \subset \mathcal{S}_k$, there will always be a beam $\mathcal{Y}$ such that all remaining cherries of $\mathcal{Y}$ are in $\mathrm{Ex}(C)$; therefore, if there are no extremal red cherries, the beam $\mathcal{Y}$ will have at least one extremal green cherry and no red cherries — Bob can take a green cherry here. Therefore, Bob will never reach the FAIL branch. ☐

Definition 10 *A beam $\mathcal{Y}$ is* semi-exposed *in C if it has $|\mathcal{Y} \cap \mathrm{R}(C)| = 2$ and $|\mathcal{Y} \cap \mathrm{R}(C) \cap \mathrm{Ex}(C)| = 1$.*

Lemma 11 *If Bob has been following the Careful greedy tactic from the beginning of the game on the sun $\mathcal{S}_k$, then, when it is Alice's turn, there will never be a semi-exposed beam.*

Proof. Assume to the contrary that a semi-exposed beam exists after a move by Bob and that it is the first time this happens. Lemma 8 shows it cannot happen before a bounding half plane has emerged. Suppose now we are at a point in the gameplay when a bounding half-plane exists and there is a semi-exposed beam after Bob's move.

Removing a cherry can either (1) reveal a beam in full, or (2) reveal no new cherries, or (3) reveal a single cherry. Clearly, neither (1) nor (2) can produce a semi-exposed beam.

In (3), a semi-exposed beam can be produced either by taking the centre cherry ζ, or by taking a green cherry from a beam which contains two red cherries (this beam then becomes semi-exposed). Neither of these moves can be performed by Bob because the Careful greedy tactic allows taking a green cherry only if it is from a beam and this beam does not contain any red cherries. We are left only with the option that it was Alice who generated a semi-exposed beam by (3).

Since taking a red cherry from a semi-exposed beam is the top priority in Bob's Careful greedy tactic, the only possible reason for Bob leaving a semi-exposed beam is if Alice leaves two semi-exposed beams after her turn. Alice can produce a semi-exposed beam by (3); however, (3) can only produce a single semi-exposed beam at a time. Therefore, there was at least one semi-exposed beam before Alice's move, which is a contradiction with this being the first time there is a semi-exposed beam after Bob's move. ☐

Definition 12 *In any moment of a gameplay on the sun $\mathcal{S}_k$, denote $C \subseteq \mathcal{S}_k$ as the set of remaining cherries.*

We define $\mathrm{s}(C)$ as the number of beams in C which have a single *remaining red cherry.*

Furthermore, we define $\mathrm{t}(C)$ as the number of beams in C which have at least one *remaining red cherry.*

Lemma 13 *Let $C \subseteq \mathcal{S}_k$ be a remaining subset of the sun obtained by Bob following the Careful greedy tactic such that $|C|$ is odd and a bounding half-plane for C exists. It is now Alice's turn.*

From the set C, Bob will obtain at least $\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C))$ red cherries from now until the end of the game (using the Careful greedy tactic).

Proof. This can be proved by induction on $|C|$.

If $|C| \leq 1$, then $\mathrm{r}(C) = \mathrm{s}(C)$, thus the statement holds trivially (it says that Bob will get at least 0 red cherries).

If $|C| > 1$, then we assume the statement holds for $|C'| = |C| - 2$. We proceed by case analysis.

- If Alice first takes a green cherry, then the lemma holds no matter what Bob does.
  - If Bob proceeds by taking a green cherry as well, then we see that $\mathrm{r}(C') = \mathrm{r}(C)$ and that $\mathrm{s}(C') = \mathrm{s}(C)$, thus it reduces exactly to the induction hypothesis.
  - If Bob proceeds by taking a red cherry, then we observe that $\mathrm{r}(C') = \mathrm{r}(C) - 1$ and that $|\,\mathrm{s}(C') - \mathrm{s}(C)| \leq 1$, thus:

    $$\frac{1}{2}(\mathrm{r}(C') - \mathrm{s}(C')) \geq \frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C)) - 1$$

    By the induction hypothesis, Bob will obtain at least $\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C)) - 1$ red cherries in the future. And, since Bob has just taken one red cherry, Bob obtains at least $\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C))$ red cherries in total.

- If Alice first takes a red cherry, then we need to consider the properties of the game state and Bob's strategy in order to show that the lemma holds.
  - If Alice took the red cherry from a beam with only this red cherry, then we observe that $\mathrm{r}(C') = \mathrm{r}(C) - 1$ and that $\mathrm{s}(C') = \mathrm{s}(C) - 1$, thus:

    $$\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C)) = \frac{1}{2}(\mathrm{r}(C') - \mathrm{s}(C'))$$

    The rest follows by applying the induction hypothesis in the same way as in the previous cases.
  - If, prior to Alice's move, there were two red cherries on the beam, then Bob's Careful greedy tactic leads to taking the other red cherry from the same beam. By Lemma 11, the second red cherry is guaranteed to be extremal. This gives $\mathrm{r}(C') = \mathrm{r}(C) - 2$ and $\mathrm{s}(C') = \mathrm{s}(C)$, thus the induction hypothesis guarantees that Bob will be able to obtain at least

    $$\frac{1}{2}(\mathrm{r}(C') - \mathrm{s}(C')) = \frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C)) - 1$$

    future red cherries; and, since Bob has just taken one red cherry, Bob obtains at least $\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C))$ red cherries in total. □

Lemma 14 *Let $C \subseteq \mathcal{S}_k$ be a remaining subset of the sun obtained by Bob following the Careful greedy tactic such that $|C|$ is odd and a bounding half-plane for C does not yet exist. It is now Alice's turn.*

There exists a half-plane $U \in \mathcal{U}_C$ such that Bob will obtain at least $\mathrm{r}(C) - \mathrm{t}(U \cap C)$ red cherries from now until the end of the game by using the Careful greedy tactic.

Proof. We proceed by induction on $\mathrm{r}(C)$. Note that all extremal cherries are green, by Lemma 8, and each of them lies on a beam that has a red cherry (in particular, the beam has the same number of green and red cherries).

Bob's Careful greedy tactic dictates to always take the neighboring red cherry from the same beam as Alice just took her cherry from. Finally, as our base case, we utilize Lemma 13 once a bounding half-plane emerges.

Induction step: Consider U from the induction hypothesis. Let C' be the set of cherries remaining from C after Alice's move and Bob's move, $|C'| = |C| - 2$. The induction hypothesis says that Bob will get at least $\mathrm{r}(C') - \mathrm{t}(U \cap C')$ cherries during the remainder of the game. Since $\mathrm{r}(C) = \mathrm{r}(C') + 1$ and $\mathrm{t}(U \cap C) \geq \mathrm{t}(U \cap C')$, the difference between the lemma statement and the number from the induction hypothesis is at most one. However, Bob has just taken a red cherry, so the lemma statement is satisfied.

Base case: Once a bounding half-plane emerges, Bob is guaranteed to obtain at least $\frac{1}{2}(\mathrm{r}(C) - \mathrm{s}(C))$ more red cherries by Lemma 13. We set U to be this bounding half-plane, thus $C = U \cap C$. We observe that

$$\mathrm{r}(C) = 2 \cdot \mathrm{t}(C) - \mathrm{s}(C)$$

or, in other words, that $\mathrm{s}(C) = 2 \cdot \mathrm{t}(U \cap C) - \mathrm{r}(C)$. Using this equality, Lemma 13 can be rewritten as: Bob is guaranteed to obtain at least

$$\frac{1}{2}(\mathrm{r}(C) - (2 \cdot \mathrm{t}(U \cap C) - \mathrm{r}(C)))$$

more red cherries. That is equal to $\mathrm{r}(C) - \mathrm{t}(U \cap C)$. This is what we wanted to prove. □

Theorem 6 *From the sun $\mathcal{S}_k$, Bob will get at least $\frac{3k-1}{2} = \frac{3}{4}\,\mathrm{r}(\mathcal{S}_k) - \frac{1}{2}$ red cherries by using the Careful greedy tactic, no matter how Alice plays. As a result, we obtain the desired property $\mathcal{M}(\mathcal{S}_k) \geq \frac{3}{4}\,\mathrm{r}(\mathcal{S}_k) - \frac{1}{2}$.* (restated)

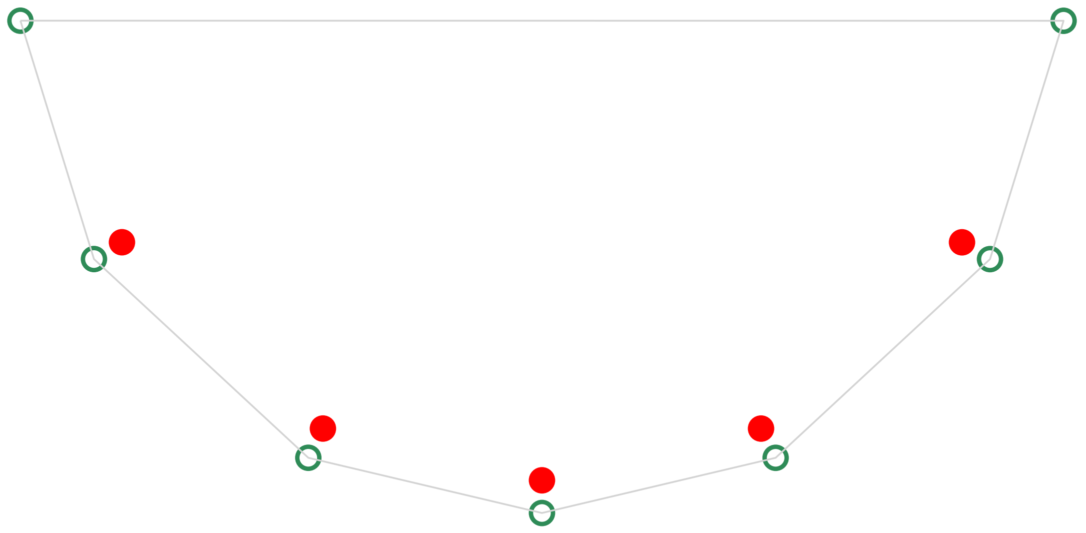


Figure 4: Moon ($\mathscr{L}_6$).

Proof. Given the properties of the sun $\mathcal{S}_k$, we see that any half-plane $U \in \mathcal{U}_{\mathcal{S}_k}$ has

$$\mathrm{t}(U \cap \mathcal{S}_k) \leq \frac{k+1}{2}$$

and thus, by Lemma 14, Bob is guaranteed to obtain at least

$$\mathrm{r}(\mathcal{S}_k) - \mathrm{t}(U \cap \mathcal{S}_k) \geq 2k - \frac{k+1}{2} = \frac{3k-1}{2}$$

red cherries using his Careful greedy tactic. □

4 Moon configuration

We present a family of even-sized cakes which we call the moon configuration, on which Bob can easily obtain all red cherries.

Definition 15 *Let $n \in \mathbb{N} \setminus \{0, 1\}$. We define the* moon *$\mathscr{L}_n$ (see Figure 4 for an example $\mathscr{L}_6$) as follows.*

Choose a centre point S and draw two circles; $\alpha(S, 1)$, called outer; and $\beta(S, 1-\varepsilon)$, called inner, where $0 < \varepsilon < 1 - \cos(90°/n)$. Draw n lines through S such that they are rotationally symmetric with a period of $180°/n$. Pick one line, called the main line. The main line defines two half-planes. The "upper" half-plane is discarded. The "other" half-plane will create the moon.

Place a green cherry at each intersection of any line with the outer circle α. Then place a red cherry at each intersection of any line except the main line with the inner circle β.

We have constructed a cake $\mathscr{L}_n$ where $\mathrm{r}(\mathscr{L}_n) = n-1$ and $\mathrm{g}(\mathscr{L}_n) = n+1$, while $\mathrm{Ex}(\mathscr{L}_n) = \mathrm{G}(\mathscr{L}_n)$.

Lemma 16 *The moon $\mathscr{L}_n$ has the following properties:*

1. *The removal of a green cherry will reveal a (single) red cherry.*
2. *The set $C \subset \mathscr{L}_n$, $|C| = |\mathscr{L}_n| - 2$, obtained by the removal of a green cherry followed by a removal of a red cherry, will be order-equivalent to $\mathscr{L}_{n-1}$.*

Proof. From the definition. □

Definition 17 *In the convex grabbing game on a moon, we say that a player follows the* Simple greedy tactic *if the player chooses a move according to this rule:*

Is there any extremal red cherry?
YES $\longrightarrow$ Take an extremal red cherry.
NO $\longrightarrow$ Take any extremal cherry.

Theorem 18 *From the moon $\mathscr{L}_n$ if Bob follows the Simple greedy tactic he will obtain all red cherries. This results in $\mathcal{M}(\mathscr{L}_n) = \mathrm{r}(\mathscr{L}_n) = n-1$.*

Proof. We proceed by induction on n.

In the case with $n = 2$, the moon will have four cherries in total; three extremal green cherries and one red cherry lying inside their triangle; therefore, Alice can select any green cherry and Bob will take the only red cherry by the Simple greedy tactic.

Assume that $n > 2$ and the theorem holds for $\mathscr{L}_{n-1}$. For Alice's first move a_1, only green cherries are available, and so she will take one of them. By Lemma 16, this will reveal a single red cherry, hence Bob, by following the Simple greedy tactic, will always take this red cherry for his first move b_1.

By Lemma 16 again, the remaining set of cherries $\mathscr{L}_n \setminus \{a_1, b_1\}$ is order-equivalent to $\mathscr{L}_{n-1}$. Therefore, by the induction hypothesis, Bob will obtain all $n-2$ red cherries from $\mathscr{L}_n \setminus \{a_1, b_1\}$; and, since he already took a red cherry in his first move, from $\mathscr{L}_n$ he obtains a total of $n - 2 + 1 = n - 1$ red cherries. □

5 Miscellaneous

In order to obtain configurations which are favourable for Alice on even-sized and odd-sized cakes, a single red cherry can be placed outside the convex hull for the sun configuration and moon configuration respectively.

Adding the extra red cherry swaps the parity of our constructions. We obtain the following cakes $\mathcal{C}$ and $\mathcal{D}$ that are good for Alice.

Corollary 19 *Theorem 6 implies that there exists an even-sized cake $\mathcal{C}$ such that $\mathcal{M}(\mathcal{C}) \leq \frac{1}{4}\,\mathrm{r}(\mathcal{C}) + \frac{1}{4}$. And, in a similar manner, Theorem 18 implies that there exists an odd-sized cake $\mathcal{D}$ with any desired $\mathrm{r}(\mathcal{D}) \in \mathbb{N}$ such that $\mathcal{M}(\mathcal{D}) = 0$.*

Furthermore, we would like to know what the optimal gameplay looks like in general. We came up with the following conjectures regarding the tactics which each player could employ in order to select their next move.

Conjecture 1 *Greedy-move conjecture.*
If $\mathrm{Ex}(C) \cap \mathrm{R}(C) \neq \emptyset$, there exists a move that takes a red cherry from $\mathrm{Ex}(C) \cap \mathrm{R}(C)$ such that the move is optimal.

Note that the Careful greedy tactic (Definition 5) and the Simple greedy tactic (Definition 17) are refinements of what the Greedy-move conjecture says.

Conjecture 2 *Strong greedy-move conjecture.*
If $\mathrm{Ex}(C) \cap \mathrm{R}(C) \neq \emptyset$, then every move that takes a red cherry is optimal.

We will soon show that, even though we don't know whether the Greedy-move conjecture and the Strong greedy-move conjecture hold, we can easily prove that the former implies the latter (while the other implication holds trivially).

Conjecture 3 *No-reveal-move conjecture.*
If $\mathrm{Ex}(C) \cap \mathrm{R}(C) = \emptyset$ and we have a set of non-revealing moves $N = \{c \in \mathrm{Ex}(C) \mid \mathrm{Ex}(C \setminus \{c\}) \cap \mathrm{R}(C) = \emptyset\}$ that is not empty, then there exists $c \in N$ such that selecting c is optimal.

We later found a counterexample that disproved the No-reveal-move conjecture, which we will show soon.

Proposition 20 *The Greedy-move conjecture implies the Strong greedy-move conjecture.*

Proof. Consider the following set of red cherries $\mathrm{R}_{\mathrm{ext}}(C) = \{c \in \mathrm{R}(C) \mid c \notin \mathrm{conv}(\mathrm{G}(C))\}$. We prove the proposition by induction on $|\,\mathrm{R}_{\mathrm{ext}}(C)|$. If $|\,\mathrm{R}_{\mathrm{ext}}(C)| = 1$, both conjectures are trivially equivalent.

Assume that the Greedy-move conjecture holds in general and that the Strong greedy-move conjecture holds for up to $|\,\mathrm{R}_{\mathrm{ext}}(C)| = n-1$ red cherries. We want to prove that the Strong greedy-move conjecture holds for up to $|\,\mathrm{R}_{\mathrm{ext}}(C)| = n$ red cherries. Seeking contradiction, assume that Alice has two possible moves taking a red cherry $c_i, c_j \in \mathrm{R}_{\mathrm{ext}}(C) \cap \mathrm{Ex}(C)$ that lead to different outcomes $\mathrm{B}(\mathbf{q})$.

If Alice starts by taking c_i, then by the induction hypothesis, c_j is among Bob's optimal moves. If Alice starts by taking c_j, then by the induction hypothesis, c_i is among Bob's optimal moves. Either way, this leaves $C' = C \setminus \{c_i, c_j\}$. In the first case, Alice ends up with $\mathrm{w}(c_i) + \mathrm{r}(C') - \mathcal{M}(C')$ points. In the second case, Alice ends up with $\mathrm{w}(c_j) + \mathrm{r}(C') - \mathcal{M}(C')$ points. Since they are both equal to $1 + \mathrm{r}(C') - \mathcal{M}(C')$, we obtain a contradiction. □

Proposition 21 *The No-reveal-move conjecture is false.*

Proof. We show a sketch of the proof through the construction in Figure 5.

In this construction, the only non-revealing first move is to select $a_1 = c_2$. If, in the gameplay $\mathbf{q}$, Alice starts by taking this green cherry c_2, giving $\mathbf{q} = (c_2, b_1, \ldots, b_5)$, then Bob can select c_4, giving $\mathbf{q} = (c_2, c_4, a_2, \ldots, b_5)$, and they end up with $A(\mathbf{q}) = 1$ because the remaining part of the cake gives $\mathcal{M}(C \setminus \{c_2, c_4\}) = 3$.

However, if Alice selects $a_1 = c_1$ for her first move, she reveals two red cherries at the same time. Alice is therefore able to take a red cherry in her second move, after Bob moves. For Bob's first two moves, in order

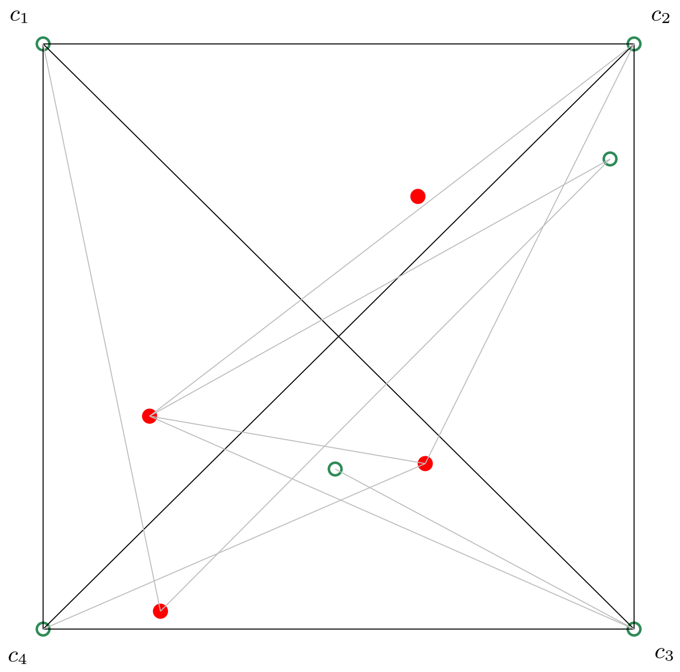


Figure 5: Our counterexample to the No-reveal-move conjecture, with lines added for visual aid.

for Alice to not obtain a second red cherry on her third move, he has to take one of the two red cherries which Alice revealed in her first move, and c_2; however, the order of Bob selecting these does not matter.

If Alice selects $a_3 = c_3$, she once again reveals two red cherries, and she is then guaranteed to be able to select a second red cherry in her fourth move. Therefore, by not selecting the non-revealing cherry in her first move, Alice is able to get a result of $A(\mathbf{q}) = 2$. $\square$

6 Conclusion

We solved the open problem from [3] by providing a construction that builds an odd-sized cake $\mathcal{S}_k$ such that $\mathcal{M}(\mathcal{S}_k) - (\mathrm{r}(\mathcal{S}_k) - \mathcal{M}(\mathcal{S}_k)) \geq x$ for any $x \in \mathbb{N}$. It could be interesting to know whether the result can be made even stronger. Now consider the value:

$$\gamma = \limsup_{p \to \infty} \left(\max_{\substack{\text{odd-sized} \\ \text{cake } \mathcal{C}}} \left\{ \frac{\mathcal{M}(\mathcal{C})}{\mathrm{r}(\mathcal{C})} \;\middle|\; \mathrm{r}(\mathcal{C}) = p \right\} \right)$$

Our construction provides a lower bound $\gamma \geq \frac{3}{4}$. On the other hand, [3] shows that Alice can always obtain at least one red cherry on any odd-sized cake. However, this only gives the trivial upper bound $\gamma \leq 1$. We pose a new open question of determining the value γ.

Analysis of gameplays would be easier if the state space of need-to-be-considered gameplays were limited by knowing which moves are optimal (or which moves cannot be optimal) in certain situations. We therefore leave the reader with another open question: Does the Greedy-move conjecture hold?

References

[1] Cibulka, J., Kynčl, J., Mészáros, V., Stolař, R., Valtr, P.: Graph sharing games: Complexity and connectivity. Theoret. Comput. Sci. 494, 49–62 (2013)

[2] Dvorak, M., Nicholson, S. (2021). Massively Winning Configurations in the Convex Grabbing Game on the Plane. arXiv:2106.11247 [Math]. http://arxiv.org/abs/2106.11247

[3] Matsumoto, N., Nakamigawa, T., Sakuma, T.: Convex Grabbing Game of the Point Set on the Plane. Graphs Combin. 36(1), 51–62 (2020)

[4] Micek, P., Walczak, B.: A graph-grabbing game. Combin. Probab. Comput. 20, 623–629 (2011)

[5] Pilz, A., Welzl, E.: Order on Order Types. International Symposium on Computational Geometry 34, 285–299 (2015)

[6] Seacrest, D.E., Seacrest, T.: Grabbing the gold. Discrete Math. 312, 1804–1806 (2012)

[7] Winkler, P.M.: Mathematical puzzles: A Connoisseur's Collection. A K Peters, Natick, MA (2003)

Yin-Yang Puzzles are NP-complete

Erik D. Demaine* Jayson Lynch† Mikhail Rudoy‡ Yushi Uno§

Abstract

We prove NP-completeness of Yin-Yang / Shiromaru-Kuromaru pencil-and-paper puzzles. Viewed as a graph partitioning problem, we prove NP-completeness of partitioning a rectangular grid graph into two induced trees (normal Yin-Yang), or into two induced connected subgraphs (Yin-Yang without 2×2 rule), subject to some vertices being pre-assigned to a specific tree/subgraph.

1 Introduction

The Yin-Yang puzzle is an over-25-year-old type of pencil-and-paper logic puzzle, the genre that includes Sudoku and many other puzzles made famous by e.g. Japanese publisher Nikoli.[1] Figure 1 shows a simple example of a puzzle and its solution. In general, a Yin-Yang puzzle consists of a rectangular $m \times n$ grid of unit squares, called ***cells***, where each cell either has a black circle, has a white circle, or is empty. The goal of the puzzle is to fill each empty cell with either a black circle or a white circle to satisfy the following two constraints:

- ***Connectivity constraint***: For each color (black and white), the circles of that color form a single connected group of cells, where connectivity is according to four-way orthogonal adjacency.
- ***2 × 2 constraint***: No 2×2 square contains four circles of the same color.

In this paper, we prove NP-completeness of deciding whether a Yin-Yang puzzle has a solution, with or without the 2×2 constraint.

1.1 Graph Partitioning

We can view Yin-Yang puzzles as a type of graph partitioning problem, by taking the dual graph with a vertex for each unit-square cell and edges between orthogonally adjacent vertices/cells. The result is a rectangular $m \times n$ grid graph, with some vertices precolored black or white. The goal is to complete a black/white coloring of the vertices subject to dual versions of the constraints. The

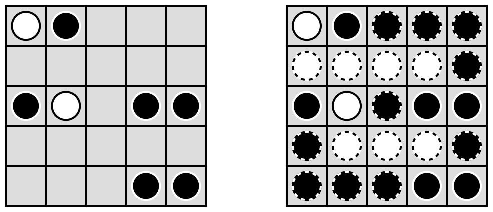


Figure 1: A simple Yin-Yang puzzle (left) and its unique solution (right). Circles added in the solution have dotted boundaries.

connectivity constraint above constrains each color class to induce a connected subgraph, so with this constraint alone, the problem is to partition the graph's vertices into two connected induced subgraphs. We thus also prove NP-completeness of this graph partitioning problem:

> ***Grid Graph Connected Partition Completion***: Given an $m \times n$ grid graph $G = (V, E)$ and given a partition of V into A, B, and U, is there a partition of V into A' and B' such that $A \subseteq A'$, $B \subseteq B'$, and $G[A']$ and $G[B']$ are connected?

The 2×2 constraint forbids induced 4-cycles in each color class, which is the ***thin*** constraint of [7, 14]. Any larger-than-4 induced cycle in a color class must enclose a vertex of the opposite color, so by the connectivity constraint, only one color class can have such a cycle and it can have only one such cycle, which (if it exists) must be exactly the boundary vertices of the $m \times n$ grid graph. If we exclude the possibility of a single outer cycle (e.g., by restricting to instances that precolor at least one boundary vertex of each color), then the problem (with both constraints) is to partition the vertices of a rectangular $m \times n$ grid graph into two induced (connected) trees. We also prove NP-completeness of this graph partitioning problem:

> ***Grid Graph Tree Partition Completion***: Given an $m \times n$ grid graph $G = (V, E)$ and given a partition of V into A, B, and U, is there a partition of V into A' and B' such that $A \subseteq A'$, $B \subseteq B'$, and $G[A']$ and $G[B']$ are trees?

*Massachusetts Institute of Technology, USA, edemaine@mit.edu

†University of Waterloo, Canada, jayson.lynch@uwaterloo.ca

‡LeapYear Technologies, mrudoy@gmail.com

§Osaka Prefecture University, uno@cs.osakafu-u.ac.jp

[1]However, Yin-Yang is not a Nikoli puzzle.

1.2 History

The origin of Yin-Yang is not clear. An early example of this puzzle is from 1994 in the (discontinued) Japanese puzzle magazine *Puzzler* [11] (or its original form in 1993). This reference calls the puzzle by the domestic Japanese name "白丸黒丸" ("Shiromaru-Kuromaru", which means "white circle / black circle"). It seems that the puzzle or slight variations have been invented independently many times under different names. Most recently, it is often referred to as the "Yin-Yang" puzzle, and we could find the puzzle introduced in some puzzle books, magazines, and websites, e.g., [35].

The computational complexity of puzzles has seen significant study, partly for the recreational element but also because many puzzles have direct connection to important problems such as geometric packing/partitioning or path/tree drawing under constraints. Surveys have been written on the topic of the computational complexity of games and puzzles [12, 20, 29]. Many pencil-and-paper puzzles have been shown to be NP-complete, including: Bag / Corral [17], Country Road [24], Dosun-Fuwari [27], Fillomino [40], Fillmat [39] Hashiwokakero [6], Heyawake [23], Hiroimono / Goishi Hiroi [5], Hitori [20, Section 9.2], Juosan [28] Kakuro / Cross Sum [41], Kurodoko [30], Kurotto [28], Light Up / Akari [33], LITS [34], Masyu / Pearl [18], Nonogram / Paint By Numbers [38], Numberlink [31], Nurikabe [32, 22], Pencils [36], Shakashaka [13, 2], Slitherlink [41, 40, 1], Spiral Galaxies / Tentai Show [19], Sto-Stone [4], Sudoku [41, 40], Tatamibari [3], Usowan [26], Yajilin [24], and Yosenabe [25].

Graph partitioning problems involve splitting the vertices of a graph into subsets based on various criteria. Common criteria include connectedness of the subgraphs, balancing the number or weight of vertices or edges in the partitions, and minimizing or maximizing the edge cut between partitions. Different objectives have a variety of applications; see [10] for a survey on the topic. Connected balanced partitions in grid graphs has been of specific interest with several NP-hardness results known, including 2-color weighted and only 3-rows [8], 3-color unweighted [9], and k-color in solid grid graphs [16]. On the positive side, Hettle et al. [21] give polynomial-time approximation algorithms for partitioning grid and planar graphs, and apply these to real-world police and fire-department districting problems. We use only two colors in a solid unweighted grid, but introduce the new constraint of pre-assigning more than one vertex to each partition. When exactly one vertex of each color is already assigned, the problem is often called a rooted partition problem.

2 Hardness Proof

All of the problems we consider are obviously in NP, by using the partition/solution as a witness. We show that solving $n \times n$ Yin-Yang puzzles is NP-hard with or without the 2×2 constraint by reductions from the following NP-hard problem [15]:

Planar 4-Regular Tree-Residue Vertex Breaking (TRVB): Given a planar 4-regular multigraph, is there a subset of vertices that, after being "broken", results in a single connected tree? ***Breaking*** a vertex involves deleting that vertex from the graph and adding a degree-1 vertex to each of its neighbors (thus modifying the edges incident to the broken vertex to instead be incident to a newly created degree-1 vertex):

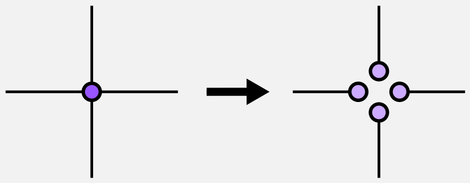


Figure 2 shows an example of this problem.

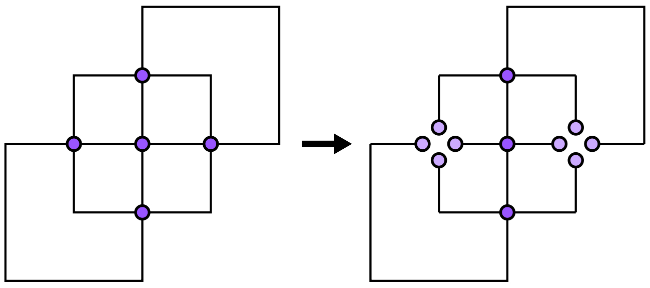


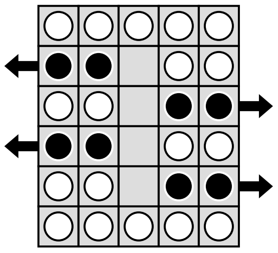
Figure 2: A 5-vertex instance of Planar 4-Regular TRVB (left) and a solution (right).

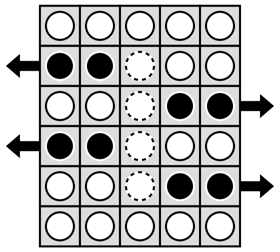
We will use the fact that planar maximum-degree-4 multigraphs can be drawn (with grid-routed edges) in a square grid of area $O(n^2)$ in polynomial time, as in Figure 2. We can use such an embedding for simple graphs (e.g., [37]), and adapt it to multigraphs by subdividing multiple edges and loops, applying the simple embedding, and removing the subdivision vertices.

Before diving into the reductions, we develop a tool that helps force local solutions to Yin-Yang puzzles:

Lemma 1 *In a valid Yin-Yang puzzle solution (with or without the* 2×2 *constraint), no* 2×2 *square can contain diagonally opposite white and black circles.*

Proof. Consider the black circles. These two circles are not orthogonally adjacent, and thus must be connected by a path of black circles. However, this path


(a) Vertex gadget


(b) Broken solution

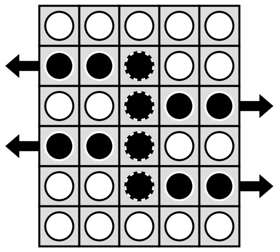

(c) Unbroken solution

Figure 3: Vertex gadget without the 2×2 constraint and its two possible local solutions. The arrows represent the four outgoing edges.

will separate the pair of white circles which are also not orthogonally adjacent and thus must be connected by their own path of white circles. □

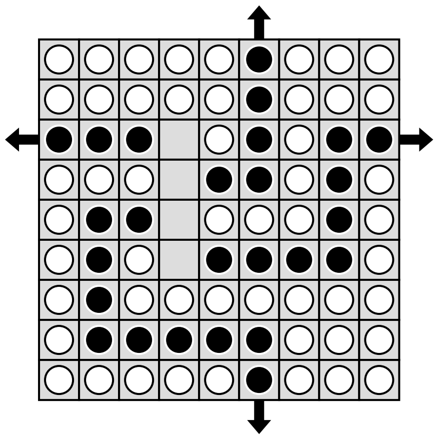
2.1 Connected Partition: Without 2×2 Constraint

In this section, we prove NP-hardness of Grid Graph Connected Partition Completion, or equivalently, Yin-Yang puzzles without the 2×2 constraint. We present this proof first as it is simpler but uses the same ideas.

In this reduction, edges will be represented by orthogonal paths (***wires***) of black circles, and everything that is not an edge or vertex gadget will be filled with white circles.

Vertex Gadget. Figure 3a shows the vertex gadget. If a solution fills the top empty cell (say) black or white, then by repeated application of Lemma 1, all four empty cells must be filled the same color. In the local solution of Figure 3b, the black-circle wires are disconnected from each other; while in the local solution of Figure 3c, the black-circle wires are all connected together. Thus these two local solutions correspond to breaking and not breaking a vertex of the TRVB instance.

Layout. We take an orthogonal grid drawing of the given TRVB graph, and then scale the grid by a factor of 9. This scaling allows us to place the 9×9 tiled versions of the vertex gadget and edge gadgets (corners and straights) shown in Figure 4, which make it easy to align black-circle wires in row 3 and column 6. All re-


(a) Vertex gadget

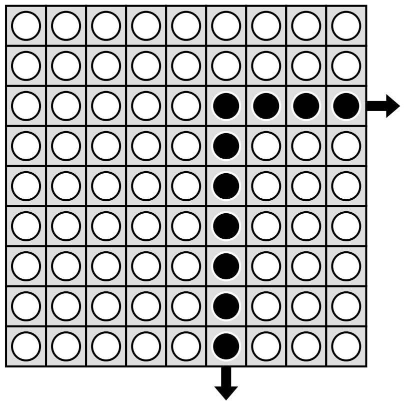

(b) One of six edge gadgets

Figure 4: 9×9 tiling versions of gadgets without the 2×2 constraint.

maining cells are filled with white circles, leaving empty only the four cells in each vertex gadget.

Theorem 2 *It is NP-complete to decide whether there is a solution to a Yin-Yang puzzle without the 2×2 constraint, or Grid Graph Connected Partition Completion, on an $n \times n$ grid.*

Proof. If the TRVB instance has a solution, we fill broken vertices with four white circles (as in Figure 3b) and unbroken vertices with four black circles (as in Figure 3c). All edge gadgets touching an unbroken vertex are connected, and thus the fact that the TRVB solution is connected ensures the black circles form a single connected component. For a broken vertex, the added white circles connect the white circles in the four faces of the graph drawing. The TRVB solution being a tree ensures that this yields a single white connected component.

Conversely, consider any solution to the Yin-Yang puzzle. As argued above, each vertex gadget must be

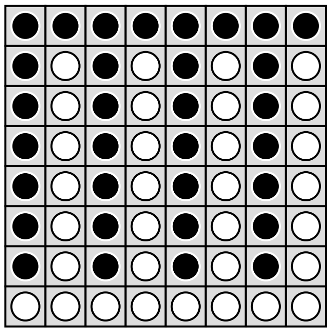


Figure 5: Background filler satisfying the 2×2 constraint.

solved using one of the two valid local solutions from Figure 3, and therefore we can translate this Yin-Yang solution to an assignment of whether to break each vertex in the TRVB instance. We claim that this assignment is in fact a solution to the TRVB instance. The region containing the black circles in the Yin-Yang solution has the same shape (in particular, toplogy) as the graph resulting from breaking the broken vertices in the candidate solution to the TRVB instance. If the resulting graph were disconnected, then the black circles would also form a disconnected region, contradicting the Yin-Yang connectivity constraint on black. If the resulting graph had a cycle, then the black circles would also form a nontrivial cycle, which would separate the white circles interior and exterior to that cycle, contradicting the Yin-Yang connectivity constraint on white. Therefore the resulting graph is connected and acyclic, so we have a solution to the TRVB instance. □

2.2 Tree Partition: With 2×2 Constraint

In this section, we prove NP-hardness of Grid Graph Tree Partition Completion as well as Yin-Yang puzzles (with both constraints). We fully precolor the boundary vertices to not form a cycle, so these problems become equivalent. The reduction idea is the same as the proof without the 2×2 constraint from Section 2.1, but respecting this constraint requires a more complicated filler, and because of this filler, a more complex vertex gadget. To get a sense of the overall structure, refer ahead to Figure 10 for a full example.

2.2.1 Background Filler

The background filler consists of alternating columns of white and black circles, except for a full row of black circles at the top and a full row of white circles at the bottom (to ensure connectivity); see Figure 5. On top of this filler gadget, we draw vertex and edge gadgets.

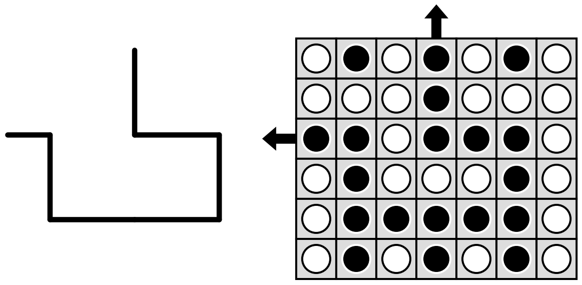


Figure 6: An edge route (left) and the corresponding edge gadget (right).

2.2.2 Edge Gadget

We again represent edges as orthogonal paths (wires) of black circles but now, wherever an edge travels horizontally, we also add a row of white circles immediately above the path, except where the edge turns upward; see Figure 6. Because of the background filler, each horizontal segment of an edge gadget will have black downward tendrils (paths) from every other column, similar to the downward black tendrils from the top row of the filler. (Similarly, the white circles immediately above a horizontal segment of an edge gadget will have white upward tendrils from every other column, similar to the upward white tendrils from the bottom row of the filler.) Edge gadgets can travel vertically only on the black parity of the background filler, so that these circles of the edge gadget match the background filler.

2.2.3 Vertex Gadget

Figure 7 shows the vertex gadget. By the 2×2 constraint, the leftmost two empty cells must be filled with circles of opposite color; refer to Figure 8. Once this choice gets made, repeated application of Lemma 1 forces the coloring of the horizontal rows of empty squares to be uniform and opposite from each other, and then forces the rightmost two empty cells to match the leftmost two empty cells. The top three empty cells are then forced to be filled in with circles of the same color as the bottom row, in order to prevent local isolation of black or white circles.

Figure 8 shows the two possible resulting local solutions. Figure 8a corresponds to breaking the TRVB vertex, as it keeps the four incident edges disconnected from each other; while Figure 8b corresponds to not breaking the corresponding TRVB vertex, as it connects together the four incident edges. Both solutions also connect together the five white paths at the top and the two outermost white paths on the bottom, and the broken solution further connects these white paths to the three middle white paths at the bottom. These connections correspond to exactly one white region per face around the vertex: one face for a broken vertex and

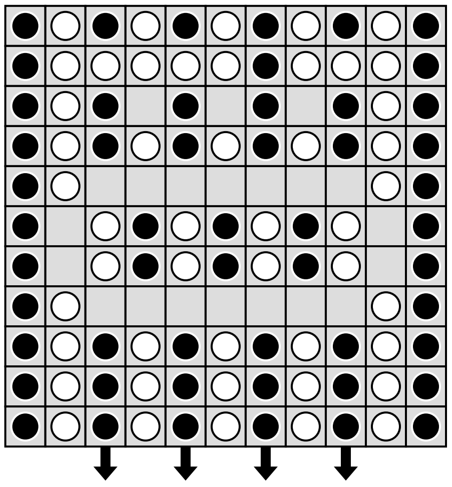


Figure 7: Vertex gadget. The arrows represent the four outgoing edges.

four faces for an unbroken vertex.

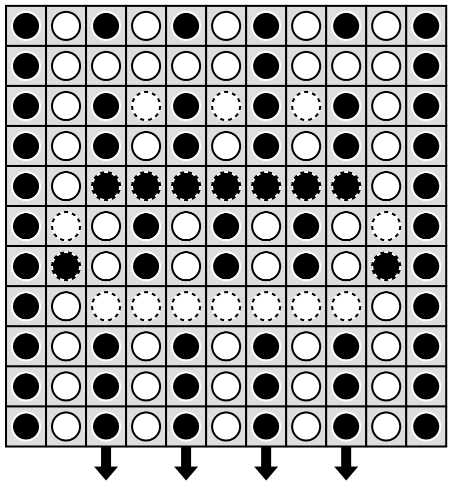
2.2.4 Reduction Layout

We take an orthogonal grid drawing of the given TRVB graph, and then scale the grid by a factor of 16. This scaling allows us to place the 16×16 tiled versions of the vertex gadget and edge gadgets (corners and straights) shown in Figure 9, which make it easy to align black-circle wires in row 12 and column 11. Any absent 16×16 squares get filled with alternating columns of black and white. Then we add an extra row at the top and bottom of the puzzle, filled with black and white circles respectively, to complete the filler of Figure 5. Finally, for one topmost horizontal segment of an edge gadget, in a black-parity column, we change one of the cells above the segment from white to black, thereby attaching the edge gadget to a black tendril and thus the top row of black circles; we call the changed cell the ***exceptional cell***.

Figure 10 shows a complete example of the reduction applied to the instance from Figure 2, and Figure 11 shows the corresponding solution.

Theorem 3 *It is NP-complete to decide whether there is a solution to a Yin-Yang puzzle (with both constraints), or Grid Graph Tree Partition Completion, on an $n \times n$ grid.*

Proof. Because our reduction instance has black and white circles on the boundary, the Yin-Yang puzzle with 2×2 constraint is equivalent to Grid Graph Tree Partition Completion. Furthermore, if the black circles form a connected and acyclic subset of cells, then so do the white circles: if the white circles formed an induced cycle of length > 4, then there would be black circles both interior (because the cycle is induced) and exterior (on the boundary), a contradiction. Therefore, it suffices to prove that the black circles are connected and acyclic if and only if the chosen broken/unbroken solutions from Figure 8 for each vertex gadget corresponds to a solution to the TRVB instance.


(a) Broken vertex solution.

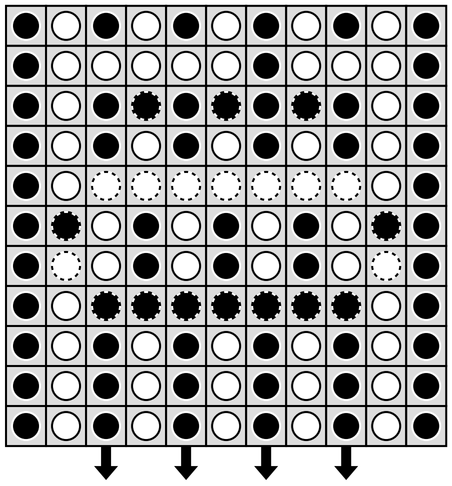
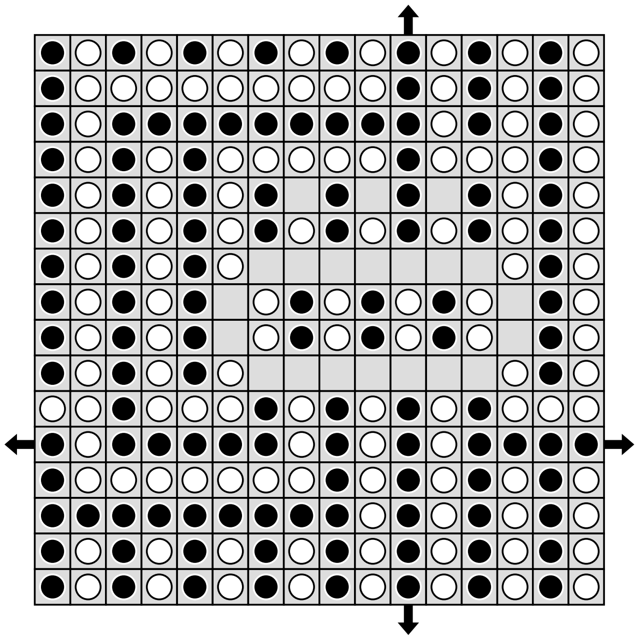

(b) Unbroken vertex solution.

Figure 8: The two local solutions to the vertex gadget of Figure 7.

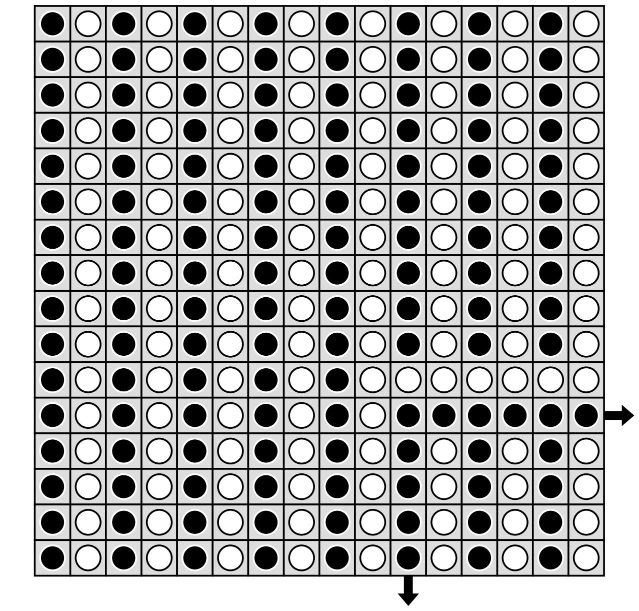
Define the ***important*** cells to consist of the following:

1. black edge wires, or more precisely, the shortest paths among black circles connecting pairs of the vertex gadgets' ports (marked with arrows in Figure 7); and
2. the bottom row of initially empty cells in each vertex gadget (instance of Figure 7), together with the shortest paths of black circles within the gadget connecting those cells to the ports (arrows).

The important cells directly represent the connectivity of the TRVB instance, so the important black circles



(a) Vertex gadget



(b) One of six edge gadgets

Figure 9: 16×16 tiling versions of the vertex and edge gadgets from Figures 7 and 6.

are connected and acyclic if and only if the Yin-Yang solution corresponds to a TRVB solution.

Thus it suffices to show that the important black circles are connected and acyclic if and only if all of the black circles are connected and acyclic. Unimportant black circles are either part of a vertex gadget, part of the top row, or part of a downward black tendril from the top row or a horizontal segment of an edge gadget. Figure 8 shows that unimportant black circles within a vertex gadget are all attached to the downward black tendrils above the gadget, with one acyclic connected component per tendril, aside from a few black circles (in the unbroken case) attached directly to important black circles. For unimportant black circles in black tendrils, we can trace the tendrils up to find their connection to either important black circles or the top row. (Tendrils are never connected to black circles at their bottom, except to $O(1)$ black circles within a vertex gadget.) The black tendrils that drop from an important black circle are all connected together if and only if the important black circles are themselves connected. The black tendrils that drop from the top row are all connected together via the top row of unimportant black circles, and the exceptional cell connects one of these tendrils to the important black circles. Therefore the unimportant black circles form trees attached to the important black cells, so they do not affect connectivity or acyclicity. □

3 Open Problems

Our reduction from Planar 4-regular TRVB is parsimonious (preserves the number of solutions). It is unknown whether the Another Solution Problem or counting versions of TRVB are (NP- or #P-) hard, but such results would carry over to Yin-Yang puzzles.

In our reductions, we fill almost the entire board with circles. However, elegant puzzles tend to have only a few pre-marked circles and so one may wonder whether this problem remains hard with a much sparser clue set. It seems likely small modifications to our reduction would allow for reductions with only $O(n)$ circles, but getting $o(n)$ would require fundamentally different techniques and some sort of clever encoding of problems in the relative spacing of the circles. On the extreme, it seems reasonably likely this problem is FPT with respect to the number of pre-assigned circles.

Both from a computational and puzzle design standpoint, we could imagine generalizing the number of colors of circles to more than two, or looking at other families of graphs. The 2×2 constraint can generalize to forbidding all vertices around a face from being the same color in an embedded graph. It seems very natural to look at such puzzles on triangular and hexagonal grids and in those cases the three-fold symmetry may aesthetically adapt to three colors. Generalizing the number of colors trivially remains NP-hard, as we can simply add a row for each additional color at the top of our two color reduction, ensuring those new colors will not be able to be placed. We also believe similar proof techniques will work for hexagonal and triangular grids.

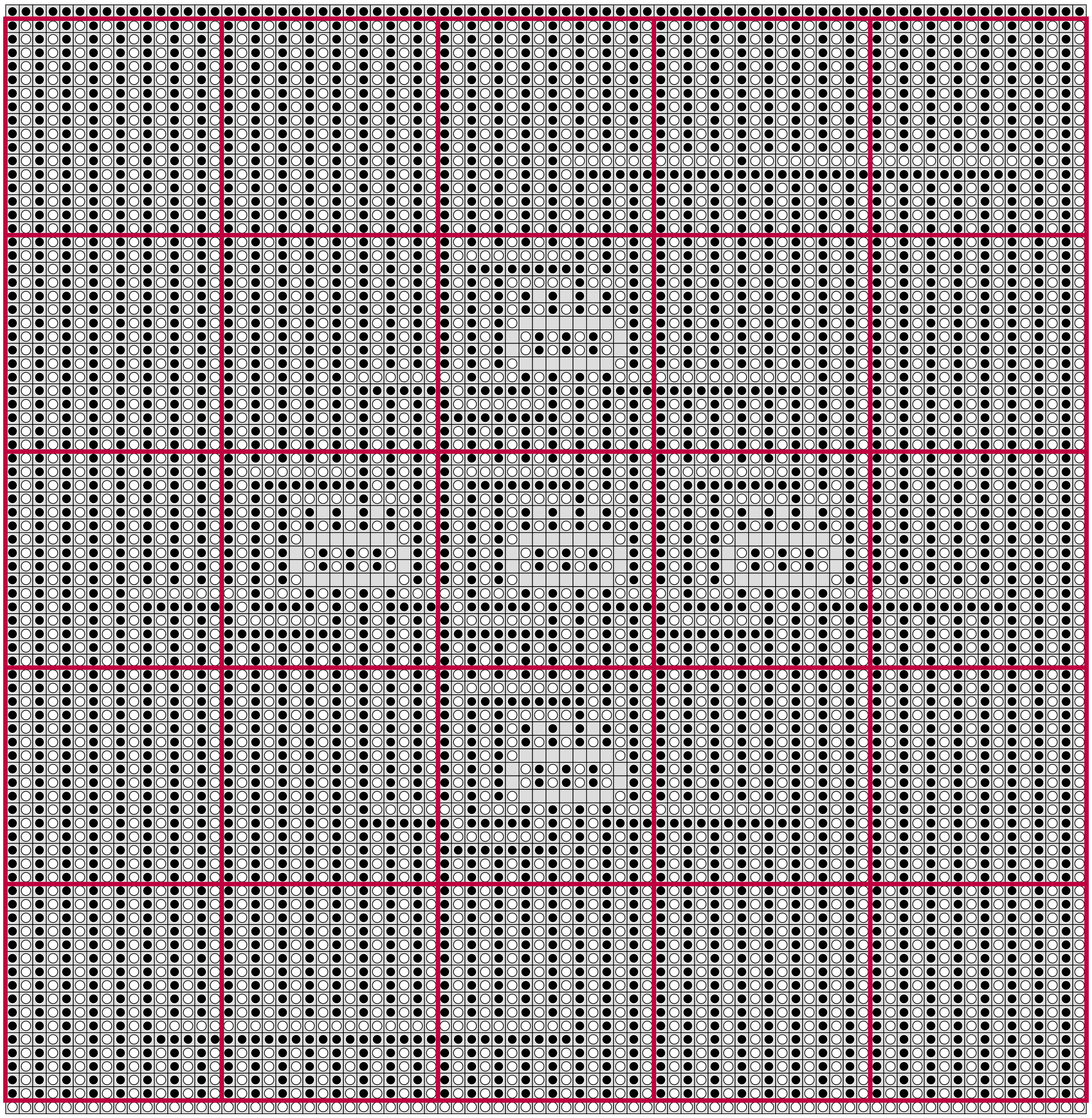


Figure 10: Full reduction from the 5-vertex instance of TRVB from Figure 2.

Acknowledgments

This research started at the 34th Bellairs Winter Workshop on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint in 2019. We thank the other workshop participants for fruitful discussions and for providing an inspiring research atmosphere.

Research supported in part by NSERC and JSPS KAKENHI Grant Numbers JP17K00017, JP21K11757 and JP20H05964.

Most figures of this paper were drawn using SVG Tiler [https://github.com/edemaine/svgtiler]. Source files for these figures are freely available [https://github.com/edemaine/yin-yang-svgtiler].

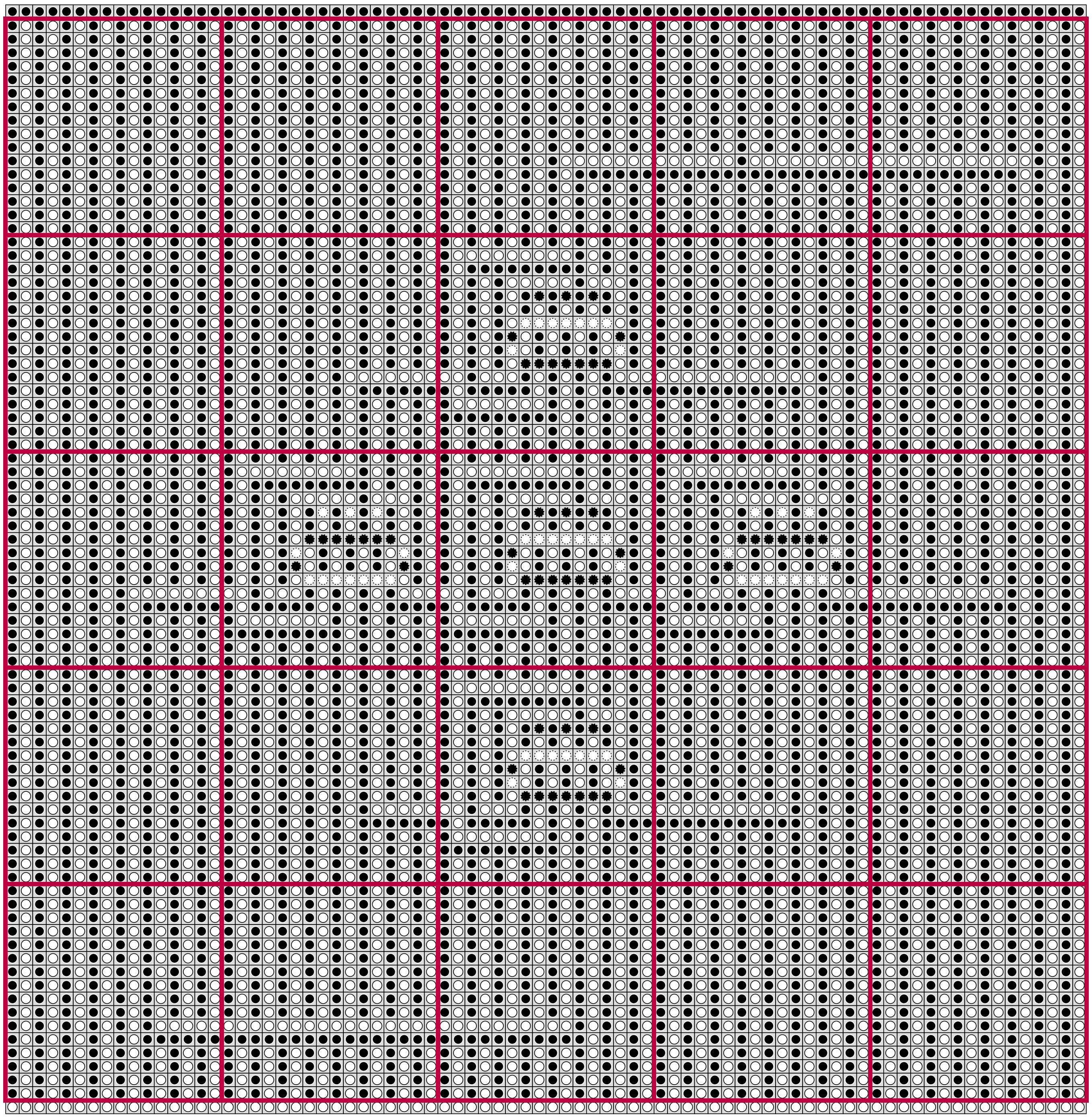


Figure 11: Solution to the puzzle in Figure 10 corresponding to the TRVB solution in Figure 2.

References

[1] Zachary Abel, Jeffrey Bosboom, Erik D. Demaine, Linus Hamilton, Adam Hesterberg, Justin Kopinsky, Jayson Lynch, and Mikhail Rudoy. Who witnesses The Witness? Finding witnesses in The Witness is hard and sometimes impossible. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 3:1–3:21, La Maddalena, Italy, June 2018.

[2] Aviv Adler, Michael Biro, Erik Demaine, Mikhail Rudoy, and Christiane Schmidt. Computational complexity of numberless Shakashaka. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015)*, Kingston, Canada, August 2015.

[3] Aviv Adler, Jeffrey Bosboom, Erik D. Demaine,

Martin L. Demaine, Quanquan C. Liu, and Jayson Lynch. Tatamibari is NP-complete. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, LIPIcs, 2020.

[4] Addison Allen and Aaron Williams. Sto-stone is NP-complete. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG 2018)*, pages 28–34, 2018.

[5] Daniel Andersson. HIROIMONO is NP-complete. In *Proceedings of the 4th International Conference on Fun with Algorithms (FUN 2007)*, volume 4475 of *Lecture Notes in Computer Science*, pages 30–39, 2007.

[6] Daniel Andersson. Hashiwokakero is NP-complete. *Information Processing Letters*, 109(19):1145–1146, 2009.

[7] Esther M. Arkin, Sándor P. Fekete, Kamrul Islam, Henk Meijer, Joseph S. B. Mitchell, Yurai Núñez-Rodríguez, Valentin Polishchuk, David Rappaport, and Henry Xiao. Not being (super) thin or solid is hard: A study of grid hamiltonicity. *Computational Geometry: Theory and Applications*, 42(6–7):582–605, 2009.

[8] Ronald Becker, Isabella Lari, Mario Lucertini, and Bruno Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115–125, 1998.

[9] Cedric Berenger, Peter Niebert, and Kevin Perrot. Balanced connected partitioning of unweighted grid graphs. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *LIPIcs*, 2018.

[10] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Selected Results and Surveys on Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, 2016.

[11] 世界文化社. しろまるくろまる. パズラー *(Puzzler)*, 150, May 1994.

[12] Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In *Games of No Chance 3*, pages 3–56. Cambridge University Press, 2009. arXiv:cc.CC/0106019.

[13] Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of Shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97-A(6):1213–1219, 2014.

[14] Erik D. Demaine and Mikhail Rudoy. Hamiltonicity is hard in thin or polygonal grid graphs, but easy in thin polygonal grid graphs. arXiv:1706.10046, 2017. https://arXiv.org/abs/1706.10046.

[15] Erik D. Demaine and Mikhail Rudoy. Tree-Residue Vertex-Breaking: a new tool for proving hardness. In *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *LIPIcs*, pages 32:1–32:14, 2018.

[16] Andreas Emil Feldmann. Fast balanced partitioning is hard even on grids and trees. *Theoretical Computer Science*, 485:61–68, 2013.

[17] Erich Friedman. Corral puzzles are NP-complete. https://erich-friedman.github.io/papers/corral.pdf, August 2002.

[18] Erich Friedman. Pearl puzzles are NP-complete. https://erich-friedman.github.io/papers/pearl.pdf, August 2002.

[19] Erich Friedman. Spiral Galaxies puzzles are NP-complete. https://erich-friedman.github.io/papers/spiral.pdf, March 2002.

[20] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation.* A K Peters, July 2009.

[21] Cyrus Hettle, Shixiang Zhu, Swati Gupta, and Yao Xie. Balanced districting on grid graphs with provable compactness and contiguity. arXiv:2102.05028, 2021. https://arXiv.org/abs/2102.05028.

[22] Markus Holzer, Andreas Klein, and Martin Kutrib. On the NP-completeness of the NURIKABE pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on Fun with Algorithms (FUN 2004)*, pages 77–89, Isola d'Elba, Italy, May 2004.

[23] Markus Holzer and Oliver Ruepp. The troubles of interior design—a complexity analysis of the game Heyawake. In *Proceedings of the 4th International Conference on Fun with Algorithms (FUN 2007)*, volume 4475 of *Lecture Notes in Computer Science*, pages 198–212, 2007.

[24] Ayaka Ishibashi, Yuichi Sato, and Shigeki Iwata. NP-completeness of two pencil puzzles: Yajilin and Country Road. *Utilitas Mathematica*, 88:237–246, 2012.

[25] Chuzo Iwamoto. Yosenabe is NP-complete. *Journal of Information Processing*, 22(1):40–43, 2014.

[26] Chuzo Iwamoto and Masato Haruishi. Computational complexity of usowan puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 101(9):1537–1540, 2018.

[27] Chuzo Iwamoto and Tatsuaki Ibusuki. Dosun-fuwari is NP-complete. *Journal of Information Processing*, 26:358–361, 2018.

[28] Chuzo Iwamoto and Tatsuaki Ibusuki. Polynomial-time reductions from 3SAT to kurotto and juosan puzzles. *IEICE Transactions on Information and Systems*, 103(3):500–505, 2020.

[29] Graham Kendall, Andrew Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[30] Jonas Kölker. Kurodoko is NP-complete. *Journal of Information Processing*, 20(3):694–706, 2012.

[31] Kouichi Kotsuma and Yasuhiko Takenaga. NP-completeness and enumeration of Number Link puzzle. *IEICE Technical Report*, 109(465):1–7, March 2010.

[32] Brandon McPhail. The complexity of puzzles. Undergraduate thesis, Reed College, Portland, Oregon, 2003.

[33] Brandon McPhail. Light Up is NP-complete. http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf, 2005.

[34] Brandon McPhail. Metapuzzles: Reducing SAT to your favorite puzzle. CS Theory talk, December 2007.

[35] Gareth Moore. *Paper, Pencil & You: Mindfulness: Relaxing Brain-Training Puzzles for Stressed-Out People*. Greenfinch, 2020.

[36] Daniel Packer, Sophia White, and Aaron Williams. A paper on Pencils: A pencil and paper puzzle: Pencils is NP-complete, pencils. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG 2018)*, page 35, 2018.

[37] Achilleas Papakostas and Ioannis G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry: Theory and Applications*, 9(1):83–110, 1998.

[38] Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, May 1996.

[39] Akihiro Uejima and Hiroaki Suzuki. Fillmat is NP-complete and ASP-complete. *Journal of Information Processing*, 23(3):310–316, 2015.

[40] Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, University of Tokyo, Tokyo, Japan, January 2003.

[41] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E86-A(5):1052–1060, 2003. Also IPSJ SIG Notes 2002-AL-87-2, 2002.

Constant Delay Lattice Train Schedules

Jean-Lou De Carufel[*][‡] Darryl Hill[†] Anil Maheshwari[†][‡] Sasanka Roy[§]
Luís Fernando Schultz Xavier da Silveira[†]

Abstract

The following geometric vehicle scheduling problem has been considered: given continuous curves $f_1, \ldots, f_n : \mathbb{R} \to \mathbb{R}^2$, find non-negative delays $t_1, \ldots, t_n$ minimizing $\max\{t_1, \ldots, t_n\}$ such that, for every distinct i and j and every time t, $|f_j(t - t_j) - f_i(t - t_i)| > \ell$, where ℓ is a given safety distance.

We study a variant of this problem where we consider trains (rods) of fixed length ℓ that move at constant speed and sets of train lines (tracks), each of which consisting of an axis-parallel line-segment with endpoints in the integer lattice $\mathbb{Z}^d$ and of a direction of movement (towards ∞ or $-\infty$). We are interested in upper bounds on the maximum delay we need to introduce on any line to avoid collisions, but more specifically on universal upper bounds that apply no matter the set of train lines.

We show small universal constant upper bounds for $d = 2$ and any given ℓ and also for $d = 3$ and $\ell = 1$. Through clique searching, we are also able to show that several of these upper bounds are tight.

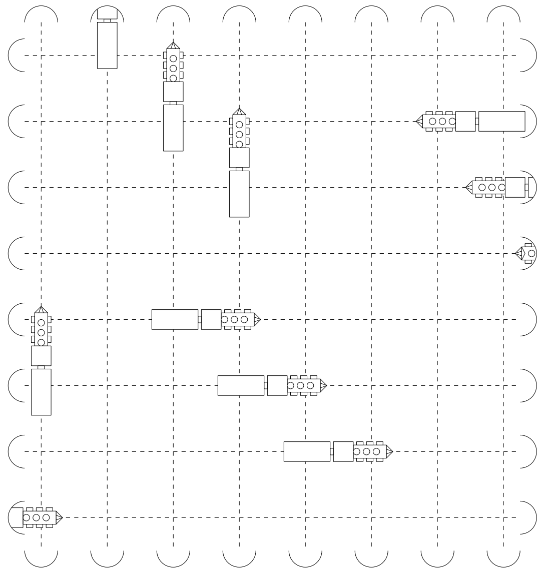


Figure 1: Train lines on axis parallel tracks with stations (here seen as tunnels) on the integer lattice.

1 Introduction

Ajaykumar and Roy [1] considered the following problem: given a set $S = \{f_1, \ldots, f_n\}$ of curves, find a maximum subset $S' \subseteq S$ such that, for any two distinct $f_i, f_j \in S'$, there is no time $0 \leqslant t \leqslant 1$ when $f_i(t) = f_j(t)$. Though there are some approximation algorithms, it is shown that most versions of these problems are NP-hard. This is true even when the curves are same-sized L-shapes and trajectories have the same constant speed. During a talk by Sasanka Roy on this problem at Stony Brook University, Joseph S. B. Mitchell and Esther M. Arkin posed the following variation: given $S = \{f_1, \ldots, f_n\}$ where $f_i : \mathbb{R} \to \mathbb{R}^2$ is a continuous function, we wish to assign delays $t_i \geqslant 0$ so that, for any two distinct $f_i, f_j \in S$, there is no time $t \in \mathbb{R}$ when $|f_j(t - t_j) - f_i(t - t_i)| \leqslant \ell$; we also wish that the maximum delay is minimized.

Automated guided vehicles (AGVs) and automated vehicle routing are currently some of the most prolific fields in the domain of motor vehicle technology, enjoying a large body of research addressing a large number of related algorithmic and optimization problems. The field of automated guided vehicles has a long and rich history and is closely related to our work. For an elaborate review, the reader is referred to a survey by Vis [15], though [3] is a more recent one. A central topic in this field is collision avoidance. Kim and Tanchoco [5] propose an algorithm based on Dijkstra's that facilitates conflict-free routing of AGVs. It runs in $\mathcal{O}(v^4 n^2)$, where v is the number of vehicles and n is the number of nodes, i.e., path intersections. Arora et al. [14] used techniques based on game theory to design a methodology for AGV traffic control. Yan et al. [16] studied collision-free routing of AGVs on both unidirectional and bidirectional paths using digraphs. For more information on conflict-free routing of AGVs, the reader is referred to [6, 9, 7]. For an elaborate survey on routing and scheduling algorithms for AGVs, see [8]. Other related work in this area includes the study of One Way Road Networks (OWRNs) by Ajaykumar et al. [4], its motivation [12] and [10, 2, 13].

[*]School of Computer Science and Electrical Engineering, University of Ottawa, Ottawa, Canada
[†]School of Computer Science, Carleton University, Ottawa, Canada
[‡]Research supported in part by NSERC
[§]Indian Statistical Institute, Kolkata, India

1.1 New Results

In this paper, we study a simple vehicle scheduling problem in which, many times, we are guaranteed schedules with at most a constant delay. We consider trains of fixed length ℓ that move at constant speed and sets of train lines (tracks), each of which consisting of an axis-parallel line-segment with endpoints in the integer lattice $\mathbb{Z}^d$ and of a direction of movement (towards ∞ or $-\infty$). For an example see Figure 1. We are interested in upper bounds on the maximum delay we need to introduce on any line to avoid collisions, but more specifically on universal upper bounds that apply no matter the set of train lines.

We show small universal constant upper bounds for $d = 2$ and any given ℓ (Theorem 9) and also for $d = 3$ and $\ell = 1$ (Theorem 10). Through clique searching, we are also able to show that several of these upper bounds are tight (Section 4). The results are summarized in Table 1.

2 Preliminaries

Definition 1 *A (d-dimensional) train line consists of:*

- *A track, which is a line segment in $\mathbb{R}^d$ with distinct endpoints distinguished between a departure point and an arrival point;*
- *A train length, which is a positive real number; and*
- *A speed, which is also a positive real number.*

Furthermore, a set of train lines with non-overlapping (but possibly crossing) tracks is called a train network.

Definition 2 *For a set of real numbers X and a real number y, we denote $\{x + y : x \in X\}$ by $X + y$.*

Definition 3 *A (collision-free) schedule for a train network is an assignment of a delay, which is a non-negative real number, to each of its lines. It also must result in no collisions between the trains. More precisely, for any two lines whose tracks cross, the open intervals $(0; \ell/v) + t + \delta/v$ and $(0; \ell'/v') + t' + \delta'/v'$ must not intersect, where:*

- *t and t' are the respective delays assigned to the lines;*
- *δ and δ' are the respective distances between the departure points of the lines and the crossing point;*
- *ℓ and ℓ' are the respective train lengths of the lines; and*
- *v and v' are the respective speeds of the lines.*

The delay of the schedule is the maximum delay it assigns. An integer schedule is one that only assigns integer delays.

An alternative way to understand schedules is to imagine each line's train starts "underground" with the front at the departure point and moves towards the arrival point at the line's speed, where it moves "underground" again ("underground" trains cannot collide with other trains); see Figure 2. A delay assignment is then a schedule if, and only if, the line segments that correspond to the "above-ground" parts of these trains never cross. For an example, see Figure 3.

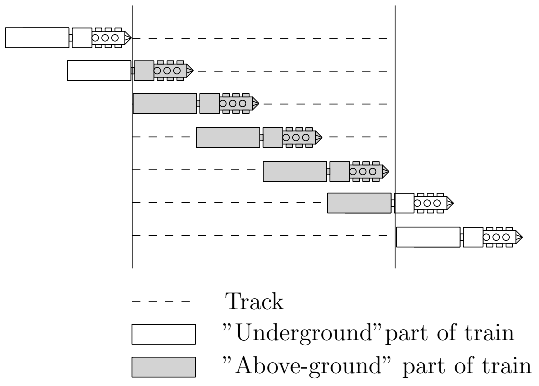


Figure 2: An illustration of the journey of a train.

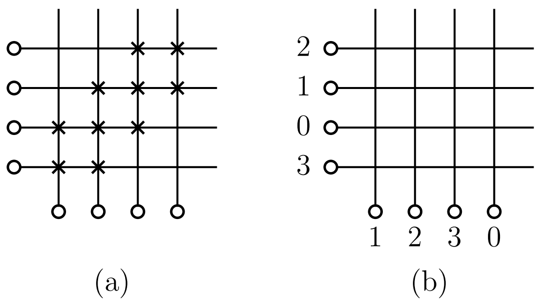


Figure 3: If unit speed trains of length 2 depart simultaneously from the circles in (a), there will be collisions at the marked crossings. However, if delays are introduced as in (b), no collisions will occur.

Definition 4 *A d-dimensional train network is regular if all its lines' speeds are 1, all its train lengths are the same and integer, all its tracks are axis-parallel and all its departure and arrival points are in $\mathbb{Z}^d$.*

Theorem 5 *All regular train networks admit an integer schedule of minimum delay (among all schedules, integer or otherwise).*

Proof. It is enough to show that, for an arbitrary schedule, assigning to each line a delay equal to the floor of the original delay will not result in any collisions. Indeed, take two lines whose tracks cross and define t, t', δ, δ', v, v', ℓ and ℓ' as in Definition 3. We thus have

$v = v' = 1$ and $\ell = \ell' \in \mathbb{N}\backslash\{0\}$. The original schedule ensures that

$$\big((0;\ell) + t + \delta\big) \cap \big((0;\ell) + t' + \delta'\big) = \{\},$$

which is equivalent to

$$(0;\ell) \cap \big((0;\ell) + t' - t + \delta' - \delta\big) = \{\}.$$

Because $t'-t$ and $\lfloor t'\rfloor - \lfloor t\rfloor$ differ by less than 1, $\lfloor t'\rfloor - \lfloor t\rfloor$ is either $\lfloor t'-t\rfloor$ or $\lceil t'-t\rceil$. Therefore, since ℓ and $\delta'-\delta$ are integers, the open interval $(0;\ell)+\lfloor t'\rfloor-\lfloor t\rfloor+\delta'-\delta$ cannot intersect $(0;\ell)$, which implies our delay assignment is collision-free. $\square$

Definition 6 *Consider a train line with an axis-parallel track that departs from $p \in \mathbb{R}^d$ and arrives at $p + \alpha e_i$, where e_i is the i-th vector of the canonical basis of $\mathbb{R}^d$ $(0 \leqslant i < d)$ and $\alpha \in \mathbb{R}\backslash\{0\}$. The axis of the line is the number i. Furthermore, we say this line is positive if $\alpha > 0$ and negative if $\alpha < 0$, with its sign being 1 or -1, respectively.*

Definition 7 *For any modulus $k \in \mathbb{N}\backslash\{0\}$, we extend the modulo k function to real numbers as follows: if $x \in \mathbb{R}$, then $x \bmod k = (\lfloor x\rfloor \bmod k) + x - \lfloor x\rfloor$. Furthermore, for a set $X \subseteq \mathbb{R}$, we denote $\{x \bmod k : x \in X\}$ by $X \bmod k$.*

3 Results

Theorem 8 *Any regular d-dimensional train network with trains of length ℓ and only positive lines has a schedule with delay at most $d\ell - 1$.*

Proof. Assign delay $\left(\ell a + \sum_{i=0}^{d-1} p_i\right) \bmod (d\ell)$ to lines departing from $p = (p_0, \ldots, p_{d-1}) \in \mathbb{Z}^d$ with axis $a \in \{0, \ldots, d-1\}$. Consider then a crossing between a line departing from $p = (p_0, \ldots, p_{d-1}) \in \mathbb{Z}^d$ with axis a and another line departing from $p' = (p'_0, \ldots, p'_{d-1}) \in \mathbb{Z}^d$ with axis $a' \neq a$. In the notation of Definition 3, we have:

- $t = \left(\ell a + \sum_{i=0}^{d-1} p_i\right) \bmod (d\ell)$;
- $t' = \left(\ell a' + \sum_{i=0}^{d-1} p'_i\right) \bmod (d\ell)$;
- $\delta = p'_a - p_a$; and
- $\delta' = p_{a'} - p'_{a'}$.

Thus, modulo $d\ell$,

$$\begin{aligned} & t' - t + \delta' - \delta \\ &\equiv \ell(a'-a) + \sum_{i=0}^{d-1}(p'_i - p_i) - (p'_{a'} - p_{a'}) - (p'_a - p_a) \\ &\equiv \ell(a'-a) + \sum_{\substack{i=0 \\ i \notin \{a,a'\}}}^{d-1} (p'_i - p_i) \\ &\equiv \ell(a'-a) \end{aligned}$$

since, for axes $i \notin \{a, a'\}$, we have $p_i = p'_i$ from the fact that the lines cross. However, there cannot be an intersection between $\big((0;\ell) + t' - t + \delta' - \delta\big) \bmod (d\ell)$ and $(0;\ell) \bmod (d\ell) = (0;\ell)$ because $a \neq a'$. Therefore, $(0;\ell) \cap \big((0;\ell) + t' - t + \delta' - \delta\big) = \{\}$ and there are no collisions. $\square$

Theorem 9 *Any regular 2-dimensional train network with trains of length ℓ has a schedule with delay at most $M - 1$, where*

$$M = \begin{cases} 2, & \ell = 1 \\ 8, & \ell = 2 \\ 6\ell, & \ell \geqslant 3. \end{cases}$$

Proof. Assign delay

$$\sigma\Big(x + y + (1-a)\big(-2(y \bmod \ell) - \ell + 1\big) + a\big(-2(x \bmod \ell) + 2\ell - 1\big)\Big) \bmod M$$

to lines departing from (x, y) with axis $a \in \{0, 1\}$ and sign $\sigma \in \{-1, 1\}$. Consider then a horizontal line departing from (x, y) with sign σ that crosses a vertical line departing from (x', y') with sign σ'. In the notation of Definition 3, we have:

- $t = \sigma\big(x + y - 2(y \bmod \ell) - \ell + 1\big)$;
- $t' = \sigma'\big(x' + y' - 2(x' \bmod \ell) + 2\ell - 1\big)$;
- $\delta = \sigma(x' - x)$; and
- $\delta' = -\sigma'(y' - y)$.

Note that $M \geqslant 2\ell$ and so it is enough to show that

$$(t' - t + \delta' - \delta) \bmod M \in \{\ell, \ldots, M - \ell\}.$$

Indeed, then $(0;\ell) \bmod M = (0;\ell)$ does not intersect $((0;\ell) + t' - t + \delta' - \delta) \bmod M$, which implies $(0;\ell) \cap ((0;\ell) + t' - t + \delta' - \delta) = \{\}$, i.e., a lack of collisions. We prove this by cases starting when $\sigma = \sigma' = 1$, which gives us, modulo M,

$$\begin{aligned} & t' - t + \delta' - \delta \\ &\equiv x' + y' - 2(x' \bmod \ell) + 2\ell - 1 - y' + y - x - y \\ &\quad + 2(y \bmod \ell) + \ell - 1 - x' + x \\ &\equiv 2(y \bmod \ell) - 2(x' \bmod \ell) + 3\ell - 2. \end{aligned}$$

Because $(y \bmod \ell), (x' \bmod \ell) \in \{0, \ldots, \ell - 1\}$, we have that

$$\begin{aligned} 2(y \bmod \ell) - 2(x' \bmod \ell) + 3\ell - 2 \in &\{\ell, \ldots, 5\ell - 4\} \\ \subseteq &\{\ell, \ldots, M - \ell\}. \end{aligned}$$

In case $\sigma = -1$ and $\sigma' = 1$, we have, modulo M,

$$\begin{aligned} t' - t + \delta' - \delta &\equiv x' + y' - 2(x' \bmod \ell) + 2\ell - 1 - y' + y \\ &\quad + x + y - 2(y \bmod \ell) - \ell + 1 + x' - x \\ &\equiv 2\big(x' - (x' \bmod \ell) + y - (y \bmod \ell)\big) + \ell. \end{aligned}$$

Because $x' - (x' \bmod \ell)$ and $y - (y \bmod \ell)$ are multiples of ℓ, we must have that $t' - t + \delta' - \delta$ is an odd multiple of ℓ. Moreover, since M is an even multiple of ℓ, $(t' - t + \delta' - \delta) \bmod M$ must indeed be in $\{\ell, \ldots, M - \ell\}$.

To complete the remaining two cases, note that if σ and σ' were simultaneously multiplied by -1, so would be $t' - t + \delta' - \delta$. However, $\{\ell, \ldots, M - \ell\}$ is closed under negation modulo M. □

Theorem 10 *Any regular 3-dimensional train network with trains of length* 1 *has a schedule with delay at most* 5.

Proof. We delay each line by the only integer in $\{0, \ldots, 5\}$ that is equivalent to $\sigma(p_0 + p_1 + p_2 + a)$ modulo 3 and equivalent to $p_0 + p_1 + p_2 + (\sigma + 1)/2$ modulo 2, where $p = (p_0, p_1, p_2) \in \mathbb{Z}^3$ is its departure point, $a \in \{0, 1, 2\}$ is its axis and $\sigma \in \{-1, 1\}$ is its sign (the existence and uniqueness of this integer is assured by the Chinese Remainder Theorem).

Consider then two crossing lines that:

- Depart respectively from $p = (p_0, p_1, p_2) \in \mathbb{Z}^3$ and $p' = (p'_0, p'_1, p'_2) \in \mathbb{Z}^3$;
- Have respective signs $\sigma, \sigma' \in \{-1, 1\}$; and
- Have $a \in \{0, 1, 2\}$ and $a + 1 \in \{0, 1, 2\}$ as their respective axes (axis arithmetic is done modulo 3).

Thus, in the notation of Definition 3, we have $\delta = \sigma(p'_a - p_a)$ and $\delta' = -\sigma'(p'_{a+1} - p_{a+1})$. Modulo 2, we also have $t \equiv p_0 + p_1 + p_2 + (\sigma + 1)/2$ and $t' \equiv p'_0 + p'_1 + p'_2 + (\sigma' + 1)/2$. Therefore, in case $\sigma \neq \sigma'$, also modulo 2,

$$\begin{aligned} t' - t + \delta' - \delta &\equiv \sum_{i=0}^{2} (p'_i - p_i) + (\sigma' + 1)/2 - (\sigma + 1)/2 \\ &\quad - \sigma'(p'_{a+1} - p_{a+1}) - \sigma(p'_a - p_a) \\ &\equiv \sum_{i=0}^{2} (p'_i - p_i) - (p'_{a+1} - p_{a+1}) \\ &\quad - (p'_a - p_a) + (\sigma' - \sigma)/2 \\ &\equiv p'_{a+2} - p_{a+2} + (\sigma' - \sigma)/2. \end{aligned}$$

However, since the lines cross, $p_{a+2} = p'_{a+2}$ and, since $\sigma, \sigma' \in \{-1, 1\}$ and $\sigma \neq \sigma'$, it must be that $(\sigma' - \sigma)/2 \equiv 1$ modulo 2, so $(t' - t + \delta' - \delta) \bmod 2 = 1$. The collision avoidance condition for these lines is $(0; 1) \cap \big((0; 1) + t' - t + \delta' - \delta\big) = \{\}$ and must be true since $(0; 1) \bmod 2 = (0; 1)$ but $\big((0; 1) + t' - t + \delta' - \delta\big) \bmod 2 = (1; 2)$.

On the other hand, if $\sigma = \sigma'$, then note that, modulo 3,

$$t \equiv \sigma(p_0 + p_1 + p_2 + a)$$

and

$$t' \equiv \sigma(p'_0 + p'_1 + p'_2 + a + 1).$$

Therefore, again modulo 3,

$$\begin{aligned} t' - t + \delta' - \delta &\equiv \sigma\left(\sum_{i=0}^{2} (p'_i - p_i) + a + 1 - a\right) \\ &\quad -\sigma(p'_a - p_a) - \sigma(p'_{a+1} - p_{a+1}) \\ &\equiv \sigma(p'_{a+2} - p_{a+2} + 1) \\ &\equiv \sigma \end{aligned}$$

as, once more, $p_{a+2} = p'_{a+2}$. As before, $(0; 1) \bmod 3 = (0; 1)$ but

$$\big((0; 1) + t' - t + \delta' - \delta\big) \bmod 3 = (0; 1) + (\sigma \bmod 3),$$

which is either $(1; 2)$ or $(2; 3)$. No collisions are therefore possible. □

4 Clique Searching and Lower Bounds

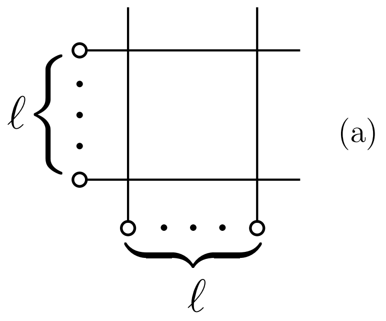
Whether a train network admits an integer schedule with delay at most $D \in \mathbb{N}$ can be decided with a clique search: create a graph G containing a vertex $v_{L,t}$ for each line L and time $t \in \{0, \ldots, D\}$; for every pair of distinct lines L and L' and for each pair of times $t, t' \in \{0, \ldots, D\}$ which would not result in a collision if assigned as delays respectively to L and L', put an edge between $v_{L,t}$ and $v_{L',t'}$. Note that if L and L' do not cross, then $v_{L,t}v_{L',t'}$ is an edge for all $t, t' \in \{0, \ldots, D\}$. Note also that an integer schedule with delay at most D exists if, and only if, G has a clique with as many vertices as there are lines. This is because two vertices associated with the same line cannot be selected and we encoded potential collisions in the edges between vertices associated with different lines. So, essentially, the clique is a delay assignment.

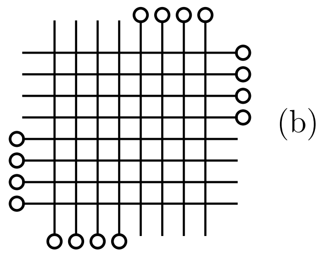
Due to Theorem 5, we can decide whether a regular train network has a schedule with delay at most a given number $D \in \mathbb{R}$. We have implemented this algorithm with the Cliquer clique solver [11] and recorded some lower bounds for regular train networks in Table 1.

For greater exposition we will walk through a simple example of determining the lower bounds for a train network with only positive lines and a train length of 2. Referring to Theorem 8 we see that a delay of $2\ell - 1 =$

d	ℓ	σ	Delay	Tight?	Figure
2	≤ 10	$+$	$2\ell - 1$	Theorem 8	4a
2	1	$\pm$	1	Theorem 9	4a
2	2	$\pm$	7	Theorem 9	4b
3	1	$+$	2	Theorem 8	4c

Table 1: A report on some useful regular train network classes including: the networks' dimension d, the length ℓ of their trains, whether their lines are all positive or of unrestricted sign, a lower bound on the delay of schedules for some networks in the class, our knowledge on whether this is an upper bound on the delay required to schedule all networks in the class and a reference to a figure describing a network that, when input to our algorithm, will produce the lower bound.





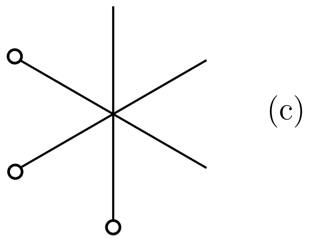


Figure 4: Regular train networks (∘ denotes the departure point of a line).

$4 - 1 = 3$ is possible. Therefore we will test a train network using the configuration in Figure 4a and a delay of 2. If we construct a graph as outlined above and cannot find a clique of size 4, then our configuration does not admit a network with a delay of 2, and by Theorem 8 and Theorem 5 our lower bound on the delay must then be 3. Although not necessary to our lower bound proof, for completeness we also show a graph built using a delay of 3 and illustrate a clique of size 4, which represents an assignment of delays that would result in no collisions.

We will express our train network in the following format:[1]

`<label> <t_len> <axis><dir> <x> <y> <z>`

where:

`<label>`	is the line's label;
`<t_len>`	is the line's train length, a positive integer;
`<axis>`	is the axis the track is parallel to ("x", "y", or "z");
`<dir>`	is the line's direction of movement ("-" or "+");
`<x> <y> <z>`	are the line's departure point.

Our train network in the above format based on the configuration of Figure 4a and illustrated in Figure 5, is then:

A 2 x+ 0 1 0
B 2 x+ 0 2 0
C 2 y+ 1 0 0
D 2 y+ 2 0 0

We will refer to the above train network as Network 1. We create a vertex for each combination of line and possible delay, then for each pair of vertices consisting of distinct lines, we connect them with an edge if the delay assignment does not result in a collision. See Figure 6. We may then search this graph for cliques of size 4, the number of train lines. To facilitate this search, we built our graph using a Python script, found in the full paper, and fed the output into the Cliquer clique solver [11]. Figure 7 shows the graph resulting from Network 1 with a maximum delay of 3, and the resulting clique which gives a delay assignment that will not result in any collisions.

5 Open Problems

The main problem that remains open is whether, for values of $\ell \geqslant 2$, every three-dimensional regular train network with trains of length ℓ admits a schedule with delay bounded by a constant. However, we are also interested in extending the lower bounds in Table 1 to more values of ℓ.

[1]This input format is a slightly altered version of the input we used to run our Python script, which built the graph in Cliquer readable format, and for which we refer the reader to the full paper.

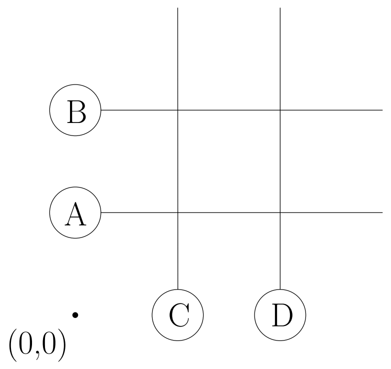


Figure 5: Network 1.

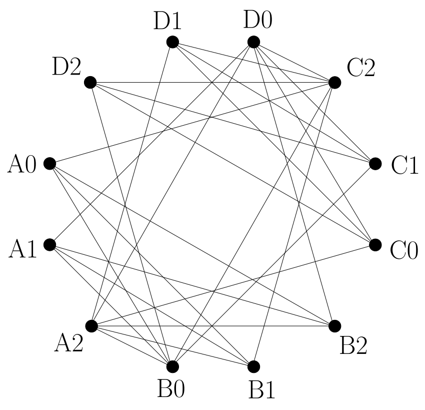


Figure 6: The resulting graph for Network 1 and a maximum delay of 2. The reader may verify there are no cliques of size 4.

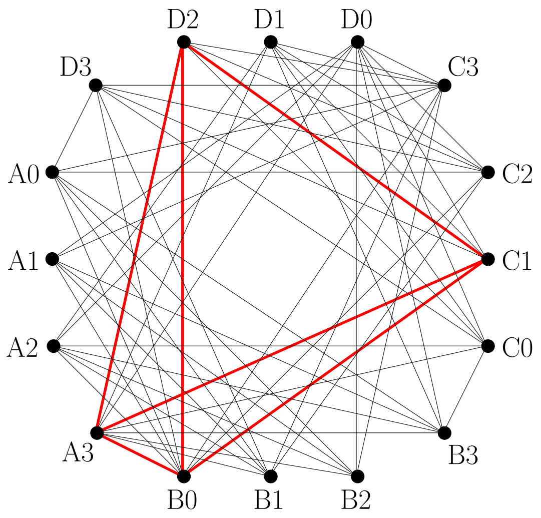


Figure 7: The resulting graph for Network 1 and a maximum delay of 3. Note the clique of size 4 which gives a delay assignment of A:3, B:0, C:1 and D:2.

Acknowledgements

We would like to thank Joseph S. B. Mitchell, Esther M. Arkin, Sándor Fekete and members of the Computational Geometry group at Carleton University.

References

[1] Ajay, Jammigumpula and Jana, Satyabrata and Roy, Sasanka Collision-free routing problem with restricted L-path. *Discrete Applied Mathematics*, 2021

[2] Dasler, Philip and Mount, David M. On the complexity of an unregulated traffic crossing. *Algorithms and Data Structures*, pages 224–235, 2015.

[3] Hyla, P. and Szpytko, J. Automated guided vehicles: the survey. *Journal of KONES*, 24, 2017.

[4] Ajaykumar, Jammigumpula and Das, Avinandan and Saikia, Navaneeta and Karmakar, Arindam Problems on one way road networks. *Proceedings of the 28th Canadian Conference on Computational Geometry*, pages 303–308, 2016.

[5] Kim, Chang W. and Tanchoco, J. M. A. Conflict-free shortest-time bidirectional AGV routeing. *The International Journal of Production Research*, 29(12):2377–2391, 1991.

[6] Koff, Gary A. Automatic guided vehicle systems: applications, controls and planning. *Material flow*, 4(1–2):3–16, 1987.

[7] Zeng, Laiguang and Wang, Hsu-Pin (Ben) and Jin, Song Conflict detection of automated guided vehicles: a Petri net approach. *The International Journal of Production Research*, 29(5):866–879, 1991.

[8] Qiu, Ling and Hsu, Wen-Jing and Huang, Shell-Ying and Wang, Han Scheduling and routing algorithms for AGVs: a survey. *The International Journal of Production Research*, 40(3):745–760, 2002.

[9] Malmborg, Charles J. A model for the design of zone control automated guided vehicle systems. *The International Journal of Production Research*, 28(10):1741–1758, 1990.

[10] Kakikura, Masayoshi and Takeno, Jun Ichi and Mukaidono, Masao A tour optimization problem in a road network with one-way paths. *IEEJ Transactions on Electronics, Information and Systems*, 98(8):257–264, 1978.

[11] Niskanen, Sampo and Östergård, Patric R. J. Cliquer user's guide, version 1.0. *Communications Laboratory, Helsinki University of Technology, Espoo, Finland*, Technical Report T48, 2003.

[12] Robbins, Herbert Ellis A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.

[13] Scheffer, Christian Train scheduling hardness and algorithms. *The 13th International Conference and Workshops on Algorithms and Computation (WALCOM 2020)*, 2020.

[14] Arora, Sudha and Raina, A. K. and Mittal, A. K. Collision avoidance among AGVs at junctions. *Intelligent Vehicles Symposium*, 2000.

[15] Vis, Iris F. A. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006.

[16] Yan, Xuejun and Zhang, Canrong and Qi, Mingyao Multi-AGVs collision-avoidance and deadlock-control for item-to-human automated warehouse. *Industrial Engineering, Management Science and Application (ICIMSA)*, pages 1–5, 2017.

Block Dude Puzzles are NP-Hard (and the Rugs Really Tie the Reductions Together)

Austin Barr* Calvin Chung† Aaron Williams‡

Abstract

The computational complexity of agent-based box-pushing puzzles on grids is well-studied. In particular, the video game *Sokoban* was shown to be NP-hard, and later PSPACE-complete, in the mid-1990s, and dozens of variants have since been studied. In this paper, we change the top-down perspective to a side perspective, where the player and the boxes are subject to gravity, and the player is able to climb on top of boxes or walls of height one. We prove that determining whether a level is solvable is NP-hard when the goal is to reach the exit, or place the boxes on target locations. The former result was previously shown to be true with "*Dig Dug* gravity" (i.e. boxes are subject to gravity, but the player is not) by Friedman. We also consider the decision problems with pushable boxes being replaced by liftable blocks, or with pushable and liftable blox, or with general crates. In total, we establish NP-hardness for eight different decision problems, all based around a single reduction. The inspiration for this article was the classic TI 83/84 calculator game *Block Dude*, which requires reaching an exit in the presence of liftable blocks.

1 Introduction

This section reviews video games that use box pushing, and the complexity of related decision problems.

1.1 Box-Pushing

Sokoban (倉庫番) was created by Hiroyuki Imabayashi, and released on cassette tape by Thinking Rabbit for a variety of Japanese personal computers in 1982 [40]. In total, the company has released nearly 100 official versions of the game [37], with the most recent entry being *Everyone's Sokoban* (みんなの倉庫番) for the Nintendo Switch and PlayStation 4 (see Figure 1).

Despite the game's long history, its basic rule set has never changed[1]. The game is played on a grid with a top-down perspective. Some of the cells are filled with walls or boxes, and the player controls a character that occupies a single cell. The player can move in the cardinal directions, and has the power to push — not pull — a single box into the next cell, so long as that cell is not occupied by a wall or another box. In addition, some of the blank cells are marked as targets, and they are equinumerous with the boxes. The goal is to rearrange the k boxes so that they occupy the k target cells.

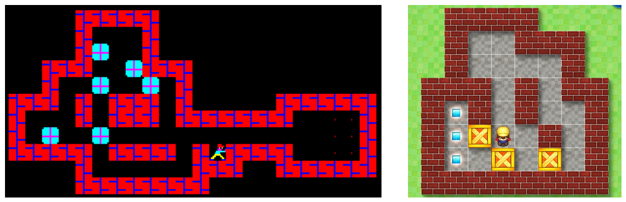


Figure 1: Thinking Rabbit's releases of Sokoban include *Sokoban* (1982) for the NEC PC-8801 (left), and *Everyone's Sokoban* (2019) for the Nintendo Switch (right).

Spectrum Holobyte published the game under the name *Soko-Ban* for American personal computers in 1988, which was the same year it brought *Tetris* to the same platform. Today, Thinking Rabbit and the Tetris Company still make modern versions of their respective games. However, the Tetris Company vigorously defends its intellectual property against other falling-block puzzles [36], whereas Thinking Rabbit does not. As a result, the term "Sokoban" has become genericized; it is synonymous with the genre of box-pushing puzzle games, and it can be found in the title of games that are not affiliated with Thinking Rabbit.

Sokoban is also a popular research topic. There are over 100 publications with "Sokoban" or "倉庫番" in the title [1], ranging from artificial intelligence solvers [20, 21] and optimizers [26], to level generation [23, 34] and evaluation [4], and human solutions [35]. Many results have also been presented in less formal venues, including websites discussing levels with the most moves [13], and undergraduate theses on levels that require an exponential number of moves to solve [28].

The Sokoban decision problem was shown to be NP-hard by Fryers and Greene in *Eureka* magazine [14], and independently by Dor and Zwick [10], and Uehara [38][2].

*Williams College, `abarr877@gmail.com`
†Williams College, `calvintchung@gmail.com`
‡Williams College, `aaron.williams@williams.edu`

[1]There is an exception: The first releases include fake wall tiles that the player must find and pass through to complete the level.

[2]The original Port Huron Statement [33, 41] does not contain a proof of NP-hardness, nor does its compromised second draft.

The exponential level constructions in [28] show that the most obvious certificate — the sequence of moves — is insufficient for establishing membership in NP. Thus, it was natural to consider more difficult complexity classes. Culberson proved that SOKOBAN[3] is PSPACE-complete by using it to simulate a bounded Turing Machine [5]. Using nondeterminstic constraint logic, Hearn and Demaine showed that PSPACE-hardness holds even for levels that contain no walls [18].

Countless variations of Sokoban exist as video games, and many of these variations have been studied through the lens of computational complexity. If the player is required to reach an exit, rather than place the boxes on target locations, then the decision problem is known as PUSH-1. If the player can push k boxes instead of a single box, then the decision problem is PUSH-K. In PUSHPUSH, the boxes slide until hitting the next box or wall. More generally, there are 16 different decision problems under the PUSH[PUSH]-1/K/*-[X] umbrella, all of which are known to be NP-hard or PSPACE-complete (see [8, 19]).

1.2 Side Perspective

In early video game history, it was common for ideas to be implemented in both top-down and side perspectives. For example, Ralph Baer's *Brown Box* (1967) and Atari's *Pong* (1972) use a top-down perspective for tennis, and were predated by William Higenbothem's *Tennis for Two* (1958), which uses a side-view on an oscilloscope. Similarly, the *trap'em-up* genre began with the top-view in *Heiankyo Alien* (1979) by Tokyo's Theoretical Science Group, before moving to a side-view in *Space Panic* (1980), and *Lode Runner* (1983).

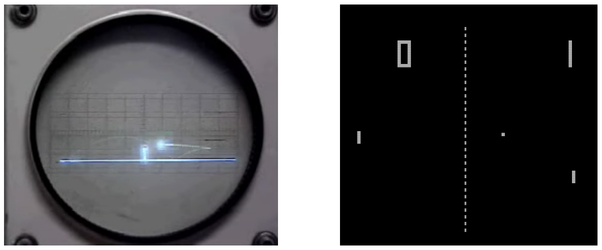

(a) *Tennis For Two* (1958)   (b) *Pong* (1972)

Figure 2: Tennis with top-down and side perspectives.

Despite *Sokoban*'s popularity, we are unaware of any video game that is based solely on box-pushing from a side perspective. Since games have inspired many of the academic investigations into motion planning[4], it is understandable that there are no computational complexity results for the corresponding decision problem.

[3]If *Title* is a video game, then TITLE refers to the following decision problem: Can a given level of *Title* be completed? The game is also suitably generalized (i.e. unbounded level size.)

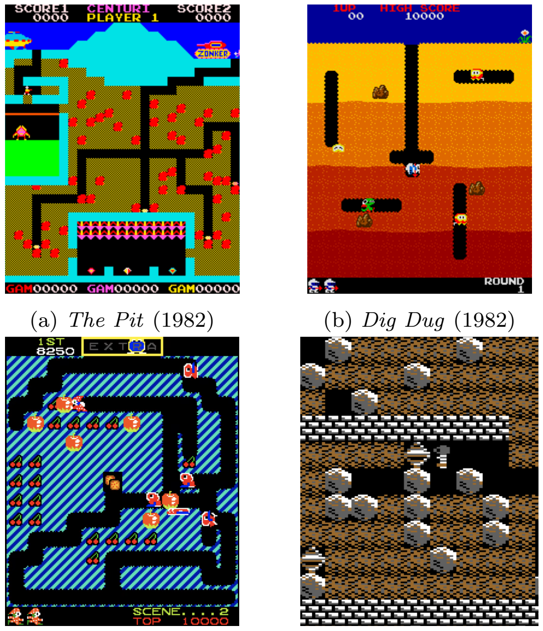
[4]For a recent example with turnstiles, see Greenblatt et al [15].

1.3 Dig Dug Gravity

Among commercial games, the closest matches to box pushing from a side perspective come from the *dig 'em up* genre (see Figure 3). In these games, the player can remove dirt, which may dislodge objects that then fall downward. In particular, the apples in *Mr. Do!* and the boulders in *Boulder Dash* can also be pushed horizontally. Unlike *Sokoban*, this genre also features action-oriented gameplay, with monsters and time limits that distract from a pure puzzle-solving experience.

The dig 'em up genre also features peculiar physics: The player and the enemies are not subject to gravity. In other words, it is as if some game elements operate according to a top-down perspective, while others adhere to a side-view. We refer to this physical model, which is as curious as it is common, as *Dig Dug gravity*, owing to the genre's most popular game.

These games also differ from box pushing games in their objectives. For example, the goal of *Dig Dug* is to eliminate a number of enemies. On the other hand, in *Boulder Dash*, the player must collect some number of diamonds, and then reach an exit.



(a) *The Pit* (1982)   (b) *Dig Dug* (1982)

(c) *Mr. Do!* (1982)   (d) *Boulder Dash* (1984)

Figure 3: Dig 'em up games use Dig Dug gravity (i.e. the player and enemies don't fall), and dirt that can be cleared by the player. The games include falling boulders (or apples), with (c) and (d) also supporting different types of pushing. They also include enemies and/or time limits, and objectives that differ from the SOKOBAN and PUSH problems.

Friedman introduced the PUSH-1-G decision problem, which adds Dig Dug gravity to PUSH-1. In other words, it asks if a player can reach an exit when the level consists of walls and pushable boxes, where the boxes are subject to gravity, but the player is not. He proved that the problem is NP-hard, as is its generalization PUSH-K-G for any $k \geq 1$ [12]. Friedman's gadgets can be implemented directly in *Boulder Dash*[5] and adding diamonds to the exit proves that the game is NP-hard.

The NP-hardness of BOULDERDASH was also established by Viglietta. More generally, Metatheorem 1 in [39] proves that avatar-based games with (a) walls, (b) *single-use paths*, and (c) *location traversal*, are NP-hard. Single-use paths become blocked when they are traversed, while (c) refers to forcing the avatar to visit certain locations (including *collecting items* from Forisek's [11]). Figure 4 shows a single-use path in *Boulder Dash*.

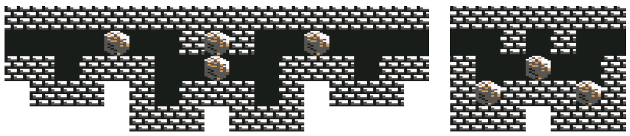

(a) Initial state. (b) Blocked state.

Figure 4: A single-use path in *Boulder Dash* [39]. Traversing (a) in either direction causes the middle of the path to become blocked, as seen in (b). Single-use paths *and* location traversal (collecting items) establish NP-hardness [39].

Metatheorem 1 does not immediately apply to *Dig Dug* or *Mr. Do!* since they do not have walls. We can apply it to *The Pit*, but we need to careful with one detail. In the arcade game, the player must collect at least one of the gems; we allow the gem requirement to be arbitrarily large in the decision problem THEPIT.

Corollary 1 (Metathm 1 [39]) THEPIT *is NP-hard.*

Proof. Boulders cannot be pushed in *The Pit*, so a single-use path can be constructed with a single boulder.

Initial state. Blocked state.

The Pit also has walls, and location traversal can be implemented in THEPIT with gems. □

We also mention an investigation of pulling block complexity by Ani et al. [3]. They prove that a wide variety of decision problems are PSPACE-complete. One exception is PULL?-1FG, which is shown to be NP-hard. The problem asks if an exit can be reached using Dig Dug physics and pullable objects. More specifically, its name can be parsed as follows: PULL? indicates that the player is not forced to pull blocks; 1 specifies that the player can only pull one block at a time; F denotes that walls are allowed; G states that Dig Dug gravity is used rather than a top-down perspective.

[5]The gadgets avoid *Boulder Dash*'s non-trivial falling physics: A boulder will suddenly fall to the side if it is on top of another boulder, and the cells to the side of both boulders are empty.

(a) TI-83+. (b) The *Puzz Pack* includes *Block Dude*.

Figure 5: *Block Dude* was developed for Texas Instrument calculators as part of the *Puzzle Pack* suite of games.

1.4 Outline

We consider the computational complexity of box pushing from a side perspective and with *normal gravity* (i.e. the player and movable objects are subject to it) and the standard goals from both SOKOBAN and PUSH.

We also consider block lifting. Lifting is the only mechanism in *Block Dude* (1999), which is perhaps the closest (non-commercial) video game analogue to *Sokoban* from a side perspective; it was also the inspiration for this article. (A type of top-down lifting was considered in the BOX MOVER PROBLEM [27].)

Section 2 defines our decision problems. In Section 5, we prove that the problems are NP-hard. This is preceded by Sections 3–4, which define basic gadgets and give a simplified 'rug' reduction. Final remarks are in Section 6. Owing to the title of the game that inspired this article, the reader should expect to find *Big Lebowski* quotes and references along the way.

2 Block Dude and Related Decision Problems

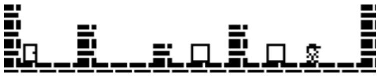
In this section, we describe the history and gameplay of Block Dude. Then we formulate a family of eight decision problems, one of which models the original game.

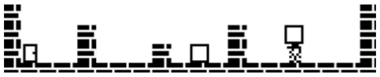
2.1 History of Block Dude

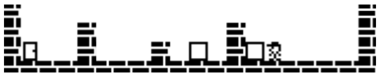
Block Dude is a TI-83/84 calculator game created by Brandon Sterner in 1999[6]. Detached Solutions included it in *PuzzPack* [9] (see Figure 5), which has been downloaded over 100,000 times from `ticalc.org` [31]. The game's popularity has never waned in the calculator community, winning the (fictitious) *Calculator Gaming Awards* 2002–2018 [22]. The game features 11 levels, and the speed-running world record is under 8 minutes [25]. A solution to the first level is in Figure 6.

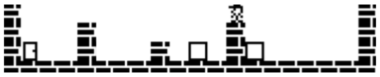
Like *Sokoban*, the basic *Block Dude* game has been ported to a variety of systems by other developers, with

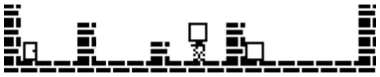
[6]Man, we've got some information, all right. Certain things have come to light. *Block Dude* wasn't the first Block Dude game! *Block-Man 1* was released commercially in 1994 (see Table 1).

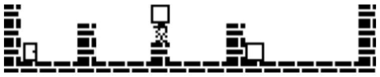

(a) The initial state of Level 1.

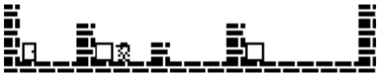

(b) Pick up a block.


(c) Drop the block to create a staircase.


(d) Climb the staircase.


(e) Fall down and pick up a block.


(f) Carry the block up the wall of height one.


(g) Drop the block to create another staircase.

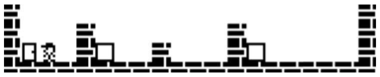

(h) Climb the second staircase and reach the door.

Figure 6: Solving Level 1 in *Block Dude*.

a sampling shown in Table 1. Readers who wish to try out the game are directed towards the browser-based port by Andrew Zich [42]. *Block Dude* was also used as a programming assignment in Harvard's CS50 [17]. Sterner discussed *Block Dude* in a Reddit AMA [32].

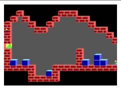
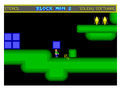
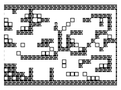
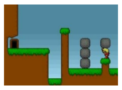
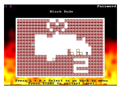
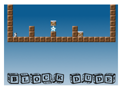
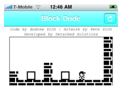
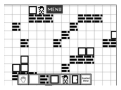
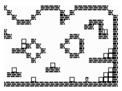
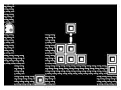
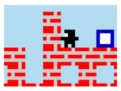
Year	Title	Developer	Platform(s)	Screenshot
1994	Block-Man 1	Soleau Software	DOS	
1995	Block-Man 2	Soleau Software	DOS	
1999	Block Dude	Brandon Sterner	TI-83 / TI-84 Calculators	
2004	BlockDude	Klas Kroon Chris Kotiesen	Adobe Flash	
2006 2010	Blockdude 1.3 Blockdude 1.4	Willems Davy	GP2X Dingoo A320	
2008	Block Dude	SG57	Sony PSP	
2008	Block Dude	Emmanuel Vincent	GNU/Linux	
2009	Block Dude	Andrew Zich Pete Zich	Apple iPhone JavaScript	
2010	Block Dude Evolved	Billy Connolly	Apple iPhone iPod Touch	
2011	Block Dude X	Amelia "Hinchy" Hinchliffe	Mac OS Windows	
2017	BlockDude	Brick Buddy Hazardous Dude	Android Steam	
2018	Block Dude	Mitch Kendall	Nintendo NES	
2019	Box Dude	Parker Phair	iOS	
2019	BlockDude	Dmitry Krapivin	ZX Spectrum 48K	
2021	BlockBot	Mark Horan	HTML5	

Table 1: Selected games related to *Block Dude*.

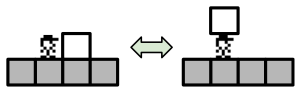
2.2 Block Dude Gameplay

Block Dude's objective is to move the avatar (i.e. the Dude) through the grid to the exit door. The Dude occupies one cell, as do the obstacles, which are immovable *bricks*, and movable *blocks*. The Dude moves left or right, and can climb onto obstacles one cell above his feet. They can also turn around in-place when there are obstacles on their left and right. The game has normal gravity, and the Dude safely falls from any height.

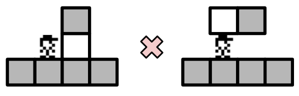
The Dude can only pick up a block that is directly in front of him, and can only do this when two cells are empty: the one above his head, and the one diagonally in front and above his head. If the cell in front of his feet is empty, then the Dude can drop a block into that position. Otherwise, if the cell in front and above the Dude is empty, then he can drop the block there. Any block that is dropped is subject to gravity, and it will continue falling until landing on a brick or block.

The Dude can hold and carry one block at a time, and he does so on top of his head. Thus, a height of at least two is required for the Dude to carry a block through a passageway. If the Dude attempts to carry a block past a brick that is directly above his head, then the brick will push the block off his head; the Dude will move forward while the block will fall behind him.

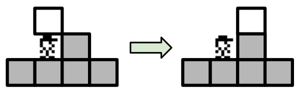
Figure 7 illustrates several of the finer points mentioned above, where grey squares are bricks.



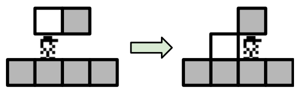
(a) The Dude does not move when picking up or dropping a block.



(b) A block cannot pass through a brick when it is picked up or dropped.



(c) A block can be dropped onto a brick (or block), but not picked up from it.



(d) A carried block will fall off of the Dude's head when passing under a brick.

Figure 7: *Block Dude* physics. Arrow directions indicate valid moves between states. The Dude can also turn around in tight spaces.

2.3 BlockDude Family

I'm the Dude. So that's what you call me. You know, that or, His Dudeness, or Duder, or El Duderino if you're not into the whole brevity thing

— Jeffrey Lebowski, *The Big Lebowski*

Rather than studying Block Dude in isolation, we consider a family of decision problems. The problems use the same perspective and movements as Block Dude. Thus, the Dude can walk left or right, climb on top of a single occupied cell that is immediately in front of them, and fall down from any height.

A level is completed in one of two ways:

(A) The level is completed when the Dude reaches the exit goal (denoted with ⚑). This is the *exit goal*.

(B) The level is completed when the Dude moves the k moveable objects onto the k *target* cells denoted by *. This is the *targets goal*, and it models the completion condition of Sokoban.

There are four types of moveable objects.

(1) A *block* can be picked up, but not pushed. These are identical to the blocks in Block Dude.

(2) A *box* can be pushed, but not picked up. These are similar to Sokoban boxes, but they fall like blocks.

(3) A *blox* can be picked up and pushed. Thus, a blox behaves like a block and a box.

(4) A *crate* is a block, box, or blox. In other words, a crate is the generic term for a moveable object.

We form our decision problems by combining the two different goals with four types of moveable objects. We name each problem using the type of moveable object followed by Dude/Duderino for exit/targets goals. For example, the original *Block Dude* game is type 1A, meaning that it has an exit goal, and all of the movable objects are blocks, and its full name is BlockDude. The decision problem names are summarized in Table 2.

To avoid repetition, we will reuse our arguments and figures as much as possible. For example, our levels will contain both an exit and targets; the former is ignored in instances of type B, and the latter are ignored in instances of type A. For convenience, we also use different colors for the various crates and locations. These colors are purely cosmetic, and are used as hints to more easily understand how to complete the targets goal.

2.4 Membership in PSPACE

We complete this section by noting that all of our decision problems can be solved in polynomial-space.

Theorem 2 *The problems in Table 2 are in PSPACE.*

Proof. Following the standard technique, we prove that the decision problems are in NPSPACE. Consider a CrateDude or CrateDuderino level on an r-by-c level. The current state of a level can be encoded as an r-by-c grid, where each entry is one of five possibilities: blank, box, block, blox, or Dude. (The remainder of the level — bricks, exit, target — is static.) Therefore, the state of the level can be encoded in $3 \cdot r \cdot c$ bits, and so the level can have at most $2^{3 \cdot r \cdot c}$ states. Therefore, if a level is solvable, then we can find a solution by non-deterministically making at most $2^{3 \cdot r \cdot c}$ moves. Storing a move counter that ranges from 0 to $2^{3 \cdot r \cdot c} - 1$ requires $\log_2(2^{3 \cdot r \cdot c}) = 3 \cdot r \cdot c$ bits, which is polynomial in the size of the input. Therefore, the CrateDude and CrateDuderino are in NPSPACE, and hence PSPACE by Savitch's Theorem [29] . This implies that the other six problems are also in PSPACE. □

3 Basic Gadgets and Approach

In this section, we discuss the general approach used by our reductions, and present our variable and clause gadgets. Each literal instance (i.e. each copy of x_i or $\overline{x_i}$) is represented by one crate in Section 4, and by a sequence of surplus crates in Section 5. We refer to these crates as *literal instance crates*, or simply *literal crates*.

Our gadgets include elevated bricks that are one cell below a ceiling brick. These *guards* restrict crates from being pushed or carried out of the gadget.

3.1 Variable Gadgets

Our gadget for variable x_i appears in Figure 8a, and it is drawn to match $i = 1$ from (1). From the gadget's start position, the Dude can drop down to the left or right. Left corresponds to setting x_i to false, and the Dude can reach every crate associated with a negative literal $\overline{x_i}$. Likewise, right corresponds to setting x_i to true, and the Dude can reach every crate associated with a

	Boxes (Push-Only)	Blocks (Lift-Only)	Bloxes (Push and Lift)	Crates (Push and/or Lift)
Exit Goal	BoxDude	BlockDude	BloxDude	CrateDude
Targets Goal	BoxDuderino	BlockDuderino	BloxDuderino	CrateDuderino

Table 2: The Block Dude family of decision problems. See Section 6.1 for a note on Dude vs Push notation.

positive literal x_i. In the full reduction, the Dude can "activate" each reachable crate by sending it downward into a clause gadget. The Dude cannot build a staircase back up to the start position of a variable gadget, so they must eventually return to the center and drop down to the start position of the x_{i+1} gadget.

Observation 1 *In the x_i variable gadget, the Dude can reach the positive literal crates x_i, or the negative literal crates $\overline{x_i}$, but not both, before exiting. This corresponds to setting $x_i = true$, or $x_i = false$, respectively.*

Actually sending crates downward will be a challenge. Section 4 uses a "cheat" to simplify the task, and Section 5 provides the actual implementation.

3.2 Clause Gadgets

Our clause gadget appears in Figure 8b. The basic idea is that the gadget is traversable if at least one of its literal crates has been sent downward into it. The flag illustrates where the next clause gadget begins.

Observation 2 *In the clause gadget for $(\ell_1 \vee \ell_2 \vee \ell_3)$, the Dude can reach the exit, if and only if, at least one of the literal crates for ℓ_1, ℓ_2, ℓ_3 has been sent downward into the gadget.*

Nihilism versus Dudeism

Is a Boolean value always true or false? Careful consideration of our reduction will reveal that the Dude is not obliged to activate a reachable crate within a variable gadget. This corresponds to believing nothing about the crate's literal instance: It is neither true nor false. Without loss of generality, we can assume that the Dude avoids this type of nihilism during gameplay, but in practice, the Dude is most certainly a lazy man, and liable to go with the flow.

4 NP-Hardness with Rugs

That rug really tied the room together.

— Walter in *The Big Lebowski*

In this section, we provide a single reduction that shows that all eight of our problems are NP-hard. However, there is a catch. We assume that the puzzles can use additional an gameplay element: a *rug*. Rugs are defined in Section 4.1, but in brief, they are similar to a trapdoor in that they allow crates to pass through without changing where the Dude can move.

Our reduction is from Monotone 3SAT, which is a restriction of 3SAT to monotone clauses. A *monotone clause* is a *positive clause* with three positive literals, or a *negative clause* with three negative literals. In other words, every clause has the form $(x_i \vee x_j \vee x_k)$ or $(\overline{x_i} \vee \overline{x_j} \vee \overline{x_k})$. In particular, we illustrate the reduction using the following instance with four clauses,

$$\phi = (\overline{x_3} \vee \overline{x_2} \vee \overline{x_1}) \wedge (\overline{x_5} \vee \overline{x_3} \vee \overline{x_1}) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4), \quad (1)$$

whose colors and indices are matched in Figure 10. The source problem is NP-complete by Schaefer's dichotomy theorem [30], and the monotone property will allow us to organize all of the clause gadgets along a single row.

4.1 Rugs

A rug can be placed in any otherwise empty cell of the grid. Rugs are not strong enough to hold up a crate. Therefore, any crate that is pushed on top of a rug, or dropped onto a rug, will fall through it. Similarly, if a crate falls onto a rug from above, then it will continue falling through the rug. However, the Dude never falls through a rug, even when he is carrying a crate. In other words, the Dude is never let down by a rug[7].

4.2 Literal Gadgets

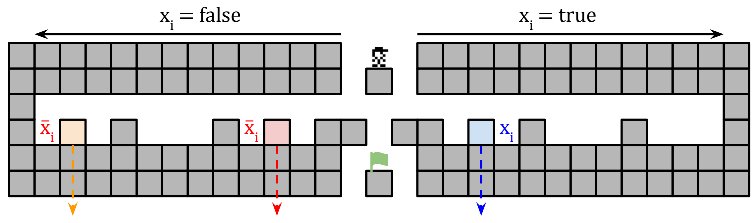
In this reduction, we create a single crate for every instance of a literal. Thus, if the Monontone 3SAT instance has k clauses, then the corresponding level will have $3k$ crates. As mentioned in Section 3.1, the Dude needs to be able to send these crates down to the respective clause gadgets. Rugs allow us to implement these *literal gadgets* very easily, as shown in Figure 9.

Observation 3 *In the literal gadget, the Dude can send the literal instance, x_i or $\overline{x_i}$, down to its clause gadget.*

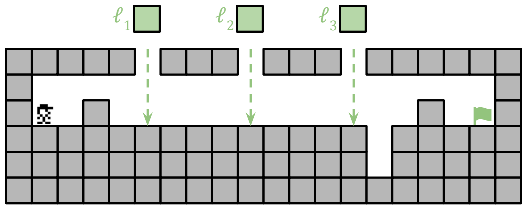
4.3 Rug Reduction

We can now present a single reduction from Monotone 3SAT. The reduction is illustrated in Figure 10. In

[7]This is true even if the rug has been micturated upon.

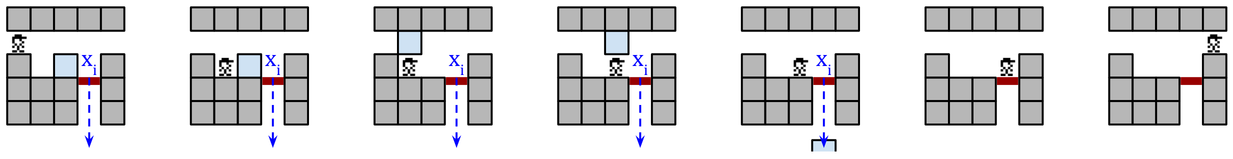


(a) Variable gadget for x_i. The Dude can reach the negative x_i literals by walking left, or the positive x_i literates by walking right, but not both. Then they can return to the center and fall down to the flag.



(b) Clause gadget ($\ell_1 \cup \ell_2 \cup \ell_3$). If at least one crate has fallen to the floor, then the Dude can push or drop it into the pit, and then walk to the flag.

Figure 8: A simplified view of our basic gadgets. In both cases, it is not possible to move a crate out of the gadget.



(a) (b) (c) (d) (e) (f) (g)

Figure 9: Activating a positive literal crate by sending the crate downward through a rug. If the crate is a block, then all steps (a)–(g) are used. Otherwise, if it is a box or blox, then it is pushed from (b) to (e), and steps (c) and (d) are omitted.

this image, and others, we allow some of the gadgets to overlap, in order to save space. In particular, literal instance gadgets in the same clause share the guards.

Theorem 3 *In the rug reduction for* Monotone 3SAT *instance ϕ, the Dude can reach the flag, or place the crates on the targets, if and only if, ϕ has a satisfying assignment. This is true regardless of the type of crate that is used for each literal instance.*

Proof. Suppose that ϕ is satisfiable and consider a specific satisfying assignment. By following this satisfying assignment when navigating the variable gadgets, the Dude is ensured that each clause gadget will have at least one crate in it. Therefore, they can traverse the clause gadgets and reach the exit. To accomplish the targets goal, they continue up the *target goal staircase* on the right, and then navigate the variable gadgets using the complement of the satisfying assignment.

For the other direction, suppose that ϕ is not satisfiable. When the Dude reaches the clause gadgets, there will be at least one that does not have a crate in it. Hence, the Dude cannot traverse all of the clause gadgets, and so they cannot accomplish either goal. □

5 NP-Hardness without Rugs

Now we establish the NP-hardness of the decision problems in their original form, without rugs. To do this, we need to simulate the rugs. We do this with two new gadgets, which are introduced in Section 5.1. Then we show how to implement these two gadgets with boxes in Section 5.2, and with blocks in Section 5.3. The former result also establishes hardness for crates, while the latter construction also works for blox.

5.1 Drop-Down and Fall-Through Gadgets

Rugs are used in two different ways in the rug reduction. The rugs that appear next to a crate allow for the initial dropping down of a crate, while the remaining rugs allow for crates to continue passing through them. We mirror this with the following two types of gadgets.

1. *Drop-down gadget.* In this gadget, the Dude is able to send one crate downward. The Dude can traverse the gadget left-to-right then right-to-left (or vice versa) and cannot go downward without getting stuck.
2. *Fall-through gadget.* In this gadget, the Dude can send one crate downward, so long as one crate has previously been sent downward into it. The Dude can traverse the gadget left-to-right then right-to-left (or vice versa) and cannot go downward without getting stuck.

As in the rug reduction, these gadgets allow crates, which represent literal instances, to be sent downward. However, in the rug reduction, an individual crate is sent straight downward from the variable gadget to the clause gadget. With these gadgets, the crate that is sent down changes after each successive gadget. In other words, these gadgets work together to send down a surplus of one crate, rather than an individual crate. Furthermore, the surplus crate that is sent down shifts two columns horizontally after each fall-through gadget.

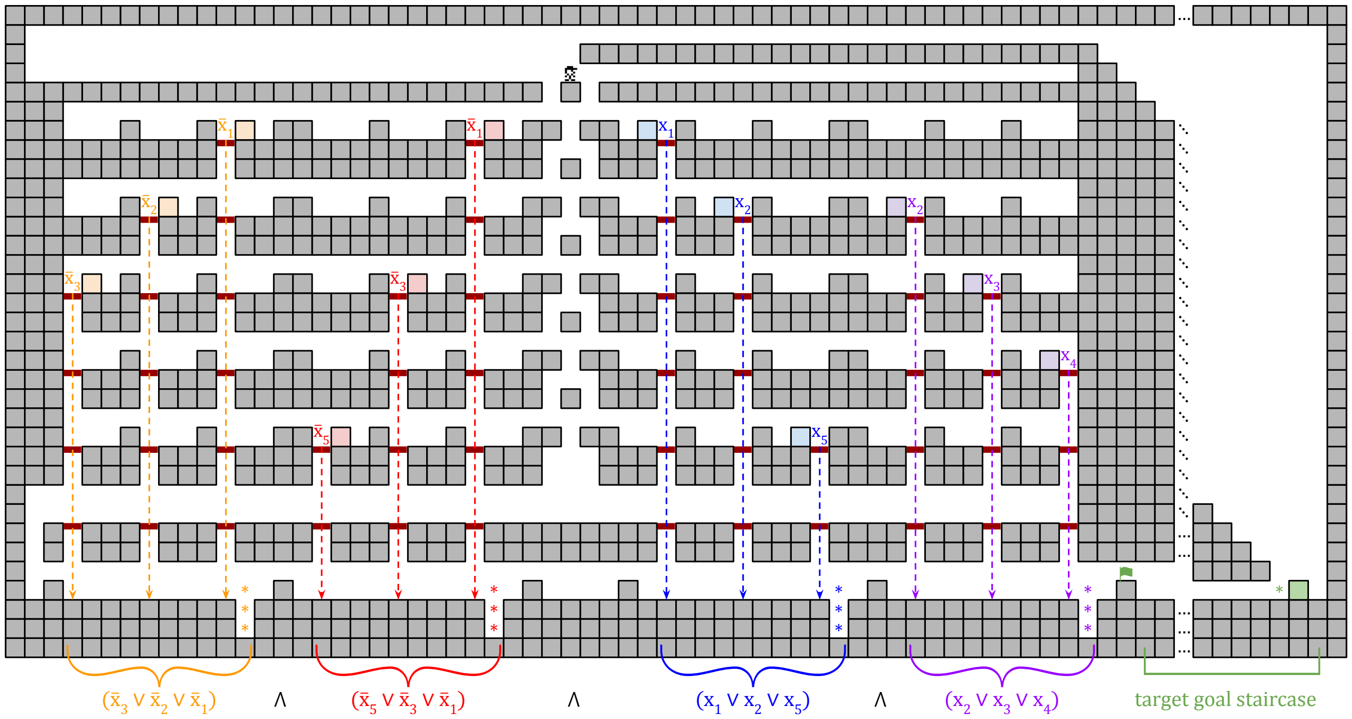


Figure 10: The rug reduction for $\phi = (\overline{x_3} \vee \overline{x_2} \vee \overline{x_1}) \wedge (\overline{x_5} \vee \overline{x_3} \vee \overline{x_1}) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4)$ in (1). To complete the exit goal, the Dude drops into the five variable gadgets on the left, right, left, left, and left, respectively. The resulting assignment $x_1 = x_3 = x_4 = x_5 =$ FALSE and $x_2 =$ TRUE makes the clause gadgets traversable[8]. To complete the targets goal, the Dude continues past the exit and walks up the target goal staircase. Then they drop into the variable gadgets on the opposite sides — right, left, right, right, right — and fill the clause gadget targets.

To implement these gadgets we use empty vertical stacks of cells called *pits*, which must be partially filled in order for the Dude to traverse them. More specifically, a pit of depth d must be filled with $d-1$ crates before it can be traversed. We say that a gadget has a *surplus*, if a crate has been dropped into its pit before the Dude has entered the gadget. When the Dude enters a gadget, he will be able to send a crate downward, if and only if, the gadget has a surplus. In other words, the Dude can move a surplus down to the next gadget. The depth of each pit is chosen to prevent the Dude from trying to take shortcuts in the level. More specifically, if the Dude drops down to the gadget below, then the depth of the pit will ensure that he gets stuck.

As a final note, we mention that these gadgets require more space than the corresponding gadgets in the rug reduction. As a result, the reader should observe that the layout in Figure 10 can be widened and lengthened without affecting its functionality.

5.2 Box Gadgets

Figure 11 contains our implementation of the drop-down gadget and fall-through gadget using boxes.

[8]A lazy Dude can drop left then right, and then go directly to the clause gadgets since $x_1 =$ FALSE and $x_2 =$ TRUE satisfies (1).

Lemma 4 *Figure 11 implements a drop-down gadget and fall-through gadget using boxes.*

Proof. For ease of reading, the explanation of how the gadget works is given in the caption of Figure 11. □

The next corollary follows from Lemma 4 and Theorem 3, and the fact that instances of BOXDUDE and BOXDUDERINO are also instances of CRATEDUDE and CRATEDUERINO, respectively.

Corollary 5 *BOXDUDE, BOXDUDERINO, CRATEDUDE, and CRATEDUDERINO are NP-hard.*

5.3 Block and Blox Gadgets

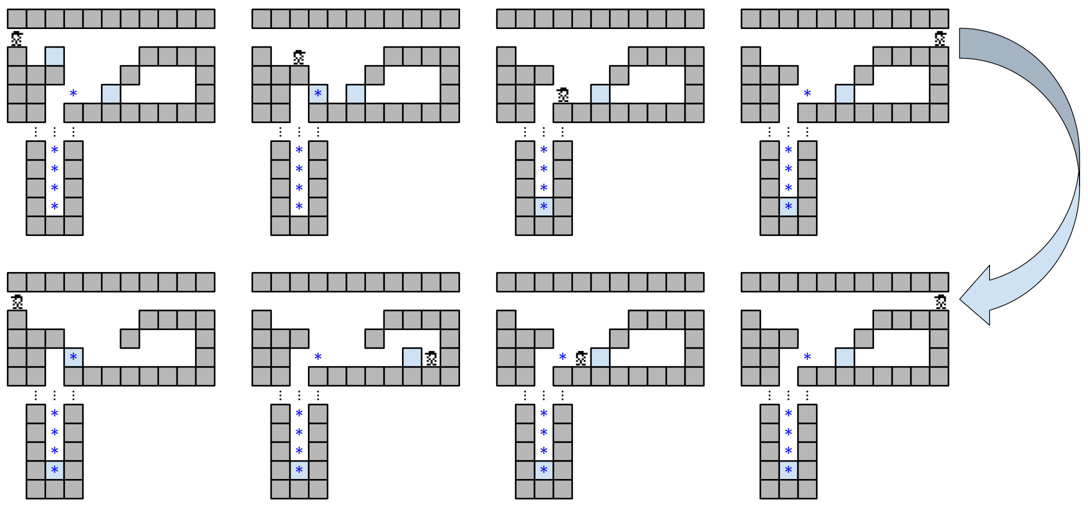
Figure 12 contains our implementation of the drop-down gadget and fall-through gadget using blocks. Inspection shows that it also suffices for blox.

Lemma 6 *Figure 12 implements a drop-down gadget and fall-through gadget using blocks or blox.*

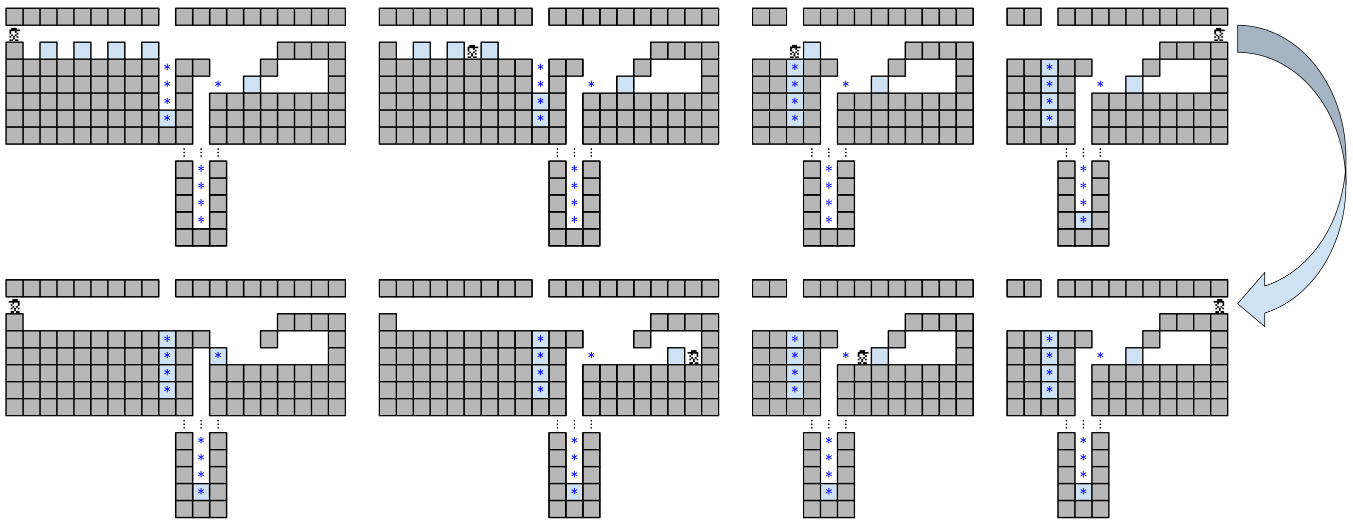
Proof. For ease of reading, the explanation of how the gadget works is given in the caption of Figure 12. □

Theorem 3 and Lemma 6 provide the final corollary.

Corollary 7 *BLOCKDUDE, BLOCKDUDERINO, BLOXDUDE and BLOXDUDERINO are NP-hard.*

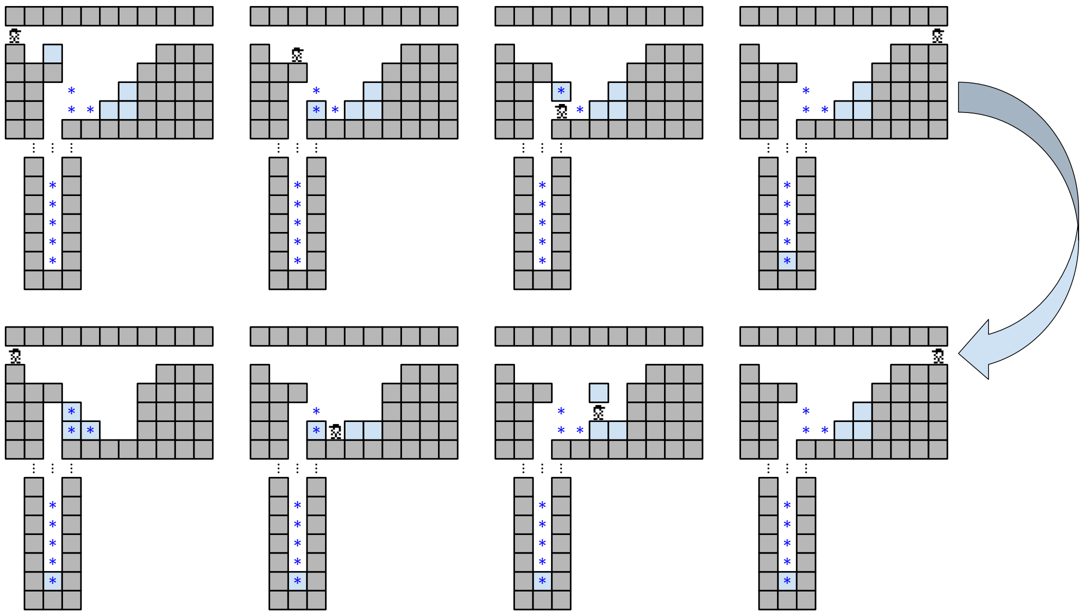


(a) Drop-down gadget with boxes. The Dude enters the gadget on the left (top-left) and pushes the top box once to the right and once to the left, so that it falls downward into the pit of the gadget below. The Dude then exits on the right (top-right). Re-entering (bottom-right), they use the hollow area to push the remaining box into the target positions (*), and then they can exit on the left (bottom-left).



(b) Fall-through gadget with boxes. The Dude enters the gadget from the left, and to proceed to the right, they must push boxes into the pit, starting with the rightmost of the four boxes. If a surplus box is in the pit when they enter the gadget (as illustrated), then there will be an additional box that can be pushed down as a surplus box into the pit of the gadget below. The Dude then exits the gadget (top-right). Reentering (bottom-right), the Dude proceeds to the exit on the left in the same way as in the drop-down gadget.

Figure 11: Box gadgets. The drop-down (top) and fall-through (bottom) gadgets for BoxDude and BoxDuderino. In both cases, the images illustrate the gadget for a positive literal instance, and all crates are boxes. The initial states are shown in the top-left corner, and the remaining images illustrate how the Dude can traverse the gadget left-to-right and send a surplus crate downward, then traverse it from right-to-left. The Dude cannot take a shortcut in either gadget since the pits have depth four, and at most two boxes can be sent downward from the gadget above. Note: To conserve space, the leftmost 7 columns are omitted from the rightmost four images in the fall-through.


(a) Drop-down gadget with blocks. The Dude enters on the left (top-left) and picks-up and drops the top block twice, so that it falls down into the pit of the gadget below. Then they exit on the right (top-right). Re-entering (bottom-right), they pick-up and drop the three blocks to form a staircase on the target positions (*), and exit on the left (bottom-left).

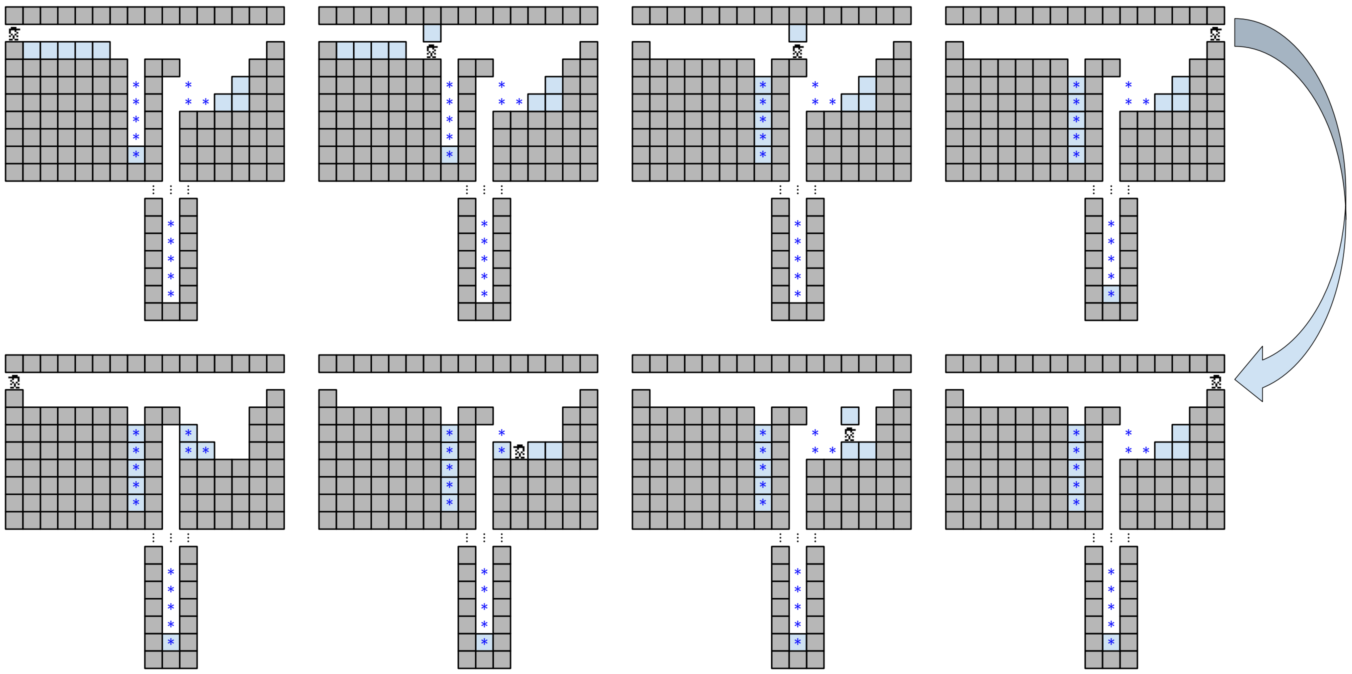

(b) Fall-through gadget with blocks. The Dude enters on the left, and to proceed they must pick up and drop blocks into the pit. If a surplus block is in the pit when they enter (as illustrated), then there will be an additional block that can be picked-up and dropped as a surplus block into the gadget below. The Dude then exits the gadget (top-right). Re-entering (bottom-right), they proceed to the exit on the left in the same way as in the drop-down gadget.

Figure 12: Block gadgets. The drop-down (top) and fall-through (bottom) gadgets for BlockDude and BlockDuderino. In both cases, the images illustrate the gadget for a positive literal instance, and all crates are blocks. The initial states are shown in the top-left corner, and the remaining images illustrate how the Dude can traverse the gadget left-to-right and send a surplus crate downward, then traverse right-to-left. The Dude cannot take a shortcut in either gadget since the pits have depth six, and at most four blocks can be sent down from the gadget above.

6 Final Remarks

We have shown that 8 decision problems are NP-hard, including one that models *Block Dude*. It remains open which are in NP (and hence NP-complete), and which are PSPACE-hard (and hence PSPACE-complete).

- CrateDude and CrateDuderino are the best candidates for being PSPACE-complete. The recent motion planning framework in Lynch's PhD thesis [24] (see also [7, 6]) could provide a pathway to this result, although normal gravity seems to make that route more challenging. A more modest goal is to establish the existence of levels that require exponentially many moves to solve. This would show that the most natural certificate (i.e. the sequence of moves) is insufficient for establishing NP membership (e.g. see [16]).
- BoxDude and BoxDuderino seem to be the best candidates for NP-completeness. This is because boxes cannot move up, so their y-position is a non-renewable resource, which is a hallmark of NP.

6.1 Nomenclature: Dude vs Push

A significant drawback to our exposition is that the Dude-based decision problem names do not fit into the standard Push-based nomenclature. One reason for this departure was to allow multiple types of movable objects (e.g. CrateDude includes boxes, blocks, and blox) at once; this is not easily supported by Push notation. However, we are pacifists, not conscientious objectors, so community members should feel free to introduce terminology like Lift-1NG for BlockDude, where 1NG denotes lifting one block at a time with normal gravity.

6.2 Open Problems

Besides the obvious open problems listed earlier, future research can consider other types of movable objects in the presence of normal gravity from a side perspective.

- Objects that slide horizontally when pushed. This would be a PushPush-style problem, but with a different physical model. We mention that ice blocks are used in the side-view game in the NES game *Fire 'n Ice* (1993) with normal gravity.
- Objects that can be pulled. See Section 1.3 for a discussion of Pull?-1FG with Dig Dug gravity.
- Objects that can be moved as if they were *pocketed*. When the player picks up a box in *Loader Larry* (1995), they carry it in the same cell as their avatar, as if it were placed in their pocket (see Figure 13).

We also note that the Dude never needs to fall more than three cells at a time in our reduction. Limiting the Dude's ability to fall only from a height of one may be an interesting problem to investigate, although that would put the Dude in the running for "the weakest video game character" worldwide, which would surely rattle those *Spelunker* (1983) fans and their used game bins [2] — ah, look at me. I'm ramblin' again.

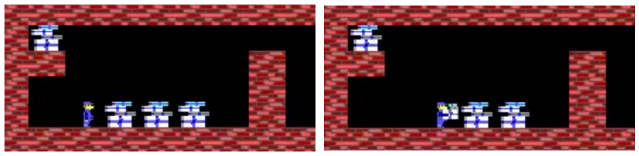

(a) Before pocketing the object. (b) After pocketing the object.

Figure 13: Pocket carrying from *Larry Loader* (1985).

6.3 Acknowledgements

We thank the anonymous referees for their helpful (and humorous) comments, which were used to significantly improve the paper. Two referees requested *more* Big Lebowski references; the authors were happy to abide.

References

[1] Google Scholar. scholar.google.com/scholar?q=sokoban, 2021.

[2] 1983parrothead. Famicom / NES Spelunker. knowyourmeme.com/memes/fc-nes-spelunker, 2010.

[3] J. Ani, S. Asif, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, J. Lynch, S. Scheffler, and A. Suhl. Pspace-completeness of pulling blocks to reach a goal. *J. Inf. Process.*, 28:929–941, 2020.

[4] D. Ashlock and J. Schonfeld. Evolution for automatic assessment of the difficulty of Sokoban boards. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[5] J. Culberson. Sokoban is PSPACE-complete. In *Proceedings of the 1st International Conference on Fun with Algorithm*, pages 65–76, 1998.

[6] E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In H. Ito, S. Leonardi, L. Pagli, and G. Prencipe, editors, *9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy*, volume 100 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[7] E. D. Demaine, D. H. Hendrickson, and J. Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 62:1–62:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[8] E. D. Demaine and M. Hoffmann. Pushing blocks is np-complete for noncrossing solution paths. In *Proc. 13th Canad. Conf. Comput. Geom*, pages 65–68, 2001.

[9] Detached Solutions. Puzzpack homepage. www.detachedsolutions.com/puzzpack, 2001.

[10] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215 – 228, 1999.

[11] M. Forišek. Computational complexity of two-dimensional platform games. In *FUN*, 2010.

[12] E. Friedman. Pushing blocks in gravity is NP-hard.

[13] E. Friedman. Problem of the month (March 2000). web.archive.org/web/20190218043712/www2.stetson.edu/~efriedma/mathmagic/0300.html, 2000.

[14] M. Fryers and M. T. Greene. Sokoban, 1995.

[15] A. Greenblatt, O. Hernandez, R. A. Hearn, Y. Hou, H. Ito, M. J. Kang, A. Williams, and A. Winslo. Turning around and around: Motion planning through thick and thin turnstiles. In *CCCG*, 2021.

[16] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. Mazezam levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, pages 109–110, 2017.

[17] Harvard CS50. Problem: Blockdude. docs.cs50.net/2016/ap/problems/blockdude/blockdude.html, 2016.

[18] R. A. Hearn and E. D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1–2):72–96, Oct. 2005.

[19] M. Hoffmann. Push-* is NP-hard. In *CCCG*, 2000.

[20] A. Junghanns and J. Schaeffer. Sokoban: Improving the search with relevance cuts. *Journal of Theoretical Computing Science*, 252:151–175, 1999.

[21] A. Junghanns and J. Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129:219–251, 2001.

[22] J. Kaplowitz. 'Block Dude' sweeps calculator gaming awards sixteenth year straight. thehardtimes.net/harddrive/block-dude-sweeps-calculator-gaming-awards-sixteenth-year-straight, 2018.

[23] B. Kartal, N. Sohre, and S. J. Guy. Data driven sokoban puzzle generation with Monte Carlo tree search. In *Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016.

[24] J. R. Lynch. *A framework for proving the computational intractability of motion planning problems.* PhD thesis, MIT, 9 2020. dspace.mit.edu/handle/1721.1/129205.

[25] NuclearHydrargyrum. Block Dude full game: 7 minutes, 43.640 seconds [WR]. www.youtube.com/watch?v=-YHLzswiEks, 2020.

[26] A. G. Pereira, M. Ritt, and L. S. Buriol. Optimal sokoban solving using pattern databases with specific domain knowledge. *Artificial Intelligence*, 227:52 – 70, 2015.

[27] V. Polishchuk. The box mover problem. In *CCCG*, 2004.

[28] J. Potma. *An exponential construction for Sokoban.* Bachelor's thesis, 2018.

[29] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[30] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, page 216–226, New York, NY, USA, 1978. Association for Computing Machinery.

[31] B. Sterner. Puzzpack v2.0. www.ticalc.org/archives/files/fileinfo/102/10289.html.

[32] B. Sterner. I am Brandon Sterner, the creator of Block Dude . . . from the PuzzPack for TI-83+. AMA. www.reddit.com/r/IAmA/comments/13ozy1/i_am_brandon_sterner_the_creator_of_block_dude, 2012.

[33] Students for a Democratic Society. Port Huron Statement. www2.iath.virginia.edu/sixties/HTML_docs/Resources/Primary/Manifestos/SDS_Port_Huron.html, 1962.

[34] M. O. Suleman, F. Syed, T. Syed, S. Arfeen, S. I. Behlim, and B. Mirza. Generation of sokoban stages using recurrentneural networks. *International Journal of Advanced Computer Science and Applications*, 8, 2017.

[35] J. Taylor, I. Parberry, and T. Parsons. Comparing player attention on procedurally generated vs. hand crafted Sokoban levels with an auditory stroop test. In *Foundations of Digital Games (FDG)*, 2015.

[36] The IT Law Wiki. Tetris Holding v. Xio Interactive. itlaw.wikia.org/wiki/Tetris_Holding_v._Xio_Interactive, 2012.

[37] Thinking Rabbit. History of Sokoban. sokoban.jp/history.html, 2021.

[38] R. Uehara. 日の目を見なかった問題たち その2 [Problems that didn't see the light of day Part 2]. www.jaist.ac.jp/ uehara/etc/la/99/index.html, 1999.

[39] G. Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54:595–621, 2014.

[40] Wikipedia. (倉庫番) — Wikipedia, the free encyclopedia. ja.wikipedia.org/w/index.php?title=%E5%80%89%E5%BA%AB%E7%95%AA&oldid=80549687, 2020.

[41] Wikipedia. Port Huron Statement. en.wikipedia.org/wiki/Port_Huron_Statement, 2021.

[42] A. Zich. Block Dude. azich.org/blockdude.

Flood Risk Analysis on Terrains

Pankaj K. Agarwal*

Abstract

An important problem in terrain analysis is modeling how water flows across a terrain and creates floods by filling up depressions. This talk discusses a number of flood-risk related problems: Given a terrain T, represented as a triangulated xy-monotone surface, a rain distribution R, and a volume of rain V, determine which portions of T are flooded and how water flows across T. Efficient algorithms are presented for flood-risk analysis under both single-flow-direction (SFD) as well as multi-flow-directions (MFD) models – in the former, water at a point can flow along one downward slope edge while in the latter, it can flow along multiple downward slope edges; the latter more accurately represents flooding events, but it is computational more challenging. These algorithms are also extended to handle uncertainty in terrain data, which must be incorporated into the terrain analysis.

About the Speaker

Prof. Agarwal earned his PhD in Computer Science from the Courant Institute of Mathematical Sciences at New York University. He joined Duke University in 1989 where he is currently the RJR Nabisco Professor of Computer Science and Mathematics. He served as the Chair of the Department of Computer Science 2004-10 and 2017-20. His research interests include geometric computing, spatial databases, ecological modeling, geographic information systems, sensor networks, computational molecular biology, and robotics. A Sloan Fellow, an ACM Fellow, and a National Young Investigator, Dr. Agarwal has authored four books and more than four hundred research articles. He serves or has served on the editorial boards of various journals and on the advisory boards of many institutes and centers.

*Duke University

Automatically Testing Containedness between Geometric Graph Classes defined by Inclusion, Exclusion and Transfer Axioms

Hannes Frey [*]   Lucas Böltz [†]

Abstract

We study classes of geometric graphs which all correspond to the following structural characteristic. For each instance of a vertex set drawn from a universe of possible vertices, each pair of vertices is either required to be connected, forbidden to be connected or existence or non-existence of an edge is undetermined. The conditions which require or forbid edges are universally quantified predicates defined over the vertex pair, and optionally over existence or non-existence of another edge originating at the vertex pair. We consider further two operations for symmetrizing such graph classes by either removing directed edges where the reverse edge is missing or by adding the missing reverse edges. We derive and prove correctness of a logical expression which is a necessary and sufficient condition for containedness relations between graph classes which are described this way. We apply the proposed method on classes of geometric graphs which are used as theoretical wireless network graph models. The models are constructed from three base class types and intersection combinations of them, with some considered directly and some considered in the two symmetrized variants.

1 Introduction

In this paper we study for a specific type of axiomatically described graphs how to prove containedness between geometric graph classes automatically. The method is illustrated with simple geometric graph models used in theoretical wireless network research. The method in general is applicable to any geometric graph class which can be described using a class of axioms containing *inclusion* conditions, which enforce the existence of an edge between two vertices, *exclusion* conditions, which prohibit the existence of an edge between two vertices and *transfer* conditions, which enforce the existence of edges between two vertices depending on existence of other edges originating from the two vertices to be connected. The axioms are depending on properties of the two vertices (inclusion, exclusion) or the three vertices (transfer) but not on the remaining vertices of the vertex set. For any two vertices where neither an edge is enforced or prohibited by the inclusion, exclusion or transfer conditions, we are free to choose to connect them by an edge or not.

Furthermore, we consider a class of transformations on graph classes, which we call *simple*. A simple transformation maps a source graph on an image graph with the same vertex set. The existence or nonexistence of an edge between two vertices in the image graph depends on the existence or nonexistence of the two possible edges between the two vertices in the source graph. Such dependencies can be described using Boolean expressions with two variables. In this work we focus on three of such simple transformations. The question we are interested in is to check containedness between graph classes defined this way: Given two graph classes A, B which can be described using *inclusion*, *exclusion* and *transfer* axioms, let $\mathsf{C} = \gamma_1(\mathsf{A})$ and $\mathsf{D} = \gamma_2(\mathsf{B})$ be the classes obtained by applying simple graph transformations γ_1 and γ_2. We want to check if class C is contained in class D.

The studied problem is not trivial. On the one hand, the predicates used for enforcing resp. prohibiting edges can be themselves formulas in a complex theory (in which properties or norms, costs or distances need to be specified). On the other hand, in order to prove or disprove that a class C is contained in a class $\mathsf{D} = \gamma_2(\mathsf{B})$ obtained from class B after applying a (simple) transformation γ_2 one would typically need to show that for every graph H in class C there exists a graph G in class B such that $H = \gamma_2(G)$, or to give an example of a graph H in C such that for every graph G in B we have $H \neq \gamma_2(G)$. A naive logical encoding of such properties would require quantification over binary relations. In order to avoid these problems, in this paper we show how expressions (involving only the predicates for enforcing and prohibiting edges and for transfer) for testing containedness between graph classes can be generated in a systematic way. For this, we prove a small model theorem. Our results can be regarded as a form of second-order quantifier elimination [4, 1].

The remainder of the paper is structured as follows. In the next section we illustrate the concept of the considered graph classes and transformations with an example referring to graph models used in theoretical wireless network research. In Section 3 we formally introduce the terminology, in particular notions such as in-

*University of Koblenz-Landau, frey@uni-koblenz.de
†University of Koblenz-Landau, boeltz@uni-koblenz.de

clusion, exclusion and transfer axioms and graph classes defined with this type of axioms, as well as the notion of simple transformations. In Section 4 we propose a method of generating finite axiomatizations for a class $\gamma(\mathsf{C})$ which do not refer to the edges of graphs in C and prove a small model theorem (i.e. we show that searching for a counter example with at most four vertices is sufficient for checking containedness relations for the graph classes and transformations we study in this paper). We use this for reducing subclass relationship tests to checking the satisfiability of expressions (built in a systematic way) which do not refer to the edges of the graphs but only to the inclusion, exclusion and transfer predicates used in the description of the classes. Based on these results we demonstrate in Section 5 how containedness relations can be proved for the 24 possible graph classes resulting from the examples discussed in Section 2. We conclude this contribution in Section 6 and discuss future research directions on extending the concept of logical graph class definitions and the concept of logical described transformations.

2 Geometric graph models

We illustrate our theory with graph models applied in theoretical wireless network research. Initial theoretical studies modeled such networks as *unit disk graphs* UDG where network vertices are connected iff their euclidean distance is less or equal than 1 and vertices are not connected by self-loops. The model was extended in several directions. We consider the extensions *quasi units disk graphs* [2, 5] and *directed transmission graphs* [3, 8, 7][1].

Quasi unit disk graphs $\mathsf{QUDG}(r)$ have a maximum distance 1 and a minimum distance $0 < r \leq 1$. Two distinct vertices with euclidean distance less or equal r are always connected. Two vertices with euclidean distance greater than 1 are never connected. Any other vertex pairs can be but need not be connected. To our knowledge, quasi unit disk graphs have only been studied as undirected graphs. We assume here as well undirected graphs when we speak of quasi unit disk graphs.

With directed transmission graphs $\mathsf{DTG}(r)$ every vertex v has an individual maximum communication distance $r(v) \leq r$ for some general $r > 0$. A directed edge exists from v to w iff the distance between v and w is less or equal $r(v)$.

Directed transmission graphs can be made symmetric by removing all directed edges where the reverse edge is missing as described in [6, 3] or by adding the missing reverse edges as it is additionally considered in [3]. We denote these graphs by $\mathsf{DTG}(r)^-$ and $\mathsf{DTG}(r)^+$, respectively.

[1]Note, we consider a maximum communication distance of 1. This is no limitation to other work where a maximum communication distance $r > 0$ is used instead. Obviously, scaling vertex positions with $1/r$ yields the same model with maximum communication distance 1.

In the following we describe the defining properties of the discussed graph models in terms of three base graph classes and the two discussed basic graph transformations, i.e. either removing directed edges where the reverse edge is missing or adding the missing reverse edges.

Definition 1 (The classes CRG, MinDG, MaxDG) *Let $0 < r \leq 1$. We consider a graph $G = (V, E)$. Let $d(u, v)$ be any metric defined on the vertices (e.g. Euclidean distance when vertices are located in Euclidean space). Let $E(u, v)$ be* true *if $uv \in E$ and* false *otherwise. Graph G is in the class of* min disk graphs $\mathsf{MinDG}(r)$, max disk graphs $\mathsf{MaxDG}(r)$, *or* connected region graphs CRG *iff it satisfies the following corresponding conditions:*

$$\begin{aligned}
\mathsf{MinDG}(r) \quad & \forall u, v \in V : u \neq v \wedge d(u, v) \leq r \Rightarrow E(u, v) \\
\mathsf{MaxDG}(r) \quad & \forall u, v \in V : d(u, v) > r \Rightarrow \neg E(u, v) \\
\mathsf{CRG} \quad & \forall u, v, w \in V : E(u, w) \wedge u \neq v \\
& \wedge\, d(u, v) \leq d(u, w) \Rightarrow E(u, v)
\end{aligned}$$

Definition 2 (Symmetric super- and subgraph) *Given a graph $G = (V, E)$ the graph transformations $G^+ = (V, E^+)$ and $G^- = (V, E^-)$, denoted as* symmetric supergraph *and* symmetric subgraph*, respectively, are defined by:*

$$\begin{aligned}
E^+ &= \{uv : uv \in E \text{ or } vu \in E\} \\
E^- &= \{uv : uv \in E \text{ and } vu \in E\}
\end{aligned}$$

When applied on a whole graph class C we write C^- and C^+ which means that the transformation is applied on each graph contained in C.

Obviously, the graph class of unit disk graphs can be built from the defined base classes as follows

$$\mathsf{UDG} = \mathsf{MinDG}(1) \cap \mathsf{MaxDG}(1)$$

where intersection $\cap$ means that the graph is contained in both classes, or stated alternatively, has to satisfy the logical predicates of both classes.

Together with the two basic graph transformations, the class of quasi unit disk graphs [2, 5] is given by

$$\begin{aligned}
\mathsf{QUDG}(r) &= (\mathsf{MinDG}(r) \cap \mathsf{MaxDG}(1))^- \\
&= (\mathsf{MinDG}(r) \cap \mathsf{MaxDG}(1))^+
\end{aligned}$$

The latter equality holds since an edge uv is optional iff vu is optional, i.e. $r < d(u, v) = d(v, u) \leq 1$. Obviously, $\mathsf{QUDG}(1)$ is the special case of unit disk graphs UDG.

The class of directed transmission graphs [3, 8, 7] is given by

$$\mathsf{DTG}(r) = \mathsf{MaxDG}(r) \cap \mathsf{CRG}$$

This follows immediately by setting the individual maximum communication distance of vertex v to $r(v) = \max\{d(v,w) : vw \in E\}$.

The symmetric variant considered in [6] has no maximum communication distance and is obtained by removing all directed edges where the reverse edge is missing. It is thus described by CRG^-. The symmetric variants considered in [3] are $\mathsf{DTG}(r)^-$ and $\mathsf{DTG}(r)^+$ in our graph class notation.

3 Axiomatic description of graph classes

3.1 Inclusion, exclusion and transfer predicates

Generally speaking, in the motivating example we consider graph classes C where each graph $G = (V, E)$ consists of vertices V of a vertex universe Ω and an edge relation E which satisfies

$$\forall u, v \in V : \pi^i_{\mathsf{C}}(u,v) \Rightarrow E(u,v) \quad (1)$$
$$\forall u, v \in V : \pi^e_{\mathsf{C}}(u,v) \Rightarrow \neg E(u,v) \quad (2)$$
$$\forall u, v, w \in V : E(u,w) \wedge \pi^t_{\mathsf{C}}(u,v,w) \Rightarrow E(u,v) \quad (3)$$
$$\forall u, v, w \in V : \neg E(u,w) \wedge \pi^t_{\mathsf{C}}(u,w,v) \Rightarrow \neg E(u,v) \quad (4)$$

We term $\pi^i_{\mathsf{C}}(u,v)$, $\pi^e_{\mathsf{C}}(u,v)$ and $\pi^t_{\mathsf{C}}(u,v,w)$ *inclusion*, *exclusion*, and *transfer predicates*, respectively. We will refer to the conditions (1), (2) and (3) as *inclusion*, *exclusion* and *transfer axioms*, accordingly (though conditions (3) and (4) are equivalent, we list both here for later reference).

We say that a graph $G = (V, E)$ is in class C (and write $G \in \mathsf{C}$) if G satisfies the conditions (1), (2), and (3). In this case we sometimes also write *G satisfies class* C. If the graph class C consists of all graphs satisfying a set of axioms of the form (1), (2), and (3), we will say that C is a graph class described by inclusion, exclusion and transfer axioms. In this work we will study such graph classes and transformations on them. We make the following additional assumptions:

Assumption 1. We assume that the inclusion, exclusion and transfer predicates are determined by the properties of their arguments $u, v, w \in V$ and are independent of any other vertices in V. We can ensure this, for instance, by requiring that these predicates are expressed as quantifier-free formulae over a background theory.

The inclusion, exclusion and transfer predicates defining the classes $\mathsf{MinDG(r)}$, $\mathsf{MaxDG(r)}$ and CRG in Section 2 can be expressed as quantifier-free formulas over a theory in which we can talk about points and distances between points:

$$\begin{aligned}\pi^i_{\mathsf{C}}(u,v) &:= u \neq v \wedge d(u,v) \leq r\\ \pi^e_{\mathsf{C}}(u,v) &:= d(u,v) > r\\ \pi^t_{\mathsf{C}}(u,v,w) &:= u \neq v \wedge d(u,v) \leq d(u,w)\end{aligned}$$

Assumption 2. We assume the transfer predicates $\pi^t_{\mathsf{C}}(u,v,w)$ to be *transitive*, i.e.

$$\pi^t_{\mathsf{C}}(u,v,w) \wedge \pi^t_{\mathsf{C}}(u,w,x) \Rightarrow \pi^t_{\mathsf{C}}(u,v,x)$$

Obviously, the transfer predicate defining CRG in Section 2 is transitive.

Assumption 3. We assume inclusion, exclusion and transitive transfer predicates to be *sound*, which means that they are not contradicting:

$$\forall u, v \in V : \pi^i_{\mathsf{C}}(u,v) \Rightarrow \neg\pi^e_{\mathsf{C}}(u,v)$$
$$\forall u, v, w \in V : \pi^i_{\mathsf{C}}(u,w) \wedge \pi^t_{\mathsf{C}}(u,v,w) \Rightarrow \neg\pi^e_{\mathsf{C}}(u,v)$$

We term a graph class to be sound in this case.

It is easy to see, as long $r_1 \leq r_2$ any conjunction of the predicates $\mathsf{MinDG}(r_1)$, $\mathsf{MaxDG}(r_2)$ and CRG defined in Section 2 satisfies the soundness condition.

3.2 Simple graph class transformations

The most general form of a graph transformation γ is a mapping which transfers a given graph G into a graph H. A graph transformation applied on a graph class C results in graph class $\gamma(\mathsf{C})$ which consists of all graphs H for which there exists a graph $G \in \mathsf{C}$ with $H = \gamma(G)$.

We say a graph transformation γ is *vertex preserving* if $\gamma(G)$ has the same vertex set as G, i.e. given $G = (V, E)$ the transformation is $\gamma(G) = (V, F)$ for some F. Obviously, a vertex preserving transformation γ on the *null graph* $(\emptyset, \emptyset)$ yields the null graph.

We term a vertex preserving graph transformation *simple* if there exists a Boolean function in two variables $\delta : \mathbb{B}^2 \to \mathbb{B}$ (where $\mathbb{B}$ is the two-element Boolean algebra) such that for all graphs $G = (V, E)$ the transformation $\gamma(G) = (V, F)$ satisfies

$$\forall u, v \in V : F(u,v) = \delta(E(u,v), E(v,u))$$

Symmetric super graph $(\cdot)^+$ and symmetric subgraph $(\cdot)^-$ are obviously simple transformations with boolean functions $\vee$ and $\wedge$, respectively. As there are 16 possible Boolean functions of the form $\delta : \mathbb{B}^2 \to \mathbb{B}$, in total there exist 16 possible vertex preserving simple graph transformations. In this work we consider three of them, $(\cdot)^+$, $(\cdot)^-$ and the identity $(\cdot)^{\text{id}}$ (the identity defined as $\delta(x,y) = x$).

4 Proving graph class containedness relations

4.1 General properties

As stated by the following lemmas, it is easy to see that graph classes described by inclusion, exclusion and transfer axioms which satisfy Assumption 1, and graph classes obtained by applying a simple transformation on such graph classes are closed under induced subgraphs[2]. Moreover, given a sound graph class, for each finite vertex set V we can find an edge configuration such that the given graph class axioms are satisfied.

Lemma 1 *Let C be a graph class described by inclusion, exclusion and transfer predicates satisfying Assumption 1. It holds:*

- *The null graph $(\emptyset, \emptyset)$ satisfies C.*
- *Let $G = (V, E)$ satisfy C. Every subgraph G' induced by $V' \subseteq V$ satisfies C.*

Proof. Obviously $V = \emptyset$ satisfies any universally quantified condition and thus also conditions (1), (2) and (3).

For the second statement, consider a graph $G = (V, E)$ which satisfies C. Conditions (1), (2) and (3) are satisfied for V. Let $V' \subseteq V$. Consider one of the conditions defining the class C, say condition (1):

$$\forall u, v:\ \pi^i_{\mathsf{C}}(u, v) \Rightarrow E(u, v).$$

Let $u, v \in V'$. Then $u, v \in V$. Assume that $\pi^i_{\mathsf{C}}(u, v)$ is true (relative to the set of edges V'). By Assumption 1, the value of this predicate is independent of the other vertices in the graph, so $\pi^i_{\mathsf{C}}(u, v)$ is true also when we consider the whole set of vertices in $G = (V, E)$. As G satisfies condition (1), it follows that $E(u, v)$ is true (i.e. there is an edge from u to v in G). Since $E' = E \cap (V' \times V')$, there is an edge from u to v also in G'.

The same line of argumentation can be used for conditions (2) and (3). This shows that also the subgraph G' induced by V' satisfies C. $\square$

Remark 1 *The first part of Lemma 1 holds for every graph class which consists of all graphs satisfying a set of universally quantified axioms.*

The second part of Lemma 1 is vacuously true if Assumption 3 does not hold, because in this case the graph class contains only the empty graph which does not have proper subgraphs.

If Assumption 1 is not satisfied, then the second part of Lemma 1 does not necessarily hold. Consider for instance the graph class C described by axioms of the form (1), (2), and (3), where for all u, v, x:

$$\pi^i_{\mathsf{C}}(u, v) = \forall w((w \neq u \wedge w \neq v) \Rightarrow (E(u, w) \wedge E(w, v)))$$
$$\pi^e_{\mathsf{C}}(u, v) = \text{false}, \quad \pi^t_{\mathsf{C}}(u, x, v) = \text{false}.$$

In this graph class, the graph $G = (\{a, b, c, d, e\}, \{(a, b), (b, e), (a, d), (d, e)\})$ satisfies the axioms (1), (2), and (3), but the subgraph induced by $V' = \{a, b, d, e\}$ does not satisfy axiom (1).

[2]Given a graph $G = (V, E)$, the graph G' induced by $V' \subseteq V$ consists of vertex set V' and all edges from E connecting the vertices in V'.

Lemma 2 *Let C be a graph class defined by inclusion, exclusion and transfer predicates satisfying Assumption 1 and γ be a vertex preserving graph transformation. It holds:*

- *The null graph $(\emptyset, \emptyset)$ satisfies $\gamma(\mathsf{C})$.*
- *If γ is simple and if $H = (V, F)$ satisfies $\gamma(\mathsf{C})$, then every subgraph H' induced by $V' \subseteq V$ satisfies $\gamma(\mathsf{C})$.*

Proof. The first statement follows immediately: $(\emptyset, \emptyset) = \gamma((\emptyset, \emptyset))$ since γ is vertex preserving, and $\gamma((\emptyset, \emptyset)) \in \gamma(\mathsf{C})$ since $(\emptyset, \emptyset) \in \mathsf{C}$ with Lemma 1.

For the second statement, consider a graph $H = (V, F)$ which satisfies $\gamma(\mathsf{C})$. With γ being vertex preserving, there exists a graph $G = (V, E) \in \mathsf{C}$ such that $H = \gamma(G)$. Let $V' \subseteq V$, $H' = (V', F')$ be the subgraph of H and $G' = (V', E')$ be the subgraph of G induced by V', respectively.

Let δ be the boolean function of the simple transformation γ. For all $u, v \in V'$ holds: uv is an edge of $\gamma(G')$ iff

$$\begin{aligned}\delta(E'(u, v), E'(v, u)) &\Leftrightarrow \delta(E(u, v), E(v, u)) \\ &\Leftrightarrow F(u, v) \Leftrightarrow F'(u, v) \qquad (5)\end{aligned}$$

This implies $H' = \gamma(G')$, and thus $H' \in \gamma(\mathsf{C})$, since $G' \in \mathsf{C}$ with Lemma 1. $\square$

Remark 2 *The second part of Lemma 2 does not necessarily hold if the transformation γ is not simple. Consider for instance the transformation defined by*

$$\forall u, v\Big(\ \phi(u, v) \to F(u, v)\Big),$$
$$\forall u, v\Big(\neg\phi(u, v) \to \neg F(u, v)\Big)$$

where $\phi(u, v) := \forall w\Big((u \neq w \wedge v \neq w) \to (E(u, w) \wedge E(w, v))\Big).$

Let C be a graph class defined by inclusion, exclusion and transfer axioms containing the graph $G = (\{a, b, c, d\}, \{(a, b), (b, d)\})$ due to the fact that in this specific graph $\pi^i_{\mathsf{C}}(a, b) = \pi^i_{\mathsf{C}}(b, d) = \text{true}$, $\pi^e_{\mathsf{C}}(x, y) = \text{true}$ for all other pairs of vertices, and $\pi^t_{\mathsf{C}}(u, v, w)$ is false everywhere. Then $H = \gamma(G) = (\{a, b, c, d\}, \emptyset) \in \gamma(\mathsf{C})$. However, the subgraph H' of H induced by $V' = \{a, b, d\}$ is not in $\gamma(\mathsf{C})$: Assume there exists a graph $G' \in \mathsf{C}$ with $H' = \gamma(G')$. Since $G' \in \mathsf{C}$, due to the fact that $\pi^i_{\mathsf{C}}(a, b) = \pi^i_{\mathsf{C}}(b, d) = \text{true}$, $\pi^e_{\mathsf{C}}(x, y) = \text{true}$ for all other pairs of vertices, and π^t_{C} is false everywhere, the set of edges of G' must be $E = \{(a, b), (b, d)\}$. But then in this case $\forall w(a \neq w \wedge d \neq w \to E(a, w) \wedge E(w, d))$ holds, so in $\gamma(G')$, an edge (a, d) should exist. However, this edge is missing in H'.

Lemma 3 *Let C be a graph class defined over sound conditions (Assumption 3). For each finite set $V \subseteq \Omega$ at least one graph $G = (V, E)$ exists which satisfies C.*

Proof. Consider any such set V. Set $E(u, v) =$ true if $\pi^i_{\mathsf{C}}(u, v)$ holds or if there exist a $w \in V$ such that $\pi^i_{\mathsf{C}}(u, w) \wedge \pi^t_{\mathsf{C}}(u, v, w)$ holds. Otherwise set $E(u, v) =$ false. This configuration obviously satisfies (1) and (3), and does not contradict (2) since the conditions are sound. Thus, the such defined graph $G = (V, E)$ satisfies C. □

Note that if C is a graph class defined using inclusion, exclusion and transfer axioms and γ is a simple vertex preserving transformation, it is not immediate that $\gamma(\mathsf{C})$ can also be described by inclusion, exclusion and transfer axioms.

In Section 4.2 we show that when $\gamma \in \{(\cdot)^{\text{id}}, (\cdot)^-, (\cdot)^+\}$ we can describe the properties of the edge-relation F of graphs $H = (V, F)$ in $\gamma(\mathsf{C})$ using axioms in which only F and the inclusion, exclusion and transfer predicates $\pi^i_{\mathsf{C}}(u, v), \pi^e_{\mathsf{C}}(u, v)$ and $\pi^t_{\mathsf{C}}(u, v, w)$ are used.

4.2 Axiomatization for $\gamma(\mathsf{C})$

We next show the connection between edge configurations $F(u, v)$ for a graph being in a class transformed by $(\cdot)^{\text{id}}$, $(\cdot)^-$ or $(\cdot)^+$ and the inclusion, exclusion and transfer predicates of that class.

In this subsection and the following ones, all proofs are given for $(\cdot)^{\text{id}}$ and $(\cdot)^+$ transformation. The proofs for $(\cdot)^-$ are omitted. They are basically symmetric to the proofs for $(\cdot)^+$.

Lemma 4 *Let B be a sound graph class described by inclusion, exclusion and transitive transfer predicates $\pi^i_{\mathsf{B}}(u, v)$, $\pi^e_{\mathsf{B}}(u, v)$, $\pi^t_{\mathsf{B}}(u, v, w)$, respectively. Let γ be any of the vertex preserving, simple graph transformations in $\{(\cdot)^{\text{id}}, (\cdot)^-, (\cdot)^+\}$. A graph $H = (V, F)$ is in class $\gamma(\mathsf{B})$ iff for every $u, v \in V$ the edge relation $F(u, v)$ satisfies the implications depicted in Table 1 for that transformation.*

Proof. The proof for $(\cdot)^{\text{id}}$ is trivial. By definition, $H \in \gamma(\mathsf{B})$ if and only if there exists a graph $G \in \mathsf{B}$ such that $H = \gamma(G) = G$. But this is the case if and only if $H \in \mathsf{B}$. Thus, in this case $\gamma(\mathsf{B}) = \mathsf{B}$, so $\gamma(\mathsf{B})$ is described by the same axioms as B.

For $(\cdot)^+$, two implication directions "⇒" and "⇐" have to be shown.

⇒: With $H = (V, F) \in \mathsf{B}^+$ there exists a graph $G = (V, E) \in \mathsf{B}$ with $H = G^+$. We consider the cases $F(u, v)$ and $\neg F(u, v)$ for each $u, v \in V$.

γ	Edge inclusion and exclusion implications			$\mu(\gamma)$
$H \in \mathsf{B}$	$F(u,v)$	$\Rightarrow$	$\neg\pi^e_{\mathsf{B}}(u,v)$	3
	$\neg F(u,v)$	$\Rightarrow$	$\neg\pi^i_{\mathsf{B}}(u,v) \wedge \forall w :$ $\neg F(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,v,w)$	
$H \in \mathsf{B}^-$	$F(u,v)$	$\Rightarrow$	$F(v,u) \wedge \neg\pi^e_{\mathsf{B}}(u,v) \wedge \forall w :$ $\neg\pi^e_{\mathsf{B}}(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,w,v)$	4
	$\neg F(u,v)$	$\Rightarrow$	$(\neg\pi^i_{\mathsf{B}}(u,v) \wedge \forall x : \neg F(u,x) \wedge$ $\neg\pi^i_{\mathsf{B}}(u,x) \vee \neg\pi^t_{\mathsf{B}}(u,v,x)) \vee$ $(\neg\pi^i_{\mathsf{B}}(v,u) \wedge \forall y : \neg F(v,y) \wedge$ $\neg\pi^i_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,u,y))$	
$H \in \mathsf{B}^+$	$F(u,v)$	$\Rightarrow$	$(\neg\pi^e_{\mathsf{B}}(u,v) \wedge \forall x : F(u,x) \wedge$ $\neg\pi^e_{\mathsf{B}}(u,x) \vee \neg\pi^t_{\mathsf{B}}(u,x,v)) \vee$ $(\neg\pi^e_{\mathsf{B}}(v,u) \wedge \forall y : F(v,y) \wedge$ $\neg\pi^e_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,y,u))$	4
	$\neg F(u,v)$	$\Rightarrow$	$\neg F(v,u) \wedge \neg\pi^i_{\mathsf{B}}(u,v) \wedge \forall w :$ $\neg\pi^i_{\mathsf{B}}(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,v,w)$	

Table 1: The expressions implied by edge existence and non-existence under the considered vertex preserving simple graph class transformations. Quantification is over all vertices V of the considered graph $H = (V, F)$. The maximum size of a minimum witness graph to find a counter example is expressed by $\mu(\gamma)$.

Consider $\neg F(u, v)$ first. Definition of G^+ implies that $\neg E(u, v)$ has to be satisfied. With graph class axiom (1) this implies

$$\neg\pi^i_{\mathsf{B}}(u, v) \tag{6}$$

Moreover, graph class axiom (3) implies that for all $w \in V$ the expression $\neg E(u, w) \vee \neg\pi^t_{\mathsf{B}}(u, v, w)$ has to be satisfied. Since $\neg E(u, w)$ implies $\neg\pi^i_{\mathsf{B}}(u, w)$ with graph class axiom (1) we have that

$$\forall w : \neg\pi^i_{\mathsf{B}}(u, w) \vee \neg\pi^t_{\mathsf{B}}(u, v, w) \tag{7}$$

has to be satisfied.

With (6) and (7) and as well the fact that the transformation $(\cdot)^+$ is symmetric, i.e. $\neg F(u, v) \Leftrightarrow \neg F(v, u)$, we get the edge exclusion implication

$$\neg F(u, v) \Rightarrow \neg F(v, u) \wedge \neg\pi^i_{\mathsf{B}}(u, v)$$
$$\wedge \forall w : \neg\pi^i_{\mathsf{B}}(u, w) \vee \neg\pi^t_{\mathsf{B}}(u, v, w)$$

as depicted for B^+ in Table 1.

Consider $F(u, v)$ next. Definition of G^+ implies that $E(u, v) \vee E(v, u)$ has to be satisfied. Consider first that $E(u, v)$ is satisfied. Graph class axiom (2) implies $\neg\pi^e_{\mathsf{B}}(u, v)$. Moreover, graph class axiom (4) implies that for all $x \in V$ the expression $E(u, x) \vee \neg\pi^t_{\mathsf{B}}(u, x, v)$ holds. $E(u, x)$ implies $\neg\pi^e_{\mathsf{B}}(u, x)$ with graph class axiom (2). In addition, $E(u, x)$ implies $F(u, x)$ since $ux \in G^+$ requires only $ux \in E$ or $xu \in E$. Thus, we have that

$$\neg\pi^e_{\mathsf{B}}(u, v) \wedge \forall x : F(u, x) \wedge \neg\pi^e_{\mathsf{B}}(u, x) \vee \neg\pi^t_{\mathsf{B}}(u, x, v) \tag{8}$$

has to be satisfied.

The case that $E(v,u)$ is satisfied is symmetric to the case that $E(u,v)$ is satisfied. With the same arguments follows that

$$\neg\pi^e_{\mathsf{B}}(v,u) \wedge \forall y : F(v,y) \wedge \neg\pi^e_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,y,u) \quad (9)$$

has to be satisfied.

With (8) and (9) we get the edge inclusion implication

$$\begin{aligned} &F(u,v) \Rightarrow \\ &(\neg\pi^e_{\mathsf{B}}(u,v) \wedge \forall x : F(u,x) \wedge \neg\pi^e_{\mathsf{B}}(u,x) \vee \neg\pi^t_{\mathsf{B}}(u,x,v)) \vee \\ &(\neg\pi^e_{\mathsf{B}}(v,u) \wedge \forall y : F(v,y) \wedge \neg\pi^e_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,y,u)) \end{aligned}$$

as depicted for B^+ in Table 1.

$\Leftarrow$: Assume now that we have a graph $H = (V,F)$ such that the edge inclusion and exclusion implications depicted for B^+ in Table 1 are satisfied. We construct a graph $G = (V,E)$ such that $G \in \mathsf{B}$ and $G^+ = H$, which then implies $H \in \mathsf{B}^+$.

Construction of $G = (V,E)$ is done in two steps, a *preprocessing* and a *postprocessing* step. We begin with a configuration, where all $E(u,v)$ are set to true. First we keep those $E(u,v)$ to true resp. set those $E(u,v)$ to false which are enforced or forbidden due to the graph class axioms of class B. We then set the remaining edges $E(u,v)$ which have not been determined in preprocessing accordingly such that G still satisfies B and at the same time satisfies $G^+ = H$.

Preprocessing (forbidden edges): In order to satisfy graph class axiom (2), for all $u,v \in V$ which satisfy $\pi^e_{\mathsf{B}}(u,v)$, we set $E(u,v)$ to false; in this case, in order to satisfy also axiom (3), we set $E(u,y)$ to false for all $y \in V$ such that $\pi^t_{\mathsf{B}}(u,v,y)$ is satisfied.

Preprocessing (enforced edges): In order to satisfy graph class axiom (1), for all $u,v \in V$ which satisfy $\pi^i_{\mathsf{B}}(u,v)$, we set $E(u,v)$ to true; in this case, in order to satisfy also axiom (3), we set $E(u,x)$ to true for all $x \in V$ for which $\pi^t_{\mathsf{B}}(u,x,v)$ is satisfied. Soundness of inclusion, exclusion and transfer predicates assures that we do not set any $E(u,v)$ and $E(u,y)$ to true which we have set to false before.

Postprocessing (omitted optional edges): After the preprocessing we continue as follows. For $\neg F(u,v)$ we have by exclusion implication for B^+ in Table 1 that

$$\neg\pi^i_{\mathsf{B}}(u,v) \wedge \forall w : \neg\pi^i_{\mathsf{B}}(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,v,w)$$

is satisfied. Assume $E(u,v)$ was set to true in the preprocessing. Since $\neg\pi^i_{\mathsf{B}}(u,v)$ holds, $E(u,v)$ could only be set to true due to a w such that $\pi^i_{\mathsf{B}}(u,w)$ and $\pi^t_{\mathsf{B}}(u,v,w)$ holds. This, however, is not possible since $\neg\pi^i_{\mathsf{B}}(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,v,w)$ holds for all w. Thus, $E(u,v)$ was not set to true in the preprocessing. It was either already set to false, or if not, we can set it to false in the postprocessing.

If we set $E(u,v)$ to false in the postprocessing, we have to assure that graph class axiom (3) remains valid. We have to set $E(u,w)$ to false for all $w \in V$ which satisfy $\pi^t_{\mathsf{B}}(u,v,w)$. Assume any such w was set to true in the preprocessing. Since $\pi^t_{\mathsf{B}}(u,v,w)$ holds, the preprocessing then would have set $E(u,v)$ to true which is a contradiction. Thus, all $E(u,w)$ which satisfy $\pi^t_{\mathsf{B}}(u,v,w)$ can be set to false, if they have not already been set to false in the preprocessing.

Moreover, the exclusion implication also contains $\neg F(u,v) \Rightarrow \neg F(v,u)$. Thus, also

$$\neg\pi^i_{\mathsf{B}}(v,u) \wedge \forall w : \neg\pi^i_{\mathsf{B}}(v,w) \vee \neg\pi^t_{\mathsf{B}}(v,u,w)$$

is satisfied. Analogously we have that $E(v,u)$ can be set to false if it was not already set to false in the preprocessing step.

Thus, after postprocessing for each $\neg F(u,v)$ in H we can set $E(u,v)$ and $E(v,u)$ to false while satisfying the graph class axioms for B. Since $E(u,v)$ and $E(v,u)$ are false, the edge uv will not be present in G^+.

Postprocessing (added optional edges): For $F(u,v)$ we have by inclusion implication for B^+ in Table 1 that

$$\begin{aligned} &(\neg\pi^e_{\mathsf{B}}(u,v) \wedge \forall x : F(u,x) \wedge \neg\pi^e_{\mathsf{B}}(u,x) \vee \neg\pi^t_{\mathsf{B}}(u,x,v)) \vee \\ &(\neg\pi^e_{\mathsf{B}}(v,u) \wedge \forall y : F(v,y) \wedge \neg\pi^e_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,y,u)) \end{aligned}$$

is satisfied.

Assume first that $\neg\pi^e_{\mathsf{B}}(u,v) \wedge \forall x : F(u,x) \wedge \neg\pi^e_{\mathsf{B}}(u,x) \vee \neg\pi^t_{\mathsf{B}}(u,x,v)$ holds. Assume $E(u,v)$ was already set to false before. This could not happen during the preprocessing. In fact, $\pi^e_{\mathsf{B}}(u,v)$ is not satisfied, and for all $x \in V$ which satisfy $\pi^e_{\mathsf{B}}(u,x)$, the transfer predicate $\pi^t_{\mathsf{B}}(u,x,v)$ is not satisfied.

Thus, $E(u,v)$ must have been set to false in the postprocessing step where optional edges were omitted. Since $F(u,v)$ holds, $E(u,v)$ was not set to false in that step in order to omit $F(u,v)$. It could thus only be set to false since another edge $F(u,w)$ was omitted. In this case $E(u,w)$ is set to false. However, since $F(u,w) \vee \neg\pi^t_{\mathsf{B}}(u,w,v)$ holds for all w, edge $E(u,v)$ was not set to false in this case either.

Thus, in none of the previous steps $E(u,v)$ was set to false and we can keep it true without violating the graph class axioms. Since $E(u,v) = \text{true}$, edge uv will be present in G^+.

Assume next that $\neg\pi^e_{\mathsf{B}}(v,u) \wedge \forall y : F(v,y) \wedge \neg\pi^e_{\mathsf{B}}(v,y) \vee \neg\pi^t_{\mathsf{B}}(v,y,u)$ holds. With the assumed $\neg F(u,v) \Rightarrow \neg F(v,u)$ we have $F(v,u) = \text{true}$ since $F(u,v)$ is assumed. Thus, the situation is symmetric to the previous case. It follows analogously that in none of the previous steps $E(v,u)$ was set to false and we can keep it true without violating the graph class axioms. Again, since $E(v,u) = \text{true}$, edge vu will be present in G^+. □

4.3 Minimal counter examples

We are interested in solving the following problem: Assume that we have two graph classes, A and B, defined by inclusion, exclusion and transfer axioms, and two transformations γ_1 and γ_2. We are interested in checking whether $\gamma_1(A) \subseteq \gamma_2(B)$.

In general, $\mathsf{A} \not\subseteq \gamma(\mathsf{B})$ iff there exists a graph $G = (V, E) \in \mathsf{A}$ which is not contained in $\gamma(\mathsf{B})$. We term such graph a *witness graph*. Moreover, we term G a *minimum witness graph* if $|V| \leq |V'|$ for all witness graphs $G' = (V', E')$. In general, a minimum witness graph is not necessarily unique.

The size of a minimum witness graph depends on the transformation γ and the transformed class B. As summarized in the next theorem, for the simple vertex preserving transformations $(\cdot)^{\text{id}}$, $(\cdot)^-$, and $(\cdot)^+$, with Lemma 4 we get upper bounds on the number of vertices of a *minimum witness graph*. We term this value the *maximum size of a minimum witness graph* $\mu(\gamma)$.

Theorem 5 *Let γ_1 and γ_2 be any of the vertex preserving simple graph transformations $(\cdot)^{\text{id}}$, $(\cdot)^-$, and $(\cdot)^+$. Let A and B be graph classes defined by inclusion, exclusion and transfer predicates which satisfy Assumption 1, 2 and 3. Let $\mu(\gamma_2)$ as defined in Table 1 for transformation γ_2. $\gamma_1(\mathsf{A}) \not\subseteq \gamma_2(\mathsf{B})$ iff there exists a graph $G = (V, E)$ with $V \subseteq \Omega$ and $|V| \leq \mu(\gamma_2)$ such that $G \in \gamma_1(\mathsf{A})$ and $G \notin \gamma_2(\mathsf{B})$.*

Proof. With $\gamma_1(\mathsf{A}) \not\subseteq \mathsf{B}$ follows that there exists a graph $G = (V, E)$ with $V \subseteq \Omega$ such that $G \in \gamma_1(\mathsf{A})$ and $G \notin \mathsf{B}$. Consider such graph $G = (V, E)$ which is minimal in $|V|$. Assume $|V| > 3$.

Since $G \notin \mathsf{B}$, Lemma 4 implies that there exist $u_0, v_0, x_0 \in V$ such that the following expression is satisfied

$$\begin{aligned}&[F(u_0, v_0) \wedge \pi^e_{\mathsf{B}}(u_0, v_0)] \qquad (10)\\ \vee&[\neg F(u_0, v_0) \wedge (\pi^i_{\mathsf{B}}(u_0, v_0) \vee F(u_0, x_0) \wedge \pi^t_{\mathsf{B}}(u_0, v_0, x_0))]\end{aligned}$$

Since $|V| > 3$ there exists a $y_0 \in V$ which differs from u_0, v_0, and x_0. Consider the subgraph G' induced by $V \setminus \{y_0\}$. Edge u_0v_0 exists in G' iff it exists in G. Moreover, edge u_0x_0 exists in G' iff it exists in G. Thus, expression (10) also holds for G' and thus G' is not contained in B.

However, with Lemma 2 the subgraph G' induced by $V \setminus \{y_0\}$ is a graph in $\gamma_1(\mathsf{A})$. Since G was assumed to be minimal in $|V|$, graph G' must be contained in B (otherwise G would not be a minimum witness graph), a contradiction.

With $\gamma_1(\mathsf{A}) \not\subseteq \mathsf{B}^+$ follows that there exists a graph $G = (V, E)$ with $V \subseteq \Omega$ such that $G \in \gamma_1(\mathsf{A})$ and $G \notin \mathsf{B}^+$. Consider such graph $G = (V, E)$ which is minimal in $|V|$. Assume $|V| > 4$.

Since $G \notin \mathsf{B}^+$, Lemma 4 implies that there exist $u_0, v_0, x_0, y_0 \in V$ such that the following expression is satisfied

$$\begin{aligned}&[F(u_0, v_0) \wedge (\pi^e_{\mathsf{B}}(u_0, v_0)\\ &\vee (\neg F(u_0, x_0) \vee \pi^e_{\mathsf{B}}(u_0, x_0)) \wedge \pi^t_{\mathsf{B}}(u_0, x_0, v_0))\\ &\wedge (\pi^e_{\mathsf{B}}(v_0, u_0)\\ &\vee (\neg F(v_0, y_0) \vee \pi^e_{\mathsf{B}}(v_0, y_0)) \wedge \pi^t_{\mathsf{B}}(v_0, y_0, u_0))]\\ &\vee [\neg F(u_0, v_0) \wedge (F(v_0, u_0)\\ &\vee \pi^i_{\mathsf{B}}(u_0, v_0) \vee \pi^i_{\mathsf{B}}(u_0, x_0) \wedge \pi^t_{\mathsf{B}}(u_0, v_0, x_0))] \qquad (11)\end{aligned}$$

Since $|V| > 4$ there exists a $z_0 \in V$ which differs from u_0, v_0, x_0 and y_0. Consider the subgraph G' induced by $V \setminus \{z_0\}$. Edge u_0v_0 exists in G' iff it exists in G. Moreover, edges u_0x_0 and v_0y_0 exist in G' iff they exist in G. Thus, expression (11) also holds for G' and thus G' is not contained in B^+.

However, with Lemma 2 the subgraph G' induced by $V \setminus \{z_0\}$ is a graph in $\gamma_1(\mathsf{A})$. Since G was assumed to be minimal in $|V|$, graph G' must be contained in B^+ (otherwise G would not be a minimum witness graph), a contradiction. □

4.4 Expressions for proving subclass relations

In this subsection we consider different combinations of existing and non existing edges and provide formulas for the transformations $(\cdot)^{\text{id}}$, $(\cdot)^-$, and $(\cdot)^+$ such that if these formulas are true, then the considered combination of existing and non existing edges is not possible for these transformations.

A combination of these formulas in negated and non-negated form yields an expression for checking if one graph class is contained in another one or not. If not, the expression yields axioms for a counter example.

Theorem 6 *Let γ_1 and γ_2 be two of the vertex preserving simple graph transformations $(\cdot)^{\text{id}}$, $(\cdot)^-$, and $(\cdot)^+$. Let A and B be graph classes defined by inclusion, exclusion and transfer predicates satisfying Assumptions 1, 2 and 3. Let $\rho^k_{\gamma_1(\mathsf{A})}(\cdot,\cdot)$ resp. $\rho^k_{\gamma_2(\mathsf{B})}(\cdot,\cdot)$ be the expressions listed in Table 2 for γ_1 and γ_2. Moreover, let $\mu(\gamma_2)$ be the maximum size of a minimum witness graph as defined for γ_2 in Table 1. It holds: $\gamma_1(\mathsf{A}) \not\subseteq \gamma_2(\mathsf{B})$ iff there exists a graph $H = (V, F)$ with $|V| \leq \mu(\gamma_2)$ such that*

$$\bigvee_{k=1}^{6} \neg\rho^k_{\gamma_1(\mathsf{A})}(S^k, T^k) \wedge \rho^k_{\gamma_2(\mathsf{B})}(S^k, T^k) \qquad (12)$$

is satisfied, where S^k and T^k are specific subsets $S^k \subseteq F$ and $T^k \subseteq V \times V \setminus F$ as defined in Table 2 and V is the vertex set of H for each $\rho^k_{\mathsf{X}}(\cdot,\cdot)$ expression.

Proof. Theorem 5 states that $\gamma_1(\mathsf{A}) \not\subseteq \gamma_2(\mathsf{B})$ is satisfied iff there exists a counter example with size at most $\mu(\gamma_2)$.

γ	Edge inclusion and exclusion implications		
$(\cdot)^{\mathrm{id}}$	$\rho^1_{\mathsf{B}}(\{uv\},\{\emptyset\})$	$=$	$\pi^e_{\mathsf{B}}(u,v)$
	$\rho^2_{\mathsf{B}}(\{\emptyset\},\{uv\})$	$=$	$\pi^i_{\mathsf{B}}(u,v)$
	$\rho^3_{\mathsf{B}}(\{uv\},\{vu\})$	$=$	$\rho^1_{\mathsf{B}}(\{uv\},\{\emptyset\}) \vee \rho^2_{\mathsf{B}}(\{\emptyset\},\{vu\})$
	$\rho^4_{\mathsf{B}}(\{uv\},\{uw\})$	$=$	$\rho^1_{\mathsf{B}}(\{uv\},\{\emptyset\}) \vee \rho^2_{\mathsf{B}}(\{\emptyset\},\{uw\}) \vee \pi^t_{\mathsf{B}}(u,w,v) \vee v = w$
	$\rho^5_{\mathsf{B}}(\{uv,vu\},\{uw,vx\})$	$=$	$\rho^4_{\mathsf{B}}(\{uv\},\{uw\}) \vee \rho^4_{\mathsf{B}}(\{vu\},\{vx\}) \vee v = w \vee u = x$
	$\rho^6_{\mathsf{B}}(\{uw,vx\},\{uv,vu\})$	$=$	$\rho^4_{\mathsf{B}}(\{uw\},\{uv\}) \vee \rho^4_{\mathsf{B}}(\{vx\},\{vu\}) \vee v = w \vee u = x$
$(\cdot)^+$	$\rho^1_{\mathsf{B}+}(\{uv\},\{\emptyset\})$	$=$	$\pi^e_{\mathsf{B}}(u,v) \wedge \pi^e_{\mathsf{B}}(v,u)$
	$\rho^2_{\mathsf{B}+}(\{\emptyset\},\{uv\})$	$=$	$\pi^i_{\mathsf{B}}(u,v) \vee \pi^i_{\mathsf{B}}(v,u)$
	$\rho^3_{\mathsf{B}+}(\{uv\},\{vu\})$	$=$	true
	$\rho^4_{\mathsf{B}+}(\{uv\},\{uw\})$	$=$	$\rho^1_{\mathsf{B}+}(\{uv\},\{\emptyset\}) \vee \rho^2_{\mathsf{B}+}(\{\emptyset\},\{uw\}) \vee (\pi^t_{\mathsf{B}}(u,w,v) \wedge \pi^e_{\mathsf{B}}(v,u)) \vee v = w$
	$\rho^5_{\mathsf{B}+}(\{uv,vu\},\{uw,vx\})$	$=$	$\rho^4_{\mathsf{B}+}(\{uv\},\{uw\}) \vee \rho^4_{\mathsf{B}+}(\{vu\},\{vx\}) \vee (\pi^t_{\mathsf{B}}(u,w,v) \wedge \pi^t_{\mathsf{B}}(v,x,u)) \vee v = w \vee u = x$
	$\rho^6_{\mathsf{B}+}(\{uw,vx\},\{uv,vu\})$	$=$	$\rho^4_{\mathsf{B}+}(\{uw\},\{uv\}) \vee \rho^4_{\mathsf{B}+}(\{vx\},\{vu\}) \vee v = w \vee u = x$
$(\cdot)^-$	$\rho^1_{\mathsf{B}-}(\{uv\},\{\emptyset\})$	$=$	$\pi^e_{\mathsf{B}}(u,v) \vee \pi^e_{\mathsf{B}}(v,u)$
	$\rho^2_{\mathsf{B}-}(\{\emptyset\},\{uv\})$	$=$	$\pi^i_{\mathsf{B}}(u,v) \wedge \pi^i_{\mathsf{B}}(v,u)$
	$\rho^3_{\mathsf{B}-}(\{uv\},\{vu\})$	$=$	true
	$\rho^4_{\mathsf{B}-}(\{uw\},\{uv\})$	$=$	$\rho^1_{\mathsf{B}-}(\{uw\},\{\emptyset\}) \vee \rho^2_{\mathsf{B}-}(\{\emptyset\},\{uv\}) \vee (\pi^t_{\mathsf{B}}(u,v,w) \wedge \pi^i_{\mathsf{B}}(v,u)) \vee v = w$
	$\rho^5_{\mathsf{B}-}(\{uv,vu\},\{uw,vx\})$	$=$	$\rho^4_{\mathsf{B}-}(\{uv\},\{uw\}) \vee \rho^4_{\mathsf{B}-}(\{vu\},\{vx\}) \vee v = w \vee u = x$
	$\rho^6_{\mathsf{B}-}(\{uw,vx\},\{uv,vu\})$	$=$	$\rho^4_{\mathsf{B}-}(\{uw\},\{uv\}) \vee \rho^4_{\mathsf{B}-}(\{vx\},\{vu\}) \vee (\pi^t_{\mathsf{B}}(u,v,w) \wedge \pi^t_{\mathsf{B}}(v,u,x)) \vee v = w \vee u = x$

Table 2: The expressions to be combined line by line as nonnegated form on the left side and negated form on the right side to test for subclass relations.

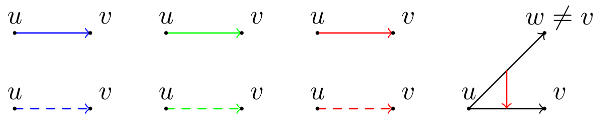


Figure 1: Illustration of the used notation. Solid blue line: $uv \in G$, dashed blue line $uv \notin G$, solid green line: $uv \in H$, dashed green line $uv \notin H$, solid red line: $\pi^i_{\mathsf{B}}(u,v)$ holds, dashed red line $\pi^e_{\mathsf{B}}(u,v)$ holds, red link from uw to uv: $\pi^t_{\mathsf{B}}(u,v,w)$ holds.

To obtain a graph H such that $H \in \gamma_1(A)$, but $H \notin \gamma_2(B)$, for the expressions listed in Table 2, $\rho^k_{\gamma_1(A)}(S^k, T^k)$ has to be non-satisfiable, but $\rho^k_{\gamma_2(B)}(S^k, T^k)$ has to be satisfiable. Hereby $G = (V, E)$ is the graph before considering the graph transformations and $\rho^k_{\mathsf{X}}(S^k, T^k)$ means that it is not possible to include all edges in S^k, while excluding all edges in T^k Consequently, $\neg\rho^k_{\mathsf{X}}(S^k, T^k)$ means that is possible to include all edges in S^k, while excluding all edges in T^k

In the following all possible counterexamples for the transformations $(\cdot)^{\mathrm{id}}$ and $(\cdot)^+$ are considered.

The proof now follows a sequence of cases. We illustrate the cases in Fig. 2a and Fig. 2b. The color coding used in those figures is defined in Fig. 1.

First the $(\cdot)^{\mathrm{id}}$ transformation is considered: here first the case that an edge uv, where the case of a loop with $u = v$ is possible, can not be included is considered. This situation occurs if the exclusion predicate $\pi^e_{\mathsf{B}}(u,v)$ is true.

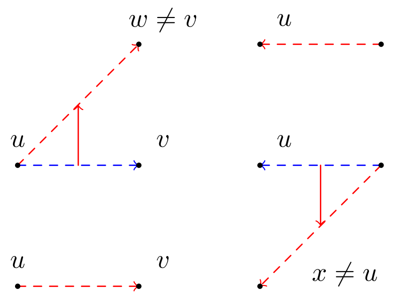
Therefore the corresponding expression in Table 2 is given by $\rho^1_{\mathsf{B}}(\{uv\},\{\emptyset\}) = \pi^e_{\mathsf{B}}(u,v)$, where $T^1 = \{\emptyset\}$ means that no edge has to be explicitly excluded.

The case that an edge uv can not be excluded is similar. Also here the case of a loop with $u = v$ is possible. An edge uv can not be included if the inclusion predicate $\pi^i_{\mathsf{B}}(u,v)$ is true.

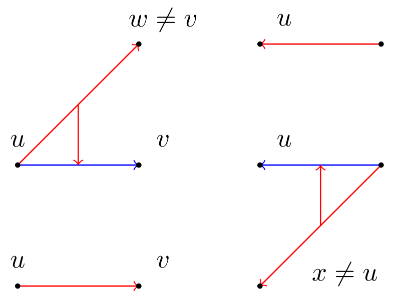
Therefore the corresponding expression in Table 2 is given by $\rho^2_{\mathsf{B}}(\{\emptyset\},\{uv\}) = \pi^i_{\mathsf{B}}(u,v)$, where $S^2 = \{\emptyset\}$ means that no edge has to be explicitly included.

For the $(\cdot)^{\mathrm{id}}$ transformation it remains the case that it is not possible to include an edge uv, while excluding an edge uw. Obviously it is not possible to include uv while excluding uw, if it is not possible to include uv or it is not possible to exclude uw. Furthermore it is not possible to include the edge uv, while excluding uw, if the transfer predicate $\pi^t_{\mathsf{B}}(u,w,v)$ is true, since the inclusion of the edge uv would lead to the inclusion of the edge uw. Also if $v = w$ excluding uv while including uw is not possible, since in this case the edge uv has to be included and excluded at the same time, which is obviously not possible. Therefore the corresponding expression in Table 2 is given by $\rho^4_{\mathsf{B}}(\{uv\},\{uw\}) = \rho^1_{\mathsf{B}}(\{uv\},\{\emptyset\}) \vee \rho^2_{\mathsf{B}}(\{\emptyset\},\{uw\}) \vee \pi^t_{\mathsf{B}}(u,v,w) \vee v = w$.

All further combinations of included and excluded edges that are not possible for the $(\cdot)^{id}$ transformation lead back to a disjunction of the considered cases.

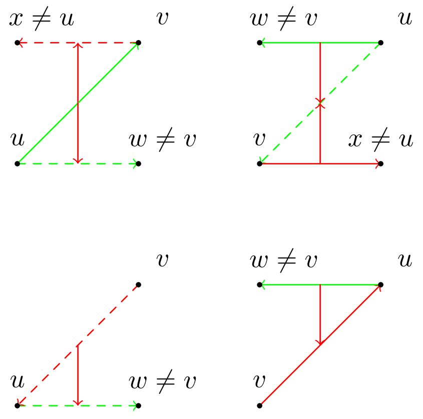


(a) $\rho^1_{\mathsf{B}}(\{uv\}, \{\emptyset\})$

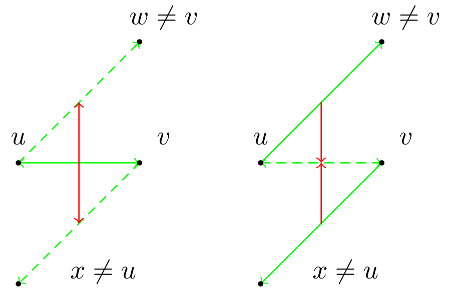


(b) $\rho^2_{\mathsf{B}}(\{\emptyset\}, \{uv\})$

Figure 2: Counterexamples for including and excluding a single edge uv.



(a) $\rho^4_{\mathsf{B}+}(\{uv\}, \{uw\})$ and $\rho^4_{\mathsf{B}-}(\{uw\}, \{uv\})$



(b) $\rho^5_{\mathsf{B}+}(\{uv, vu\}, \{uw, vx\})$ and $\rho^6_{\mathsf{B}-}(\{uw, vx\}, \{uv, vu\})$.

Figure 3: Counterexamples for including and excluding more than one edge.

Now the$(\cdot)^+$ transformation is considered: an edge uv can not be included in H if both of the edges uv and vu can not be included in G. This situation occurs if for both edges uv and vu the exclusion predicate is satisfied (see Fig. 2a on the left for the edge uv and on the right for the edge vu). Therefore, the corresponding expression in Table 2 is given by $\rho^1_{\mathsf{B}+}(\{uv\}, \{\emptyset\}) = \pi^e_{\mathsf{B}}(u,v) \wedge \pi^e_{\mathsf{B}}(v,u)$.

An edge uv can not be excluded in H if one of the edges uv and vu can not be excluded in G. This situation occurs if for one of the edges uv or vu the inclusion predicate is satisfied (see Fig. 2b on the left for the edge uv and on the right for the edge vu). Therefore, the corresponding expression in Table 2 is given by $\rho^2_{\mathsf{B}+}(\{\emptyset\}, \{uv\}) = \pi^i_{\mathsf{B}}(u,v) \vee \pi^i_{\mathsf{B}}(v,u)$. This finishes the proof for the case $(\cdot)^{\mathrm{id}}$.

The transformation $(\cdot)^+$ yields a symmetric graph, therefore including uv, while excluding vu is never possible and consequently $\rho^3_{\mathsf{B}+}(\{uv\}, \{vu\})$ is always true.

Now the case that it is not possible to include an edge uv, while excluding an edge uw is considered. Obviously it is not possible to include uv while excluding uw, if it is not possible to include uv or it is not possible to exclude uw. Furthermore, it is not possible to include the edge uv, while excluding uw, if the exclusion predicate $\pi^e_{\mathsf{B}}(v,u)$ and the transfer predicate $\pi^t_{\mathsf{B}}(u,w)$ is true, since the exclusion of the edge uv in G would lead to the exclusion of the edge uv in H and the inclusion of the edge uv in G would lead to the inclusion of the edge uw in G and therefore also the edge uw has to be included in H. Also if $v = w$, including uv while excluding uw is not possible, since in this case the edge uv has to be included and excluded at the same time, which is obviously not possible (see Fig. 3a on the left). Therefore the corresponding expression in Table 2 is given by $\rho^4_{\mathsf{B}+}(\{uv\}, \{uw\}) = \rho^1_{\mathsf{B}+}(\{uv\}, \{\emptyset\}) \vee \rho^2_{\mathsf{B}+}(\{\emptyset\}, \{uw\}) \vee (\pi^t_{\mathsf{B}}(u,w) \wedge \pi^e_{\mathsf{B}}(v,u)) \vee v = w$.

The case remains that it is not possible to exclude the edges uw and vx, while including the edge uv. Obviously this is not possible, if it is not possible to exclude the edge uw, while including the edge uv or if it is not possible to exclude the edge vx, while including the edge uv. Furthermore, it is not possible to exclude the edges uw and vx, while including the edge uv if both transfer predicates $\pi^t_{\mathsf{B}}(u,w,v)$ and $\pi^t_{\mathsf{B}}(v,x,u)$ are true, since the inclusion of the edge uv in G would lead to the inclusion of the edge uw in G and therefore also the edge uw has to be included in H and the inclusion of the edge vu in G would lead to the inclusion of the edge vx in G and therefore also the edge vx has to be included in H. So both of the edges uv and vu have to be excluded in G, but then uv has to be excluded in H. Also if $v = w$ or $u = x$ including uv while excluding uw and vx is not possible, since in this case the edge uv or the edge vu has to be included and excluded at the same time, which is obviously not possible (see Fig. 3b on the left). Therefore the corresponding expression in Table 2 is given by $\rho^5_{\mathsf{B}+}(\{uv, vu\}, \{uw, vx\}) = \rho^4_{\mathsf{B}+}(\{uv\}, \{uw\}) \vee \rho^4_{\mathsf{B}+}(\{vu\}, \{vx\}) \vee (\pi^t_{\mathsf{B}}(u, w, v) \wedge \pi^t_{\mathsf{B}}(v, x, u)) \vee v = w \vee u = x$.

All further combinations of included and excluded edges that are not possible for the $(\cdot)^+$ transformation lead back to a disjunction of the considered cases. $\square$

The expression of Theorem 6 simplifies significantly when testing $\mathsf{A}^- \not\subseteq \mathsf{A}^+$ or $\mathsf{A}^+ \not\subseteq \mathsf{A}^-$ and if inclusion and exclusion predicates are symmetric, as defined in the following corollary.

Corollary 1 *Let* A *be a graph class defined by inclusion, exclusion and transfer axioms, satisfying Assumptions 1,2 and 3. Let inclusion predicate* $\pi^i_{\mathsf{A}}(u, v)$ *and exclusion predicate* $\pi^e_{\mathsf{A}}(u, v)$ *be symmetric, i.e.,* $\pi^i_{\mathsf{A}}(u, v) = \pi^i_{\mathsf{A}}(v, u)$ *and* $\pi^e_{\mathsf{A}}(u, v) = \pi^e_{\mathsf{A}}(v, u)$ *for all* $u, v \in \Omega$. *Let furthermore* $\pi^i_{\mathsf{A}}(u, v) \wedge \pi^t_{\mathsf{A}}(u, w, v) \Rightarrow \pi^i_{\mathsf{A}}(u, w)$ *and* $\pi^e_{\mathsf{A}}(u, v) \wedge \pi^t_{\mathsf{A}}(u, v, w) \Rightarrow \pi^e_{\mathsf{A}}(u, w)$ *for all* $u, v, w \in \Omega$. *We have:* $\mathsf{A}^- \not\subseteq \mathsf{A}^+$ *iff*

$$\neg\pi^e_{\mathsf{A}}(v, w) \wedge \neg\pi^i_{\mathsf{A}}(u, v) \wedge \neg\pi^i_{\mathsf{A}}(w, x) \wedge \pi^t_{\mathsf{A}}(v, u, w) \wedge \pi^t_{\mathsf{A}}(w, x, v) \quad (13)$$

and $\mathsf{A}^+ \not\subseteq \mathsf{A}^-$ *iff*

$$\neg\pi^i_{\mathsf{A}}(v, w) \wedge \neg\pi^e_{\mathsf{A}}(u, v) \wedge \neg\pi^e_{\mathsf{A}}(w, x) \wedge \pi^t_{\mathsf{A}}(v, w, u) \wedge \pi^t_{\mathsf{A}}(w, v, x) \quad (14)$$

is satisfied for some u, v, w, x.

5 Example application

5.1 Manual application of the derived concept

We first illustrate the presented concept manually by applying it to the graph classes discussed in Section 2. We use the concept to show that the class $(\mathsf{QUDG}(r))^-$ is the same as class $(\mathsf{QUDG}(r))^+$ (as already discussed). Moreover, we apply the concept to show that the classes $(\mathsf{DTG}(r))^-$ and $(\mathsf{DTG}(r))^+$ studied in the literature are in fact classes with none being contained in the other one. Inclusion predicate $\mathsf{MinDG}(r)$ and exclusion predicate $\mathsf{MaxDG}(\mathsf{r})$ are obviously symmetric as long as $d(u, v)$ is symmetric (which we assume for this example). The class $\mathsf{QUDG}(r)$ has inclusion and exclusion but no transfer predicate, i.e., the transfer predicate is always false. Thus, for testing $(\mathsf{QUDG}(r))^- \not\subseteq (\mathsf{QUDG}(r))^+$ and $(\mathsf{QUDG}(r))^+ \not\subseteq (\mathsf{QUDG}(r))^-$ neither (13) nor (14) of Corollary 1 are satisfied. Thus, both inclusions must be satisfied and we have in fact equality of both graph classes.

Now the relation between $(\mathsf{DTG}(r))^+$ and $(\mathsf{DTG}(r))^-$ is considered. Applying Corollary 1 together with $\mathsf{DTG}(r) = \mathsf{MaxDG}(r) \cap \mathsf{CRG}$ (i.e. we have only exclusion and transfer predicate) we have to check satisfiability of

$$(\neg d(v, w) > r) \wedge (v \neq u \wedge d(v, u) \leq d(v, w)) \wedge (w \neq x \wedge d(w, x) \leq d(w, v))$$

for testing $(\mathsf{DTG}(r))^- \not\subseteq (\mathsf{DTG}(r))^+$ and

$$(\neg d(v, u) > r) \wedge (\neg d(w, x) > r) \wedge (v \neq w \wedge d(v, w) \leq d(v, u)) \wedge (w \neq v \wedge d(w, v) \leq d(w, x))$$

for testing $(\mathsf{DTG}(r))^+ \not\subseteq (\mathsf{DTG}(r))^-$.

This yields the counterexample $d(v, u)$, $d(w, x) \leq d(v, w) \leq r$, $u \neq v$, $w \neq x$, $u \neq w$ and $v \neq x$ which proves $(\mathsf{DTG}(r))^- \not\subseteq (\mathsf{DTG}(r))^+$. As well, this yields the counterexample $d(v, w) \leq d(v, u)$, $d(w, x) \leq r$, $v \neq w$, $u \neq w$ and $v \neq x$ which proves $(\mathsf{DTG}(r))^+ \not\subseteq (\mathsf{DTG}(r))^-$. Thus, neither $(\mathsf{DTG}(r))^+$ nor $(\mathsf{DTG}(r))^-$ is contained in the other class.

Now let's drop the assumption of the examples in Section 2 that $d(u, v)$ is a metric. Assume that $d(u, v) = d(v, u)$ is not required. In this case the inclusion and exclusion predicate are not necessarily symmetric. Corollary 1 is not applicable. We have to resort to Theorem 6. The relation between $(\mathsf{QUDG}(r))^-$ and $(\mathsf{QUDG}(r))^+$ is now considered again. For testing the relation $(\mathsf{QUDG}(r))^- \not\subseteq (\mathsf{QUDG})^+$ the second conjunction of expression (12) yields $\neg\rho^2_{\mathsf{A}-}(\{\emptyset\}, \{uv\}) \wedge \rho^2_{\mathsf{A}+}(\{\emptyset\}, \{uv\})$. Replacing the formula with the predicates by using Table 2 yields $\big(\neg\pi^i_{\mathsf{A}}(u, v) \vee \neg\pi^i_{\mathsf{A}}(v, u)\big) \wedge \big(\pi^i_{\mathsf{A}}(u, v) \vee \pi^i_{\mathsf{A}}(v, u)\big) = \big(\neg\pi^i_{\mathsf{A}}(u, v) \wedge \pi^i_{\mathsf{A}}(v, u)\big) \vee \big(\pi^i_{\mathsf{A}}(u, v) \wedge \neg\pi^i_{\mathsf{A}}(v, u)\big)$.

This formula is satisfied if $d(u, v) < r$ and $d(v, u) \geq r$ or $d(u, v) \geq r$ and $d(v, u) < r$. Thus, we have found a counter example that the relation $(\mathsf{QUDG}(r))^- = (\mathsf{QUDG}(r))^+$ does not necessarily hold when $d(u, v)$ is not a metric.

5.2 Automatically derived graph class relations

The following Table 3 shows the automatically derived relations between all example geometric graph classes resulting from wireless networks which we defined in Section 2. To avoid clutter in that table we depict the graph classes without the fixed parameters r and 1.

The table is to be read from left to right as follows. Class in row i is related to class in column j according to table entry (i, j) by relation:

$=$	the classes are the same
$\subset$	the class is contained in the other class but the classes are not the same
$\supset$	the class contains the other class but the classes are not the same

$\times$ neither of the two classes is contained in the other[3]

The class name encoding in the table is as defined at the beginning of the paper, summarized by the following Hasse diagram.

$$\begin{tikzcd}
 & \mathsf{All} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\mathsf{MinDG}(r) \arrow[d, dash] \arrow[dr, dash] & \mathsf{MaxDG}(1) \arrow[dl, dash] \arrow[dr, dash] & \mathsf{CRG} \arrow[dl, dash] \arrow[d, dash] \\
\mathsf{QUDG}(r) \arrow[dr, dash] & \mathsf{BG}(r) \arrow[d, dash] & \mathsf{DTG}(1) \arrow[dl, dash] \\
 & \mathsf{MDTG}(r) &
\end{tikzcd}$$

Figure 4: The relations of the base graph classes.

Class All (not defined so far) means all of the predicates (i.e. inclusion, exclusion, and transfer) are false. With all preconditions being false, all graphs satisfy inclusion, exclusion, and transfer predicate. Thus, the class just contains all graphs over the given vertex universe Ω.

Classes $\mathsf{MinDG}(r)$, $\mathsf{MaxDG}(1)$ and CRG are as already defined based only on inclusion, exclusion and transfer predicate, respectively. All remaining classes result from the intersection of the classes which are predecessors in the diagram (seen from top to bottom).

The class names $\mathsf{BG}(r)$ and $\mathsf{MDTG}(r)$ were not used before. We define them here for the sake of completeness of the tables. For graphs in class $\mathsf{BG}(r)$ (which we term *boost graphs*) an arbitrary long link can be added to a vertex, which boosts connectivity of that vertex. It will be connected to all other vertices with the same or less distance. The graphs in class $\mathsf{MDTG}(r)$ (which we term *minimum radius DTGs*) are a DTG but also with a minimum connectivity radius where vertices are always connected.

Opposed to the ususal assumption that $\mathsf{QUDG}(r)$ actually stands for a symmetric subgraph/supergraph of $\mathsf{MinDG}(r) \cap \mathsf{MaxDG}(1)$ we denote it here just as $\mathsf{MinDG}(r) \cap \mathsf{MaxDG}(1)$ to stay consistent with the notation used in the table. Class $\mathsf{QUDG}(r)$ as unusually used in the literature can be found under $\mathsf{QUDG}^-/\mathsf{QUDG}^+$ in the table.

All relations in Table 3 were automatically determined within approximately 120 seconds running *Mathematica 12.1* on an Intel Core i7-10610U CPU @ 1.80 GHz with 16 GBytes DDR3 RAM. The relations were computed applying `FindInstance` on the derived expression (11) using the Euclidean metric.

In connection with the relations of the base classes shown in the Hasse diagram in Fig.4, the following observations can be made for the derived table.

In the upper left of Table 3, the row for All is the row for MDTG but read reversed from right to left instead of left to right. The same applies for the row pairs ($\mathsf{MinDG}, \mathsf{DTG}$), ($\mathsf{MaxDG}$, BG) and (CRG, QUDG) (which are not directly connected in the Hasse diagram).

Similar, in the middle and the lower right of Table 3 the same row pairs with one being the reversed variant of the other can be seen for transformation $(\cdot)^-$ and $(\cdot)^+$, respectively.

Moreover, the transformations $(\cdot)^-$ and $(\cdot)^+$ yield obviously the same graph class when applied on All. The same applies to QUDG, as already discussed. As well, applied on MinDG and as well on MaxDG, transformations $(\cdot)^-$ and $(\cdot)^+$ yield the same classes, respectively.

These base classes (All, MaxDG, MinDG and QUDG) are as well exactly those classes whose symmetric variants due to $(\cdot)^-$ and $(\cdot)^+$ are contained in the base class. For the other classes neither subset relation nor superset relation is satisfied between base classes and their symmetric variants.

Both transformations $(\cdot)^-$ and $(\cdot)^+$ obviously preserve the subset relation order of the base classes. Moreover, neither $(\cdot)^-$ nor $(\cdot)^+$ yield two base classes collapsing into one single class.

Finally, Table 3 is trivially a symmetric matrix (however, with reversed relations $\subset$ and $\supset$) and its diagonals are $=$.

6 Conclusion

In this paper we described how to prove or disprove containedness relations of specific axiomatic described geometric graph classes on a meta-level. We proved correctness of general logical existentially quantified expressions over placeholders for inclusion, exclusion and transitive transfer predicates. We illustrated the concept with concrete predicates in the context of simple theoretical wireless network models.

Though we focused on geometric classes in this work, by its generality, the concept is also applicable to any graph class which can be desribed by means of inclusion, exclusion and transfer axioms and simple graph transformations.

The decidability of the problem of testing containedness for concrete inclusion, exclusion and transfer predicates depends on the concrete theories we consider. These are typically extensions of the theory of real numbers with norms, distances, cost functions or more general function symbols. In ongoing work we are investigating the decidability of such theories.

Though a logical representation of graphs is not new in general, we believe with the axiomatic graph class representation and the discussed simple transformations

[3]By lemma 1 and 2 each of the studied graph classes contain at least the null graph. Thus, none of the considered graph classes are disjoint.

$\rightarrow$	All	MinDG	MaxDG	CRG	QUDG	BG	DTG	MDTG	All$^-$	MinDG$^-$	MaxDG$^-$	CRG$^-$	QUDG$^-$	BG$^-$	DTG$^-$	MDTG$^-$	All$^+$	MinDG$^+$	MaxDG$^+$	CRG$^+$	QUDG$^+$	BG$^+$	DTG$^+$	MDTG$^+$
All	=	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃	⊃
MinDG	⊂	=	×	×	⊃	⊃	×	⊃	×	⊃	×	×	⊃	⊃	×	⊃	×	⊃	×	×	⊃	⊃	×	⊃
MaxDG	⊂	×	=	×	⊃	×	⊃	⊃	×	×	⊃	×	⊃	×	⊃	⊃	×	×	⊃	×	⊃	×	⊃	⊃
CRG	⊂	×	×	=	×	⊃	⊃	⊃	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
QUDG	⊂	⊂	⊂	×	=	×	×	⊃	×	×	×	×	⊃	×	×	⊃	×	×	×	×	⊃	×	×	⊃
BG	⊂	⊂	×	⊂	×	=	×	⊃	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
DTG	⊂	×	⊂	⊂	×	×	=	⊃	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
MDTG	⊂	⊂	⊂	⊂	⊂	⊂	⊂	=	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
All$^-$	⊂	×	×	×	×	×	×	×	=	⊃	⊃	⊃	⊃	⊃	⊃	⊃	=	⊃	⊃	⊃	⊃	⊃	⊃	⊃
MinDG$^-$	⊂	⊂	×	×	×	×	×	×	⊂	=	×	×	⊃	⊃	×	⊃	⊂	=	×	×	⊃	⊃	×	⊃
MaxDG$^-$	⊂	×	⊂	×	×	×	×	×	⊂	×	=	×	⊃	×	⊃	⊃	⊂	×	=	×	⊃	×	⊃	⊃
CRG$^-$	⊂	×	×	×	×	×	×	×	⊂	×	×	=	×	⊃	⊃	⊃	⊂	×	×	×	×	×	×	×
QUDG$^-$	⊂	⊂	⊂	×	⊂	×	×	×	⊂	⊂	⊂	×	=	×	×	⊃	⊂	⊂	⊂	×	=	×	×	⊃
BG$^-$	⊂	⊂	×	×	×	×	×	×	⊂	⊂	×	⊂	×	=	×	⊃	⊂	⊂	×	×	×	×	×	×
DTG$^-$	⊂	×	⊂	×	×	×	×	×	⊂	×	⊂	⊂	×	×	=	⊃	⊂	×	⊂	×	×	×	×	×
MDTG$^-$	⊂	⊂	⊂	×	⊂	×	×	×	⊂	⊂	⊂	⊂	⊂	⊂	⊂	=	⊂	⊂	⊂	×	⊂	×	×	×
All$^+$	⊂	×	×	×	×	×	×	×	=	⊃	⊃	⊃	⊃	⊃	⊃	⊃	=	⊃	⊃	⊃	⊃	⊃	⊃	⊃
MinDG$^+$	⊂	⊂	×	×	×	×	×	×	⊂	=	×	×	⊃	⊃	×	⊃	⊂	=	×	×	⊃	⊃	×	⊃
MaxDG$^+$	⊂	×	⊂	×	×	×	×	×	⊂	×	=	×	⊃	×	⊃	⊃	⊂	×	=	×	⊃	×	⊃	⊃
CRG$^+$	⊂	×	×	×	×	×	×	×	⊂	×	×	×	×	×	×	×	⊂	×	×	=	×	⊃	⊃	⊃
QUDG$^+$	⊂	⊂	⊂	×	⊂	×	×	×	⊂	⊂	⊂	×	=	×	×	⊃	⊂	⊂	⊂	×	=	×	×	⊃
BG$^+$	⊂	⊂	×	×	×	×	×	×	⊂	⊂	×	×	×	×	×	×	⊂	⊂	×	⊂	×	=	×	⊃
DTG$^+$	⊂	×	⊂	×	×	×	×	×	⊂	×	⊂	×	×	×	×	×	⊂	×	⊂	⊂	×	×	=	⊃
MDTG$^+$	⊂	⊂	⊂	×	⊂	×	×	×	⊂	⊂	⊂	×	⊂	×	×	×	⊂	⊂	⊂	⊂	⊂	⊂	⊂	=

Table 3: Relations between all studied graph classes under transformation $(\cdot)^{\mathrm{id}}$, $(\cdot)^-$, and $(\cdot)^+$.

described in this work, we discovered a promising graph theoretical novel concept for containedness verification which can be generalized in many further directions.

Future work includes extending the proofs on minimal witness graphs and the logical expressions to other simple graph transformations. Furthermore, the theory could be extended beyond simple graph transformations. Moreover, one can drop the condition on transfer predicates to be transitive or even further that the edges involved don't have to be originated at the two vertices the transfer predicate is applied to.

Acknowledgements

We would like to thank for all discussions with Viorica Sofronie-Stokkermans providing us with background information on the general theory of formal verification.

References

[1] L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierarchic first-order theories. *Appl. Algebra Eng. Commun. Comput.*, 5:193–212, 1994.

[2] L. Barrière, P. Fraigniaud, L. Narayanan, and J. Opatrny. Robust position-based routing in wireless ad hoc networks with irregular transmission ranges. *Wireless Communications and Mobile Computing*, 3(2):141–153, Mar 2003.

[3] D. M. D. Blough, M. Leoncini, G. Resta, and P. Santi. The k-neighbors approach to interference bounded and symmetric topology control in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(9):1267–1282, Sep 2006.

[4] D. M. Gabbay, R. A. Schmidt, and A. Szalas. *Second-Order Quantifier Elimination - Foundations, Computational Aspects and Applications*, volume 12 of *Studies in logic: Mathematical logic and foundations*. College Publications, 2008.

[5] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad hoc networks beyond unit disk graphs. *Wireless Networks*, 14(5):715–729, Oct 2008.

[6] K. Moaveninejad, W.-Z. Song, and X.-Y. Li. Robust position-based routing for wireless ad hoc networks. *Ad Hoc Networks*, 3(5):546–559, Sep 2005.

[7] D. Peleg and L. Roditty. Localized spanner construction for ad hoc networks with variable transmission range. *ACM Transactions on Sensor Networks*, 7(3):1–14, Oct 2010.

[8] P. von Rickenbach, R. Wattenhofer, and A. Zollinger. Algorithmic models of interference in wireless ad hoc and sensor networks. *IEEE/ACM Transactions on Networking*, 17(1):172–185, Feb 2009.

Algorithms for Covering Barrier Points by Mobile Sensors with Line Constraint[*]

Princy Jain[†] Haitao Wang[‡]

Abstract

We study the problem of covering barrier points by mobile sensors. Each sensor is represented by a point in the plane with the same covering range r so that any point within distance r from the sensor can be covered by the sensor. Given a set B of m points (called "barrier points") and a set S of n points (representing the "sensors") in the plane, the problem is to move the sensors so that each barrier point is covered by at least one sensor and the maximum movement of all sensors is minimized. The problem is NP-hard. In this paper, we consider two line-constrained variations of the problem and present efficient algorithms that improve the previous work. In the first problem, all sensors are given on a line ℓ and are required to move on ℓ only while the barrier points can be anywhere in the plane. We propose an $O((n+m)\log(n+m))$ time algorithm for the problem. We also consider the weighted case where each sensor has a weight; we give an $O((n+m)\log^2(n+m))$ time algorithm for this case. In the second problem, all barrier points are on ℓ while all sensors are in the plane but are required to move to ℓ to cover all barrier points. We solve the weighted case in $O(m\log m + n\log^2 n)$ time.

1 Introduction

Let B be a set of m points and D be a set of n disks of the same radius r in the plane. We consider the problem of moving the disks of D to cover all points of B so that the maximum moving distance of all disks is minimized. The problem is NP-hard.[1] In this paper, we consider two line-constrained variations of the problem and present efficient algorithms for them.

Due to its potential applications in barrier coverage of mobile sensors in wireless sensor networks [14, 15, 17], we consider the problem from the barrier coverage point of view. We call the points of B the *barrier points*. Let S be the set of centers of all disks of D, and points of S are called *sensors*. All sensors have the same *covering range* (or *sensing range*) r so that any point within distance r from a sensor s can be covered by s (i.e., s covers all points in the disk centered at s with radius r). Hence, our problem becomes the following: move sensors of S to cover all barrier points of B such that the maximum moving distance of all sensors is minimized.

We study a line-constrained variation of the problem where all sensors are given on a line ℓ and are required to move on ℓ only while the barrier points can be anywhere in the plane. We also consider its *weighted case* where each sensor s_i has a weight $w_i > 0$ and the *moving cost* of s_i is defined to be its moving distance times w_i.

To the best of our knowledge, we are not aware of any previous work on this particular problem. If all barrier points are all on ℓ, which becomes a 1D problem (our original problem can be considered as a 1.5D problem), the algorithm of Li and Wang [18] can solve the unweighted case in $O(m\log m + n\log m\log n)$ time. In this paper, we present an $O((n+m)\log(n+m))$ time for the unweighted case and an $O((n+m)\log^2(n+m))$ time algorithm for the weighted case. Hence, our algorithm for the unweighted case, albeit solving the 1.5D problem, improves the algorithm of [18] by roughly a logarithmic factor.

We also consider another problem variation in which all barrier points are on a line ℓ while sensors can be anywhere in the plane. We want to move all sensors to ℓ to cover all barrier points so that the maximum moving cost of all sensors is minimized. Previously, Huang et al. [14] studied the unweighted case and gave an $O(n(m + n\log n)\log(n+m))$ time algorithm. Our techniques solve the weighted case in $O(m\log m + n\log^2 n)$ time. This improves the algorithm of Huang et al. [14] by almost a linear factor. Note that we do not have a faster algorithm for the unweighted case. As all barrier points are on ℓ and all sensors will finally move to ℓ, once a sensor s moves to ℓ, the portion of the covering disk of s that is relevant is an interval of ℓ. For this reason, we refer to this problem as the *mobile interval coverage* problem; for differentiation, we refer to the first problem above as the *mobile disk coverage problem.* Note that if sensors have different ranges, even the 1D problem (i.e., all sensors and barrier points are on ℓ) is NP-hard [14].

[*]This research was supported in part by NSF under Grant CCF-2005323. A full version can be found in the first author's thesis, available at `https://digitalcommons.usu.edu/etd/8130/`.

[†]Department of Computer Science, Utah State University, Logan, UT 84322, USA, `princy.jain@usu.edu`

[‡]Department of Computer Science, Utah State University, Logan, UT 84322, USA, `haitao.wang@usu.edu`

[1]This can be proved by an easy reduction from the minimum disk coverage problem [13]; e.g., see [19] for a reduction for a similar problem.

1.1 Related work

Many variations of mobile sensor barrier coverage problem have been studied in the literature.

Czyzowicz et al. [6] studied the problem of covering a barrier segment on a line ℓ by moving a set of n sensors on ℓ (the sensors are initially given on ℓ); they gave an $O(n^2)$ time algorithm. Chen et al. [3] presented a more efficient $O(n \log n)$ time algorithm. Chen et al. [3] also studied the case where sensors may have different covering ranges and proposed an $O(n^2 \log n)$ time algorithm. For the weighted case where the sensors have weights as defined in our problems (but sensors have the same range), Lee et al. [16] derived an algorithm of $O(n^2 \log n \log \log n)$ time.

Li and Shen [17] studied the same problem as our interval coverage problem except that their barrier is not a set of points but a single line segment on ℓ. They proposed an $O(n^3 \log n)$ time algorithm. The algorithm was later improved to $O(n^2 \log n \log \log n)$ time by Li and Wang [18]. Li and Wang [18] also studied a more general problem setting where the barrier is a set of disjoint line segments on ℓ (and the sensors are still in the plane and are required to move to ℓ); they gave an $O(n^2 \log n \log \log n + nm \log m)$ time algorithm. Further, for the 1D case where all sensors are initially on ℓ, the algorithm of Li and Wang [18] solves the problem in $O(m \log m + n \log n \log m)$ time. These results are for the case where sensors have the same range; if sensors have different ranges, even the 1D problem is NP-hard by a reduction from the Partition Problem as in [6].

The min-sum version of the line-constrained barrier coverage was also studied in the literature where sensors are given on ℓ and a barrier segment is also on ℓ, and the goal is to move sensors to cover the barrier segment such that the total sum of the moving distances of all sensors is minimized. If sensors have different ranges, the problem is NP-hard [7]. Otherwise, Czyzowicz et al. [7] solved the problem in $O(n^2)$ time. Later Andrews and Wang [1] proposed a faster algorithm of $O(n \log n)$ time.

A circular barrier coverage problem was also considered, where the barrier is a circle and sensors are initially located inside the circle and the goal is to move all sensors to the circle to form a regular n-gon (to form a coverage) so that the maximum moving distance of all sensors is minimized. Bhattacharya [2] first gave an algorithm of $O(n^{3.5} \log n)$ time. An improved algorithm of $O(n \log^3 n)$ time was later derived by Chen et al. [4].

There are also other variations of the barrier coverage problem, e.g., see [8, 9, 20, 21].

1.2 Our approach

We first discuss the mobile disk coverage problem. Let λ^* denote the optimal moving cost, i.e., the maximum moving cost of all sensors in an optimal solution. In both the unweighted and weighted cases, we first consider the *decision problem*: Given any value λ, determine whether $\lambda \geq \lambda^*$.

For the unweighted case, a critical property is an *order-preserving property*: There exists an optimal solution in which the order of the sensors are consistent with their order in the input. Due to the property, we can solve the decision problem in linear time by a simple greedy algorithm (after all barrier points and all sensors are sorted). Next, we use the decision algorithm to compute λ^*. To this end, we define $2m$ arrays of size n each and we show that λ^* must be an element of one of the arrays. To search λ^* in these arrays in an efficient way, we form these arrays implicitly. A helpful observation is that each of these arrays is sorted. Consequently, by using our decision algorithm, we apply a *sorted matrix searching* technique [10, 11, 12] (or a simpler implementation called *binary search on sorted arrays* in [5]) to find λ^* in these arrays in $O((n+m)\log(n+m))$ time.

For the weighted case, unfortunately the order-preserving property does not hold anymore. In fact, the major difficulty is to find the correct order for sensors in an optimal solution. This is also the case for solving the decision problem. So we have to use a different approach to solve the decision problem. The runtime of the algorithm is $O((n+m)\log(n+m))$. To compute the optimal cost λ^*, we implicitly form $2n$ arrays of size m each such that λ^* is one of the array elements. To apply the sorted matrix searching technique, we manage to find a way to order the array elements implicitly so that the arrays are still sorted. Then, with the decision algorithm, the value λ^* can be found in $O((n+m)\log^2(n+m))$ time.

For the mobile interval coverage problem, we solve the weighted cases directly (without having a faster algorithm for the unweighted case). As above, we also solve the decision problem first, and then form sorted arrays and apply the sorted array searching technique. To solve the decision algorithm, we use an algorithm similar to the weighted case of the above mobile disk coverage problem, but with a simpler and slightly faster implementation. The runtime of our decision algorithm is $O(m + n \log n)$ after $O((n+m)\log(n+m))$ time preprocessing for sorting all sensors and barrier points. The time of the overall algorithm (for computing the optimal value λ^*) is $O(m \log m + n \log^2 n)$.

Outline. The rest of the paper is organized as follows. We define notation in Section 2. In Section 3, we present our algorithm for the unweighted case of the mobile disk coverage problem, while the weighted case is discussed in Section 4. The algorithm for the mobile interval coverage is described in Section 5.

2 Preliminaries

For each problem we consider, we use λ^* to denote the optimal moving cost. Given any λ, the *decision problem* is to decide whether $\lambda \geq \lambda^*$, i.e., whether it is possible to move sensors to cover all barrier points so that the moving cost of each sensor is at most λ. If $\lambda \geq \lambda^*$, we say that λ is a *feasible value*. We use *feasibility test* to refer to the procedure for determining whether $\lambda \geq \lambda^*$. For differentiation, we refer to our original problem for computing λ^* as the *optimization problem*.

Without loss of generality, we assume that the line ℓ is the x-axis. Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of sensors (unless otherwise stated, the order is arbitrary). For each s_i, we use (x_i, y_i) to denote its coordinate in the input. For differentiation, for each barrier point $b \in B$, we use (x_b, y_b) to denote its coordinate.

In each problem, we use a *configuration* to refer to a specification on where each sensor s_i is located. For example, in the input configuration, each sensor s_i is at (x_i, y_i).

For each sensor s, we use $D(s)$ to refer to its covering disk, i.e., the disk of radius r centered at s. For any disk D, we use ∂D to denote its boundary, which is a circle. The *left half-circle* of ∂D refers to the portion of ∂D to the left of the vertical line through the center of D; the *right half-circle* is defined similarly.

For the mobile disk coverage problem, for simplicity of discussion, we assume that all barrier points above or on ℓ since if a barrier point is below ℓ, then we can use its symmetric point about ℓ to replace it and that does not affect the solution of the problem.

For any point p on ℓ, for convenience, sometimes we also use p to refer to its x-coordinate. For example, for two points p and q on ℓ, $p \leq q$ means that p is to the left of q (including the case where p and q are coincident) and $p < q$ means that p is *strictly* to the left of q.

For each problem, for ease of exposition, we assume that it is always possible to cover all barrier points by moving sensors (i.e., the covering range r is big enough). Our algorithm can actually determine whether the assumption is true or not. This implies that in the mobile disk coverage problem, for each barrier point b, $y_b \leq r$ must hold since otherwise no sensor on ℓ can cover b. Also, for each problem we assume that $\lambda^* > 0$, i.e., one has to move at least one sensor in order to form a coverage for all barrier points. Note that whether $\lambda^* = 0$ can be easily determined in $O(n+m)\log(n+m)$ time for each problem (which does not affect the time complexity of the overall algorithm asymptotically).

For a barrier point b and the covering disk $D(s)$ of a sensor s, we say that $D(s)$ is *strictly to the left (resp., right)* of b if $D(s)$ does not cover b and the intersection between $D(s)$ and the horizontal line through b is strictly to the left (resp., right) of b.

3 The mobile disk coverage problem: the unweighted case

In this section, we consider the unweighted case of the mobile disk coverage problem. In this problem, all sensors of S are on the line ℓ while each barrier of B can be anywhere in the plane.

We first present an algorithm to solve the decision algorithm. Consider a value λ. If $\lambda \geq \lambda^*$, we use a *feasible solution* to refer to a configuration in which all barrier points are covered and the moving cost of each sensor is no more than λ. As all sensors have the same range, it is not difficult to see that *the order-preserving property* in the following observation holds.

Observation 1 (The order-preserving property) *If $\lambda \geq \lambda^*$, then there exists a feasible solution in which the order of sensors is the same as in the input.*

Due to the order-preserving property, we can solve the decision problem by a simple greedy algorithm in linear time (after sensors and barrier points are sorted).

Lemma 1 *After $O(n \log n + m \log m)$ time preprocessing, given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O(n+m)$ time.*

Proof. In the preprocessing, we sort all sensors of S from left to right on ℓ; let $S = \{s_1, s_2, \ldots, s_n\}$ be the sorted list. We also sort all barrier points of B by their x-coordinates from left to right; let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list. Given any λ, in what follows we describe our $O(n+m)$ time algorithm for deciding whether $\lambda \geq \lambda^*$, which is based on the greedy strategy.

We first move each sensor rightwards on ℓ by distance λ and we use C_0 to refer to the configuration, i.e., in C_0, the location of each s_i is $x_i + \lambda$. Then, during the algorithm, each sensor will not be allowed to move rightwards anymore but can move leftwards by 2λ.

Starting from $i = 1$ and $j = 1$, we process sensors s_i and barrier points b_j incrementally. We first check whether b_j is covered by s_i. If yes, we increase j by one (if $j = m$ before the increase, then all barrier points are covered and we have found a feasible solution; in this case, we can stop the algorithm and report that λ is a feasible value, i.e., $\lambda \geq \lambda^*$). Otherwise, either b_j is to the right of the covering disk $D(s_i)$ of s_i or b_j is to the left of $D(s_i)$. In the former case, we increase i by one and proceed as above (if $i = n$ before the increase, then we can stop the algorithm and report that λ is not a feasible value, i.e., $\lambda < \lambda^*$). In the latter case, we check whether it is possible to move s_i leftwards by distance at most 2λ to cover b_j. If not, then we can stop the algorithm and report that λ is not a feasible value. Otherwise, we move s_i leftwards until b_j is covered (i.e., b_j is on the left half-circle of $\partial D(s_i)$); we then increase j by one and proceed as above (if $j = m$ before the increase, then all barrier points are covered and thus we can stop the algorithm and report that λ is a feasible value). This finishes the description of the algorithm.

The correctness of the algorithm is based on the order-preserving property. It is not difficult to see that the running time of the algorithm is $O(n+m)$. $\square$

We next tackle the optimization problem for computing λ^*, by making use of our decision algorithm in Lemma 1 as a subroutine. For this, we have the following lemma.

Lemma 2 *λ^* is equal to $x_i - \sqrt{r^2 - y_b^2} - x_b$ or $x_b - \sqrt{r^2 - y_b^2} - x_i$ for a sensor s_i and a barrier point b.*

Proof. Consider an optimal solution OPT, where λ^* is the maximum moving distance of all sensors. Then, λ^* is equal to the moving distance of a sensor s_i. Let x_i' be the position of s_i in OPT. If $x_i' < x_i$, then s_i has been moved leftwards. In this case, there must be a barrier point b on the left half-circle of $\partial D(s_i)$ since otherwise we could move $D(s_i)$ rightwards slightly so that $D(s_i)$ still covers the same set of barrier points as before but the moving distance of s_i is strictly smaller than λ^*, a contradiction to the definition of λ^*. Thus, we have $x_i' = \sqrt{r^2 - y_b^2} + x_b$. Hence, $\lambda^* = x_i - x_i' = x_i - \sqrt{r^2 - y_b^2} - x_b$. If $x_i' > x_i$, then by similar analysis as above, we can show that $\lambda^* = x_b - \sqrt{r^2 - y_b^2} - x_i$. $\square$

We sort all sensors of S from left to right on ℓ; let $S = \{s_1, s_2, \ldots, s_n\}$ be the sorted list. For each barrier point $b \in B$, we define two arrays $A_b[1 \cdots n]$ and $A_b'[1 \cdots n]$ of size n each as follows: For each $i \in [1, n]$, define $A_b[i] = x_i - \sqrt{r^2 - y_b^2} - x_b$ and $A_b'[i] = x_b - \sqrt{r^2 - y_b^2} - x_i$. According to Lemma 2, λ^* is an element in one of the $2m$ arrays A_b and A_b' for all $b \in B$. We next find λ^* in these arrays. Computing these arrays explicitly will take $\Omega(nm)$ time. Below, we present a near linear time algorithm without computing these arrays explicitly. Indeed, given an index $i \in [1, n]$ and a barrier point $b \in B$, we can obtain the values $A_b[i]$ and $A_b'[i]$ in constant time.

An easy observation is that elements of the array A_b are sorted in ascending order and elements of A_b' are sorted in descending order. Therefore, we are searching λ^* in $2m$ sorted arrays of size n each. Note that λ^* is actually the smallest feasible value in these $2m$ arrays. We can use the sorted matrix searching techniques [10, 11, 12] (or a simpler implementation, called *binary search on sorted arrays*, in [5]) to search sorted arrays with the following lemma.

Lemma 3 [5, 10, 11, 12] *Suppose we have a set of M sorted arrays of size at most N each such that each array element can be evaluated in $O(1)$ time (i.e., given the index of an array, the element of the array can be obtained in $O(1)$ time). Then, the smallest feasible value in these arrays can be computed by $O(\log(N+M))$ feasibility tests and the total time of the algorithm excluding the feasibility tests is $O(M \log N)$.*

Applying Lemma 3 and using our decision algorithm in Lemma 1, λ^* can be found in $O((n+m)\log(n+m))$ time. We summarize our result in the following theorem.

Theorem 4 *Given a set of m barrier points in the plane and a set of n sensors on a line ℓ, the problem of moving sensors on ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O((n+m)\log(n+m))$ time.*

4 The mobile disk coverage problem: the weighted case

In this section, we solve the weighted case of the mobile disk coverage problem. Here also, we start with the decision problem and later solve the optimization problem by applying sorted array searching techniques in Lemma 3. In the weighted case, each sensor s_i is associated with a weight $w_i > 0$.

4.1 The decision problem

Given any λ, the problem is to decide whether $\lambda \geq \lambda^*$. Although our algorithm is similar in spirit to those in the previous work [3, 16, 18], our algorithm is for a more general problem setting in that the barrier points are in the plane while the barriers in all previous work [3, 16, 18] are on ℓ. In the following, we first describe our algorithm, and then prove its correctness; finally, we will discuss how to efficiently implement the algorithm in $O((n+m)\log(n+m))$ time.

4.1.1 The algorithm description

For each sensor s_i, define $x_i^l = x_i - \lambda/w_i$ and $x_i^r = x_i + \lambda/w_i$, i.e., x_i^l is the leftmost location on ℓ where s_i can move to and x_i^r is the rightmost location on ℓ where s_i can move to with respect to λ. We call x_i^l (resp., x_i^r) the *leftmost (resp., rightmost) λ-reachable location.*

For each barrier point b, we use $c(b)$ to denote the center of the circle of radius r whose center is at ℓ and whose left half-circle contains b, i.e., $c(b) = x_b + \sqrt{r^2 - y_b^2}$. We sort all barrier points $b \in B$ in the order of the values $c(b)$. Alternatively, it is also the order of the barrier points of B encountered by sweeping a left half-circle centered at ℓ from left to right. Let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list.

Initially, we move each sensor s_i to x_i^r and thus s_i will not be allowed to move rightwards anymore but can move leftwards by $2\lambda/w_i$. Let C_0 denote the resulting configuration. If $\lambda \geq \lambda^*$, our algorithm will find a subset of sensors with their new locations such that all barrier points are covered and the maximum moving cost of each sensor is at most λ (sensors not in the subset are still in their positions of C_0).

Consider the i-th iteration of the algorithm (initially, $i = 1$). Let C_{i-1} be the configuration right before the iteration. Our algorithm maintains the following invariants.

1. A subset of sensors $S_{i-1} = \{s_{g_1}, \ldots, s_{g_{i-1}}\}$ has been computed, where g_j is the index of the sensor s_{g_j} for each $j \in [1, i-1]$.

2. In C_{i-1}, each sensor s_k of S_{i-1} is at a location, denoted by x'_k, which may not be equal to x^r_k, while sensors of $S \setminus S_{i-1}$ are still in their locations of C_0 (i.e., each sensor of $S \setminus S_{i-1}$ is at its rightmost λ-reachable location).

3. An index h_{i-1} of a barrier point is maintained such that in the configuration C_{i-1}, the barrier point $b_{h_{i-1}}$ is not covered by any sensor of S_{i-1} while b_k is covered by a sensor in S_{i-1} for each $k < h_{i-1}$ (note that it is possible that b_k for some $k > h_{i-1}$ is also covered by a sensor in S_{i-1}, which cannot happen in the problem settings of the previous work [3, 16, 18]; this case makes our problem more challenging to solve).

4. Each sensor of S_{i-1} covers at least one barrier point b_j with $j < h_{i-1}$ in C_{i-1}.

5. The locations of the sensors $s_{g_1}, s_{g_2}, \ldots, s_{g_{i-1}}$ in C_{i-1} are sorted from left to right on ℓ.

6. The barrier point $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$ of $s_{g_{i-1}}$ if $S_{i-1} \neq \emptyset$.

Initially when $i = 1$, we have $S_0 = \emptyset$ and we set $h_0 = 1$; thus, all algorithm invariants trivially hold. The i-th iteration of the algorithm finds a sensor s_{g_i} from $S \setminus S_{i-1}$ and move it to a new location $x'_{g_i} \in [x^l_{g_i}, x^r_{g_i}]$ to obtain a new configuration C_i with $S_i = S_{i-1} \cup \{s_{g_i}\}$. The details of the i-th iteration of the algorithm are described below.

Define S_{i1} to be the set of sensors that cover the barrier point $b_{h_{i-1}}$ in the configuration C_{i-1}. According to our algorithm invariants, $b_{h_{i-1}}$ is not covered by any sensor in S_{i-1}. Hence, $S_{i1} \subseteq S \setminus S_{i-1}$.

If $S_{i1} \neq \emptyset$, we pick an arbitrary sensor from S_{i1} as s_{g_i} and set $x'_{g_i} = x^r_{g_i}$ (i.e., the sensor does not move from its position in C_{i-1}); thus $C_i = C_{i-1}$. We set $h_i = k+1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = n+1$, all barrier points b_j for all $j \in [h_{i-1}, n]$ are covered, and thus we can stop the algorithm and report $\lambda \geq \lambda^*$.

Lemma 5 *All algorithm invariants hold.*

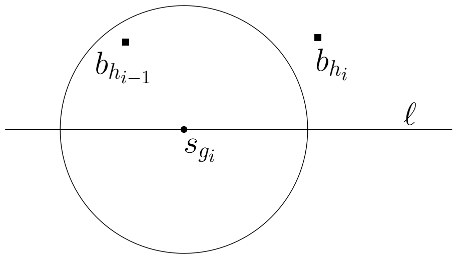


Figure 1: Illustrating the Invariant (6) in the proof of Lemma 5: the circle is the boundary of $D(s_{g_i})$.

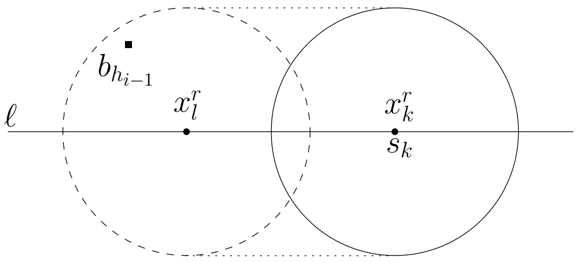


Figure 2: Illustrating the definition of S_{i2}: The solid circle shows the position of s_k in C_{i-1}, i.e., at x^r_k, and the dashed circle shows its leftmost λ-reachable location, i.e., x^l_k.

Proof. We go through every invariant. Invariant (1) trivially holds. Invariant (2) holds because $C_i = C_{i-1}$. Invariant (3) follows immediately from how our algorithm computes h_i. Invariant (4) holds because s_{g_i} covers $b_{h_{i-1}}$ in C_i. For Invariant (5), it suffices to show that $s_{g_{i-1}}$ is to the left of the s_{g_i} in C_i. Indeed, according to Invariant (6) in C_{i-1}, $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$. Since $b_{h_{i-1}}$ is covered by s_{g_i} in C_i, we obtain that $s_{g_{i-1}}$ must be to the left of s_{g_i} in C_i. For Invariant (6), since the sensor s_{g_i} covers $b_{h_{i-1}}$ but does not cover b_{h_i} and $h_{i-1} < h_i$, according to the definition of the indices of the barrier points, we can obtain that b_{h_i} must be strictly to the right of the covering disk $D(s_{g_i})$ of s_{g_i} (e.g., see Fig. 1). This proves Invariant (6). $\square$

If $S_{i1} = \emptyset$, we define $S_{i2} = \{s_k \mid x^l_k \leq c(b_{h_{i-1}}) < x^r_k, s_k \in S \setminus S_{i-1}\}$, i.e., the set of sensors s_k that do not cover $b_{h_{i-1}}$ in C_{i-1} but can be moved leftwards to cover $b_{h_{i-1}}$; e.g., see Fig. 2. Note that each sensor of S_{i2} is currently at its rightmost λ-reachable location in C_{i-1}.

If $S_{i2} \neq \emptyset$, then among all sensors of S_{i2}, we choose the leftmost one (with respect to their positions in C_{i-1}) as s_{g_i} and add it to S_{i-1} to obtain S_i. We move s_{g_i} leftwards until $b_{h_{i-1}}$ is covered (i.e., it is on the left half-circle of ∂D_{g_i}); this obtains the configuration C_i. Next, we set $h_i = k+1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = n+1$, then all barrier points are covered and thus we can stop the algorithm and report $\lambda \geq \lambda^*$. Following the similar analysis as

Lemma 5, we can show that all algorithm invariants hold.

If $S_{i2} = \emptyset$, then we terminate the algorithm and report that $\lambda < \lambda^*$.

In summary, if $S_{i1} = S_{i2} = \emptyset$, then the algorithm will terminate and report $\lambda < \lambda^*$. Otherwise, a sensor s_{g_i} is found from either S_{i1} (if it is not empty) or S_{i2} and added to S_{i-1} to obtain S_i. In either case, $h_i = k + 1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = n+1$, then the algorithm will terminate and report $\lambda \geq \lambda^*$; otherwise, the algorithm will proceed to the next iteration $i+1$ and all algorithm invariants hold. As there are m barrier points and a new barrier point is covered in each iteration, the algorithm has at most m iterations. On the other hand, as there are n sensors and each iteration finds a new sensor to form S_i, the algorithm has at most n iterations. Hence, the algorithm will stop in $\min\{n, m\}$ iterations.

The proof of the algorithm correctness is omitted but can be found in the full paper.

4.1.2 The algorithm implementation

We now provide an efficient way to implement the algorithm in $O((n+m)\log(n+m))$ time. For differentiation, we use "algorithm implemntation" to refer to the algorithm we will discuss below and use "algorithm description" to refer to the algorithm we described before in Section 4.1.1.

We sweep a point p on ℓ from left to right. The event point set is $E = \{c(b) \mid b \in B\} \cup \{x_i^l, x_i^r \mid s_i \in S\}$. We sort all points of E from left to right on ℓ and put them in a list, still denoted by E. Using the sorted list E as a guide, we sweep p on ℓ from left to right. When p encounters a point x_k^l for some sensor s_k, we insert s_k to a balanced binary search tree T in which the sensors s_k are ordered by their values x_k^r. As will be shown later, the tree T is used to maintain the set S_{i2}. When p encounters a point x_k^r, we remove s_k from T and store s_k at a variable s^* (if s^* already stores a sensor, we simply update s^* to s_k). Our algorithm implementation maintains the following invariant: the sensor s_k stored in s^* and all sensors of T are at their positions in C_0.

Now consider the case where p encounters $c(b_j)$ for some barrier point b_j. We assume that j is equal to h_{i-1} for some i as defined in the algorithm description. The assumption is true initially when $j = 1$ and $i = 1$. This means that we are at the beginning of the i-th iteration in the algorithm description. We first need to check whether $S_{i1} = \emptyset$. To this end, we have the following Lemma 6. But before giving Lemma 6, we prove the following observation, which will be used in the proofs of Lemma 6 and other lemmas.

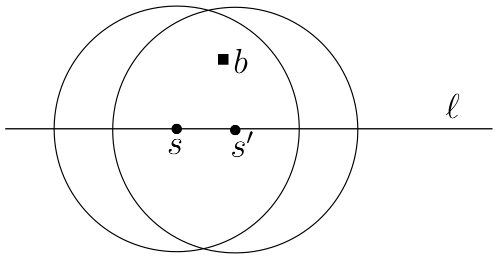


Figure 3: Illustrating Observation 2.

Observation 2 *Consider a barrier point b and two sensors s and s'. Suppose the followings hold (e.g., see Fig. 3): (1) s' is to the right of s; (2) s covers b; (3) b is to the right of the left half-circle of $\partial D(s')$. Then, s' also covers b.*

Proof. Assume to the contrary that s' does not cover b. Then, since b is to the right of the left half-circle of $\partial D(s')$, b must be strictly to the right of the right half-circle of $\partial D(s')$. Because s' is to the right of s, b must also be strictly to the right of the right half-circle of $\partial D(s)$. But this means that s does not cover b, a contradiction. □

Lemma 6 *If the sensor s_k stored in s^* covers b_j when s_k is at x_k^r, then $s_k \in S_{i1}$; otherwise (including the case where s^* does not store any sensor) $S_{i1} = \emptyset$.*

Proof. Suppose the sensor s_k stored in s^* covers b_j when s_k is at x_k^r. To prove the lemma, it suffices to show that if $S_{i1} \neq \emptyset$, then s_k must be S_{i1}. In the following, we assume that $S_{i1} \neq \emptyset$. Our goal is to prove that s_k is in S_{i1}. Since s_k is stored in s^*, according to our algorithm implementation invariant, s_k is at x_k^r. Hence, to prove $s_k \in S_{i1}$, by the definition of S_{i1}, it is sufficient to show that s_k covers b_j (when s_k is at x_k^r).

Let s_a be a sensor of S_{i1}. If s_a is s_k, then it is vacuously true that $s_k \in S_{i1}$. In what follows, we assume that s_a is not s_k. Because s_a is in S_{i1}, according to our algorithm description, s_a is at x_a^r and has never been moved during the algorithm, and further, s_a covers b_j. Since the sweeping point p is at $c(b_j)$, which is the rightmost position on ℓ for the center of a circle of radius r to cover b_j, p must have passed x_a^r. Therefore, according to our algorithm implementation, s_a had been stored in s^* before and later s^* got updated to s_k. This implies that s_k is to the right of s_a (and both of them are at their rightmost λ-reachable locations). Because p is now at $c(b_j)$, p has already passed x_k^r. Therefore, b_j is to the right of left half-circle of $\partial D(s_k)$. Since b_j is covered by s_a and s_k is to the right of s_a, by Observation 2, b_j must be covered by s_k. □

By Lemma 6, if s^* does not store any sensor or if the sensor stored at s^* does not cover b_j, then $S_{i1} = \emptyset$.

Otherwise, the sensor stored at s^*, denoted by s_k, covers b_j and is in S_{i1}. Depending on whether $S_{i1} = \emptyset$, there are two cases to proceed.

The case $S_{i1} \neq \emptyset$. We first consider the case $S_{i1} \neq \emptyset$. In this case, according to our algorithm description, we can simply choose s_k as s_{g_i} and add it to S_{i-1} to obtain S_i. Next, we need to determine h_i, which is equal to $l + 1$ with l as the largest index such that all barrier points $b_j, b_{j+1}, \ldots, b_l$ can be covered by sensors of S_i. To find l, we initialize $l = j$ and then keep sweeping p rightwards. If p encounters a point x^l_k or x^r_k, we process the event in the same way as before. If p encounters a point $c(b_{j'})$, we know that $j' = l + 1$. We need to determine whether $b_{j'}$ can be covered by sensors of S_i. For this, we have the following lemma.

Lemma 7 *$b_{j'}$ can be covered by sensors of S_i if and only if $b_{j'}$ can be covered by s_{g_i}.*

Proof. If $b_{j'}$ is covered by s_{g_i}, then it is vacuously true that $b_{j'}$ is covered by sensors of S_i because s_{g_i} is in S_i.

Now assume that $b_{j'}$ is covered by a sensor $s_{g_a} \in S_i$. We need to prove that s_{g_i} also covers $b_{j'}$. This is obviously true if $a = i$. We now assume $a \neq i$, implying that $a < i$. According our algorithm implementation, $b_{j'}$ is to the right of the left half-circle of $\partial D(s_k)$ and $s_{g_i} = s_k$. According to our algorithm invariants in the algorithm description, s_{g_a} is to the left of s_{g_i}. Since s_{g_a} covers $b_{j'}$, by Observation 2, s_{g_i} also covers $b_{j'}$. □

In light of Lemma 7, we check whether $b_{j'}$ is covered by s_{g_i}. If yes, we increment l by one and proceed as above (if $l = n$, then all barrier points are covered and we can stop the algorithm and report $\lambda \geq \lambda^*$). Otherwise, we set $h_i = j'$; in this case, we have finished the i-th iteration of the algorithm and we then proceed to the $(i + 1)$-th iteration.

The case $S_{i1} = \emptyset$. We now consider the case $S_{i1} = \emptyset$. In this case, we need to know whether $S_{i2} = \emptyset$, and if not, we need to find the leftmost sensor in S_{i2}. For this, we have the following lemma.

Lemma 8 *The sensors stored in the current tree T are exactly the sensors of S_{i2}.*

Proof. We prove the lemma by analyzing our algorithm implementation. Recall that the sweeping point p is now at $c(b_j)$ and $j = h_{i-1}$.

- Let s_a be a sensor of S_{i2}. We show that s_a is stored in T. Indeed, since s_a is in S_{i2}, by the definition of S_{i2}, we have $x^l_a \leq c(b_j) < x^r_a$. According to our algorithm implementation, when p encounters x^l_a, s_a is inserted to T and will not be removed from T until p counters x^r_a. Since p is at $c(b_j)$ right now and $c(b_j) < x^r_a$, s_a is still in T.
- Let s_a be a sensor stored in T. We show that s_a is in S_{i2}. Indeed, since s_a is in T, according to our algorithm implementation, p has already passed x^l_a but not encountered x^r_a yet. Since p is at $c(b_j)$ right now, we obtain that $x^l_a \leq c(b_j) < x^r_a$. Further, according to our algorithm implementation invariant, s_a has not been moved from its position in C_0, i.e., s_a is still at x^r_a. Therefore, s_a is in S_{i2}.

This proves the lemma. □

In light of Lemma 8, we can use T to find the leftmost sensor of T in $O(\log n)$ time; let s_k denote the sensor. We choose s_k as s_{g_i} and add it to S_{i-1} to obtain S_i. Then, we move s_k leftwards to $c(b_j)$, i.e., setting $x'_k = c(b_j)$, and remove s_k from T. We also remove both events x^l_k and x^r_k from the list E because we do not need to process these two events anymore.[2] Next, we need to determine h_i. This can be done using the same method as in the above case where $S_{i1} \neq \emptyset$ (i.e., keep sweeping p rightwards and making use of Lemma 7, which is still applicable here). After h_i is found, we finish the i-th iteration of the algorithm and begin the $(i + 1)$-th iteration.

This finishes the description of the algorithm implementation. The proof of the following lemma analyzes the running time of the algorithm.

Lemma 9 *Given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O((n + m) \log(n + m))$ time.*

Proof. We analyze the running time of our implementation. In the beginning, computing the sorted list E takes $O((n + m) \log(n + m))$ time. There are $O(n + m)$ operations on E, each of which takes $O(1)$ time. The time we spent on the binary search tree T is bounded by $O(n \log n)$ as there are n sensors and each sensor can be inserted and removed from T at most once (also, there are at most n operations of "finding the leftmost sensor"). Therefore, the total time of the algorithm is $O((n+m) \log(n+m))$. More specifically, after the points of E are sorted in $O((n + m) \log(n + m))$ time, the rest of the algorithm takes $O(m + n \log n)$ time. □

4.2 The optimization problem

We now solve the optimization problem, i.e., computing λ^*, by using the algorithm of Lemma 9 as a subroutine. We begin with the following lemma.

Lemma 10 *λ^* is equal to $(x_i - \sqrt{r^2 - y^2_{b_j}} - x_{b_j})/w_i$ or $(x_{b_j} - \sqrt{r^2 - y^2_{b_j}} - x_i)/w_i$ for a sensor s_i and a barrier point b_j.*

[2]To implement each remove operation in constant time, we can store the list E by a doubly-linked list and associate each of the values x^l_a and x^r_a for all sensors $s_a \in S$ with a pointer pointing to its location in E.

Proof. The proof is almost the same as that of Lemma 2 except that we have to consider the weight in the last step of the proof. We briefly discuss it below.

Consider an optimal solution OPT, where λ^* is the maximum moving cost of all sensors. Then, λ^* is equal to the moving cost of some sensor s_i. Let x'_i be the x-coordinate of s_i in OPT. If $x'_i < x_i$, then s_i has been moved leftwards and there must be a barrier point b_j on the left-circle of $\partial D(s_i)$. Thus, we have $x'_i = \sqrt{r^2 - y_{b_j}^2} + x_{b_j}$. Hence, $\lambda^* = (x_i - x'_i)/w_i = (x_i - \sqrt{r^2 - y_{b_j}^2} - x_{b_j})/w_i$. If $x'_i > x_i$, by similar analysis, we can show that $\lambda^* = (x_{b_j} - \sqrt{r^2 - y_{b_j}^2} - x_i)/w_i$. □

For each sensor s_i, we will define two sorted arrays $A_i[1 \cdots m]$ and $B_i[1 \cdots m]$ of size m each. Unlike the unweighted case where defining sorted arrays is relatively straightforward, here the definitions are quite subtle. We define the array A_i first, which consists of the values $(x_i - \sqrt{r^2 - y_{b_j}^2} - x_{b_j})/w_i$ for all $j = 1, \ldots, m$. For each $j \in [1, m]$, let $a_j = \sqrt{r^2 - y_{b_j}^2} + x_{b_j}$. We sort the values a_j for all $j = 1, \ldots, m$ in ascending order. For each $j \in [1, m]$, we let $\pi(j) = k$ if a_k ranks the j-th place in the above sorted list. Hence, $\pi(\cdot)$ is a permutation of the indices $1, 2, \ldots, m$; note that we can obtain $\pi(\cdot)$ in $O(m \log m)$ time. For each $j \in [1, m]$, we define $A_i[j] = (x_i - a_{\pi(j)})/w_i$. In light of the definition of $\pi(\cdot)$, A_i is a sorted array. Analogously, we can define a sorted array B_i for the m values $(x_{b_j} - \sqrt{r^2 - y_{b_j}^2} - x_i)/w_i$, $j = 1, \ldots, m$. Note that the permutation $\pi(\cdot)$ can be used to define A_i for all $i = 1, 2, \ldots, n$. Hence, in $O(n + m \log m)$ time, we can implicitly form $2n$ sorted arrays A_i and B_i for all $i = 1, 2, \ldots, n$, such that given any index j and any array A_i (resp., B_i), we can obtain the array element $A_i[j]$ (resp., $B_i[j]$) in $O(1)$ time. Also, Lemma 10 implies that λ^* is the smallest feasible value of all elements of these arrays. By applying Lemma 3 and using our decision algorithm in Lemma 9, we can find λ^* in $O((n + m) \log^2(n + m))$ time. We summarize our result in the following theorem.

Theorem 11 *Given a set of m barrier points in the plane and a set of n weighted sensors on a line ℓ, the problem of moving sensors on ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O((n + m) \log^2(n + m))$ time.*

5 The mobile interval coverage problem

In this section, we consider the mobile interval coverage problem, where the barrier points are on the x-axis ℓ while the sensors can be anywhere in the plane. The problem is to move all sensors to ℓ to cover all barrier points so that the minimum moving cost of all sensors is minimized.

We first sort all barrier points from left to right on ℓ in $O(m \log m)$ time; let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list. Recall that for each sensor $s_i \in S$, (x_i, y_i) is its coordinate. In the weighted case, each sensor s_i has a weight $w_i > 0$. In the following, we only give an algorithm for the weighted case because we do not have a faster algorithm for the unweighted case. Our goal is to compute the optimal moving cost λ^*. Note that since we require that all sensors finally move to ℓ, it must hold that $\lambda^* \geq \max_{1 \leq i \leq n} w_i \cdot y_i$.

We again first consider the decision problem: Given any λ, decide whether $\lambda \geq \lambda^*$. We present an algorithm of $O(m + n \log n)$ time (not including the time for sorting the barrier points) for the problem. Later we will solve the optimization problem (i.e., computing λ^*) using Lemma 3 and the decision algorithm.

5.1 The decision problem

Consider a value λ. We assume that $\lambda \geq \max_{1 \leq i \leq n} w_i \cdot y_i$ since otherwise it is impossible to move all sensors to ℓ (and thus we immediately report $\lambda < \lambda^*$). For each sensor s_i, define $x_i^r = x_i + \sqrt{(\lambda/w_i)^2 - y_i^2}$ and $x_i^l = x_i - \sqrt{(\lambda/w_i)^2 - y_i^2}$. We call x_i^r (resp., x_i^l) the *rightmost (resp., leftmost) λ-reachable location* of s_i.

At the outset, we move each sensor s_i to x_i^r on ℓ. Let C_0 denote the resulting configuration. The rest of the algorithm is similar to the one in Section 4.1. In fact, we can basically apply the same algorithm. But since the problem setting here is simpler (because all barrier points are now on ℓ), below we describe the algorithm in a simpler way (the running time is also slightly faster if m is significantly larger than n).

Consider the i-th iteration of the algorithm (initially $i = 1$). Let C_{i-1} denote the configuration right before the iteration. Our algorithm maintains the following invariants:

1. A subset $S_{i-1} = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i-1)}\}$ of sensors has been computed.
2. In C_{i-1}, each sensor s_k of S_{i-1} is at a location, denoted by x'_k, which may not be equal to x_k^r, while sensors of $S \setminus S_{i-1}$ are still in their locations of C_0.
3. An index h_{i-1} of a barrier point is maintained such that in the configuration C_{i-1}, the barrier point $b_{h_{i-1}}$ is not covered by any sensor of S_{i-1} while b_k is covered by a sensor in S_{i-1} for each $k < h_{i-1}$
4. Each sensor of S_{i-1} covers at least one barrier point b_j with $j < h_{i-1}$ in C_{i-1}.
5. The locations of the sensors $s_{g_1}, s_{g_2}, \ldots, s_{g_{i-1}}$ in C_{i-1} are sorted from left to right on ℓ.

6. The barrier point $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$ of $s_{g_{i-1}}$ if $S_{i-1} \neq \emptyset$.

Initially when $i = 1$, we have $S_0 = \emptyset$ and set $h_0 = 1$; thus all algorithm invariants hold. The i-th iteration of the algorithm finds a sensor s_{g_i} from $S \setminus S_{i-1}$ and move it to a new location x'_{g_i}; we thus obtain a new configuration C_i with $S_i = S_{i-1} \cup \{s_{g_i}\}$. We briefly discuss algorithm below.

Define S_{i1} be the set of sensors that cover the barrier point $b_{h_{i-1}}$ in C_{i-1}. Again, due to our algorithm invariants, $S_{i1} \subseteq S \setminus S_{i-1}$.

If $S_{i1} \neq \emptyset$, we choose an arbitrary sensor in S_{i1} as s_{g_i} and set $x'_{g_i} = x^r_{g_i}$. Hence, $C_i = C_{i-1}$. Next, we set $h_i = k + 1$, where k is the largest index such that all barrier points of $[h_{i-1}, k]$ are covered by S_i (it is easy to see that a barrier point b_l with $l \geq h_{i-1}$ is covered by S_i if and only if b_l is covered by s_{g_i}, i.e., Lemma 7 is still applicable). If $k = m$, then we stop the algorithm and report $\lambda \geq \lambda^*$.

If $S_{i1} = \emptyset$, we define S_{i2} as the set of sensors of $S \setminus S_{i-1}$ that do not cover $b_{h_{i-1}}$ in C_{i-1} but can be moved leftwards to cover $b_{h_{i-1}}$. If $S_{i2} \neq \emptyset$, we choose the leftmost sensor of S_{i2} as s_{g_i} and set $x'_{g_i} = x_b + r$ to obtain a new configuration C_i, where $b = b_{h_{i-1}}$. Next, we set h_i in the same way as above. If $S_{i2} = \emptyset$, then we terminate the algorithm and report $\lambda < \lambda^*$.

The algorithm will terminate in at most $\min\{m, n\}$ iterations. The correctness of the algorithm can be proved in a similar way as before.

To implement the algorithm, we first sort the barrier points in the preprocessing, which takes $O(m \log m)$ time. Then, given any λ, we can implement the algorithm in $O(m + n \log n)$ time using essentially the same implementation as in Section 4.1. We briefly discuss it below.

We first compute x^r_i and x^l_i for each sensor $s_i \in S$, and sort all these $2n$ values in $O(n \log n)$ time. Then, we compute the value $c(b)$ for each barrier point $b \in B$. Unlike in Section 4.1, here the value $c(b)$ is fixed and does not depend on λ, and the sorted list of $c(b)$ of all barrier points $b \in B$ is consistent with the sorted list of all barrier points $b \in B$. Since the sorted list of B is already computed in the preprocessing, we can obtain the sorted list of $c(b)$ for all barrier points $b \in B$ in $O(m)$ time. By merging it with the sorted list of x^r_i and x^l_i for all sensors $s_i \in S$, we can obtain the sorted list of the event set $E = \{c(b) \mid b \in B\} \cup \{x^l_i, x^r_i \mid s_i \in S\}$ in additional $O(n + m)$ time. Using E, we run the same sweeping algorithm as before. We still use a binary search tree T to maintain the sensors of S_{i2} and use a variable s^* to store a sensor of S_{i1}. When p encounters x^l_k for a sensor s_k, we insert s_k to T. When p encounters x^r_k, we remove s_k from T and set s^* to s_k. When p encounters a barrier point b_j, we determine the sensor s_{g_i} using the variable s^* and the tree T in the same way as before. As analyzed in the proof of Lemma 9, the total time of the algorithm is $O(m + n \log n)$.

Lemma 12 *After $O(m \log m)$ time preprocessing, given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O(m + n \log n)$ time.*

5.2 The optimization problem

We now show how to compute λ^*. We first implicitly form $2n$ sorted arrays as follows. For each sensor s_i, we define two sorted arrays $A_i[1 \ldots m]$ and $B_i[1 \cdots m]$ of size m each: for each $1 \leq j \leq m$, $A_i[j] = (\sqrt{x_i^2 + y_i^2} - r - x_{b_j})/w_i$ and $B_i[j] = (x_{b_j} - r - \sqrt{x_i^2 + y_i^2})/w_i$. One can verify that λ^* must be one of the elements of these arrays (e.g., using analysis similar to Lemmas 2 and 10) and each array is sorted. Then, applying Lemma 3 with our decision algorithm in Lemma 12, λ^* can be computed in $O(m \log m + (m+n \log n) \log(n+m))$ time, which is bounded by $O(m \log m + n \log^2 n)$.[3]

Theorem 13 *Given a set of m barrier points on a line ℓ and a set of n weighted sensors in the plane, the problem of moving sensors to ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O(m \log m + n \log^2 n)$ time.*

References

[1] A.M. Andrews and H. Wang. Minimizing the aggregate movements for interval coverage. *Algorithmica*, 78:47–85, 2017.

[2] B. Bhattacharya, B. Burmester, Y. Hu, E. Kranakis, Q. Shi, and A. Wiese. Optimal movement of mobile sensors for barrier coverage of a planar region. *Theoretical Computer Science*, 410(52):5515–5528, 2009.

[3] D.Z. Chen, Y. Gu, J. Li, and H. Wang. Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain. *Discrete and Computational Geometry*, 50:374–408, 2013.

[4] D.Z. Chen, X. Tan, H. Wang, and G. Wu. Optimal point movement for covering circular regions. *Algorithmica*, 72:379–399, 2015.

[5] D.Z. Chen, C. Wang, and H. Wang. Representing a functional curve by curves with fewer peaks. *Discrete and Computational Geometry*, 46(2):334–360, 2011.

[6] J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the maximum sensor movement for barrier coverage of a line segment. In *Proceedings of the 8th International Conference on Ad-Hoc, Mobile and Wireless Networks*, pages 194–212, 2009.

[3]To see this, first notice that $m \log m + (m + n \log n) \log(n + m) = O(m \log m + n \log n \log(n + m))$. Further, if $m \geq n^2$, then $m \log m + n \log n \log(n + m) = O(m \log m)$; otherwise, $\log(n + m) = \Theta(\log n)$ and thus $m \log m + n \log n \log(n + m) = O(m \log m + n \log^2 n)$.

[7] J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the sum of sensor movements for barrier coverage of a line segment. In *Proceedings of the 9th International Conference on Ad-Hoc, Mobile and Wireless Networks*, pages 29–42, 2010.

[8] S. Dobrev, S. Durocher, M. Eftekhari, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, S. Shende, and J. Urrutia. Complexity of barrier coverage with relocatable sensors in the plane. *Theoretical Computer Science*, 579:64–73, 2015.

[9] H. Fan, M. Li, X. Sun, P. Wan, and Y. Zhao. Barrier coverage by sensors with adjustable ranges. *ACM Transactions on Sensor Networks*, 11:14:1–14:20, 2014.

[10] G. Frederickson and D. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.

[11] G.N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the 2nd Annual ACM-SIAM Symposium of Discrete Algorithms (SODA)*, pages 168–177, 1991.

[12] G.N. Frederickson. Parametric search and locating supply centers in trees. In *Proceedings of the 2nd International Workshop on Algorithms and Data Structures (WADS)*, pages 299–319, 1991.

[13] D.S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32:130–136, 1985.

[14] P. Huang, W Zhu, and L. Guo. On the complexity of and algorithms for min-max target coverage on a line boundary. In *Proceedings of the 15th International Conference on Theory and Applications of Models of Computation (TAMC)*, pages 313–324, 2019.

[15] S. Kumar, T.H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 284–298, 2005.

[16] V.C.S Lee, H. Wang, and X. Zhang. Minimizing the maximum moving cost of interval coverage. *International Journal of Computational Geometry and Applications*, 27:187–205, 2017.

[17] S. Li and H. Shen. Minimizing the maximum sensor movement for barrier coverage in the plane. In *Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 244–252, 2015.

[18] S. Li and H. Wang. Algorithms for covering multiple barriers. *Theoretical Computer Science*, 758:61–72, 2019.

[19] D. Liang, H. Shen, and L. Chen. Maximum target coverage problem in mobile wireless sensor networks. *Sensors*, 21:1–13, 2015. Article No. 184.

[20] M. Mehrandish, L. Narayanan, and J. Opatrny. Minimizing the number of sensors moved on line barriers. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 653–658, 2011.

[21] X. Zhang, H. Fan, V.C.S. Lee, M. Li, Y. Zhao, and C. Liu. Minimizing the total cost of barrier coverage in a linear domain. *Journal of Combinatorial Optimization*, 36:434–457, 2018.

Approximability of (Simultaneous) Class Cover for Boxes

Jean Cardinal [*] Justin Dallant [†] John Iacono [‡]

Abstract

Bereg et al. (2012) introduced the Boxes Class Cover problem, which has its roots in classification and clustering applications: Given a set of n points in the plane, each colored red or blue, find the smallest cardinality set of axis-aligned boxes whose union covers the red points without covering any blue point. In this paper we give an alternative proof of APX-hardness for this problem, which also yields an explicit lower bound on its approximability. Our proof also directly applies when restricted to sets of points in general position and to the case where so-called half-strips are considered instead of boxes, which is a new result.

We also introduce a symmetric variant of this problem, which we call Simultaneous Boxes Class Cover and can be stated as follows: Given a set S of n points in the plane, each colored red or blue, find the smallest cardinality set of axis-aligned boxes which together cover S such that all boxes cover only points of the same color and no box covering a red point intersects a box covering a blue point. We show that this problem is also APX-hard and give a polynomial-time constant-factor approximation algorithm.

1 Introduction

Many approaches to data mining, classification and clustering tasks show a close proximity to computational geometry, often embedding data in some space and reducing or formalizing the considered problem as a geometric one. Some well-known approaches include support vector machines, nearest neighbour classifiers or k-means clustering. In this paper we study the computational hardness and approximability of two variants of another geometric problem which has its roots in classification and clustering, known as the Class Cover problem. It can be stated as follows: given a set of n points, each colored red or blue, find the smallest number of balls centered at red points which cover all red points without covering any blue points (see Section 2 for the exact definition of "cover" we use here). This problem was introduced by Cannon and Cowen [4], motivated by connections to the measure of separability between two classes of points defined by Cowen and Priebe [6] as well as applications to classification and data reduction. In this paper they showed that the problem was NP-hard but admitted a polynomial-time $(1 + \ln(n))$-factor approximation. In the Euclidean setting for constant dimension they show that the problem even admits a polynomial-time approximation scheme (PTAS). The problem was studied by others in the context of applications to pattern recognition [7] and classifiers [12].

Here we study a variant introduced by Bereg et al. [2] which can be formulated as follows:

Definition 1 (Boxes Class Cover (BCC)) *Given a set of n points in the plane, each colored red or blue, find the smallest cardinality set of axis-aligned boxes which together cover the red points without covering any blue point.*

The authors show that this variant is NP-hard and give a polynomial-time algorithm which achieves a $O(1+\log c)$-approximation, where c is the size of an optimal cover. They also study a few restricted cases, among those covering with squares or so-called half-strips, where they show that they remain NP-hard but admit a $O(1)$-factor approximation in polynomial time. The variant for squares was later shown to admit a PTAS [1] while for the general BCC problem it was shown by Shanjani [9] that no PTAS can exist unless P = NP (in this paper the author also notes that the original reduction used by Bereg et al. in fact already shows this result).

Note that in all the variations mentioned above, there is a clear asymmetry between the roles of the two color classes, which is not always warranted in applications. As a measure of separability of two classes this can also lead to a phenomenon where the first class can be "separated" from the second using few boxes, while separating the second from the first requires many boxes. This motivates us to consider a symmetric version of the problem, which we formulate as follows:

Definition 2 (Simultaneous BCC (SBCC)) *Given a set S of n points in the plane, each colored red or blue, find the smallest cardinality set of axis-aligned boxes which together cover S such that all boxes cover only points of the same color and no box covering a red*

[*]Department of Computer Science, Université libre de Bruxelles, jcardin@ulb.ac.be

[†]Department of Computer Science, Université libre de Bruxelles, justin.dallant@ulb.be Supported by the French Community of Belgium via the funding of a FRIA grant.

[‡]Department of Computer Science, Université libre de Bruxelles, john@johniacono.com Supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1.

point intersects the interior of a box covering a blue point.

In Section 2 we go over some basic definitions and lemmas. In Section 3 we give an alternative proof to the fact that there is no PTAS for BCC unless P = NP, by a reduction from Vertex Cover. While this is already known, our proof is more direct than previous ones and allows us to exhibit a specific lower bound on the approximation factor. The proof also works for half-strips and for the restriction where we consider only point sets in general position, which are new results. In Section 4 we show that for the SBCC problem there is again no PTAS unless P = NP. Finally in Section 5 we give a polynomial-time constant-factor approximation algorithm for SBCC. In the way to doing so we show that requiring all boxes to be independent in the SBCC blows up the size of an optimal solution by at most a factor of 9.

2 Some definitions and lemmas

We start by giving some definitions of the object we will consider in this paper. Note that we sometimes use the name of a problem (such as "BCC") to denote the corresponding structure we want to minimize (in case of a BCC, this would be a set of axis-aligned boxes covering the red points without covering any blue point).

Definition 3 *A* bichromatic *set of points is a set of points where some points are said to be red and the rest are said to be blue.*

We will use the notation $S = R \cup B$ to denote such a set of points, where R is the set of red points, B is the set of blue points, and $R \cap B = \emptyset$. In what follows and in the rest of this paper, we will consider only closed axis-aligned boxes with non-empty interiors and unless otherwise specified, they are bounded.

We use the following slightly technical definition of a rectilinear polygon, which will make some of the results and proofs easier to phrase:

Definition 4 *A* rectilinear polygon *P is a connected set of points in the plane such that:*

- *the closure of P can be obtained as the union of a finite number of axis-aligned boxes $\mathcal{B}$,*
- *every pair of boxes in $\mathcal{B}$ is either disjoint or intersects in more than one point,*
- *and all connected components of $\mathbb{R}^2 \setminus P$ have non-empty interior.*

Note that from this definition a rectilinear polygon is not necessarily closed (nor open). We extend the usual definitions of vertices and edges of a polygon to rectilinear polygons. Note that such a polygon may or may not have holes (bounded connected components of $\mathbb{R}^2 \setminus P$) which also contribute vertices and edges.

We add a few more basic definitions:

Definition 5 *We call* outer-hull *of P, denoted as $\mathrm{oh}(P)$, the union of all edges adjacent to the unbounded connected component of $\mathbb{R}^2 \setminus P$. The* outer-complexity *of P is the number of vertices in $\mathrm{oh}(P)$. A* convex vertex *of P is a vertex such that the right angle formed by the two edges adjacent to the vertex is directed towards the interior of P. Otherwise we call the vertex* reflex.

For a polygon P, we let $|P|$ denote the number of vertices in P. The same applies for a union of polygons or the outer-hull of a (union of) polygon(s). In all other cases we let $|S|$ denote the cardinality of the set S (in such cases S will always be finite).

Definition 6 *Let $S = R \cup B$ be a bichromatic set of points. We say that an axis-aligned box K* covers *a point p if p is in the interior of K. We say that K is red (resp. blue) if it covers only red (resp. blue) points of S. It is* monochromatic *if it is either red or blue.*

We say that a set Z of axis-aligned boxes is a BCC *of S if all red points in S are covered by some box in Z and all boxes in Z are red. We say that Z is an* SBCC *S if all points in S are covered by some box in Z, all boxes in Z are monochromatic and no two boxes of different color have their interiors intersecting.*

We say that Z covers *a rectilinear polygon P if P is included in the union of all boxes in Z. We say that this cover is* exact *(or that Z covers P exactly) if the closure of P is equal to the union of all boxes in Z.*

We finish this section with two lemmas which will prove useful to us.

Lemma 7 *Let P be a rectilinear polygon. Let c be the number of convex vertices of P, r be the number of reflex vertices and h be the number of holes. Then we have $r = c + 4(h-1)$. In particular, if P is obtained as the union of k axis-aligned boxes, then it has outer-complexity less than $8k$, as each box contributes at most 4 convex vertices to the outer face.*

Proof. It is well known (see for example Lemma 1 in [11]) that for a rectilinear polygon with no holes we have $r = c-4$. Now consider some hole H of P. If we view H as a rectilinear polygon itself and denote c_H and r_H the number of convex and reflex vertices of H respectively, then we have $r_H = c_H - 4$ (as H has no hole). Because a convex (resp. reflex) vertex for H is a reflex (resp. convex) vertex for P, the hole H contributes $r_H + 4$ reflex vertices and r_H convex vertices to P. Because for a rectilinear polygon with no holes we have $r = c - 4$ and every hole contributes 4 more reflex vertices than convex vertices, it holds that in a a rectilinear polygon with h holes we have $r = c - 4 + 4h = c + 4(h-1)$. □

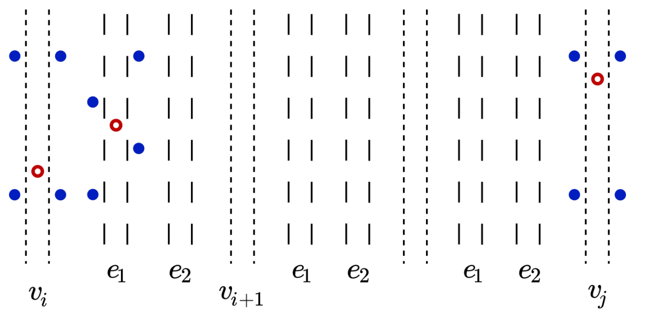


Figure 1: The gadget corresponding to an edge e_1 between vertices v_i and v_j. The vertex lanes are represented with short dashes and the edge lanes with long ones. Here there are two edge lanes between consecutive vertex lanes, thus representing a graph with two edges (assuming all lanes appear on this figure).

Lemma 8 ([8]) *Any rectilinear polygon with n vertices and h holes can be covered exactly with at most $n/2 + h - 1$ axis-aligned boxes.*

3 An approximation-preserving reduction from Minimum Vertex Cover to Minimum Class Cover

Here we show how to reduce the (minimum) Vertex Cover problem to BCC in a way that shows the following:

Theorem 9 *If BCC can be approximated in polynomial time within a constant factor of $(1 + \epsilon)$ for $\epsilon > 0$ then Vertex Cover can be approximated in polynomial time within a constant factor of $(1 + (d+1)\epsilon)$ on graphs with maximum degree bounded by d.*

Using the fact that Vertex Cover is NP-hard to approximate within a constant factor of $1+1/52$ on graphs with maximum degree at most 4 [5] we also get the following:

Corollary 10 *Approximating BCC within a constant factor of $1 + 1/260$ (≈ 1.0038) is NP-hard.*

Note that as mentionned in the introduction, the reduction used by Bereg et al. from Rectilinear Polygon Cover (which itself admits a reduction from Vertex Cover in bounded degree graphs) also proves APX-hardness. However the reduction here is more direct, which allows us to easily get the more precise $(1+(d+1)\epsilon)$ relationship and an explicit lower bound on the approximation factor. It also has the advantage that it is easy to see that it works for points in general position also (by perturbing the points slightly), whereas this is not immediately the case for the reduction from Rectilinear Polygon Cover.

Proof. [of Theorem 9] Suppose we are given a simple undirected graph G with n vertices denoted as $v_1, v_2 \dots, v_n$ and m edges denoted as $e_1, e_2 \dots, e_m$, of maximum degree at most d.

Imagine creating n disjoint vertical slabs in the plane, one for each vertex (in the order given by the vertex indices). We call such a slab a vertex lane. Between any two consecutive vertex lanes, we create m additional disjoint vertical slabs (also disjoint from the previously created vertex lanes), one for each edge. We call these the edge lanes.

Now we describe a gadget which will encode an edge of G as a set of points in the plane to cover. Consider an edge e between v_i and v_j, $i < j$. We place a red point on the vertex lane corresponding to v_i and one with larger y-coordinate on the vertex lane corresponding to v_j. We also add a red point on the unique edge lane corresponding to e between the lanes of v_i and v_{i+1}, with y-coordinate between those of the two previously placed points. Next, we add a few blue points which restrict the type of boxes which can be used to cover these red points (see Figure 1). We create an instance S of the BCC problem by placing the edge gadgets such that their minimal bounding boxes are all disjoint.

Consider the minimum vertex cover of G. Denote its size as opt_G. For every vertex in this cover we can create a box covering the corresponding vertex lane entirely. We thus obtain a set of opt_G boxes such that in every gadget, either the left or right vertex lane is covered (or both). In each gadget we need exactly one additional box to fully cover the red points. Thus, we obtain a solution to the BCC problem of size $\text{opt}_G + m$. In particular, if we let opt_S denote the minimum size of a BCC of S, we have $\text{opt}_S \leq \text{opt}_G + m$.

Now consider a solution to the BCC problem using b boxes. We assume without loss of generality that no two boxes cover the exact same subset of red points.

Consider the gadget corresponding to some edge. Notice that a red box covering the middle red point cannot cover any red point in any different gadget (as this point is the only one in its edge lane). Moreover, a box covering the left or right point can only additionally cover the middle point or other points on the same vertex lane (but not both simultaneously).

If there is a box covering the middle point which covers neither the left nor right point, we can either delete this box (if another box also covers the middle point) or replace it with a box covering, say, both the middle and left point. This does not make any other red point uncovered. Consider this change done from now on. If the middle red point is covered twice, then we can replace the box which also covers, say, the right red point with a box covering the whole vertex lane corresponding to that right point without making any red point uncovered. If any vertex lane is fully covered multiple times after this process we can simply discard all but one of the boxes covering it.

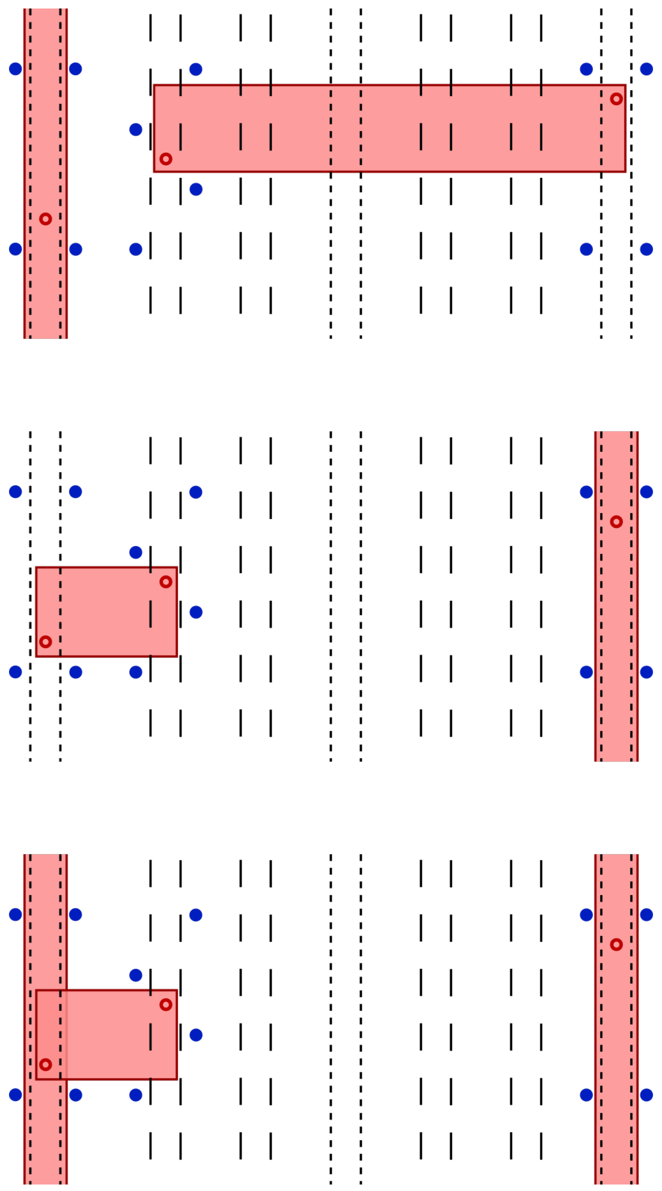


Figure 2: Regardless if the left vertex lane, right vertex lane or both are covered, there is always exactly one additional box covering the remaining red points in an edge-gadget.

After making these changes, we end up with a set of at most b boxes such that for any of the m edge-gadgets, either the lane corresponding to the left point is fully covered or the lane corresponding to the right point is fully covered (or both). In all cases, there is one additional box covering the middle point (see Figure 2. Thus, if ℓ is the number of lanes covered, the total number of boxes is $\ell + m \leq b$. We can create a vertex cover of G of size $\ell \leq b - m$ by choosing every vertex such that the corresponding lane is fully covered.

Now say that we can approximate BCC in polynomial time within a factor of $(1+\epsilon)$. We can run this algorithm on S to obtain a class cover of size at most $(1+\epsilon)\text{opt}_S$. By the process described above we can then obtain a vertex cover of G of size at most

$$\begin{aligned}(1+\epsilon)\text{opt}_S - m &\leq (1+\epsilon)(\text{opt}_G + m) - m \\ &= (1+\epsilon)\text{opt}_G + \epsilon \cdot m.\end{aligned}$$

Because G is of maximum degree at most d, every vertex can cover at most d edges and we have $\text{opt}_G \geq \frac{m}{d}$, i.e. $m \leq d \cdot \text{opt}_G$. Thus, the vertex cover we obtain is of size at most

$$\begin{aligned}(1+\epsilon)\text{opt}_G + \epsilon \cdot m &\leq (1+\epsilon)\text{opt}_G + \epsilon \cdot d \cdot \text{opt}_G \\ &= (1+(d+1)\epsilon)\text{opt}_G.\end{aligned}$$

This concludes the proof. □

Note that this proof works exactly the same if we replace the axis-aligned boxes with axis-aligned half-strips (axis-aligned boxes which are unbounded in one of the four axis-aligned directions). By a simple perturbation argument, it also yields the same results when restricted to sets of points in general position (that is, where no three points are collinear).

4 An approximation hardness proof for Simultaneous Boxes Class Cover

In this section we prove that SBCC is APX-hard. Here, the proof strategy used in the previous section breaks down because of the interactions between the boxes covering the red points and those covering the blue points. Instead, we revert back to the proof strategy used by Bereg et al. by a reduction from Rectilinear Polygon Cover.

Definition 11 (Rectilinear Polygon Cover (RPC)) *Given a closed rectilinear polygon P, find the smallest cardinality set of axis-aligned boxes covering P exactly.*

In our case we are covering the red and blue points simultaneously so we consider a slightly different problem, where we want to cover P and its complement simultaneously.

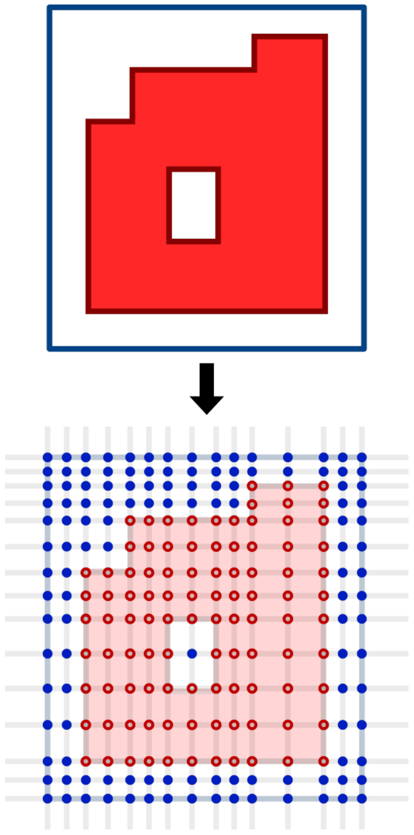


Figure 3: Reduction from SRPC to SBCC.

Definition 12 (Simultaneous RPC (SRPC)) *Given a closed rectilinear polygon P and an axis-aligned box K containing P in its interior, find the smallest cardinality set $\mathcal{R} \cup \mathcal{B}$ of axis-aligned boxes such that:*

- *the boxes in $\mathcal{R}$ cover P exactly,*
- *and the boxes in $\mathcal{B}$ cover $K \setminus P$ exactly.*

We start with the following:

Theorem 13 *If SBCC can be approximated in polynomial time within a constant factor of $(1+\epsilon)$, then SRPC can also be approximated in polynomial time within a constant factor of $(1+\epsilon)$.*

The reduction here is almost identical to the previously mentionned one from RPC to BCC. We include it for the sake of completeness.

Proof. We first show how to create a corresponding SBCC instance from a SRPC instance in polynomial time. Consider an axis aligned box K and a closed rectilinear polygon P contained in its interior (this constitutes our instance of the SRPC problem). Consider the set L_1 of all axis aligned lines which pass through a vertex of P or K. Between any two consecutive vertical (resp. horizontal) lines of L_1 draw a vertical (resp. horizontal) line. Call the set of newly drawn lines L_2. The SBCC instance we consider is the set $S = R \cup B$ of pairwise intersections between lines of $L_1 \cup L_2$, colored red if they are in P or on its boundary and blue otherwise. See Figure 3 for an illustration.

For any solution to the SRPC problem on P, we can slightly extend the boxes covering P and shrink those covering the complement of P to obtain a solution of the SBCC problem on S of the same cardinality (in polynomial time).

Let us see how we can go in the reverse direction. Consider some solution $\mathcal{R} \cup \mathcal{B}$ to the SBCC problem on S. Start by expanding the boxes in the solution in the four directions as much as possible without covering a point of the opposite color. Now for each red box K_R, replace it with the smallest axis-aligned box K'_R containing $K_R \cap P$. For each blue box K_B, replace it the smallest axis-aligned box K'_B containing $K_B \setminus P$. Notice that every point of S which does not lie on the boundary of P is still covered. Now consider some cell c of the grid $L_1 \cup L_2$. By construction, one of the corner points p of c is the intersection of two lines in L_1. Any box covering this corner has been expanded to cover the whole cell (possibly excluding the edges of the cell contained in edges of P with the). Thus P is covered by red boxes and $K \setminus P$ is covered by blue boxes. It remains to show that these covers are exact. Suppose some blue box K'_B intersects the interior of P. Because K'_B covers at least one blue point b, its interior intersects $K \setminus P$ and thus also an edge e of P. Because K'_B covers no red point, no vertex of P lies inside K'_B and e must join two opposite edges of K'_B. Thus there is an axis aligned line passing through b and intersecting e inside K'_B. By construction a red point lies on this intersection. This is a contradiction as K'_B covers no red point. We conclude that K'_B does not intersect the interior of P. The same reasoning shows that no red box K'_R intersects $K \setminus P$.

In short, the optimal solutions for the SRPC problem on P and the SBCC problem on S have the same size, and any solution to the latter can be transformed into a solution to the former of the same size in polynomial time. $\square$

Ideally at this point we would like to show that, say, the existence of a PTAS for SRPC implies the existence of a PTAS for RPC. Then, because a PTAS for RPC cannot exist unless P = NP, this would imply that the same holds for SRPC and thus also for SBCC. Unfortunately, the trivial reduction from RPC to SRPC is not approximation-preserving. Intuitively, an approximation to SRPC can be good overall because the optimal cover of the complement is large and well approximated while the optimal cover of the polygon itself is small and poorly approximated (thus yielding a poor solution to the RPC instance). To get around this, we focus on polygons where the size of the optimal cover of the complement is upper-bounded by a constant times the size of the optimal cover of the polygon itself.

Definition 14 *Let P be a closed rectilinear polygon and let K be an-axis aligned box containing P in its interior. We say that P has a* d-small-complement *if the cardinality of the smallest exact box-cover of $K \setminus P$ is at most d times the cardinality of the smallest exact box-cover of P.*

Note that the specific choice of K here does not impact the definition.

Lemma 15 *If SRPC can be approximated within a constant factor of $(1+\epsilon)$ in polynomial time, then RPC on rectilinear polygons with d-small-complements can be approximated within a constant factor of $(1+(d+1)\epsilon)$ in polynomial time.*

Proof. Let P be a closed polygon with a d-small-complement, and let K be an-axis aligned box containing P in its interior. Let opt (resp. $\mathrm{opt}_{\mathrm{sim}}$ denote the minimum cardinality of a RPC (resp. SRPC) on P. Let $\overline{\mathrm{opt}}$ denote the minimum exact box-cover of $K \setminus P$. We have $\mathrm{opt}_{\mathrm{sim}} = \mathrm{opt} + \overline{\mathrm{opt}}$ and $\overline{\mathrm{opt}} \leq d \cdot \mathrm{opt}$. Consider some solution of size $\mathrm{sol}_{\mathrm{sim}} \leq (1+\epsilon)\mathrm{opt}_{\mathrm{sim}}$ to the SRPC on P. This solution $\mathrm{sol}_{\mathrm{sim}}$ consists of a solution to the RPC problem on P of size sol together with an exact box-cover of $K \setminus P$ of size $\overline{\mathrm{sol}}$. Finally we have

$$\begin{aligned}
\mathrm{sol} &= \mathrm{sol}_{\mathrm{sim}} - \overline{\mathrm{sol}} \\
&\leq (1+\epsilon)\mathrm{opt}_{\mathrm{sim}} - \overline{\mathrm{sol}} \\
&\leq (1+\epsilon)\mathrm{opt} + (1+\epsilon)\overline{\mathrm{opt}} - \overline{\mathrm{sol}} \\
&\leq (1+\epsilon)\mathrm{opt} + \epsilon\overline{\mathrm{opt}} \\
&\leq (1+\epsilon)\mathrm{opt} + \epsilon d \cdot \mathrm{opt} \\
&\leq (1+(d+1)\epsilon)\mathrm{opt}
\end{aligned}$$

□

We have the following criteria to identify polygons with d-small-complements

Lemma 16 *Let $\alpha \geq 0$ be a constant and let P be a closed rectilinear polygon with c convex vertices and $h \leq \alpha \cdot c$ holes. Then P has a $(4+8\alpha)$-small-complement.*

Proof. Let K be an-axis aligned box containing P in its interior. Let r be the number of reflex vertices of P and $n = r + c$ be the number of vertices of P. Let s (resp. $\overline{s}$) denote the cardinality of the smallest exact box-cover of P (resp. $K \setminus P$). By Lemma 7, we have $r = c + 4(h-1) \leq c(1+4\alpha) - 4$. Because any box in an exact cover can cover at most 4 convex vertices of P, we have $s \geq c/4$. The set $K \setminus P$ is a disjoint union of rectilinear polygons with a total of $n+4$ vertices, and only one of these polygons has a hole (corresponding to P). By Lemma 8, $K \setminus P$ can be exactly covered with at most $(n+4)/2 = (r+c+4)/2 \leq \frac{c}{4}(4+8\alpha)$ boxes. Thus $\overline{s} \leq \frac{c}{4}(4+8\alpha) \leq s(4+8\alpha)$ and the claim holds. □

If we chain all these implications together, we get the following:

Theorem 17 *If there is a* PTAS *for SBCC, then there is a* PTAS *for RPC restricted to rectilinear polygons where the number of holes is at most α times the number of convex vertices, for any constant $\alpha \geq 0$.*

Berman and DasGupta showed that there is no PTAS for RPC unless P = NP [3], by reducing Vertex Cover to RPC in an approximation-preserving way. One interesting (and in our case, useful) thing to note is that the instances of RPC produced by this reduction are rectilinear polygons where every hole contributes at least one convex vertex to the polygon. In particular these instances have at least as many convex vertices as they have holes. Thus, we can state Berman and DasGupta's result in a slightly more precise way as follows:

Theorem 18 ([3]) *There is no* PTAS *for RPC unless* P = NP*, even when restricted to rectilinear polygons with no more holes than convex vertices.*

This combined with the previous theorem now immediately yields:

Theorem 19 *There is no* PTAS *for SBCC unless* P = NP.

Note that the only place where we have needed to exploit sets of points which are not in general position is in the proof of Theorem 13.

5 A constant-factor approximation for Simultaneous Boxes Class-Cover

Here we prove that SBCC can be approximated within a constant factor in polynomial time. To do so, we show that the minimum size of a SBCC of S is bounded above and below by the minimum size of a SBCC of S with interior-disjoint boxes. This latter problem can be approximated within a constant factor using the results from [10].

Theorem 20 *The size of the minimum SBCC of S with interior-disjoint boxes is at most 9 times larger than the minimum SBCC of S.*

Proof. Let $(\mathcal{R}, \mathcal{B})$ be an SBCC of $S = R \cup B$ of size k. By shrinking the boxes in $\mathcal{R}$ (resp. $\mathcal{B}$), we can assume without loss of generality that every pair of boxes in $\mathcal{R}$ (resp. $\mathcal{B}$) either intersects at more than one point or is disjoint, and that blue boxes are disjoint from red boxes.

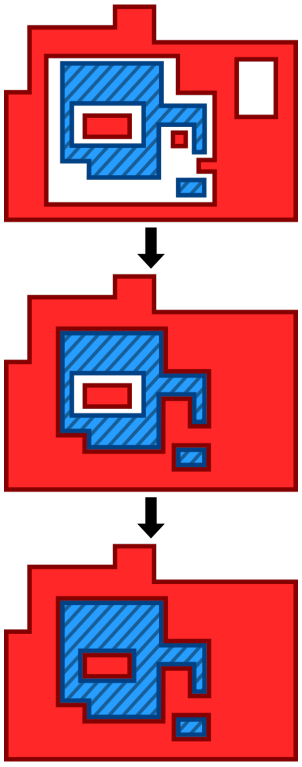


Figure 4: Illustration of the Fill procedure in the proof of Theorem 20, first applied to the holes of the larger red region then to the hole of the larger blue region.

The union of all boxes thus determines a partition Π of the plane into red rectilinear polygons (red regions), blue rectilinear polygons (blue regions) and uncovered regions (which also have rectilinear boundaries). Call a bounded uncovered region of Π a hole of Π (notice the distinction between a hole of Π and a hole of some region in Π). Call outer-complexity of Π the sum of outer-complexities of all red and blue regions in Π.

Call Fill the process of taking a hole H of Π and giving it the color of the polygon K that surrounds it. Suppose without loss of generality that K is red. By applying the Fill procedure we have replaced the region K of Π with a new red rectilinear polygon K' with the same outer-hull as K. Notice that this increases neither the number of red or blue regions in Π, and decreases the number of holes by one. Moreover, it does not increase the outer-complexity of Π, as any blue region or red region which is not adjacent to H is unaffected, while any red region adjacent to H gets merged with K' and thus does not contribute to the outer-complexity of Π any longer. The red and blue regions of Π are still disjoint and still correctly cover all points of S by construction.

Repeatedly apply Fill as long as there are holes in Π. This procedure terminates as Π starts out with a finite number of holes and every application of Fill decreases the number of holes. Moreover, because Π no longer has holes by the end of the procedure, every hole of every region in Π must coincide with the outer-hull of some other region. On the other hand, the outer-hull of any region can coincide with at most one such hole. Recall that k is the number of boxes we started with in the original SBCC. Because Π starts out with at most k colored regions and Fill never increases the number of colored regions, Π still has at most k colored regions.

By Lemma 8, any colored region P of Π can be exactly covered with a number of boxes which is at most

$$\begin{aligned} &|P|/2 + |\text{holes}(P)| - 1 \\ &= |\text{oh}(P)|/2 + \sum_{H \in \text{holes}(P)} (|H|/2+1) \; - 1. \end{aligned}$$

Because every colored region in Π coincides with at most one hole of another colored region, and every hole in every colored region coincides with another colored region (more precisely, its outer-hull) by summing over all colored regions P we get a total number of boxes smaller than

$$\begin{aligned} &\sum_P |\text{oh}(P)|/2 + \sum_P (|\text{oh}(P)|/2+1) \\ &\leq \sum_P (|\text{oh}(P)|+1). \end{aligned}$$

Because Π contains at most k colored regions, this is at most $c+k$, where c is the outer-complexity of Π. By Lemma 7 and because the applications of Fill did not increase the outer-complexity of Π, we have $c \leq 8k$. Thus the claim holds. □

In [10], Mitchell showed the following:

Theorem 21 ([10]) *Given a set S of n points and a set of axis-aligned boxes $\mathcal{R}$, we can, in polynomial time, find a $O(1)$-approximation for the minimum cardinality set of interior-disjoint axis-aligned sub-boxes of $\mathcal{R}$ which cover all points in S.*

Using this result, we get our main theorem in this section:

Theorem 22 *There is a polynomial-time algorithm to approximate minimum simultaneous class cover within a constant factor.*

Proof. Consider some set of n red and blue points S. Call opt the minimum size of a SBCC of S and opt_{ind} the minimum size of a SBCC of S with interior-disjoint boxes.

Let $\mathcal{R}$ be the set of monochromatic boxes on S. We shrink these boxes appropriately to consider only boxes which have a point of S near every edge, so that the number of boxes in $\mathcal{R}$ is polynomial in n (and $\mathcal{R}$ can also be computed in $O(\text{poly}(n))$ time). Using Theorem

21 with this set $\mathcal{R}$, we can get a SBCC with interior-disjoint boxes (which is also a SBCC by definition) of size $k \in O(\text{opt}_{\text{ind}})$ in $O(\text{poly}(n))$ time. By Theorem 9 we have $\text{opt}_{\text{ind}} \leq 9 \cdot \text{opt}$. Thus $k \in O(\text{opt})$ and the claim holds. $\square$

6 Conclusion

In this paper we have given an alternative proof of the APX-hardness of the Boxes Class Cover problem which yields a more precise statement on the lower bound for a polynomial-time approximation factor (under the assumption that $\mathsf{P} \neq \mathsf{NP}$), and works as well for the case where half-strips are used instead of boxes. We have also explored the related Simultaneous Boxes Class Cover problem, giving proofs for a lower and upper bound which match up to a constant factor. These exhibit interesting connections with the Independent Sub-Box Cover problem studied by Mitchell [10]. The problem of finding a constant-factor approximation to the original BCC problem remains open. Perhaps the methods used here could help find better upper bounds for this problem, or match the existing upper-bounds by more elementary means (as current upper bounds rely on the computation of weighted ϵ-nets).

References

[1] R. Aschner, M. J. Katz, G. Morgenstern, and Y. Yuditsky. Approximation schemes for covering and packing. In S. K. Ghosh and T. Tokuyama, editors, *WALCOM: Algorithms and Computation*, pages 89–100, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[2] S. Bereg, S. Cabello, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, and I. Ventura. The class cover problem with boxes. *Comput. Geom.*, 45(7):294–304, 2012.

[3] P. Berman and B. DasGupta. Complexities of efficient solutions of rectilinear polygon cover problems. *Algorithmica*, 17(4):331–356, 1997.

[4] A. Cannon and L. Cowen. Approximation algorithms for the class cover problem. *Ann. Math. Artif. Intell.*, 40(3-4):215–224, 2004.

[5] M. Chlebík and J. Chlebíková. Inapproximability results for bounded variants of optimization problems. In A. Lingas and B. J. Nilsson, editors, *Fundamentals of Computation Theory, 14th International Symposium, FCT 2003, Malmö, Sweden*, volume 2751 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2003.

[6] L. J. Cowen and C. E. Priebe. Randomized nonlinear projections uncover high-dimensional structure. *Advances in Applied Mathematics*, 19(3):319–331, 1997.

[7] J. G. DeVinney. *The class cover problem and its application in pattern recognition.* PhD thesis, Johns Hopkins University, 2003.

[8] D. Eppstein. Graph-theoretic solutions to computational geometry problems. In C. Paul and M. Habib, editors, *Graph-Theoretic Concepts in Computer Science, 35th International Workshop, WG 2009, Montpellier, France*, volume 5911, pages 1–16, 2009.

[9] S. Hajiaghaei Shanjani. Hardness of approximation for red-blue covering. In *Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada*, 2020.

[10] J. Mitchell. Approximation algorithms for geometric separation problems. Technical report, Dept. of Applied Math. and Statistics, State University of New York at Stony Brook, 1993.

[11] J. O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21(1):118–130, 1983.

[12] C. E. Priebe, D. J. Marchette, J. DeVinney, and D. A. Socolinsky. Classification using class cover catch digraphs. *J. Classif.*, 20(1):003–023, 2003.

Efficiently Stabbing Convex Polygons and Variants of the Hadwiger-Debrunner (p, q)-Theorem.[*]

Justin Dallant [†] Patrick Schnider [‡]

Abstract

Hadwiger and Debrunner showed that for families of convex sets in $\mathbb{R}^d$ with the property that among any p of them some q have a common point, the whole family can be stabbed with $p-q+1$ points if $p \geq q \geq d+1$ and $(d-1)p < d(q-1)$. This generalizes a classical result by Helly. We show how such a stabbing set can be computed for a family of convex polygons in the plane with a total of n vertices in $\mathcal{O}((p-q+1)n^{4/3}\log^8 n(\log\log n)^{1/3} + np^2)$ expected time. For polyhedra in $\mathbb{R}^3$, we get an algorithm running in $\mathcal{O}((p-q+1)n^{5/2}\log^{10} n(\log\log n)^{1/6} + np^3)$ expected time. We also investigate other conditions on convex polygons for which our algorithm can find a fixed number of points stabbing them. Finally, we show that analogous results of the Hadwiger and Debrunner (p, q)-theorem hold in other settings, such as convex sets in $\mathbb{R}^d \times \mathbb{Z}^k$ or abstract convex geometries.

1 Introduction

A classical result in convex geometry by Helly [18] states that if a family of convex sets in $\mathbb{R}^d$ is such that any $d+1$ sets have a common intersection, then all sets do. In 1957, Hadwiger and Debrunner [15] considered a generalization of this setting. Let $\mathcal{F}$ be a family of sets in $\mathbb{R}^d$ and let $p \geq q \geq d+1$ be integers. We say that $\mathcal{F}$ has the (p, q)-property if $|\mathcal{F}| \geq p$ and for every choice of p sets in $\mathcal{F}$ there exist q among them which have a common intersection. We further say that a set of points S stabs $\mathcal{F}$ if every set in $\mathcal{F}$ contains at least one point from S. Then the following holds.

[*]A full version of the paper is available at `https://arxiv.org/abs/2002.06947`. A preliminary version was presented at EuroCG 2020, University of Würzburg.

[†]Department of Computer Science, Université libre de Bruxelles, `justin.dallant@ulb.be`. This work was partially supported by the French Community of Belgium via the funding of a FRIA grant.

[‡]Department of Mathematical Sciences, University of Copenhagen, `ps@math.ku.dk`. Has received funding from the European Research Council under the European Unions Seventh Framework Programme ERC Grant agreement ERC StG 716424 - CASe. Part of this work was done when the author was employed at ETH Zürich.

Theorem 1 (Hadwiger and Debrunner [15]) *Let $d \geq 1$ be an integer. Let p and q be integers such that $p \geq q \geq d+1$ and $(d-1)p < d(q-1)$, and let $\mathcal{F}$ be a finite family of convex sets in $\mathbb{R}^d$. Suppose that $\mathcal{F}$ has the (p, q)-property. Then there exist $p-q+1$ points in $\mathbb{R}^d$ stabbing $\mathcal{F}$.*

Note that the bound on the number of points needed is tight. That is, for every $p \geq q \geq d+1$ there exist families of convex sets with the (p, q)-property where at least $p-q+1$ points are needed to stab the whole family. This is easily seen by considering any family of $p-q+1$ disjoint convex sets where one of them is taken with multiplicity q. It is also known that whenever $q \leq d$, there exist families of convex sets with the (p, q)-property where arbitrary large number of points are needed. This can be seen by taking n hyperplanes in general position in $\mathbb{R}^d$ (meaning that no two hyperplanes are parallel and no $d+1$ hyperplanes intersect at the same point). Then any d hyperplanes intersect at some point (in other words, they have the (d, d)-property) and any single point stabs at most d hyperplanes. Thus, at least $\lfloor n/d \rfloor$ points are necessary to stab all hyperplanes.

Many related results have since been established. Among the most famous is one from Alon and Kleitman [3] who in 1992 proved that for any $p \geq q \geq d+1$, there exists a finite upper bound on the maximum number of points needed to stab a family of convex sets with the (p, q)-property. However, all the known upper bounds are probably far from being tight in the general case. As an example, for $(p, q, d) = (4, 3, 2)$, their proof yields an upper bound of 4032 (while the best known lower bound is 3). Since then, this number has been proven to lie between 3 and 13 (inclusive) [22]. A further improvement to the upper bound from 13 to 9 by McGinnis has recently appeared as a preprint [26]. Still, the only values of $p \geq q \geq d+1$ for which exact bounds are known are those corresponding to Theorem 1. There is a lot of work in this more general setting, both improving the bounds (e.g. [21]) as well as adapting to generalizations of convex sets (e.g. [20, 29]), and it is an interesting open problem to study algorithmic questions connected to these results.

Special cases where some further restrictions are imposed on the considered sets have also led to interesting results. One much studied example is obtained by considering only axis-aligned boxes in $\mathbb{R}^d$. In this case, we can already start by strengthening the result given by Helly's theorem, as for a family of axis-aligned boxes in $\mathbb{R}^d$, if all pairs intersect then the whole family intersect. As is expected, this additional structure leads to stronger (p, q) results. One early result by Hadwiger and Debrunner [16] is the following (notice the weaker conditions on p and q and the independence on d).

Theorem 2 ([16]) *Let $d \geq 1$ be an integer. Let p and q be integers such that $2q - 2 \geq p \geq q \geq 2$ and let $\mathcal{F}$ be a finite family of axis-aligned boxes in $\mathbb{R}^d$. Suppose that $\mathcal{F}$ has the (p, q)-property. Then there exist $p-q+1$ points in $\mathbb{R}^d$ stabbing $\mathcal{F}$.*

Another example is when all sets are translations either with or without scaling of some convex set K. Here, strong results exist only for some very simple cases such as K being a d-dimensional cube or ball. For example the maximum number of points needed to stab families of discs in the plane with the $(p, 2)$-property lies between $4p - 4$ and $7p - 10$ inclusive [32]. These bounds are tight for $p = 2$, that is in the case of pairwise intersecting discs (which was already shown by Stachó [31] and Danzer [10]).

From an algorithmic point of view, little work seems to have been done towards computing these stabbing points. One instance which has recently received some attention is the aforementioned case of pairwise intersecting discs in the plane. Har-Peled et al. [17] showed how such a family can be stabbed with 5 points in linear time (which is one more point than the theoretical optimum). Shortly after another paper, yet to be formally published, claimed to find a linear time algorithm for stabbing such a family with only 4 points [6]. However, the computation of small stabbing sets for families of general convex polyhedra with the (p, q)-property seems to not have been studied and will constitute one part of this paper, in the setting of Theorem 1.

For a great overview of the studied questions and known results around (p, q) problems, we refer the interested reader to the 2003 survey by Eckhoff [12].

Before continuing, we would also like to mention that Helly's theorem has been generalized to many other settings, as this will come in play in the second part of this paper. In general, we say that a set system has *Helly number h* if the following holds: if any h sets in the set system have a common intersection, then the whole set system does. Helly numbers have been shown to exist for many set systems, such as convex sets in $\mathbb{R}^d \times \mathbb{Z}^k$ [4, 19] or abstract convex geometries [13, 23], which include subtrees of trees and ideals of posets. In many cases, the proofs can be adapted to show a constant stabbing number analogous to the result by Alon and Kleitman. In this work, we show that under some weak conditions, the existence of a Helly number implies a tight Hadwiger-Debrunner type result.

Some details have been left out of this paper, and are included in the full version [9].

2 Stabbing convex polytopes

2.1 A proof of the Hadwiger-Debrunner (p, q)-theorem

We will first consider a proof of Theorem 1 which will naturally lead to an algorithm for finding stabbing points. In a book by Matoušek [25], the proof of this theorem is left as an exercise, yet the hint suggests that the intended solution is close to the proof below. The main differences with other proofs for this theorem are that it is more constructive and does not make use of a separating hyperplane, which will make it easier to adapt to other settings later on.

The proof makes use of a lemma which can also be found in the same book. For a non-empty compact set S, let $\operatorname{lexmin}(S)$ denote its lexicographical minimum point. Then we have the following.

Lemma 3 ([25, Lemma 8.1.2]) *Let $\mathcal{F}$ be a family of at least $d + 1$ compact convex sets in $\mathbb{R}^d$, such that $I := \bigcap \mathcal{F}$ is non-empty. Let $x := \operatorname{lexmin}(I)$. Then, there exists a subfamily $\mathcal{H} \subset \mathcal{F}$ of size d such that $x = \operatorname{lexmin}(\bigcap \mathcal{H})$.*

We now sketch the idea of the proof of the Hadwiger-Debrunner theorem (see the full version of this paper [9] for a complete proof).

Proof. [Proof idea of the Hadwiger-Debrunner theorem] Call a pair of integers (p, q) *admissible* if $p \geq q \geq d+1$ and $(d-1)p < d(q-1)$. Let (p, q) be an admissible pair, and let $\mathcal{F}$ be a family of compact convex sets in $\mathbb{R}^d$ with the (p, q)-property. Construct a point $x^*(\mathcal{F})$ defined as the lexicographically maximum point among all lexicographically minimum points in the intersection of d sets in $\mathcal{F}$. We choose it as one of our stabbing points and remove all sets stabbed by $x^*(\mathcal{F})$. Using Lemma 3 it can be shown that the remaining sets either have the $(p - d, q - d + 1)$-property, where $(p - d, q - d + 1)$ is admissible, or consist of $p - q + k$ sets, $k < q - d$, where some $k+1$ have a common intersection. In the first case, we can continue inductively, in the second case we can trivially stab the remaining sets using $p - q$ points. □

This proof naturally leads to an algorithm. In the following we will assume that the convex sets are convex polytopes described as intersections of a total of n halfspaces in general position. The dimension d is constant. As testing if convex polytopes intersect can be done in linear time in the number of defining halfspaces (using linear programming in constant dimension), computing $x^*(\mathcal{F})$ can be done in $\mathcal{O}(n^d)$ time by computing all d-wise intersections (see the proof of Lemma 15). This needs to be computed at most $p-q+1$ times. For the case where there are only $p-q+k$ sets, using Lemma 3 together with the fact that $p-q+k<p-d$ we can compute a point stabbing $k+1$ sets in $\mathcal{O}(np^d)$ time (or $\mathcal{O}(p^{d+1})$ if all polyhedra are of at most constant size). We can then easily choose $p-q-1$ points in $\mathcal{O}(n)$ time to stab the remaining $p-q-1$ sets. Thus, in the plane, we get a total runtime of $\mathcal{O}((p-q+1)n^2+np^2)$. In the next section, we will show how to get a subquadratic runtime with respect to n.

Note also that for $d \leq 3$ we can get the vertex representation of our polytopes from the halfspace representation in $O(n \log n)$ time by computing the convex hulls of the dual point sets. In what follows we will assume that we have access to the vertices and edges of our polygons as the $O(n \log n)$ overhead will be dominated by the rest of our algorithms.

2.2 A more efficient algorithm for the planar case

In this whole section, the family $\mathcal{F}$ consists of compact convex polygons with a total of n (distinct) vertices in the plane. We further assume that $\mathcal{F}$ has the (p,q)-property, for some admissible pair (p,q). For the sake of simplicity, we will assume that the lines defining the polygon edges are in general position, non-vertical and that all points defined as the lexicographical minimum in the intersection of a pair of sets have different x-coordinates. With these assumption the lexicographical minimum in a polygon (or intersection of polygons) is the leftmost point.

We break down the computation of $x^*(\mathcal{F})$ into two parts. Consider two intersecting polygons P_1 and P_2. The point x which is the leftmost of $P_1 \cap P_2$ can be of one of two types. Either (Case 1) x is the leftmost point of P_1 (resp. P_2) and is contained in the interior of P_2 (resp. P_1) or (Case 2) x is the proper intersection of an upper-hull edge e_u of P_1 (resp. P_2) and a lower-hull edge e_ℓ of P_2 (resp. P_1) with the following property: the outward facing normal vectors of e_u and e_ℓ form a (counter-clockwise orientated) angle of less than 180 degrees. Reciprocally, an upper-hull and a lower-hull edge which intersect with this property define the leftmost point of an intersection of two polygons.

We define $x_1^*(\mathcal{F})$ to be the rightmost point among all pairs of intersecting polygons in $\mathcal{F}$ corresponding to the first case (or $x_1^*(\mathcal{F}) = (-\infty, \infty)$ if there is no such pair), and similarly for $x_2^*(\mathcal{F})$ and the second case. Then $x^*(\mathcal{F})$ is the rightmost point of $\{x_1^*(\mathcal{F}), x_2^*(\mathcal{F})\}$.

We will use the following result, which can be obtained by an adaptation of a proof by Matoušek [24, Theorem 6.2] with the halfspace partition tree construction from Chan [8] (see the full version of this paper [9]). This theorem essentially states that there are efficient range-counting data structures for ranges defined as the conjunction of half-spaces on different liftings of a point set.

Theorem 4 *Let S be a set of n objects, k a constant, and $\phi_1, \phi_2, \ldots, \phi_k$ mappings from S to $\mathbb{R}^d$. Let ϕ_S map k-tuples of halfspaces $H_1, H_2, \ldots, H_k$ of $\mathbb{R}^d$ to the set*

$$\phi_S(H_1, H_2, \ldots, H_k) :=$$
$$\{s \in S \mid \phi_1(s) \in H_1, \phi_2(s) \in H_2 \ldots, \phi_k(s) \in H_k\}.$$

Suppose we have computed the point sets $\phi_1(S), \ldots, \phi_k(S)$ and let $n \leq m \leq n/\log^{\omega(1)} n$. Then we can preprocess the point sets in $\mathcal{O}(n \log^k n + m)$ time such that $|\phi_S(H_1, H_2, \ldots, H_k)|$ can be computed in $\mathcal{O}((n/m^{1/d})(\log n)^{2(k+(k-d-1)/d)}(\log\log n)^{1/d})$ expected time for any k-tuple of halfspaces.

We can use this to prove the following.

Lemma 5 *The point $x_1^*(\mathcal{F})$ can be computed in $\mathcal{O}(n^{4/3}\log^4 n(\log\log n)^{1/3})$ expected time.*

Proof. To compute $x_1^*(\mathcal{F})$, we can test for each polygon if its leftmost point is contained in the interior of another, and keep the rightmost point among those which are. We triangulate all polygons, so that this reduces to testing, for each of the $\mathcal{O}(n)$ leftmost points, if it is in the interior of one of the $\mathcal{O}(n)$ triangles. In the dual plane, this can be expressed as the composition of three half-plane range queries. Using Theorem 4 with $d=2$, $k=3$, and $m = n^{4/3}\log^4 n(\log\log n)^{1/3}$, we can thus preprocess the $\mathcal{O}(n)$ triangles in $\mathcal{O}(m)$ time such that counting how many triangles contain a particular point can be done in $\mathcal{O}(n^{1/3}\log^4 n(\log\log n)^{1/3})$ expected time. By querying all points we get the result. □

It remains to see how to compute $x_2^*(\mathcal{F})$ in subquadratic time. For this we use a a simple but remarkably powerful technique discovered by Chan [7], which reduces many optimization problems to the corresponding decision problem, with no blow-up in expected runtime.

Lemma 6 *Let $\alpha < 1$ and r be fixed constants. Let $f : \mathcal{P} \to \mathcal{Q}$ be a function that maps inputs to values in a totally ordered set (where elements can be compared in constant time) with the following properties:*

1. *For any input $P \in \mathcal{P}$ of constant size, $f(P)$ can be computed in constant time.*

2. *For any input $P \in \mathcal{P}$ of size n, we can construct inputs $P_1, \ldots, P_r \in \mathcal{P}$ each of size at most $\lceil \alpha n \rceil$ in time $T(n)$, such that $f(P) = \max\{f(P_1), \ldots, f(P_r)\}$.*

3. *For any input $P \in \mathcal{P}$ of size n and any $t \in \mathcal{Q}$, we can decide $f(P) \leq t$ in time $T(n)$.*

Then for any input $P \in \mathcal{P}$, we can compute $f(P)$ in $\mathcal{O}(T(n))$ expected time, assuming that $T(n)/n^{\epsilon}$ is monotone increasing for some constant $\epsilon > 0$.

We can apply this technique to the computation of x_2^*. Here, each $P \in \mathcal{P}$ is a set of edges which are oriented depending on which side the polygon it bounds lies on, $\mathcal{Q}$ is the plane with lexicographical order, and $f(P)$ is x_2^* (we abuse notation slightly by using x_2^* both for sets of oriented edges and sets of polygons). We make the following observations.

- For any constant-size set $\mathcal{E}$ of oriented edges, $x_2^*(\mathcal{E})$ can be computed in constant time. This verifies Property 1.
- For any family $\mathcal{E}$ of n oriented edges, we can partition it into 3 disjoint subfamilies S_1, S_2, S_3 of size between $\lfloor n/3 \rfloor$ and $\lceil n/3 \rceil$ each. Then, let $\mathcal{E}_1 := S_2 \cup S_3$, $\mathcal{E}_2 := S_1 \cup S_3$ and $\mathcal{E}_3 := S_1 \cup S_2$. Every set $\mathcal{E}_i$ is of size $|\mathcal{E}_i| \leq \lceil 2n/3 \rceil$. Thus, $x_2^*(\mathcal{E})$ is the rightmost point among $\{x_2^*(\mathcal{E}_1), x_2^*(\mathcal{E}_2), x_2^*(\mathcal{E}_3)\}$. These families can be constructed in $\mathcal{O}(n)$ time. This verifies Property 2, assuming $T(n) \geq \Omega(n)$ (which it will be).

Thus, in order to apply Chan's framework, it remains to decide $x_2^*(\mathcal{E}) \leq_{lex} t$ quickly.

Lemma 7 *For any point t in the plane and a set of n oriented edges $\mathcal{E}$, we can decide $x_2^*(\mathcal{E}) \leq_{lex} t$ in $\mathcal{O}(n^{4/3} \log^8 n (\log \log n)^{1/3})$ expected time.*

Proof. We can rephrase $x_2^*(\mathcal{E}) \leq_{lex} t$ as deciding whether there exist two oriented edges in $\mathcal{E}$ which intersect at an appropriate angle to the right of the vertical line ℓ passing through t. Thus we start by discarding all the (parts of) segments in $\mathcal{E}$ which lie to the left of ℓ. We then want to preprocess the $\mathcal{O}(n)$ segments corresponding to upper-hull edges (i.e. those with an outward facing normal pointing up) such that for any lower-hull edge e_ℓ we can detect if there is an upper-hull edge which intersects it at an appropriate angle quickly. Finding the edges which intersect a query edge e_ℓ at the appropriate angle can be expressed as the composition of 5 half-plane queries on different 2D liftings of the upper-hull edges. We can again use Theorem 4 as we did for x_1^*, this time with $k = 5$, to query all lower-hull edges in $\mathcal{O}(n^{4/3} \log^8 n (\log \log n)^{1/3})$ expected total time. □

We can thus use Lemma 6 to compute $x_2^*(\mathcal{F})$ in the same asymptotic expected time. Note that Hopcroft's problem reduces to computing $x_2^*(\mathcal{E})$ for a general set of oriented edges $\mathcal{E}$, and thus this runtime is likely close to optimal (see the lower bound by Erickson [14] in a quite general model of computation). Putting everything together we get the following.

Theorem 8 *Let (p, q) be an admissible pair for $d = 2$ and let $\mathcal{F}$ be a family of compact convex polygons in the plane with a total of n vertices and the (p, q)-property. Then we can compute a set of at most $p - q + 1$ points stabbing $\mathcal{F}$ in $\mathcal{O}((p-q+1)n^{4/3} \log^8 n (\log \log n)^{1/3} + np^2)$ expected time.*

If the case where every polygon has at most a constant number of vertices, then there is a simpler way to use Lemma 6 together with a Theorem by Agarwal et al. [1, Theorem 2.7] to compute $x^*(\mathcal{F})$ in $\mathcal{O}(n^{4/3} \log^{2+\epsilon} n)$ expected time, yielding a slightly faster algorithm for our problem.

2.3 The 3D case

Here we deal with the analogous case for the 3D polyhedra. In this case the Helly number becomes 4, and $x^*(\mathcal{F})$ is defined in terms of triplets of convex polyhedra with non-empty common intersection.

Theorem 9 *Let (p, q) be an admissible pair for $d = 3$ and let $\mathcal{F}$ be a family of compact convex polyhedra in $\mathbb{R}^3$ with a total of n vertices and the (p, q)-property. We can compute a set of at most $p - q + 1$ points stabbing $\mathcal{F}$ in $\mathcal{O}((p - q + 1)n^{5/2} \log^{10} n (\log \log n)^{1/6} + np^3)$ expected time.*

Proof. First, we compute the family $\mathcal{F}_2$ of all polyhedra obtained as pair-wise intersections of polyhedra in $\mathcal{F}$. This can be done in $\mathcal{O}(n^2)$ time using a linear-time algorithm to compute each intersection (for example the one by Chan [8]). Assuming the planes defining the polyhedra are in general position and none of the edges lie in a plane parallel to the yz-plane, the leftmost point in the intersection of three polyhedra in $\mathcal{F}$ is either

1. the leftmost point of a polyhedron in $\mathcal{F}_2$ contained in the interior of a polyhedron of $\mathcal{F}$,
2. the leftmost point of a polyhedron in $\mathcal{F}$ contained in the interior of a polyhedron of $\mathcal{F}_2$,
3. the intersection of an edge of a polyhedron in $\mathcal{F}_2$ and the interior of a facet of a polyhedron in $\mathcal{F}$ (note that not all such intersections define the leftmost point of the intersection of three polyhedra in $\mathcal{F}$).

The rightmost point corresponding to the first two cases can be found in expected time $\mathcal{O}(n^{9/4} \log^{\mathcal{O}(1)} n)$ by triangulating all polyhedra and then using the same methods as for the 2D case. We now focus on the third case. In what follows, we only deal with triangular facets, as the general case reduces to this one by triangulating all facets in $\mathcal{O}(n^2)$ total time. We will preprocess the triangles and then query each edge to count the number of triangles which intersect it and define the leftmost point of a three-wise intersection of polyhedra. Testing if a segment intersects a triangle in $\mathbb{R}^3$ can be done by comparing the signs of three polynomial functions of degree three on the coordinates of the points [30]. It can be checked that each of these tests can be linearized by liftings which map the triangles to points in 5 dimension. Agarwal et al. [1] further showed that when given an edge e of a polyhedron and a facet f of another polyhedron such that e and f intersect, testing if $e \cap f$ is the leftmost point of the intersection of the corresponding polyhedra can be expressed as testing if the outward normal vector of f lies in a the intersection of three halfspaces.

Using again Theorem 4, this time in dimension $d = 5$ and with $k = 6$, we can preprocess the $\mathcal{O}(n^2)$ facets of $\mathcal{F}_2$ in $\mathcal{O}(n^{5/2} \log^{10} n(\log\log n)^{1/6})$ time such that we can query any oriented edge in $\mathcal{O}(n^{3/2} \log^{10} n(\log\log n)^{1/6})$ expected time. By querying all $\mathcal{O}(n)$ edges in $\mathcal{F}$ we can detect a leftmost point in the intersection of three polyhedra in $\mathcal{F}$ corresponding to the third case exists in $\mathcal{O}(n^{5/2} \log^{10} n(\log\log n)^{1/6})$ total expected time. By applying Lemma 6 as we did in the planar case, we thus get the result. $\square$

2.4 Other conditions

We investigate two further conditions that are sufficient for a family of convex polygons to be stabbed by a fixed number of points and for which our algorithm can also be used.

The first condition we investigate considers *holes* in the union of sets. Let $\mathcal{F}$ be a finite family of convex sets in the plane and let $A \subset \mathbb{R}^2$ be the union of the sets in $\mathcal{F}$. A hole is a bounded connected component of $\mathbb{R}^2 \setminus A$.

There is an equivalent formulation of Helly's theorem due to Breen, which in the plane can be stated as follows: let $\mathcal{F}$ be a finite family of pairwise intersecting convex sets in the plane with the property that the union of any three of them has no hole, then $\mathcal{F}$ can be stabbed by a single point [5, 27]. We prove the following generalization of this result.

Theorem 10 *Let $\mathcal{F}$ be a finite family of pairwise intersecting convex sets in the plane with the property that the union of any $k+3$ of them has at most k holes, then $\mathcal{F}$ can be stabbed by $k+1$ points. Further, the $k+1$ stabbing points can be chosen to lie on a single line.*

Proof. Recall that $x^*(\mathcal{F})$ denotes the lexicographically maximum point among all lexicographically minimum points in the intersection of two sets in $\mathcal{F}$. Call F_1 and F_2 two sets in $\mathcal{F}$ such that the lexicographically minimum point of their intersection is $x^*(\mathcal{F})$. Consider the vertical line v through $x^*(\mathcal{F})$ and let F_1' and F_2' be the parts of F_1 and F_2, respectively, that lie to the left of v. Let now ℓ be a line through $x^*(\mathcal{F})$ which separates F_1' and F_2'. Such a line exists as otherwise $x^*(\mathcal{F})$ would not be the lexicographical minimum in the intersection of F_1' and F_2'. Further note that any set in $\mathcal{F}$ that is not stabbed by $x^*(\mathcal{F})$ must intersect ℓ to the left of $x^*(\mathcal{F})$: there cannot be intersections exclusively to the right of $x^*(\mathcal{F})$ by its definition, and as any set intersects F_1 and F_2, it follows from convexity that it must also intersect ℓ.

Let now $\mathcal{R}$ be the family of remaining sets, that is, the sets not stabbed by $x^*(\mathcal{F})$. We claim that among any $k+1$ of them, some two intersect along ℓ. Indeed, if there were $k+1$ sets whose intersections with ℓ are pairwise disjoint, the union of these sets with F_1 and F_2 would have $k+1$ holes, which is excluded by the assumptions of the theorem. We can thus apply the Hadwiger-Debrunner (p, q)-theorem on ℓ to stab $\mathcal{R}$ with k points, so in total we have stabbed $\mathcal{F}$ with $k+1$ collinear points. $\square$

Note that opposed to the proof of the Hadwiger-Debrunner (p, q)-theorem, we only compute $x^*(\mathcal{F})$ once. After this, we only need the 1D-variant, where stabbing points of n intervals can easily be computed in time $O(n \log n)$. We thus get the following:

Proposition 11 *Let $\mathcal{F}$ be a family of compact convex polygons in the plane with a total of n vertices and with the property that the union of any $k+3$ of them has at most k holes. We can compute a set of at most $k+1$ collinear points stabbing $\mathcal{F}$ in $\mathcal{O}(n^{4/3} \log^8 n(\log\log n)^{1/3})$ expected time.*

Another result due to Montejano and Soberón [28] and which admits a similar proof in the plane, yields the following:

Proposition 12 *Let p, q, r be integers with $p > q > 2$ and $r > \binom{p}{q} - \binom{p-1}{q-1}$. Let $\mathcal{F}$ be a family of compact convex polygons in the plane with a total of n vertices and with the property that for any p of them at least r of their q-tuples intersect. We can compute a set of at most $p - q + 1$ collinear points stabbing $\mathcal{F}$ in $\mathcal{O}(n^{4/3} \log^8 n(\log\log n)^{1/3})$ expected time.*

3 Other Hadwiger-Debrunner type results

3.1 Ordered-Helly systems

By taking a close look at the proof for the Hadwiger-Debrunner (p, q)-theorem, we can observe that we made use of relatively few properties of compact convex sets. These properties are (i) closure under intersection, (ii) existence of a lexicographically minimum point, (iii) Helly's theorem as well as (iv) the fact that the set of all points lexicographically smaller than some point y is convex (this last property doesn't appear explicitly but is needed in the proof of Lemma 3). We define *Ordered-Helly systems* as set systems with analogous properties.

Definition 1 (Ordered-Helly system)
An Ordered-Helly system $\mathfrak{S}$ is a tuple $(\mathcal{B}, \mathcal{C}, \mathcal{D}, h, \preceq)$ consisting of

- *a set $\mathcal{B}$, called the* base-set*;*
- *a family $\mathcal{C}$ of subsets of $\mathcal{B}$, whose members are called* convex sets *or* $\mathfrak{S}$-convex sets*;*
- *a family $\mathcal{D} \subset \mathcal{C}$ whose members are called* compact sets *or* $\mathfrak{S}$-compact sets*;*
- *a total order $\preceq$ on $\mathcal{B}$;*
- *and an integer $h \geq 2$, called the* Helly-number *of $\mathfrak{S}$*

with the following properties.

1. *(Intersection closure)*
   $\mathcal{D}$ is closed under intersection, i.e. for all $S_1, S_2 \in \mathcal{D}$ we have $S_1 \cap S_2 \in \mathcal{D}$.
2. *(Attainable minimum)*
   For all non-empty $S \in \mathcal{D}$, there exists $x \in S$ such that for all $y \in S$, $x \preceq y$. This x is necessarily unique and we call x the $\preceq$-min of S. We define the $\preceq$-max of a set similarly, if it exists.
3. *(Convex order)*
   For all $t \in \mathcal{B}$, we have $\{x \in \mathcal{B} \mid x \preceq t \text{ and } x \neq t\} \in \mathcal{C}$.
4. *(Helly property)*
   If $\mathcal{F} \subset \mathcal{C}$ is a finite subset of $n \geq h$ sets of $\mathcal{C}$ such that every subfamily of h members of $\mathcal{F}$ has a non-empty common intersection, then all members of $\mathcal{F}$ have a non-empty common intersection.

As was stated earlier, this structure is enough to carry out a similar proof as the one we saw for the Hadwiger-Debrunner theorem. Call a pair (p, q) of integers h-admissible if $p \geq q \geq h$ and $(h-2)p < (h-1)(q-1)$. Then we have the following.

Theorem 13 *Let $\mathfrak{S} = (\mathcal{B}, \mathcal{C}, \mathcal{D}, h, \preceq)$ be an Ordered-Helly system. Let (p, q) be an h-admissible pair of integers. Let $\mathcal{F}$ be a finite family of non-empty sets of $\mathcal{D}$. Suppose that $\mathcal{F}$ has the (p, q)-property. Then there exist $p - q + 1$ elements of $\mathcal{B}$ stabbing $\mathcal{F}$.*

Note that the existence of a Helly number alone is not enough to show such a result. Alon et al. [2] give an example of a set system with Helly number 2 but no general (p, q)-theorem.

The proof of this theorem makes use of an analogue to Lemma 3. Let us state and prove this analogous lemma.

Lemma 14 *Let $\mathfrak{S} = (\mathcal{B}, \mathcal{C}, \mathcal{D}, h, \preceq)$ be an Ordered-Helly system. Let $\mathcal{F} \subset \mathcal{D}$ be a family of $n \geq h$ sets in $\mathcal{D}$ such that $I := \bigcap \mathcal{F}$ is non-empty. Let x be the $\preceq$-min of I (which exists by the properties of intersection closure and attainable minimum). Then, there exists a subfamily $\mathcal{G} \subset \mathcal{F}$ of size $h-1$ such that x is the $\preceq$-min of $\bigcap \mathcal{G}$.*

Proof. Let $\mathcal{F}$, I and x be as specified in the statement. Let S_x denote $\{y \in \mathcal{B} \mid y \preceq x \text{ and } y \neq x\}$, which is a $\mathfrak{S}$-convex set by the property of convex order. It is disjoint from I as $\preceq$ is a total order. By the Helly property, there exists a subfamily of h members of $\mathcal{F} \cup \{S_x\}$ with an empty common intersection. These members have to include S_x, as all members of $\mathcal{F}$ have a non-empty common intersection. Let $\mathcal{G} \subset \mathcal{F}$ be the remaining $h-1$ sets and let $x_\mathcal{G}$ be the $\preceq$-min of $I' := \bigcap \mathcal{G}$ (which is a non-empty $\mathfrak{S}$-compact set). We know that $x_\mathcal{G} \preceq x$ because $x \in I'$. If we now suppose $x_\mathcal{G} \neq x$ this implies that $x_\mathcal{G} \in S_x$ and contradicts the fact that $I' \cap S_x = \emptyset$. Thus, $x_\mathcal{G} = x$. $\square$

The proof of Theorem 13 is now analogous to the proof of the Hadwiger-Debrunner (p, q)-theorem sketched above. This once again leads to an algorithm computing stabbing elements of a family of $\mathfrak{S}$-compact sets with the (p, q)-property for an admissible pair (p, q), given we have access to some oracles. We will write the run-times in terms of the description complexity of a set, which depends on the exact context. Thus, for a $\mathfrak{S}$-compact set S, let $\#S$ denote this complexity (of at least 1), and for a family $\mathcal{F}$ of $\mathfrak{S}$-compact sets, let $\#\mathcal{F} := \sum_{S \in \mathcal{F}} \#S$.

Consider an Ordered-Helly system $\mathfrak{S} = (\mathcal{B}, \mathcal{C}, \mathcal{D}, h, \preceq)$ (for a constant h) and suppose we have access to the following oracles.

- For two elements $b_1, b_2 \in \mathcal{B}$, we can test $b_1 \preceq b_2$ in constant time.
- For a family $\mathcal{F} \subset \mathcal{D}$ of at most $h-1$ $\mathfrak{S}$-compact sets, we can test if the sets in $\mathcal{F}$ have a common intersection and compute the $\preceq$-min of that intersection if it is non-empty in $\mathcal{O}(\#\mathcal{F})$ time.
- For a $\mathfrak{S}$-compact set $S \in \mathcal{D}$ and a point $b \in \mathcal{B}$ we can test if $b \in S$ in $\mathcal{O}(\#S)$ time.

We could naturally consider other run-times for these oracles. We only specify them in order to showcase an example of run-time analysis which is tighter than if we had worked with general run-times and swapped in concrete functions afterwards (and matches the case of convex polytopes in $\mathbb{R}^d$ for small d). Other run-times might require other specialized forms of analysis.

Now, let $\mathcal{F} \subset \mathcal{D}$ be a family of $\mathfrak{S}$-compact sets. Among all points in $\mathcal{B}$ defined as the $\preceq$-min of the intersection of $h-1$ sets in $\mathcal{F}$, let $b^*(\mathcal{F})$ be the $\preceq$-max of those.

Let us state two lemmas which will be useful for our algorithm.

Lemma 15 *We can compute $b^*(\mathcal{F})$ in $\mathcal{O}(\#\mathcal{F} \cdot |\mathcal{F}|^{h-2})$ time.*

Proof. We can compute $b^*(\mathcal{F})$ by testing for intersection in every subfamily $\mathcal{G}$ of $\mathcal{F}$ of size $h-1$ and computing the $\preceq$-min of that intersection if it is non-empty.

If we consider some fixed subfamily $\mathcal{G}$, the computation for that subfamily will cost at most $c \cdot \#\mathcal{G}$ for some constant c which doesn't depend on $\mathcal{G}$. Charge this cost to the sets $S \in \mathcal{G}$ by attributing a cost of $c \cdot \#S$ to a set S.

Now, consider the cost charged to some fixed set S for the whole computation. As S appears in no more than $|\mathcal{F}|^{h-2}$ subfamilies of size $h-1$, its total cost charge is upper bounded by $c \cdot \#S \cdot |\mathcal{F}|^{h-2}$. Summing across all sets $S \in \mathcal{F}$, we get a total cost of $\mathcal{O}(\#\mathcal{F} \cdot |\mathcal{F}|^{h-2})$. $\square$

Lemma 16 *Suppose there exists some subfamily $\mathcal{G} \subset \mathcal{F}$ of size $k+1$ such that all sets in $\mathcal{G}$ have a common intersection, where k is a known parameter. We can compute $|\mathcal{F}| - k$ points in $\mathcal{B}$ stabbing $\mathcal{F}$ in $\mathcal{O}(\#\mathcal{F} \cdot |\mathcal{F}|^{h-1})$ time.*

Proof. If $k+1 \leq h-1$, then we can test every subfamily of size $k+1$ for common intersection and compute its $\preceq$-min for a total cost of $\mathcal{O}(\#\mathcal{F} \cdot |\mathcal{F}|^{k+1}) \leq \mathcal{O}(\#\mathcal{F} \cdot |\mathcal{F}|^{h-1})$.

If $k+1 > h-1$, then we know from Lemma 14 that the $\preceq$-min of the intersection of all sets in $\mathcal{G}$ is also the $\preceq$-min of the intersection of some $h-1$ sets in $\mathcal{F}$. Thus, one can find a point stabbing at least $k+1$ sets by computing the $\preceq$-min point for each subfamily of size $h-1$ (in $\mathcal{O}(\#\mathcal{F}|\mathcal{F}|^{h-1})$ time) and counting the number of sets intersected for each of the $\mathcal{O}(|\mathcal{F}|^{h-1})$ computed points (in $\mathcal{O}(\#\mathcal{F}|\mathcal{F}|^{h-1})$ time as well).

As soon as we find a point b stabbing at least $k+1$ sets, we return b along with the $\preceq$-min of every set in $\mathcal{F}$ which is not stabbed by b. $\square$

With these algorithms, a similar proof to the Euclidean case gives the following.

Theorem 17 *Let $\mathcal{F}$ be a family of $\mathfrak{S}$-compact sets with the (p,q)-property. Suppose we have access to the relevant oracles described above. We can compute a set of at most $p-q+1$ elements stabbing $\mathcal{F}$ in time*

$$\mathcal{O}((p-q+1)(\#\mathcal{F})^{h-1} + (\#\mathcal{F})p^{h-1}).$$

With access to the right oracle, we could for example apply Lemma 6 analogously to what we did for convex polytopes in the Euclidean setting and get the corresponding speedup.

3.2 Examples of Ordered-Helly systems

Until now, the only Ordered-Helly system we have seen is the one corresponding to compact convex sets in $\mathbb{R}^d$. We will see that this structure does have some other interesting representatives and is not restricted to this single example (in which case the usefulness of introducing it would have been doubtful).

Let us start by stating and proving some Hadwiger-Debrunner type results for sets which are defined as the intersection of a compact convex set in $\mathbb{R}^d$ with a subset $S \in \mathbb{R}$. To do so define the S-Helly number as follows.

Definition 2 *Let S be a subset of $\mathbb{R}^d$. The S-Helly number, denoted by $h(S)$, is the smallest integer $k > 0$ such that the following holds:*
Given a finite family $\mathcal{F}$ of convex sets in $\mathbb{R}^d$, if in every subfamily of $\mathcal{F}$ of size k all sets share a point in S, then all sets in $\mathcal{F}$ share a point in S.
If no such k exists, then $h(S) = \infty$.

One of the first results concerning S-Helly numbers was discovered by Doignon [11], and is the case $S = \mathbb{Z}^d$.

Theorem 18 (Doignon) *Let $\mathcal{F}$ be a family of $n \geq 2^d$ convex sets in $\mathbb{R}^d$. If in every subfamily of $\mathcal{F}$ of size 2^d all sets share a point in $\mathbb{Z}^d$, then all sets in $\mathcal{F}$ share a point in $\mathbb{Z}^d$.*

In a paper by Hoffman [19] a mixed-integer version of this theorem is stated, which generalizes both Helly's theorem and Doignon's version. It was later rediscovered and proved in detail by Averkov and Weismantel [4].

Theorem 19 (Mixed-Integer Helly) *Let $\mathcal{F}$ be a family of $n \geq (d+1)2^k$ convex sets in $\mathbb{R}^{d+k}$, where $d, k \geq 0$ and $d+k \geq 1$. If in every subfamily of $\mathcal{F}$ of size $(d+1)2^k$ all sets share a point in $\mathbb{R}^d \times \mathbb{Z}^k$, then all sets in $\mathcal{F}$ share a point in $\mathbb{R}^d \times \mathbb{Z}^k$.*

It is easy to see that we can get a corresponding Ordered-Helly system, and thus Theorem 13 immediately gives the following.

Theorem 20 (Mixed-Integer HD) *Let $d, k \geq 0$ be integers such that $d+k \geq 1$. Let (p,q) be a $(d+1)2^k$-admissible pair. Let $\mathcal{F}$ be a finite family of sets obtained as the intersection of $\mathbb{R}^d \times \mathbb{Z}^k$ with a compact convex set in $\mathbb{R}^{d+k}$. Suppose that $\mathcal{F}$ has the (p,q) property. Then there exist $p-q+1$ points in $\mathbb{R}^d \times \mathbb{Z}^k$ stabbing $\mathcal{F}$.*

More generally, every upper bound on an S-Helly number leads to the corresponding Hadwiger-Debrunner version if S is closed in $\mathbb{R}^d$. The corresponding algorithmic results also follow, provided we have access to the required oracles.

Let us now explore how the structure of Ordered-Helly systems relates to the structure of abstract convex geometries as introduced by Edelman and Jamison [13]. Convex geometries are an abstraction capturing the basic combinatorial structure of classical convexity in a similar manner to matroids capturing the basic combinatorial properties of linear independence. Convex geometries appear in many contexts outside of convex sets such as graph theory or order theory. We refer the interested reader to the book of Edelman and Jamison [13] or to Chapter III in the book of Korte et al. [23] for an in-depth overview. We will only go over the basic definitions and theorems needed for our purpose, which can all be found in the two sources we just mentioned.

For the following definitions, it is useful to imagine the operator τ as analogous to the convex hull operator on a point set.

Definition 3 *Consider some finite set E and a family $\mathcal{N}$ of subsets of E. Let τ be the operator defined on subsets of E as $\tau(A) = \bigcap\{X \mid A \subset X,\ X \in \mathcal{N}\}$. We say that $(E, \mathcal{N})$ is a convex geometry if it has the following properties:*

1. *$\emptyset \in \mathcal{N}$, $E \in \mathcal{N}$.*
2. *$X, Y \in \mathcal{N}$ implies $X \cap Y \in \mathcal{N}$.*
3. *If $y, z \notin \tau(X)$ and $z \in \tau(X \cup \{y\})$ then $y \notin \tau(X \cup \{z\})$.*

The sets in $\mathcal{N}$ are called convex.

Extreme points are defined in the same way as in the Euclidean setting.

Definition 4 *For a set $A \subset E$, we say that $x \in A$ is an extreme point of A if $x \notin \tau(A \setminus \{x\})$. The set of extreme points of A is denoted by $ex(A)$.*

A set $X \subset E$ is called free if $X = ex(X)$.

We will use the following concept.

Definition 5 *A sequence $x_1, \ldots, x_k$ of points of E is called a shelling sequence if for all $1 \leq i \leq k$, x_i is an extreme point of $E \setminus \{x_1, \ldots, x_{i-1}\}$.*

A shelling sequence can be thought of as a way to reach a convex set by starting with the whole set E and stripping away points one after the other in such a way that the set remains convex at each step. A useful characterisation of convex sets for our purpose is the following, where we describe a convex set via a shelling process.

Proposition 21 ([13]) *A set $X \subset E$ is convex if and only if there exists a shelling sequence $x_1, \ldots, x_k$ such that $X = E \setminus \{x_1, \ldots, x_k\}$.*

The final ingredient we need is the following Helly-type theorem for convex geometries.

Theorem 22 ([13]) *Let $h(\mathcal{N})$ denote the smallest integer k such that the following holds:*

For a family $\mathcal{F}$ of convex sets, if every subfamily of size at most k has a non-empty intersection, then $\mathcal{F}$ has a non-empty intersection.

Then $h(\mathcal{N})$ is equal to the maximum size of a free convex set.

Now we can state and prove a Hadwiger-Debrunner-type theorem for convex geometries.

Theorem 23 *Consider a convex geometry $(E, \mathcal{N})$. Let h be the size of a maximum free convex set and let (p, q) be an h-admissible pair. Let $\mathcal{F} \subset \mathcal{N}$ be a family of $n \geq p$ non-empty convex sets. If $\mathcal{F}$ has the (p, q)-property then there exist $p - q + 1$ elements of E stabbing $\mathcal{F}$.*

Proof. We know that $\emptyset$ is convex, thus there exists a shelling sequence $S = \{x_1, \ldots, x_k\}$ such that $\emptyset = E \setminus S$, i.e. $S = E$. Let for $1 \leq i, j \leq k$, let us say that $x_i \preceq x_j$ if and only if $i \geq j$. Let $1 \leq t \leq k$ be an integer. Because $\{x_1, x_2, \ldots, x_{t-1}\}$ is a valid shelling sequence, $\{x \in E \mid x \leq x_t\}$ is a convex set. Thus, $\preceq$ has the convex order property.

Let h be the maximum size of a free convex set. Then, it is easy to verify that $\mathfrak{S} = (E, \mathcal{N}, \mathcal{N}, h, \preceq)$ also has the intersection closure and attainable minimum properties. The Helly property (for Helly number h) is given by Theorem 22.

Thus, $\mathfrak{S}$ is an Ordered-Helly system and we get the result from Theorem 13. $\square$

We will now give a few illustrative examples of abstract convex geometries and the resulting Hadwiger-Debrunner type results we obtain for them. One such convex geometry, arguably the most natural, is obtained by taking convex hulls of subsets on a finite point set in $\mathbb{R}^d$. This is conceptually similar to the case of polytopes in Euclidean space which we have already discussed. The following two examples are perhaps not so immediately related.

Subtrees of a tree

Proposition 24 ([13]) *Let T be a tree on a set of vertices V. Let $\mathcal{N}$ be the family of all sets of vertices corresponding to subtrees of T. Then $(V, \mathcal{N})$ is a convex geometry with Helly number $h(\mathcal{N}) = 2$.*

This means that if in a given family of subtrees of T all pairs of subtrees intersect at some vertex, then all subtrees share a vertex. Using Theorem 23 we thus get the following.

Corollary 25 *Let T be a tree and let $\mathcal{F}$ be a family of subtrees of T (represented as sets of vertices). Let (p, q) be a 2-admissible pair. Let $\mathcal{F} \subset \mathcal{N}$ be a family of non-empty subtrees of T with the (p, q)-property. Then $\mathcal{F}$ can be stabbed with $p - q + 1$ vertices.*

Ideals of a partially ordered set For a poset $(E, \leq)$, we say that a set $S \subset E$ is an ideal of E if for all $x \in S$ and all $y \in E$, $y \leq w \Rightarrow y \in S$. Let $width(E)$ denote the maximum size of an antichain in E. Then the following holds.

Proposition 26 ([13]) *Let $(E, \leq)$ be a finite poset. Let $\mathcal{F} = \{S \subset E \mid S \text{ is an ideal}\}$. Then $(E, \mathcal{F})$ is a convex geometry with Helly number $width(E)$.*

Using Theorem 23 we can thus get the following result.

Corollary 27 *Let $(E, \leq)$ be a finite poset. Let (p, q) be a $width(E)$-admissible pair and let $\mathcal{F}$ be a family of non-empty ideals of E with the (p, q)-property. Then $\mathcal{F}$ can be stabbed by $p - q + 1$ elements of E.*

4 Conclusion

We have shown how to stab convex polygons with a total of n vertices and the (p, q)-property (for admissible (p, q)) in expected $\tilde{\mathcal{O}}(n^{4/3})$ time with respect to n. As an intermediate step, we compute a certain quantity x_2^*, which is a Hopcroft-Hard problem, in $\tilde{\mathcal{O}}(n^{4/3})$ expected time. While this is believed to be near optimal, finding a non-trivial lower-bound for the original problem remains open. For the 3D case, we have an algorithm running in expected $\tilde{\mathcal{O}}(n^{5/2})$ time with respect to n.

We have also considered other conditions which allow to conclude that a set of polygons can be stabbed with a fixed number of points, and applied our algorithm to those. One of these conditions is a new generalization of Helly's theorem in the plane in terms of holes in the union of convex sets.

Finally, we have derived (p, q)-theorems along with algorithms in other settings where Helly-type theorems are known. An interesting question would be to try deriving other related results in these settings, such as colourful or fractional versions of (p, q)-theorems.

A natural next step in the Euclidean setting would be to drop the restriction $(d - 1)p < d(q - 1)$ and find efficient algorithms for the Alon-Kleitman (p, q)-theorem. Their proof of existence of stabbing sets of constant size uses the fractional Helly theorem, whose proof is similar to the above proof of the Hadwiger-Debrunner (p, q)-theorem. It is thus conceivable that similar ideas could be applied to this more general case.

References

[1] P. K. Agarwal, M. de Berg, S. Har-Peled, M. H. Overmars, M. Sharir, and J. Vahrenhold. Reporting intersecting pairs of convex polytopes in two and three dimensions. *Computational Geometry*, 23(2):195 – 207, 2002.

[2] N. Alon, G. Kalai, J. Matoušek, and R. Meshulam. Transversal numbers for hypergraphs arising in geometry. *Advances in Applied Mathematics*, 29(1):79–101, 2002.

[3] N. Alon and D. J. Kleitman. Piercing convex sets. *Bulletin of the American Mathematical Society*, 27(2):252–257, Aug. 1992.

[4] G. Averkov and R. Weismantel. Transversal numbers over subsets of linear spaces. *Advances in Geometry*, 12, 02 2012.

[5] M. Breen. Starshaped unions and nonempty intersections of convex sets in $\mathbb{R}^d$. *Proceedings of the American Mathematical Society*, 108(3):817–820, 1990.

[6] P. Carmi, M. J. Katz, and P. Morin. Stabbing Pairwise Intersecting Disks by Four Points. *arXiv e-prints*, Dec. 2018.

[7] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22(4):547–567, Dec. 1999.

[8] T. M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discret. Comput. Geom.*, 56(4):860–865, 2016.

[9] J. Dallant and P. Schnider. Efficiently stabbing convex polygons and variants of the Hadwiger-Debrunner (p, q)-theorem. *arXiv preprint arXiv:2002.06947*, 2021.

[10] L. Danzer. Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene. *Studia Sci. Math. Hungar.*, 21(1–2):111–134, 1986.

[11] J.-P. Doignon. Convexity in cristallographical lattices. *Journal of Geometry*, 3(1):71–85, Mar. 1973.

[12] J. Eckhoff. A survey of the Hadwiger-Debrunner (p, q)-problem. In *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, pages 347–377. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[13] P. H. Edelman and R. E. Jamison. The theory of convex geometries. *Geometriae Dedicata*, 19(3), Dec. 1985.

[14] J. Erickson. New lower bounds for hopcroft's problem. *Discret. Comput. Geom.*, 16(4):389–418, 1996.

[15] H. Hadwiger and H. Debrunner. Über eine Variante zum Hellyschen Satz. *Archiv der Mathematik*, 8(4):309–313, oct 1957.

[16] H. Hadwiger, H. E. Debrunner, and V. Klee. *Combinatorial Geometry in the Plane.* New York (N.Y.) : Holt, 1964.

[17] S. Har-Peled, H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, M. Sharir, and M. Willert. Stabbing Pairwise Intersecting Disks by Five Points. In *29th International Symposium on Algorithms and Computation (ISAAC)*, 2018.

[18] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[19] A. J. Hoffman. Binding constraints and Helly numbers. *Annals of the New York Academy of Sciences*, 319:284 – 288, 12 2006.

[20] A. F. Holmsen and D. Lee. Radon numbers and the fractional helly theorem. *Israel Journal of Mathematics*, 241(1):433–447, 2021.

[21] C. Keller, S. Smorodinsky, and G. Tardos. Improved bounds on the Hadwiger–Debrunner numbers. *Israel Journal of Mathematics*, 225(2):925–945, 2018.

[22] D. Kleitman, A. Gyárfás, and G. Tóth. Convex sets in the plane with three of every four meeting. *Combinatorica*, 21:221–232, 04 2001.

[23] B. Korte, R. Schrader, and L. Lovász. *Greedoids*. Springer Berlin Heidelberg, 1991.

[24] J. Matousek. Range searching with efficient hierarchical cuttings. In D. Avis, editor, *Proceedings of the Eighth Annual Symposium on Computational Geometry, Berlin, Germany, June 10-12, 1992*, pages 276–285. ACM, 1992.

[25] J. Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 2002.

[26] D. McGinnis. A family of convex sets in the plane satisfying the $(4, 3)$-property can be pierced by nine points. *arXiv preprint arXiv:2010.13195*, 2020.

[27] L. Montejano. A new topological Helly theorem and some transversal results. *Discrete & Computational Geometry*, 52(2):390–398, 2014.

[28] L. Montejano and P. Soberón. Piercing numbers for balanced and unbalanced families. *Discrete & Computational Geometry*, 45(2):358–364, 2011.

[29] S. Moran and A. Yehudayoff. On weak epsilon-nets and the radon number. In *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129, page 51. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[30] R. J. Segura and F. R. Feito. An algorithm for determining intersection segment-polygon in 3d. *Comput. Graph.*, 22(5):587–592, 1998.

[31] L. Stachó. A solution of gallai's problem on pinning down circles. *Matematikai Lapok*, 32(1–3):19–47, 1984.

[32] G. Wegner. Über Helly-Gallaische Stichzahlprobleme. *3. Kolloquium über Diskrete Geometrie*, pages 277–282, 1985.

Rectangle Stabbing and Orthogonal Range Reporting Lower Bounds in Moderate Dimensions

Peyman Afshani[*] Rasmus Killmann[†]

Abstract

We study the orthogonal range reporting and rectangle stabbing problems in moderate dimensions, i.e., when the dimension is $c\log(n)$ for some constant c. In orthogonal range reporting, the input is a set of n points in d dimensions, and the goal is to store these n points in a data structure such that given a query rectangle, we can report all the input points contained in the rectangle. The rectangle stabbing problem is the "dual" problem where the input is a set of rectangles, and the query is a point.

Our main result is the following: assume using $S(n)$ space, we can solve either problem using $Q(n) + O(t)$ time in the pointer machine model of computation where t is the output size. Then, we show that if $Q(n) = O(n^{1-\gamma})$, the space must be $n^{1-\gamma+\Omega(\sqrt{c\gamma})}$ where $c = d/\log n \geq 1$ and $0 \leq \gamma \leq \frac{4}{4+\log c}$. Interestingly, to obtain this lower bound using a non-constructive method, we show the existence of particular types of codes that generalize a specific aspect of error correction codes. Our result overcomes the shortcomings of the previous lower bounds in the pointer machine model for non-constant dimension [3, 4, 5, 12], as the previous results could not be extended for $d = \Omega(\sqrt{\log n})$.

The only known lower bounds for rectangle stabbing, when the dimension is non-constant, are based on conditional lower bounds upon the best-known results on CNF-SAT [20]. Therefore, our lower bound is the first non-trivial unconditional lower bound for orthogonal range reporting and rectangle stabbing with non-constant dimension.

1 Introduction

The focus of this paper is to study two fundamental problems within the area of *range searching*. The first problem is *orthogonal range reporting* which is the problem of storing a set of n points in $\mathbb{R}^d$ in a data structure, such that given a query hyper rectangle[1] in d-dimensions the t points contained in the hyper rectangle can be reported efficiently. The second problem of *rectangle stabbing* is the "dual" in the sense that the input set and the query is swapped.

Both problems are fundamental and have attracted significant attention [7, 6]. Using standard reductions, it is possible to reduce orthogonal range searching in d dimensions to rectangle stabbing in $2d$ dimensions and vice versa[2]. While in low dimensions, the constant factor increase in dimension is very impactful, the focus of this paper is when the dimension is large, $d = c\log n$, for a parameter c, and thus, the two problems essentially become equivalent.

We look into proving unconditional space and query lower bounds for both problems, although in the somewhat restricted pointer machine model [18]. Along the way, we find an interesting connection to *error correction* codes, which is a particular set of strings/codes which can be utilized to communicate over an insecure line of communication.

1.1 Previous results

There are plenty of known results for both problems when the dimension is low, while when the dimension grows beyond a constant, the problems are less characterized.

Rectangle stabbing. In the one-dimensional case, rectangle stabbing, also called interval stabbing, can be solved using a diverse range of methods [14]. The optimal solutions use linear space and have a query time of $O(\log n + t)$. In the two-dimensional case, Chazelle [11] presented an optimal data structure using linear space and $O(\log n + t)$ query time. Almost optimal results have also been obtained for $d = 3$, after a long series of papers [16, 10, 15, 4]. Using range trees, the two-dimensional and three-dimensional results can be generalized to higher dimensions by paying a $\log(n)$ factor in both space and query time for each dimension added. In the pointer machine model, it is also known that using $O(n\log^{O(1)} n)$ space, the query time must be $\Omega(\log n(\log n/\log\log n)^{d-2} + t)$ [4].

Orthogonal range reporting. For $d = 1$, a simple binary search tree answers range queries in $O(\log n + t)$

[*]Aarhus University, `peyman@cs.au.dk`

[†]Aarhus University, `killmann@cs.au.dk`

[1]By hyper rectangle, we mean the Cartesian product of d intervals.

[2]Both problems can be reduced to dominance reporting in 2d dimensions.

time, while using linear space. For $d = 2$, the problem can be solved with the same query time but using $O(n \log n/ \log\log n)$ space [11], which is optimal in the pointer machine model [12]. Much later, the optimal result was obtained for three dimensions in the pointer machine model [3]. Similar to the rectangle stabbing problem, the results can be generalized to a higher dimension by paying a $\log n$ factor in both space and query time for each dimension added. In the pointer machine model, it is also known that answering queries in $O(\log^{O(1)} n + t)$ time requires $\Omega(n(\log n/ \log\log n)^{d-1})$ space [12].

These approaches are unsuitable for the case where $d = c\log(n)$, since the dependency on the dimension penalizes this construction too much, often referred to in the community as the "*curse of dimensionality*". The best known upper bound, devised by Chan [9], when the dimension is $c\log(n)$, is a data structure which achieves $Q(n) = O(n^{1-1/c\log(c)})$ expected query time and space $S(n) = n^{1+\delta}$ for any $\delta > 0$.

Previous lower bounds. Currently, the available lower bounds are conditional lower bounds, using the best known results on CNF-SAT [20], which gives preprocessing/query time lower bounds. Furthermore, obtaining unconditional lower bounds in unrestricted computational models, like the RAM, is completely hopeless. Fortunately, suppose we restrict the memory model of the data structure to operate in the pointer machine model. In that case, one can obtain reasonable lower bounds that are tight for a wide range of problems. In fact, for the orthogonal range reporting problem and the rectangle stabbing problem, there are a number of lower bounds available [3, 4, 12]. However, the dependency of these lower bounds on the dimension is not very good, and in fact, it is not explicitly mentioned by the lower bounds. There is one very related instance where we are aware of the explicit dependency on the dimension. Afshani and Driemel [5] study multilevel structures, motivated by answering Fréchet queries. However, by a careful look at the details of the lower bounds, one can see that the hidden constant is at least $d^{\Omega(d)}$, meaning the lower bounds becomes useless as d approaches $\log n/ \log\log n$. They study the Cartesian product of t 2D "simplex stabbing" problems, and they show that for any data structure with $S(n)$ space and $Q(n)$ query time we must have

$$S(n) = \Omega((\frac{n}{Q(n)})^2)\frac{\left(\frac{\log(n/Q(n))}{\log\log n}\right)^{t-1}}{2^{O(2^t)}} \quad \text{and}$$

$$S(n) = \Omega\left(\frac{n}{Q(n)}\right)^2 \Theta\left(\frac{\log(n/Q(n))}{t^{3+o(1)}\log\log n}\right)^{t-1-o(t)}.$$

It has been shown that this problem can be modelled using the Cartesian product of t two-dimensional problems and furthermore it contains a subproblem involving an instance of the t-dimensional orthogonal range reporting problem. However, as it can be seen in the lower bounds, the dependency on t is such that the lower bounds becomes trivial as soon as t gets close to $(\log n)^{1/3}$.

1.2 Our results

Our main result states that any data structure operating in the pointer machine model, which solves the rectangle stabbing problem (or the orthogonal range reporting problem), when the dimension is $c\log(n)$ for $c \geq 1$ and uses query time $Q(n) = O(n^{1-\gamma})$, must use $n^{1-\gamma+\Omega(\sqrt{c\gamma})}$ space, where $0 \leq \gamma \leq \frac{4}{4+\log c}$. As far as we know, this is the first non-conditional lower bound for the two problems when the dimension is non-constant.

We obtain our results using an interesting extension of error correction codes. We show that given a set of binary strings S of size $O((\frac{d}{r})^{r/4})$, where r is the number of 1's in the strings and d is the length of each string, then for any h of these strings $s_1, \ldots, s_h \in S$ the string $s_1 \vee \cdots \vee s_h$ has more than M 1's, where "the error parameter" M can be chosen to be $\min\{\sqrt{rd}/e^2, rh/8\}$. In particular this gives a set of strings in which for any h strings they are fairly different. Notice that for $h = 2$ this is precisely the property sought after when constructing error correction codes.

Our technical contribution. We believe our main technical contribution is a simple non-constructive idea for showing the existence of "difficult input instances". Previously, two main techniques were used to build "difficult input instances" for range searching lower bounds. The first one is a deterministic and constructive approach that builds an explicit set of points, and it was first employed by Chazelle [12] for the 2D orthogonal range reporting problem. Since then it has been generalized to higher dimensions [2, 4]. Unfortunately, this approach suffers exponentially by the dimension, and thus it cannot be applied in dimensions above $\log n/ \log\log n$. The second approach is a randomized approach that uses the classical "alteration" technique (basically a 'sample and refine' strategy) [8]. This was also used in Chazelle's original paper [12] and since then, it has been the dominant strategy. Unfortunately, applying this technique in a way that does not degrade exponentially on the dimension is challenging. It leads to lengthy calculations, making it difficult to know the optimal choices of the parameters involved. Here instead, we introduce a new and more straightforward *non-constructive* approach. We show that if the structures we are looking for do not exist, we can upper bound a certain count, and then we reach a contradiction by showing that the upper bound is smaller than the count, meaning the desired structure should exist.

2 Technical preliminaries

In this section, we present some of the technicalities and preliminaries needed to prove our lower bound. We will explain the underlying model of computation, the pointer machine model, and an existing lower bound framework in this model. Finally, we will introduce the notion of error correction codes.

2.1 Model of computation and framework

The Pointer Machine Model [18] is an underlying model of computation which models data structures that only navigate through pointers to access the underlying memory locations, e.g., any tree-based data structure. The model has proven to be efficient in proving lower bounds for data structure problems. Consider an abstract "reporting" problem where the input is a set $\mathcal{U}$ and each query q reports a (implicitly defined) subset $q_{\mathcal{U}}$, of $\mathcal{U}$. The underlying data structure is a directed graph $\mathcal{G}$, where each node has out-degree two and represents exactly one memory cell; furthermore, there is a special root node $r(\mathcal{G})$. Each node can store exactly one element from the universe $\mathcal{U}$ and edges between nodes represent pointers between memory cells. Any additional information can be stored and accessed for free by the data structure, and the data structure has unlimited computational resources. Given a query q, the data structure must start at the root node $r(\mathcal{G})$ and explore a connected subgraph $\mathcal{G}' \subseteq \mathcal{G}$, such that each element of the query $q_{\mathcal{U}}$ is contained in at least one node of the subgraph $\mathcal{G}'$. The size of the subgraph $\mathcal{G}'$ gives a lower bound on the query time and the size of the entire graph $\mathcal{G}$ gives a lower bound on the space required to solve the reporting problem.

One of the first lower bound frameworks in the pointer machine model was given by Bernard Chazelle [12, 13]. In this paper we will use a slightly different framework developed by Afshani [1], which states the following.

Theorem 1 *Assume we have a data structure for a geometric stabbing problem that uses at most $S(n)$ space and answers queries in $Q(n) + O(k)$ time in which n is the input size, and k is the output size. Assume for this problem we can construct an input set, inside the unit cube in $\mathbb{R}^d$, of n ranges such that*

1. *Every point of the unit cube is contained in t ranges in which $t \geq Q(n)$.*
2. *The volume of the intersection of every α ranges is at most v, for $\alpha < t$.*

Then, $S(n) = \Omega(tv^{-1}/2^{O(\alpha)}) = \Omega(Q(n)v^{-1}/2^{O(\alpha)})$.

We will also utilize error correction codes from the field of information theory to satisfy the second condition of the framework. In fact, it turns out that what we need is an extension of error correction codes with additional properties.

2.2 Error detection and correction codes

Error-correcting codes are an old technique studied within the field of information theory and were originally studied by Richard Hamming in 1947 [19]. Hamming developed the famous Hamming code, which first appeared in Claude Shannon's *A Mathematical Theory of Communication*[17]. The Hamming code is a family of linear error-correcting codes that can be used for both error detection and error correction. The underlying problem of error detection and error correction are the following.

Alice and Bob want to transmit a string s from a fixed set of S binary strings. They communicate across an insecure line, and hence a number of bits may be flipped. In error detection Alice sends a string $s \in S$, and Bob must determine whether the received string s' is equal to the original string s sent by Alice. In error correction, Alice sends a string $s \in S$, but now Bob must recover the original string s from the received string s'. Note that any error correction scheme can also be used as an error detection scheme, but not the other way around. In section 3.1 we look into error correction and give, in our opinion, a natural and interesting extension of the problem.

2.3 Bounding the binomial coefficient

Our lower bound is proven by using a counting argument. At times we need to bound the binomial coefficient to arrive at our desired result. The following simple Lemma is known, but for completeness, we present a proof in the appendix.

Lemma 1 *For positive integers n and k where $1 \leq k \leq n$, it holds that*

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

3 The lower bound construction

In this section, we present the details of our lower bound construction. We start this section with a discussion of error correction codes, and a subproblem related to them. We start by defining what an interesting string is, which is both used in the error correction section 3.1 and when concluding the lower bound in section 3.2.

Definition 1 *A binary string is interesting if it has length d and has exactly r 1's, for some fixed parameters r and d.*

The parameters r and d will be fixed in our final lower bound.

Theorem 2 *Given r, d and h, such that $r < \frac{d}{4}$, it holds that for any $N \le (\frac{d}{r})^{r/4}/2$, there exists a set S of N interesting strings such that for any h strings $s_1, ..., s_h \in S$ the string $s_1 \vee \cdots \vee s_h$ has more than M 1's, where M can be chosen to be $\min\{\frac{\sqrt{dr}}{e^2}, \frac{rh}{8}\}$ and $\vee$ is the logical OR operation.*

The error parameter M can be thought of as a measure of how distinct the h strings can be, and therefore it translates into how many flipped bits the construction can withstand and still recover. We will discuss this below.

The above Theorem helps us prove our lower bound on rectangle stabbing when the dimension is $c\log(n)$. This paper's main result is formulated in the following Theorem and will be proven at the end of this section.

Theorem 3 *Consider a pointer machine data structure that solves the rectangle stabbing problem in $c\log(n)$ dimensions, for some parameters $c \ge 1$ and $0 \le \gamma \le \frac{4}{4+\log c}$ which achieves query time $Q(n) = O(n^{1-\gamma})$, then it must use $n^{1-\gamma+\Omega(\sqrt{c\gamma})}$ space.*

And as we mentioned earlier since we are in non-constant dimension this result also gives us a lower bound for orthogonal range reporting.

Corollary 1 *Consider a pointer machine data structure that solves the orthogonal range reporting problem in $c\log(n)$ dimensions, for some parameters $c \ge 2$ and $0 \le \gamma \le \frac{4}{4+\log c}$ which achieves query time $Q(n) = O(n^{1-\gamma})$, then it must use $n^{1-\gamma+\Omega(\sqrt{c\gamma})}$ space.*

3.1 Codes with large spread

In this section, we consider a subproblem that we believe is an interesting generalization of error-correcting codes. Error-correcting codes deal with two parties, Alice and Bob, who have to communicate over an insecure line, meaning that a number of bits may be flipped in any message that they send to each other. Here, we only consider binary messages. To reduce the impact of errors, they agree to only communicate messages from a fixed set S. If Alice sends a message $s \in S$ we want Bob to be able to retrieve the correct s from the string s' he received. This is typically expressed as every two messages $s_1, s_2 \in S$ having Hamming distance at least $2\delta + 1$, for some parameter δ. Now, if at most δ bits change in the message that Alice sends to Bob, Bob can correctly identify Alice's message in S. Observe that if both s_1 and s_2 have r 1s, then the fact that their Hamming distance is at least $2\delta + 1$ is equivalent to the fact that $s_1 \vee s_2$ has at least $r + 2\delta + 1$ 1s. Thus, in Theorem 2, if we set $h = 2$, we get a set of error correcting codes, i.e., a set of codes whose pairwise Hamming distance is at least $M - r$.

The initial and immediate approach to prove Theorem 2 would be to do random sampling: consider all the strings with r 1s and pick a random sample of them with probability p; then delete the strings that result in some h strings to have few 1s in their logical OR, then optimize for the value of p that gives the most strings. Unfortunately, this natural approach seems difficult to analyse, due to very complex binomial coefficients involved. Instead, we come up with an alternative approach that we believe is simple but non-constructive.

Our non-constructive proof is a counting argument which fortunately turns out to be easy to analyse. It involves two different ways of counting the different number of ways to construct the set S. To gain a second way of counting the ways to construct S we need to assume the negated Statement of Theorem 2, if we obtain a contradiction by assuming this negated Statement, then the Theorem is proven. We start the proof by writing down the negated Statement.

Statement 1 *Given r, d and h, then for all sets of N interesting strings there exists a subset $s_1, s_2, \ldots, s_h$ such that $s_1 \vee s_2 \vee \cdots \vee s_h$ contains less than M 1's.*

The idea is to reach a contradiction by a counting argument. The total number of ways to pick N interesting strings is $\binom{\binom{d}{r}}{N}$. From Statement 1 we can bound this count in a different way. Assuming Statement 1 every set S of N interesting strings has a subset of h strings $s_1, ..., s_h$ such that $s_1 \vee \cdots \vee s_h$ has less than M 1's. We now describe choosing S in a different way. To do that, first we choose the strings $s_1, ..., s_h$ and then the remaining strings of S. To choose $s_1, ..., s_h$, first we choose the string $s_0 = s_1 \vee ... \vee s_h$. Then we choose $s_1, ..., s_h$ as a "subset" of s_0. To choose s_0, we need to pick M positions that contain 1s. The number of ways to pick these M positions is $\binom{d}{M}$. The number of ways to pick h interesting strings among these M positions is $\binom{\binom{M}{r}}{h}$. The ways to pick the remainder of the N interesting strings after having picked these h strings becomes $\binom{\binom{d}{r}}{N-h}$. If we combine all of this, we get an expression that overcounts the number of ways to pick N interesting strings, which means that we would expect this count to be bigger than the exact count. Therefore we would expect the following inequality to hold

$$\binom{\binom{d}{r}}{N} < \binom{d}{M}\binom{\binom{M}{r}}{h}\binom{\binom{d}{r}}{N-h}.$$

Thus, if this inequality does not hold, then it follows that Statement 1 is incorrect which in turn prove Theorem 2.

3.1.1 Arriving at a contradiction

This section is dedicated to finalizing our proof of Theorem 2. The way we do this, as mentioned above, is to

$$\begin{array}{ll}
\text{String 1} & 00\cdots 00\overbrace{100111\cdots 11}^{M}00\cdots 00 \\
\text{String 2} & 00\cdots 00\overbrace{110010\cdots 10}^{M}00\cdots 00 \\
 & \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
\text{String } h & 00\cdots 00\overbrace{010111\cdots 00}^{M}00\cdots 00
\end{array}$$

Figure 1: h strings which have their r 1's concentrated around the same M positions.

arrive at a contradiction of Statement 1. To do that, we will try to arrive at the opposite inequality, i.e., we will try to satisfy the opposite inequality. The first step is to utilize the following Lemma to restructure our expression; we refer to the appendix for the proof.

Lemma 2 $\frac{\binom{A}{B}}{\binom{A}{B-C}} \geq (\frac{A}{2B})^C$, *if* $B > 2$ *and* $A > 2B$.

We can apply Lemma 2 to our expression by setting $A = \binom{d}{r}$, $B = N$ and $C = h$. This gives a new expression we need to satisfy in order to arrive at our contradiction,

$$\left(\frac{\binom{d}{r}}{2N}\right)^h > \binom{d}{M}\binom{\binom{M}{r}}{h}.$$

Now we need to apply Lemma 1. The idea is to create a new Statement, where we increase the right side and/or decrease the left side. If we can satisfy this new inequality, our original inequality will also be satisfied. We apply both the lower bound on the left side since we want to decrease this expression and the upper bound on the right side since we want to increase that expression.

$$\left(\frac{\left(\frac{d}{r}\right)^r}{2N}\right)^h > \left(\frac{de}{M}\right)^M \left(\frac{e\left(\frac{eM}{r}\right)^r}{h}\right)^h \Leftarrow$$
$$\left(\frac{d}{r}\right)^r > \left(\frac{de}{M}\right)^{\frac{M}{h}} e\left(\frac{eM}{r}\right)^r 2N.$$

Where the implication follows from dividing by h in the exponent and removing the h in the denominator on the right side. We are going to proceed by splitting this expression into three pieces. By ensuring that these three inequalities are satisfied

1. $\left(\frac{d}{r}\right)^{\frac{r}{4}} \geq 2N$
2. $\left(\frac{d}{r}\right)^{\frac{r}{2}} > e\left(\frac{eM}{r}\right)^r$
3. $\left(\frac{d}{r}\right)^{\frac{r}{4}} > \left(\frac{de}{M}\right)^{\frac{M}{h}}$

we arrive at the desired contradiction. The first inequality directly gives us the upper bound on the size of N namely that $\left(\frac{d}{r}\right)^{r/4}/2 \geq N$. For the second inequality, we get an upper bound on the value of M

$$\left(\frac{d}{r}\right)^{\frac{r}{2}} > e\left(\frac{eM}{r}\right)^r \Leftrightarrow \left(\frac{d}{r}\right)^{\frac{1}{2}} > e^{1/r}\left(\frac{eM}{r}\right) \Leftrightarrow$$
$$\frac{\sqrt{dr}}{e^{1+1/r}} > M \Leftarrow \frac{\sqrt{dr}}{e^2} > M.$$

The third inequality gives us a bound on M and a relation between r and d:

$$\left(\frac{d}{r}\right)^{\frac{r}{4}} > \left(\frac{de}{M}\right)^{M/h} \Leftrightarrow \frac{rh}{4}\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{M}\right)$$

Note that we have $M \geq r$ and thus, we can increase the RHS by replacing M with r,

$$\frac{rh}{4}\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{M}\right) \Leftarrow$$
$$\frac{rh}{4}\log\left(\frac{d}{r}\right) > M\log\left(\frac{de}{r}\right) \Leftarrow$$
$$\frac{rh}{4}\log\left(\frac{d}{r}\right) > M\left(\log\left(\frac{d}{r}\right) + 2\right) \Leftarrow \frac{rh}{8} > M.$$

Therefore, since we have two bounds on M we can choose M to be $\min\{\frac{\sqrt{dr}}{e^2}, \frac{rh}{8}\}$ as long as $\left(\frac{d}{r}\right)^{r/4}/2 > N$ and $r < \frac{d}{4}$. When the parameters satisfy these inequalities we arrive at a contradiction of Statement 1, which then proves Theorem 2.

3.2 Utilizing the framework

This section is dedicated to proving a lower bound on rectangle stabbing when the dimension is $c\log(n)$ for some constant $c \geq 1$. The idea is to utilize the framework by Afshani, Theorem 1 and use Theorem 2 from the previous section to satisfy the second condition of Afshani's framework.

Assume we have a data structure that uses $S(n)$ space and has a query time of $Q(n) + O(k)$. To use the framework, we need to build a set of ranges. Each range will be an axis-aligned rectangle in d dimensions. Note that while the framework requires $t \geq Q(n)$, it is also sufficient to create $t \geq Q(n)/2$. To see this, let c_0 be the constant hidden in the O-notation in the query time, meaning, we assume the query time is at most $Q(n) + c_0k$. Observe that as long as for the output size k we have $k \geq Q(n)/2$, then $Q(n)+c_0k \leq Q(n)/2+(1+c_0)k$. And thus, this only changes the constant factors involved in the lower bound. Thus, in the rest of the proof, we assume that we create $t \geq Q(n)/2$ different shapes, and we will use each shape to cover the unit cube in $d = c\log(n)$ dimensions. For each shape, we tile the unit cube with $\frac{n}{Q(n)}$ copies. If we do that, this ensures that we have n

rectangles in total. We build our shapes in the following way: for each shape, we have r side lengths of length $1/2$ and $d-r$ side lengths of unit length. This implies that each rectangle of any shape has volume $\frac{Q(n)}{n} = \frac{1}{2^r}$.

Observation 1 *Our shapes satisfies the first condition of Theorem 1.*

Every point of the unit cube has been covered by at least $Q(n)/2$ different shapes, and thus the answer to any query point will have a size of at least $Q(n)/2$ and as described above, this is sufficient to satisfy the first condition.

Now we are only left with making sure that these shapes can satisfy the second condition of the framework, namely that the volume of the intersection of every α shapes is at most v.

We say that a rectangle R is *represented* by a binary string s of length d if the following holds: for every i, $1 \le i \le d$, if s contains a 1 at position i, then R has side-length $\frac{1}{2}$ along dimension i but otherwise, R has side-length 1.

Observation 2 *Every rectangle represented by an interesting string has volume $\frac{1}{2^r}$.*

The next Observation reveals why we had to study this particular type of error correction codes in section 3.1.

Observation 3 *Let T be a set of interesting strings given by Theorem 2, for some parameters r, h which is to be decided later. Let $R_1, \cdots, R_h$ be h rectangles represented by h distinct strings $s_1, \cdots, s_h \in T$, then the volume of the intersection of $R_1, \cdots, R_h$ is at most 2^{-M}.*

Proof. Simply observe that $R_1 \cap \cdots \cap R_h$ will have a side-length of at most $1/2$ at dimension i if any of the strings has a 1 at position i. The Lemma follows from Theorem 2 i.e., that the logical OR of the strings has at least M 1s. $\square$

By the above observation, it thus follows that the goal is to pick $Q(n)$ interesting strings, such that for any α strings in the set we have that $s_1 \vee ... \vee s_\alpha$ have more than β 1's. This will ensure that the corresponding shapes, for any α shapes, the volume of their intersection will be less than $v = \frac{1}{2^\beta}$. We utilize Theorem 2, since we have strings of length d with exactly r 1's, h becomes α and M becomes our β.

We know that $d = c\log(n)$. Let γ be such that $Q(n) = n^{1-\gamma}$. Observe that since $Q(n)/n = (1/2)^r$, we get that $r = \gamma \log(n)$. Now we are only left with making sure that these parameters fit Theorem 2. To do that, we must ensure that $N = Q(n)/2$:

$$\begin{aligned}
\frac{Q(n)}{2} &\le \left(\frac{d}{r}\right)^{r/4}/2 \quad \Leftrightarrow \\
n^{1-\gamma} &\le \left(\frac{c}{\gamma}\right)^{r/4} \quad \Leftrightarrow \\
\log(n)(1-\gamma) &\le \frac{\gamma \log(n)}{4} \log\left(\frac{c}{\gamma}\right) \quad \Leftrightarrow \\
(1-\gamma)4 &\le \gamma \log\left(\frac{c}{\gamma}\right) \quad \Leftrightarrow \\
(1-\gamma)4 &\le \gamma\log(c) - \gamma\log(\gamma) \Leftarrow \\
(1-\gamma)4 &\le \gamma \log(c).
\end{aligned}$$

Where the last step follows since $\gamma \le 1$ and thus $-\gamma\log(\gamma)$ is non-negative, and therefore, we can make the RHS smaller by removing the $-\gamma\log(\gamma)$ term. Then we just end up with $(1-\gamma)4 \le \gamma\log(c)$ and so we get that $\gamma \ge 4/(4+\log(c))$. We also need that $c \ge 1$; otherwise, γ will not be between 0 and 1. Note that $\gamma \ge 4/(4+\log(c))$ implies that $Q(n) \le n^{1-\frac{4}{4+\log(c)}}$.

Now we are left with finding the possible value of $M = \beta$. Recall that by Theorem 2, we have $\beta \le \min\{\frac{\sqrt{dr}}{e^2}, \frac{rh}{8}\} = \min\{\frac{\sqrt{c\log(n)\gamma\log(n)}}{e^2}, \frac{\gamma\log(n)\alpha}{8}\}$. We pick α to balance the two terms in the min function. This yields $\alpha = \Theta(\sqrt{\frac{c}{\gamma}})$ and $\beta = \Theta(\alpha r)$. We are now ready to plug these values into our lower bound framework.

Thus, if the query time is $Q(n) \le n^{1-\frac{4}{4+\log(c)}}$ then the space becomes

$$\begin{aligned}
S(n) &= \Omega\left(\frac{Q(n)2^\beta}{2^{O(\alpha)}}\right) = n^{1-\gamma}2^{\Omega(\alpha r)} \\
&= n^{1-\gamma}2^{\Omega\left(\sqrt{\frac{c}{\gamma}}\gamma\log(n)\right)} = n^{1-\gamma}n^{\Omega\left(\sqrt{c\gamma}\right)} \\
&= n^{1-\gamma+\Omega\left(\sqrt{c\gamma}\right)}.
\end{aligned}$$

Which concludes our lower bound construction and proves Theorem 3.

4 Conclusions

This paper considered the orthogonal range reporting problem and the rectangle stabbing problem in moderate dimensions. Both are classical problems in the field of data structures and computational geometry. We presented the first unconditional lower bound (in the pointer machine model) that works when the dimension is beyond logarithmic. We believe our work leads to two interesting problems. First, can we improve the query lower bound? There are still some room for improvement since the best known algorithm have query time $Q(n) = n^{1-\frac{1}{c}}$ and space $S(n) = n^{1+\varepsilon}$ for some $\varepsilon < 1$. We find improvements upon these bounds, either improving the algorithm or improving upon the lower bound presented in this paper to be interesting future work.

Second, can we obtain similar improvements in other problems where we have instances of high-dimensional orthogonal range reporting? Two important problems are multi-level data structures, e.g., range trees and range searching with respect to the Fréchet distance [5].

References

[1] P. Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. In *Symposium on Computational Geometry (SoCG)*, pages 339–346, 2012.

[2] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 149–158, 2009.

[3] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.

[4] P. Afshani, L. Arge, and K. G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012.

[5] P. Afshani and A. Driemel. On the complexity of range searching among curves. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 898–917, 2017.

[6] P. K. Agarwal. Range searching. In J. E. Goodman, J. O'Rourke, and C. Toth, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., 2016.

[7] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*. AMS Press, 1999.

[8] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016.

[9] T. M. Chan. Orthogonal Range Searching in Moderate Dimensions: k-d Trees and Range Trees Strike Back. In B. Aronov and M. J. Katz, editors, *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] T. M. Chan, Y. Nekrich, S. Rahul, and K. Tsakalidis. Orthogonal Point Location and Rectangle Stabbing Queries in 3-d. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 31:1–31:14, 2018.

[11] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.

[12] B. Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *Journal of the ACM (JACM)*, 37(2):200–212, 1990.

[13] B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Computational Geometry*, 5(5):237 – 247, 1996.

[14] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008.

[15] S. Rahul. *Improved Bounds for Orthogonal Point Enclosure Query and Point Location in Orthogonal Subdivisions in $\mathbb{R}^3$*, pages 200–211.

[16] S. Rahul. An (almost) optimal solution for orthogonal point enclosure query in $\mathbb{R}^3$. *Mathematics of Operations Research*, 45(1):369–383, 2020.

[17] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[18] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences (JCSS)*, 18(2):110 – 127, 1979.

[19] T. M. Thompson. *From Error-Correcting Codes Through Sphere Packings to Simple Groups*, volume 21. Mathematical Association of America, 1 edition, 1983.

[20] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357 – 365, 2005. International Colloquium on Automata, Languages and Programming (ICALP).

Appendix

Lemma 1 *For positive integers n and k where $1 \leq k \leq n$, it holds that*

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k.$$

Proof. We start by proving the lower bound. We first observe that for $k = 1$, the bound trivially holds

$$\left(\frac{n}{k}\right)^k = n = \binom{n}{k}.$$

Hence look at the case where $k > 1$ and let $0 < i < k \leq n$.

$$k \leq n \Leftrightarrow \frac{i}{n} \leq \frac{i}{k} \Leftrightarrow 1 - \frac{i}{k} \leq 1 - \frac{i}{n} \Leftrightarrow \frac{k-i}{k} \leq \frac{n-i}{n} \Leftrightarrow \frac{n-i}{k-i},$$

and from this we obtain

$$\left(\frac{n}{k}\right)^k = \frac{n}{k} \cdot \dots \cdot \frac{n}{k} \leq \frac{n}{k} \cdot \dots \cdot \frac{n-k+1}{1} = \binom{n}{k},$$

which gives us the desired lower bound. For the upper bound we see that

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-k+1)}{k!} \leq \frac{n^k}{k!}.$$

Since

$$e^k = \sum_{j=0}^{\infty} \frac{k^j}{j!}$$

we can look at the term where $j = k$ and get

$$e^k \geq \frac{k^k}{k!} \Leftrightarrow \frac{1}{k!} \leq \left(\frac{e}{k}\right)^k \Leftrightarrow \frac{n^k}{k!} \leq \left(\frac{en}{k}\right)^k,$$

which gives us the upper bound. □

Lemma 2 $\frac{\binom{A}{B}}{\binom{A}{B-C}} \geq (\frac{A}{2B})^C$, *if $B > 2$ and $A > 2B$.*

Proof. The proofs goes through, using some simple calculations and bounds,

$$\frac{\binom{A}{B}}{\binom{A}{B-C}} = \frac{\frac{A!}{B!(A-B)!}}{\frac{A!}{(B-C)!(A-B+C)!}} = \frac{(B-C)!(A-B+C)!}{B!(A-B)!}$$
$$= \frac{(A-B+1)\dots(A-B+C)}{(B-C+1)\dots B}.$$

and we are going to make this expression smaller by taking the smallest term in the numerator and raising it to the number of terms in the numerator, i.e. C and we take the biggest term in the denominator and do the same, raise it to the power C we get

$$\frac{(A-B+1)\dots(A-B+C)}{(B-C+1)\dots B} > \frac{(A-B+1)^C}{B^C}$$
$$> \left(\frac{A}{B} - 1\right)^C$$

and lastly since we assumed that $A > 2B$ we conclude the proof

$$\left(\frac{A}{B} - 1\right)^C > \left(\frac{A}{2B}\right)^C$$

□

Decomposing Polygons into Fat Components

Maike Buchin[*] Leonie Selbach[†]

Abstract

We study the problem of decomposing (i.e. covering or partitioning) polygons into components that are α-fat, which means that the aspect ratio of each subpolygon is at most α. We consider decompositions without Steiner points and decompose polygons by adding diagonals between their vertices. We present a polynomial-time algorithm for simple polygons that finds the minimum α such that an α-fat partition exists. Furthermore, we show that finding an α-fat partition or covering with minimum cardinality is NP-hard for polygons with holes.

1 Introduction

A decomposition of a polygon P is a set of subpolygons whose union is exactly P. If the subpolygons are not allowed to overlap, the set is a *partition* of P. Otherwise, we call the set a *covering*. Here, we consider decompositions without Steiner points and polygons are decomposed by adding diagonals between their vertices. Polygon decomposition problems arise in many theoretical and practical applications and can be categorized with regard to the type of subpolygon that is used [5]. We study decompositions into fat components.

A polygon P is called *α-fat* if its aspect ratio (AR) is at most α. There are different definitions for the aspect ratio and in this paper we consider the following two (see Fig. 1):

square-fatness: AR_{square} = ratio between the side length of the smallest axis-parallel square containing P and the side length of the largest axis-parallel square contained in P [4].

disk-fatness: AR_{disk} = ratio between the diameter of the smallest circle enclosing P (minimum circumscribed circle or MCC) and the diameter of the largest circle enclosed in P (maximum inscribed circle or MIC) [2].

A polygon P is called *α-small* if the side length of the enclosing square or respectively the diameter of the enclosing circle is at most α.

[*]Department of Computer Science, Ruhr University Bochum, maike.buchin@rub.de

[†]Department of Computer Science, Ruhr University Bochum, leonie.selbach@rub.de

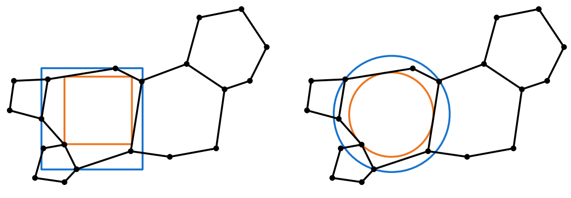

(a) square-fatness (b) disk-fatness

Figure 1: Comparison of fatness definitions. The partition is 1.5-fat in (a) and and 1.4-fat in (b).

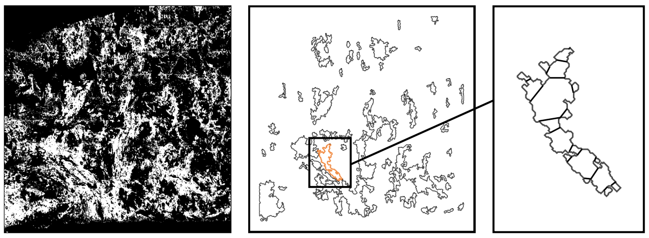


Figure 2: Fat decomposition of one connected component in a preprocessed tissue sample.

Our research is motivated by an application in protein diagnostics. Namely, the processing of tissue samples for molecular analysis using laser capture microdissection (LCM). The size and shape of the tissue fragments are two of the main factors that affect the success of the dissection. The fragmentation of tissue samples into feasible regions can be modeled as a polygon decomposition problem [6]. Preliminary results show that specifically round shapes of a certain size are desirable (see Fig. 2). As roundness can be measured by the fatness of a polygon, fatness is a suitable shape criterion for this practical application. The goal is to obtain decompositions such that the fatness is optimized.

Thus, we consider the min-fat partition problem and other related fat decomposition problems. The *min-fat partition problem* is finding the smallest α for which an α-fat partition exists. The *minimum α-fat partition* (or *covering*) *problem* is finding a partition (or covering) with minimum cardinality such that every subpolygon is α-fat. We have analog problems with α-smallness instead of α-fatness. An overview of related results regarding these problems is presented in Table 1.

Worman showed that the minimum α-small partition problem as well as the covering problem are NP-hard for polygons with holes if square-smallness is ap-

Table 1: Overview of results for fat decomposition problems using different smallness and fatness definitions. Results marked with * are presented in this paper.

Problems	Simple polygons	Polygons with holes
Minimum α-small partition	$\mathcal{O}(n^3m)$ (disk) [2]	NP-hard (square [7] and disk*)
Minimum α-small covering	?	NP-hard (square [7] and disk*)
Minimum α-fat partition	$\mathcal{O}(n^4m^3)$ (disk) [1]	NP-hard (square and disk)*
Minimum α-fat covering	?	NP-hard (square and disk)*
Min-fat partition	$\mathcal{O}(n^3m^5 \log n)$ (disk)*	?

plied [7]. They claim that the same reduction holds for disk-smallness, however, we show that this is only correct if some adjustments in the construction are made. In all following results, disk-smallness and -fatness was applied. Damian and Pemmaraju showed that the minimum α-small partition problem is polynomial-time solvable for simple polygons and that a faster 2-approximation algorithm exists [2]. Additionally, the authors presented an approximation algorithm for convex polygons. Damian proved that the minimum α-fat partition problem can be solved in polynomial time for simple polygons and conjectured that this problem is NP-hard for polygons with holes [1]. So far, the min-fat partition problem was just stated as an open problem by Damian [3]. This problem is of particular interest for us, as the corresponding partition is the most suitable for our practical application.

This paper includes two main results. In Section 2, we consider the min-fat partition problem using disk-fatness and present a polynomial-time algorithm for simple polygons. In Section 3, we consider the minimum α-fat partition and covering problem. We confirm the conjecture that these problems are NP-hard for polygons with holes and present the two reductions for square- and respectively disk-fatness. Additionally, we present a disk-smallness construction for the NP-hardness reduction of the minimum α-small partition problem in Section 4.

2 Min-fat partition problem for simple polygons

Our goal is to find an optimal partition of a given polygon such that the largest aspect ratio (AR) of any subpolygon is minimized. Here we consider the aspect ratio with regard to disk-fatness. The value of the largest AR in an optimal partition equals the desired α in the min-fat partition problem. With our algorithm, we extend the method of Damian [1] for this optimization problem.

Let P be a simple polygon with vertices labeled from 1 to n counterclockwise. A diagonal (i,j) is a line segment that connects two vertices i and j and does not intersect the outside of P. Let $G(P)$ be the visibility graph of P consisting of the n vertices of P and m diagonals. We define S as the set consisting of all vertices and edges of $G(P)$. For each diagonal (i,j) with $i < j$, let $P_{i,j}$ be the subpolygon with vertices $\{i, i+1, \dots, j\}$ (see Fig. 3a). To solve the min-fat partition problem, we use a dynamic programming approach. Namely, we compute optimal partitions for each $P_{i,j}$ belonging to a diagonal (i,j) with increasing value of $j-i$.

Let $Z_{i,j}$ be an optimal partition of $P_{i,j}$. Let Q be the polygon in $Z_{i,j}$ adjacent to (i,j). Note that the vertices of Q represent a path q from i to j in the visibility graph and we have $Z_{i,j} = \bigcup_{(k,l)\in q} Z_{k,l} \cup Q$ (see Fig. 3b). Thus, we can find an optimal partition by computing the optimal path q. For this we define edge weights $w(i,j)$ as the value of an optimal partition $Z_{i,j}$ of $P_{i,j}$:

$$w(i,j) = \max_{P' \in Z_{i,j}} AR(P') = \max\{\max_{(k,l)\in q} w(k,l), AR(Q)\}$$

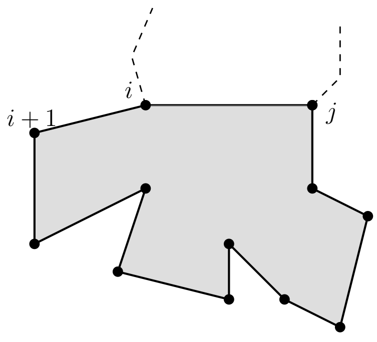
for $AR(Q) = d(\mathrm{MCC})/d(\mathrm{MIC})$ being the ratio between the diameters $d(\cdot)$ of the minimum circumscribed circle MCC and maximum inscribed circle MIC of Q. If j is equal to $i+1$, the partition $Z_{i,i+1}$ is empty and we set $w(i,i+1) = 0$. Otherwise, $w(i,j)$ equals the value of the largest AR in an optimal min-fat partition of $P_{i,j}$.

However, the computation of $w(i,j)$ presents the following problem: Finding the path q and its correct edges requires knowledge about the resulting polygon Q and its aspect ratio, which is not available beforehand. To solve this, we consider different reduced visibility graphs that ensure that the aspect ratio of each possible polygon is below a certain value. We compute optimal paths on these graphs and then choose the best one. For this, we consider all pairs of circles (C, I) such that the following properties (A) hold:

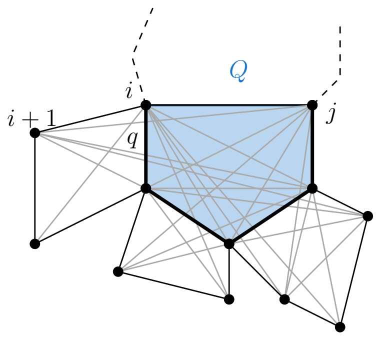
- (i,j) lies completely inside of C and outside of I,
- I is tangent to 3 elements in S,
- and C either touches 3 vertices of P or its diameter connects 2 vertices.

Note that for any subpolygon the pair (MCC,MIC) fulfills these properties. For each (C, I), we denote by $G_{i,j}^{(C,I)}$ be the subgraph of $G(P)$ consisting of edges that lie outside of I and inside of $P_{i,j}$ and C (see Fig. 3c).

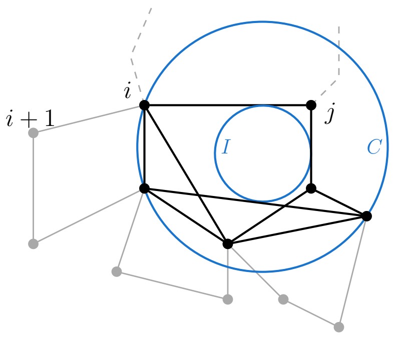
Using these graphs, we are able to compute the weights $w(i,j)$ by using dynamic programming with increasing values of $j-i$. For each pair of circles (C, I)



(a) Subpolygon $P_{i,j}$.



(b) Polygon Q in an optimal partition.



(c) reduced visibility Graph $G_{i,j}^{(C,I)}$.

Figure 3: In (a): Subpolygon $P_{i,j}$ induced by the edge (i,j). In (b): Polygon Q (blue) in an optimal partition of $P_{i,j}$ induced by a path q (fat edges) in the visibility graph. In (c): Reduced visibility graph $G_{i,j}^{(C,I)}$ (fat edges) for a pair of circles (C,I).

that fulfill (A), we compute a corresponding weight $W(C,I)$ and use those values to determine an intermediate edge weight $w'(i,j)$ as follows:

$$W(C,I) = \min_{q \in G_{i,j}^{(C,I)}} \max\{\max_{(k,l)\in q} w'(i,j), d(C)/d(I)\}$$
$$w'(i,j) = \min_{(C,I) \text{ fulfilling (A)}} W(C,I)$$

The weights of all edges except for (i,j) have already been computed. We search in $G_{i,j}^{(C,I)}$ for the path q such that the value $\max\{\max_{(k,l)\in q} w'(k,l), d(C)/d(I)\}$ is minimized. We denote this optimal value as $W(C,I)$. Over all possible combinations of circles, we search for the pair (C,I) with minimum $W(C,I)$ and set $w'(i,j) = W(C,I)$. If no pair of circles exist, we set the weight $w'(i,j) = 0$. We can show by induction that $w'(i,j)$ is actually equal to the largest aspect ratio in the corresponding partition and that this partition is indeed optimal and hence $w'(i,j) = w(i,j)$.

Lemma 1 *For an edge (i,j) in the visibility graph $G(P)$ with $j \neq i+1$, let $w'(i,j)$ be the computed weight and $Z_{i,j}$ the corresponding partition. Then,* $w'(i,j) = \max_{P' \in Z_{i,j}} AR(P')$.

Proof. Let (C',I') be the pair for circles, q' the corresponding path in $G_{i,j}^{(C',I')}$ and (k',l') the edge in q' such that

$$w'(i,j) = W(C',I') = \max\{\max_{(k,l)\in q'} w'(k,l), d(C')/d(I')\}$$
$$= \max\{w'(k',l'), d(C')/d(I')\}. \quad (1)$$

The computation induces a partition $Z_{i,j}$ in which we denote the polygon adjacent to (i,j) by Q'. By induction over $j-i$, we show that $w'(i,j) = \max_{P' \in Z_{i,j}} AR(P')$.
For $j-i=2$, consider an edge $(i,j) = (i,i+2)$ in the visibility graph. There is only one possible pair (C',I'), which is equal to (MCC,MIC) of $P_{i,j}$. Thus, there is only one possible path $q' = \{(i,i+1),(i+1,i+2)\} \in G_{i,j}^{(C',I')}$. This path induces the partition $Z_{i,j} = \{P_{i,j}\}$. Therefore, we have $Q' = P_{i,j}$ and $d(C')/d(I') = AR(Q')$. Since $w'(i,i+1) = w'(i+1,i+2) = 0$, we have

$$w'(i,j) = \frac{d(C')}{d(I')} = AR(Q') = \max_{P' \in Z_{i,j}} AR(P').$$

Our induction hypothesis is that $w'(i,j) = \max_{P' \in Z_{i,j}} AR(P')$ for all (i,j) such that $j-i \leq s$. Now, we consider an edge (i,j) such that $j-i = s+1$. Using the induction hypothesis (IH), we have the following:

$$\max_{P' \in Z_{i,j}} AR(P') = \max_{(k,l)\in q}\{\max_{P' \in Z_{k,l}} AR(P'), AR(Q)\}$$
$$\underset{(IH)}{=} \max\{w'(k',l'), AR(Q)\}$$

Since $d(C')/d(I') \geq AR(Q')$, the inequality $w'(i,j) \geq \max_{P' \in Z_{i,j}} AR(P')$ holds. For the opposite inequality we consider the two possible cases for Equation 1:

Case 1: $w'(k',l') \geq d(C')/d(I')$ $(\geq AR(Q'))$.
Obviously, we have $w'(i,j) = w'(k',l') = \max_{P' \in Z_{i,j}} AR(P')$.

Case 2: $w'(k',l') < d(C')/d(I')$.
Then, the weight $w'(i,j)$ is equal to $d(C')/d(I')$.

a) $d(C')/d(I') = AR(Q')$.
We have $w'(i,j) = AR(Q') = \max_{P' \in Z_{i,j}} AR(P')$.

b) $d(C')/d(I') > AR(Q')$.
Let (C'',I'') be pair of circles such that C'' is the MCC and I'' the MIC of Q'. During our computation, we consider (C'',I'') and find the same path q' in $G_{i,j}^{(C'',I'')}$. Thus,

$W(C'', I'') \leq \max\{w'(k', l'), d(C'')/d(I'')\} = \max\{w'(k', l'), AR(Q')\}$. Since both $w'(k', l') < d(C')/d(I')$ and $AR(Q') < d(C')/d(I')$, it follows that $W(C'', I'') < W(C', I') = w'(i, j)$. This contradicts our assumption that $w'(i, j)$ is minimal.

With this, we showed that $w(i, j) = \max_{P' \in Z_{i,j}} AR(P')$ holds. □

Lemma 2 *The computed partition $Z_{i,j}$ is an optimal partition of $P_{i,j}$, meaning that the largest aspect ratio of any subpolygon is minimized.*

Proof. We show this by induction over $j - i$. For $j - i = 1$, we consider an edge $(i, j) = (i, i + 1)$. Thus, $Z_{i,j}$ is the empty set. Our induction hypothesis is that the computed partition $Z_{i,j}$ is an optimal partition of $P_{i,j}$ for all (i, j) such that $j - i \leq s$. Now, we consider an edge (i, j) such that $j - i = s + 1$. Let $Z^*_{i,j}$ be an optimal partition of $P_{i,j}$. Hence, $\max_{P' \in Z^*_{i,j}} AR(P')$ is minimal over all partitions. Let Q^* be the polygon in $Z^*_{i,j}$ adjacent to (i, j) and let q^* be the path induced by Q^*. The path q^* is contained in $G^{(C^*,I^*)}_{i,j}$ for $(C^*, I^*) = (\mathrm{MCC}, \mathrm{MIC})$ of Q^*. Thus, q^* is a candidate for the min-max path computed by our algorithm and we have $W(C^*, I^*) = \max\{\max_{(k,l)\in q^*} w'(k, l), AR(Q^*)\}$. Let $Z_{i,j} = \bigcup_{(k,l)\in q'} Z_{k,l} \cup Q$ be the partition computed by our algorithm and (C', I') the corresponding circles. We have:

$$\begin{aligned}
& W(C', I') \leq W(C^*, I^*) \\
\Rightarrow \quad & \max\{\max_{(k,l)\in q'} w'(k, l), d(C')/d(I')\} \\
& \leq \max\{\max_{(k,l)\in q^*} w'(k, l), AR(Q^*)\} \\
\underset{Lemma\ 1}{\Rightarrow} \quad & \max_{P' \in Z_{i,j}} AR(P') \leq \max_{P' \in Z^*_{i,j}} AR(P').
\end{aligned}$$

Since the maximal aspect ratio in $Z_{i,j}$ is not larger that the maximal aspect ration in the optimal partition, it follows that the computed partition is optimal. □

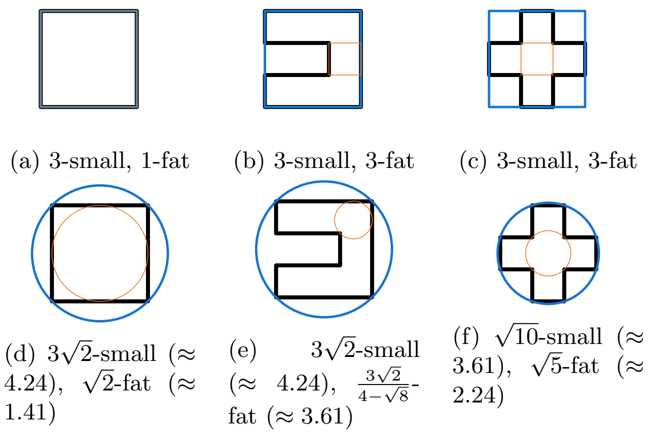
Theorem 3 *For a simple polygon P, the min-fat partition problem using disk-fatness can be solved in time $\mathcal{O}(n^3 m^5 \log n)$ with n being the number of vertices of P and m being the number of edges in the visibility graph $G(P)$.*

Proof. First, we have to compute the visibility graph of P which takes $\mathcal{O}(n + m)$ time. For every edge (i, j) in the visibility graph, we determine an optimal partition $Z_{i,j}$ by computing the optimal weight $w(i, j)$. For each (i, j), we consider pairs of circles (C, I). There are $\mathcal{O}(n^3)$ circles C and $\mathcal{O}(m^3)$ circles I to consider. Computing the optimal path in $G^{(C,I)}_{i,j}$ can be done by an adjusted version of Dijkstra's Algorithm and therefore takes $\mathcal{O}(m \log n)$ time. Thus, the overall runtime of the algorithm is $\mathcal{O}(n^3 m^5 \log n)$. □

3 Minimum α-fat decomposition problems for non-simple polygons

This section deals with the minimum α-fat decomposition problems on polygons with holes. We consider the corresponding *α-fat partition* (resp. *covering*) *decision problem.* That is, deciding whether there exists an α-fat partition (resp. covering) with a given number of components. We show NP-hardness with a reduction from *planar 3,4-SAT*. For a boolean formula ϕ, let $G(\phi) = (V, E)$ be the graph with $V = U \cup C$ and $E = \{(u, c) \mid u \in U, c \in C, u \text{ or } \bar{u} \text{ is a literal in } c\}$, where U are the literals and C the clauses. Planar 3,4-SAT is the problem of deciding if ϕ is satisfiable under the following three restrictions: In conjunctive normal form, ϕ has exactly 3 literals per clause, each literal appears in at most 4 clauses, and the graph $G(\phi)$ is planar. We construct a polygon representing the graph $G(\phi)$ that has an α-fat partition of size k if and only if ϕ is satisfiable. The value of α is fixed and k is determined during this construction.

The related result for α-smallness was proven by Worman [7]. In their construction only (orthogonal) subpolygons below a certain height and width are feasible, as only the side length of the enclosing square is bounded. However, with α-fatness it is the aspect ratio that is bounded. This ratio might differ considerably even for polygons that have the same smallness factor α. This makes the construction more challenging when using fatness rather than smallness. Additionally, the fatness and smallness factors are affected by the definition (square or disk) that is applied. Hence, it is no surprise that the polygon construction for the NP-hardness proof differ also based on these respective definitions. Figure 4 exemplifies for small polygons on a grid the differences that can occur when considering smallness and fatness for both definitions.


(a) 3-small, 1-fat (b) 3-small, 3-fat (c) 3-small, 3-fat

(d) $3\sqrt{2}$-small ($\approx$ 4.24), $\sqrt{2}$-fat ($\approx$ 1.41) (e) $3\sqrt{2}$-small ($\approx$ 4.24), $\frac{3\sqrt{2}}{4-\sqrt{8}}$-fat ($\approx 3.61$) (f) $\sqrt{10}$-small ($\approx$ 3.61), $\sqrt{5}$-fat ($\approx$ 2.24)

Figure 4: Comparison of smallness and fatness factors for the square (top) and disk (bottom) definition. The denoted values are sharp bounds.

3.1 Reduction for square-fatness

The goal is to construct a polygon that has an α-fat partition of a certain size k if and only if a boolean formula ϕ is satisfiable. In the following construction where square-fatness is applied, we set $\alpha = 1.2$. All polygon components are orthogonal and at most 5 units wide, thus the side length of the enclosing squares cannot exceed 6. The construction consists of three different polygon components: variable, wire and clause polygons.

The *variable polygon* is shown in Figure 5. It has four terminals at which wires can be connected. This polygon can be minimally partitioned into 8 α-fat subpolygons in two ways. These partitions represent the `True` and `False` assignment, which is carried to the corresponding clauses.

Each wire consists of a set of individual *wire polygons* that are connected with each other (as depicted in Figure 6) to carry the variable assignment to the clause polygon. The wires can be attached at the terminals in two possible orientations (see Fig. 7) depending on whether the variable appears in the clause negated or unnegated. If the wire is connected in the unnegated orientation and the variable is set to `True`, the gray polygon in Figure 7a can cover the top part of the wire as well, but this is not the case if the variable is set to `False`. The reverse is true if the wire is connected in the negated orientation. If a wire is partitioned in this way (unnegated position and `True` assignment or negated position and `False` assignment), we say that it carries `True`.

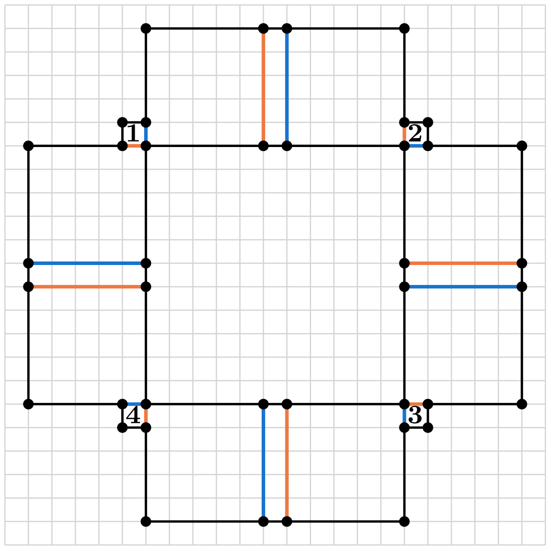


Figure 5: The variable polygon with four terminals indicated by the bold numbers 1, 2, 3, 4 and its minimal 1.2-fat partitions representing the `True` (blue) and `False` (orange) assignment.

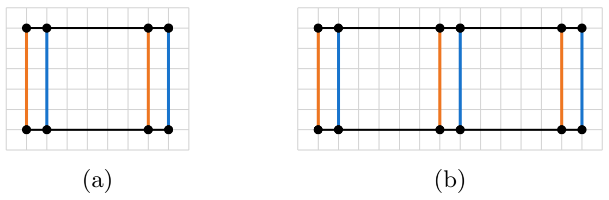
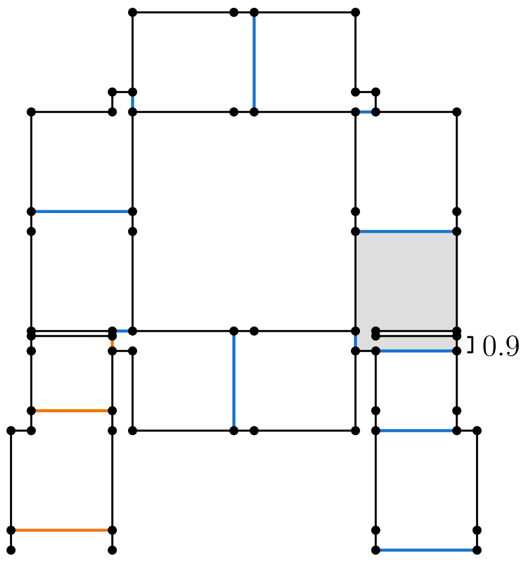
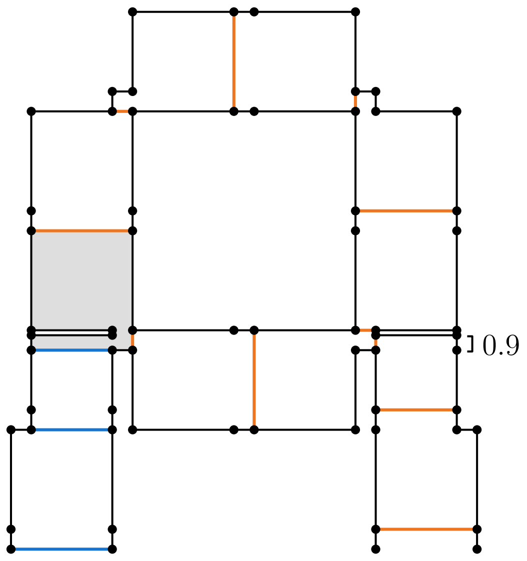


Figure 6: A single wire polygon (a) with its two partitions that represent the `True` (blue) or `False` (orange) assignment. Two wire polygons (b) are connected and the respective truth assignment is transmitted.



(a) `True` assignment of variable.



(b) `False` assignment of variable.

Figure 7: Attaching a wire to the variable in unnegated orientation (on bottom right terminal) or in negated orientation (on bottom left terminal). The latter switches the `True`/`False` value.

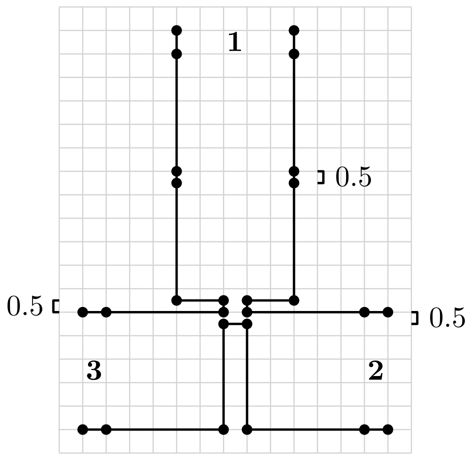


Figure 8: The clause polygon with three terminals indicated by the bold numbers $1, 2, 3$.

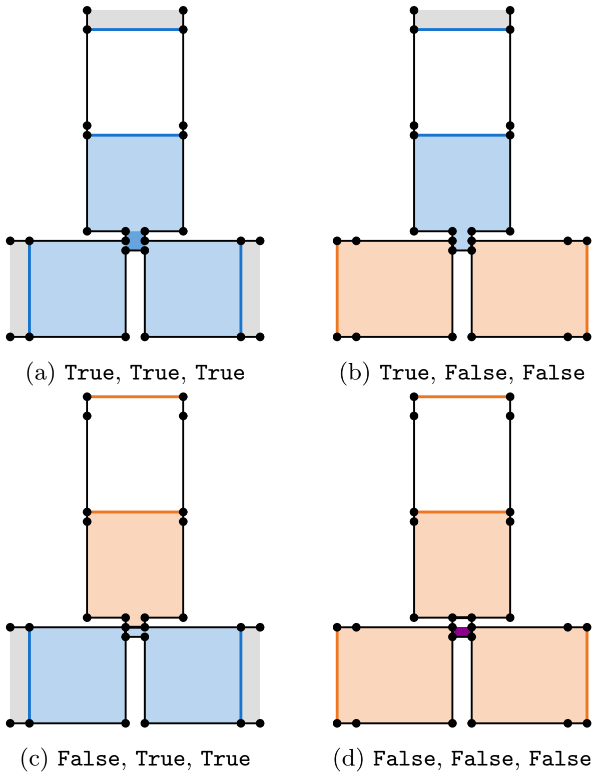


Figure 9: Partition of the clause polygon depending on different assignments that are transmitted by the wires (`True` blue edges, `False` orange edges).

The variable assignment is carried to the *clause polygon* (see Fig. 8). For each of the three variables contained in the clause, this polygon has one terminal, where the wires will be attached. Depending on the values these wires carry, a different number of polygons is needed to partition the clause polygon into 1.2-fat components. If some wire carries `True` (see Fig. 9a to 9c), the tip of the connected terminal (gray) is already

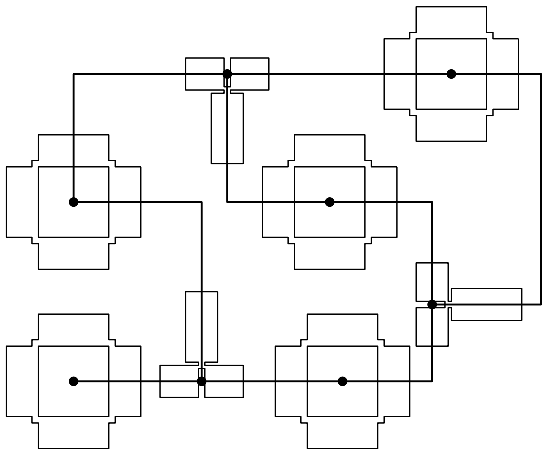


Figure 10: Placement of 5 variable and 3 clause polygons on a planar orthogonal grid drawing.

covered and center part of the clause polygon (dark blue) can be covered as well. If more than one wire carries `True` either one of the corresponding polygons (light blue) can be used to cover the center. In either case, the partition requires exactly four polygons. If all wires carry `False` (see Fig. 9d), the part of center of the clause polygon depicted in magenta cannot be covered by any of the bigger polygons (in light orange). Thus, an additional fifth polygon is needed to cover this area. As a 1.2-fat polygon is required, only the 1x1 square can be chosen.

The whole polygon representing $G(\phi)$ is constructed based on a *planar orthogonal grid drawing* of $G(\phi)$. That is a planar embedding of the graph such that every vertex is located at an integer grid point, the edges are non-overlapping, and every edge is a chain of orthogonal lines that bend at integer grid points. A schematic example for the placement of variable and clause polygons on the vertices of the drawing can be seen in Figure 10. To construct the edges, the wires have to be bend, shifted or offset. This is achieved by the constructions presented in Figure 11. The drawing of $G(\phi)$ is scaled to accommodate the size of the variable and clause polygons as well as the needed adjustments of the wire polygons.

As we consider the decision problem, the number k of allowed subpolygons is fixed. We have $k = 8v + 4c + w$ where v is the number of variables, c the number of clauses and w the number of wire polygons needed in the construction. Bending, shifting and offsetting a wire counts as 3, 2, and respectively 5 wire polygons. Note that we can find a minimum 1.2-fat covering with the same number of components as the minimum partition and that the constructed polygon is orthogonal. This means that the following theorem remains true for the corresponding covering problem and also for orthogonal polygons with holes.

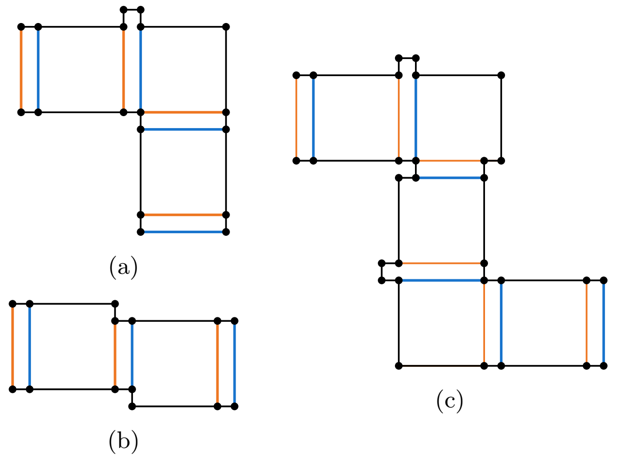


Figure 11: Bending (a), shifting (b) and offsetting (c) a wire that carries `True` (blue edges) or `False` (orange edges).

Theorem 4 *The α-fat partition decision problem is NP-complete for polygons with holes if square-fatness is applied.*

Proof. Given a set of k α-fat polygons, we can verify in polynomial time if this is a partition of our polygon P. Hence, the problem is in NP.

Let ϕ be an instance of planar 3,4-SAT and P be the polygon representing $G(\phi)$ constructed as described in Section 3. We show that P has a minimum 1.2-fat partition of a certain size k if and only if ϕ is satisfiable.

Assume that ϕ can be satisfied with a truth assignment T. Then, the polygon can be minimally partitioned into $k = 8v + w + 4c$ components. The values v and c are the numbers of variables and clauses respectively. The value w corresponds to the number of wire polygons needed to construct $G(\phi)$ with P. For each of the v variables, we partition the variable polygon into 8 components according to T. The wires are partitioned into w subpolygons. Depending on the assignment of the variables the tip of the terminal (the part where the wire and clause polygon overlap) might be uncovered. Since the assignment T satisfies ϕ, we know that for each clause there is at least one wire carrying `True` such that the tip of the terminal is already covered and the clause polygon can be partitioned into 4 components.

Assume that P has a minimum 1.2-fat partition Z of size $k = 8v+w+4c$ as above. There are two ways how to minimally partition a variable polygon into 1.2-fat polygons, which both require 8 components. To partition a sequence of l wire polygons, l subpolygons are needed. If the tip of the wire at the variable terminal is already covered, the wire carries `True` and the tip at the clause terminal will be covered as well. If we use w polygons to partition the wires, $4c$ polygons remain for the partition of the clauses. Since each of the c clause polygons can only be partitioned into 4 components if the tip of one of the terminals is already covered by another polygon – which is only the case if the corresponding wire carries `True` – we know that each clause is satisfied. Hence, ϕ is satisfied. $\square$

3.2 Reduction for disk-fatness

The construction presented in Section 3.1 does not work if disk-fatness is applied. For all previously feasible (with $AR_{square} \leq 1.2 = 6/5$) subpolygons to remain feasible, we would have to set $\alpha_{disk} = 6/5\sqrt{2}$. However, there exist other subpolygons in the construction with $AR_{disk} \leq 6/5\sqrt{2}$ but $AR_{square} > 6/5$ – specifically at the wire bends and inside the clause polygon. These would then become feasible and cause an inconsistent transmission of `True`/`False` values. We adjust the construction such that the aspect ratio AR_{disk} of all subpolygons that are supposed to be feasible is at most $\alpha = \sqrt{61}/5 \approx 1.56$.

For variable polygons and their attachment to the wires nothing changes. Wires can be shifted in the same way, but bending and offsetting has to be adjusted. We have to remove the 1x1-square bulge at the corners because the corner polygon in the orange partition in Figures 11a and 11c would no longer be feasible. The adjusted wires and their feasible partitions are shown in Figure 12 and the respective corner polygons are depicted in Figure 13. The two polygons in Figure 13a and 13b have the correct aspect ratio of $AR_{disk} = \sqrt{61}/5$. The other polygon shown in 13c has a MCC of diameter $6\sqrt{2}$ and MIC of diameter $2(9-\sqrt{40})$. Therefore, the aspect ratio is larger than $\sqrt{61}/5 = \alpha$, which means that it is not feasible in our construction. Note that this polygon, however, would be feasible regarding square-fatness with $\alpha = 1.2$.

Furthermore, we have to adjust the clause polygon. This is done by changing the placement of four boundary vertices, which are marked with a x in Figure 14. Figure 15 exemplifies the idea behind this construction. Figure 15a shows the previous construction for

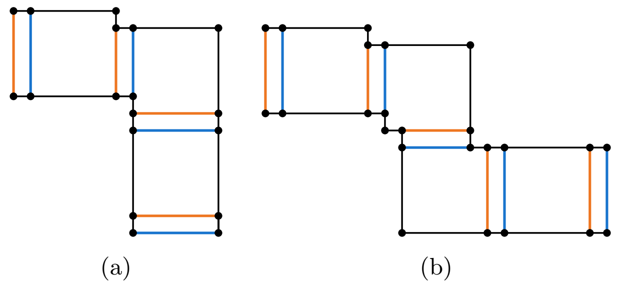
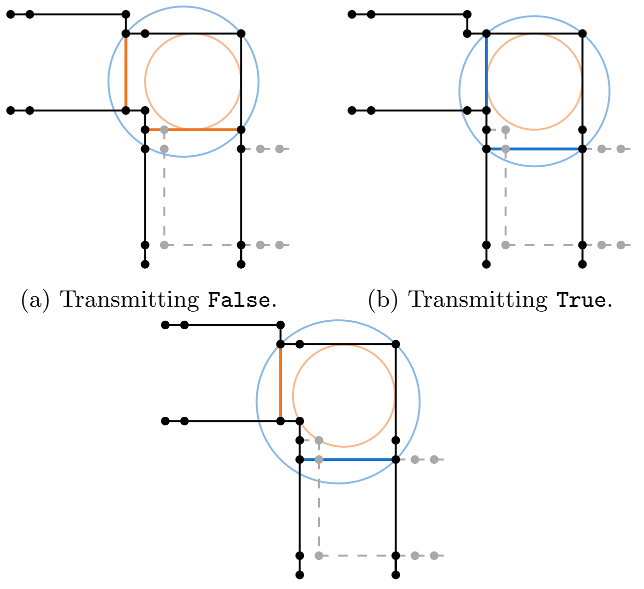


Figure 12: Bending (a) and offsetting (b) a wire that carries `True` (blue edges) or `False` (orange edges) in the disk-fatness construction.


(a) Transmitting False. (b) Transmitting True.

(c) Not feasible.

Figure 13: Three possible corner polygons in a wire polygon that is bend (or offset in gray) with their corresponding MCC and MIC. Only the polygons in (a) and (b) are α-fat with the given $\alpha = \sqrt{61}/5$ and hence feasible.

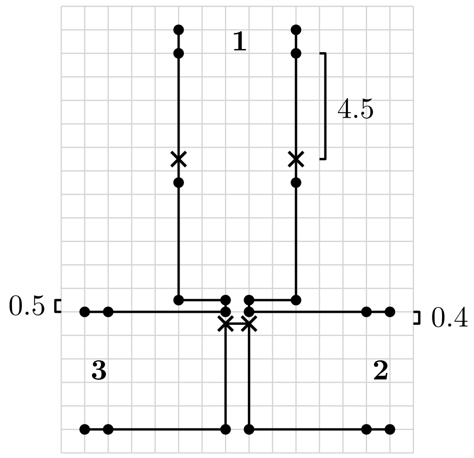


Figure 14: The clause polygon for disk-fatness. The marked vertices are moved in comparison to the construction for square-fatness.

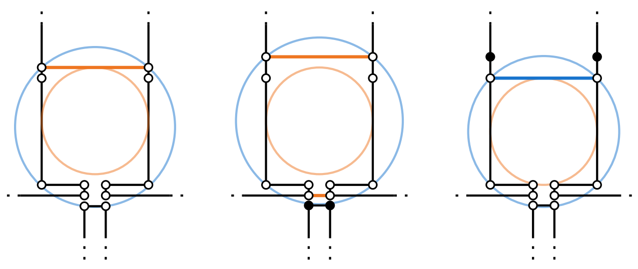
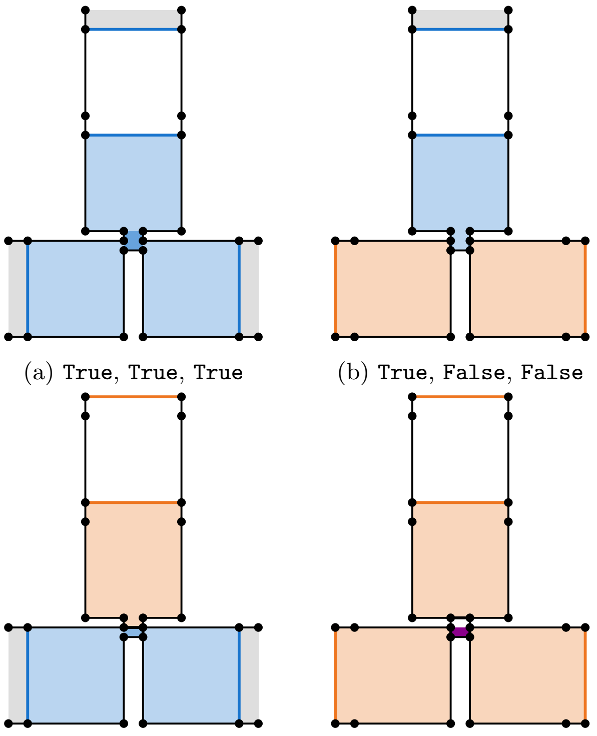

(a) (b) (c)

Figure 15: $\sqrt{61}/5$-fat subpolygons in the different clause polygons. The white vertices contained in the subpolygons. The clause polygon for square-fatness (a) contains a subpolygon that is not supposed to be feasible. With the adjusted construction for disk-fatness the correct subpolygons are feasible (b and c).


(a) True, True, True (b) True, False, False

(c) False, True, True (d) False, False, False

Figure 16: Partition of the clause polygon for disk-fatness depending on different assignments that are transmitted by the wires (True blue edges, False orange edges).

square-fatness. The illustrated polygon was not feasible regarding square-fatness, because its aspect ratio is $AR_{square} = 6.5/5$ - but it is feasible regarding disk-fatness with the given α. As depicted this polygon would be able to cover the entire center of the clause polygon despite coming from a wire carrying False. With the adjusted placement of the boundary vertices, we can ensure that the center can only be covered by a subpolygon coming from a wire carrying True. Let P_F and P_T be the two polygons depicted in Figure 15b and 15c respectively. The polygon P_F has $AR_{disk} = \sqrt{61}/5 = \alpha$ and P_T has $AR_{disk} < \alpha$. The position of the four vertices marked in Fig. 14 was adjusted in such a way that the lower two are contained in the MCC of P_T but not in the MCC of P_F. As the aspect ratio of P_F is exactly α, no larger polygon (coming from terminal 1) can be chosen to cover the area between these vertices and thereby the center area of the clause polygon. Thus, the center can only be covered if another wire carries `True` or an additional fifth polygon is included in the partition (see Fig. 16).

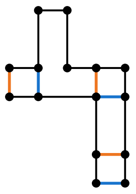
Note that the construction resulting from these adjustments would not work for square-fatness. Again other subpolygons that interfere with the correct transmission of `True` and `False` values would become feasible. For example, the corner polygon in Fig. 13c would be feasible with the previous α for square-fatness. Using the adjusted construction, we can reduce from planar 3,4-SAT and thus prove the NP-completeness in the same way as before. As argued earlier, this result remains true for the covering problem and additionally for orthogonal polygons with holes.

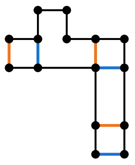
Theorem 5 *The α-fat partition decision problem is NP-complete for polygons with holes if disk-fatness is applied.*

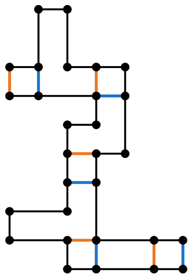
4 Minimum α-small decomposition problem with disk-fatness

Worman showed that the minimum α-small decomposition problems are NP-complete if the definition of square-smallness is used [7]. In their construction, they used $\alpha_{square} = 3$ and all feasible subpolygons can be enclosed with a 3x3 square. They claim that their construction holds for disk-smallness as well if $\alpha_{disk} = \sqrt{18}$, which is the diameter of the MCC of the 3x3 square. However, this is incorrect. Their feasible wire polygons are 1x3 rectangles, but the wires can also be partitioned in 1x4 rectangles. The latter were not feasible for α_{square}, but they would become feasible with α_{disk} because the diameter of their MCC is only $\sqrt{17}$. Additionally, there are subpolygons inside the clause polygon, which have an MCC of diameter $\sqrt{17}$ but are not supposed to be feasible. For their result to be true, a different α_{disk} has to be chosen and additionally the construction has to be adjusted such that the correct polygons stay feasible with the new α.

We choose $\alpha_{disk} = \sqrt{13}$. The variable polygon stays the same, as α_{disk} is exactly the diameter of the MCC of the subpolygons that supposed to be feasible. The basic wire polygon can also stay the same, but bending and offsetting a wire has to be adjusted (see Fig. 17). Similar to the adjustments in Section 3, this is done by removing the bulges at the corner polygons. Note that all feasible subpolygons now fit inside a 2x3 rectangle and therefore fulfill α_{disk}. Additionally, we have to adjust the clause polygon. It can be constructed as a slimmed down version of the clause polygon presented in Section 3 (see Fig. 18). The center square that can only be covered by another polygon coming from the terminals if at least one of the connected wires carries `True`. Note that this adjusted construction does


(a) Bending a wire for square-smallness.


(b) Bending a wire bend for disk-smallness.


(c) Offsetting a wire for square-smallness.

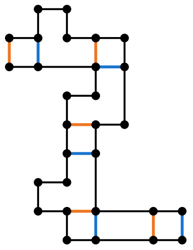
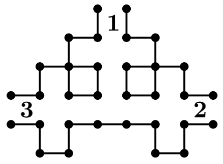
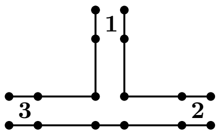

(d) Offsetting a wire for disk-smallness.

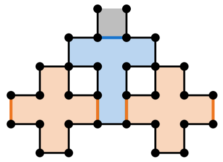
Figure 17: Comparison of wire construction for square-smallness (left) and disk-smallness (right). Bending (top) and offsetting (bottom) a wire that carries `True` (blue edges) or `False` (orange edges).



(a) The clause polygon for square-smallness.



(b) The clause polygon for disk-smallness.


(c) `True, False, False` for square-smallness.

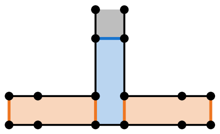

(d) `True, False, False` for disk-smallness.

Figure 18: Comparison of clause construction for square-smallness (left) and disk-smallness (right). Bottom: Partition of clause polygon for one truth assignment transmitted by wires.

not work for square-smallness anymore. Using the presented construction, we can prove that the problem is NP-complete with an analog reduction from planar 3,4-SAT. Again, the same result holds for the covering problem and also for orthogonal polygons with holes.

Theorem 6 *The α-small partition decision problem is NP-complete for polygons with holes if disk-smallness is applied.*

5 Conclusion

We answered some of the open problems on decompositions of polygons into fat components. For the min-fat partition problem on simple polygons, we presented a polynomial-time algorithm that applies disk-fatness. This algorithm takes $\mathcal{O}(n^3 m^5 \log n)$ for n being the number of vertices and m the number of edges in the visibility graph. This runtime is likely to be inefficient for practical application. Thus, our future work contains finding a faster (approximation) algorithm.

Furthermore, we considered the α-fat partition and covering problems on polygons with holes. We proved that both problems are NP-hard for square- as well as disk-fatness. Additionally, we included the disk-smallness construction for the NP-hardness proof of the α-small partition and covering problem.

For polygons with holes, the min-fat partition problem remains open. It is also unclear if the minimum α-fat or min-fat covering problems are solvable for simple polygons. Moreover, there are no results for any related small or fat decomposition problems that allow Steiner points yet.

References

[1] M. Damian. Exact and approximation algorithms for computing optimal fat decompositions. *Computational Geometry*, 28(1):19–27, 2004.

[2] M. Damian and S. V. Pemmaraju. Computing optimal diameter-bounded polygon partitions. *Algorithmica*, 40(1):1–14, 2004.

[3] E. Demaine and J. O'Rourke. Open problems from CCCG 2002. In *Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 178–181, 2003.

[4] M. J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry*, 8(6):299–316, 1997.

[5] J. M. Keil. Polygon decomposition. *Handbook of computational geometry*, 2:491–518, 2000.

[6] L. Selbach, T. Kowalski, K. Gerwert, M. Buchin, and A. Mosig. Shape Decomposition Algorithms for Laser Capture Microdissection. In *20th International Workshop on Algorithms in Bioinformatics*, volume 172 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:17, 2020.

[7] C. Worman. Decomposing polygons into diameter bounded components. In *Proceedings of the 15th Canadian Conference on Computational Geometry*, pages 103–106, 2003.

Convex Bichromatic Quadrangulation of Point Sets with Minimum Color Flips*

Allan Sapucaia[†] Andre A. Cire[‡] Pedro J. de Rezende[§] Cid C. de Souza[¶]

Abstract

In this paper, we investigate an $\mathbb{NP}$-hard optimization variant of the well-known convex quadrangulation problem. Given a point set P and a bicoloring C, we wish to find an alternative bicoloring C' that is (a) closest to C with respect to their symmetric difference, and (b) such that P admits a bichromatic convex quadrangulation under the new coloring C', i.e., a partition of the convex hull of P into convex quadrangles where the endpoints of each edge have distinct colors in C'. Thus, the resulting problem combines, and provides solution insights to, two decision problems from the literature: whether P has a convex quadrangulation, and whether P admits a bichromatic convex quadrangulation under the original coloring C.

Due to our optimization perspective, we present two mathematical programming approaches to address the problem. The first is an extension of the standard integer linear model for the quadrangulation problem, and the second is a novel set-partition formulation that enumerates colored convex quadrangles. We also derive new heuristics that exploit the models' linear programming relaxations, drawing connections to existing matching approaches used for quadrangulation problems in the literature. We present an empirical study assessing the effectiveness of the models for both exact and heuristic approaches, along with comparisons between the heuristics. The benchmark used to evaluate the performance of the models and heuristics is also made publicly available.

1 Introduction

A *quadrangulation* of a set $P \subset \mathbb{R}^2$ with n points is a partition of its convex hull, denoted by CH(P), into interior disjoint quadrangles whose vertices belong to P and are *empty*, which means that no points of P lie in their interior. We say that P is *quadrangulable* if it admits a quadrangulation. A necessary condition for a point set P to be quadrangulable is that CH(P) has an even number of vertices [3].

One of the most studied problems in computational geometry, both in theory and in practice, is the Triangulation Problem, i.e., how to partition the convex hull of a point set into interior disjoint triangles. The study of quadrangulations follows naturally because it allows for the decomposition to use half as many partitions.

A quadrangulation is said to be *convex* if its quadrangles are convex. The Convex Quadrangulation of Point Sets Problem (CQPS) decides whether a point set admits a convex quadrangulation, which is a required property in many practical applications [13]. The question of its complexity, however, is still open. As there are no known polynomial algorithms for this problem, applications often relax the quadrangulation requirement and strive for including as many quadrangles as possible, either allowing some triangles [1, 9] or employing Steiner points to achieve a quadrangulation [6, 10]. An exact algorithm for an optimization version of the problem was proposed in [5], with complexity $O(n^{3h+1})$, where h is the number nested convex hulls of P.

A *coloring* of a point set P is a partition of its points into disjoint subsets. Each of these subsets is referred to as a *color*. A *bicoloration* $C = (R, B)$ of a point set P is a coloring of P into exactly two colors R and B. We say that a point $p \in P$ is colored *red*, with respect to (w.r.t.) $C = (R, B)$ if $p \in R$. Similarly, if $p \in B$, p is said to be colored *blue*. A point set P together with a bicoloration $C = (R, B)$ defines a *bichromatic point set*, denoted $P|C$.

Given a bichromatic point set $P|C$, a quadrangle with vertices in P is said to be *bichromatic* if all its edges have endpoints of different colors. A quadrangulation of a point set P is said to be *bichromatic w.r.t. a bicoloration C* if all its quadrangles are bichromatic. In the Convex Quadrangulation of Bichromatic Point Sets Problem (CQBPS), we decide whether a bichromatic point set admits a *bichromatic quadrangulation.* This problem was proposed in [8], where it was proved to be $\mathbb{NP}$-hard.

All colorings are to be understood as bicolorations

*This work was supported in part by grants from: *Brazilian National Council for Scientific and Technological Development* (CNPq) #313329/2020-6, #309627/2017-6, #306454/2018-1; *São Paulo Research Foundation* (FAPESP) #2018/26434-0, #2018/14883-5, #2014/12236-1.

[†]Institute of Computing, University of Campinas, Campinas, Brazil `allansapucaia@gmail.com`

[‡]Department of Management, University of Toronto Scarborough, `andre.cire@utoronto.ca`

[§]Institute of Computing, University of Campinas, Campinas, Brazil `rezende@ic.unicamp.br`

[¶]Institute of Computing, University of Campinas, Campinas, Brazil `cid@ic.unicamp.br`

henceforth. A *flip* of a point $p \in P$ w.r.t. a coloring C corresponds to changing its color. The *flip distance* between two colorings C and C' is the number of flips necessary to make them equal and is denoted by $F(C, C')$. The flip distance between $C = (R, B)$ and $C' = (R', B')$ is given by $F(C, C') = R\Delta R' = B\Delta B' = F(C', C)$, where Δ denotes the symmetric difference. Two colorings of P are *opposite* if their flip distance is $n = |P|$. It is easy to see that a quadrangulation of a point set P induces two opposite colorings of P.

Problem Statement. In the Convex Bichromatic Quadrangulation of Point Sets with Minimum Color Flips Problem (CQBPSMF), given a point set P and a coloring C, we want to find a coloring C' of P, if it exists, with minimum flip distance from C such that P admits a bichromatic convex quadrangulation with respect to C'.

Unlike the previously introduced quadrangulation problems, the CQBPSMF is an optimization problem and can have three possible answers. Either P does not admit any convex quadrangulation and the sought after coloring C' of P does not exist; or P admits a bichromatic convex quadrangulation w.r.t. C, i.e., $C' = C$; or C' exists and $F(C, C') > 0$, meaning that P admits convex quadrangulations but no bichromatic convex quadrangulation w.r.t. C. It follows that the CQBPSMF is also $\mathbb{NP}$-hard from the hardness proof of the CQBPS [8].

Our Contributions. In this paper, we introduce a new optimization $\mathbb{NP}$-hard variant of the quadrangulation problem and present two Integer Linear Program models to solve it. We also propose several heuristics that build upon such exact models. These heuristics are compared with the matching approach for other quadrangulation problems from the literature. A comprehensive empirical study assessing the models for both exact and heuristic approaches is reported, as well as comparisons between the heuristics. To allow for future comparisons to our results, the complete benchmark used to appraise the performance of the algorithms is made publicly available.

The work is organized as follows. Section 2 presents an Integer Linear Programming (ILP) model for the CQBPSMF, while Section 3 introduces a set-partition based formulation. Section 4 studies heuristics based on the previously introduced models. An empirical study of the exact and heuristic methods is shown in Section 5, and Section 6 presents our conclusions.

2 Integer Linear Programming Model

In this section, we introduce a formulation (model) for the the CQBPSMF. This is done by starting from a standard set-partition formulation for the quadrangulation problem and adding constraints and variables to assign colors and count flips. We begin by defining important ILP concepts and notations.

A *linear relaxation* of an ILP model is obtained by dropping the integrality requirement of its variables. In the case of binary variables, instead of assuming values in the set {0,1}, relaxed variables assume values in the interval [0,1]. In this text, we will refer to the linear relaxation simply as relaxation.

Relaxations are a major component of the most popular algorithm for solving ILP models, known as *Branch-and-Bound*. In a Branch-and-Bound algorithm, the set of solutions of a model is searched in a rooted tree-like form. At each node, a linear relaxation is solved obtaining an optimal solution x^*. If the relaxation has a worse objective function value than the best known integral solution or is infeasible, nothing is done. If there are any variables assuming a fractional value, one of them, say x_i^*, is selected for branching leading to two subproblems, one with the additional constraints $x_i \leq \lfloor x_i^* \rfloor$ and the other with additional constraints $x_i \geq \lceil x_i^* \rceil$. Starting from a root node consisting of the relaxation of the original model with no additional constraints, the algorithm proceeds until there are no nodes left for processing. Additional heuristics are run at each node with the goal of finding improved integral solutions, which results in fewer nodes having to be explored.

A *set-partition constraint* is of the form $\sum_{x \in \mathcal{X}} x = 1$, where $\mathcal{X}$ is a set of binary variables. A *set-partition formulation* is a formulation comprised exclusively of set-partition constraints.

Modern ILP solvers are equipped with different heuristics, preprocessing techniques, and methods to derive additional constraints that reduce solving times of many formulations, such as the ones investigated here.

Given a set P of n points, let $L(P)$ denote the set of $\Theta(n^2)$ line segments $\overline{jk}$, where j and k are distinct points in P. The *complete (geometric) graph* induced by P is $G(P) = (P, E(P))$, where $E(P) = \{\{j, k\} : \overline{jk} \in L(P)\}$. We refer to a segment $\overline{jk} \in L(P)$ and the corresponding edge $\{j, k\} \in E(P)$ interchangeably. An edge $\{j, k\} \in E(P)$ is said to be *bichromatic with respect to a coloring* $C = (R, B)$ *of* P if its endpoints have different colors, i.e., $1 = |\{j, k\} \cap R| = |\{j, k\} \cap B|$.

Let $\mathcal{A}(P)$ denote the set of bounded faces of the planar arrangement induced by $L(P)$, and $\mathcal{Q}(P)$ denote the set of $O(n^4)$ empty convex quadrangles with endpoints in P. We say that a quadrangle $q \in \mathcal{Q}$ *covers* a face $f \in \mathcal{A}(P)$, denoted by $f \subset q$, if the interior of f is contained in the interior of q.

We associate a binary variable r_i to each point $i \in P$, such that point i is assigned the color red if, and only if, $r_i = 1$. To each edge $e \in E(P)$, we associate a binary variable x_e so that e is an edge of the quadrangulation if, and only if, $x_e = 1$. Finally, to each quadrangle $q \in \mathcal{Q}(P)$, we associate a binary variable u_q that takes value 1 if, and only if, q is part of the quadrangulation.

This allows us to express our Model (1) as follows.

$$\min \sum_{i\in B} r_i + \sum_{i\in R}(1-r_i) \tag{1a}$$
$$\text{s.t. } x_{ij} + r_i + r_j \leq 2 \quad \forall\{i,j\}\in E(P) \tag{1b}$$
$$x_{ij} \leq r_i + r_j \quad \forall\{i,j\}\in E(P) \tag{1c}$$
$$\sum_{q\in\mathcal{Q}(P):f\subseteq q} u_q = 1 \quad \forall f \in \mathcal{A}(P) \tag{1d}$$
$$\sum_{q\in\mathcal{Q}(P):\{i,j\}\in q} u_q = 2x_{ij} \quad \forall\{i,j\}\in E(P)\backslash \mathrm{CH}(P) \tag{1e}$$
$$\sum_{q\in\mathcal{Q}(P):\{i,j\}\in q} u_q = x_{ij} \quad \forall\{i,j\}\in \mathrm{CH}(P) \tag{1f}$$
$$r_i \in \{0,1\} \quad \forall i \in P \tag{1g}$$
$$u_q \in \{0,1\} \quad \forall q \in \mathcal{Q}(P) \tag{1h}$$
$$x_{ij} \in \{0,1\} \quad \forall\{i,j\}\in E(P) \tag{1i}$$

Constraints (1b) and (1c) ensure that an edge can only be selected if its endpoints have different colors. Constraints (1d) enforce the partitioning of CH(P) into empty convex quadrangles. Finally, Constraints (1e) and (1f) links edges and quadrangles.

In [11], we showed that for problems where the convex hull of a point set P has to be partitioned into polygons with endpoints in P, only $O(n^2)$ faces of $\mathcal{A}(P)$ need to be considered (as in Constraints (1d)). This will be discussed in detail in the next section.

Constraints (1d), together with quadrangle variables, define a standard set-partition formulation, with a known strong linear relaxation [7]. A similar formulation was used to solve the Minimum Convex partition Problem [11]. Formulations for the CQPS and CQBPS can be trivially obtained from them.

On the other hand, Constraints (1b) and (1c), together with the edge and color variables, correspond to a formulation for the classical *MaxCut* problem with positive weights [4]. In the MaxCut problem, the goal is to find a bi-partition of the graph (in our case each part corresponds to a color) such that the total weight of the edges between both parts is maximized. This formulation only works for positive weights, as we are allowed to not select an edge even if its endpoints are in different parts. Notably, this MaxCut formulation is known for its weak linear relaxation. We will show in Section 5 that the linear relaxation of Model (1) behaves poorly as the size of instances grows, resulting in the total solving times increasing very quickly.

3 Pure Set-Partition Formulation

In the previous section, we presented a model for the CQBPSMF and discussed that it is not a strong one due to a subset of its constraints that define a weak formulation for the MaxCut problem.

In this section, we introduce a new model that consists entirely of set-partition constraints, which are known to lead to a strong linear relaxation. To this end,

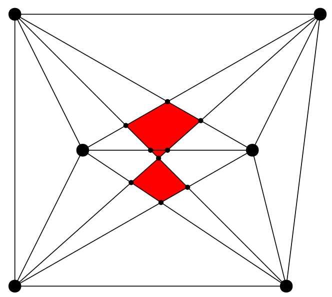


Figure 1: Illustration of the arrangement induced by $L(P)$ for the set P of points indicated by the larger disks. The smaller disks show vertices of the arrangement that are crossings of line segments. Faces of the arrangement that are not i-wedges are colored red.

we introduce colored quadrangles and explore properties of the set partition constraints (1d).

First, notice that each quadrangle in $\mathcal{Q}(P)$ can be colored in only two ways such that its edges are bichromatic. Let us denote the set of all *colored empty convex quadrangles* of P by $\mathcal{Q}_c(P)$.

As proved in [11], to reach a partition of CH(P), we are only required to use Constraints (1d) for faces of $\mathcal{A}(P)$ with at least one vertex in P. The faces with $i \in P$ as one of their vertices are called i-wedges and we say that i is an *anchor* of these i-wedges. Note that faces may have multiple anchors. We denote the set of anchors of an i-wedge f by ANC(f). Finally, let us denote by $\mathcal{W}$ the set of i-wedges for all $i \in P$.

Given an i-wedge $f \in \mathcal{W}$ and an anchor $i \in$ ANC(f), we can split the colored quadrangles that cover f into two sets based on the color assigned to i in their coloration. We denote by R_i^f and B_i^f the sets of colored quadrangles that cover f where point i is colored red and blue, respectively. Figure 1 illustrate an arrangement and its i-wedges.

We write Constraints (1d) for the i-wedges in $\mathcal{W}$ considering the set of colored convex quadrangles as

$$\sum_{q\in\mathcal{Q}_C:f\subset q} u_q = 1, \forall f \in \mathcal{W}.$$

The summation on the left-hand side can be split based on the color assignments to the anchors of f, leading to

$$\sum_{q\in R_i^f} u_q + \sum_{q\in B_i^f} u_q = 1, \forall f \in \mathcal{W}, \forall i \in \mathrm{ANC}(f)$$

However, in any integral solution, there is only one colored polygon covering each face f and, for all anchors $i \in$ ANC(f), it must be from R_i^f when $r_i = 1$, and from B_i^f, otherwise. This allows us to split these constraints, leading to the following model.

$$
\begin{aligned}
\min \quad & \sum_{i \in B} r_i + \sum_{i \in R} (1 - r_i) && && \text{(2a)}\\
\text{s.t.} \quad & \sum_{q \in R_i^f} u_q = r_i, && \forall f \in \mathcal{W}, \forall i \in \text{ANC}(f) && \text{(2b)}\\
& \sum_{q \in B_i^f} u_q = 1 - r_i, && \forall f \in \mathcal{W}, \forall i \in \text{ANC}(f) && \text{(2c)}\\
& r_i \in \{0,1\} && \forall i \in P && \text{(2d)}\\
& u_q \in \{0,1\} && \forall q \in \mathcal{Q}(P) && \text{(2e)}
\end{aligned}
$$

For better visualization of the model as a pure set-partition formulation and to better interpret its constraints, an equivalent model can be obtained by adding binary variables b_i for all $i \in P$ such that $b_i = 1 - r_i$, allowing us to rewrite the model as follows.

$$
\begin{aligned}
\min \quad & \sum_{i \in B} r_i + \sum_{i \in R} b_i && && \text{(3a)}\\
\text{s.t.} \quad & r_i + b_i = 1 && \forall i \in P && \text{(3b)}\\
& \sum_{q \in R_i^f} u_q + b_i = 1, && \forall f \in W, \forall i \in \text{ANC}(f) && \text{(3c)}\\
& \sum_{q \in B_i^f} u_q + r_i = 1, && \forall f \in W, \forall i \in \text{ANC}(f) && \text{(3d)}\\
& r_i \in \{0,1\} && \forall i \in P && \text{(3e)}\\
& b_i \in \{0,1\} && \forall i \in P && \text{(3f)}\\
& u_q \in \{0,1\} && \forall q \in \mathcal{Q}(P) && \text{(3g)}
\end{aligned}
$$

Constraints (3c) ensure that either point i is assigned the color blue or there is exactly one colored quadrangle covering the i-wedge f where i is colored red. A similar interpretation can be given to (3d) by exchanging the colors. Constraints (3b) where added to ensure that a single color is assigned to each point.

This model performs significantly better in practice than the one introduced in the previous section, as we will show in Section 5.

4 Linear Relaxation Based Heuristics

In this section, we propose a heuristic framework for the CQBPSMF, based on the strength of the Model (2).

Our heuristics work in three steps. First, we solve the relaxation of Model (2), obtaining a solution u^*. Based on u^*, a set of colored quadrangles $S_c \subset \mathcal{Q}_c$ is constructed. Finally, we solve Model (2) restricted to the colored quadrangles in S_c. Notice that if the relaxation is infeasible, so is the restricted problem.

This is motivated by the global view that an linear programming relaxation provides, as opposed to merely local properties provided by, for instance, projected Delaunay Tetrahedralizations [13]. Thus, even when fractional, the values assigned to variables are still within the constraints of the model, and might give a good guidance for heuristic decisions.

An advantage of relaxation-based heuristics is the possibility of integrating them with a Branch-and-Bound algorithm. This is possible since a solution to the linear relaxation is always available. In particular, modern solvers are capable of finding good set-partition solutions using their own internal heuristics. However, if new constraints and variables are added with the goal of strengthening the model, the solver might face difficulties in finding good solutions. With this in mind, even if the relaxation takes a considerable amount time to be solved, having a fast relaxation-based algorithm can significantly speedup the solution to an ILP model.

The choice of colored quadrangles in S_c plays an important role for the success of the heuristic. In particular, a poor choice of quadrangles might lead to an infeasible model. Conversely, if too many quadrangles are included, the problem may become as hard as the unrestricted version. We propose three different ways to construct S_c: one is based on quadrangles, one relies on edges and another is grounded on triangulations.

4.1 Quadrangulation-based

A trivial way to construct S_c, given a solution u^* to its relaxation, is to include all quadrangles whose variables have positive value in the relaxation, i.e., $S_c = \{q : u_q^* > 0\}$. However, this choice fails to lead to feasible solutions for many instances.

This approach works best when there are ways to ensure feasibility by adding a specific set of variables to the ILP model, or when good procedures to repair feasibility are known.

4.2 Edge-based

As an attempt to increase the number of feasible solutions found, one might expand the quadrangles selected. In our case, we can explore a neighborhood of the polygons whose values are positive in the relaxation. One way to achieve this is to project the values of polygon variables onto edges, similarly to how it is done in Model (1).

For Model (2), the projected value of an edge e can be obtained, based on Constraint (1f) and (1e), as $\sum_{u_t : e \in t} \frac{u_q}{2}$. This encourages us to include all polygons whose edges have positive projected values, i.e., $S_c = \{q : \sum_{u_t : e \in t} u_q > 0, \forall e \in q\}$. Indeed, this approach results in a higher number of feasible solutions being found. However, the growth in the number of quadrangles substantially increases solving times, leading to a heuristic that does not scale well with the size of the set of points.

4.3 Triangulation-based

Lastly, we propose a third approach based on triangulations that tries to combine the benefits of the previous heuristics. This insight comes from the fact that most of the heuristics for the CQPS are based on triangulations.

We begin with a greedy triangulation based on the value of the relaxation projected onto the edges and find quadrangles supported by those edges. Actually, when starting from a triangulation, the ILP model is not even necessary for solving the resulting restricted model, as it becomes equivalent to finding a matching of adjacent triangles, a standard approach used in the literature for quadrangulation problems [3, 9].

Let $x_e = \sum_{u_t : e \in t} u_q^*$ denote the projected value of edge $e \in E(P)$. A greedy triangulation can be constructed by sorting the edges of $E(P)$ by their projected values in descending order and attempting to add them, in that order, to an initially empty partial triangulation. An edge is successfully added if it does not cross any previously inserted edge, otherwise, it is rejected. The result is a greedy triangulation.

An edge of a triangulation admits an *edge-flip* if the two triangles adjacent to it can be merged to form a convex quadrangle – one of whose diagonals is the given edge. Its edge-flip is the other diagonal of this convex quadrangle.

If T denotes the set of all edges of a given (say, greedy) triangulation $\mathcal{T}$ and F denotes the set of edge-flips of T, we can identify the set of the aforementioned quadrangles as $S_c = \{q : e \in T \cup F, \forall e \in q\}$.

4.4 Matchings and Edge-Flips

In the literature on the CQPS, to the best of our knowledge, there are no prior attempts at edge-flips. Most of the work based on triangle matching is either focused on specific triangulations, such as Delaunay and Serpentine [2], or includes randomized components. The larger angles on a Delaunay triangulation lead to more pairs of triangles that can be matched. The Serpentine quadrangulation, in turn, admits a perfect match if convexity is not required, provided that the convex hull of the point set has an even number of points.

We recall that a matching takes place on a subgraph $H = (\Delta, D)$ of the dual graph of a triangulation $\mathcal{T}$, where each vertex in Δ corresponds to a triangle, and the edges[1] in D are those that connect pairs of vertices that represent two triangles that share a side thereby forming a convex quadrangle. A partial quadrangulation can be constructed by finding a matching M on the graph H. However, this approach usually leaves unmatched vertices of Δ, i.e., triangles of $\mathcal{T}$.

[1]The term *edge* is used in this text to refer to line segments between points, in a geometric context, and as arcs between vertices, in the context of a graph.

Geometrically speaking, to exchange an edge e of $\mathcal{T}$ for its edge-flip $e^{\perp}$ can be seen as an operation that locally modifies $\mathcal{T}$ creating a new triangulation $\mathcal{T}'$. Let H' be the subgraph of the dual graph of $\mathcal{T}'$ constructed in a fashion analogous to the description of the previous paragraph. On H', most of the matching M is preserved. However, by searching for augmenting paths for M on H', we may still be able to further reduce the number of unmatched triangles. This can be thought of as a local search procedure.

In this context, solving an ILP restricted to the quadrangles obtained from both the edges of a triangulation and their edge-flips is an attempt at globally navigating a neighborhood (by edge-flips) of a given triangulation.

As we show in the next section, the resulting ILP can easily be solved by modern solvers and this approach can straightforwardly be adapted for the CQPS.

5 Experiments

In this section, we compare the solving time and strength of the relaxation of our two exact models. We also show how our heuristics behave in practice.

To do this, we generated a benchmark comprised of two sets of instances, ranging in size from 40 to 120 points, in steps of 10. We were able to determine quite fast which point sets were not quadrangulable and collected them in the first set of instances. More on this, below. The feasible ones were gathered in the second set, since these are, obviously, the challenging instances. In both cases, 20 point sets of each size are generated by sampling points uniformly on the $[0, 1] \times [0, 1]$ square. Next, each point set is assigned three random colorings, also chosen uniformly. This results in 60 instances per size in each set of instances. All these instances and their respective solutions are available at [12].

All experiments were run on a computer featuring an Intel Xeon Silver 4114 at 2.2Ghz CPU with 10 cores, and 32GB of RAM, running Ubuntu 16.04. Models and algorithms were implemented in C++ v.11 and compiled with gcc 5.5. Geometric algorithms and data structures were implemented using CGAL 5.2 [14], and Gmpq for exact number representation. The ILP models and relaxations were coded using CPLEX 12.10 with the default parameters. A time limit of 30 minutes was set for each CPLEX run.

Throughout this section, data will be presented mostly as standard boxplots[2]. Averages will be expressed in the form avg ± std, where avg is the average and std is the standard deviation. All data used to generate the plots are accessible at [12].

In the first experiment, we compared Model (1), which we call NATURAL Model, and Model (2), referred to as SET-PARTITION Model.

[2]See: `https://en.wikipedia.org/wiki/Box_plot`.

Firstly, deciding that a point set with at most 120 points does not admit a quadrangulation was accomplished within 20 seconds using either model since the solver was able to prove infeasibility during preprocessing. This turned out to be an efficient way to separate our two sets of instances. As this is quite swift compared to optimizing (color) flips in a feasible solution, the remaining of this section will be restricted to the set of feasible instances.

To shed light on the ILP models being solved and on the intricacies our benchmark, Table 1 shows the average number of empty convex quadrangles, number of i-wedges and faces in the arrangement. For the sake of simplifying the implementation, i-wedges were considered with multiplicities (for each anchor), leading to exactly $n(n-1)$ i-wedges, which CPLEX can easily remove as duplicated rows. We confirmed that restricting our model to the i-wedges reduced the number of constraints drastically.

We say that the optimal value of a relaxation is *tight* if its ceiling (in the case of a minimization problem) is equal to the optimal integral value. Table 2 shows how many instances each model was capable to solve per size within our specified time limit. No number is shown when at least 30% of the instances failed to be solved. The last two rows show for how many instances the relaxation produced a tight bound. Figures 2 and 3 show, for the instances that *were* solved, the total solving time, grouped by instance size for the NATURAL and SET-PARTITION models, respectively. We can see that the SET-PARTITION model was much faster. While the NATURAL model began to founder for instances of just 70 points, the SET-PARTITION model only failed to solve instances of 110 or more points. As discussed in Section 3, this difference in performance is mostly attributed to the stronger relaxation of the SET-PARTITION Model.

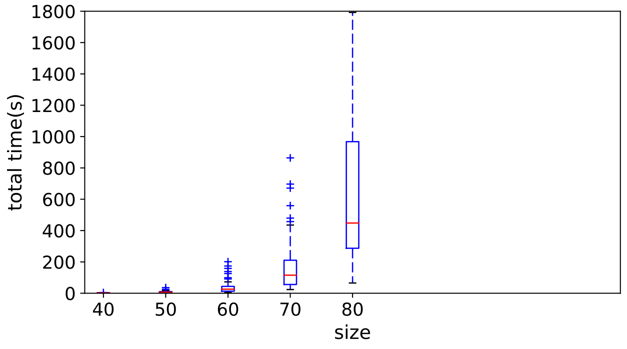


Figure 2: Solving time per instance size for the NATURAL Model.

If we denote the value of an optimal solution for an instance by OPT and the value of an optimal solution

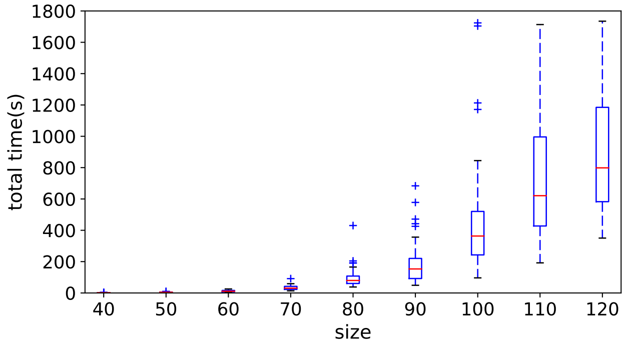


Figure 3: Solving time per instance size for the SET-PARTITION Model.

of the linear relaxation of a given model for the same instance by RLX, the *relative duality gap* can be defined as $\frac{\text{OPT}-\text{RLX}}{\text{RLX}}$. The relative gap for the NATURAL and SET-PARTITION models in shown in Figures 4 and 5, respectively. As can be seen, while the relative duality gap of the SET-PARTITION model shows very little variation as the size of instances increases, staying within 0.2, the relative duality gap of the NATURAL model surpasses 0.4 for much smaller instances. The number of tight relaxations shown in Table 2 corroborates with the observation that the SET-PARTITION model provides a much stronger relaxation.

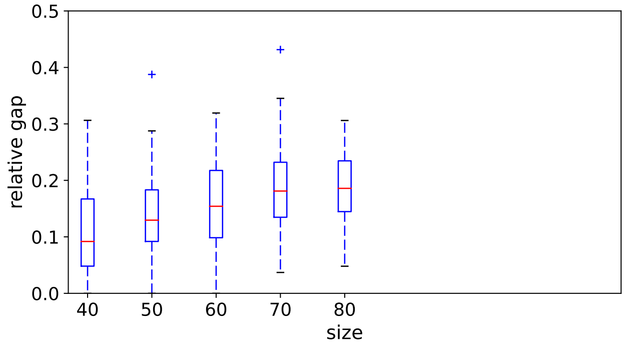


Figure 4: Relative duality gap per instance size for the NATURAL Model.

The quality of the relaxation is a key element when solving ILP models. Poor relaxations result in a larger number of nodes being explored in the search tree and more difficulty in finding good integral solutions.

The *support* of a solution of a linear program is the set of variables assigned positive values. Next, we discuss the support of the optimal solutions for both the NATURAL and the SET-PARTITION model.

Tables 3 and 4 show the average distribution of (projected) edge values in the optimal solutions found for the NATURAL and the SET-PARTITION models, as well

Table 1: Analysis of the instances in the quadrangulable benchmark, averaged for each instance size, where '# quads' indicate the number of empty convex quadrangles, '# i-wedges' the number of i-wedges and '# faces' the number of faces of the arrangement.

	40	50	60	70	80	90	100	110	120
# quads	3455 ± 323	5659 ± 474	8671 ± 731	12329 ± 1023	16790 ± 980	21546 ± 640	27493 ± 1303	33675 ± 1127	41359 ± 1284
# i-wedges	1560	2450	3540	4830	6320	8010	9900	11990	14280
# faces	64381 ± 1809	161467 ± 5184	338360 ± 11217	636033 ± 13104	1097777 ± 25193	1760375 ± 41057	2717477 ± 48764	4006901 ± 58169	5726520 ± 86319

Table 2: Number of solved instances and of tight relaxations for the Natural and Set-partition Model per size.

	40	50	60	70	80	90	100	110	120
# solved Natural	60	60	60	57	39	-	-	-	-
# solved Set-partition	60	60	60	60	60	60	60	51	32
# tight Natural	32	14	9	9	3	-	-	-	-
# tight Set-partition	54	58	47	52	44	41	35	24	26

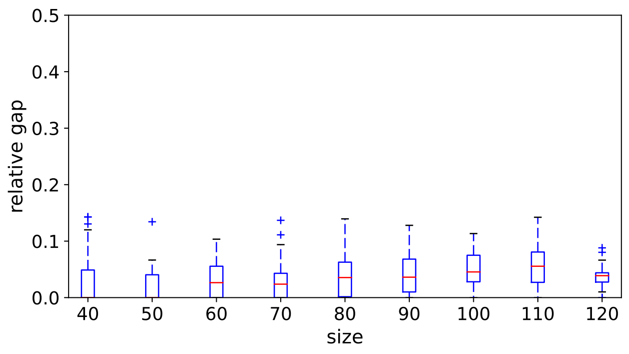


Figure 5: Relative duality gap per instance size for the Set-partition Model.

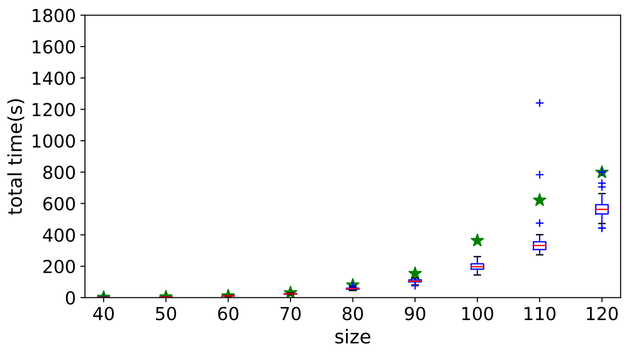


Figure 6: Solving time per instance size for the relaxation of the Set-partition Model. Green stars indicate the median solving time for the exact model.

as the size of the corresponding support. For the purpose of deriving good heuristics, solutions of the relaxation must have a large number of variables with high values (in the $[0, 1]$ range), as they are more likely to appear in an integral solution. In this case, while most of the edge variables of the Natural model have value less than 0.1, the projected edge value of the Set-partition model presented a much better distribution and are much more informative. As a consequence, we confine the analysis of our heuristic to the relaxation of the Set-partition model.

Even though the quality of the relaxation of the Set-partition model is significantly higher, it suffers from one of the major drawback of set-partition models: slow convergence. This is exacerbated as the model grows with instance size.

Total solving time for the relaxation of the Set-partition Model is shown in Figure 6. Comparing the solving time of the relaxation with the median solving time of the exact model, we can see that solving the linear relaxation is a significant bottleneck.

Lastly, we show how the proposed heuristics perform on the benchmark. As small instances were too easy to solve, the experiments with the heuristics are limited to instances with at least 70 points.

We denote our quadrangle-based heuristic *quad*, our edge-based heuristic *edge* and our heuristic based on the greedy triangulation *greedytri*. We also include the Delaunay triangulation as an alternative to the greedy triangulation, denoted by *Delaunay*, for a baseline comparison. For each heuristic that relies on triangulations, we append the suffix *-flips* to indicate an alternative when edge-flips were added.

Table 5 shows the average number of quadrangles selected by each heuristic. As can be seen, all heuristics use only a small fraction of the total number of quadrangles. Notably, for small instances, where the relaxation performed well, the edge-based heuristic starts with a small number of quadrangles, but it rapidly increases. As expected, the Delaunay triangulation admits a larger number of empty convex quadrangles when compared to the greedy triangulation.

Table 6 shows how many feasible solutions each heuristic found. The quad heuristic only found solutions where the relaxation itself had already produced an integral one. Both triangulation-based heuristics had a hard time finding quadrangulations without edge-flips.

Table 3: Description of the support of optimal linear relaxations found by the Natural Model as a function of instance size. The first ten lines indicate the percentage of edge values within the given interval. The last two lines show the number of edge variables in the support and the total number of edges.

	40	50	60	70	80
% in (0.0,0.1]	30 ± 12	40 ± 13	46 ± 12	57 ± 6	60 ± 6
% in (0.1,0.2]	16 ± 8	17 ± 7	17 ± 6	16 ± 5	16 ± 5
% in (0.2,0.3]	10 ± 5	10 ± 5	9 ± 3	7 ± 3	7 ± 2
% in (0.3,0.4]	7 ± 4	6 ± 3	6 ± 2	5 ± 2	4 ± 1
% in (0.4,0.5]	5 ± 4	4 ± 2	4 ± 3	3 ± 1	3 ± 1
% in (0.5,0.6]	3 ± 2	3 ± 2	3 ± 1	2 ± 1	2 ± 1
% in (0.6,0.7]	3 ± 2	3 ± 1	2 ± 1	2 ± 1	2 ± 1
% in (0.7,0.8]	2 ± 2	2 ± 2	2 ± 1	2 ± 1	1 ± 1
% in (0.8,0.9]	2 ± 2	2 ± 2	2 ± 1	2 ± 1	1 ± 1
% in (0.9,1.0]	21 ± 27	13 ± 19	9 ± 13	5 ± 3	4 ± 2
Support size	215 ± 55	342 ± 74	482 ± 96	699 ± 78	913 ± 107
# edges	780	1225	1770	2415	3160

Table 4: Description of the support of optimal linear relaxations found by the Set-partition Model as a function of instance size. The first ten lines indicate the percentage of projected edge values within the given interval. The last two lines show the number of projected edges variables in the support and the total number of edges.

	40	50	60	70	80	90	100	110	120
% in (0.0,0.1]	1 ± 3	3 ± 7	4 ± 9	8 ± 12	14 ± 15	14 ± 15	23 ± 17	27 ± 16	32 ± 15
% in (0.1,0.2]	3 ± 8	7 ± 12	7 ± 10	9 ± 12	10 ± 9	13 ± 12	16 ± 9	15 ± 9	17 ± 7
% in (0.2,0.3]	4 ± 10	7 ± 12	6 ± 10	5 ± 7	9 ± 9	9 ± 9	9 ± 6	11 ± 7	11 ± 5
% in (0.3,0.4]	4 ± 10	7 ± 12	9 ± 15	7 ± 10	9 ± 12	8 ± 10	7 ± 6	10 ± 9	7 ± 3
% in (0.4,0.5]	17 ± 24	8 ± 14	13 ± 19	11 ± 15	11 ± 14	9 ± 12	10 ± 11	9 ± 11	7 ± 9
% in (0.5,0.6]	1 ± 4	2 ± 4	2 ± 4	3 ± 4	3 ± 4	3 ± 3	4 ± 3	4 ± 2	3 ± 2
% in (0.6,0.7]	2 ± 6	4 ± 8	4 ± 7	3 ± 6	4 ± 6	4 ± 5	4 ± 3	4 ± 4	4 ± 2
% in (0.7,0.8]	2 ± 4	3 ± 6	2 ± 3	2 ± 3	2 ± 3	3 ± 4	3 ± 3	3 ± 2	4 ± 3
% in (0.8,0.9]	0 ± 1	1 ± 1	1 ± 2	2 ± 3	1 ± 2	2 ± 3	3 ± 3	3 ± 2	3 ± 2
% in (0.9,1.0]	67 ± 35	57 ± 37	52 ± 37	49 ± 36	36 ± 34	35 ± 35	22 ± 22	16 ± 18	13 ± 11
Support size	96 ± 31	142 ± 54	185 ± 74	234 ± 101	330 ± 138	380 ± 161	513 ± 182	628 ± 197	757 ± 218
# edges	780	1225	1770	2415	3160	4005	4950	5995	7140

Table 5: Number of quadrangles in each heuristic, and the number of quadrangles in an unrestricted model per instance size.

	70	80	90	100	110	120
Delaunay	137 ± 4	158 ± 5	182 ± 5	202 ± 6	227 ± 7	247 ± 6
Delaunay-flips	538 ± 19	628 ± 16	719 ± 15	815 ± 23	919 ± 21	1000 ± 23
quad	163 ± 106	252 ± 148	293 ± 173	423 ± 203	499 ± 208	637 ± 243
greedytri	113 ± 10	135 ± 12	154 ± 14	177 ± 13	197 ± 12	217 ± 12
greedytri-flips	376 ± 40	448 ± 48	519 ± 52	601 ± 52	676 ± 50	743 ± 53
edges	293 ± 288	502 ± 419	597 ± 514	951 ± 647	1122 ± 687	1565 ± 889
unrestricted	12329 ± 1023	16790 ± 980	21546 ± 640	27493 ± 1303	33675 ± 1127	41359 ± 1284

The greedytri heuristic was the only one competitive with the edge-based one.

The solving time for each heuristic is shown in Table 7. We can see that the time just to solve the relaxation is more than 200 times larger than the specific solving time of each of the heuristics.

One way to take advantage of this discrepancy in solving times is to add randomization to the heuristics. For instance, one might consider adding perturbations to the weights before running the greedy triangulation. Another approach is to run the heuristic before the relaxation is solved optimally. With the exception of the edge-based one, all other heuristics performed fast enough to be executed multiple times during the solution of the relaxation of a node in a Branch-and-Bound algorithm. Alternative methods to solve the linear re-

Table 6: Number of feasible solutions found by each heuristic (out of 60).

	70	80	90	100	110	120
Delaunay	0	0	0	0	0	0
Delaunay-flips	3	3	0	0	2	0
quad	17	10	10	1	2	1
greedytri	19	10	11	1	2	0
greedytri-flips	51	44	51	48	42	37
edges	**52**	**55**	**52**	**53**	**46**	**48**

Table 7: Solving time (in seconds) for each heuristic by instance size. For the ones based on the relaxation, (+) indicates that the average time spend solving the relaxation is shown separately.

	70	80	90	100	110	120
Delaunay	0.07 ± 0.00	0.09 ± 0.00	0.10 ± 0.00	0.12 ± 0.00	0.14 ± 0.01	0.16 ± 0.01
Delaunay-flips	0.11 ± 0.04	0.13 ± 0.04	0.15 ± 0.01	0.18 ± 0.01	0.22 ± 0.04	0.25 ± 0.01
quad (+)	0.08 ± 0.04	0.13 ± 0.05	0.15 ± 0.06	0.22 ± 0.08	0.27 ± 0.09	0.37 ± 0.14
greedytri (+)	0.08 ± 0.01	0.09 ± 0.01	0.10 ± 0.01	0.12 ± 0.01	0.14 ± 0.01	0.17 ± 0.01
greedytri-flips (+)	0.17 ± 0.04	0.20 ± 0.05	0.25 ± 0.05	0.31 ± 0.08	0.35 ± 0.10	0.39 ± 0.11
edges (+)	0.25 ± 0.35	0.51 ± 0.61	0.63 ± 0.78	1.31 ± 1.62	1.46 ± 1.52	3.12 ± 3.80
(+) relax time	24.02 ± 2.36	57.35 ± 5.61	105.90 ± 9.53	197.42 ± 24.73	363.48 ± 140.25	610.19 ± 128.95

Table 8: Relative primal gap for each heuristic per instance size. Heuristics that did not find solutions when the relaxation was not integral are indicated by *.

	70	80	90	100	110	120
Delaunay	-	-	-	-	-	-
Delaunay-flips	0.74 ± 0.17	0.57 ± 0.10	-	-	-	-
quad	*	*	*	*	*	*
greedytri	*	*	0.03 ± 0.11	*	*	-
greedytri-flips	0.12 ± 0.15	0.21 ± 0.23	0.22 ± 0.24	0.26 ± 0.17	0.33 ± 0.21	0.30 ± 0.20
edges	0.02 ± 0.06	0.02 ± 0.04	0.01 ± 0.03	0.02 ± 0.07	0.03 ± 0.07	0.01 ± 0.02

laxation approximately are also worth of consideration.

Let HEU be the number of (color) flips in a solution computed by a heuristic for a given instance and recall that OPT denotes the optimal number of flips for that instance. The *relative primal gap* is given by $\frac{\text{HEU}-\text{OPT}}{\text{OPT}}$, similarly to the relative duality gap.

Table 8 shows the relative primal gap for all the heuristics. Even though the heuristic based on the greedy triangulation with edge-flips proved competitive, yielding non-trivial solutions for *all* instances, it is clear that the edge-based one was superior in finding solutions of the highest quality.

6 Concluding Remarks

In this paper, we introduced and studied a new optimization variant of the quadrangulation problem. This study focused on providing exact ILP models and heuristics derived from their linear relaxations. Both approaches provide insight into two quadrangulation decision problems from the literature.

We proposed two ILP models to solve this novel problem: one that arises naturally from a model for the more general convex partitioning problem and another that is obtained from formulating the problem as a set-partition problem.

While the natural model failed to solve instances with 70 points within 30 minutes of computing time, the set-partition model was successful in solving all instances of up to 100 points. This is due to a much stronger linear relaxation obtained from the set-partition formulation.

We also proposed several heuristics based on the linear relaxation of the set-partition model. These heuristics use a solution to the linear relaxation of the set-partition model as a guide to select only a subset of all quadrangles and, subsequently, solve much smaller instances of the set-partition model restricted to them.

Among these heuristics, our edge-based one shows superior performance in finding many good quality feasible solutions. On average, it solved most of the benchmark instances and arrived at solutions at most 3% worse than the optimal. Therefore, it is an excellent candidate to be included in a Branch-and-Bound algorithm, to be run at the end of the computation in each node.

The heuristic that is based on the greedy triangulation with edge-flips found a large number of feasible solutions for all sizes, and was notably faster than the edge-based one. As a result, it might be useful for successive executions, say, within a randomization scheme.

Overall, the quality of the relaxation-based heuristic was also quite good. Finding faster ways of solving or even approximating the linear relaxation is clearly a good venue for future research related to heuristics.

Another topic of investigation that can benefit both the heuristics and the exact algorithm is to look for preprocessing routines capable of fixing variables a priori, leading to a reduced model size.

References

[1] A. Biniaz, A. Maheshwari, and M. H. M. Smid. Compatible 4-holes in point sets. In S. Durocher and S. Kamali, editors, *Proceedings of the 30th Canadian Conference on Computational Geometry, CCCG, August 8-10, 2018, University of Manitoba, Winnipeg, Canada*, pages 346–352, 2018.

[2] P. Bose, S. Ramaswami, G. T. Toussaint, and A. Turki. Experimental results on quadrangulations of sets of fixed points. *Computer Aided Geometric Design*, 19(7):533–552, 2002.

[3] P. Bose and G. T. Toussaint. No quadrangulation is extremely odd. In J. Staples, P. Eades, N. Katoh, and A. Moffat, editors, *Proceedings of the 6th International Symposium on Algorithms and Computation, ISAAC, Cairns, Australia, December 4-6, 1995*, volume 1004 of *LNCS*, pages 372–381. Springer, 1995.

[4] W. F. de la Vega and C. Kenyon-Mathieu. Linear programming relaxations of Maxcut. In N. Bansal, K. Pruhs, and C. Stein, editors, *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, New Orleans, USA, January 7-9, 2007*, pages 53–61. SIAM, 2007.

[5] T. Fevens, H. Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1):95–107, 2001.

[6] V. M. Heredia and J. Urrutia. On convex quadrangulations of point sets on the plane. In J. A. et al., editor, *Discrete Geometry, Combinatorics and Graph Theory, 7th China-Japan Conference, Tianjin, China, Nov 18-20, 2005*, volume 4381 of *LNCS*, pages 38–46. Springer, 2005.

[7] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, June 1993.

[8] A. Pilz and C. Seara. Convex quadrangulations of bichromatic point sets. *International Journal on Computational Geometry and Applications*, 29(4):289–299, 2019.

[9] S. Ramaswami, P. A. Ramos, and G. T. Toussaint. Converting triangulations to quadrangulations. *Computational Geometry*, 9(4):257–276, 1998.

[10] S. Ramaswami, M. Siqueira, T. A. Sundaram, J. H. Gallier, and J. C. Gee. Constrained quadrilateral meshes of bounded size. *International Journal on Computational Geometry and Applications*, 15(1):55–98, 2005.

[11] A. Sapucaia, P. J. de Rezende, and C. C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, 99:101794, 2021.

[12] A. Sapucaia, C. C. de Souza, and P. J. de Rezende. Convex bichromatic quadrangulation of point sets: Benchmark instances. www.ic.unicamp.br/∼cid/Problem-instances/BichromaticPartition, 2021.

[13] T. Schiffer, F. Aurenhammer, and M. Demuth. Computing convex quadrangulations. *Discrete and Applied Mathematics*, 160(4-5):648–656, 2012.

[14] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2 edition, 2020.

An Improved Kernel for the Flip Distance Problem on Simple Convex Polygons

Miguel Bosch Calvo Steven Kelk*

Abstract

The complexity of computing the flip distance between two triangulations of a simple convex polygon is unknown. Here we approach the problem from a parameterized complexity perspective and improve upon the $2k$ kernel of Lucas [10]. Specifically, we describe a kernel of size $\frac{4k}{3}$ and then show how it can be improved to $(1+\epsilon)k$ for every constant $\epsilon > 0$. By ensuring that the kernel consists of a single instance our result yields a kernel of the same magnitude (up to additive terms) for the almost equivalent rotation distance problem on rooted, ordered binary trees. The earlier work of Lucas left the kernel as a disjoint set of instances, potentially allowing very minor differences in the definition of the size of instances to accumulate, causing a constant-factor distortion in the kernel size when switching between flip distance and rotation distance formulations. Our approach avoids this sensitivity.

1 Introduction

Triangulating a set of points on a plane is a common operation in computational geometry. The operation of *flipping a diagonal* is defined as removing one edge of a triangulation, creating a convex quadrangle, and then adding to the triangulation the opposing diagonal of that quadrangle, as seen in Figure 1.

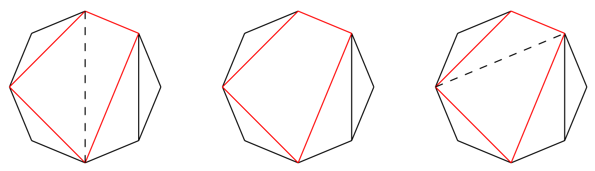


Figure 1: Flipping a diagonal of a triangulation of a simple convex polygon.

The *flip distance* between two triangulations of the same set of points on a plane is the minimum number of flips needed to transform one triangulation into another. Computing flip distance is NP-hard, even for the case of simple polygons [1]. In this article we will be working in a more restricted setup by considering only triangulations of simple *convex* polygons. The complexity of the problem is unknown. Indeed, there is a well known correspondence - essentially, an equivalence - between this problem and the computation of rotation distance between two rooted, ordered binary trees. It has been an open question for several decades whether rotation distance is polynomial-time solveable. Some of the results in this area have been obtained using the rotation distance formulation, but most of the work has been undertaken in the flip distance formulation.

Here we adopt a parameterized complexity perspective; in particular, a kernelization perspective [6]. Cleary et al. [3] proved that the problem is fixed parameter tractable, by providing a $5k$ kernel, where the parameter k is the flip distance. Lucas [10] employed different reduction strategies to obtain a kernel of size $2k$. In this article we will show how to improve upon the kernelization result of Lucas. We describe a $\frac{4k}{3}$ kernel and then extend the approach to yield a $(1+\epsilon)k$ kernel for every constant $\epsilon > 0$; the running time grows sharply in $1/\epsilon$ but remains polynomial for fixed ϵ. Our article extends the decomposition-based approach of Lucas in two ways. We strengthen the bound on the size of the kernel, and potentially lower the flip distance, by solving small decomposed instances to optimality. Secondly, we show how to "reverse" the decomposition strategy adopted by Lucas, thus merging the separate instances into a single reduced instance at the end. This merging step ensures that the size of the kernel remains (up to additive terms) unchanged whether we view the problem from the flip distance or rotation distance perspective. As we note in the Discussion section, this is not as straightforward for Lucas' kernel result: there a subtle constant-factor distortion occurs when switching from one formulation to the other.

2 Preliminaries

We are working here with simple convex polygons. Such a polygon can be viewed without loss of generality as a simple cycle on n edges and n vertices. A triangulation of a simple convex polygon on n edges contains exactly $n-3$ diagonals. Hence a triangulation of a simple convex polygon can be represented as a list of $n-3$ edges and

*Both authors are at the Department of Data Science and Engineering, Maastricht University, The Netherlands. Email: `miguel.bosch@student.maastrichtuniversity.nl`, `steven.kelk@maastrichtuniversity.nl` two triangulations are considered equal if the $n-3$ edges (i.e. the diagonals) are identical. Thus, there is a finite number of triangulations of simple convex polygons of a given size. Precisely, the number of triangulations of a simple convex polygon of size n is given by the $(n-2)$th Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$.

We define $\mathcal{P}_n$ as the set containing all triangulations of simple convex polygons of size n. Thus $|\mathcal{P}_n| = C_{n-2}$. We say that $(P, P') \in \mathcal{P}_n$ if both P and P' are triangulations of simple convex polygons of size n.

Henceforth, for the sake of brevity, we will refer to triangulations of simple convex polygons as triangulations or simply polygons.

Given $(P, P') \in \mathcal{P}_n$, we call a *shortest path* from P to P' to the sequence of polygons $P = P_0, P_1, P_2, \ldots, P_m = P'$ such that we can transform P_i into P_{i+1} by just flipping one diagonal and m is the minimum among all possible sequences. Given a pair of polygons $(P, P') \in \mathcal{P}_n$, the *flip distance* $d(P, P')$ between P and P' is the length of a shortest path from P to P'.

One of the earliest results in this area is the upper bound on flip distance proved by Culik et al. [4]. Precisely, the flip distance between two polygons $(P, P') \in \mathcal{P}_n$ is at most $2n-6$ for all $(P, P') \in \mathcal{P}_n$. Later, Sleator et al. [12] improved the bound to $2n-10$ for all $(P, P') \in \mathcal{P}_n, n > 12$, and by making use of hyperbolic geometry proved that the bound is tight.

Also, since every flip of a diagonal only affects one diagonal, the flip distance between (P, P') is at least the number of non-common diagonals of (P, P') [10].

There is another result from Sleator et al. [12] that is of importance to us. It implies that common diagonals belong to every polygon of every shortest path, and therefore that they should not be flipped at any point: Given $(P, P') \in \mathcal{P}_n$, if there is a common diagonal between P and P', then every shortest path from P to P' does not flip that diagonal.

We now present a formal definition of the PARAMETERIZED FLIP DISTANCE problem, which is the problem we will be addressing in this article:

PARAMETERIZED FLIP DISTANCE
Input: A pair of polygons $(P, P') \in \mathcal{P}_n$ and a parameter $k \in \mathbb{N}$.
Question: Is the flip distance between P and P' at most k?

As is standard in the study of kernelization, we will apply polynomial-time reduction rules to yield instances whose size is bounded by a function purely of k. We omit a formal definition of kernelization, referring to standard texts such as [6] for more details. We emphasize that the size of an instance, n, refers to the number of outer edges in the polygons.

The kernel we propose uses some of the ideas presented by Lucas at [10] combined with new reduction rules to tighten the bound on the kernel, plus a new merging step. Lucas' idea is based on dividing the original pair of polygons along their common diagonals by using the results by Sleator et al. [12]. An example of such division is shown in Figure 2.

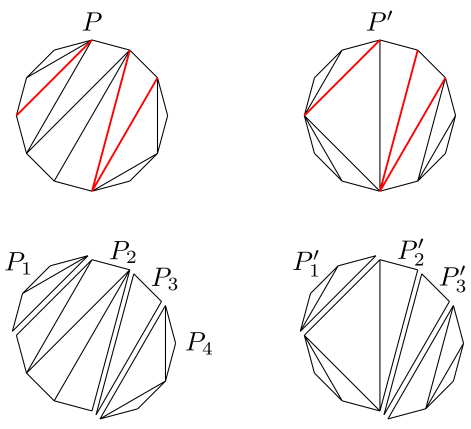


Figure 2: An example of splitting a polygon pair (P, P') along its common diagonals into m disjoint pairs. Here the instances (P, P') have size 12, so $(P, P') \in \mathcal{P}_{12}$, and they have 3 common diagonals, so they are divided into $m = 4$ disjoint pairs $(P_1, P'_1), (P_2, P'_2), (P_3, P'_3), (P_4, P'_4)$.

We will first present the operations that allows us to obtain a $\frac{4k}{3}$ kernel and then we will extend those ideas to derive the $(1+\epsilon)k$ kernel.

3 Results

3.1 $\frac{4k}{3}$ kernel

Lucas [10], using the results of Sleator et al. [12], showed that given two polygons $(P, P') \in \mathcal{P}_n$ with $m-1$ common diagonals, we can create m disjoint pairs of polygons $(P_i, P'_i), i \in [1, m]$ by dividing the original polygons along their common diagonals, so each common diagonal becomes an outer edge of one of the instances (P_i, P'_i) and each pair does not have any common diagonal. Thus we derive the following lemma:

Lemma 1 *The flip distance of $(P, P') \in \mathcal{P}_n$ is equal to the sum of the distances between all m pairs (P_i, P'_i) resulting from the division of (P, P') along its $m-1$ common diagonals, i.e. $d(P, P') = \sum_{i=1}^{m} d(P_i, P'_i)$.*

It is useful to apply the division along common diagonals into m pairs to the parameterized version of the problem, given by $(P, P') \in \mathcal{P}_n, k \in \mathbb{N}$.

Given a set of m pairs of polygons $(P_i, P'_i) \in \mathcal{P}_{n_i}$, let d_i be the number of diagonals of instance i, so that $d_i = n_i - 3$.

The upper bound of the problem of roughly $2n$ can be applied to every pair, and the pairs do not have any common diagonal, so we can deduce $d_i \leq d(P_i, P'_i) \leq 2d_i$.

Since $d(P,P') = \sum d(P_i,P'_i)$ we can output a trivial YES answer if $\sum d_i \leq k/2$, and a trivial NO if $\sum d_i > k$, so for all non-trivial instances we have $k/2 < \sum_{i=1}^{m} d_i \leq k$.

Now we present a trivial observation and a lemma that will be useful to prove our final result.

Observation 1 *A pair of quadrilaterals with no common diagonals have distance 1. Similarly, a pair of pentagons with no common diagonals have distance 2.*

Lemma 2 *Given a set of m pairs of polygons $(P_i,P'_i) \in \mathcal{P}_{n_i}$, we can build a pair of polygons $(P,P') \in \mathcal{P}_n$ with $n = \sum_{i=1}^{m} d_i + m + 2$, and $d(P,P') = \sum_{i=1}^{m} d(P_i,P'_i)$.*

Proof. Given two pairs of polygons $(P_1,P'_1),(P_2,P'_2)$ with sets of outer edges $\{e^1_1,e^1_2,\ldots,e^1_{n_1}\}$ and $\{e^2_1,e^2_2,\ldots,e^2_{n_2}\}$ we can create a pair of polygons with edges:

$$\{e^1_1,e^1_2,\ldots,e^1_{n_1-1},e^2_1,e^2_2,\ldots,e^2_{n_2-1}\}$$

We add to that polygon all diagonals present in both (P_1,P'_1) and (P_2,P'_2), plus a diagonal δ_s in place of $e^1_{n_1}$ and $e^2_{n_2}$. It is clear that we can add those diagonals and they will be non-crossing. We can see an example of this operation in Figure 3.

This way we have a new pair of polygons (P,P') in which the edges $\{e^1_1,e^1_2,\ldots,e^1_{n_1-1},\delta_s\}$ induce the polygons (P_1,P'_1) and $\{e^2_1,e^2_2,\ldots,e^2_{n_2-1},\delta_s\}$ the polygons (P_2,P'_2). Since δ_s is a common diagonal and thus is never flipped in a shortest path, $d(P,P') = d(P_1,P'_1) + d(P_2,P'_2)$. The size of (P,P') is $n_1+n_2-2 = d_1+d_2+4$, and it has d_1+d_2+1 diagonals, of which at least one of them is common. Repeated applications of this operation complete the proof. $\square$

Theorem 3 *There is a kernel of size $\frac{4k}{3} + \mathcal{O}(1)$ for the* Parameterized Flip Distance *problem. Specifically, given a pair of polygons $(P,P') \in \mathcal{P}_n$ and a parameter k we can output in polynomial time another pair of polygons $(P^*,P^{*\prime})$ of size at most $\frac{4k}{3}+2$, and a parameter $k' \leq k$ such that:*

$$d(P,P') \leq k \iff d(P^*,P^{*\prime}) \leq k'$$

Proof. Given $(P,P') \in \mathcal{P}_n$, and a parameter k, the following algorithm outputs a kernel of the problem of size at most $\frac{4k}{3}+2$.

1. Divide (P,P') along their common diagonals to obtain m pairs of polygons (P_i,P'_i) and discard all pairs that have only three edges, because their distance is 0. Now we have m' pairs, with $m' \leq m$, so we re-number the pairs to have $(P_i,P'_i), i \in [1,m']$.

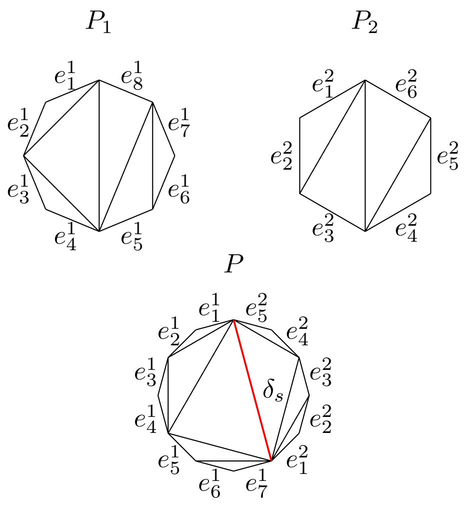


Figure 3: Given two pairs of polygons (P_1,P'_1) and (P_2,P'_2) we can generate a new pair (P,P') that has distance equal to the sum of the distances of the original pair. In this example (P_1,P'_1) has 8 outer edges $\{e^1_1,e^1_2,e^1_3,e^1_4,e^1_5,e^1_6,e^1_7,e^1_8\}$ and (P_2,P'_2) has 6 outer edges $\{e^2_1,e^2_2,e^2_3,e^2_4,e^2_5,e^2_6\}$, resulting in a pair (P,P') with 12 edges $\{e^1_1,e^1_2,e^1_3,e^1_4,e^1_5,e^1_6,e^1_7,e^2_1,e^2_2,e^2_3,e^2_4,e^2_5\}$ (in this figure only one of the polygons of the pair is shown since we operate identically with the other).

2. Making use of Observation 1, discard all pairs with four edges and reduce the parameter k by one per each pair removed that way. Proceed the same way with pairs of five edges reducing the parameter by two instead and renumber the pairs as we did in the previous step. We get a new parameter $k' \leq k$.

3. If $\sum d_i > k'$ output NO. If $\sum d_i \leq k'/2$ output YES.

4. Use Lemma 2 to create a new polygon $(P^*,P^{*\prime})$ from all the remaining pairs. The new instance is defined by $(P^*,P^{*\prime})$ and k'.

Since we have removed all pairs with $d_i < 3$, each pair has at least 3 diagonals, none of them common, so m is at most $k'/3$, or otherwise $\sum d_i > k'$, and we could have output a trivial NO answer. Also, we have that $k'/2 < \sum_{i=1}^{m} d_i \leq k'$, and by making use of Lemma 2 to obtain the pair of polygons $(P^*,P^{*\prime})$, they will be of size $\sum_{i=1}^{m} d_i + m + 2 \leq k' + k'/3 + 2 \leq 4k/3 + 2$.

Also, from Lemma 1, Lemma 2 and Observation 1 it is clear that $d(P,P') \leq k$ if and only if $d(P^*,P^{*\prime}) \leq k'$, and Lucas [9] showed that the first step can be done in $\mathcal{O}(n^2)$ time, while the last step can be done in time $\mathcal{O}(n)$, completing the proof. $\square$

3.2 $(1+\epsilon)$-kernel

In this section we will show a procedure that allows us to obtain a kernel of size $(1+\epsilon)k+\mathcal{O}(1)$ in time that is polynomial in n when $\epsilon > 0$ is constant. The procedure is based on the algorithm of the previous section and on the trivial exponential-time algorithm that allows us to solve an instance of size n in time $\mathcal{O}(n^{2n})$ by trying all possible diagonal flips recursively.

Theorem 4 *Given a pair of polygons $(P, P') \in \mathcal{P}_n$ and a parameter k, we can output another pair of polygons $(P^*, P^{*\prime})$ of size at most $(1+\epsilon)k+2$ and a parameter $k' \leq k$ such that:*

$$d(P, P') \leq k \iff d(P^*, P^{*\prime}) \leq k'$$

In time $\mathcal{O}(n^2 + f(\epsilon)n)$, where $f(\epsilon)$ is a function that only depends on ϵ.

Proof. Given two polygons $(P, P') \in \mathcal{P}_n$, a parameter $k \in \mathbb{N}$ and some $\epsilon > 0$, apply steps 1 and 2 of the algorithm described in Theorem 3. Then solve all instances of size less than $1/\epsilon + 3$, i.e. instances that have fewer than $1/\epsilon$ diagonals, using the trivial exponential-time algorithm. We can do this in time $\mathcal{O}((1/\epsilon)^{2/\epsilon} \cdot n)$, (because there can be at most $n-2$ instances after splitting common diagonals) and discard all pairs solved this way, reducing the parameter k by the sum of the flip distances of the pairs solved this way. Finally, apply steps 3 and 4 of the algorithm.

Now each remaining pair before step 3 will have at least $1/\epsilon$ diagonals, so m must be at most $\epsilon k'$, by a similar reasoning as in Theorem 3. Then we have that the polygons $(P^*, P^{*\prime})$ will have size at most $\sum_{i=1}^{m} d_i + m + 2 \leq k' + \epsilon k' + 2 \leq (1+\epsilon)k + 2$.

Since the steps common with the $\frac{4k}{3}$ kernelization algorithm can be done in time $\mathcal{O}(n^2)$, and the additional time spent on solving small instances is at most $\mathcal{O}((1/\epsilon)^{2/\epsilon} \cdot n)$, the total time required to produce the kernel is $\mathcal{O}(n^2 + (1/\epsilon)^{2/\epsilon} \cdot n) = \mathcal{O}(n^2 + f(\epsilon)n)$, as we wanted to prove. □

We note that by using the recent FPT algorithm of [5] instead of the trivial exponential-time algorithm, we can solve an instance of size less than $1/\epsilon$ in time $\mathcal{O}(32^{1/\epsilon} \cdot \text{poly}(1/\epsilon))$ instead of $\mathcal{O}((1/\epsilon)^{2/\epsilon})$, which is a significant improvement.

As mentioned earlier there is a near equivalence between the flip distance problem on simple convex polygons, and the rotation distance problem on two ordered, rooted binary trees. The definition of the rotation distance problem is rather technical so we omit details. In any case, it is well-known that an instance of rotation distance of size n (where the size here denotes the number of non-leaf nodes in one of the input trees) can be easily mapped to an instance of flip distance of size $n+2$, such that the distance is preserved. The mapping goes both ways [12]. Hence, the kernel obtained in Theorem 4 (and that of Theorem 3) also goes through for rotation distance, up to additive terms.

Corollary 5 *For each $\epsilon > 0$ there is a kernel of size $(1+\epsilon)k$ for the rotation distance problem.*

4 Discussion

Our kernel makes use of the fact that two polygons of size n with no common diagonals have $n-3$ non-common diagonals, which is a lower bound on the flip distance. Hence, for such "fully reduced" instances the ratio of the instance size to the flip distance is at most $\frac{n}{n-3}$ which is $1+o(1)$. In this sense, our $(1+\epsilon)k$ kernel feels like a natural result for this problem. It would be interesting to explore alternative, less inflated parameterizations of the problem. For example, if we let d denote the number of non-common diagonals in an instance, we could ask: is the flip distance $\leq d+k$? It could also be interesting to study the possibility of reduction rules that reduce the number of non-common diagonals in an instance, since so far all work undertaken has been done by reducing the common diagonals. We also note that there has been quite sophisticated parameterized complexity work undertaken on the flip distance problem in recent years, although most of it has been done on more general versions of it: in triangulations of point sets on the plane [5, 8]. We wonder whether those results can be strengthened in our more restricted setting i.e. the simple convex polygon case.

We note in passing that our improved kernel does not lead to an improvement of the polynomial-time approximation algorithm by Cleary et al. [2]. That article uses a similar technique to Lucas, but the limiting factor there is the algorithmic upper bound, which is an algorithm that takes in the worse case two flips to fix each non-common diagonal.

Finally, we return to rotation distance. As stated in Corollary 5, we obtain (up to an additive difference of 2) the same kernel result for rotation distance. For us, the additive term is insignificant, but for Lucas [10] it can be of importance. Lucas uses the correspondence with rotation distance to derive the $2k$ kernel. The bound there is based on the observation that, after splitting at common diagonals and deleting distance-0 subinstances, and letting d be the total number of non-common diagonals, there can be at most d subinstances of pairs of polygons, each with a corresponding pair of trees, and each such subinstance (i.e. pair of trees) has at least one non-root interior node. The worst case is when there are d subinstances, each with exactly one non-root interior node. (In the rotation distance problem non-root interior nodes correspond to diagonals in the flip distance problem.) In the rotation distance literature the size of an instance is usually taken to be the number of interior nodes, *including* the root ([4, 9, 11, 12] among others). This yields a bound of $2d \leq 2k$. However, when translated to flip distance, the worst case corresponds to d subinstances, each of which has exactly one non-common diagonal (and no common diagonals). Such subinstances are squares, and in the vast majority of the literature the size of the polygons is regarded as the number of outer edges [5, 7, 8, 12]. Taking that metric, Lucas' kernel would yield $4d \leq 4k$ for flip distance, not $2k$, so the kernel distorts when using the usual sizes of the problems. In a nutshell: Lucas left the kernel as a set of subinstances, but this can cause small additive terms to accumulate when switching between frameworks. Our kernel avoids such problems by merging the subinstances into a single instance at the end; this is the significance of the merging step.

5 Acknowledgements

We thank Steve Chaplick for useful discussions.

References

[1] O. Aichholzer, W. Mulzer, and A. Pilz. Flip distance between triangulations of a simple polygon is NP-Complete. *Discrete and Computational Geometry*, 54:368–389, 09 2015.

[2] S. Cleary and K. John. A Linear-Time Approximation Algorithm for Rotation Distance. *Journal of Graph Algorithms and Applications*, 14, 03 2009.

[3] S. Cleary and K. St. John. Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 07 2009.

[4] K. Culik and D. Wood. A note on some tree similarity measures. *Information Processing Letters*, 15(1):39–42, 1982.

[5] Q. Feng, S. Li, X. Meng, and J. Wang. An improved FPT algorithm for the flip distance problem. *Information and Computation*, page 104708, 2021.

[6] F. Fomin, D. Lokshtanov, S. Saurabh, and M. Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

[7] F. Hurtado and M. Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Computational Geometry*, 13(3):179–188, 1999.

[8] I. Kanj and G. Xia. Flip distance is in FPT time $O(n + k \cdot c^k)$. *Proceedings of STACS 2015; Leibniz International Proceedings in Informatics, LIPIcs*, 30:500–512, 02 2015.

[9] J. M. Lucas. Untangling Binary Trees via Rotations. *The Computer Journal*, 47(2):259–269, 01 2004.

[10] J. M. Lucas. An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12):481–484, 2010.

[11] J. Pallo. An efficient upper bound of the rotation distance of binary trees. *Information Processing Letters*, 73(3):87–92, 2000.

[12] D. Sleator, R. Tarjan, and W. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1(3):647–681, July 1988.

Minimum-Link Shortest Paths for Polygons amidst Rectilinear Obstacles[*]

Mincheol Kim[†] Hee-Kap Ahn[‡]

Abstract

Consider two axis-aligned rectilinear simple polygons in the domain consisting of axis-aligned rectilinear obstacles in the plane such that the bounding boxes, one for each obstacle and one for each polygon, are disjoint. We present an algorithm that computes a minimum-link rectilinear shortest path (a rectilinear shortest path with the minimum number of line segments) connecting the two polygons in $O((N+n)\log(N+n))$ time using $O(N+n)$ space, where n is the number of vertices in the domain and N is the total number of vertices of the two polygons.

1 Introduction

The problem of finding paths connecting two objects amidst obstacles has been studied extensively in the past. It varies on the underlying metric (Euclidean, rectilinear, etc.), types of obstacles (simple polygons, rectilinear polygons, rectangles, etc.), and objective functions (minimum length, minimum number of links, or their combinations). See the survey in Chapter 31 of Handbook of Discrete and Computational Geometry [18] on various approaches to this problem and results.

For two points p and q contained in the plane, possibly with rectilinear polygonal obstacles (i.e., a *rectilinear domain*), a rectilinear shortest path from p to q is a rectilinear path from p to q with minimum total length that avoids the obstacles. In the rest of the paper, we say a *shortest path* to refer to a rectilinear shortest path unless stated otherwise. A rectilinear path consists of horizontal and vertical segments, each of which is called *link*. Among all shortest paths from p to q, we are interested in a *minimum-link* shortest path from p to q, that is, a shortest path with the minimum number of links (or one with the minimum number of bends). There has been a fair amount of work on finding minimum-link shortest paths connecting two points amidst rectilinear obstacles in the plane [1, 10, 19, 20, 22].

These definitions are naturally extended to two more general objects contained in the domain. A shortest path connecting the objects is one with minimum path length among all shortest paths from a point of one object to a point of the other object. A minimum-link shortest path connecting the objects is a minimum-link path among all shortest paths.

In this paper, we consider the problem of finding minimum-link shortest paths connecting two objects in a rectilinear domain, which generalizes the case of connecting two points, in some modest environment. The rectilinear polygonal obstacles are considered as open sets. Two axis-aligned rectilinear polygons are said to be *box-disjoint* if the axis-aligned bounding boxes, one for each rectilinear polygon, are disjoint in their interiors. A set of axis-aligned rectilinear polygons is box-disjoint if the polygons of the set are pairwise box-disjoint. The rectilinear domain induced by a set of box-disjoint rectilinear polygons in the plane is called a *box-disjoint rectilinear domain.* We require the input objects and the obstacles in the domain to be also pairwise box-disjoint, unless stated otherwise.

Problem definition. Given two axis-aligned rectilinear simple polygons S and T in a rectilinear domain in the plane such that S, T, and the obstacles in the domain are pairwise box-disjoint, find a minimum-link rectilinear shortest path from S to T.

Related work. Computing shortest paths or minimum-link paths in a polygonal domain has been studied extensively. When obstacles are all rectangles, Rezende et al. [6] presented an algorithm with $O(n\log n)$ time and $O(n)$ space to compute a shortest path connecting two points amidst n rectangles. For a rectilinear domain with n vertices, Mitchell [12] gave an algorithm with $O(n\log n)$ time and $O(n)$ space to compute a shortest path connecting two points using a method based on the continuous Dijkstra paradigm [11]. Later, Chen and Wang [2] improved the time complexity to $O(n+h\log h)$ for a triangulated polygonal domain with h holes.

Computing a minimum-link path, not necessarily

[*]This research was partly supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)).

[†]Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. rucatia@postech.ac.kr

[‡]Graduate School of Artificial Intelligence, Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. heekap@postech.ac.kr

shortest, in a polygonal domain has also been studied well. For a minimum-link rectilinear path connecting two points in a rectilinear domain with n vertices, Imai and Asano [8] gave an algorithm with $O(n \log n)$ time and space. Then a few algorithms improved the space complexity to $O(n)$ without increasing the running time [4, 13, 16]. Very recently, Mitchell et al. [14] gave an algorithm with $O(n + h \log h)$ time and $O(n)$ space for triangulated rectilinear domains with h holes.

Yang et al. [20] considered the problem of finding a rectilinear path connecting two points amidst rectilinear obstacles under a few optimization criteria, such as a minimum-link shortest path, a shortest minimum-link path, and a least-cost path (a combination of link cost and length cost). By constructing a path-preserving graph, they gave a unified approach to compute such paths in $O(ne+n\log n)$ time, where n is the total number of polygon edges and e is the number of polygon edges connecting two convex vertices. The space complexity is $O(ne)$ due to the path-preserving graph of size $O(ne)$. Since e is $O(n)$, the running time becomes $O(n^2)$ in the worst case, even for convex rectilinear polygons (obstacles). A few years later, they gave two algorithms on the problem [22], improving their previous result, one with $O(n \log^2 n)$ time and $O(n \log n)$ space and the other with $O(n \log^{3/2} n)$ time and $O(n \log^{3/2} n)$ space using a combination of a graph-based approach and the continuous Dijkstra approach. It is claimed in [10] that a minimum-link shortest path can be computed in $\Theta(n \log n)$ time and $O(n)$ space when obstacles are rectangles by the algorithm in another paper [21] by the same authors, but the paper [21] is not available.

Later, Chen et al. [1] gave an algorithm improving the previous results for finding a minimum-link shortest path connecting two points in $O(n \log^{3/2} n)$ time and $O(n \log n)$ space using an implicit representation of a reduced visibility graph, instead of computing the whole graph explicitly. Very recently, Wang [19] pointed out a flaw in the algorithm in [1] and claimed that to make it work, each vertex of the graph must store a constant number of nonlocal optimum paths together with local optimum paths. Wang gave an algorithm with $O(n + h \log^{3/2} h)$ time and $O(n+h \log h)$ space using a reduced path-preserving graph from the corridor structure [14] and the histogram partitions [17], where h is the number of holes (obstacles) in the rectilinear domain.

However, we are not aware of any result on computing the minimum-link shortest path connecting two objects other than points.

Our results. We consider the minimum-link shortest path problem for two axis-aligned rectilinear polygons S and T in a box-disjoint rectilinear domain. This generalizes the two-point shortest path problem to two-polygon shortest path problem. The theorem below summarizes our results.

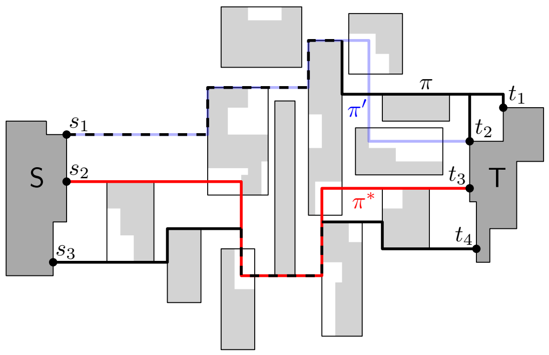


Figure 1: The box-disjoint rectilinear domain with S, T. Light gray rectilinear polygons are obstacles. There are six pairs of points, (s_1, t_1), (s_1, t_2), (s_2, t_3), (s_2, t_4), (s_3, t_3), and (s_3, t_4), that determine the length of a shortest path from S to T. The path π from s_1 to t_1 (or t_2) is a minimum-link shortest path from S to T with eight links among all shortest paths without intersecting the interiors of bounding boxes of obstacles. However, the blue path π' from s_1 to t_2 has seven links and the red path π^* from s_2 to t_3 has five links, which is optimal.

Theorem 1 *Let S and T be two axis-aligned rectilinear simple polygons with N vertices in a rectilinear domain with n vertices in the plane such that S, T, and the obstacles in the domain are pairwise box-disjoint. We can compute a minimum-link shortest path from S to T in $O((N + n) \log(N + n))$ time using $O(N + n)$ space.*

Sketch of our results. The main difficulty lies in computing a shortest path from S to T. The length of a shortest path from S to T is determined by a pair of points, one lying on the boundary of S and one lying on the boundary of T. Such a point is a vertex of S or T, or the point on the boundary of S or T where a horizontal or vertical ray emanating from a vertex in the domain hits first. Since the domain has $O(N + n)$ vertices, there are $O(N + n)$ such points on the boundaries of S and T, and $O((N + n)^2)$ pairs of points, one from S and the other from T, to consider in order to determine the length of a shortest path. Thus, if we use a naive approach that computes a minimum-link shortest path for each point pair, it may take $\Omega((N + n)^2)$ time. Theorem 1 shows that our algorithm computes a minimum-link shortest path from S to T efficiently. Also, a minimum-link shortest path may intersect the bounding box of an obstacle, although S, T, and obstacles are pairwise box-disjoint. See Figure 1.

We first consider a simpler problem for an axis-aligned line segment S and a point t contained in the domain consisting of axis-aligned rectangular obstacles. We partition the domain into at most eight regions using eight

xy-monotone paths from S. We observe that every shortest path from S to a point in a region is either x-, y-, or xy-monotone [6]. Moreover, we define a set of $O(n)$ *baselines* for each region, and show that there is a minimum-link shortest path from S to t consisting of segments contained in the baselines. Based on these observations, our algorithm applies a plane sweep technique with a sweep line moving from S to t and computes the minimum numbers of links from S to the intersections of the baselines and the sweep line efficiently. After the sweep line reaches t, our algorithm reports a minimum-link shortest path that can be obtained from a reverse traversal from t using the number of links stored in baselines. During the sweep, our algorithm maintains a data structure storing baselines (and their minimum numbers of links) and updates the structure for the segments (events) on the boundary of the region.

It takes, however, $O(n^2)$ time using $O(n)$ space. To reduce the time complexity without increasing the space complexity, our algorithm maintains another data structure, a *balanced binary search tree,* each node of which corresponds to a set of consecutive baselines. This tree behaves like a segment tree [5]. Instead of updating the minimum numbers of links of $O(n)$ baselines at each event of the plane sweep algorithm, we update $O(\log n)$ nodes of the tree that together correspond to the baselines. This improves the time for handling each sweep-line event from $O(n)$ to $O(\log n)$, and thus improving the total time complexity to $O(n \log n)$.

Then we extend our algorithm to handle a line segment T (not a point t) and box-disjoint rectilinear obstacles (not necessarily rectangles). We observe that every shortest path contained in a region from S to any point of T is either x-, y-, or xy-monotone, so our algorithm partitions the domain into at most eight regions again. Then T intersects at most five regions. Our algorithm computes a minimum-link shortest path from S to T' for the portion T' of T contained in each region, and then returns the minimum-link shortest path among the paths.

When S or T intersects some bounding boxes of obstacles, we consider each portion of S or T contained in a bounding box independently. The portion not contained in any bounding box can be handled as we do for segments disjoint from the boxes. For the portion contained in a bounding box $B(P)$ for a rectilinear polygon P, every minimum-link shortest path from S to T is the concatenation of a subpath contained in $B(P)$ and the subpath not contained in $B(P)$ such that both subpaths are minimum-link shortest paths sharing one point on the boundary of $B(P)$. Thus, our algorithm finds a subpath contained in $B(P)$ and a subpath not contained in $B(P)$ that together form a minimum-link shortest path from S to T. We observe that there is a minimum-link shortest path from S to T through certain points on the boundary of $B(P)$. By computing these points and their distances and minimum number of links to S and T, our algorithm computes a minimum-link shortest path from S to T in $O(n \log n)$ time for S or T intersecting the bounding boxes. Since an axis-aligned line segment intersects at most two bounding boxes, the overall running time remains to be $O(n \log n)$ time using $O(n)$ space.

Finally, we consider that the input objects are rectilinear simple polygons S and T with N vertices. Recall that there are $O((N+n)^2)$ pairs of points that determine the length of a shortest path from S to T. To handle them efficiently, we add $O(N)$ additional baselines and $O(N)$ events induced by S and T during the plane sweep algorithm. Then the number of events becomes $O(N+n)$ and the time to handle each event takes $O(\log(N+n))$, so we obtain Theorem 1.

Missing proofs and details can be found in the full version [9].

2 Preliminaries

Let R be a set of n disjoint axis-aligned rectangles in $\mathbb{R}^2$. Each rectangle $R \in \mathsf{R}$ is considered as an open set and plays as an obstacle in computing a minimum-link shortest path in the plane. We let $\mathsf{D} := \mathbb{R}^2 - \cup_{R \in \mathsf{R}} R$ and call it the *rectangular domain* induced by R in the plane. For two points p and q in D, $d(p, q)$ denotes the L_1 distance (or the Manhattan distance) from p to q in D, that is, the length of a shortest path from p to q avoiding the obstacles. A path is *x-monotone* if the intersection of the path with any line perpendicular to the x-axis is connected. Likewise, a path is *y-monotone* if the intersection of the path with any line perpendicular to the y-axis is connected. If a path is x-monotone and y-monotone, the path is *xy-monotone*.

For two objects S and T in D, $d(\mathsf{S}, \mathsf{T}) = \min_{p \in \mathsf{S}, q \in \mathsf{T}} d(p, q)$. A shortest path from S to T is a path in D from a point $p \in \mathsf{S}$ to a point $q \in \mathsf{T}$ of length $d(\mathsf{S}, \mathsf{T})$. A *minimum-link shortest path* from S to T is a path that has the minimum number of links among all shortest paths from S to T in D, and we use $\lambda(\mathsf{S}, \mathsf{T})$ to denote the number of links of a minimum-link shortest path from S to T. We call a pair (p, q) of points with $p \in \mathsf{S}$ and $q \in \mathsf{T}$ such that $d(\mathsf{S}, \mathsf{T}) = d(p, q)$ a *closest pair* of points of S and T. We say p is a closest point of S from T, and q is a closest point of T from S. Note that there can be more than one closest pair of points of S and T.

We make an assumption that the rectangles are in *general position*, that is, no two rectangles in R have corners, one corner from each rectangle, with the same x- or y-coordinate. A horizontal line segment H can be represented by the two x-coordinates $x_1(H)$ and $x_2(H)$

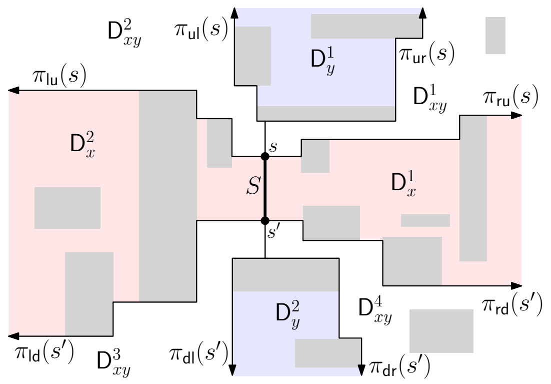


Figure 2: Eight disjoint regions of D by eight xy-monotone paths from s or s'. Gray rectangles are obstacles.

of its endpoints $(x_1(H) < x_2(H))$ and the y-coordinate $y(H)$ of them. Likewise, a vertical line segment V can be represented by the two y-coordinates $y_1(V)$ and $y_2(V)$ of its endpoints $(y_1(V) < y_2(V))$ and the x-coordinate $x(V)$ of them.

2.1 Eight disjoint regions of a rectangular domain

Given a rectangular domain D and a vertical segment S, we partition D into at most eight disjoint regions by using eight xy-monotone paths from the endpoints of S in a way similar to the one by Choi and Yap [3]. Consider a horizontal ray from a point $p = p_1$ on S going rightwards. The ray stops when it hits a rectangle $R \in \mathsf{R}$ at a point p'_1. Let p_2 be the top-left corner of R. We repeat this process by taking a horizontal ray from p_2 going rightwards until it hits a rectangle, and so on. The last horizontal ray goes to infinity. Then we obtain an xy-monotone path $\pi_{\mathsf{ru}}(p) = (p = p_1p'_1p_2p'_2\ldots)$. In other words, $\pi_{\mathsf{ru}}(p)$ is an xy-monotone path from p that alternates going *rightwards* (until hitting a rectangle) and going *upwards* (to the top-left corner of the rectangle).

By choosing two directions, one going either rightwards or leftwards horizontally, and one going either upwards or downwards vertically, and ordering the chosen directions, we define eight rectilinear xy-monotone paths with directions: rightwards-upwards (ru), upwards-rightwards (ur), upwards-leftwards (ul), leftwards-upwards (lu), leftwards-downwards (ld), downwards-leftwards (dl), downwards-rightwards (dr), and rightwards-downwards (rd). We use $\pi_\alpha(p)$ to denote them, where α is one in $\{\mathsf{ru},\mathsf{ur},\mathsf{ul},\mathsf{lu},\mathsf{ld},\mathsf{dl},\mathsf{dr},\mathsf{rd}\}$. Also, we use $\pi_\alpha(p,q)$ to denote the subpath of $\pi_\alpha(p)$ from p to $q \in \pi_\alpha(p)$.

Figure 2 illustrates these eight xy-monotone paths, four upward paths from the upper endpoint s of S and four downward paths from the lower endpoint s' of S. Observe that for a point $p \in \mathsf{D}$, the eight paths $\pi_\alpha(p)$ do not cross each other. Thus, by the eight paths, D is partitioned into eight regions. See Figure 2. We denote by D^1_{xy} (and D^2_{xy}, D^3_{xy}, D^4_{xy}) the region bounded by $\pi_{\mathsf{ru}}(s)$ and $\pi_{\mathsf{ur}}(s)$ (and by $\pi_{\mathsf{ul}}(s)$ and $\pi_{\mathsf{lu}}(s)$, by $\pi_{\mathsf{ld}}(s')$ and $\pi_{\mathsf{dl}}(s')$, by $\pi_{\mathsf{dr}}(s')$ and $\pi_{\mathsf{rd}}(s')$). We denote by D^1_x (and D^2_x) the region bounded by $\pi_{\mathsf{ru}}(s)$ and $\pi_{\mathsf{rd}}(s')$ (and by $\pi_{\mathsf{lu}}(s)$ and $\pi_{\mathsf{ld}}(s')$), and denote by D^1_y (and D^2_y) the region bounded by $\pi_{\mathsf{ur}}(s)$ and $\pi_{\mathsf{ul}}(s)$ (and by $\pi_{\mathsf{dl}}(s')$ and $\pi_{\mathsf{dr}}(s')$).

Lemma 2 *For a point $t \in \cup_{1\le i\le 4}\mathsf{D}^i_{xy}$, every shortest path from S to t is xy-monotone. For a point $t \in \cup_{1\le i\le 2}\mathsf{D}^i_x$, every shortest path from S to t is x-monotone. For a point $t \in \cup_{1\le i\le 2}\mathsf{D}^i_y$, every shortest path from S to t is y-monotone.*

From now on we simply use D_{xy}, D_x and D_y to denote D^1_{xy}, D^1_x and D^1_y, respectively, and assume that t lies in a region D' of the regions. The case that t lies in other regions can be handled analogously. For each horizontal side of the rectangles incident to D', we call the horizontal line containing the side a *horizontal baseline* of D'. Similarly, for each vertical side of the rectangles incident to D', we call the vertical line containing the side a *vertical baseline* of D'. The two vertical lines through S and t, and the three horizontal lines through s, s' and t are also regarded as vertical and horizontal baselines of D', respectively. We say a minimum-link shortest path π is *aligned to the baselines* if every segment of π is contained in a baseline of the corresponding region. By using Lemma 3, we find a minimum-link shortest path aligned to the baselines of each region.

Lemma 3 *There is a minimum-link shortest path from S to t that is aligned to the baselines of D'.*

3 t lies in D_{xy}

We consider the case that t lies in D_{xy}. By Lemma 2, every shortest path from S to t is xy-monotone and connects the upper endpoint s of S and t. Let c be the point with the maximum x-coordinate and the maximum y-coordinate among the points in $\pi_{\mathsf{ur}}(s) \cap \pi_{\mathsf{ld}}(t)$. Observe that c is defined uniquely as $\pi_{\mathsf{ur}}(s) \cap \pi_{\mathsf{ld}}(t)$ is connected and xy-monotone by the definition. Likewise, let c' be the point with the maximum x-coordinate and the maximum y-coordinate among the points in $\pi_{\mathsf{ru}}(s) \cap \pi_{\mathsf{dl}}(t)$. Then we use $\mathsf{D}_{xy}(s,t)$ to denote the region of D_{xy} enclosed by the closed curve composed of $\pi_{\mathsf{ur}}(s,c)$, $\pi_{\mathsf{ld}}(t,c)$, $\pi_{\mathsf{ru}}(s,c')$, and $\pi_{\mathsf{dl}}(t,c')$. We denote by $\partial_{xy}(s,t)$ the rectilinear chain of the outer boundary of $\mathsf{D}_{xy}(s,t)$ from s to t in clockwise order, and denote by $\partial_{xy}(t,s)$ the rectilinear chain of the outer boundary of $\mathsf{D}_{xy}(s,t)$ from t to s in clockwise order. See Figure 3(a) for an illustration. By Lemma 2, every shortest path from s to t is contained in $\mathsf{D}_{xy}(s,t)$, and therefore every minimum-link shortest path from s to t is also contained in $\mathsf{D}_{xy}(s,t)$.

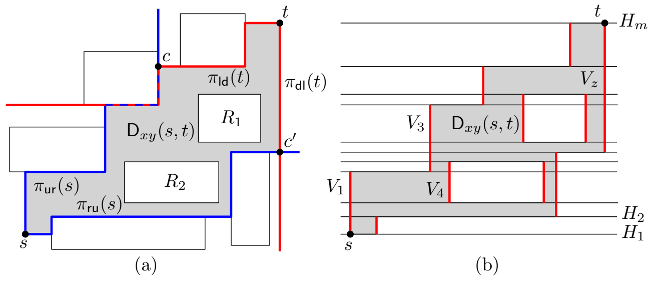


Figure 3: (a) $\mathsf{D}_{xy}(s,t)$ is the region of D_{xy} enclosed by the closed curve composed of $\pi_{\mathsf{ur}}(s,c)$, $\pi_{\mathsf{ld}}(t,c)$, $\pi_{\mathsf{ru}}(s,c')$, and $\pi_{\mathsf{dl}}(t,c')$. R_1 and R_2 are the holes of $\mathsf{D}_{xy}(s,t)$. (b) Horizontal baselines $H_1, H_2, \ldots, H_m$ of $\mathsf{D}_{xy}(s,t)$ and vertical segments (red) $V_1, V_2, \ldots, V_z$ on the boundary of $\mathsf{D}_{xy}(s,t)$.

We focus on the baselines of D_{xy} that are defined by s, t, and the rectangles incident to $\mathsf{D}_{xy}(s,t)$, which we call the baselines of $\mathsf{D}_{xy}(s,t)$. Figure 3(b) shows the horizontal baselines of $\mathsf{D}_{xy}(s,t)$. Note that a baseline may cross rectangles incident to $\mathsf{D}_{xy}(s,t)$. Let $H_1, H_2, \ldots, H_m$ be the m horizontal baselines of $\mathsf{D}_{xy}(s,t)$ such that $y(H_1) < y(H_2) < \ldots < y(H_m)$. Note that s is on H_1 and t is on H_m.

3.1 Computing the minimum number of links

Consider a minimum-link shortest path aligned to the baselines of $\mathsf{D}_{xy}(s,t)$. For the rightmost vertical segment V of $\mathsf{D}_{xy}(s,t)$, we have $y_2(V) = y(t)$ and $y_1(V) = y(H_{m'})$ for some horizontal baseline $H_{m'}$ with $m' < m$. We can compute a minimum-link shortest path once we have a minimum-link shortest path from s to the intersection point c_i of V and H_i for each $i = m', m'+1, \ldots, m$, since t is the endpoint of V.

We compute $\lambda(s,t)$ by applying the plane sweep algorithm, and then report a minimum-link shortest path aligned to the baselines of $\mathsf{D}_{xy}(s,t)$ that can be obtained from a reverse traversal from t using $\lambda(s,t)$.

Imagine a vertical line L sweeping $\mathsf{D}_{xy}(s,t)$ rightwards. Our plane sweep algorithm maintains a data structure storing horizontal baselines and their minimum numbers of links among shortest paths from s to intersections of baselines and L such that the line segments incident to the intersections of those shortest paths are horizontal. The algorithm updates their status and minimum numbers of links when L encounters the vertical segments (vertical baselines) on the boundary of $\mathsf{D}_{xy}(s,t)$.

We define the status for each horizontal baseline as follows. For the intersection point $c_i = H_i \cap L$ for each $i = 1, \ldots, m$, if $c_i \in \mathsf{D}_{xy}(s,t)$, then H_i is *active*. Otherwise, H_i is *inactive*. Observe that a baseline may switch its status between active and inactive, depending on the position of L, and these switches occur only when L encounters a vertical segment on the boundary of $\mathsf{D}_{xy}(s,t)$. During the sweep, we maintain the active baselines of $\mathsf{D}_{xy}(s,t)$ in a set of ranges with respect to their indices in a range tree $\mathcal{T}_{\mathsf{ran}}$. A range $[a,b]$ contained in $\mathcal{T}_{\mathsf{ran}}$ represents a set of active baselines $H_a, H_{a+1}, \ldots, H_b$, consecutive in their indices from a to b. Every range $[a,b]$ in $\mathcal{T}_{\mathsf{ran}}$ is *maximal* in the sense that H_{a-1} and H_{b+1} are inactive or not defined in $\mathsf{D}_{xy}(s,t)$. We use $M(i)$ to denote the minimum number of links among all shortest paths from s to c_i whose segment incident to c_i is horizontal.

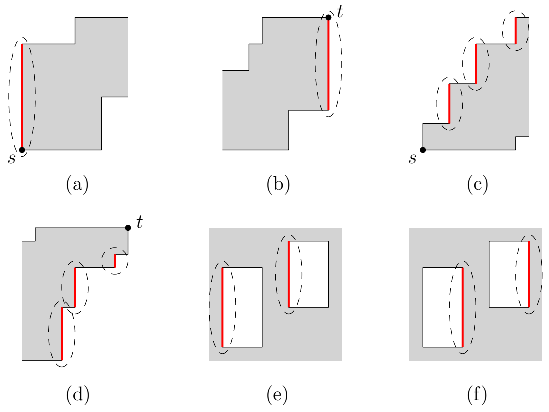


Figure 4: Six types of events of the plane sweep algorithm. (a) originate event. (b) terminate event. (c) attach event. (d) detach event. (e) split event. (f) merge event.

We maintain $M(i)$'s for horizontal baselines during the plane sweep as follows. There are vertical line segments $V_1, V_2, \ldots, V_z$ on the boundary of $\mathsf{D}_{xy}(s,t)$, satisfying $x(V_1) < x(V_2) < \ldots < x(V_z)$. Note that the lower endpoint of V_1 is s and the upper endpoint of V_z is t. We consider each vertical segment V_j $(1 \le j \le z)$ of $\mathsf{D}_{xy}(s,t)$ as an event, denoted by E_j, because we compute a minimum-link shortest path aligned to the baselines of $\mathsf{D}_{xy}(s,t)$, so $M(i)$ changes only when L encounters a vertical segment. For each E_j, we use $\alpha(j)$ and $\beta(j)$ (with $\alpha(j) < \beta(j)$) to denote the indices such that $y_1(V_j) = y(H_{\alpha(j)})$ and $y_2(V_j) = y(H_{\beta(j)})$, respectively. E_j belongs to one of the following six types depending on the boundary part of $\mathsf{D}_{xy}(s,t)$ that V_j lies on. See Figure 4 for an illustration of each type.

- E_1 belongs to type originate and E_z belongs to type terminate.
- E_j for each $j = 2, \ldots, z-1$ belongs to type attach if V_j lies on $\partial_{xy}(s,t)$, and to type detach if V_j lies on $\partial_{xy}(t,s)$.
- E_j belongs to type split if V_j is the left side of a hole of $\mathsf{D}_{xy}(s,t)$, and to type merge if V_j is the right side of a hole.

The events are sorted by their x-coordinates. During the sweep, L encounters E_j when $x(L) = x(V_j)$. Initially, the tree $\mathcal{T}_{\mathsf{ran}}$ contains no range, and $M(i)$ is set to ∞ for all horizontal baselines H_i. When L encounters E_1, which is the originate event with $\alpha(1) = 1$, we update $M(\alpha(1)) := 1$ and $M(k) := 2$ for each $k \in [\alpha(1)+1, \beta(1)]$, and insert the range $[\alpha(1), \beta(1)]$ into $\mathcal{T}_{\mathsf{ran}}$.

If E_j is an attach event, the inactive baselines H_i for i from $\alpha(j)+1$ to $\beta(j)$ become active. Observe that there always exists a range $[a', \alpha(j)]$ in $\mathcal{T}_{\mathsf{ran}}$ with $a' < \alpha(j)$. Thus, we remove $[a', \alpha(j)]$ from $\mathcal{T}_{\mathsf{ran}}$ and insert $[a', \beta(j)]$ into $\mathcal{T}_{\mathsf{ran}}$. Then we update $M(i) := \min_{k\in[a',\alpha(j)]}\{M(k)+2\}$ for each $i \in [\alpha(j)+1, \beta(j)]$.

If E_j is a detach event, the active baselines H_i for i from $\alpha(j)$ to $\beta(j)-1$ become inactive. Observe that there always exists a range $[\alpha(j), b']$ in $\mathcal{T}_{\mathsf{ran}}$ with $\beta(j) < b'$. Thus, we remove $[\alpha(j), b']$ from $\mathcal{T}_{\mathsf{ran}}$, and insert $[\beta(j), b']$ into $\mathcal{T}_{\mathsf{ran}}$. Then we update $M(i) := \min\{M(i), \min_{k\in[\alpha(j),\beta(j)-1]}(M(k)+2)\}$ for each $i \in [\beta(j), b']$.

If E_j is a split event, the active baselines lying in between $H_{\alpha(j)}$ and $H_{\beta(j)}$ become inactive. If there is such a baseline, there always exists a range $[a', b']$ in $\mathcal{T}_{\mathsf{ran}}$ with $a' < \alpha(j)$ and $\beta(j) < b'$. In this case, we remove $[a', b']$ from $\mathcal{T}_{\mathsf{ran}}$, insert $[a', \alpha(j)]$ and $[\beta(j), b']$ into $\mathcal{T}_{\mathsf{ran}}$, and update for each $i \in [\beta(j), b']$ $M(i) := \min\{M(i), \min_{k\in[a',\beta(j)-1]}\{M(k)+2\}\}$ for each $i \in [\beta(j), b']$.

If E_j is a merge event, the inactive baselines lying in between $H_{\alpha(j)}$ and $H_{\beta(j)}$ become active. If there is such a baseline, there always exist two ranges $[a', \alpha(j)]$ and $[\beta(j), b']$ in $\mathcal{T}_{\mathsf{ran}}$ with $a' < \alpha(j)$ and $\beta(j) < b'$. In this case, we remove $[a', \alpha(j)]$ and $[\beta(j), b']$ from $\mathcal{T}_{\mathsf{ran}}$, insert $[a', b']$ into $\mathcal{T}_{\mathsf{ran}}$, and update

$$M(i) := \begin{cases} \min_{k\in[a',\alpha(j)]} M(k) + 2 \\ \qquad \text{for } i \in [\alpha(j)+1, \beta(j)-1], \\ \min\{M(i), \min_{k\in[a',\alpha(j)]} M(k) + 2\} \\ \qquad \text{for } i \in [\beta(j), b']. \end{cases} \tag{1}$$

Our algorithm eventually finds $\lambda(s,t)$ when L encounters the terminate event E_z with $\beta(z) = m$. Then $\mathcal{T}_{\mathsf{ran}}$ has exactly one range $[\alpha(z), \beta(z)]$, and we remove it from $\mathcal{T}_{\mathsf{ran}}$. We take $\lambda(s,t) = \min\{M(m), \min_{k\in[\alpha(z),\beta(z)-1]} M(k) + 1\}$.

3.2 Computing a minimum-link shortest path

We compute a minimum-link shortest path from s to t aligned to the baselines of $\mathsf{D}_{xy}(s,t)$ using $\lambda(s,t)$. To do this, we add a horizontal line segment at each event, which we call a *canonical segment*. Then we report a minimum-link shortest path using these canonical segments.

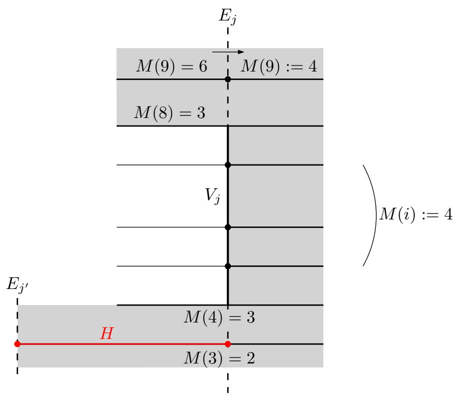


Figure 5: A merge event E_j. $M(3) = \min_{k\in[a',\alpha(j)]} M(k)$, which was updated from $E_{j'}$. The baselines H_i for $i = 5, \dots, 7$ become active, and $M(i)$ is updated to 4 by $M(3)$ (Equation 1). $M(9)$ is also updated by $M(3)$ (Equation 1). H is the canonical segment for E_j.

For instance, consider a merge event E_j. Recall that $\mathcal{T}_{\mathsf{ran}}$ has two disjoint ranges $[a', \alpha(j)]$ and $[\beta(j), b']$ with $a' < \alpha(j)$ and $\beta(j) < b'$. We update $M(i)$ using Equation 1. Let $k^* \in [a', \alpha(j)]$ be the smallest index such that $M(k^*)$ equals $\min_{k\in[a',\alpha(j)]} M(k)$. Assume that $M(k^*)$ was updated lately to the current value at an event $E_{j'}$ before L encounters E_j. Obviously, $x(V_{j'}) < x(V_j)$. We add a horizontal line segment H, which we call a canonical segment for E_j with $x_1(H) = x(V_{j'})$, $x_2(H) = x(V_j)$ and $y(H) = y(H_{k^*})$. See Figure 5.

We add one canonical segment for the merge event E_j. Likewise, we add one canonical segment per event of other types, except for the originate event. Since the x-coordinates of the events are distinct by the general position assumption, the right endpoints of the canonical segments we add are also distinct. Once the plane sweep algorithm is done, by following lemma, we can report a shortest path that has $\lambda(s,t)$ links.

Lemma 4 *There is a minimum-link shortest path from s to t whose horizontal line segments are all canonical segments.*

$\mathsf{D}_{xy}(s,t)$ can be obtained by ray shooting queries, each taking $O(\log n)$ time, using the data structure of Giora and Kaplan [7] with $O(n \log n)$ preprocessing time. Let h be the number of holes in $\mathsf{D}_{xy}(s,t)$, and o be the complexity of the outer boundary of $\mathsf{D}_{xy}(s,t)$. We can construct $\mathsf{D}_{xy}(s,t)$ in $O((o+h)\log n)$ time using $O(o+h)$ space.

Let z denote the number of events occurring during the sweep. At each of the z events, we remove and insert some ranges. Because the ranges in $\mathcal{T}_{\mathsf{ran}}$ are disjoint by the definition of $\mathcal{T}_{\mathsf{ran}}$, we can insert and remove a range in $O(\log m)$ time by using a simple balanced binary search tree for $\mathcal{T}_{\mathsf{ran}}$. We also set or update some $M(i)$'s at each event. For each event, if we know $\min_{k\in[a_1,b_1]} M(k)$ for a range $[a_1, b_1]$, we can update $M(i)$ for $i \in [a_2, b_2]$ (with $b_1 < a_2$) in time linear to the number of consecutive baselines from H_{a_2} to H_{b_2}, and the number of $M(i)$'s is $O(m)$. Therefore, it takes $O(m)$ time to handle an event. We use $O(m)$ space to maintain $\mathcal{T}_{\mathsf{ran}}$ and $M(i)$'s. In total, we use $O(o+h+m)$ space to compute $\lambda(s, t)$. We can report a minimum-link shortest path using $O(z)$ canonical segments. Thus, our algorithm takes $O((n+o+h)\log n + mz) = O(n^2)$ time and $O(o + h + m + z) = O(n)$ space.

Reducing the time complexity. To reduce the time complexity of our algorithm to $O(\log m)$ for handling each event while keeping the space complexity to $O(n)$ space, we build another balanced binary search tree $\mathcal{T}_{\mathsf{seg}}$, a variant of a segment tree in [5]. The idea is to use $\mathcal{T}_{\mathsf{seg}}$ together with $\mathcal{T}_{\mathsf{ran}}$ to maintain $O(\log m)$ nodes corresponding $O(m)$ $M(i)$'s efficiently, instead of updating $M(i)$'s for each event immediately. For each event, we have the range of indices of baselines inserted into (or removed from) $\mathcal{T}_{\mathsf{ran}}$. Using the range, we find $O(\log m)$ nodes in $O(\log m)$ time and update information for each node in constant time. The details can be found in the full version [9].

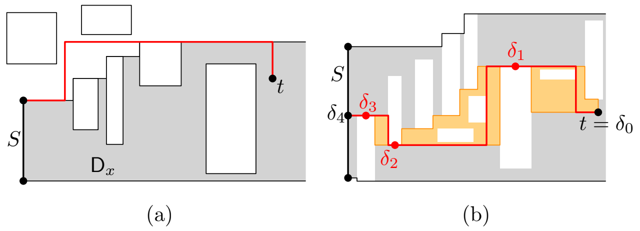
Lemma 5 *For a point t in D_{xy}, we can compute a minimum-link shortest path from S to t in $O(n \log n)$ time using $O(n)$ space.*

4 t lies in D_x or D_y

In this section, we assume that t lies in D_x. Then every shortest path from S to t is x-monotone by Lemma 2. In case that t lies in D_y, every shortest path is y-monotone and we can handle the case in a similar way. Unlike the case of $t \in \mathsf{D}_{xy}$, there can be a shortest path from S to t not contained in D_x. See Figure 6(a). However, we can compute a minimum-link shortest path from S to t using the algorithm in Section 3 as a subprocedure.

Let π denote a minimum-link shortest path from S to t aligned to the baselines, and let s^* and t be the two endpoints of π with $s^* \in S$. By definition, (s^*, t) is a closest pair of S and t. Since π is x-monotone, it is a concatenation of xy-monotone paths such that every two consecutive xy-monotone subpaths of π change their directions between monotone increasing and monotone decreasing on a horizontal segment, which we call a *winder*, of π.

Lemma 6 *Every winder of a shortest path from S to t contains one entire horizontal side of a rectangle in* R *incident to* D_x.



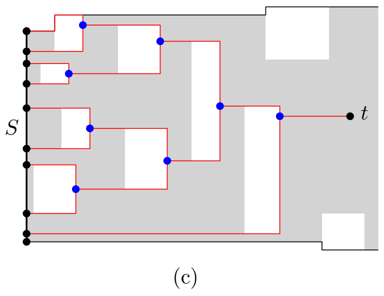


Figure 6: (a) A minimum-link shortest path from S to t. It is not contained in D_x. (b) Every shortest path from S to t passes through δ_i for $i = 0, \ldots, 4$ and is contained in $\bigcup_{i=0,\ldots,3} \mathsf{D}_{xy}(\delta_i, \delta_{i+1})$. (c) Shortest paths from S to t such that the closest pair of S and t is not unique.

Consider the horizontal sides of rectangles contained in the winders of a minimum-link shortest path π. Let g be the number of winders of π, and let δ_i be the midpoint of the horizontal side contained in the ith winder in order along π from t to s^*. We call such a midpoint a *divider* of π. For convenience, we let $t = \delta_0$ and $s^* = \delta_{g+1}$. Then the subpath from δ_i to δ_{i+1} for $0 \le i \le g$ of π is xy-monotone by the definition of the winders of π.

By Lemma 6, every winder of a minimum-link shortest path from S to t contains a horizontal side of a rectangle incident to D_x. Therefore, in the following we compute the dividers of a minimum-link shortest path among the midpoints of the top and bottom sides of each rectangle incident to D_x, and compute the xy-monotone paths connecting the dividers, in order, which together form a minimum-link shortest path.

We compute $d(S, t)$ by a plane sweep algorithm and find the dividers $\delta_1, \ldots, \delta_g$ of a minimum-link shortest path π as follows. For a rectangle $R \in \mathsf{R}$ incident to D_x, let $\delta(R)$ and $\delta'(R)$ denote the midpoints of the top and bottom sides of R, respectively. Then $\delta(R)$ and $\delta'(R)$ are candidates of the dividers of π. We consider each midpoint as an event during the sweep.

While sweeping D_x with a vertical line L moving rightwards, L encounters $\delta(R)$ and $\delta'(R)$ of a rectangle R at the same time. Consider two horizontal rays, one from $\delta(R)$ and one from $\delta'(R)$, going leftwards. We show how to handle the ray γ from $\delta(R)$. The ray from $\delta'(R)$ can be handled similarly. Let p_γ be the point of the vertical segment on the boundary of

D_x which γ hits first. If $p_\gamma \in S$, the shortest path from S to $\delta(R)$ is simply $p_\gamma\delta(R)$. If p_γ lies in $\pi_{\mathsf{ru}}(s)$ (or $\pi_{\mathsf{rd}}(s')$), every shortest path from S to $\delta(R)$ is xy-monotone. For these two cases, we store at $\delta(R)$ the distance $d(S,\delta(R))$ and the closest point (one of p_γ, s, or s') of S from $\delta(R)$. Consider the case that p_γ is in the right side of a rectangle $R' \in \mathsf{R}$ incident to D_x. We already have $d(S,\delta(R'))$ stored at $\delta(R')$ and $d(S,\delta'(R'))$ stored at $\delta'(R')$ during the plane sweep. Observe that every shortest path from $\delta(R)$ to $\delta(R')$ or to $\delta'(R')$ is xy-monotone, and the closest point s^o of S from $\delta(R)$ is the closest point of S from $\delta(R')$ or from $\delta'(R')$. Since $d(s^o,\delta(R)) = \min\{d(S,\delta(R')) + d(\delta(R'),\delta(R)), d(S,\delta'(R')) + d(\delta'(R'),\delta(R))\}$ by definition, we can compute s^o and $d(s^o,\delta(R))$ in constant time.

When L encounters t, we again consider a horizontal ray γ from t going leftwards and the point p_γ of the vertical segment on the boundary of D_x which γ hits first. If $p_\gamma \in S$, the minimum-link shortest path is simply $p_\gamma t$ and we are done. For p_γ lying on a side of a rectangle $R \in \mathsf{R}$ incident to D_x, if $d(t,\delta(R)) + d(S,\delta(R)) < d(t,\delta'(R)) + d(S,\delta'(R))$ (or the other way around without equality), we conclude there is no shortest path from S to t passing through $\delta'(R)$ (or through $\delta(R)$). Assume that π passes through $\delta(R)$. By the general position assumption, $y(\delta(R)) > y(t)$. Let R' be the rectangle incident to D_x that the horizontal ray from $\delta(R)$ going leftwards hits first. If $d(S,\delta(R')) + d(\delta(R'),\delta(R)) > d(S,\delta'(R')) + d(\delta'(R'),\delta(R))$ or there is no such rectangle R', $\delta(R)$ is a divider of π. Moreover, $\delta(R)$ is the first divider δ_1 of π from t, and thus every shortest path from $\delta_1 = \delta(R)$ to t is xy-monotone. Therefore, we construct $\mathsf{D}_{xy}(\delta(R),t)$ and apply the algorithm in Section 3. Then we apply this procedure from $\delta(R)$, recursively, and compute every xy-monotone subpath of π using canonical segments by Lemma 4, and glue them into one to form π. See Figure 6(b). Finally we obtain $g+1$ xy-monotone paths with dividers $\delta_0 = t, \delta_1, \ldots, \delta_{g+1} = s^* \in S$.

During the plane sweep, we find in $O(\log n)$ time the first rectangle hit by the horizontal ray γ emanating from a midpoint of a rectangle going leftwards using the data structure supporting ray shooting queries by Giora and Kaplan [7]. Thus, it takes $O(n\log n)$ time for ray shootings from midpoints in total. It takes $O(K_i)$ time to find a divider δ_i, where K_i is the number of the recursion depth of the algorithm to compute δ_i from δ_{i-1}. As shown in Section 3, computing an xy-monotone minimum-link shortest path from δ_i to δ_{i-1} takes $O(D_i \log D_i)$ time with $O(D_i)$ space after $O(n\log n)$-time preprocessing, where D_i is the number of the baselines defined by the rectangles incident to $\mathsf{D}_{xy}(\delta_i,\delta_{i-1})$. Observe that $\Sigma_{1\le i\le g+1}K_i = O(n)$, and $\Sigma_{1\le i\le g+1}D_i = O(n)$ because the regions $\mathsf{D}_{xy}(\delta_i,\delta_{i-1})$'s are disjoint in their interiors. Thus, the total time complexity is $O(n\log n)$ and the total space complexity is $O(n)$.

Handling degenerate cases. There can be two shortest paths from S to t, one passing through $\delta(R)$ and one passing through $\delta'(R)$ for a rectangle R. In this case, we have $d(t,\delta(R)) + d(S,\delta(R)) = d(t,\delta'(R)) + d(S,\delta'(R))$, which can be found in handling the midpoints of R during the plane sweep. Observe that this equality may occur multiple times in finding dividers of a minimum-link shortest path. Thus we need to devise an efficient way of maintaining all sequences of dividers, each of which may define a shortest path. See Figure 6(c). we show how to maintain these sequences of dividers and to find a minimum-link shortest path without increasing the time and space complexities in Lemma 7. The details can be found in the full version [9].

Lemma 7 *For a point t in $\mathsf{D}_x \cup \mathsf{D}_y$, we can compute a minimum-link shortest path from S to t in $O(n\log n)$ time using $O(n)$ space.*

5 Extending to a line segment T

Consider the case that the target is not just a point but an axis-aligned line segment T. We explain how the algorithm presented in previous sections works for T. Assume that T is a vertical line segment and $x(S) < x(T)$. We partition the domain D into eight regions using the eight monotone paths π_α's from S defined in Section 2.1. Then T intersects at most five regions D^1_x, D^1_{xy}, D^4_{xy}, D^1_y, and D^2_y. For the portion T' of T contained in each region, we compute a minimum-link shortest path from S to T'.

For the portion of T contained in a region of $\mathsf{D}^1_{xy}, \mathsf{D}^4_{xy}, \mathsf{D}^1_y$ and D^2_y, the closest point of S from T' is an endpoint of S and the closest point in T' from S is an endpoint of T' by Lemma 2. Thus we just apply the algorithms in Sections 3 and 4 for the corresponding endpoints of S and T'.

Consider the case that $T' \subset \mathsf{D}^1_x$. A minimum-link shortest path from S to T' connects S and an endpoint of T' or the intersection point t' of T' with a horizontal baseline of D_x. We can compute the distance from S to two endpoints of T' using the algorithm in Section 4. There are $O(n)$ intersection points on T' with horizontal baselines of D_x. During the plane sweep, we have $d(S,\delta(R))$ and $d(S,\delta'(R))$ for each hole R of D_x such that the horizontal baselines defined by R intersects T'. Thus, we can compute the distance from S to each intersection point t' on T' after the plane sweep. Then we obtain all the closest pairs of S and T'.

If there is only one closest pair, or the closest point of T' from S is the same for all closest pairs, we can compute a minimum-link shortest path from S to T'

as we do in Section 4. Otherwise, we can compute a minimum-link shortest path from S to T' using technical lemmas in the full version [9].

We can compute the portions T' of T contained in each of the five regions in $O(\log n)$ time using binary search along each path π_α and computing an intersection of T and π_α. For $T' \subset \mathsf{D}_x^1$, we can find the closest pairs in $O(n \log n)$ time if we use the ray shooting structure of Giora and Kaplan [7]. For each T' we use our algorithm in Sections 3 and 4 with $O(n \log n)$ time and $O(n)$ space, and eventually find a minimum-link shortest path from S to T by choosing $\min \lambda(S, T')$ for all T'.

Lemma 8 *Given two axis-aligned line segments S and T in a rectangular domain with n disjoint rectangular obstacles in the plane, we can compute a minimum-link shortest path from S to T in $O(n \log n)$ time using $O(n)$ space.*

6 Extending to box-disjoint rectilinear polygons

We show how to extend our algorithm in previous sections so that it handles box-disjoint rectilinear polygons. Let R_P be a set of box-disjoint rectilinear polygons, and let $B(P)$ denote the bounding box of a polygon $P \in \mathsf{R}_P$. We use $\mathsf{C} := \mathbb{R}^2 - \cup_{P \in \mathsf{R}_P} P$ to denote a *box-disjoint rectilinear domain* induced by R_P in the plane. A set Q is *rectilinear convex* if and only if any line parallel to the x- or y-axis intersects Q in at most one connected component. The rectilinear convex hull of P, denoted by $\mathsf{CH}(P)$, is the common intersection of all rectilinear convex sets containing P.

We assume that both S and T are disjoint from the rectangles $B(P)$ for $P \in \mathsf{R}_P$. Then no shortest path intersects the interior of $\mathsf{CH}(P)$ for $P \in \mathsf{R}_P$. If there is a shortest path π intersecting the interior of $\mathsf{CH}(P)$ for a rectilinear polygon $P \in \mathsf{R}_P$, π can be shortened by replacing each connected portion of π contained in the interior with the boundary curve of $\mathsf{CH}(P)$ between the endpoints of the portion, a contradiction. Thus, we replace each polygon P with $\mathsf{CH}(P)$ and find a minimum-link shortest path from S to T avoiding $\mathsf{CH}(P)$'s. We assume that each polygon $P \in \mathsf{R}_P$ is rectilinear convex in this subsection. If there is a shortest path π from S to T intersecting $B(P)$ for $P \in \mathsf{R}_P$, the subpath $\pi \cap B(P)$ can be replaced with a subpath along the boundary of $B(P)$ without increasing the length. This implies that there is a shortest path from S to T avoiding $B(P)$ for all $P \in \mathsf{R}_P$. From Lemma 2, every shortest path from S to T avoiding $B(P)$ for all $P \in \mathsf{R}_P$ is either x-, y-, or xy-monotone. The two subpaths have same length and endpoints, so they have the same monotonicity: One is X-monotone if and only if the other is X-monotone, for $X \in \{x, y, xy\}$. Therefore, every shortest path from S to T contained in C is either x-, y-, or xy-monotone.

Here we partition the domain into eight disjoint regions using eight xy-monotone paths as follows. We define the eight xy-monotone paths from S in a way slightly different to the one in Section 2.1. Consider the horizontal ray emanating from $s = p_1$ going rightwards. Let $P \in \mathsf{R}_P$ be the polygon such that $B(P)$ is the first rectangle hit by the ray among the rectangles, at point b on its left side. If the upper endpoint q of the leftmost vertical side of P lies above b, we set p'_1 to b and continue with the vertical ray from p'_1 to q, and continue along the boundary chain of P from q to the left endpoint p_2 of the topmost side of P in clockwise order. Otherwise, the horizontal ray continues going rightwards until it hits P at a point b'. Then we set p'_1 to b' and continue along the boundary chain of P from p'_1 to the left endpoint p_2 of the topmost side of P in clockwise order. We repeat this process by taking the horizontal ray from p_2 going rightwards. Then we obtain an xy-monotone path $\pi_{\mathsf{ru}}(p) = (p = p_1, p'_1, p_2, p'_2, \ldots)$, by following the boundary chain of P from p'_i to p_{i+1} in clockwise order. Thus, $\pi_{\mathsf{ru}}(p)$ is an xy-monotone path from p that alternates going horizontally rightwards and going vertically upwards. We define eight xy-monotone paths $\pi_\alpha(p)$ as in Section 2.1. Using these eight xy-monotone paths, we construct at most eight disjoint regions.

Using those regions, we compute a minimum-link shortest path from S to the portion of T contained in each region. Let T' be the portion of T contained in D_{xy}. The closest pair (s, t) of S and T' consists of their endpoints. We compute $\mathsf{D}_{xy}(s, t)$ using the method in Section 3. Observe that every shortest path from S to T' is contained in $\mathsf{D}_{xy}(s, t)$. With $O(n)$ baselines defined by the sides of $B(P)$ and the boundary segments of P incident to D_{xy} for all $P \in \mathsf{R}_P$, we can show that there is a minimum-link shortest path from S to T' which is aligned to the baselines using an argument similar to the proof of Lemma 3. Hence, we can compute a minimum-link shortest path from S to T' in the same time and space as in Lemma 5. Similarly, we can compute a minimum-link shortest path from S to T' for the portions T' of T contained in other regions. When T' is contained in D_x, a minimum-link shortest path may have some winders, each of which contains the topmost or the bottommost side of P for a rectilinear polygon $P \in \mathsf{R}_P$. This can be shown by an argument similar to the proof of Lemma 6. Thus we can compute $d(S, T')$ using the same plane sweep algorithm on D_x, and find the dividers which are midpoints of the topmost or the bottommost side of P as we do in Section 4 in the same time and space stated in Lemma 7.

S or T intersects bounding boxes. When S or T intersects some bounding boxes of obstacles, we consider each portion of S or T contained in a bounding box independently. The portion not contained in any bounding box can be handled as we do for segments disjoint from the boxes. For the portion contained in a bounding box $B(P)$ for a rectilinear polygon P, every minimum-link shortest path from S to T is the concatenation of a subpath contained in $B(P)$ and the subpath not contained in $B(P)$ such that both subpaths are minimum-link shortest paths sharing one point on the boundary of $B(P)$. Using that property, we can compute a minimum-link shortest path from S to T. The overall running time remains to be $O(n \log n)$ time using $O(n)$ space. See the full version [9] for details.

Lemma 9 *For two axis-aligned line segments S and T in C such that both S and T are disjoint from $B(P)$ for all $P \in \mathsf{R}_P$, we can compute a minimum-link shortest path from S to T in C in $O(n \log n)$ time using $O(n)$ space.*

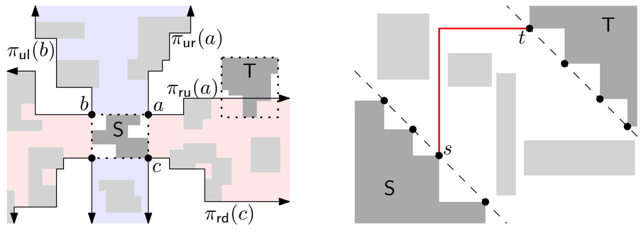
7 Extending to two polygons S and T

Now we consider two rectilinear polygons S and T with N vertices in C. We can compute a minimum-link shortest path from S to T using our algorithms in previous sections. Since S, T, and obstacles are pairwise box-disjoint, the distance $d(\mathsf{S}, \mathsf{T})$ between S and T can be represented as $d(\mathsf{S}, \mathsf{T}) = \min_{s \in B(\mathsf{S}), t \in B(\mathsf{T})}\{d(s,t) + \min_{s' \in \mathsf{S}} d(s,s') + \min_{t' \in \mathsf{T}} d(t,t')\}$. If we construct the L_1 Voronoi diagram of N boundary segments of S (or T) [15] in $O(N \log N)$ time using $O(N)$ space, we can maintain and report $\min_{s' \in \mathsf{S}} d(s,s')$ and $\min_{t' \in \mathsf{T}} d(t,t')$ for any $s \in B(\mathsf{S})$ and $t \in B(\mathsf{T})$ in $O(\log N)$ query time. From this observation, together with Lemma 2, we have the following lemma.

Lemma 10 *If there is an x-monotone shortest path from S to T, then every shortest path from S to T is x- or xy-monotone. If there is a y-monotone shortest path from S to T, then every shortest path from S to T is y- or xy-monotone.*

From Lemma 10, we can partition the box-disjoint rectilinear domain into eight disjoint regions using eight xy-monotone paths from $B(\mathsf{S})$ as done in Section 6. See Figure 7(a). There are $O(N)$ vertical and horizontal baselines defined by the boundary segments of S and T, and Lemma 3 also holds. Thus, we compute a minimum-link shortest path aligned to the baselines of each region in which the portion of T is contained.

Let T' be the portion of T contained in D_{xy}. Since S and T' are rectilinear polygons, there can be more than one closest pair of points for S and T'. Moreover, the points appearing in the closest pairs are on line segments with slopes ± 1. See Figure 7(b). If we compute $\mathsf{D}_{xy}(s,t)$ for every closest pair (s,t) of S and T', the time and space complexities may increase. Instead, we modify the plane sweep algorithm in Section 3 slightly. There can be more than one originate



(a) (b)

Figure 7: (a) Eight regions of C by eight xy-monotone paths from four corners of $B(\mathsf{S})$ with box-disjoint obstacles. T intersects at most five regions. (b) There are nine closest pairs of S and T. Among all paths connecting closest pairs, the minimum-link shortest path from s to t is the optimal.

and terminate events during the plane sweep because there can be more than one closest pair of S and T'. Also, there are no attach and detach events since we do not compute $\mathsf{D}_{xy}(s,t)$. For each event E_j in Section 3, however, we use $\alpha(j)$, $\beta(j)$ and $\mathcal{T}_{\mathsf{ran}}$ to maintain active baselines. Since we have all points in closest pairs of S and T, we set two horizontal baselines with the smallest and largest y-coordinate from those points, respectively. The horizontal baselines between the two baselines are used for inserting the range, which represents active baselines, into $\mathcal{T}_{\mathsf{ran}}$ of each event. Since there are $O(N+n)$ baselines between the two baselines, the time to handle an event takes $O(\log(N+n))$ time. Also, there are additional $O(N)$ originate and terminate events with $O(n)$ the other events, so we can compute a minimum-link shortest path from S to T' in $O((N+n)\log(N+n))$ time using $O(N+n)$ space.

Let T' be the portion of T contained in D_x. Every shortest path from S to T' is x-monotone, so we can compute the closest pairs of S and T' as the sweep line encounters each vertical line segments of T' using the plane sweep algorithm in Section 4. Then we can compute dividers in the same way without modifying the algorithm in Section 4. Lemmas related to dividers in Section 4 still hold, so we can compute a minimum-link shortest path connecting dividers similarly. As above, we can compute a minimum-link shortest path from S to a divider (or from a divider to T'). This implies we obtain a minimum-link shortest path from S to T'. We omit the details.

Therefore, we have Theorem 1.

8 Conclusion

We present an algorithm to compute a minimum-link shortest path connecting two rectilinear polygons in the box-disjoint rectilinear domain efficiently. Our algorithm computes a minimum-link shortest path from a point to the line segment using plane sweep, based on the monotonicity of the optimal path. Then we can extend objects to rectilinear polygons and apply a slightly modified algorithm.

Still there are quite a few problems to study. One typical problem is to compute a minimum-link shortest path connecting two objects in a general rectilinear domain such that the obstacles in the domain are not necessarily box-disjoint. There is a previous work in a general rectilinear domain, but there seem some gaps to the optimal time and space complexities.

References

[1] D. Chen, O. Daescu, and K. Klenk. On geometric path query problems. *International Journal of Computational Geometry & Applications*, 11(6):617–645, 2001.

[2] D. Chen and H. Wang. L_1 shortest path queries among polygonal obstacles in the plane. In *30th International Symposium on Theoretical Aspects of Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[3] J. Choi and C. Yap. Monotonicity of rectilinear geodesics in d-space. In *Proceedings of the Annual Symposium on Computational Geometry*, pages 339–348, 1996.

[4] G. Das and G. Narasimhan. Geometric searching and link distance. In *Workshop on Algorithms and Data Structures*, pages 261–272. Springer, 1991.

[5] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[6] P. De Rezende, D.-T. Lee, and Y.-F. Wu. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete & Computational Geometry*, 4:41–53, 1989.

[7] Y. Giora and H. Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3):28:1–51, 2009.

[8] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM Journal on Computing*, 15(2):478–494, 1986.

[9] M. Kim and H.-K. Ahn. Minimum-link shortest paths for polygons amidst rectilinear obstacles. *arXiv preprint arXiv:2106.14185*, 2021.

[10] D.-T. Lee, C.-D. Yang, and C. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70(3):185–215, 1996.

[11] J. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles in the plane. In *Abstracts of the 1st Canadian Conference on Computational Geometry*, volume 22, 1989.

[12] J. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1–6):55–88, 1992.

[13] J. Mitchell, V. Polishchuk, and M. Sysikaski. Minimum-link paths revisited. *Computational Geometry*, 47(6):651–667, 2014.

[14] J. Mitchell, V. Polishchuk, M. Sysikaski, and H. Wang. An optimal algorithm for minimum-link rectilinear paths in triangulated rectilinear domains. *Algorithmica*, 81(1):289–316, 2019.

[15] E. Papadopoulou and D. Lee. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry & Applications*, 11(05):503–528, 2001.

[16] M. Sato, J. Sakanaka, and T. Ohtsuki. A fast line-search method based on a tile plane. In *IEEE International Symposium on Circuits and Systems*, volume 5, pages 588–591, 1987.

[17] S. Schuierer. An optimal data structure for shortest rectilinear path queries in a simple rectilinear polygon. *International Journal of Computational Geometry & Applications*, 6(02):205–225, 1996.

[18] C. Toth, J. O'Rourke, and J. Goodman. *Handbook of discrete and computational geometry*. CRC press, 3rd edition, 2017.

[19] H. Wang. Bicriteria rectilinear shortest paths among rectilinear obstacles in the plane. *Discrete & Computational Geometry*, 62:525–582, 2019.

[20] C.-D. Yang, D.-T. Lee, and C. Wong. On bends and lengths of rectilinear paths: a graph-theoretic approach. *International Journal of Computational Geometry & Applications*, 2(01):61–74, 1992.

[21] C.-D. Yang, D.-T. Lee, and C. Wong. On minimum-bend shortest recilinear path among weighted rectangles. In *Tech. Report 92-AC-122*. Dept. of EECS, Northwestern Univ, 1992.

[22] C.-D. Yang, D.-T. Lee, and C. Wong. Rectilinear path problems among rectilinear obstacles revisited. *SIAM Journal on Computing*, 24(3):457–472, 1995.

An Optimal Algorithm for L_1 Shortest Paths in Unit-Disk Graphs[*]

Haitao Wang[†] Yiming Zhao[‡]

Abstract

A unit-disk graph $G(P)$ of a set P of points in the plane is a graph with P as its vertex set such that two points of P are connected by an edge if the distance between the two points is at most 1 and the weight of the edge is equal to the distance of the two points. Given P and a source point $s \in P$, we consider the problem of finding shortest paths in $G(P)$ from s to all other vertices of $G(P)$. In the L_2 case where the distance is measured by the L_2 metric, the problem has been extensively studied and the current best algorithm runs in $O(n \log^2 n)$ time, with $n = |P|$. In this paper, we study the L_1 case in which the distance is measured under the L_1 metric (and each disk becomes a diamond); we present an $O(n \log n)$ time algorithm, which matches the $\Omega(n \log n)$-time lower bound.

1 Introduction

Let P be a set of n points in the plane. The *unit-disk graph* $G(P)$ of P is a graph with P as its vertex set such that two points of P are connected by an edge if the distance between the two points is at most 1. Alternatively, $G(P)$ is the intersection graph of the set of disks centered at the points of P with radii equal to 1/2. Each edge of $G(P)$ has a weight that is equal to the distance of the two incident vertices of the edge.

In this paper, we consider the *single-source shortest path* (SSSP) problem on $G(P)$, i.e., given P and a source point $s \in P$, compute shortest paths in $G(P)$ from s to all other points of P. In particular, we consider the L_1 case of the problem in which the distance is measured under the L_1 metric (and each disk becomes a diamond).

The L_2 case of the problem where the distance is measured under the L_2 metric has been extensively studied [3,5,8,9,15,16]. The current best algorithm, which was given by Wang and Xue [16], runs in $O(n \log^2 n)$ time. The L_1 case, however, has not been particularly studied before. To solve the L_1 problem, we follow the algorithmic framework of Wang and Xue [16] but give a faster implementation. The runtime of Wang and Xue's algorithm [16] is dominated by a bottleneck subproblem. Due to some special properties of the L_1 metric, we derive a more efficient algorithm for the bottleneck subproblem in L_1 case, which leads to an overall $O(n \log n)$-time algorithm for the shortest path problem.

More specifically, the bottleneck subproblem is the offline insertion-only additively-weighted nearest-neighbor problem, where we are given an offline sequence of k insertions and queries such that an *insertion* inserts a weighted point to a point set U (which is $\emptyset$ initially) and a *query* asks for the additively-weighted nearest neighbor in U of a query point. The goal is to answer all queries. Wang and Xue [16] solved the problem in $O(k \log^2 k)$ time by using the standard logarithmic method [1,2]. This leads to the overall $O(n \log^2 n)$ time for their shortest path algorithm [16]; reducing the time for the subproblem to $O(k \log k)$ would solve the shortest path problem in $O(n \log n)$ time. The difficulty in doing so is that there does not exist a semi-dynamic (for insertions only) weighted Voronoi diagram data structure that can perform each insertion in $O(\log k)$ amortized time (in order to answer queries, an efficient dynamic point location data structure is also needed). For solving our L_1 shortest path problem, we first observe that in the bottleneck subproblem U and V are separated by an axis-parallel line ℓ, where V is the set of all query points. Without loss of generality, we assume that ℓ is horizontal and U is below ℓ. Based on the properties of the L_1 metric, a critical observation we find is that the portion of the weighted L_1 Voronoi diagram of U above ℓ only consists of a set of vertical lines. Then, we can easily maintain these vertical lines by a balanced binary search tree so that each query can be answered in $O(\log k)$ time. Further, the special structure also allows us to update the portion of the Voronoi diagram above ℓ in $O(\log k)$ amortized time for each insertion. As such, the bottleneck subproblem can be solved in $O(k \log k)$ time in the L_1 case, which leads to an overall $O(n \log n)$ time algorithm for the shortest path problem. Note that the space of our shortest path algorithm is $O(n)$.

Cabello and Jejčič [3] observed that by a simple reduction from the max-gap problem, deciding whether the unit-disk graph $G(P)$ is connected requires $\Omega(n \log n)$ time even if all points of P are on a line. This implies that $\Omega(n \log n)$ is a lower bound for solving the shortest path problem in unit-disk graphs for both the L_1 and L_2 cases (because both cases are the same when all points

[*]This research was supported in part by NSF under Grant CCF-2005323.

[†]Department of Computer Science, Utah State University, Logan, UT 84322, USA. haitao.wang@usu.edu

[‡]Corresponding author. Department of Computer Science, Utah State University, Logan, UT 84322, USA. yiming.zhao@usu.edu

of P are on a line). As such, our algorithm for the L_1 case is optimal.

1.1 Related work

Before Wang and Xue's work [16], the shortest path problem in the L_2 case had been studied by many others. Roditty and Segal [15] gave the first sub-quadratic algorithm of $O(n^{4/3+\epsilon})$ time for any constant $\epsilon > 0$. Cabello and Jejčič [3] later proposed an improved algorithm of $O(n^{1+\epsilon})$ time. Following the framework of Cabello and Jejčič [3] but with a more efficient data structure for the bichromatic closest pair problem, Kaplan et al. [9] gave a randomized algorithm that solves the problem in $O(n \log^{12+o(1)} n)$ expected time. Approximation algorithms for the problem have also been developed, e.g., see [5,8,16]

The shortest path problem we consider is actually on a *weighted* unit-disk graph. In the *unweighted* case, the weight of each edge of the graph is 1. The unweighted problem is much easier. The L_2 unweighted problem can be solved in $O(n \log n)$ time [3,5]. In particular, if all input points of P are presorted by their x- and y-coordinates, the algorithm of Chan and Skrepetos [4] runs in $O(n)$ time.

As an important class of geometric intersection graphs, unit-disk graphs have been widely studied due to many of their applications, e.g., in wireless sensor networks [13,14]. In addition to the shortest path problem, many other problems on unit-disk graphs have also been considered in the literature, such as the clique problem [6], the independent set problem [12], all pairs of shortest paths [4,5,8], the diameter problem [4,5,8], etc. Comparing to general graphs, these problems in unit-disk graphs can be solved more efficiently by exploiting their underlying geometric structures.

Outline. In the following, we describe the main algorithm in Section 2 while the bottleneck subproblem is tackled in Section 3.

2 The main algorithm

In this section, we describe the main algorithm for the shortest path problem. Our algorithm follows Wang and Xue's algorithmic framework [16]. In the following, we will adapt their algorithm to the L_1 case. We will also borrow some of their notation.

For any two points p and q in the plane, we use $d(p,q)$ to denote their L_1 distance. For any point p, we use $\odot_p$ to denote the unit disk centered at p, which is a diamond in the L_1 metric. Let s be the source point of P. Throughout the paper, we will use the points of P and the vertices of the unit-disk graph $G(P)$ interchangeably.

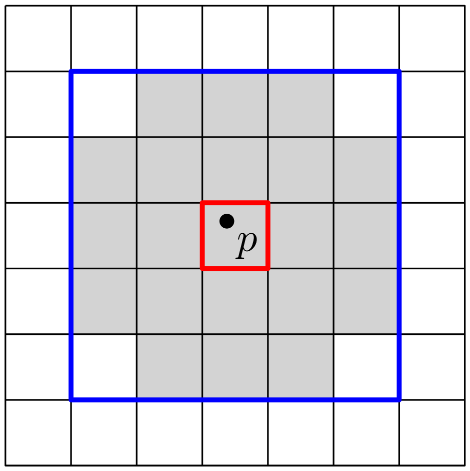


Figure 1: The side length of each square cell in the grid Γ is $\frac{1}{2}$. For the black point p, the red cell that contains it is $\square_p$, and the square area bounded by blue segments which contains 5×5 cells is the patch $\boxplus_p$. For any point in $\square_p$, its neighboring points in $G(P)$ must lie in the grey region.

The algorithm follows the basic idea of Dijkstra's shortest path algorithm with the help of a grid. At the outset, we implicitly build a grid Γ of square cells of side length 1/2. For simplicity of discussion, we assume that each vertex of $G(P)$ lies in the interior of a single cell of Γ. A *patch* of Γ is a square area consisting of 5×5 cells of Γ. For any point p in the plane, let $\square_p$ denote the cell of Γ that contains p and $\boxplus_p$ denote the patch whose central cell is $\square_p$ (e.g., see Fig. 1). Since the side length of each cell of Γ is 1/2, if two vertices of $G(P)$ are in a single cell of Γ, they must be connected by an edge in $G(P)$. On the other hand, if two points p and q are connected by an edge in $G(P)$, then q must be in a cell of $\boxplus_p$. Unlike Dijkstra's shortest path algorithm, which selects one single vertex in each iteration to compute shortest-path information, our algorithm tries to compute shortest-path information for all vertices in a cell of Γ and then pass shortest-path information to the vertices in the neighboring cells.

For a subset $Q \subseteq P$ and a cell $\square$ (resp., a patch $\boxplus$) of Γ, define $Q_\square = Q \cap \square$ (resp., $Q_\boxplus = Q \cap \boxplus$).

To implicitly compute the grid Γ, we actually perform the following preprocessing. We compute $P_\square$ for all cells $\square$ of Γ that contain at least one point of P. We also associate pointers to each point $p \in P$ such that from p we can access $\square_p$ and $\boxplus_p$. The preprocessing can be done in $O(n \log n)$ time and $O(n)$ space [16].

The algorithm will compute a table $dist[\cdot]$ for all vertices of $G(P)$, where $dist[p]$ is the length of a shortest path between s and a point $p \in P$. Note that we should also maintain the corresponding path-predecessor information to form a shortest path tree; this can be done by standard techniques [16], so we omit the discussions.

One important subroutine that will be extensively used in the algorithm is UPDATE(U, V). For two subsets $U, V \subseteq P$, UPDATE(U, V) is to update the shortest-path information of vertices in the set V by using the shortest-path information of vertices in U. More specifically, for each $v \in V$, let $q_v = \arg\min_{u \in U \cap \bigodot_v}\{dist[u] + d(u, v)\}$. The purpose of UPDATE$(U, V)$ is to find q_v for all $v \in V$ and update $dist[v] = \min\{dist[v], dist[q_v] + d(q_v, v)\}$.

With UPDATE(U, V), the algorithm works as follows (refer to Algorithm 1 for the pseudocode). Initially, for each vertex $p \in P$, $dist[p]$ is set to ∞, except that $dist[s] = 0$. Initialize $Q = P$. In the main loop, as long as $Q \neq \emptyset$, in each iteration we find a vertex $q \in Q$ who has a minimum $dist[q]$. Subsequently there are two subroutines UPDATE$(Q_{\boxplus_q}, Q_{\square_q})$ and UPDATE$(Q_{\square_q}, Q_{\boxplus_q})$. Finally, vertices in $Q_{\square_q}$ are removed from Q, because $dist[p]$ for all $p \in Q_{\square_q}$ have been correctly computed. Refer to [16] for the correctness proof, which is applicable to the L_1 case.

Algorithm 1: The SSSP Algorithm [16]

```
Function SSSP(P, s):
    for each $p \in P$ do
        $dist[p] = \infty$
    end
    $dist[s] = 0$
    $Q = P$
    while $Q \neq \emptyset$ do
        $q = \arg\min_{p \in Q}\{dist[p]\}$
        UPDATE($Q_{\boxplus_q}, Q_{\square_q}$) // first update
        UPDATE($Q_{\square_q}, Q_{\boxplus_q}$) // second update
        $Q = Q \setminus Q_{\square_q}$
    end
    return $dist[\cdot]$
end
```

Implementing the algorithm efficiently hinges on the two UPDATE procedures.

The first update. For the first update UPDATE$(Q_{\boxplus_q}, Q_{\square_q})$, the key is to find a point $q_v \in Q_{\boxplus_q} \cap \bigodot_v$ that minimizes $dist[q_v] + d(q_v, v)$ for each point $v \in Q_{\square_q}$. If we assign each point in $Q_{\boxplus_q}$ a weight equal to its $dist$-value, then q_v is essentially the additively-weighted nearest neighbor of v in $Q_{\boxplus_q} \cap \bigodot_v$. To find q_v efficiently, a crucial observation found by Wang and Xue [16] (see Lemma 2.5 in [16], whose proof is applicable to the L_1 case) is that any point $p \in Q_{\boxplus_q}$ that minimizes $dist[p] + d(p, v)$ must be in $\bigodot_v$, i.e., the nearest neighbor of v in $Q_{\boxplus_q}$ is also the nearest neighbor of v in $Q_{\boxplus_q} \cap \bigodot_v$. Due to this observation, we can find q_v for all $v \in Q_{\square_q}$ as follows. First, we build an L_1 additively-weighted Voronoi diagram on vertices in $Q_{\boxplus_q}$ and then using the diagram to find the nearest neighbor for each $v \in Q_{\square_q}$. Constructing the diagram can be done in $O(|Q_{\boxplus_q}| \log |Q_{\boxplus_q}|)$ time and $O(|Q_{\boxplus_q}|)$ space (e.g., by using the abstract Voronoi diagram algorithm [11]), and all queries together take $O(|Q_{\square_q}| \log |Q_{\boxplus_q}|)$ time (e.g., build a point location data structure on the diagram in $O(|Q_{\boxplus_q}|)$ time [7, 10] and then perform point location queries for points of $Q_{\square_q}$, which take $O(\log |Q_{\boxplus_q}|)$ time each).

The second update. Implementing the second update UPDATE$(Q_{\square_q}, Q_{\boxplus_q})$ is not that easy anymore because the above crucial observation does not hold. Since $Q_{\boxplus_q}$ has $O(1)$ cells of Γ, it suffices to perform UPDATE$(Q_{\square_q}, Q_{\square})$ for all cells $\square \in \boxplus_q$.

If $\square$ is $\square_q$, then $Q_{\square_q} = Q_{\square}$. Since the distance between any two points in $\square_q$ is at most 1, we can use the following algorithm to implement UPDATE$(Q_{\square_q}, Q_{\square})$. We first build an L_1 weighted Voronoi diagram on points of $Q_{\square_q}$ in $O(|Q_{\square_q}| \log |Q_{\square_q}|)$ time and $O(|Q_{\square_q}|)$ space [11], and then use it to find the weighted nearest neighbor q_v for each point $v \in Q_{\square_q}$. Clearly, the total time is $O(|Q_{\square_q}| \log |Q_{\square_q}|)$.

If $\square$ is not $\square_q$, then a critical property is that $\square$ and $\square_q$ are separated by an axis-parallel line ℓ. To perform UPDATE$(Q_{\square_q}, Q_{\square})$, Wang and Xue [16] proposed the following approach (see Algorithm 2 for the pseudocode). Let $U = Q_{\square_q}$ and $V = Q_{\square}$. We first sort vertices in $U = \{u_1, u_2, ..., u_{|U|}\}$ by their $dist$-values such that $dist[u_1] \leq dist[u_2] \leq ... \leq dist[u_{|U|}]$. Then we partition V into subsets $V_i = \{v \in V \mid v \in \bigodot_{u_i}, v \notin \bigodot_{u_j} \text{ for all } j < i\}$, for all $i = 1, 2, \ldots, |U|$. For each $1 \leq i \leq |U|$, for each vertex $v \in V_i$, we find $q_v = \arg\min_{p \in U_i}\{dist[p] + d(p, v)\}$, where $U_i = \{u_i, u_{i+1}, \ldots, u_{|U|}\}$, and update $dist[v] = \min\{dist[v], dist[q_v] + d(q_v, v)\}$. This step is implemented by a for loop (Lines 6–13) in Algorithm 2. By the definition of V_i, we have $U \cap \bigodot_v \subseteq U_i$ for all $v \in V_i$. Also, Wang and Xue [16] proved that q_v found as above must be in $\bigodot_v$ (see Lemma 2.6 in [16], whose proof is applicable to the L_1 case). As such, $q_v = \arg\min_{p \in U \cap \bigodot_v}\{dist[p] + d(p, v)\}$. This proves the correctness of the algorithm.

We now analyze the runtime of the above algorithm. Sorting the vertices of U takes $O(|U| \log |U|)$ time. To compute the subsets V_i, $1 \leq i \leq |U|$, Wang and Xue [16] gave an algorithm of $O(k \log k)$ time (and $O(k)$ space) for the L_2 case (see Section 2.2.1 [16]) by making use of the property that U and V are separated by ℓ, where $k = |U| + |V|$. For the L_1 case, we can use the same algorithm; in fact, the algorithm becomes easier as a disk in the L_1 case is a diamond. We omit the details and conclude that the subsets V_i, $1 \leq i \leq |U|$, can be computed in $O(k \log k)$ time in the L_1 case. Next, the for loop

Algorithm 2: UPDATE(U, V) [16]

```
Function Update(U, V):
    Sort($U = \{u_1, u_2, ..., u_{|U|}\}$) // dist[$u_1$] $\leq$ ... $\leq$ dist[$u_{|U|}$]
    for $i = 1, 2, ..., |U|$ do
        $V_i = \{v \in V \mid v \in \bigodot_{u_i}, v \notin \bigodot_{u_j}$ for all $j < i\}$
    end
    $U' = \emptyset$
    for $i = |U|, |U| - 1, ..., 1$ do
        $U' = U' \cup \{u_i\}$
        for each $v \in V_i$ do
            $q_v = \arg\min_{u \in U'}\{dist[u] + d(u, v)\}$
            $dist[v] = \min\{dist[v], dist[q_v] + d(q_v, v)\}$
        end
    end
end
```

(Lines 6–13) is for the bottleneck subproblem mentioned in Section 1, i.e., the offline insertion-only additively-weighted nearest-neighbor problem. Indeed, if we assign each vertex in U a weight equal to its *dist*-value, then q_v is essentially the additively-weighted nearest neighbor of v in U', where $U' = U_i$ in the i-th iteration of the for loop. The set U' is dynamically changed with point insertions. Using the standard logarithmic method [1,2], Wang and Xue [16] solves the problem in $O(k \log^2 k)$ time. By exploring the properties of the L_1 metric, we give an $O(k \log k)$ time (and $O(k)$ space) algorithm in Section 3. As such, UPDATE($Q_{\square_q}, Q_{\square}$) can be performed in $O(k \log k)$ time and $O(k)$ space, with $k = |Q_{\square_q}| + |Q_{\square}|$.

In summary, since $Q_{\boxplus_q}$ has $O(1)$ cells, the second update UPDATE($Q_{\square_q}, Q_{\boxplus_q}$) can be implemented in $O(|Q_{\boxplus_q}| \log |Q_{\boxplus_q}|)$ time as $Q_{\square_q} \subseteq Q_{\boxplus_q}$. This leads to the following theorem.

Theorem 1 *Given a set P of n points in the L_1 plane and a source point $s \in P$, the shortest paths from s to all vertices in the unit-disk graph $G(P)$ can be computed in $O(n \log n)$ time and $O(n)$ space.*

Proof. As discussed before, constructing the grid Γ implicitly can be done in $O(n \log n)$ time and $O(n)$ space [16]. We have shown that both UPDATE procedures can be implemented in $O(|Q_{\boxplus_q}| \log |Q_{\boxplus_q}|)$ time and $O(|Q_{\boxplus_q}|)$ space. As such, each iteration of the while loop of Algorithm 1 can be implemented in $O(|Q_{\boxplus_q}| \log |Q_{\boxplus_q}|)$ time and $O(|Q_{\boxplus_q}|)$ space. As $\sum_{q \in Q} |Q_{\boxplus_q}| \leq 25n$, the total time of the algorithm is $O(n \log n)$. Note that the overall time of Line 8 and Line 11 of Algorithm 1 can be easily bounded by $O(n \log n)$ by using a balanced binary search tree. The total space of the algorithm is $O(n)$. $\square$

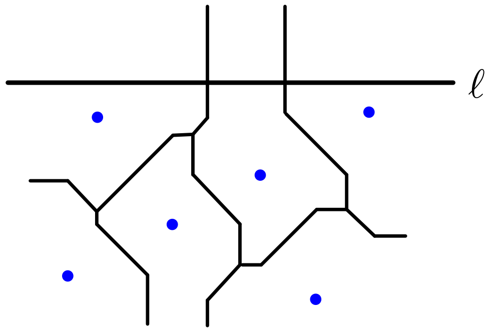


Figure 2: Illustrating $VD(U')$, where U' has six blue points (with the same weight). $VD_h(U')$ consists of two vertical half-lines.

3 The bottleneck subproblem

In this section, we present an $O(k \log k)$ time and $O(k)$ space algorithm to solve the bottleneck subproblem on U and V, with $k = |U| + |V|$. Recall U and V are separated by an axis-parallel line ℓ. Without loss of generality, we assume that ℓ is horizontal such that U is below ℓ and V is above ℓ. Our goal is to find $q_v \in U'$ for all $v \in V_i$ (i.e., Line 10 in Algorithm 2), for a subset $U' \subseteq U$.

In the following, we first discuss some observations about the geometric structure of the problem and then describe the algorithm.

3.1 Observations

Let $VD(U')$ denote the weighted Voronoi diagram of U'. To find q_v, it suffices to locate the cell of $VD(U')$ that contains v. Let h denote the upper half-plane bounded by ℓ. As v is above ℓ, it suffices to maintain the portion of $VD(U')$ above ℓ, denoted by $VD_h(U')$. In what follows, we first show that $VD_h(U')$ has a very simple structure: it only consists of a set of vertical half-lines with endpoints on ℓ and going upwards to the infinity (e.g., see Fig. 2). Then, we will show that $VD_h(U')$ can be updated in $O(\log k)$ amortized time for each insertion (i.e., inserting a point into U').

We say a vertical half-line is *grounded* on ℓ if it goes upwards to the infinity and has its endpoint on ℓ. For any point or a vertical line segment p in the plane, we use $x(p)$ to denote its x-coordinate. For each point $u \in U$, we define its weight $w(u) = dist[u]$.

Properties of bisectors of two weighted points. Consider two weighted points a and b in the plane with nonnegative weights $w(a)$ and $w(b)$, respectively. The *bisector* $B(a, b)$ of a and b is the locus of points with equal (additively-)weighted distance to a and b, i.e., $B(a,b) = \{p \in \mathbb{R}^2 \mid w(a) + d(a,p) = w(b) + d(b,p)\}$ (e.g., see Fig. 3). Note that in the degenerate case it is possible that an entire quadrant of the plane is in $B(a,b)$ (e.g., see Fig. 3b), in which case we only consider the vertical boundary of the quadrant to be in $B(a,b)$. Hence, $B(a,b)$ in general consists of three parts: two axis-parallel half-lines with a segment in the middle. Suppose both a and b are below the line ℓ and $x(a) \le x(b)$. Define $B_h(a,b) = B(a,b) \cap h$. Then either $B_h(a,b) = \emptyset$ or $B_h(a,b) \cap h$ is a vertical half-line grounded on ℓ; in the latter case $x(a) \le x(B_h(a,b)) \le x(b)$. Note that if $x(a) = x(b)$, then $B(a,b)$ is a horizontal line between a and b and thus $B_h(a,b) = \emptyset$.

Geometric structure of $VD_h(U')$. Since all points of U are below ℓ, according to the discussion above, for any two points u_i and u_j of U, $B_h(u_i,u_j)$ is either $\emptyset$ or a vertical half-line grounded on ℓ (and the vertical half-line is between u_i and u_j). These properties guarantee that $VD_h(U')$ consists of a set of $O(|U'|)$ vertical half-lines grounded on ℓ (e.g., see Fig. 2), and between each pair of adjacent half-lines is the portion of the Voronoi cell of a vertex $u \in U'$. As such, we can use a balanced binary search tree $T(U')$ to store the x-coordinates of the vertical half-lines of $VD_h(U')$. Given a query point $v \in V$, we can use $T(U')$ to find the cell of $VD_h(U')$ containing v and thus obtain q_v in $O(\log |U'|)$ time, which is $O(\log k)$ as $|U'| \le |U| \le k$. In the following, we will discuss how to update $VD_h(U')$ after a point of U is inserted to U'. We first prove some properties about the geometric structure of $VD_h(U')$.

For each point $u \in U'$, let $R(u)$ denote the Voronoi cell of u in $VD(U')$ and let $R_h(u) = R(u) \cap h$. The above shows that if $R_h(u)$ is not empty, then it is bounded by two vertical half-lines from the left and right; let l_u and r_u denote these two half-lines, respectively. We call l_u the *left bounding half-line* and r_u the *right bounding half-line* of $R_h(u)$. Note that if $R_h(u)$ is the leftmost (resp., rightmost) cell of $VD_h(U')$, then we let l_u (resp., r_u) refer to the vertical half-line grounded on ℓ with x-coordinate $-\infty$ (resp., $+\infty$).

We say that a point $u \in U'$ is *relevant* if $R_h(u) \neq \emptyset$ and *irrelevant* otherwise. The following lemma proves several properties about the geometric structure of $VD_h(U')$, which will be useful for processing insertions.

Lemma 2 *Suppose $u^1, u^2, \ldots, u^t$ is the list of relevant vertices of U' whose Voronoi cells intersect h in the order from left to right. Then, the followings hold.*

1. *$x(u^1) < x(u^2) < \cdots < x(u^t)$.*

2. *For each $1 \le i < t$, r_{u^i} is $l_{u^{i+1}}$.*

3. *For each $1 \le i \le t$, $x(l_{u^i}) \le x(u^i) \le x(r_{u^i})$.*

4. *For each $1 \le i \le t$, p^i is in $R_h(u^i)$, where p^i is the vertical projection of u^i on ℓ.*

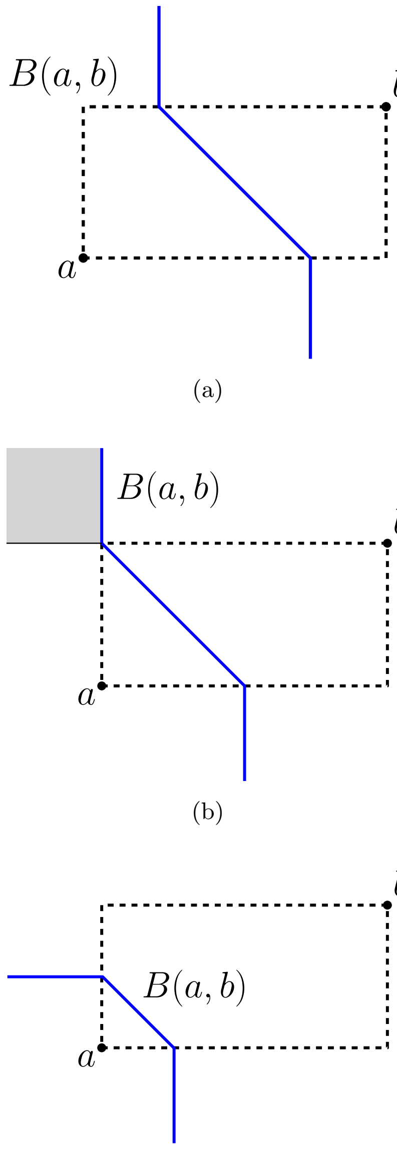


Figure 3: Possible cases for the bisector $B(a,b)$ of two weighted points a and b.

Proof. Consider a point u^i for any $i > 1$. By the definition of the list $u^1, u^2, \ldots, u^t$, l_{u^i} belongs to the bisector $B(u^{i-1}, u^i)$ of u^{i-1} and u^i, i.e., $l_{u^i} = B_h(u^{i-1}, u^i)$. According to the properties of bisectors, $x(u_{i-1}) \le x(l_{u^i}) \le x(u^i)$. Note that $x(u^{i-1}) = x(u^i)$ is not possible since otherwise $B_h(u^{i-1}, u^i)$ would be $\emptyset$ (contradicting with $l_{u^i} = B_h(u^{i-1}, u^i)$). As such, $x(u^{i-1}) < x(u^i)$ holds. This proves the first lemma statement.

According to our definition of the list $u^1, u^2, \ldots, u^t$, the left bounding half-line of $R_h(u^{i+1})$ must be the right bounding half-line of $R_h(u^i)$. Hence, the second lemma

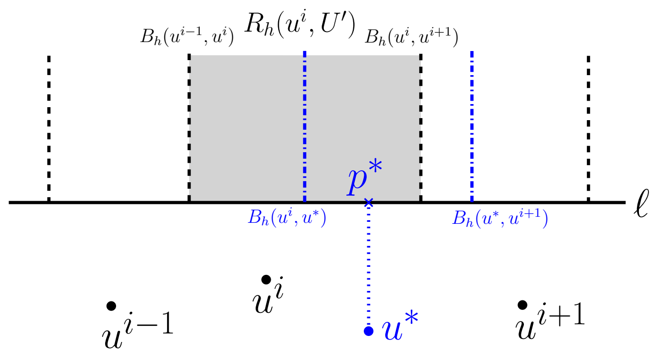


Figure 4: Illustrating $VD_h(U')$, and $VD_h(U'')$ after u^* is inserted. The two dash dotted blue segments are new half-lines in $VD_h(U'')$ while $B_h(u^i,u^{i+1})$ does not appear in $VD_h(U'')$. $R_h(u^i,U')$ is the grey area and $R_h(u^*,U'')$ is the region between the two dash dotted blue segments. Note that $B_h(u^{i-1},u^i)$ is $l_{u^i} = r_{u^{i-1}}$ and $B_h(u^i,u^{i+1})$ is $r_{u^i} = l_{u^{i+1}}$.

statement holds.

The above shows that $x(l_{u^i}) \leq x(u^i)$ for $i > 1$. If $i = 1$, $x(l_{u^i}) \leq x(u^{i-1})$ also holds, for $x(l_{u^i}) = -\infty$. This proves that $x(l_{u^i}) \leq x(u^i)$ for any $1 \leq i \leq t$. By a symmetric analysis, we can show that $x(u^i) \leq x(r_{u^i})$ for any $1 \leq i \leq t$. This proves the third lemma statement.

The fourth lemma statement is an immediate consequence of the third lemma statement. □

3.2 Processing insertions

We are now in a position to describe our algorithm for processing insertions.

Consider inserting a point $u^* \in U \setminus U'$ into U'. As $u^* \in U$, u^* is below ℓ. Let $U'' = U' \cup \{u^*\}$. Our goal is to construct $VD_h(U'')$ by modifying $VD_h(U')$, or more precisely, obtain the tree $T(U'')$ by modifying $T(U')$. For differentiation, for each vertex $u \in U''$, we use $R(u, U'')$ to denote the Voronoi cell of u in $VD(U'')$ and use $R(u, U')$ to denote the Voronoi cell of u in $VD(U')$. We define $R_h(u, U'')$ and $R_h(u, U')$ similarly. Let $u^1, u^2, \ldots, u^t$ be the list of relevant vertices of U' whose Voronoi cells intersect h ordered from left to right.

We first compute the vertical projection of u^* on ℓ and let p^* denote the projection point (e.g., see Fig. 4). Then, using the tree $T(U')$, we find the cell $R_h(u^i, U')$ of $VD_h(U')$ that contains p^*, for some relevant point $u^i \in U'$. For ease of discussion, we assume $1 < i < t$ and other cases can be handled similarly. The following lemma is obtained based on Lemma 2.

Lemma 3 *$R_h(u^*, U'') \neq \emptyset$ if and only if $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$, and if $R_h(u^*, U'') \neq \emptyset$, then $p^* \in R_h(u^*, U'')$.*

Proof. If $R_h(u^*, U'') \neq \emptyset$, then by Lemma 2, p^* must be in $R_h(u^*, U'')$ and this implies $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$ must hold. On the other hand, suppose $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$. Then, since $p^* \in R_h(u^i, U')$, $d(p^*, u^i) + w(u^i) \leq d(p^*, u) + w(u)$ holds for any vertex $u \in U'$. Therefore, $d(p^*, u) + w(u) \geq d(p^*, u^*) + w(u^*)$ holds for any $u \in U''$. This implies that u^* is the nearest neighbor of p^* in U''. As such, the point p^* must be in $R_h(u^*, U'')$ and $R_h(u^*, U'')$ cannot be empty. □

With Lemma 3, our insertion algorithm proceeds as follows. We check whether $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$. If not, then $R_h(u^*, U'') = \emptyset$ by Lemma 3 and thus $VD_h(U'') = VD_h(U')$; hence, $T(U'') = T(U')$ and we are done with processing the insertion of u^*. In the following, we assume that $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$. By Lemma 3, $R_h(u^*, U'') \neq \emptyset$ and thus $VD_h(U'') \neq VD_h(U')$. Below we discuss how to modify $VD_h(U')$ to obtain $VD_h(U'')$.

For each vertex $u \in U'$, we still use l_u and r_u to denote the left and right bounding vertical half-lines of $R_h(u, U')$, respectively.

Since $p^* \in R_h(u^i, U')$, we have $x(u^*) = x(p^*) \in [x(l_{u^i}), x(r_{u^i})]$. By Lemma 2, $x(u^{i-1}) \leq x(r_{u^{i-1}}) = x(l_{u^i})$ and $x(r_{u^i}) = x(l_{u^{i+1}}) \leq x(u^{i+1})$. Therefore, $x(p^*) \in [x(u^{i-1}), x(u^{i+1})]$. Also by Lemma 2, $x(u^{i-1}) < x(u^i) < x(u^{i+1})$. Without loss of generality, we assume that $x(u^i) \leq x(p^*) < x(u^{i+1})$. We first discuss how to obtain the portion of $VD_h(U'')$ to the left of p^*. To this end, we consider the points $u^i, u^{i-1}, \ldots, u^1$ in this order.

First, for u^i, we compute the bisector $B(u^i, u^*)$ of u^i and u^*. Depending on whether $B_h(u^i, u^*) = B(u^i, u^*) \cap h$ is $\emptyset$, there are two cases.

- If $B_h(u^i, u^*) \neq \emptyset$, then $B_h(u^i, u^*)$ is a vertical half-line grounded on ℓ. Since $x(u^i) \leq x(u^*)$, according to the properties of bisectors, $x(u^i) \leq x(B_h(u^i, u^*)) \leq x(u^*)$. As $x(l_{u^i}) \leq x(u^i)$ and $x(u^*) \leq x(r_{u^i})$, $B_h(u^i, u^*)$ must be in the Voronoi cell $R_h(u^i, U')$ between l_{u^i} and p^* (e.g., see Fig. 4). Hence, $B_h(u^i, u^*)$ must be the right bounding half-line of the cell $R_h(u^i, U'')$ in $VD_h(U'')$ as well as the left bounding half-line of the cell $R_h(u^*, U'')$. We update the tree $T(U')$ accordingly (i.e., insert $B_h(u^i, u^*)$ to $T(U')$) and then halt the algorithm (i.e., the construction of $VD_h(U'')$ on the left of p^* is finished).

- If $B_h(u^i, u^*) = \emptyset$, then by our definition of bisectors (including our way for handling the degenerating case), since $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$, $d(p, u^i) + w(u^i) \geq d(p, u^*) + w(u^*)$ holds for any point $p \in h$. This implies that u^i is *dominated* by u^* with respect to the points of h, and thus

u^i becomes irrelevant in $VD_h(U'')$. As such, we remove l_{u^i} from $T(U')$. Note that l_{u^i} is $r_{u^{i-1}}$ by Lemma 3.

Next, we consider u^{i-1} in a way similar to the above for u^i. If $B_h(u^{i-1}, u^*) \neq \emptyset$, then $B_h(u^{i-1}, u^*)$ becomes the right bounding half-line of the cell $R_h(u^{i-1}, U'')$ in $VD_h(U'')$ as well as the left bounding half-line of $R_h(u^*, U'')$. We insert $B_h(u^{i-1}, u^*)$ into $T(U')$ and halt the algorithm. If $B_h(u^{i-1}, u^*) = \emptyset$, then since $p^* \in R_h(u^*, U'')$ by Lemma 3, $d(p^*, u^{i-1}) + w(u^{i-1}) \geq d(p^*, u^*) + w(u^*)$. Further, by our definition of bisectors (including our way for handling the degenerating case), $d(p, u^{i-1}) + w(u^{i-1}) \geq d(p, u^*) + w(u^*)$ holds for any point $p \in h$. Therefore, as above, u^{i-1} becomes irrelevant in $VD_h(U'')$. Accordingly, we remove $l_{u^{i-1}}$ from $T(U')$. We then proceed to considering u^{i-2} in the same way as above.

The above describes the algorithm for constructing $VD_h(U'')$ to the left of p^*. The algorithm for constructing $VD_h(U'')$ to the right of p^* is similar. One slight difference is that the algorithm starts with considering u^{i+1} instead of u^i by first removing r_{u^i} from $T(U')$. Then, we compute the bisector $B(u^*, u^{i+1})$. If $B_h(u^*, u^{i+1}) \neq \emptyset$, then $B_h(u^*, u^{i+1})$ becomes the right bounding half-line of $R_h(u^*, U'')$ as well as the left bounding half-line of $R_h(u^{i+1}, U'')$. We insert $B_h(u^*, u^{i+1})$ into $T(U')$ and halt the algorithm. If $B_h(u^*, u^{i+1}) = \emptyset$, then u^{i+1} becomes irrelevant and we proceed to considering u^{i+2} in the same way.

The above describes the algorithm for constructing $VD_h(U'')$ from $VD_h(U')$. The resulting tree $T(U')$ is $T(U'')$. The following lemma summarizes the time complexity of the insertion algorithm described above and proves the correctness of the algorithm.

Lemma 4 *After a point $u^* \in U$ is inserted into U', $VD_h(U'')$ can be computed from $VD_h(U')$ in $O((\delta + 1)\log k)$ time, where $U'' = U' \cup \{u^*\}$ and δ is the number of relevant vertices of $VD_h(U')$ that become irrelevant in $VD_h(U'')$.*

Proof. The runtime of the insertion algorithm is obvious from our algorithm description. In the following, we prove the correctness of the algorithm.

If $d(p^*, u^i) + w(u^i) < d(p^*, u^*) + w(u^*)$, then $VD_h(U'') = VD_h(U')$ by Lemma 3 and thus our algorithm is correct in this case. In the following, we assume that $d(p^*, u^i) + w(u^i) \geq d(p^*, u^*) + w(u^*)$ and prove that the diagram $VD_h(U'')$ constructed by our algorithm is correct.

Let p be any point in h and let u be the point of U'' such that p is in the cell of u after our insertion algorithm for u^* is finished, i.e., $p \in R_h(u, U'')$. To prove the correctness of our algorithm, it suffices to show

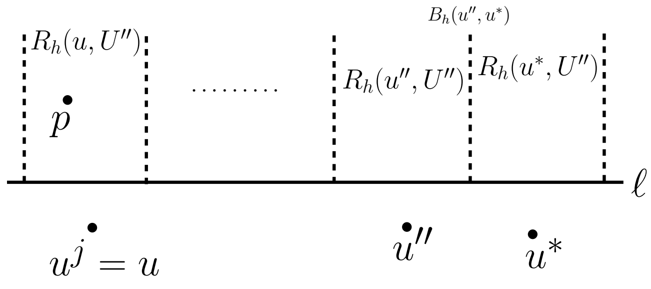


Figure 5: Illustrating the proof of Lemma 4 for the case where u is not adjacent to u^* in L.

that $d(p, u) + w(u) \leq d(p, u') + w(u')$ holds for every point $u' \in U''$. Depending on whether $u = u^*$, there are two cases. Let u^j be the point of U' such that $p \in R_h(u^j, U')$.

- We first consider the case $u = u^*$. As $p \in R_h(u^j, U')$, $d(p, u^j) + w(u^j) \leq d(p, u') + w(u')$ holds for any $u' \in U'$. As p is in the cell of u^* after the insertion algorithm finishes, according to our algorithm, $d(p, u^*) + w(u^*) \leq d(p, u^j) + w(u^j)$ must hold. Since $u = u^*$, we obtain that $d(p, u) + w(u) = d(p, u^*) + w(u^*) \leq d(p, u^j) + w(u^j) \leq d(p, u') + w(u')$ holds for any $u' \in U''$.

- We then consider the case $u \neq u^*$. In this case, according to our algorithm, u must be u^j and u and u^* define different cells in $VD_h(U'')$, i.e., $R_h(u, U'') \neq R_h(u^*, U'')$. Without loss of generality, we assume that $R_h(u, U'')$ is to the left of $R_h(u^*, U'')$. Depending on whether u is adjacent to u^* in the relevant point list L after the insertion algorithm (L is defined in the same way as Lemma 2 with respect to $VD_h(U'')$), there are two subcases.

  If u is adjacent to u^* in L, then since p is in the cell of u after the insertion algorithm, it holds that $d(p, u) + w(u) \leq d(p, u^*) + w(u^*)$. Since $u = u^j$ and $d(p, u^j) + w(u^j) \leq d(p, u') + w(u')$ holds for any $u' \in U'$, we obtain that $d(p, u) + w(u) \leq d(p, u') + w(u')$ holds for any $u' \in U''$.

  If u is not adjacent to u^* in L, then let u'' be the left neighboring relevant point of u^* in L (e.g., see Fig 5). Since $R_h(u, U'')$ is to the left of $R_h(u^*, U'')$ and $p \in R_h(u, U'')$, p must be to the left of $B_h(u'', u^*)$, which is the right bounding half-line of $R_h(u'', U'')$. As u'' is the left neighboring relevant point of u^* in L, according to our insertion algorithm, $d(p', u'') + w(u'') \leq d(p', u^*) + w(u^*)$ for any point $p' \in h$ to the left of $B_h(u'', u^*)$. Because p is in h to the left of $B_h(u'', u^*)$, $d(p, u'') + w(u'') \leq d(p, u^*) + w(u^*)$ holds. As $d(p, u^j) + w(u^j) \leq d(p, u') + w(u')$ for any $u' \in U'$, we have $d(p, u^j) + w(u^j) \leq d(p, u'') + w(u'')$. We thus derive $d(p, u^j) + w(u^j) \leq d(p, u^*) + w(u^*)$. Since $u = u^j$, we obtain that $d(p,u)+w(u) \leq d(p,u')+w(u')$ for any $u' \in U''$.

In summary, $d(p,u)+w(u) \leq d(p,u')+w(u')$ holds for every point $u' \in U''$. This proves the correctness of our algorithm. □

Note that once a relevant point becomes irrelevant after an insertion, it will never become relevant again for any insertions in future. Therefore, the total sum of δ in Lemma 4 for processing all insertions of U is at most k. As such, by Lemma 4, the total time for processing all insertions is $O(k \log k)$.

Recall that all query operations can be performed in overall $O(k \log k)$ time by using the tree $T(U')$. Note that the space of our algorithm is bounded by $O(k)$. Therefore, we finally obtain the following result.

Lemma 5 *The bottleneck subproblem on U and V can be solved in $O(k \log k)$ time and $O(k)$ space, where $k = |U| + |V|$.*

References

[1] J.L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8:244–251, 1979.

[2] M. de Berg, K. Buchin, B.M.P. Jansen, and G. Woeginger. Fine-grained complexity analysis of two classic TSP variants. *ACM Transactions on Algorithms*, 17(1):5:1–5:29, 2021.

[3] S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry: Theory and Applications*, 48(4):360–367, 2015.

[4] T.M. Chan and D. Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

[5] T.M. Chan and D. Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 24:1–24:13, 2018.

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

[7] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[8] J. Gao and L. Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

[9] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2495–2504, 2017.

[10] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[11] R. Klein. Concrete and abstract Voronoi diagrams. volume 400 of *Lecture Notes in Computer Science*, Springer-Verlag, 1989.

[12] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Japanese Conference on Discrete and Computational Geometry*, pages 194–200, 1998.

[13] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, pages 234–244, 1994.

[14] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, 1999.

[15] L. Roditty and M. Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

[16] H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020.

Total Domination in Geometric Unit Disk Graphs

Sangram K. Jena[*] Gautam K. Das[†]

Abstract

Let $G = (V, E)$ be an undirected graph. We call $D_t \subseteq V$ a total dominating set (TDS) of G if each vertex $v \in V$ has a dominator in D_t other than itself. Here we consider the TDS problem in geometric unit disk graphs, where the objective is to find a minimum cardinality total dominating set for an input graph. We prove that the problem of deciding whether a geometric unit disk graph (UDG) G has a TDS of cardinality at most k (here k is a positive integer) is NP-complete. Next, we propose an almost linear time 8-factor approximation algorithm in geometric UDGs for the TDS problem. It is an improvement over the best-known 10-factor approximation algorithm with the same running time available in the literature [M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi and D. J. Rosenkrantz, *Simple heuristics for unit disk graphs*, *Networks*, 25(2):59–68, 1995]. We also show that the TDS problem admits a polynomial-time approximation scheme in geometric UDGs.

keywords: Total dominating set, approximation algorithm, PTAS, unit disk graph

1 Introduction

Let us consider a simple undirected graph $G = (V, E)$. The *open neighborhood* (*resp. closed neighborhood*) of a vertex $v \in V$ is the set $N_G(v) = \{u \in V : uv \in E\}$ (resp. $N_G[v] = N_G(v) \cup \{v\}$). A *dominating set* (DS) of G is a subset $D \subseteq V$ such that for each vertex $v \in V$, $|D \cap N_G[v]| \geq 1$. A *total dominating set* (TDS) is a subset $D_t \subseteq V$ of G such that for each vertex $v \in V$, $|D_t \cap N_G(v)| \geq 1$. Therefore, a vertex $v \in D$ (dominating set) dominates all its neighbors and itself whereas a vertex $v \in D_t$ (total dominating set) dominates all its neighbors other than itself. The objective of TDS (resp. DS) problem is to find a minimum size subset $D_t \subseteq V$ (resp. $D \subseteq V$) such that D_t (resp. D) dominates all the vertices in V.

The intersection graph of equal-radii disks with known position in the plane is called a *geometric unit disk graph* (UDG). Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$. Let $d_i = \{p \in \mathbb{R}^2 | d(p_i, p) \leq \frac{1}{2}\}$ be a disk of radius $\frac{1}{2}$ with centered at p_i for $1 \leq i \leq n$, where $d(.,.)$ denotes Euclidean distance between two points in $\mathbb{R}^2$. Let $S = \{d_1, d_2, \ldots, d_n\}$ be the set of n equal-radii circular disks. We define the geometric UDG $G = (V, E)$ corresponding to the point set P as follows: each vertex $v_i \in V$ corresponds to the point $p_i \in P$, and $v_iv_j \in E$ if and only if the disks d_i and d_j intersect i.e., the Euclidean distance between p_i and p_j is at most one unit.

One of the applications of the problem is identifying the location of monitoring devices, such as surveillance cameras or fire alarms, to safeguard a system that can be modeled by total domination in graphs. The problem of placing monitoring devices in a system, where each monitoring device can be placed in such a way that every site in the system (including the monitors themselves) is adjacent to one of the monitoring devices. In this case, placing the monitoring devices in each solution point of the total domination of the system solves the problem.

1.1 Related Work

In 1980, Cockayne et al. [6] introduced the total domination problem and proved that for any connected graph G of $n(\geq 3)$ vertices the cardinality of minimum total dominating set, denoted by λ_t, is less than or equal to $\frac{2}{3}n$ i.e., $\lambda_t \leq \frac{2}{3}n$. Brigham et al. [2] proved that the total domination number is exactly $\frac{2}{3}n$ for the connected graph G of order $n(\geq 3)$, where G is either C_3 (cycle graph of 3 vertices), C_6 or 2-corona of some connected graph. Sun [21] proved the bound to $\lambda_t \leq \left\lfloor \frac{4}{7}(n+1) \right\rfloor$, for connected graphs having order n with minimum degree at least 2. Chvátal and McDiarmid [5] and Tuza [23] independently proved a theorem concerning transversals in hypergraphs, which gives a bound on total domination number. The bound is $\lambda_t \leq \frac{n}{2}$ for the graphs with order n and minimum degree at least 3. For the graphs with minimum degree at least 4, Thomassé and Yeo [22] proposed a result for hypergraphs, which bounds the total domination number by $\lambda_t \leq \frac{3}{7}n$. In [8], DeLaViņa et al. proved that the total domination number of any connected graph is equal to the total domination number of a spanning tree of the same graph. Another interesting aspect of trees concerning total domination is that it is possible to characterize some vertices that are in every minimum total dominating set or not in any total dominating set [7]. Furthermore, Haynes and Henning established three equivalent conditions for a tree to have a unique minimum total dominating set [10]. Chellali and Haynes [4] proved $\lambda_t \geq \frac{n+2-\ell}{2}$ for a

[*]Department of Mathematics, Indian Institute of Technology Guwahati, sangram@iitg.ac.in

[†]Department of Mathematics, Indian Institute of Technology Guwahati, gkd@iitg.ac.in

nontrivial tree of n vertices with ℓ leaves. Dorfling et al. [9] bound the total domination number of planar graphs having different diameters and radius. Pfaff et al. [20] showed that computing λ_t for general graphs is NP-complete. In the same paper, they also showed that calculating λ_t for bipartite graphs remains NP-complete. However, a linear-time algorithm exists for computing λ_t in tree graph [19]. The total domination number in the case of star graphs, complete graphs, binary star graphs, and complete bipartite graphs is 2 [1]. In the same article, they have observed that for cycles and paths, the total dominating set can be found in polynomial time. They have also established a set of relations between (i) λ_t and the maximum degree, and (ii) λ_t and the cut vertices of the graph. See [1, 11, 12, 13, 14] for a detailed survey on the TDS problem.

Dominating set and its many variants such as weighted dominating set, connected dominating set, etc. have been extensively studied in the literature due to their wide range of applications in wireless networks. A wireless network can be modeled as a unit disk graph. Though the (variant) domination problems are still NP-hard for unit disk graphs, unlike in general graphs the problems admit constant approximation algorithms and approximation schemes [25, 26, 29, 30, 31, 24, 27, 28].

1.2 Our Contribution

In this paper, we consider the total dominating set problem in geometric UDGs. In Section 2, we prove that the decision version of the TDS problem is NP-complete in geometric UDGs. We propose an almost linear time 8-factor approximation algorithm for the same problem in Section 3. We also show that the problem admits a PTAS in Section 4. Finally, we conclude the paper in Section 5.

2 NP-Completeness

In this section, we prove that the TDS problem in UDGs belongs to the class NP-complete. In [18], Lichtenstein proved that the planar vertex cover problem is NP-hard by showing a polynomial-time reduction from the planar 3SAT problem to the planar vertex cover problem. In the reduction from an arbitrary instance of the planar 3SAT problem to an instance of planar vertex cover problem, the degree of the graph is at most 3. Therefore the planar vertex cover problem is NP-hard for the graphs with the degree at most 3. We prove the NP-hardness result of the TDS problem in UDGs by showing a polynomial-time reduction from the vertex cover problem in planar graphs of degree at most 3 to it. Now, we define the decision version of the TDS problem in UDGs and vertex cover problem in planar graphs of degree at most 3 as follows:

The TDS problem in UDGs (TDS-UDG)

Given a unit disk graph G and an integer $k(>0)$, does there exist a TDS of size at most k?

The VC problem in planar graphs (VC-PLA)

Given a planar graph G with degree at most 3 and an integer $k(>0)$, does G has a VC of size at most k?

Lemma 1 ([32]) *Let $G=(V,E)$ be a planar graph with maximum degree 3. The graph G can be embedded in linear time on a planar grid of size 4×4 using $O(|V|^2)$ area such that the coordinate of each vertex $v\in V$ is $(4i,4j)$ for some integers i,j and each edge $e\in E$ is a finite sequence of consecutive axis-parallel line segments of length 4 units along the grid lines.*

Lemma 2 *For a given* VC-PLA *instance $G=(V,E)$ with at least one edge, an instance $G'=(V',E')$ of* TDS-UDG *can be constructed in polynomial-time.*

Proof. Let $V=\{v_1,v_2,\ldots,v_n\}$ and $E=\{e_1,e_2,\ldots,e_m\}$. The construction of G' from the graph G is described in four steps as follows.

Step 1: (Embedding) We first embed G into a planar grid of size $4n\times 4n$ using Lemma 1. On the embedding, each of the vertex $v_i\in V$ becomes grid point p_i and each edge $e_j\in E$ become a finite sequence of connected axis-parallel line segment(s) of length four units along the grid lines. Assume that ℓ is the total number of line segments used in the embedding. We call the point p_i corresponding to the vertex $v_i\in V$ $(i=1,2,\ldots,n)$ in the embedding as *vertex points* (see Figure 1(a) and 1(b)). Let N be the set of vertex points. Therefore, $N=\{p_i\mid v_i\in V\}$ and $|N|=|V|(=n)$.

Step 2: (Extra points) In this step, we add some extra points on each of the ℓ line segments (obtained in embedding step) so that unit disks centered on these points and grid points (see embedding step) form a unit disk graph as follows: **(a)** for each edge p_ip_j with only one line segment i.e., length of the edge is 4 units, we add five points at distance 0.98, 1.49, 2, 2.51, 3.02 units from p_i (see edge p_4p_6 in Figure 1(c)), and **(b)** for each edge p_ip_j with more than one segment i.e., length of the edge is greater than 4 units, (i) add a point on each of the grid point on the edge other than the vertex point and name it as grid point (see square points in Figure 1(c)), and (ii) we add four points on each of the line segments connected with p_i and p_j at distances 1, 1.75, 2.5, 3.25 units from p_i and p_j, and for other line segments we add three points at distance 1 units from each other excluding the *grid*

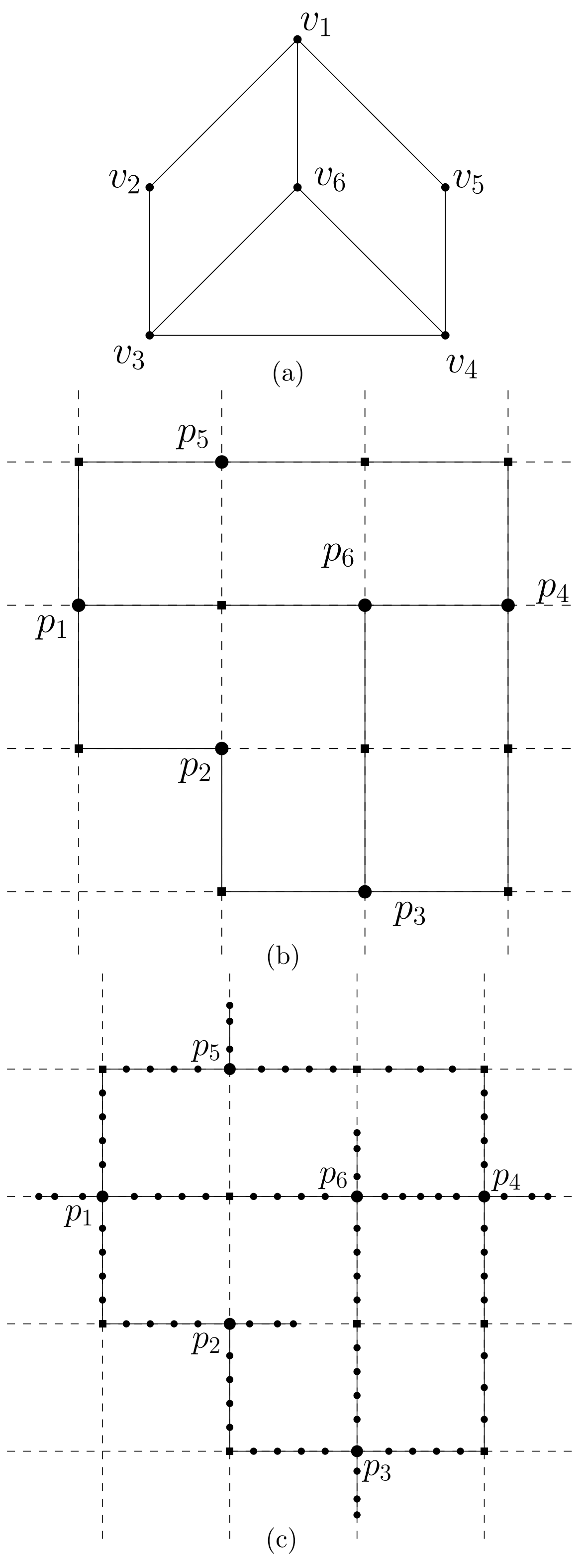


Figure 1: (a) A planar graph G with maximum degree 3, (b) its embedding on a 4×4 grid, and (c) construction of a UDG from the embedding.

points (see the edge p_3p_4 in Figure 1(c)). Let A be the set of all points added in this step. Therefore, $|A| = 4\ell + m$ (here ℓ is the total number of line segments in the embedding and m is the number of edges in G).

Step 3: (Support point) Add a new line segment of length 1.4 units at each of the vertex points p_i without coinciding with the line segments that had already been drawn in the embedding. Observe that the addition of such a line segment is possible without losing the planarity as the maximum degree of G is 3. We add three points x_i, y_i, z_i on each of these line segments at distances 0.3, 1.1, and 1.4 units from the corresponding vertex point p_i. Let S be the set of all points added in this step. Therefore, $|S| = 3n$ (here n is the number of vertices in G).

Step 4: (Construction of UDG) We construct a UDG $G' = (V', E')$, where $V' = N \cup A \cup S$ and $E' = \{u'v' : u', v' \in V'$ and the Euclidean distance between u' and v' is at most 1 unit$\}$ (see Figure 1(c)).

From Lemma 1, $\ell = O(n^2)$. Therefore both $|V'|$ and $|E'|$ are bounded by $O(n^2)$. Hence, G' can be constructed in polynomial-time. $\square$

Theorem 3 Tds-Udg *belongs to the class NP-complete.*

Proof. Let $T \subseteq V$ be an arbitrary subset of vertices and $k(> 0)$ be an integer. Observe that, we can verify whether T is a total dominating set such that $|T| \leq k$ or not in polynomial-time. Therefore, Tds-Udg $\in$ NP.

To prove NP-hardness of Tds-Udg, we will use polynomial time reduction of Vc-Pla to it. We construct an instance $G' = (V', E')$ of Tds-Udg from an arbitrary instance $G = (V, E)$ of Vc-Pla in polynomial time using the steps mentioned in Lemma 2. Next, we prove the following claim to complete the proof of NP-hardness of Tds-Udg.

Claim: *G has a vertex cover C with $|C| \leq k$ if and only if G' has a total dominating set T with $|T| \leq k + 2\ell + 2n$.*

Necessity: Let $C \subseteq V$ be a vertex cover of G such that $|C| \leq k$. Let $N' = \{p_i \in N \mid v_i \in C\}$, i.e., N' is the set of vertices (or vertex points) in G' that correspond to the vertices in C. From each segment, we choose 2 vertices (extra points) from A and corresponding to each vertex point, we choose 2 points (support points) from S, in the embedding. The set of chosen vertices, say $A'(\subseteq A)$, $S'(\subseteq S)$, together with N' will form a TDS of the desired cardinality in G'. We now discuss the process of obtaining the set A'. Initially $A' = \emptyset$. As C is a vertex cover, every edge in G has at least one of its

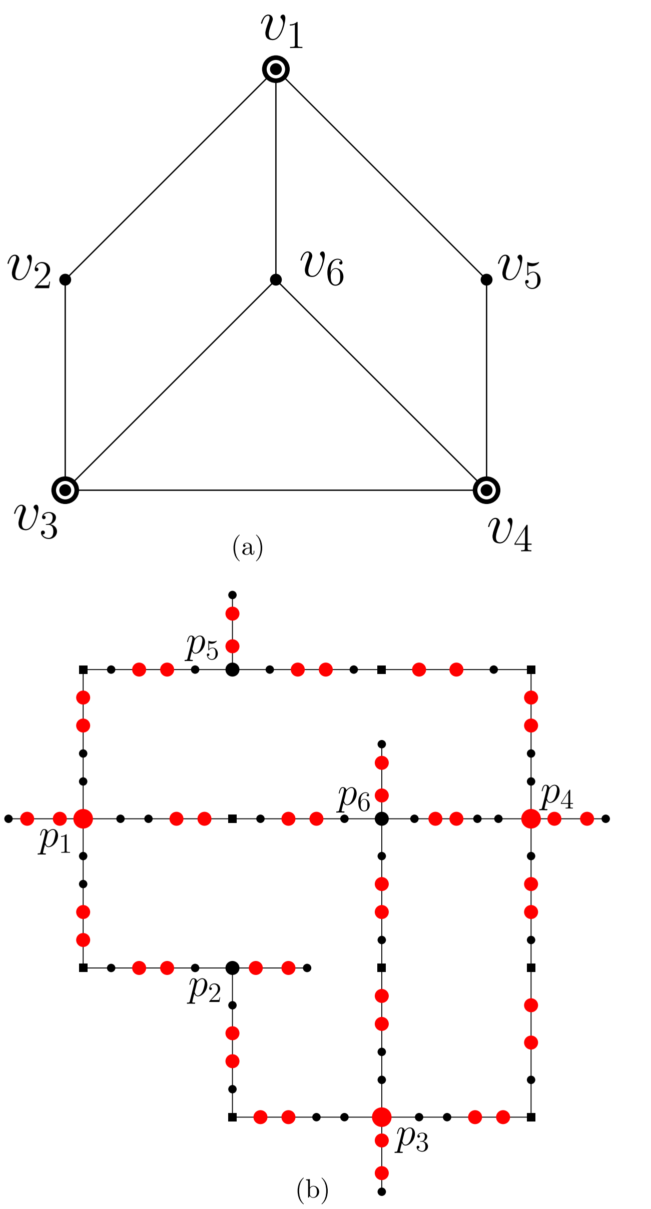


Figure 2: (a) A vertex cover $\{v_1, v_3, v_4\}$ in G, and (b) the construction of A' in G' (the tie between v_3 and v_4 is broken by choosing v_3)

end vertices in C. Let v_iv_j be an edge in G and $v_i \in C$ (choose any of them arbitrarily if both v_i and v_j are in C). Note that the edge v_iv_j is represented as a sequence of line segments in the embedding. Start traversing the segments (of v_iv_j) from p_i, where p_i corresponds to v_i, and add two consecutive vertices by leaving two consecutive vertices in between starting from p_i to A' in the traversal (see p_4p_5 in Figure 2(b)). The red bold vertices are part of A' while traversing from p_4).

Apply the above process to each edge in G. Observe that the cardinality of A' is 2ℓ as we have chosen 2 vertices from each segment in the embedding. Next, we choose $2n$ points from S in $S' = \{x_i, y_i : p_i \in N\}$. Let $T = N' \cup A' \cup S'$. Now, we argue that T is a total dominating set in G'.

For each point $p_i \in N$, p_i is dominated by x_i, x_i is dominated by y_i, y_i is dominated by x_i and z_i is dominated by y_i. So, the sets N and S satisfies total domination condition. Now it is remaining to prove that the set A satisfies total domination condition. Observe the way we have chosen points from A in T, with a gap of two consecutive points, two consecutive points are chosen in T. For each point $p_i \in T$, p_i dominates $N_G(p_i) \in A$ and the selected points of A in T can total dominate all the remaining points of A (see the edge p_1p_2 in Figure 2(b)).

Therefore, T is a TDS in G' and $|T| = |N'| + |A'| + |S'| \leq k + 2\ell + 2n$.

Sufficiency: Let $T \subseteq V'$ be a TDS of size at most $k + 2\ell + 2n$. We prove that G has a vertex cover of size at most k with the help of the following claims.

(i) Out of three support points associated with each $p_i \in N$, at least two points belongs to T, i.e., $|S \cap T| \geq 2n$.

(ii) Every segment in the embedding must contribute at least two points to T and hence $|A \cap T| \geq 2\ell$, where ℓ is the total number of segments in the embedding.

(iii) If p_i and p_j correspond to end vertices of an edge v_iv_j in G, and if both p_i, p_j are not in T, then there must be at least $2\ell' + 1$ vertices in T from the segment(s) representing the edge v_iv_j, where ℓ' is the number of segments representing the edge v_iv_j in the embedding.

Claim (i) directly follows from the definition of total dominating set. Observe that we added points x_i, y_i, z_i such that p_i is adjacent to x_i, x_i is adjacent to y_i, and y_i is adjacent to z_i in G', i.e., $\{p_ix_i, x_iy_i, y_iz_i\} \subseteq E'$ for each i. Hence, y_i must be in T as y_i is the only vertex which can dominate z_i and either x_i or z_i must be in T to dominate y_i. Therefore, any total dominating set of G' must contain two support points in T out of three support points associated with $p_i, 1 \leq i \leq n$, i.e., $|S \cap T| \geq 2n$.

Claim (ii) follows from the fact that only consecutive points are adjacent (in G') on any segment in the embedding. Let η be a segment in the embedding having vertices q_i, q_{i+1}, q_{i+2}, and q_{i+3}. On contrary, assume that η has only one of its vertices in T. Note that only q_i can not be in T. If q_i present in T, then q_{i+2} is not dominated by any point, which is a contradiction to the fact that T is a TDS. If q_{i+1} is the only point in T then q_{i+1} is not dominated by any point. If q_{i+2} will be chosen as the only point from η in T then q_{i+2} is not dominated by any other point and finally if q_{i+3} will be chosen then q_{i+1} is not dominated by any other point. In all cases, we arrived at a contradiction.

Claim (iii) follows from the definition of the total dominating set that any point chosen in the solution set dominates all its neighbors other than itself. Here any point selected from a segment in T has exactly two neighbors other than itself. So, it can dominate at most 2 points. There are ℓ' segments between two node points p_i and p_j having $4\ell'+1$ number of points and both p_i and p_j are not in T. So, the minimum number of points required in T to ensure total domination is $\left\lceil \frac{4\ell'+1}{2} \right\rceil = 2\ell'+1$.

Now, we will show that, by removing and/or replacing some vertices in T, a set of at most k points from N can be chosen such that the corresponding vertices form a vertex cover in G. The vertices in S account for $2n$ vertices in T (due to Claim (i)). Let $T = T \setminus S$ and $C = \{v_i \in V \mid p_i \in T \cap N\}$. If any edge v_iv_j in G has none of its end vertices in C, then we do the following: consider the sequence of segments representing the edge v_iv_j in the embedding. Since both p_i and p_j are not in T, there must exist a segment having three vertices in T (due to Claim (iii)). Consider the segment having its three vertices in T. Delete any one of the vertices on the segment and introduce p_i (or p_j). Update C and repeat the process till every edge has at least one of its end vertices in C. Due to Claim (ii), C is a vertex cover in G with $|C| \leq k$. Therefore, TDS-UDG $\in$ NP-hard. As TDS-UDG $\in$ NP and TDS-UDG $\in$ NP-hard, TDS-UDG $\in$ NP-complete. □

3 Approximation Algorithm

In this section, we propose an 8-factor approximation algorithm for the TDS problem in UDGs. The worst-case time complexity of our proposed algorithm is $O(n \log k)$, where k is the size of the output of our algorithm. Note that, if there exists an r-factor approximation algorithm for the dominating set problem then a $2r$-factor approximation algorithm for the total dominating set problem can be obtained by adding one neighbor for each vertex in the dominating set. A 10-factor approximation algorithm is available in the literature for the TDS problem in UDGs [26]. In this section, we improved the approximation factor of the TDS problem in UDGs to 8. The running time of the 8-factor approximation algorithm is $O(n \log k)$.

Lemma 4 *[26] Let C be a unit disk centered at a point p. If S is a set of independent[1] unit disks such that every unit disk in S contains the point p then $|S| \leq 5$.*

Lemma 5 *Let C' and C'' be two unit disks centered at points p and q respectively, such that $d(p,q) = 1$. If S is a set of independent unit disks containing the point p and/or q, then $|S| \leq 8$.*

[1]Two disks are independent if the distance between their centers is greater than one.

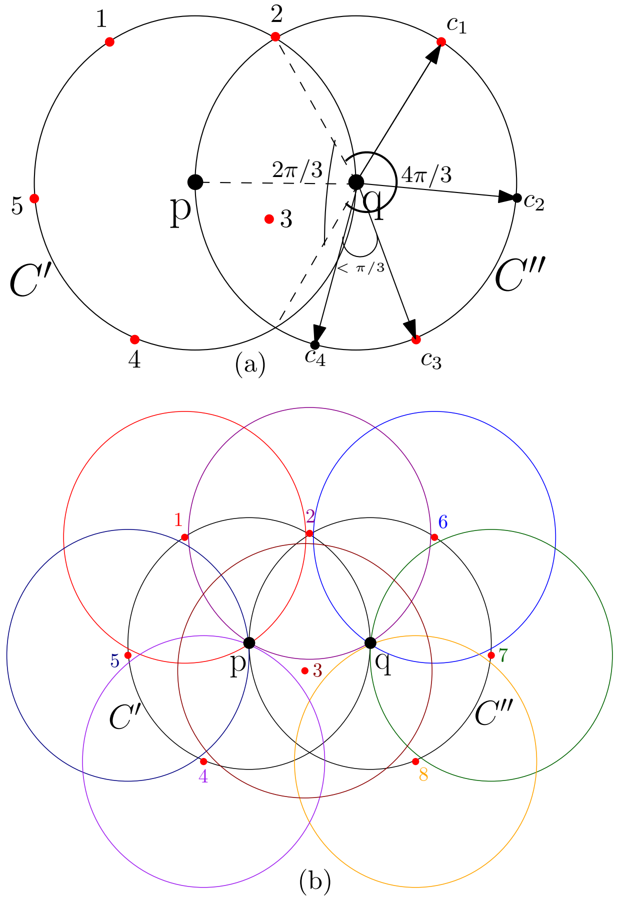


Figure 3: Illustration of Lemma 5.

Proof. Let S_p, $S_q \subseteq S$ be the sets of independent unit disks containing the point p and q respectively. This implies $S = S_p \cup S_q$. Note that, the cardinality of S will increase if the distance between p and q increases. Therefore, without loss of generality, we assume that the points p and q are one unit distance apart, i.e., the point q lies on the boundary of the unit disk C' centered at p (see Figure 3 for reference). Now, we consider the following two cases:

Case (i): $|S_p| \leq 3$. From Lemma 4, we observed that $|S_q| \leq 5$. Hence, $|S_p \cup S_q| \leq 8$.

Case (ii): $|S_p| > 3$. It is known that $|S_p| \leq 5$ (see Lemma 4). So $|S_p|$ is either 4 or 5.

(a) Assume that $|S_p| = 5$. Now it is remaining to prove that, there does not exist more than 3 independent unit disks in the set (say S'_q) $\subseteq S_q$, which contains the point q but does not contain the point p, i.e., $|S'_q| = |S_q| - |S_p \cap S_q| \leq 3$. For the sake of argument assume that $|S'_q| = 4$, that means, there exist 4 unit disks centered at c_i, $1 \leq i \leq 4$ in S'_q, which contains the point q. Let us denote the ray $\overrightarrow{qc_i}$ by r_i,

$(1 \leq i \leq 4)$. Since there are 4 rays coming out from q and the disks centered at c_i, $(i = 1, 2, 3, 4)$ does not contains p, there exist one pair of rays r_i and r_j such that the angle between them is at most $\pi/3$ (see Figure 3(a)), which leads to a contradiction that the disks centered at c_i, $(i = 1, 2, 3, 4)$ are independent. Thus, $|S'_q| \leq 3$, which implies $|S_p \cup S_q| \leq 8$ (see Figure 3(b)).

(b) Assume that $|S_p| = 4$. Now it is sufficient to prove that $|S'_q| < 5$, to prove $|S_p \cup S_q| \leq 8$. For the sake of argument assume that $|S'_q| = 5$, that means, there exist 5 independent unit disks in the set S'_q, which contains the point q but does not contain the point p. Following a similar argument as in case (a), it leads to a contradiction. Thus, $|S'_q| < 5$, which implies $|S_p \cup S_q| \leq 8$.

In either of the cases, we proved that $|S_p \cup S_q| \leq 8$. Therefore, $|S| \leq 8$. □

Let P denote the set of n points (center of the disks) given in the plane $\mathbb{R}^2$. We use $\Delta(S)$ to denote the set of unit disks centered at the points in $S \subseteq P$. A dominating set in unit disk graphs (Ds-Udg) $D \subseteq P$ of the set of disks $\Delta(P)$ is said to be an independent Ds-Udg if for each pair p, $q \in D$, $p \notin N_G(q)$.

The procedure of generating a Tds-Udg for a given points set P in $\mathbb{R}^2$ is described in Algorithm 1.

Algorithm 1 Total dominating set in P

```
Require: A set of disks $\Delta(P)$
Ensure: A total dominating set $T$ of $\Delta(P)$
 1: $D \leftarrow \emptyset$, and $T \leftarrow \emptyset$
 2: while ($P \neq \emptyset$) do
 3:     choose an arbitrary point $p \in P$
 4:     $D \leftarrow D \cup \{p\}$; $P \leftarrow P \setminus N_G[p]$
 5: end while
 6: for every $p \in D$ do
 7:     if $N_G(p) \cap T = \emptyset$ then
 8:         let $q \in N_G(p)$
 9:         $T = T \cup \{q\}$
10:     end if
11: end for
12: $T = T \cup D$
13: return $T$
```

Lemma 6 *T returned by Algorithm 1 is a* Tds-Udg *for the set of unit disks $\Delta(P)$.*

Proof. Before finding a Tds-Udg T, Algorithm 1 find an independent Ds-Udg D (see **while** loop in line number 2 of the algorithm), which ensures domination for the set of unit disks $\Delta(P)$ and total domination for the points $P \setminus D$. Next, to obtain total domination in D, for each point $p \in D$ the algorithm ensures the existence of a point $q \in N_G(p)$ in T (see **for** loop in line number 6 of the algorithm). Therefore the selected points in T along with D ensures total domination for the set of unit disks $\Delta(P)$. □

Lemma 7 *$|T| \leq 8|OPT|$, where OPT is a* Tds-Udg *for the unit disks $\Delta(P)$ of minimum size.*

Proof. Consider an arbitrary point $p \in OPT$. As OPT is a TDS of minimum size there must exist a point $q \in OPT$ such that $p \in N_G(q)$ and the point q ensures the domination for the point p. Now consider both the unit disks centered at p and q. From Lemma 5, it is proved that there exist at most 8 unit disks in an independent Ds-Udg D of $\Delta(P)$ that can contain the points p and/or q.

Now the cardinality of T follows from the fact that for every pair $p, q \in OPT$ such that $p \in N_G(q)$, there may exist at most 8 unit disks in an independent Ds-Udg D of $\Delta(P)$ that can contain the points p and/or q. Our algorithm chooses at most one neighbor for each point in D in to the solution set T, to make T as a Tds-Udg (see line numbers 6-12 of Algorithm 1). So for 8 points in D at most 16 points may be chosen in T with respect to two points $p, q \in OPT$. which leads at most 16 points are chosen by Algorithm 1 against 2 points (namely, p and q) chosen in the optimal solution of the TDS, i.e., $|T| \leq \frac{16}{2}|OPT|$. Therefore, $|T| \leq 8|OPT|$. □

Lemma 8 *The worst-case time required to generate a* Tds-Udg *for the set of disks $\Delta(P)$ by Algorithm 1 is $O(n \log k)$, where k is the size of the output.*

Proof. We now describe the time complexity of Algorithm 1 for computing a Tds-Udg T of $\Delta(P)$ as follows. Let us assume that $\mathcal{R}$ is an axis parallel rectangular region containing the points in P. We partition $\mathcal{R}$ into grid cells of size 1×1. A point $p_i = (x_i, y_i) \in P$ lies in the grid cell indexed by $[\lfloor x_i \rfloor, \lfloor y_i \rfloor]$ for $i = 1, 2, \ldots, n$. Each grid cell is attached with a list of points in P that belong to that cell. We construct an independent dominating set D for UDG corresponding to $\Delta(P)$. While considering a point $p_i \in P$, we inspect all members of D which are attached to all 9 cells $[\alpha, \beta]$, where $\lfloor x_i \rfloor - 1 \leq \alpha \leq \lfloor x_i \rfloor + 1$ and $\lfloor y_i \rfloor - 1 \leq \beta \leq \lfloor y_i \rfloor + 1$. If there does not exist any unit disk d in D that contains the point p_i, we add p_i in D. Observe that, at the end of considering all the points in P, D will be an independent Ds-Udg for the set of disks in $\Delta(P)$. Initially, take $T = \emptyset$. Now, for each point $p \in D$, if there does not exist any point q in T such that $q \in N_G(p)$, then add q in T, and the existence of q is guaranteed, otherwise finding TDS for this given instance is impossible as the point p is an isolated point. Finally, update $T = T \cup D$. After ensuring the existence of a point $q \in N_G(p)$ for each point $p \in D$, observe that T is a Tds-Udg for the set of disks in $\Delta(P)$. Note that, (i) a grid cell may contain at most 6 points in T, and (ii) the number of grid cells to be inspected while processing a point $p_i \in P$ is at most 9. We use a height-balanced binary tree to store the indices of the grid cells containing a non-zero number of points in T. Thus, the time complexity for processing a point $p \in P$ is $O(\log k)$, where $k = |T|$ and $|P| = n$. □

Theorem 9 *Algorithm 1 is an 8-factor approximation algorithm for the* TDS-UDG *problem. The running time of the algorithm is $O(n \log k)$, where n is the input size and k is the output size.*

Proof. Follows from Lemma 6, Lemma 7, and Lemma 8. □

4 Approximation Scheme

In this section, we propose a polynomial-time approximation scheme (PTAS) for the TDS problem in geometric UDGs. We use the shifting strategy [15] technique to propose a PTAS. Let P be a point set (centers of the disks) given in a rectangular region $\mathcal{R}$ along with a fixed integer $k \geq 1$.

We use a two-level nested shifting strategy to propose a PTAS for the said problem. The first level of shifting strategy is applied in the vertical direction on $\mathcal{R}$. There are k iterations in the first level and the i-th iteration $(1 \leq i \leq k)$ partition the region $\mathcal{R}$ into many vertical strips, where the first strip is of width $2i$, and remaining strips other than the last strip are of width $2k$. The width of the last strip may be less than $2k$. Without loss of generality, assume that each point lying on the left boundary of a strip belongs to its left adjacent strip (i.e., every strip is left open and right closed). Now consider all the non-empty vertical strip $\mathcal{V}$, and apply the second level of shifting strategy in the horizontal direction. In the second level of shifting strategy, the j-th iteration $(1 \leq j \leq k)$ partition each non-empty vertical strip $\mathcal{V}$ into square/rectangular cells of size $2j \times \ell$ for the first cell and $2k \times \ell$ for all other cells, where ℓ defines the width of the strip $\mathcal{V}$ ($\ell = 2i$ for the first strip and $\ell = 2k$ for all other strips except for the last strip).

We consider each non-empty $2k \times 2k$ square (conceptually extending the smaller cells into $2k \times 2k$ square) and find the optimal solution of each square. The union of the optimal solution of each $2k \times 2k$ square gives a feasible solution of each strip $\mathcal{V}$. Finally, we take the union of the solution of each non-empty vertical strip to get a feasible solution of the problem in a single iteration, i.e., (i, j)-th iteration. In the same process, we get the feasible solutions of all the iterations in the first level. We report the solution T, having minimum cardinality among all the solutions generated in each iteration as the solution of the TDS-UDG problem.

Now, we discuss the procedure of getting an optimal solution from each $2k \times 2k$ square. We first partition the cell of size $2k \times 2k$ into $(\lceil 2\sqrt{2}k \rceil)^2$ sub-cells. The size of each sub-cell is $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$. Observe that, choosing any two points inside a sub-cell of size $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$ ensures total domination for all unit disks centered in that sub-cell. Hence, the maximum number of points required to ensure total domination in a square of size $2k \times 2k$ is $2(\lceil 2\sqrt{2}k \rceil)^2$. Therefore, we have to check all possible combinations of points up to $2(\lceil 2\sqrt{2}k \rceil)^2$ to get an optimal solution in a cell χ of size $2k \times 2k$. Note that, along with the points inside a cell χ, the points within 1 unit apart from χ also play a crucial role to get an optimum solution of χ. Let n_χ be the number of points in P whose corresponding disks have a portion in the cell χ (n_χ includes the points inside χ along with the points within 1 unit apart from χ). Then, we have to choose at most $O(n_\chi^{2(\lceil 2\sqrt{2}k \rceil)^2})$ combinations of points for getting the optimum solution for the TDS-UDG problem in a cell χ of size $2k \times 2k$. Since the points in P centered in a cell are disjoint from that of the other cells, and a point in P can participate in computing the optimum solution of at most 9 cells, we have the following result.

Lemma 10 *The total time required for the (i, j)-th iteration of the algorithm is $O(n^{2(\lceil 2\sqrt{2}k \rceil)^2})$.*

Proof. The feasible solution of the (i, j)-th iteration is the union of the optimum solutions of all the cells constructed in that iteration. Finally, the algorithm returns the minimum among the k^2 feasible solutions corresponding to k^2 iterations. □

Theorem 11 *Given a set P of n points (center of the unit disks) in $\mathcal{R}$ and an integer $k \geq 1$, a total dominating set of size at most $(1+\frac{1}{k})^2 \times |OPT|$ can be computed in $O(k^2 n^{2(\lceil 2\sqrt{2}k \rceil)^2})$ time, where OPT is the optimum solution.*

Proof. Using the shifting strategy analysis given by Hochbaum and Maass [15], we analyze the approximation factor of our algorithm. Let OPT be an optimum solution for the TDS-UDG problem for the point set P, and $OPT' \subseteq OPT$ be such points chosen in OPT, which totally dominate some points outside the boundary of all the cells in an (i, j)-th iteration. Let T be a solution obtained by our algorithm in an iteration. Then, $|T| \leq |OPT| + |OPT'|$. For all the iterations of (i, j) $(1 \leq i, j \leq k)$, we have $\sum_{i=1}^{k}\sum_{j=1}^{k} |T| \leq k^2|OPT| + \sum_{i=1}^{k}\sum_{j=1}^{k} |OPT'|$. Since any point from a cell χ chosen in OPT can dominate points from no more than one horizontal strip (or vertical strip), and at most k times each horizontal (or vertical) boundary appears throughout the algorithm, we have $\sum_{i=1}^{k}\sum_{j=1}^{k}|OPT'| \leq k|OPT| + k|OPT|$. Thus, $\sum_{i=1}^{k}\sum_{j=1}^{k}|T| \leq k^2|OPT| + 2k|OPT| = (k^2+2k)|OPT|$. Therefore, $min\sum_{i=1}^{k}\sum_{j=1}^{k}|T| \leq (1+\frac{1}{k})^2 \times |OPT|$. The time complexity result follows from Lemma 10. □

5 Conclusion

In this article, we have considered the minimum total dominating set (TDS) problem in geometric unit disk graphs. We showed that the TDS problem belongs to the class NP-complete. We proposed an almost linear time 8-factor approximation algorithm, which is an improvement over the best-known 10-factor approximation algorithm with the same running time available in the literature [26]. We also proved that the TDS problem in geometric UDGs admits a PTAS.

References

[1] D. Amos and E. DeLaVina. On Total Domination in Graphs. *Senior Project, University of Houston Downtown*, 2012.

[2] R. C. Brigham, J. R. Carrington, and R. P. Vitray. Connected Graphs with Maximum Total Domination Number. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 34:81–96, 2000.

[3] Paz Carmi, Gautam K. Das, Ramesh K. Jallu, Subhas C. Nandy, Prajwal R. Prasad, and Yael Stein. Minimum Dominating Set Problem for Unit Disks Revisited. *International Journal of Computational Geometry and Applications*, 25(3):227, 2015.

[4] M. Chellali and T. W. Haynes. A Note on the Total Domination Number of a Tree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 58:189, 2006.

[5] V. Chvátal and C. McDiarmid. Small Transversals in Hypergraphs. *Combinatorica*, 12(1):19–26, 1992.

[6] E. J. Cockayne, R. Dawes, and S. T. Hedetniemi. Total Domination in Graphs. *Networks*, 10(3):211–219, 1980.

[7] E. J. Cockayne, M. A. Henning, and C. M. Mynhardt. Vertices Contained in all or in no Minimum Total Dominating Set of a Tree. *Discrete mathematics*, 260(1-3):37–44, 2003.

[8] E. DeLaViņa, Q. Liu, R. Pepper, B. Waller, and D. B. West. Some Conjectures of Graffiti. pc on Total Domination. 2007.

[9] M. Dorfling, W. Goddard, and M. A. Henning. Domination in Planar Graphs with Small Diameter ii. 2006.

[10] T. Haynes and M. Henning. Trees with Unique Minimum Total Dominating Sets. *Discussiones Mathematicae Graph Theory*, 22(2):233–246, 2002.

[11] T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs.* CRC press, 1998.

[12] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. Domination in Graphs (Advanced Topics) Marcel Dekker Publications. *New York*, 1998.

[13] M. A. Henning. A Survey of Selected Recent Results on Total Domination in Graphs. *Discrete Mathematics*, 309(1):32–63, 2009.

[14] M. A. Henning and A. Yeo. *Total Domination in Graphs.* Springer, 2013.

[15] D. S. Hochbaum and W. Maass. Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[16] J. Hopcroft and R. Tarjan. Efficient Planarity Testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

[17] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton Paths in Grid Graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[18] D. Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[19] R. Laskar, J. Pfaff, S. Hedetniemi, and S. Hedetniemi. On the Algorithmic Complexity of Total Domination. *SIAM Journal on Algebraic Discrete Methods*, 5(3):420–425, 1984.

[20] J. Pfaff, R. Laskar, and S. Hedetniemi. Np-completeness of Total and Connected Domination and Irredundance for bipartite graphs. In *Tech. Rept. 428*. Clemson University Clemson, SC, 1983.

[21] L. Sun. An Upper Bound for the Total Domination Number. *J. Beijing Inst. Tech*, 4(2):111–114, 1995.

[22] S. Thomassé and A. Yeo. Total Domination of Graphs and Small Transversals of Hypergraphs. *Combinatorica*, 27(4):473–487, 2007.

[23] Z. Tuza. Covering all Cliques of a Graph. *Discrete Mathematics*, 86(1-3):117–126, 1990.

[24] T. Nieberg and J. Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. *International Workshop on Approximation and Online Algorithms*, 296–306, 2005.

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

[26] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.

[27] C. Ambühl, T. Erlebach, M. Mihalák and M. Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. *In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 3–14, 2006.

[28] T. Erlebach and M. Mihalák. A $(4+\epsilon)$-approximation for the minimum-weight dominating set problem in unit disk graphs. *In International Workshop on Approximation and Online Algorithms*, 135–146, 2009.

[29] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-Approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238–274, 1998.

[30] H. Breu, and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

[31] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In SODA, 1: 671–679, 2001.

[32] L. G. Valiant. Universality Considerations in VLSI Circuits. *IEEE Transactions on Computers*, 100(2):135–140, 1981.

Simple Linear Time Algorithms For Piercing Pairwise Intersecting Disks*

Ahmad Biniaz† Prosenjit Bose‡ Yunkai Wang‡

Abstract

A set $\mathcal{D}$ of disks in the plane is said to be pierced by a point set P if each disk in $\mathcal{D}$ contains a point of P. Any set of pairwise intersecting unit disks can be pierced by 3 points (H. Hadwiger and H. Debrunner, Ausgewählte Einzelprobleme der kombinatorischen Geometrie in der Ebene, Enseignement Math, 1955) and Danzer established that any set of pairwise intersecting arbitrary disks can be pierced by 4 points (L. Danzer, Zur Lösung des Gallaischen Problems über Kreisscheiben in der Euklidischen Ebene, Studia Scientiarum Mathematicarum Hungarica, 1986). Existing linear-time algorithms for finding a set of 4 or 5 points that pierce pairwise intersecting disks of arbitrary radius use the LP-type problem as a subroutine. We present simple linear-time algorithms for finding 3 points for piercing pairwise intersecting unit disks, and 5 points for piercing pairwise intersecting disks of arbitrary radius. Our algorithms use simple geometric transformations and avoid heavy machinery. We also show that 3 points are sometimes necessary for piercing pairwise intersecting unit disks.

1 Introduction

Let $\mathcal{D}$ be a set of pairwise intersecting disks in the plane. Helly's theorem states that if every set of 3 disks in $\mathcal{D}$ has a non-empty intersection, then all disks in $\mathcal{D}$ can be pierced by 1 point, in other words, $\cap\mathcal{D}$ is non-empty [7, 8]. Finding a piercing point set is more difficult if the disks in $\mathcal{D}$ only intersect pairwise and $\mathcal{D}$ contains groups of 3 disks that have no common intersection. Danzer [3] and Stachó [11] independently showed that such a set $\mathcal{D}$ can be pierced by at most 4 points. Danzer's proof is based on his first unpublished proof in 1956, while Stachó's proof uses similar ideas that were used in his previous construction of 5 piercing points in 1965 [10]. Even though Danzer proved that 4 points are sufficient, the proof is not constructive [3]. Stachó's construction is simpler, but it is still not simple enough to be turned into an easy subquadratic algorithm [10, 11]. Har-Peled et al. [6] presented the first deterministic linear-time algorithm for finding 5 piercing points of a set $\mathcal{D}$ by formulating the piercing problem as an LP-type problem. An LP-type problem is an abstract generalization of a low-dimensional linear program. Chazelle and Matoušek showed that LP-type problems can be solved in deterministic linear time if we have a constant-time violation test and the range space has bounded VC-dimension [2]. More recently, Carmi et al. [1] presented a linear time algorithm for finding 4 piercing points. Their algorithm requires the computation of the smallest disk that intersects every disk in $\mathcal{D}$, which they formulated as an LP-type problem [2, 9]. They pose as an open problem to find the piercing set without using linear programming.

As for lower bounds on this problem, Grünbaum [4] provides a set of 21 pairwise intersecting disks that cannot be pierced by 3 points. Later, Danzer [3] reduced the number of disks to 10. This is close to optimal since every set of 8 pairwise intersecting disks can be pierced by 3 points [10]. However, Danzer's construction is difficult to verify since the positions of the disks cannot be visualized easily. Har-Peled et al. [6] gave a simpler construction with 13 disks.

Hadwiger and Debrunner [5] showed that if all the disks in $\mathcal{D}$ have the same radius, then 3 points are sufficient to pierce $\mathcal{D}$. Their algorithm computes the smallest regular hexagon enclosing the centers of all disks in $\mathcal{D}$. It is not clear how one can simply find such a hexagon in linear time.

1.1 Our Contributions

We present a deterministic linear time algorithm for finding 3 points that pierce a set of pairwise intersecting *unit disks* (disks of radii one), and a deterministic linear time algorithm for finding 5 points that pierce a set of pairwise intersecting *arbitrary disks* (disks of arbitrary radii). Our algorithms employ simple geometric transformations, and do not require solving any LP-type problem. We also present a set of 9 pairwise intersecting unit disks that cannot be pierced by 2 points. This shows that 3 points are sometimes necessary and always sufficient to pierce pairwise intersecting unit disks.

We denote the Euclidean distance between points a and b by $|ab|$.

*Research supported in part by NSERC.

†School of Computer Science, University of Windsor, Windsor, Canada. ahmad.biniaz@gmail.com

‡School of Computer Science, Carleton University, Ottawa, Canada. jit@scs.carleton.ca, yunkai@scs.carleton.ca

2 Piercing Pairwise Intersecting Unit Disks

In this section, we first present our deterministic linear-time algorithm for piercing pairwise intersecting unit disks by 3 points. Then we introduce a set of 9 pairwise intersecting unit disks that cannot be pierced by 2 points.

2.1 Algorithm For Computing Three Piercing Points

Let $\mathcal{D}$ be a set of pairwise intersecting unit disks, each disk D_i is centered at $c_i = (x_i, y_i)$.

Theorem 1 *Let $\mathcal{D}$ be a set of pairwise intersecting unit disks. In $O(|\mathcal{D}|)$ time, we can compute 3 points that pierce $\mathcal{D}$.*

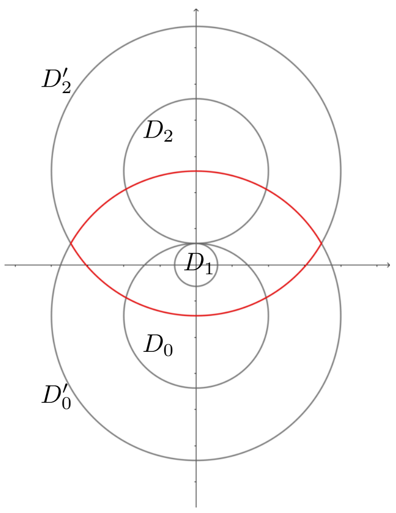


Figure 1: Configuration of Theorem 1.

Proof. Let D_1 be an arbitrary disk in $\mathcal{D}$. We reduce its radius while keeping c_1 fixed until D_1 becomes tangent to another disk $D_2 \in \mathcal{D}$. This can be completed in $O(|\mathcal{D}|)$ time by computing the distance from c_1 to all other disks in $\mathcal{D}$. Notice that the disks in $\mathcal{D}$ are still pairwise intersecting and any set of points that pierces the new set of disks also pierces the original set of disks. Let r_1 be the radius of D_1. After this transformation, $r_1 \leq 1$, and D_1 is tangent to D_2. By a translation and rotation, we move c_1 to the origin and c_2 to a point that lies on the positive y-axis with coordinate $(0, r_1 + 1)$. Let D_0 be a unit disk (not necessarily in $\mathcal{D}$) with center $c_0 = (0, r_1 - 1)$. Since $r_1 \leq 1$, $D_1 \subseteq D_0$. Any disk that intersects D_1 also intersects D_0. Let D'_0 and D'_2 be two disks with radius 2 and centers c_0 and c_2, respectively. See Figure 1. If a unit disk D_i intersects D_0 and D_2, then $|c_0c_i| \leq 2$, $|c_2c_i| \leq 2$ and $c_i \in D'_0 \cap D'_2$.

Let D_3 be the disk in $\mathcal{D}$ with the maximum x-coordinate. Since D_3 belongs to $\mathcal{D}$, it must intersect

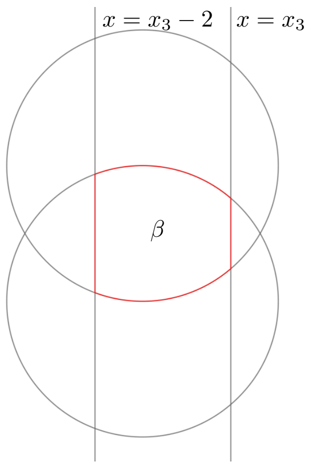


Figure 2: Area that we need to cover.

D_1 and D_2, we note that $0 \leq x_3 \leq \sqrt{3}$. $x_3 \geq 0$ since $x_3 \geq x_1 = 0$. The boundaries of D'_0 and D'_2 intersect at the point $(\sqrt{3}, r_1)$, so c_3 must either fall on or the left of the line $x = \sqrt{3}$. We conclude that $x_3 \leq \sqrt{3}$. The disk D_3 can be found in $O(|\mathcal{D}|)$ time. For every disk $D_i \in \mathcal{D}$, $|c_ic_3| \leq 2$ since D_i and D_3 intersect. We have that $|x_ix_3| \leq 2$ since both D_i and D_3 are unit disks. Therefore, in addition to being in $D'_0 \cap D'_2$, the x-coordinate of all the centers lie in the interval $[x_3 - 2, x_3]$. Let β represent the region where all the centers of disks in $\mathcal{D}$ must lie as illustrated in red in Fig 2. We say an area is covered by a point set P if every point in the area has distance at most 1 to at least 1 point in P. Therefore, if we can find 3 points that cover β, then those three points pierce every disk in $\mathcal{D}$. As noted above, we have that $0 \leq x_3 \leq \sqrt{3}$. We consider two cases, namely when $1 \leq x_3 \leq \sqrt{3}$ and $0 \leq x_3 < 1$.

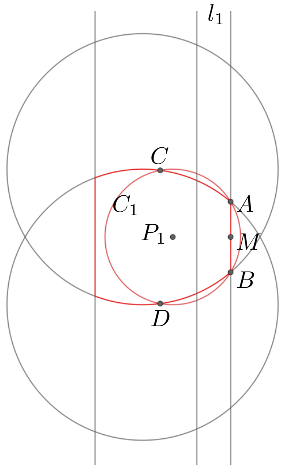


Figure 3: Location of P_1.

Case 1: $1 \leq x_3 \leq \sqrt{3}$. Let A (resp. B) be the rightmost point of β on the boundary of D'_0 (resp. D'_2). The first point P_1 is chosen be a point that falls in β

and has distance 1 to both A and B. Let C_1 be a circle of radius 1 centered at P_1; See Figure 3.

Let l_1 be the vertical line $x = x_3 - \frac{1}{2}$. First we prove that P_1 always lies to the left of l_1. Let the midpoint of line segment AB be M. $|AB|$ decreases as x_3 increases and it is maximized when $x_3 = 1$. When $x = 1$, $|AB| = 2\sqrt{3} - 2 < \sqrt{3}$. So $|AB| < \sqrt{3}$ and $|AM| < \frac{\sqrt{3}}{2}$. Since $\triangle P_1AM$ is a right triangle and $|AP_1| = 1$, by the Pythagorean theorem, $|P_1M| > \frac{1}{2}$. Therefore, P_1 always lies to the left of l_1. Let the intersection point of circle C_1 and D'_0 different from A be labelled C, and the intersection point of circle C_1 and D'_2 different from B be labelled D. P_1 lies on the bisector of the line segment AB, so P_1 lies on the line $y = r_1$, therefore, C_1 is tangent to both lines $y = r_1 + 1$ and $y = r_1 - 1$. Since the circle C_1 is tangent to these two lines, both C and D lie to the left of P_1. See Figure 3. Since the radius of C_1 is 1, the radius of D'_0 is 2, and C lies to the left of l_1, we have that the clockwise arc from C to A on the boundary of D'_0 and the clockwise arc from B to D on the boundary of D'_2 are both contained in C_1. Therefore, the center of any unit disk of $\mathcal{D}$ that lies on or to the right of l_1 is contained in the disk C_1. We now show how to compute points P_2 and P_3 to pierce all the disks that do not contain P_1, namely the disks in $\mathcal{D}$ whose centers are in β but outside disk C_1. The coordinates of A, B, P_1, P_2, and P_3 are given in Appendix A.

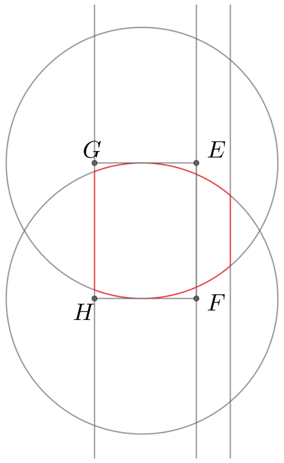


Figure 4: Remaining area to be covered.

Consider the rectangle formed by the following 4 points: $E = (x_3 - \frac{1}{2}, r_1 + 1), F = (x_3 - \frac{1}{2}, r_1 - 1), G = (x_3 - 2, r_1 + 1), H = (x_3 - 2, r_1 - 1)$. See Figure 4. Since D'_0 is tangent to the line $y = r_1 + 1$ at $(0, r + 1)$, and D'_2 is tangent to the line $y = r_1 - 1$ at $(0, r - 1)$, the area $\beta \cap \{x < x_3 - \frac{1}{2}\}$ as shown in Fig 4 is contained completely within the rectangle $EFHG$. If the points P_2 and P_3 cover this rectangle, then we are done. Let N be the midpoint of line segment EF and let O be the midpoint of line segment GH. See Figure 5. We choose P_2 to be the center of the rectangle $ENOG$. $|EN| = 1$ and $|NO| = \frac{3}{2}$, by Pythagorean theorem, P_2's distance to all four vertices of the rectangle is $\frac{\sqrt{13}}{4}$. Therefore, if a unit disk's center falls in the rectangle $ENOG$, then the disk is pierced by P_2. Symmetrically pick P_3 to be the center of the rectangle $NFHO$. Then any unit disk in $\mathcal{D}$ whose center falls to the left of l_2 is pierced by one of P_2 and P_3.

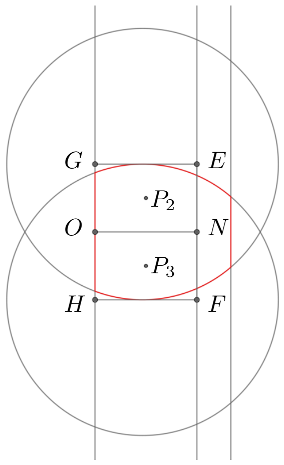


Figure 5: Location of P_2 and P_3.

Case 2: $0 \le x_3 < 1$. Let q be the maximum of x_3 and $2-\sqrt{3}$. By the definition, we know that $q \ge 2-\sqrt{3}$. We also know that the rightmost point on the lens formed by D'_0 and D'_2 is $(-\sqrt{3}, 0)$, so the line $x = x_3 - 2$ lies to the left of the point when $x_3 - 2 < -\sqrt{3}$. Therefore, we can safely say that the x-coordinate of all the centers lie in the interval $[-\sqrt{3}, -\sqrt{3}+2]$ when $x_3 < 2-\sqrt{3}$. Since q is the maximum of x_3 and $2-\sqrt{3}$, the x-coordinate of all the centers lie in the interval $[q-2, q]$ when $0 \le x_3 < 1$. $q \ge 2 - \sqrt{3}$, so $q - 2 \ge -\sqrt{3}$. $q < 1$, so $q - 2 < -1$. Therefore, we have that $-\sqrt{3} \le q-2 < -1$. If we reflect all the disks in $\mathcal{D}$ about the y-axis, then all the centers lie in the interval $[-q, |q-2|]$. Let $q' = |q-2|$, and we compute the piercing points using $x = q'$ and $x = q' - 2$ as in Case 1. Then the three computed points pierce $\mathcal{D}$. $\square$

2.2 A Lower Bound

We now present a set of 9 pairwise intersecting unit disks that cannot be pierced by 2 points. See Figure 6 for an illustration of these disks in a nutshell; details are given in Theorem 2.

Theorem 2 *There exists a set of 9 pairwise intersecting unit disks that cannot be pierced by 2 points.*

Proof. Follow Figure 7. We begin the construction by placing 3 unit disks D_1, D_2, D_3 centered at $(0, 0), (2, 0), (1, \sqrt{3})$ respectively. These points are the

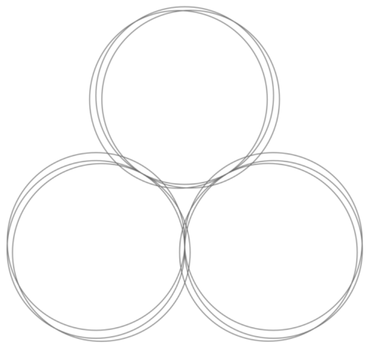


Figure 6: Nine unit disks that cannot be pierced by 2 points.

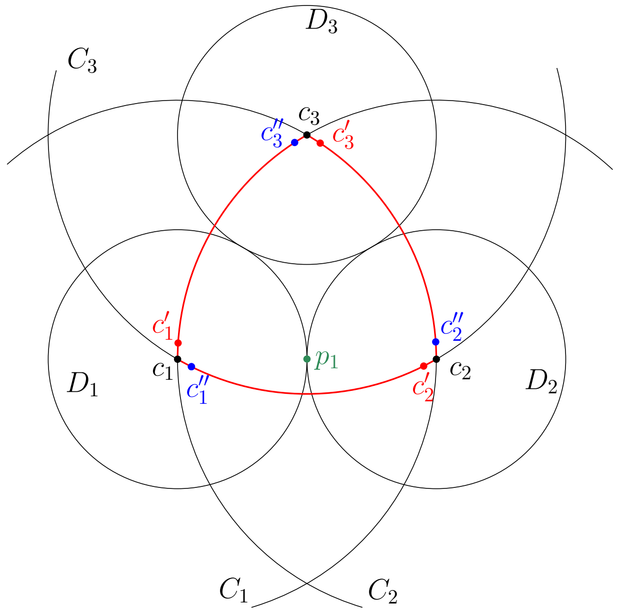


Figure 7: Illustration of the construction of a set of 9 pairwise intersecting unit disks that cannot be pierced by 2 points.

vertices of an equilateral triangle with side length 2. Notice that these disks are pairwise tangent. We denote the center of D_i by c_i. Let C_i be the circle of radius 2 centered at c_i. The intersection of C_1, C_2, and C_3 is a reuleaux triangle, which is illustrated in red in Figure 7. The center of any unit disk, that intersects D_i, lies in C_i. Therefore the center of any unit disk, that intersects the three disks D_1, D_2, and D_3, lies in the reuleaux triangle. We then introduce 6 more unit disks as follows where $\epsilon = 0.01$:

- D'_1 with center $c'_1 = (2 - \sqrt{4-\epsilon^2}, \epsilon)$ on C_2.
- D''_1 with center $c''_1 = (\epsilon, \sqrt{3} - \sqrt{4-(\epsilon-1)^2})$ on C_3.
- D'_2 with center $c'_2 = (2-\epsilon, \sqrt{3} - \sqrt{4-(\epsilon-1)^2})$ on C_3.
- D''_2 with center $c''_2 = (\sqrt{4-\epsilon^2}, \epsilon)$ on C_1.
- D'_3 with center $c'_3 = (1+\epsilon, \sqrt{4-(1+\epsilon)^2})$ on C_1.
- D''_3 with center $c''_3 = (1-\epsilon, \sqrt{4-(1+\epsilon)^2})$ on C_2.

We show that $\mathcal{D} = \{D_1, D'_1, D''_1, D_2, D'_2, D''_2, D_3, D'_3, D''_3\}$ is a desired set. Given the above coordinates of the centers of the disks in $\mathcal{D}$, one can simply verify that the distance between any two centers is at most 2 and thus the disks are pairwise intersecting.

Now we show that $\mathcal{D}$ cannot be pierced by two points. For the sake of contradiction, suppose that $\{p_1, p_2\}$ pierces all disks in $\mathcal{D}$. Then one of these points pierces at least two of the disks D_1, D_2 and D_3. Due to symmetry assume that p_1 pierces D_1 and D_2 (as in Figure 7), and thus $p_1 = (1, 0)$ since $|c_1c_2| = 2$. By our construction, p_1 does not pierces D'_1, D''_2, D_3, D'_3 and D''_3. Thus, these disks are pierced by p_2, and in particular $p_2 \in D'_1 \cap D''_2 \cap D_3$. The circumscribed circle of the triangle $c'_1c''_2c_3$ has radius 1.15, which implies that the intersection of D'_1, D''_2, and D_3 is empty, which is a contradiction. This finishes our proof. $\square$

3 Piercing Pairwise Intersecting Arbitrary Disks

We now consider a set $\mathcal{D}$ of pairwise intersecting disks of arbitrary sizes. Each disk $D_i \in \mathcal{D}$ is described by its center c_i and its radius r_i. Let D_1 be the smallest disk in $\mathcal{D}$. We shrink D_1 while fixing its center at c_1 until D_1 becomes tangent to another disk, say D_2. This can be done in linear time by computing the distance of c_1 to all c_i's and subtract the distances by the radius of the disks. In this new setting, disks in $\mathcal{D}$ are still pairwise intersecting and any set of points that pierces the new set of disks also pierces the original set of disks. After scaling, rotation and translation, assume that D_1 has radius 1 and is centered at the origin and D_2 is centered on the positive y-axis; these transformations can be performed in linear time.

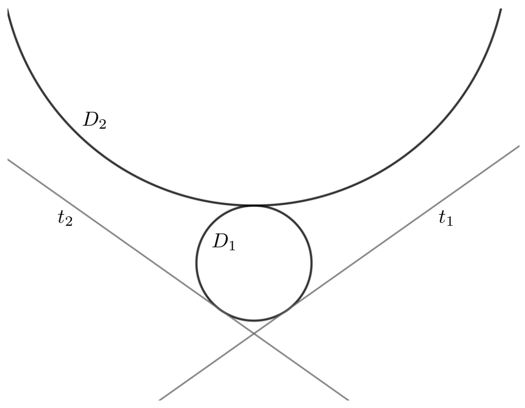


Figure 8: Configuration of Lemma 3.

Before showing our algorithm for finding the piercing set, we first present 2 geometric lemmas that will be proved later. See Figure 8 for the configuration outlined in the statement of Lemma 3 and Figure 9 for the configuration of Lemma 4. In the lemmas, we let $P_1 = (0,0)$, $P_2 = (\sqrt{3},0)$, $P_3 = (\frac{\sqrt{3}}{2},\frac{3}{2})$, $P_4 = (-\frac{\sqrt{3}}{2},\frac{3}{2})$, $P_5 = (-\sqrt{3},0)$ and let $P = \{P_1,P_2,P_3,P_4,P_5\}$. Points $\{P_2,P_3,P_4,P_5,(-\frac{\sqrt{3}}{2},-\frac{3}{2}),(\frac{\sqrt{3}}{2},-\frac{3}{2})\}$ are the vertices of a regular hexagon with sides of length $\sqrt{3}$ centered at the origin. Specifically points P_2 to P_5 are the top 4 vertices of the regular hexagon; see Figure 10.

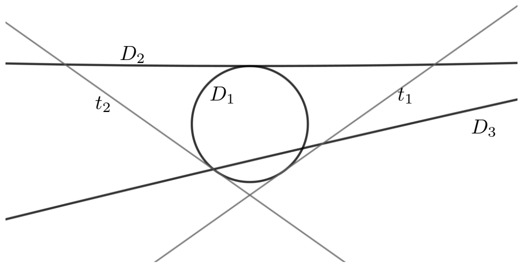


Figure 9: Configuration of Lemma 4.

Lemma 3 *If the radius of D_1 is 1 and the radius of D_2 is at most $5+2\sqrt{6}$, then P pierces $\mathcal{D}$.*

Lemma 4 *If the radius of D_1 is 1, the radius of D_2 is larger than $5+2\sqrt{6}$ and there exists at least one disk in $\mathcal{D}$ that misses all the points in P, then we can find in constant time a different set of 5 points that pierces $\mathcal{D}$.*

These two lemmas are sufficient for proving the existence of 5 piercing points for arbitrary disks.

3.1 Algorithm

1. Find the smallest disk $D_1 \in \mathcal{D}$
2. Reduce the radius of D_1 until D_1 is tangent to a disk in $\mathcal{D}$, say D_2
3. By scaling, rotation and translation of $\mathcal{D}$, let the center of D_1 be the origin and the radius of D_1 be 1. Let D_2 be centered on the y-axis above D_1
4. If $r_2 \leq 5+2\sqrt{6}$, then P pierces $\mathcal{D}$
5. If $r_2 > 5+2\sqrt{6}$ and there exist at least one disk in $\mathcal{D}$ that misses all the 5 points in P, then by Lemma 4, we find another set of 5 points that pierces $\mathcal{D}$ in constant time.

Theorem 5 *Given a set of pairwise intersecting arbitrary disks in the plane, in deterministic linear time, we can find 5 points that pierce the set.*

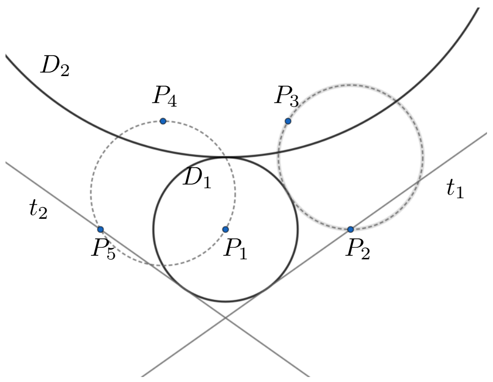


Figure 10: The first candidate set of 5 points.

Proof. Let $\mathcal{D}$ be a set of pairwise intersecting arbitrary disks. If we apply algorithm as depicted in Section 3.1 on $\mathcal{D}$, it will return 5 points. If $r_2 \leq 5+2\sqrt{6}$, by Lemma 3, P pierces $\mathcal{D}$. If $r_2 > 5+2\sqrt{6}$ and there exists at least one disk in $\mathcal{D}$ that is not pierced by any of the 5 points in P, then by Lemma 4 we can find 5 points that pierce $\mathcal{D}$.

The correctness of the algorithm comes from Lemma 3 and Lemma 4, which we prove in Section 3.2 and Section 3.3, respectively. Step 1 of the algorithm clearly takes linear time. Step 2 can also be completed in linear time by computing the distance from c_1 to all other centers in D. Step 3 takes linear time. The points P_1 to P_5 can be obtained in constant time after the transformation. Then checking whether these 5 points are sufficient takes linear time. If these 5 points are not sufficient, then by Step 5, we can compute a new set of 5 points that pierce $\mathcal{D}$ in constant time. □

We now present a definition that will be used in Section 3.2 and Section 3.3.

Definition 1 (Between) *Let A and B be two intersecting disks, and let p and q be two points in the plane. Let the center of A (resp. B) be a (resp. b). We say that A intersects B* ***between*** *p and q if the following two conditions hold:*

- *Line segment ab intersects line segment pq.*
- *Both p and q lie outside A.*

3.2 Proof for Lemma 3

Proof. Recall points P_1 to P_5 where $P_1 = (0,0)$, $P_2 = (\sqrt{3},0)$, $P_3 = (\frac{\sqrt{3}}{2},\frac{3}{2})$, $P_4 = (-\frac{\sqrt{3}}{2},\frac{3}{2})$, $P_5 = (-\sqrt{3},0)$; see Figure 10. We now argue that these 5 points pierce $\mathcal{D}$ when $r_2 \leq 5+2\sqrt{6}$. Let t_1 be the line with a positive slope that is tangent to D_1 and passing through P_2. The equation of t_1 is $t_1 = \frac{\sqrt{2}}{2}x - \frac{\sqrt{6}}{2}$. Let t_2 be the line with a negative slope that is tangent to D_1 and passing

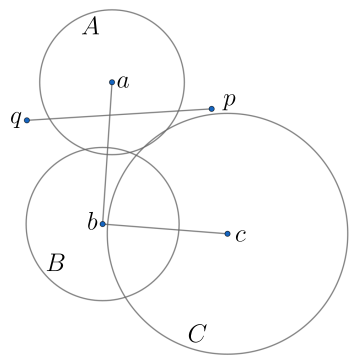


Figure 11: $\{A, B, C\}$ are three pairwise intersecting disks. A intersects B *between* p and q. C intersects B, but *not between* p and q since pq and bc do not cross.

through P_5. The equation of t_2 is $t_2 = -\frac{\sqrt{2}}{2}x - \frac{\sqrt{6}}{2}$. Since D_2 is centered on the positive y-axis, D_2 is tangent to both t_1 and t_2 when $r_2 = 5 + 2\sqrt{6}$. Therefore, when $r_2 \leq 5 + 2\sqrt{6}$, D_2 falls above t_1 and t_2.

We first prove that any disk whose center falls in the first or the second quadrant is pierced by P. Let $D_i \in \mathcal{D}$ be a disk with center c_i and radius r_i where c_i falls in the first or the second quadrant. Since D_1 is the smallest disk in $\mathcal{D}$, we have that $r_i \geq 1$. Since points P_2, P_3, P_4, P_5 are the vertices of a regular hexagon, there must exist a $j \in \{2, 3, 4, 5\}$ such that $\angle P_j P_1 c_i \leq \frac{\pi}{6}$. Let $\theta = \angle P_j P_1 c_i$. By the law of cosines,

$$|c_i P_j|^2 = |c_i P_1|^2 + |P_1 P_j|^2 - 2|c_i P_1||P_1 P_j|\cos(\theta) \quad (1)$$

$|P_1 P_j| = \sqrt{3}$ since these points all have distance $\sqrt{3}$ to the origin. $|c_i P_1| \leq r_i + 1$ since D_i and D_1 intersect. We have that $\cos(\theta) \geq \cos(\frac{\pi}{6})$ since $\theta \leq \frac{\pi}{6}$. Therefore, $-2|c_i P_1||P_1 P_j|\cos(\theta) \leq -2|c_i P_1||P_1 P_j|\cos(\frac{\pi}{6})$. By replacing terms in equation 1, we get

$$\begin{aligned} |c_i P_j|^2 &\leq |c_i P_1|^2 + (\sqrt{3})^2 - 2\sqrt{3}|c_i P_1|\cos(\frac{\pi}{6}) \\ &\leq |c_i P_1|^2 + 3 - 3|c_i P_1| \\ &\leq (|c_i P_1| - 1)^2 - |c_i P_1| + 2 \end{aligned} \quad (2)$$

When $|c_i P_1| \geq 2$, $(|c_i P_1| - 1)^2 - |c_i P_1| + 2 \leq r_i^2 + 2 - |c_i P_1| \leq r_i^2$. Therefore, $|c_i P_j| \leq r_i$ and D_i contains P_j. If $|c_i P_1| \leq 1$, c_i falls in D_1. Then D_i is pierced by P_1 since $r_i \geq 1$.

Now let us consider the case when $1 < |c_i P_1| < 2$. Let $f(x)$ be the parabola $x^2 - 3x + 3$. The vertex of $f(x)$ is $(\frac{3}{2}, \frac{3}{4})$. Therefore, when $1 < x \leq \frac{3}{2}$, $\frac{3}{4} \leq f(x) < 1$. Similarly, when $\frac{3}{2} \leq x < 2$, $\frac{3}{4} \leq f(x) < 1$. Combining these results together, we have that $f(x) < 1$ when $1 < x < 2$. Let $|c_i P_1| = x$, then we have that $|c_i P_j|^2 \leq f(x) < 1$. Therefore, $|c_i P_j| < 1$ and P_j pierces D_i since $r_i \geq 1$.

We now show that any disk in $\mathcal{D}$ whose center falls in the third or fourth quadrant is pierced by at least one of $\{P_1, P_2, P_5\}$. If all disks are pierced by at least one of these points, then we are done. So we assume that there exists at least one disk, say D_3, that is not pierced by any of these three points. Since D_2 lies completely above t_1 and t_2, D_3 must intersect D_2 between P_1 and P_2 or between P_1 and P_5. D_3's radius is at least 1 since otherwise it contradicts the assumption that D_1 is the smallest disk in $\mathcal{D}$. Then D_3 does not cross the $y = 1$ line. D_2 lies completely above the $y = 1$ line, so D_3 does not intersect D_2 and we have a contradiction. Therefore, any disk in $\mathcal{D}$ whose center falls in the third or fourth quadrant is pierced by one of $\{P_1, P_2, P_5\}$. $\square$

3.3 Proof for Lemma 4

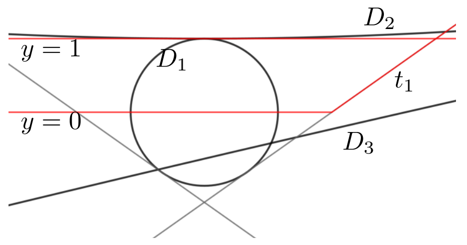
Proof. Recall the lines t_1, t_2, and the point set P from the proof of Lemma 3. Since $r_2 > 5 + 2\sqrt{6}$, D_2 intersects both t_1 and t_2. We assumed that there exists at least one disk, say $D_3 \in \mathcal{D}$ that is not pierced by P. D_3 intersects both D_1 and D_2. The center c_3 of D_3 cannot lie in the first or second quadrant since otherwise it must contain one point of P as was shown Section 3.2. Up to symmetry we may assume that the center c_3 lies in the fourth quadrant, and thus it intersects D_2 to the right side of the y-axis. This setting is depicted in Figure 12(a).

Since the interior of D_1 lies completely below the line $y = 1$ and the interior of D_2 lies completely above this line, any disk in $\mathcal{D} \setminus \{D_1, D_2\}$ must cross this line in order to intersect both D_1 and D_2. Since D_3 misses P, then D_3 must lie completely below the polygonal line

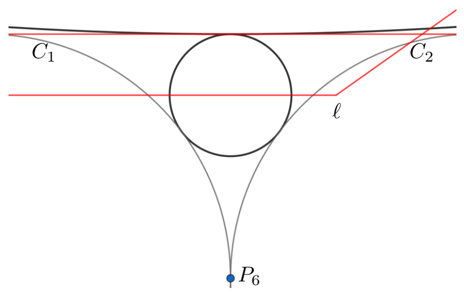
$$\ell : \begin{cases} y = 0, & x \leq \sqrt{3} \\ t_1, & x > \sqrt{3} \end{cases}$$

as shown in Figure 12(a). If D_3 crosses ℓ when $x \leq \sqrt{3}$, then either D_3 contains one of $\{P_1, P_2, P_5\}$ or it does not intersect with D_2. If D_3 crosses ℓ when $x > \sqrt{3}$, then either D_3 contains P_2 or it does not intersect with D_1. Therefore, any disk in $\mathcal{D}$ whose center falls above ℓ must cross ℓ in order to intersect with D_3.

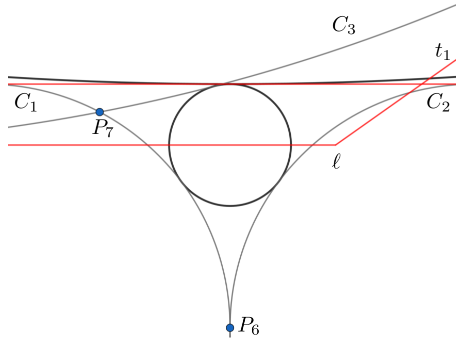
We are going to construct a point set $P' = \{P_6, P_7, P_8, P_9, P_{10}\}$ that pierces $\mathcal{D}$. Set $P_6 = (0, -3)$. In the rest of the proof we describe how to obtain P_7, P_8, P_9, and P_{10}; the coordinates of these points are given in Appendix B. Let C_1 (resp. C_2) be the circle passing through P_6 that is tangent to disk D_1 and line $y = 1$ in the left side (resp. right side) of the y-axis, as in Figure 12(b). Let C_3 be the circle that is centered above $y = 1$ and that is tangent to the disk D_1, the line t_1 and to the x-axis. The disks C_1 and C_3 intersect at two points, where we pick the intersection point that is closer to the origin as the point P_7; see Figure 12(c).



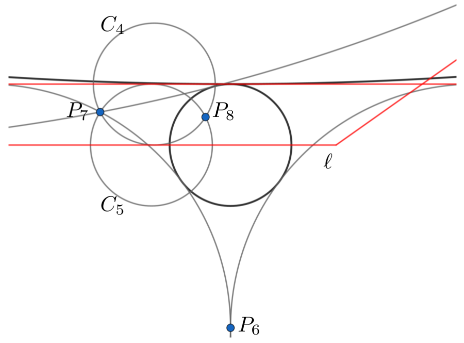
(a) Boundaries that disks in $\mathcal{D}$ must cross.



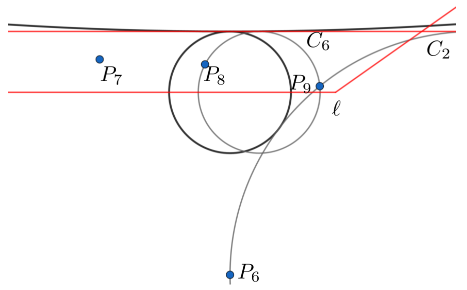
(b) Location of P_6.



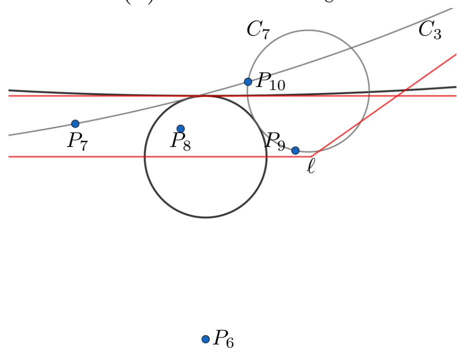
(c) Location of P_7.



(d) Location of P_8.



(e) Location of P_9.



(f) Location of P_{10}.

Figure 12: Illustration of the proof for Lemma 4.

Now let C_4 be a circle of radius 1 that passes though P_7 and that is tangent to the x-axis, and let C_5 be a circle of radius 1 that passes through P_7 and that is tangent to the the line $y = 1$. The point P_8 is the intersection point between C_4 and C_5 that is different from P_7. See Figure 12(d) for an illustration.

To obtain P_9, let C_6 be a circle of radius 1 that passes through P_8 and that is tangent to the line $y = 1$. The intersection point of C_2 and C_6 that falls in the first quadrant is P_9, as depicted in Figure 12(e). To obtain P_{10}, we draw a circle C_7 of radius 1 through P_9 and tangent to D_1. The point P_{10} is the intersection point of C_3 and C_7 that is closer to the origin, as in Figure 12(f).

Now that all five points in P' have been introduced, we are going to show that these five points pierce all disks $\mathcal{D}$. Consider the convex quadrilateral formed by P_6, P_7, P_9, and P_{10}, as in Figure 13. These four points pierce any disk of $\mathcal{D}$ whose center lies outside the quadrilateral, because any such disk must intersect D_1.

- C_3 is tangent to ℓ and D_1, and both P_7 and P_{10} lie on C_3. If a disk D_4 in $\mathcal{D}$ intersects D_1 between P_7 and P_{10}, D_4 cannot cross ℓ. Since D_3 lies completely below ℓ, D_4 does not intersect D_3 and it violates the pairwise intersecting property of $\mathcal{D}$.
- Both P_6 and P_7 lie on C_1, and C_1 is tangent to the $y = 1$ line. If a disk D_4 intersects D_1 between P_6 and P_7, then D_4 does not intersect D_2 and again contradicts our assumption that the disks in $\mathcal{D}$ are pairwise intersecting. Using a similar argument, we can also prove that there cannot exist a disk in $\mathcal{D}$ that intersects D_1 between P_6 and P_9.

- Any disk that intersects with D_1 between P_9 and P_{10} must contain one of these two points. Otherwise, its radius is smaller than 1, contradicting the fact that D_1 is the smallest disk.

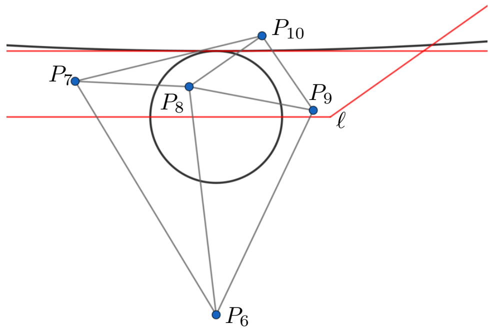


Figure 13: The points P_6, P_7, P_9, P_{10} form a quadrilateral that contains D_1.

Now we show how the disks of $\mathcal{D}$ centered inside the quadrilateral are pierced by points in P'. We divide the quadrilateral into four triangles, as in Figure 13.

- P_7 and P_8 both lie on C_5 and the radius of C_5 is 1. Therefore, any disk whose center lies in $\triangle P_6P_7P_8$ must contain one of P_7 or P_8 in order to intersect with D_2, otherwise its radius is smaller than 1.

- Similarly, P_7 and P_8 both lie on C_4 and the radius of C_4 is also 1. Therefore, any disk whose center lies in $\triangle P_7P_8P_{10}$ must contain one of P_7 and P_8 in order to intersect with D_3.

- Any disk whose center lies in $\triangle P_8P_9P_{10}$ must contain one of these three vertices because the diameter of this triangle is at most 2.

- Any disk whose center falls in $\triangle P_6P_8P_9$ must contain one of P_8 and P_9 in order to intersect D_2, otherwise its radius is smaller than 1 since C_6 has radius 1 and both P_8 and P_9 lie on C_6.

Given D_1, D_2, t_1, and t_2, the point set P' can be found in constant time. □

4 Conclusion

In this paper, we gave two simple linear time algorithms for finding 3 piercing points and 5 piercing points for pairwise intersecting unit disks and pairwise intersecting arbitrary disks, respectively. However, it is still not known whether we can find an algorithm for finding a piercing point set of size 4 for any set of pairwise intersecting arbitrary disks without solving an LP-type problem. For the lower bound, the remaining open question is whether any set of 9 pairwise intersecting disks can be pierced by 3 points or not, as it is known that any set of 8 pairwise intersecting disks can be pierced by 3 points [10]. Another interesting open question is whether we can find an efficient algorithm that decides the optimal number of piercing points for any set of pairwise intersecting arbitrary disks.

References

[1] P. Carmi, M. J. Katz, and P. Morin. Stabbing pairwise intersecting disks by four points. *CoRR*, abs/1812.06907, 2018.

[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 Scientiarum Mathematicarum Hungarica*, 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] S. Har-Peled, H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, M. Sharir, and M. Willert. Stabbing pairwise intersecting disks by five points. *Discret. Math.*, 344(7):112403, 2021.

[7] E. Helly. Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923.

[8] E. Helly. Über Systeme von abgeschlossenen Mengen mit gemeinschaftlichen Punkten. *Monatshefte für Mathematik*, 37(1):281–302, 1930.

[9] J. Matousek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[10] L. Stachó. Über ein Problem für Kreisscheibenfamilien. *Acta Scientiarum Mathematicarum (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.

A Coordinates of points in Theorem 1

Here are the coordinates of points in the proof of Theorem 1:

$$A = \left(x_3, \sqrt{4 - x_3^2} + r_1 - 1\right)$$

$$B = \left(x_3, -\sqrt{4 - x_3^2} + r_1 + 1\right)$$

$$P_1 = \left(x_3 - \sqrt{2\sqrt{4 - x_3^2} + x_3^2 - 4}, r_1\right)$$

$$P_2 = \left(x_3 - \frac{5}{4}, r_1 + \frac{1}{2}\right)$$

$$P_3 = \left(x_3 - \frac{5}{4}, r_1 - \frac{1}{2}\right)$$

B Coordinates of points in Lemma 4

For each point P_i, let x_i be its x-coordinate and y_i be its y-coordinate, and for each circle C_i, let (x'_i, y'_i) be its center and r'_i be its radius. Here are the coordinates of points P_i and equations of circles C_i:

$$P_6 = (0, -3)$$

$$C_1 : (x+4)^2 + (y+3)^2 = 16$$

$$C_2 : (x-4)^2 + (y+3)^2 = 16$$

$$C_3 : (x - x'_3)^2 + (y - y'_3)^2 = (r'_3)^2$$

$$x'_3 = -\sqrt{1 + 2r'_3}, y'_3 = r'_3$$

$$r'_3 = \frac{16 - 4\sqrt{6} + \sqrt{(16 - 4\sqrt{6})^2 - 16(\sqrt{6} - 2)^2}}{2(\sqrt{6} - 2)^2}$$

$$P_7 = \left(\frac{(-2r'_3 - 6)y_7 + (x'_3)^2 - 9}{2x'_3 + 8}, \frac{-b_7 + \sqrt{b_7^2 - 4a_7c_7}}{2a_7}\right)$$

$$a_7 = (-2r'_3 - 6)^2 + (2x'_3 + 8)^2$$

$$b_7 = 2(-2r'_3 - 6)\left((x'_3)^2 - 9\right) + 8(2x'_3 + 8)(-2r'_3 - 6) + 6(2x'_3 + 8)^2$$

$$c_7 = \left((x'_3)^2 - 9\right)^2 + 8(2x'_3 + 8)\left((x'_3)^2 - 9\right) + 9\left(2x'_3 + 8\right)^2$$

$$C_4 : \left(x - \sqrt{2y_7 - y_7^2} - x_7\right)^2 + (y-1)^2 = 1$$

$$C_5 : \left(x - \sqrt{1 - y_7^2} - x_7\right)^2 + y^2 = 1$$

$$P_8 = \left(\frac{2y_8 + q_1}{q_2}, \frac{-b_8 - \sqrt{b_8^2 - 4a_8c_8}}{2a_8}\right)$$

$$q_1 = \left(\sqrt{1 - y_7^2} + x_7\right)^2 - \left(-\sqrt{2y_7 - y_7^2} - x_7\right)^2 - 1$$

$$q_2 = 2\left(\sqrt{1 - y_7^2} + x_7\right) - 2\left(\sqrt{2y_7 - y_7^2} + x_7\right)$$

$$a_8 = 4 + q_2^2$$

$$b_8 = 4q_1 - 4q_2\left(\sqrt{1 - y_7^2} + x_7\right)$$

$$c_8 = q_1^2 + q_2^2\left(\sqrt{1 - y_7^2} + x_7\right)^2 - 2q_1q_2\left(\sqrt{1 - y_7^2} + x_7\right) - q_2^2$$

$$C_6 : \left(x - \sqrt{1 - y_8^2} - x_8\right)^2 + y^2 = 1$$

$$P_9 = \left(\frac{-b_9 + \sqrt{b_9^2 - 4a_9c_9}}{2a_9}, \frac{q_3x_9 + q_4}{6}\right)$$

$$q_3 = 8 - 2\left(\sqrt{1 - y_8^2} + x_8\right)$$

$$q_4 = \left(\sqrt{1 - y_8^2} + x_8\right)^2 - 10$$

$$a_9 = 36 + q_3^2$$

$$b_9 = 2q_3q_4 + 36q_3 - 288$$

$$c_9 = q_4^2 + 36q_4 + 324$$

C_7 is centered at

$$\left(\sqrt{4 - (y'_7)^2}, \frac{-b_{10} + \sqrt{b_{10}^2 - 4a_{10}c_{10}}}{2a_{10}}\right)$$

$$a_{10} = 4x_9^2 + 4y_9^2$$

$$b_{10} = -4y_9(x_9^2 + y_9^2 + 3)$$

$$c_{10} = \left(x_9^2 + y_9^2 + 3\right)^2 - 16x_9^2$$

$$P_{10} = \left(x'_7 - \sqrt{1 - (y_{10} - y'_7)^2}, \frac{-b_{11} - \sqrt{b_{11}^2 - 4a_{11}c_{11}}}{2a_{11}}\right)$$

$$q_5 = (x'_7)^2 + (y'_7)^2 - (x'_3)^2 - (y'_3)^2 + (r'_3)^2 - 1 - (2x'_7 - 2x'_3)x'_7$$

$$a_{11} = (2y'_3 - 2y'_7)^2 + (2x'_7 - 2x'_3)^2$$

$$b_{11} = 2q_5(2y'_3 - 2y'_7) - 2y'_7(2x'_7 - 2x'_3)^2$$

$$c_{11} = q_5^2 + \left((y'_7)^2 - 1\right)\left(2x'_7 - 2x'_3\right)^2$$

Extensions of the Maximum Bichromatic Separating Rectangle Problem

Bogdan Armaselu [*]

Abstract

In this paper, we study two extensions of the maximum bichromatic separating rectangle (MBSR) problem introduced in [2, 4]. One of the extensions, introduced in [3], is called *MBSR with outliers* or MBSR-O, and is a more general version of the MBSR problem in which the optimal rectangle is allowed to contain up to k outliers, where k is given as part of the input. For MBSR-O, we improve the previous known running time bounds of $O(k^7 m \log m + n)$ to $O(k^3 m + m \log m + n)$. The other extension is called *MBSR among circles* or MBSR-C and asks for the largest axis-aligned rectangle separating red points from blue unit circles. For MBSR-C, we provide an algorithm that runs in $O(m^2 + n)$ time.

1 Introduction

In this paper, we consider two extensions of the Maximum Bichromatic Separating Rectangle (MBSR) problem.

The MBSR problem, introduced in [2] (see also [4]), is stated as follows. Given a set of red points R and a set of blue points B in the plane, with $|R| = n, |B| = m$, compute the axis-aligned rectangle S having the following properties:

(1) S contains all points in R,

(2) S contains the fewest points in B among all rectangles satisfying (1),

(3) S has the largest area of all rectangles satisfying (1) and (2).

We call such rectangle a *maximum bichromatic separating rectangle* (MBSR) or simply *largest separating rectangle.*

Let S_{min} be the smallest axis-aligned rectangle enclosing R and discard the blue points inside S_{min}, as they cannot be avoided.

The first extension of MBSR, called *MBSR with outliers* (MBSR-O) or simply *outliers version*, was introduced in [3], and asks for the largest axis-aligned rectangle containing all red points and up to k blue points outside S_{min}, where k is given as part of the input. That is, MBSR with outliers is a relaxation of condition (2) of the original MBSR problem.

In this paper, we introduce another extension of MBSR, called *MBSR among circles* (MBSR-C) or simply *circles version*, where blue points are replaced by blue unit circles, and the goal is to find the largest rectangle containing all red points and no point of any blue circle outside S_{min}. Here we may also discard blue circles intersecting S_{min} from consideration, as they cannot be avoided.

For both extensions, we assume that all points are in general position and that an optimal bounded solution exists, that is, there are no unbounded solutions.

The circles version is motivated by problems involving "imprecise" data, such as tumor extraction with large or imprecise cells as red or blue points, or machine learning applications with probabilistic, rather than deterministic training data.

The outliers version can have applications in various domains. For instance, in VLSI or circuit design, the goal is to place a hardware component (e.g., cooler) on a board with minor fabrication defects (blue points), where up to k defects are acceptable to be covered by the component. Here the red points may denote "hot spots" that must be covered or isolated from the rest of the board.

Other applications of MBSR extensions can be in machine learning, data science, or spatial databases.

1.1 Related Work

Geometric separability of point sets, which deals with finding a geometric locus that separates two or more point sets whilst achieving a specific optimum criterion, is an important topic in computational geometry. Various approaches deal with finding a specifc type of separator (e.g., hyperplane) when the points are guaranteed to be separable. However, this is not always the case, and there is related work on weak separability, i.e., either allowing a fixed number of misclassifications or minimizing them.

Bitner and Daescu et. al [6] study the problem of finding the smallest circle that separates red and blue points (i.e., contains all red points and the fewest blue points). They provide two algorithms. The first of them runs in $O(mn \log m + n \log n)$ time and the second runs in $O(m^{1.5} \log^{O(1)} n + n \log n)$ time. Both algorithms enumerate all optimal solutions. Later, Armaselu and Daescu [5] addressed the dynamic version of the problem, in which blue points may be

[*]Work started as a student of Department of Computer Science, University of Texas at Dallas, `barmaselub@gmail.com`

inserted or removed dynamically, and provided three data structures. The first one supports insertion and deletion queries in $O(n \log m)$ time, and can be updated in $O((m+n) \log m)$ time. The other two are insertion-specific (resp., deletion-specific) and allow poly-logarithmic query time, at the expense of $O(mn \log(mn))$ update time.

The problem of computing an MBSR was considered by Armaselu and Daescu. For the case when the target rectangle has to be axis-aligned, the algorithm runs in $O(m \log m + n)$ time [2, 4]. When the target rectangle is allowed to be arbitrarily oriented, an $O(m^3 + n \log n)$ time algorithm is given, and they also provide an algorithm to find the largest separating box in 3D in $O(m^2(m+n))$ time [2].

Separability of imprecise points, where points are asscoiated with an imprecision region, has also been considered. Note that, for the problem addressed in this paper, the blue circles can be thought of as imprecision regions. When the imprecision regions are axis-aligned rectangles, de Berg et. al [11] come up with algorithms to find certain separators (with probability 1) and possible separators. For certain separators, their algorithm runs in linear time, while for possible separators, the running time is $O(n \log n)$.

Armaselu, Daescu, Fan, and Raichel considered extensions of the MBSR problem. Specifically, they give an approach to find a largest rectangle separating red points from blue axis aligned rectangles in $O(m \log m + n)$ time, as well as an approach for the largest rectangle separaing red points from blue points with k outliers in $O(k^7 m \log m + n)$ time [3].

A very popular related problem is the one of computing the largest empty (axis-aligned) rectangle problem. Given a set of planar points P, the goal is to compute the largest P-empty (axis-aligned) rectangle that has a point $p \in P$ on each of its sides. For the axis-aligned version, the best currently known bound for computing *one* optimal solution is $O(n \log^2 n)$ time by Aggarwal et. al [1]. Mukhopadhyay et. al [8] solve the version where the rectangle can be arbitrarily oriented. They provide an $O(n^3)$ time algorithm that lists all optimal solutions. Chaudhuri et. al [7] prove that there can be $\Omega(n^3)$ optimal solutions in the worst case.

Nandy et. al considered the problem of finding the maximal empty axis-aligned rectangle among a given set of rectangles isothetic to a given bounding rectangle [9]. They show how to solve the problem in $O(n \log n + R)$ time, where R is the number of rectangles. Later, they solved the version where obstacles have arbitrary orientation using an algorithm that takes $O(n \log^2 n)$ [10].

1.2 Our Results

We first improve the result in [3] for the outlier version. Specifically, we first give a slight improvement that runs in $O(k^7 m + m \log m + n)$ time for k outliers, and then a further improvement to $O(k^3 m + km \log m + n)$ time (which works when $k > (\log m)^{\frac{1}{4}}$). We also solve the circles version and provide an algorithm that runs in $O(m^2 + n)$ time.

The rest of the paper is structured as follows. In Section 2, we describe our improvements to MBSR-O, then in Section 3 we describe our algorithm for MBSR-C. Finally, in Section 4 we draw the conclusions and list the future directions.

2 Finding the Largest Separating Rectangle with k Outliers

Given an integer $k \geq 0$, the goal is to find the largest axis-aligned rectangle enclosing R that contains at most k blue points in B. We call this the maximum bichromatic separating rectangle with k outliers (MBSR-O).

The approach in [3] is, in a nutshell, as follows. First, compute the smallest R-enclosing rectangle S_{min} in $O(n)$ time. The lines defining S_{min} partiton the plane into 8 regions outside S_{min}. Namely, the four "side" regions E, N, W, S and the four corner regions (quadrants) NE, NW, SW, SE. For each region q, denote by B_q the set of blue points inside q.

Definition 1 *[3] A point $p \in B_{NE}$ dominates another point $q \in B_{NE}$, if $x(p) > x(q)$ and $y(p) > y(q)$.*

Definition 2 *[3] For each B_q and for any t such that $0 \leq t \leq k$, the t-th level staircase of B_q is the rectilinear polygon formed by the blue points in B_q that dominate exactly t blue points in B_q.*

Note that an optimal solution contains t points from B_q if and only if it is sbounded by the t-th level staircase of B_q. See figure 1 for an illustration of a staircase.

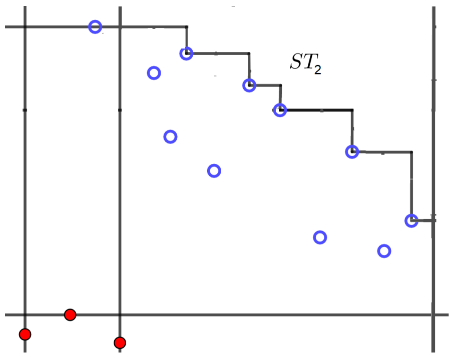


Figure 1: The 2nd level staircase of B_{NE}

For each partition of the number k into 8 smaller integers (each corresponding to a region) $k = k_E + k_{NE} + \cdots + k_{SE}$, do the following.

1. consider the $k_q + 1$-th closest to S_{min} blue point from each side region q. These points support a rectangle S_{max} which has to contain the target rectangle.

2. compute the k_q-level staircase $ST_{k_q}(q)$ of each quadrant q in $O(m \log m)$ time.

3. Solve a "staircase" problem, i.e., find the largest rectangle containing S_{min}, included in S_{max} and being supported by points of the staircases. This is done in $O(m)$ time.

There are $O(k^7)$ such partitions of k, so the running time of $O(k^7 m \log m + n)$ follows.

2.1 A Slight Improvement on the Running Time

To reduce the running time, we first prove the following lemma.

Lemma 1 *The t-level staircases $ST_t(q)$ can be computed in $O(m \log m + mk)$ time for all $t \leq k$ and for all quadrants q.*

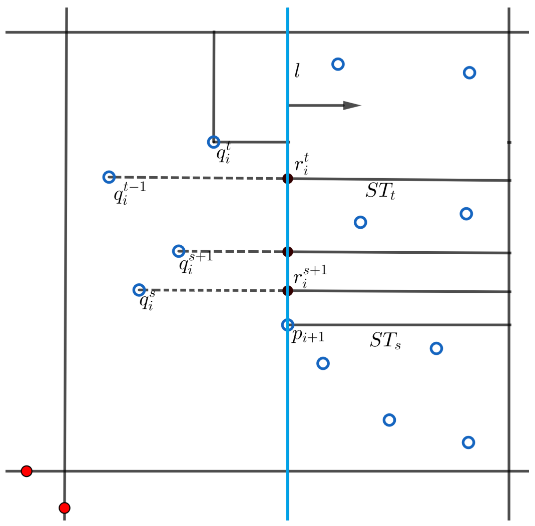


Figure 2: ST_t is updated while sweeping vertical line l over point p_{i+1}. Since $q_i^s, \ldots, q_i^t$ are higher than p_{i+1}, the Y coordinate of ST_t is changed to that of r_i^t, the projection of q_i^{t-1} on l. Similarly, the Y coordinates of $ST_{t-1}, \ldots, ST_s$ are changed to those of $r_i^{t-1}, \ldots, r_i^s$, respectively

Proof. We show how to compute $ST_t(NE)$ for all $t \leq k$ with a sweep line algorithm, as for other quadrants the approach is similar. For simplicity, denote $ST_t(NE)$ as ST_t, that is, without specifying the quadrant. Sort and label the points in B_{NE} by increasing x-coordinate, and denote the resulting sequence as $p_1, \ldots, p_m$. Sweep a vertical line l from $x_0 = \max\{x | (x, y) \in S_{min}\}$ to $x_1 = \infty$. For any given position of l, let $P_l = \{p_1, \ldots, p_i\}$ be the set of blue points to the left of l. We maintain a balanced binary search tree T over P_l, indexed by the y-coordinates of its elements. The intersection of ST_t with l is a single point q_i^t, which is the highest point on l that lies above at most t points of P_l. That is, q_i^t is the $(t+1)$-th smallest indexed entry in T, which we record. As we move l from left to right, q_i^t can only change when l intersects a point $p_i \in B_{NE}$. When we cross the point p_{i+1} (called *event point*), we insert it into T and, for each $t \leq k$, we have two cases.

1. If p_{i+1} is higher than q_i^t, then ST_t does not change height.

2. If p_{i+1} is lower than $q_i^t, q_i^{t-1}, \ldots, q_i^s$, for some $0 \leq s \leq t$, then q_i^t is set to q_i^{t-1} (which is done in $O(1)$ time), and ST_t moves to the height of this entry. This update is repeated by setting q_i^{t-1} to q_i^{t-2} and so on downto q_i^{s+1}. Finally, q_i^s is set to p_{i+1}.

If ST_t changes when sweeping over p_{i+1}, we also record a point r_i^t whose x-coordinate is that of p_{i+1} and whose y-coordinate is that of the updated q_i^t. Let Q be the set of all such points q_i^t, r_i^t recorded during this process for all $t \leq k, i = 1, \ldots, m$. It is not hard to argue that $\cup_{t \leq k} ST_t \subseteq Q$. Moreover, $|Q| \leq m$ holds, as a point is added to Q only when the sweep line crosses a point of B_{NE}. Finally, note that we encountered $O(m)$ event points p_i. For each of them, we spend $O(\log m)$ time to insert them into T and $O(k)$ time for q_i^t, r_i^t pointer updates. Thus, the running time bound follows. $\square$

Figure 2 illustrates the proof of Lemma 1.

Rather than computing the ST_t's for each partition of k, we compute them all in $O(m \log m + mk)$ time before considering such partitions. Then, for each of the $O(k^7)$ partitions, we need only solve a staircase problem in $O(m)$ time. This gives us the following result.

Theorem 2 *Given two sets R, B, with $|R| = n, |B| = m$, the largest rectangle enclosing R and containing at most k points in B can be found in $O(k^7 m + m \log m + n)$ time.*

2.2 A Closer Look at the Number of Candidate Partitions of k

In the previous section, we reduced the running time by a factor of $\log m$. However, it seems hard to further improve this bound given the high number of partitions of k. In this subsection, we show how to reduce the running time by reducing the number of candidate partitions of k.

To do that, we first compute all the t-level staircases $ST_t, 0 \leq t \leq k$ as described in the previous section. We then consider the blue points in 4 pairs of adjacent regions, e.g., N and NE. That is, we suppose the total number of outliers coming from $B_N \cup B_{NE}$, denoted $k_{NNE} = k_N + k_{NE}$, is fixed. Similarly, we suppose $k_{ESE} = k_E + k_{SE}, k_{SSW} = k_S + k_{SW}, k_{WNW} = k_W + k_{NW}$ are fixed. Let $ST(Q) = \cup_{t=1}^{k} ST_t(Q)$, for any quadrant Q. From now on, we focus on the N and NE regions and, for simplicity, we denote $ST(NE)$ as simply ST and $ST_t(NE)$ as simply ST_t.

We notice that, even though any points of any t-th level staircase, $t \le k_{NE}$, may be a corner for a candidate rectangle, most of these rectangles can be discarded as they are guaranteed to be smaller than the optimal rectangle.

For every pair $P = (P_1, P_2)$ of regions and every integer $t : 0 \le t \le k$, denote by S_t^P the set of pairs $(p, q) \in (B_{P_1} \cup B_{P_2})^2$ that may define an optimal solution, with p as top support and q with right support, among all rectangles containing t blue points from $B_{P_1} \cup B_{P_2}$. From now on, whenever understood, we are going to remove the superscript and simply write S_t, e.g., $S_{k_{NNE}}$ instead of $S_{k_{NNE}}^{NNE}$. For every t, we store S_t as an array.

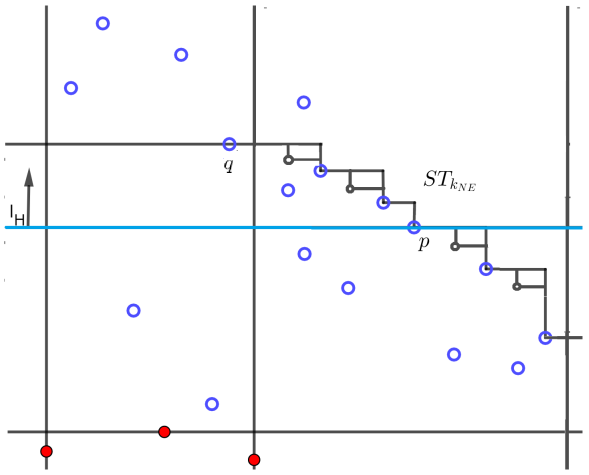


Figure 3: $B_N \cup B_{NNE}$ is swept with a horizontal line l_H sliding upwards from the lowest blue point. At each blue point p encountered, the set $S_{k_{NNE}}$ is updated. Black dots denote staircase points that are not blue points.

We extend the above definition to points $p, q \in B_q$ for other quadrants q, by flipping inequalities accordingly.

The goal is to compute $S_{k_{NNE}}$. Suppose we have already computed all $S_t : t < k_{NNE}$. Sweep $B_N \cup B_{NE}$ with a horizontal line l_H going upwards, starting at the $k_{NNE} + 1$-th lowest blue point in $B_N \cup B_{NE}$, as shown in Figure 3. For every blue point p encountered as top support, let $below(p)$ be the highest point in B_N below p, and $above(p)$ be the lowest point in B_N above p. For every $p \in B_{NE}$, let $rb(p)$ be the leftmost point in B_{NE} to the right of p and below p. Let t_N be the blue point count below p from B_N. Also let t be the number of points dominated by p from $B_N \cup B_{NE}$.

First, assume $p \in B_{NE}$ and let $t_{NE} = t - t_N$ be the number of points dominated by p from B_{NE}, i.e., $p \in ST_{t_{NE}}$. When sweeping the next blue point, say q, we do the following.

Case 1. If q is to the right of p, then q is below $above(p)$ but dominates p, the points domiated by p, and the points in $T(p, q) = \{s \in B_{NE} \cap ST_{t_{NE}} | x(p) < x(s) < x(q)\}$ (Figure 4). If $t + t_{pq} = k_{NNE}$, then we add $(q, rb(q))$ to $S_{t+t_{pq}}$, where $t_{pq} = 1 + |T(p, q)|$.

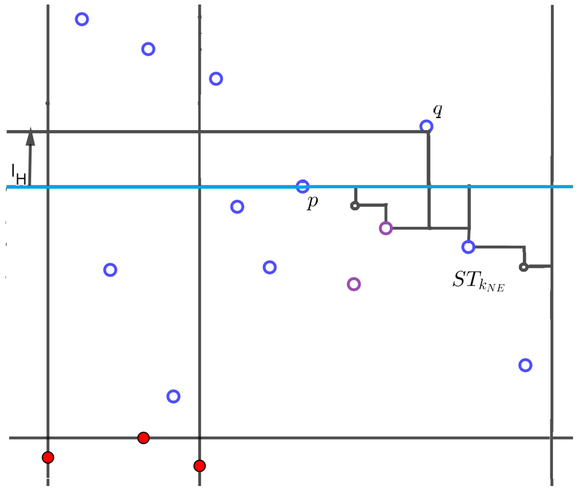


Figure 4: Case 1. $p \in B_{NE}$, q to the right of p. The purple empty dots denote $T(p, q)$.

Case 2. If $q \in B_{NE}$ and q is to the left of p, then q is below $above(p)$ but dominates the points dominated by p, except the ones in $U(q, p) = \{s \in B_{NE} | x(q) < x(s) < x(p), y(s) < y(p)\}$. For each $s \in U(q, p)$, if $t - i(s) = k_{NNE}$, then we add (q, s) to $S_{t-i(s)}$, where $i(s)$ is the index of s in $U(q, p)$ in decreasing order of X coordinates. Finally, if $t = k_{NNE}$, then we add (q, p) to S_t.

Case 3. If $q \in B_N$ then, for each $s \in U(p) = \{s \in B_{NE} | x(s) < x(p), y(s) < y(p)\}$ such that $t - i(s) = k_{NNE}$, we add (q, s) to $S_{t-i(s)}$, where $i(s)$ is the index of s in $U(p)$ in decreasing order of X coordinates. Finally, if $t = k_{NNE}$, then we add (q, p) to S_t.

Now assume $p \in B_N$ and let t_{NE} be the largest t' such that all points in any $ST_{t'}$ are below p. Let $b(p)$ be the leftmost point of $ST_{t_{NE}}$ below p. When sweeping the next blue point, say q, we do the following.

Case 4. If q is to the right of $b(p)$ then, if $t = k_{NNE} - 1$, we add $(q, rb(q))$ to S_{t+1}. For each $s \in U(q)$ such that $t - i(s) = k_{NNE}$, we also add (q, s) to $S_{t-i(s)}$.

Case 5. If $q \in B_{NE}$ and q is to the left of $b(p)$, then, if $t = k_{NNE} - 1$, we add (q, s) to S_{t+1} for every $(p, s) \in S_t$.

Case 6. If $q \in B_N$, then, if $t = k_{NNE} - 1$, we add (q, s) to S_{t+1} for every $(p, s) \in S_t$.

The following lemma puts an upper bound on the storage required by $S_{k_{NNE}}$.

Lemma 3 $|S_{k_{NNE}}| = O(m)$.

Proof. In case 1, we only one pair to $S_{k_{NNE}}$. In case 2, even though we consider $|U(q,p)|$ points, we only add the pair (q,s) such that $t - i(s) = k_{NNE}$. Similarly, in cases 3 and 4 we only add the pair $(q,s) : t - i(s) = k_{NNE}$, even though we consider $|U(p)|$ (resp., $|U(q)|$) points. In case 5, we add at most $|ST_{t_{NE}}|$ pairs if $t = k_{NNE} - 1$. However, note that for the subsequent point q' swept, we would have a larger numbert$'$ of blue points in $B_N \cup B_{NE}$ dominated by q'. Thus, we only add at most $|ST_{t_{NE}}| = O(m)$ pairs once. Similarly, in case 6 we only add $O(m)$ pairs once. □

This lemma bounds the running time of the aforementioned sweeping approach.

Lemma 4 *For any $t : 0 \leq t \leq k$, the horizontal line sweeping desccribed above takes $O(m \log m)$ time.*

Proof. We store the blue points in $B_N \cup B_{NE}$ in two balanced binrary search trees X, Y, indexed by X (resp., Y) coordinates. Thus, for each blue point p swept, we require $O(\log m)$ time. We require an extra $O(\log m)$ time to compute $above(p), below(p)$, and $rb(p)$. In case 1, note that we can compute t_{pq} by finding the position of q in the X-sorted order of $ST_{k_{NE}}$, and thus the number of blue points $s : x(p) < x(s) < x(q)$, in $O(\log m)$ time, since $ST_{k_{NE}}$ is maintained as a binary search tree. Thus, we only require an extra $O(\log m)$ time to handle case 1. In cases 2 and 4, note that we only need to add (q,s) to $S_{k_{NNE}}$ if $i(s) = t - k_{NNE}$, so we query for s in X using $O(\log m)$ time. Similarly, in case 3 we only add (q,p) to $S_{k_{NNE}}$ if $i(s) = t - k_{NNE}$, so we query X for s in $O(\log m)$ time. Now in cases 5 and 6 we spend $O(m)$ time to traverse S_t, since we store S_t as an array for any t, but they only occur once, so this gives us $O(m)$ total time. In every case, since S_t is an array, adding a pair to S_t takes $O(1)$ time. Since we sweep $O(m)$ blue points, the result follows. □

Corollary. We compute S_t in $O(km \log m)$ time for all $t : 0 \leq t \leq k$. This holds for any quadrant.

We reduce the number of candidate partitions of k from $O(k^7)$ to $O(k^3)$ as follows. By writing $k = k_{NNE} + k_{ESE} + k_{SSW} + k_{WNW}$, we can deduce $k_{NNE} = k - k_{WNW} - k_{ESE} - k_{SSW}$ for every combination of $k_{WNW}, k_{ESE}, k_{SSW}$. Therefore, there are $O(k^3)$ such combinations.

Initially, we compute S_t^Q for every quadrant Q and $0 \leq t \leq k$. Then, for each combination $(k_1, k_2, k_3), k_1, k_2, k_3 = 0, \ldots k, k_1 + k_2 + k_3 \leq k$, we set $k_4 = k - k_1 - k_2 - k_3$ and solve the staircase problem in [2] with the pairs $S_{t_1}^{WNW} \cup S_{t_2}^{ESE} \cup S_{t_3}^{SSW} \cup S_{k_4}^{NNE}$ as pairs of supports. Each staircase problem takes $O(m)$ time to solve, so we require $O(k^3 m)$ time for all candidate partitions of k. Putting this together with the result in Lemma 1, we get the following result.

We obtain the following result.

Theorem 5 *Given two sets R, B, with $|R| = n, |B| = m$, the largest rectangle enclosing R and containing at most k points in B can be found in $O(k^3 m + km \log m + n)$ time.*

3 Finding the Largest Axis Aligned Rectangle Enclosing R and avoiding Unit Circles

In this version, called the *circles* version (MBSR-C), B consists of unit circles that do not intersect S_{min}, and the goal is to find the largest axis-aligned rectangle enclosing R that avoids all circles.

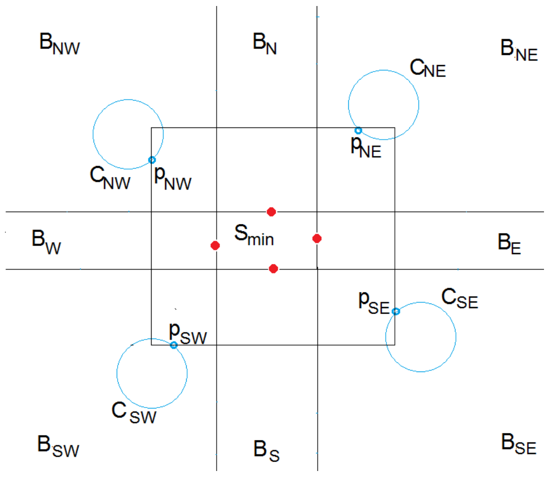


Figure 5: If the maximum rectangle separating R and B' were bounded by point $p \in C$, it would intersect C

Note that the reduction in [3] for finding the largest separating rectangle among rectangles does not work. To see why this is the case, let $C_{NW}, C_{SW}, C_{SE}, C_{NE}$ be circles in the regions $B_{NW}, B_{SW}, B_{SE}, B_{NE}$. If one picks any point p on the quadrant of C_{NW} that is the closest to S_{min}, and adds it to B' (and does the same for C_{SW}, C_{SE}, C_{NE}), then, if the maximum rectangle separating R and B' is top-bounded or right-bounded by p, it intersects C_{NW} (as shown in Figure 5).

A *candidate separating rectangle* (CSR) is a rectangle that encloses R and cannot be extended in any direction without intersecting some circle. Notice that a CSR may touch a circle either at an edge or at a corner. If it is bounded at an edge, then that edge is fixed in terms of X or Y coordinate and the arc it touches at each endpoint of the edge is uniquely determined (Figure 6). On the other hand, if it is bounded at a corner, then the corner can be slided along the appropriate arc of the circle (Figure 7). Each position of the corner determines the X or Y coordinates of its two adjacent edges, and thus the arcs pinning the two adjacent corners, if any.

We say that an edge e of a CSR is *pinned* by a circle C, if C touches the interior of e.

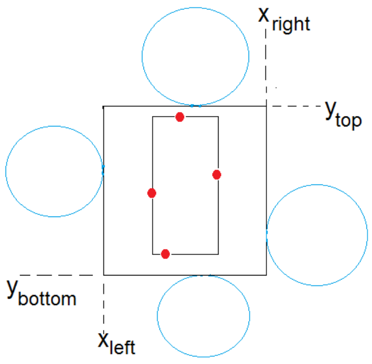


Figure 6: A circle bounding the rectangle at an edge fixes that edge in terms of X or Y coordinate

A horizontal (resp., vertical) edge e is said to be *fixed* by two circles C_1, C_2 in terms of Y (resp., X) coordinate, if:

(1) the ends of e are on C_1 and C_2, respectively, and

(2) changing its Y (resp., X) coordinate would result in either e intersecting C_1 or C_2 or failing to touch both C_1 and C_2.

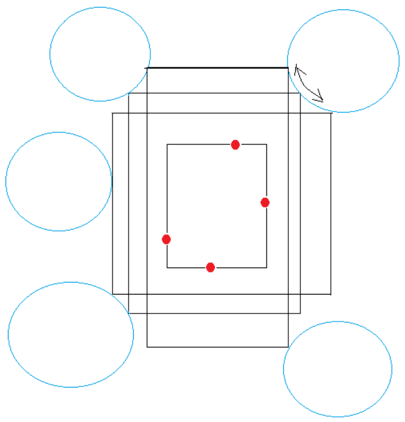


Figure 7: A circle bounding the rectangle at a corner allows the corner to slide along the arc. Each position of the corner determines its two adjacent edges

3.1 A description of all cases in which a CSR can be found

Based on the number of edges of a CSR pinning by circles, we consider the following cases.

Case 1. Three edges pinned by circles (Figures 8). In this case, we extend the the fourth edge outward from S_{min} until it touches a circle. Thus, the CSR is uniquely determined.

Case 2. Two edges are pinned by two circles C_1, C_2. Here we distinguish the following subcases.

Case 2.1. Two adjacent edges are pinned by C_1, C_2. Note that their common corner q is fixed. We extend

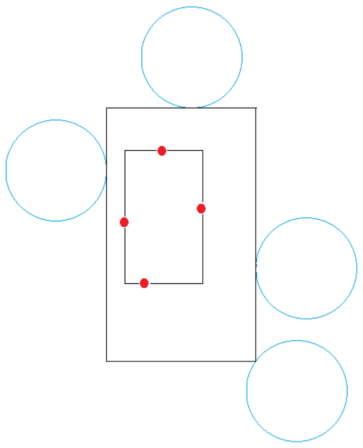


Figure 8: Case 1

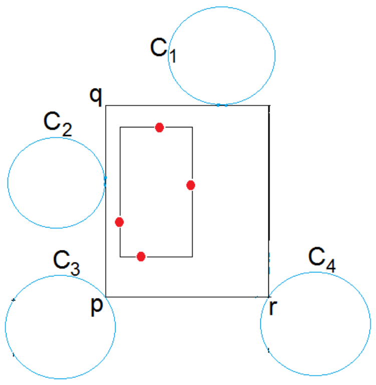


Figure 9: Case 2.1

one of the edges by moving its other end p away from q, until it touches a circle C_3, and then extend the third edge until it touches a circle C_4 at a point r (Figure 9). The resulting CSR is unique.

Case 2.2. Two adjacent edges are pinned by C_1, C_2. We extend one of the edges by moving its other end p away from q, until the orthogonal line through p touches a circle C_3 at a point r. While moving p, the point r can slide along one or more circles in the same quadrant, giving an infinite number of CSRs.

Case 2.3. Two opposite edges are pinned by C_1, C_2. We slide the other two edges outward from S_{min} until each of them touches some circle. This gives us a unique CSR.

Case 3. One edge e is pinned by a circle C_1. We have the following subcases.

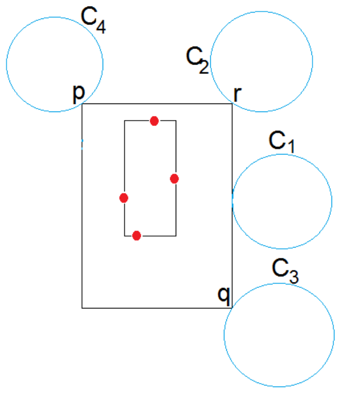


Figure 10: Case 3.1

Case 3.1. When e is extended in both directions, it touches two circles C_2, C_3 (Figure 10). We then slide the fourth edge outward from S_{min} until it touches a circle C_4 and we have a unique CSR.

Case 3.2. When e is extended in both directions, the orthogonal line through one of the ends p touches a circle C_2 at point q. While moving p, q can slide along one or more circles in the same quadrant, yielding an infinite number of CSRs. After establishing the position of q, we slide the fourth edge away from S_{min} until it touches a circle C_3.

Case 4. No edge is pinned by any circle. In this case, all corners can slide along circles until one of the edges becomes pinned by some circle, giving an infinite number of CSRs. Suppose the position of a corner p along a circle $C_1 \in B_{NE}$ is known. We consider the following subcases.

Case 4.1. While extending the CSR in the two directions away from p, the CSR touches a circle in B_{SE} or B_{NW} at some point q before touching any circle in B_{SW} (Figure 11). The other two corners are determined by sliding the edge opposite to pq outwards until it touches a circle at some point r. In this case, the CSR

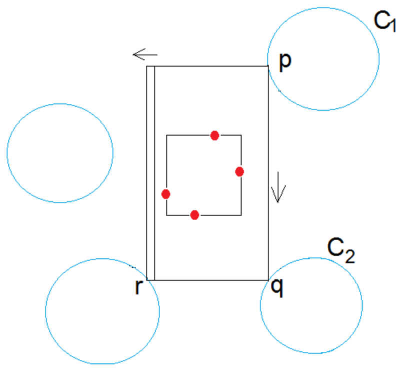


Figure 11: Case 4.1

is uniquely defined.

Case 4.2. While extending the CSR in the two directions away from p, the first circle that CSR touches, at a point q, is located in B_{SW}. This gives us an infinite number of CSRs.

3.2 Dominating envelopes

Definition 3 *For each quadrant B_i, the **dominating envelope** of B_i is a curve $\mathcal{C}$ with the following properties:*

1) $\mathcal{C}$ lies inside B_i;

2) for any point $p \in \mathcal{C}$, the rectangle cornered at p and the closest corner of S_{min} from p is empty;

3) extending $\mathcal{C}$ away from S_{min} violates property (2).

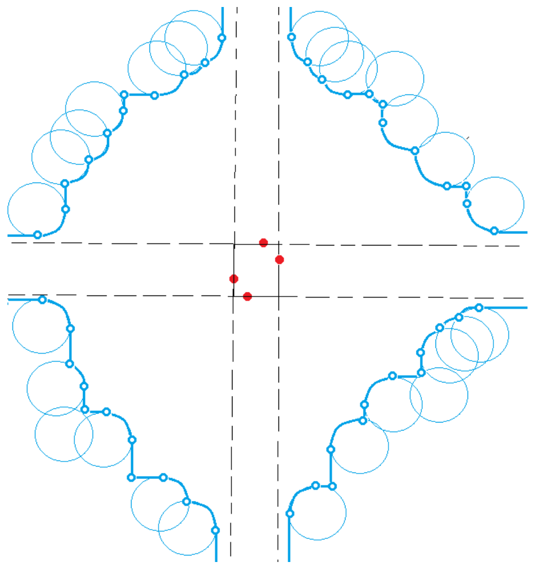


Figure 12: In each region, the circles define a dominating envelope $\mathcal{C}$, which is a sequence of arcs and horizontal or vertical segments. Two consecutive arcs or segments define a breakpoint on $\mathcal{C}$.

Note that $\mathcal{C}$ is a sequence of circle arcs, horizontal, and vertical segments, plus a horizontal and a vertical infinite ray (see Figure 12). Its use will be revealed later on.

The dominating envelope changes direction at *breakpoints*, which can be between two consecutive arcs, segments, or infinite ray. Every two consecutive breakpoints define a *range of motion* for a CSR corner.

A breakpoint p is said to be a *corner breakpoint*, if a CSR cornred at p cannot be extended away from S_{min} in all directions without crossing some circle, even if its other corners are not located on any envelope.

To compute the dominating envelope $\mathcal{C}$ of B_{NE}, we do the following. First, sort the circles by X coordinate of their centers. Let p be the current breakpoint (initially, the first breakpoint is the left endpoint l of the left-most circle, with a vertical infinite ray upwards from l). For each two adjacent circles $C_1(c_1, 1), C_2(c_2, 1)$, depending on the relative positions of c_1, c_2, we do the following.

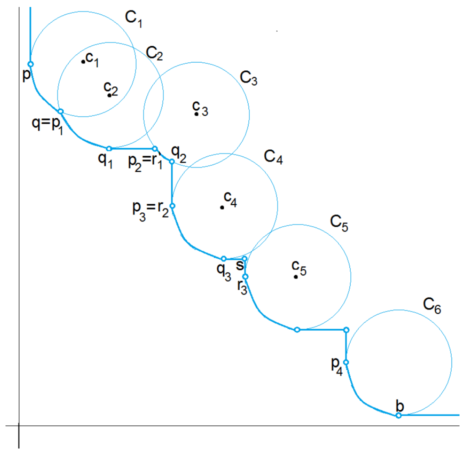


Figure 13: Each pair of adjacent circles gives a different case.

Case A. $c_2 \in C_1$. In this case, we add the lower intersection between C_1 and C_2 as a new corner breakpoint q, along with the arc pq of C_1.

Case B. $y(c_1)-1 \leq y(c_2) \leq y(c_1)$ and $x(c_2) > x(c_1)+1$. We add a breakpoint q at the bottom of C_1, the arc pq of C_1, a corner breakpoint r at the intersection between the horizontal through q and C_2, and the line segment qr.

Case C. $x(c_1) + 1 \leq x(c_2) \leq x(c_1) + 1$ and $y(c_2) < y(c_1) - 1$. We add a breakpoint r at the left endpoint of C_2, a corner breakpoint q at the intersection between the vertical through r and C_1, the line segment qr, and the arc pq of C_1.

Case D. $x(c_2) > x(c_1) + 1$ and $y(c_2) < y(c_1) - 1$. We add the breakpoints q at the bottom of C_1, r at the left end of C_2, the *corner breakpoint* s at the intersection between the horizontal through q and the vertical through r, the arc pq of C_1, and the segments qs and sr. Figure 13 illustrates this process.

Note that deciding the case a circle belongs to can be done in $O(1)$ time per circle.

3.3 Finding an optimal solution in each case

To find an optimal solution in each case, we do the following.

First, slide the edges of S_{min} outward until each of them touches a circle. If this is not possible, then the solution is unbounded, so we will assume that each edge will eventually hit a circle. Denote the resulting rectangle by S_{max} and discard the portions outside S_{max} from the dominating envelopes of all quadrants. The endpoints of the resulting envelopes are also counted as breakpoints.

For each case described in Section 3.1, we give a different algorithm to compute the largest separating rectangle. Before considering any case, we sort the blue circles by the X coordinate and then (to break ties) by the Y coordinate of their centers.

Case 1.

We consider all corner breakpoints that are defined by pairs of adjacent circles in Case D, and add them to a set B'. We also consider all arcs pq of circles that are part of pairs in Case D (p to the west of q), the vertical line l_p through p and the horizontal line l_q through q, and add $l_p \cap l_q$ to B'. We then find the largest rectangle S^* enclosing R and containing the fewest points in B' using the algorithm in [4] in $O(m + n)$ time. It is easy to check that S^* is an optimal solution for Case 1, since any circle containing points in B' intersects a circle in B. Thus, Case 1 can be done in $O(m + n)$ time.

Case 2.

Assume wlog that two circles pin the north and the west edges of a CSR. The cases where the two circles define a different pair of adjacent edges of a CSR can be handled in a similar fashion.

If we are in Case 2.1, we consider all corner breakpoints q defined by pairs of adjacent circles in Case D, as well as intersection between the south horizontal tangent t_H to the eastmost circle in B_{NE} and the east vertical tangent t_V to the northmost circle in B_{NW} south of t_H. For every such point, the north and west edges are fixed, and we either find the south edge by extending the west edge southwards until it hits a blue circle, or the east edge by extending the north edge eastwards until it hits a blue circle. In both approaches, the fourth edge is uniquely determined. For each circle in $C \in B_{NE}$, we store pointers to the northmost circle in B_{NW} south of C and to the eastmost circle in B_{SE} west of C, as well as similar pointers for the other quadrants and directions, Thus, once the first two edges are fixed, we can find the third and fouth edges in $O(1)$ time. Since there are $O(m)$ circles in Case 2.1 and they all can be found in $O(m)$ time, Case 2.1 can be solved in $O(m)$ time.

For Case 2.2, we consider all points q as in Case 2.1. Having selected such point q defined by two circles $C_1 \in B_{NE} \cup B_{NW}$ and $C_2 \in B_{NW} \cup B_{SW}$, we consider the dominating envelope of B_{SE} starting from the east tangent to C_1 or the east edge of S_{min}, whichever is eastmost, and ending at the south tangent to C_2 or the south edge of S_{min}, whichever is southmost. This gives us a range of motion for the SE corner r of the CSR spanning $O(m)$ circle arcs. For each such arc, we find the optimal CSR in $O(1)$ time as we shall prove in the next section. Since there are $O(m)$ choices of q, we therefore handle Case 2.2 in $O(m^2)$ time.

As for Case 2.3, note that the pairs of circles defining the NW and the SE corners, respectively, must belong to a dominating envelope. We scan the dominating envelope of B_{NW} for pairs of circles C_1, C_2 in Cases B and C and, for each such pair, we scan the dominating envelope of B_{SE} for pairs of circles in Cases B and C, starting from the east tangent to C_1 or the east edge of S_{min}, whichever is eastmost, and ending at the south tangent to C_2 or the south edge of S_{min}, whichever is southmost. Once these pairs are established, the CSR is determined. Since scanning each dominating envelope takes $O(m)$ time, we handle Case 2.3 in $O(m^2)$ time.

Case 3.

Assume wlog that a circle C pins the east edge of the CSR. The cases where the circle define a different edge of the CSR can be handled in a similar fashion.

For Case 3.1, we scan the dominating envelope of B_{NE} (similarly, B_{SE}) for pairs of circles (C_1, C) in cases C and D, (C is the rightmost circle of the pair). For every such pair of circles in B_{NE}, we consider all circles $C_2 \in B_{SE}$ that are intersected by the west vertical tangent to C. It is possible that some of these circles were already considered for a previous pair of circles in B_{NE}, so we may have to consider $O(m^2)$ triplets of circles (C, C_1, C_2). We also traverse the circles in B_E in increasing X order of their centers. Denote by C the current circle. We consider the sequences of circles $C_{NE} \in B_{NE}$ and $C_2 \in B_{SE}$ that are intersected by the west tangent to C. Since these sequences may include circles already considered for a previous circle in B_E, we may need to spend $O(m^2)$ to find all such triplets (C, C_1, C_2) for which there exists a vertical line intersecting both C_1, C_2. Once a triplet is established, the west, north, and south egdes are established, and the west edge can be determined in $O(1)$ time by extending the north or the south edge until it hits a circle. Thus, Case 3.1 requires $O(m^2)$ time.

For Case 3.2, we scan the dominating envelope of B_{SE} (similarly, B_{NE}) for pairs of circles (C_1, C) in cases C and D (C is the rightmost circle of the pair). We also traverse the circles $C \in B_E$ in increasing X order, and consider the sequences of circles $C_1 \in B_{SE}$ that are intersected by the west tangent to C. For each pair (C_1, C), the east and south edges of the CSR are defined. This provides a range of eligible circles from the dominating envelope of B_{NW} such that the SW and NE corners of the CSR are not supported by any circle, and the NW corner slides along some circle arc. There are $O(m)$ (C_1, C) pairs and each of them gives $O(m)$ circles from B_{NW}. Hence, Case 3.2 takes $O(m^2)$ time.

Case 4.

Consider all arcs defined by pairs of adjacent circles in one of the cases A, B, C, or D. Consider all arcs defined by pairs of adjacent circles. Each such arc a establishes the range of motion for the appropriate corner p of a CSR, say $[p_{start}, p_{end}]$ in X order. Suppose a belongs to a circle in B_{NE}, which establishes the range of motion of the NE corner p of the CSR. This gives us a range of sliding motion for the north and the east edges of the CSR, which are supported by two rays r_W, r_S shooting from p to the west and south, respectively. Since only the SW corner q may also slide along a circle, the west edge can be neither to the west of the first intersection $W(p)$ between r_W and a circle, nor to the west of the easternmost point in B_W of a circle. Similarly, the south edge can be neither be to the south of the first intersection $S(p)$ between r_S and a circle, nor to the south of the northernmost point in B_S of a circle. This gives a range of motion for q, which may span multiple arcs of circle with X coordinates within the range $arcs(p_{start}, p_{end}) = [\min(X(W(S(p_{end}))), X(S(W(p_{end})))), \max(X(W(S(p_{start}))), X(S(W(p_{start}))))]$. In fact, there are $O(m)$ arcs in the worst case, yielding $O(m^2)$ pairs of arcs for all possible pairs (p, q). By computing the pointers $west(p), south(p), east(p)$, and $north(p)$ for every p, from the dominating envelope, we can find each pair of arcs in $O(1)$ time, as we take them in X order. That is, we consider all arcs $[p_{start}, p_{end}]$ defined by pairs of adjacent circles in X order and, for each such arc, we compute the points $W(S(p_{start}))$, $S(W(p_{start}))$, $W(S(p_{end}))$, and $S(W(p_{end}))$, and then consider the arcs in B_{SW} with X coordinates within $arcs(p_{start}, p_{end})$.

In the next subsection, we also show how to handle each pair of arcs in Case 4 in $O(1)$ time, in order to find the optimal solution in $O(m^2)$ time.

3.4 Finding the CSR once the arcs pinning its corners are selected

We show how to find a maximum separating rectangle once the arcs pinning the corners are selected.

For each quadrant Q, let $\theta_Q \in [\alpha_Q, \beta_Q]$ be angular position of the corner within the arc belonging to Q, say $C(c, 1)$. Let $f(\theta_{NE}, \theta_{NW}, \theta_{SW}, \theta_{SE})$ denote the area of the CSR with the corners defined in terms of $theta_Q$ as above. Our goal is to find the maximum of f over

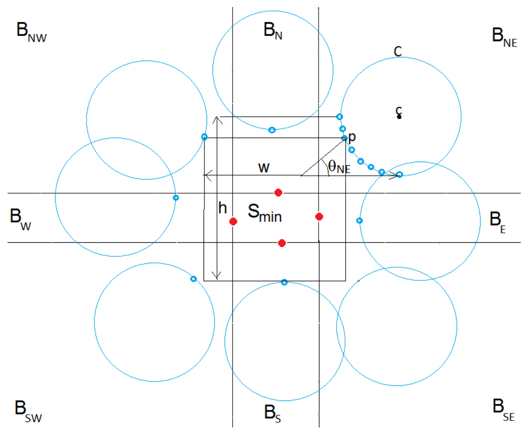


Figure 14: The function $f_{NE}(\theta_{NE})$ denoting the area of the CSR cornered on circle $C(c, 1)$

the feasible set of arguments, along with its arguments. First, assume $\theta_{SE}, \theta_{NW}, \theta_{SW}$ are fixed, with the left and bottom supports denoted as l, b (Figure 14). We refer to $f(\theta_{NE}, \theta_{NW}, \theta_{SW}, \theta_{SE})$ as simply $f_{NE}(\theta)$ (that is, refer to θ_{NE} as simply θ).

We have the following lemma.

Lemma 6 *f_{NE} has at most 3 maxima.*

We have

$$f_{NE}(\theta) = (w - \sin\theta) \cdot (h - \cos\theta), \tag{1}$$

where $w = x(c) - x(l)$ and $h = y(c) - y(b)$. For simplicity, assume that all circles are fully contained in some quadrant, so $w, h > 1$. Also,

$$f'_{NE}(\theta) = w \sin\theta - h\cos\theta + \sin^2\theta - cos^2\theta. \tag{2}$$

Letting $x = \tan\theta$, we get

$$f'_{NE}(x) = \frac{wx - h}{\sqrt{1+x^2}} + \frac{x^2-1}{1+x^2}, \tag{3}$$

so s$f'_{NE}(x) = 0 \iff (wx-h)\sqrt{1+x^2} = 1 - x^2 \iff (wx-h)^2(1+x^2) = (1-x^2)^2 \iff (w^2-1)x^4 - 2whx^3 + (w^2+h^2+2)x^2 - 2whx + h^2 - 1 = 0 \iff$
$(w^2-1)x^2(x^2-1) - 2whx(x^2-1) + (h^2-1)(x^2-1) = 0 \iff$
$((w^2-1)x^2 - 2whx + h^2 - 1)(x^2-1) = 0$, which solves to

$$x_1 = 1, \tag{4}$$

$$x_{2,3} = \frac{wh + -\sqrt{2(w^2+h^2)-1}}{w^2-1} \tag{5}$$

(we ignore negative roots since $x \geq 0$). Since $x = \tan\theta$, it follows that $\theta_1 = \frac{\pi}{4}, \theta_{2,3} = \arctan x_{2,3}$ are extrema for f_{NE}. This means there are at most three maxima for f_{NE}.

By a similar argument, there are at most three maxima for f_{NW}, f_{SW}, f_{SE}. Note that the position of two opposite corners of a CSR, say NE and SW, determine the position of the other two corners. This gives us at most $M = 9$ maxima for f, as there are only two variables. We compute the $O(1)$ maxima of f and choose the one that gives the largest CSR.

Thus, we have proved the following result.

Theorem 7 *Given a set of red points R with $|R| = n$, and a set of blue unit circles B with $|B| = m$, the largest rectangle enclosing R and avoiding all circles in B can be found in $O(m^2 + n)$ time.*

Acknowledgement

The author would like to thank Drs. Benjamin Raichel, Chenglin Fan, and Ovidiu Daescu for the useful discussions.

4 Conclusions and Future Work

We consider the outlier version of the largest axis-aligned separating rectangle (MBSR-O), for which we give an $O(k^3m + m\log m + n)$ time algorithm. We also study the problem of finding the largest axis-aligned separating rectangle among unit circles (MBSR-C) and give an $O(m^2 + n)$ time algorithm.

We leave for future consideration finding the largest, as well as the smallest enclosing circle avoiding all blue circles. A "combined" version such as MBSR-C with outliers, i.e., finding the largest rectangle enclosing all red points while containing at most k circles, would also be of interest. Finally, it would be interesting to either further improve the time bounds for the MBSR-O and MBSR-C, prove lower bounds, or come up with approximation algorithms.

References

[1] A. Agarwal and S. Suri, Fast algorithms for computing the largest empty rectangle, SoCG'1987: 278-290

[2] B. Armaselu and O. Daescu, Maximum Area Rectangle Separating Red and Blue Points, CCCG'2016: 244-251

[3] B. Armaselu and O. Daescu, C. Fan, and B. Raichel, Largest Red Blue Separating Rectangles Revisited, FWCG'2016

[4] B. Armaselu and O. Daescu, Maximum Area Rectangle Separating Red and Blue Points, arXiv:1706.03268 (2017)

[5] B. Armaselu and O. Daescu, Dynamic Minimum Bichromatic Separating Circle, Theoretical Computer Science 774: 133-142 (2019), Available online 30 Nov. 2016, http://dx.doi.org/10.1016/j.tcs.2016.11.036

[6] S. Bitner, Y. Cheung and O. Daescu, Minimum Separating Circle for Bichromatic Points in the Plane, ISVD'2010: 50-55

[7] J. Chaudhuri, S. C. Nandy and S. Das, Largest empty rectangle among a point set, Journal of Algorithms 46(1): 54-78 (2003)

[8] A. Mukhopadhyay and S.V. Rao, Computing a Largest Empty Arbitrary Oriented Rectangle. Theory and Implementation, International Journal of Computational Geometry and Applications 13(3): 257-271 (2003)

[9] S. C Nandy, B. B Bhattacharya, and S. Ray, Efficient algorithms for identifying all maximal isothetic empty rectangles in VLSI layout design, FSTTCS'1990: 255-269

[10] S. C Nandy, B. B Bhattacharya, and A. Sinha, Location of the largest empty rectangle among arbitrary obstacles, FSTTCS'1994: 159-170

[11] F. Sheikhi, A. Mohades, M. de Berg, and A. D. Mehrabi, Separability of imprecise points, Computational Geometry 61: 24-37 (2017)

Twisted Topological Tangles or: the Knot Theory of Knitting

Elisabetta Matsumoto*

Abstract

Imagine a 1D curve, then use it to fill a 2D manifold that covers an arbitrary 3D object—this computationally intensive materials challenge has been realized in the ancient technology known as knitting. This process for making functional 2D materials from 1D portable cloth dates back to prehistory, with the oldest known examples dating from the 11th century CE. Knitted textiles are ubiquitous, because they are easy and cheap to create, lightweight, portable, flexible, and stretchy. As with many functional materials, the key to knitting's extraordinary properties lies in its microstructure.

At the 1D level, knits are composed of an interlocking series of slip knots. At the most basic level, there is only one manipulation that creates a knitted stitch—pulling a loop of yarn through another loop. However, there exist hundreds of books with thousands of patterns of stitches with seemingly unbounded complexity.

The topology of knitted stitches has a profound impact on the geometry and elasticity of the resulting fabric. This puts a new spin on additive manufacturing—not only can stitch pattern control the local and global geometry of a textile, but the creation process encodes mechanical properties within the material itself. Unlike standard additive manufacturing techniques, the innate properties of the yarn and the stitch microstructure has a direct effect on the global geometric and mechanical outcome of knitted fabrics.

About the Speaker

Elisabetta Matsumoto is an assistant professor in the School of Physics at Georgia Institute of Technology. Her physics research centers around the relationship between geometry and material properties in soft systems, including liquid crystals, 3D printing and textiles. Her lab studies knitted textiles from the point of view of knot theory and as an additive manufacturing technique. She is also interested in using sewing, 3D printing and virtual reality in mathematical art and education.

*School of Physics, Georgia Institute of Technology

Axis-Aligned Square Contact Representations

Andrew Nathenson[*]

Abstract

We introduce a new class $\mathcal{G}$ of bipartite plane graphs and prove that each graph in $\mathcal{G}$ admits a proper square contact representation. A contact between two squares is *proper* if they intersect in a line segment of positive length. The class $\mathcal{G}$ is the family of quadrangulations obtained from the 4-cycle C_4 by successively inserting a single vertex or a 4-cycle of vertices into a face.

For every graph $G \in \mathcal{G}$, we construct a proper square contact representation. The key parameter of the recursive construction is the aspect ratio of the rectangle bounded by the four outer squares. We show that this aspect ratio may continuously vary in an interval I_G. The interval I_G cannot be replaced by a fixed aspect ratio, however, as we show, the feasible interval I_G may be an arbitrarily small neighborhood of any positive real.

1 Introduction

Geometric representations of graphs have many applications and yield intriguing problems [9]. Koebe's celebrated *circle packing theorem* [8], for example, states that every planar graph is a contact graph of interior-disjoint disks in the plane. Schramm [10] proved that this theorem holds even if we replace the disks with homothets of an arbitrary smooth strictly convex body in the plane. The result extends to non-smooth convex bodies in a weaker form (where a homothet may degenerate to a point, and three or more homothets may have a common point of intersection), and every planar graph is only a *subgraph* of such a contact graph.

In this paper, we consider *strong* contact representations with interior-disjoint convex bodies where no three convex bodies have a point in common. It is an open problem to classify graphs that admit a strong contact representation with homothets of a triangle or a square [1, 2]. It is known that every partial 3-tree [1] and every 4-connected planar graph admits a strong contact representation with homothetic triangles, see [5, 6]; but there are 3-connected planar graphs which do not admit such a representation. We note here that every planar graph admits a strong contact representation with (non-homothetic) triangles [3]; see also [6].

[*]California State University Northridge `andrew.nathenson.540@my.csun.edu`

Research on this paper was partially supported by the NSF award DMS-1800734.

Strong contact representations with homothetic squares have been considered only recently. Da Lozzo et al. [2] proved that every $K_{3,1,1}$-free partial 2-tree admits a proper contact representation with homothetic squares, where a contact between two squares is *proper* if they intersect in a line segment of positive length (in particular, proper contacts yield a strong contact representation). Eppstein [4] indicated that another family of graphs, defined recursively, can also be represented as a proper contact graph of squares. We remark that Klawitter et al. [7] proved that every triangle-free planar graph is the proper contact graph of (non-homothetic) axis-aligned rectangles.

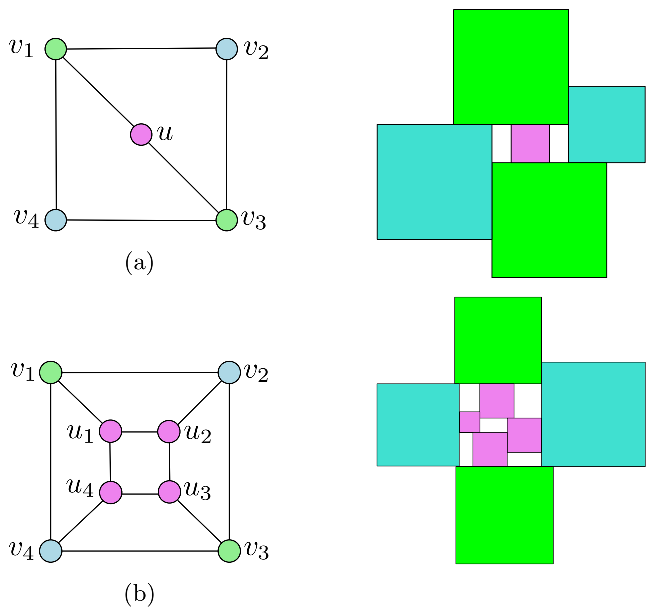


Figure 1: The two operations used to obtain a graph in $\mathcal{G}$ and their square contact representations.

Contribution. Let $\mathcal{G}$ be a family of plane bipartite graphs defined recursively as follows. (i) $\mathcal{G}$ contains the 4-cycle C_4. (ii) If $G \in \mathcal{G}$ and $f = (v_1, v_2, v_3, v_4)$ is a bounded 4-face of G, then $\mathcal{G}$ also contains the graphs G_a and G_b obtained by the following two operations: (a) insert a vertex u into f and connect it to v_1 and v_3; (b) insert four vertices $u_1, \ldots, u_4$ into f, add the cycle (u_1, u_2, u_3, u_4) and the edges $u_i v_i$ for $i = 1, \ldots, 4$; see Fig. 1.

Every maximal 2-degenerate bipartite plane graph can be constructed by operation (a); and the 1-skeleton of every polycube whose dual graph is a tree [4] can be constructed by operation (b). However, the two operations jointly produce a larger class $\mathcal{G}$, which belongs to the class of 3-degenerate bipartite plane graphs. In a square contact representation (SCR) of a graph in $\mathcal{G}$, every vertex v_i corresponds to an axis-aligned square $s(v_i)$, and every bounded face to an axis-aligned rectangle $g(f_i)$, which is also called the *gap* corresponding to f_i. We present our main result:

Theorem 1 *Every graph in $\mathcal{G}$ admits a proper square contact representation.*

We prove Theorem 1 by induction in Section 3. For the induction hypothesis we establish a stronger version of the theorem in which one specifies intervals for the aspect ratios (defined as height/width) of every gap in the representation, then recursively creates the SCR around those gaps.

Theorem 2 *Let $G \in \mathcal{G}$ be a graph with n vertices and $n-3$ bounded faces $f_1, \ldots, f_{n-3}$. For all $\alpha_1, \ldots, \alpha_{n-3} > 0$ and for all $\varepsilon > 0$, the graph G admits a proper square contact representation such that the aspect ratio of the gap corresponding to f_i is α'_i, with $|\alpha_i - \alpha'_i| < \varepsilon$, for all $i = 1, \ldots, n-3$.*

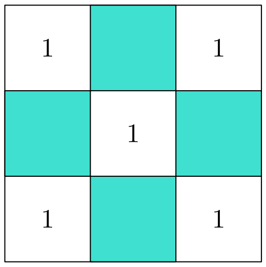


Figure 2: If all the gaps have aspect ratio 1, then scaling any of the squares to changing the point contacts into proper contacts would change the aspect ratios of the outer gaps.

Figure 2 shows an example where the aspect ratios of the gaps cannot be specified exactly in a proper contact representation.

However, it turns out that $\mathcal{G}$ includes graphs that must be bounded by a rectangle whose aspect ratio is arbitrarily close to any given value, if they are inserted into a face of another graph in $\mathcal{G}$.

Theorem 3 *For every $r, \delta > 0$, there exists a bipartite plane graph $G \in \mathcal{G}$ with a 4-cycle as its outer face such that in every SCR of G, the aspect ratio of the central gap between the four squares corresponding to that 4-cycle is confined to the interval $(r - \delta, r + \delta)$.*

Relation to rectangle tilings. Theorem 2 implies a tiling of a bounding box, where the tiles are squares (of aspect ratio 1) and rectangular gaps whose aspect ratios are prescribed up to an ε error term. Note that the contact graph of this tiling, including squares and gaps, and four additional vertices for the four sides of the outer frame, is a triangulation. Schramm [11] (see also [9, Chap. 6]) showed that for every inner triangulation G of a 4-cycle without separating triangles there exists a rectangle contact representation of G in which the rectangles have prescribed aspect ratios. However, some of the contacts between rectangles might be point contacts, and the interior of some of the separating 4-cycles may degenerate to a point. In the recursive construction of $\mathcal{G}$, step (ii) creates five separating 4-cycles in the triangulation of the tiling, one for each gap (see Fig. 3). In particular, if all five gaps degenerate to a point, then Schramm's result becomes trivial, but would not imply Theorem 2. The class of graphs defined in this paper is perhaps the first interesting case for which Schramm's approach is infeasible, as it cannot guarantee that the rectangles on the interior of the separating 4-cycles do not degenerate.

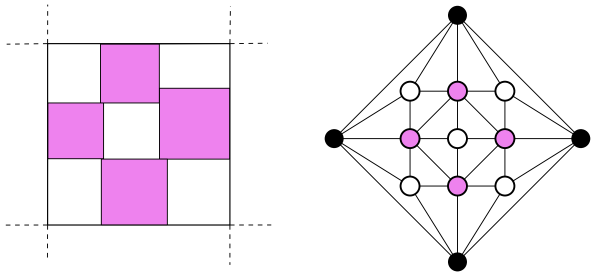


Figure 3: Left: a rectangular tiling with 9 tiles. Right: the corresponding triangulation, where the outer 4-cycle corresponds to the four edges of the outer frame.

Outlook. An obvious open problem is whether every triangle-free plane graph admits a proper square contact representation. Motivated by Schramm's results, one can also ask whether Theorem 1 generalizes to the setting where each vertex of the graph is associated with an axis-aligned rectangle of given aspect ratio.

Terminology. Let $G = (V, E)$ be an edge-maximal plane bipartite graph. In a square contact representation, every vertex v_i corresponds to an axis-aligned square $s(v_i)$, and every bounded face to an axis-aligned rectangle $g(f_i)$, which is also called the *gap* corresponding to f_i. The aspect ratio of an axis-aligned rectangle r is height(r)/width(r). The side length of a square s is denoted by len(s). Scaling up a square from a corner by (or to) x means to increase the width and height of the square by x (or to x) in such a way that the position of the specified corner remains fixed.

2 Maintaining a Square Contact Representation

In this section, we show how to maintain a square contact representation of a graph in $\mathcal{G}$ under operations (a) and (b). Specifically, we show that one can insert one or four new squares corresponding to these operations in a rectangular gap of suitable size. The following Lemmas are used in the proof of Theorem 2 to recursively construct a SCR for any given graph in $\mathcal{G}$.

Lemma 1 *For every $\alpha, \beta > 0$, there exists an axis-aligned rectangle that can be subdivided by two horizontal (resp., vertical) lines into three rectangles of aspect ratios α, 1, and β, respectively.*

Proof. Let R be a rectangle of aspect ratio $\alpha + \beta + 1$, with width x and height $(\alpha + \beta + 1)x$. Two horizontal lines at distance αx and βx from the top and bottom side of R, resp., subdivide R into rectangles of aspect ratios α, 1, and β, as required; see Fig. 4. □

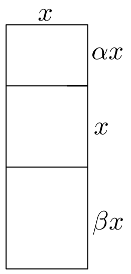


Figure 4: Constructing an outer rectangle given two inner rectangle aspect ratios.

To establish Theorem 1, we need a stronger version of Lemma 1 that allows the aspect ratios to vary within a small threshold.

Lemma 2 *For every $\alpha, \beta, \varepsilon > 0$, there exists a $\delta > 0$ such that any rectangle of aspect ratio γ with $|\gamma - (\alpha + \beta + 1)| < \delta$ can be subdivided by two horizontal lines into rectangles of aspect ratios α', 1, and β' such that $|\alpha' - \alpha| < \varepsilon$ and $|\beta' - \beta| < \varepsilon$.*

Proof. Let $\delta = \min\{\alpha, \beta, 1, \varepsilon\}$. Let R be a rectangle of aspect ratio γ, where $|\gamma - (\alpha + \beta + 1)| < \delta$, with width x and height γx. Two horizontal lines at distance αx and $(1 + \alpha)x$ from the top side of R subdivide R into rectangles of aspect ratios α, 1, and $\beta' = \gamma - \alpha - 1$. Note that $\beta' > 0$ and $|\beta' - \beta| = |\gamma - (\alpha + \beta + 1)| < \delta \leq \varepsilon$. □

Lemma 3 *For every $\alpha_1, \ldots, \alpha_5 > 0$, there exists an axis-aligned rectangle R that can be subdivided into four squares and five rectangular gaps of aspect ratios $\alpha_1, \ldots, \alpha_5$ such that (refer to Figs. 1b and 6)*

- *the four squares are each in contact with a side of R, and their contact graph is a 4-cycle (but the contacts along the 4-cycle are not necessarily proper);*
- *the first four gaps are each incident to the top-left, bottom-left, bottom-right, and top-right corner of R, respectively, and the fifth gap lies in the interior of R.*

The proof of Lemma 3 requires some preparation, and is presented later in this section. For convenience, we will rename $\alpha_1, \ldots, \alpha_5$ respectively based on the positions of the gaps to which they correspond as α_c (center), $\alpha_{t\ell}$ (top-left), α_{tr} (top-right), α_{br} (bottom-right), $\alpha_{b\ell}$ (bottom-left). Also, name the squares incident to the top, bottom, right, and left side of R as s_t, s_b, s_r, and s_ℓ, respectively.

We will prove Lemma 3 by starting with an initial configuration (Fig. 5), where the aspect ratio of the center gap is already α_c, and there are improper contacts between adjacent squares of the cycle. Then we incrementally modify the configuration, while the center gap remains fixed, until all remaining gaps have the target aspect ratios $\alpha_{t\ell}$, α_{tr}, α_{br}, and $\alpha_{b\ell}$. We denote the current aspect ratios of these gaps by $g_{t\ell}$, g_{tr}, g_{br}, and $g_{b\ell}$ in the same fashion as $\alpha_{t\ell}$, . . . ,$\alpha_{b\ell}$. We next define the initial configuration and four additional special configurations that play a role in intermediate steps of the incremental construction.

Initial configuration. To create the initial configuration, we start by drawing the interior gap and placing s_t, . . ., s_ℓ incident to it, with each of their side lengths equal to the side of the interior gap to which they are incident (see Fig. 5). Note that the aspect ratios of every outer gap is α_c^{-1} in this configuration.

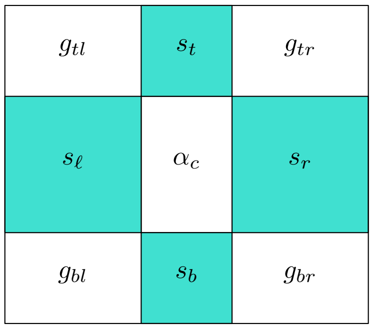


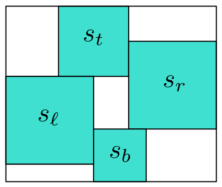
Figure 5: The initial configuration, with squares and gap aspect ratios labeled.

Pinwheel configuration. A *clockwise* pinwheel configuration is defined as follows (see Fig. 6a):

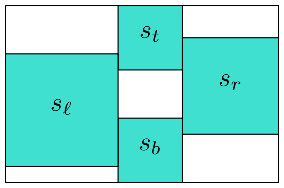
- the bottom-right corner of s_t lies on the left side of s_r,

- the bottom-left corner of s_r lies on the top side of s_b,
- the top-left corner of s_b lies on the right side of s_ℓ,
- the top-right corner of s_ℓ lies on the bottom side of s_t.

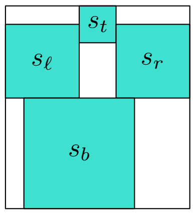
A *counterclockwise* pinwheel can be obtained by a reflection.



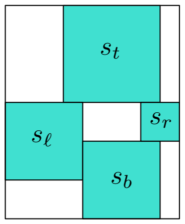
(a) Clockwise Pinwheel



(b) Vertical Stacked



(c) Downward Arrow



(d) Clockwise Near-Pinwheel with reversed contact between s_r and s_t

Figure 6: Examples of four special configurations.

Stacked configuration. We define a *vertical* stacked configuration as follows (see Fig. 6b):

- the top-right corner of s_b lies on the left side of s_r,
- the top-left corner of s_b lies on the right side of s_ℓ,
- the bottom-right corner of s_t lies on the left side of s_r,
- the bottom-left corner of s_t lies on the right side of s_ℓ.

A *horizontal* stacked configuration can be obtained by a 90° rotation.

Arrow configuration. We define a *downward* arrow configuration as follows (see Fig. 6c):

- the top-right corner of s_b lies on the bottom side of s_r,
- the top-left corner of s_b lies on the bottom side of s_ℓ,
- the bottom-right corner of s_t lies on the left side of s_r,
- the bottom-left corner of s_t lies on the right side of s_ℓ.

Upward, leftward, and *rightward* arrow configurations can be obtained by rotation. We also define the *directional* square of the arrow configuration to be the one furthest in the direction after which the configuration is named (e.g., s_b for a downward arrow configuration).

Near-pinwheel configuration. We define a *clockwise* near-pinwheel configuration as a configuration which would be a clockwise pinwheel configuration if one of the contacts between squares was changed from vertical to horizontal, or vice-versa (see Fig. 6d). This contact is called the *reversed* contact of the near-pinwheel configuration. A *counterclockwise* near-pinwheel configuration can be obtained by reflection.

Lemmas 4–8 below concern transformations of these special configurations, and are used in the proof of Lemma 3.

Lemma 4 *Assume that the top-left corner of s_r is on the right side of s_t and the bottom-left corner of s_r is on the right side of s_b, and let $\alpha_{tr} > g_{tr}$ be given. There exists a $d > 0$ such that if we slide s_r upward by d and scale it up by a factor of d/g_{br} from its bottom-left corner, then no aspect ratio other than g_{tr} changes, and after the transformation we have $\alpha_{tr} = g_{tr}$, or $\alpha_{tr} > g_{tr}$ and s_r and s_b have a point contact. Similar statements hold after reflections and rotations of the configuration.*

Proof. Let the bottom-right gap have height h_1 and width w prior to the transformation. Assume that we slide s_r upward by some amount $d > 0$ and scale it up by a factor of d/g_{br} from its bottom-left corner. After the transformation, it has height $h_1 + d$ and width $w + \frac{dw}{h_1}$. As

$$\frac{h_1}{w} = \frac{h_1 + d}{w + \frac{dw}{h_1}},$$

the aspect ratio of the bottom-right gap has not changed. Let the height of top-right gap be h_2 prior to the transformation, and note that its width is also w. After the transformation, it has height $h-d$ and width $w+\frac{d}{g_{br}}$. Thus, its height monotonically decreases in d, and its width monotonically increases in d, so g_{tr} monotonically decreases in d. We can choose $d=\min(d_1,d_2)$, where $d_1 \geq 0$ is the value which would reduce the contact between s_r and s_b to a single point after the transformation, and $d_2 \geq 0$ is the value which would achieve $\alpha_{tr} = g_{tr}$. □

Lemma 5 *A clockwise (counterclockwise) pinwheel configuration can be transformed such that g_{br} or $g_{t\ell}$ (g_{tr} or $g_{b\ell}$) increases to, or such that g_{tr} or $g_{b\ell}$ (g_{br} or $g_{t\ell}$) decreases to any amount $\gamma > 0$, while all other aspect ratios remain the same.*

Proof. Assume w.l.o.g. that we are given a *clockwise* pinwheel configuration, and we wish to increase the aspect ratio g_{br} to $\gamma > g_{br}$. If we scale up s_b from its top-left corner by some amount d_1, then $g_{b\ell}$ will increase. To account for this change, though, we can scale up s_ℓ as well so that $g_{b\ell}$ remains constant. Let h be the height of the central gap. Then,

$$g_{bl} = \frac{\text{len}(s_b) - \text{len}(s_\ell) + h}{\text{len}(s_\ell)}.$$

After increasing the length of s_b by d_1, we must then increase the length of s_ℓ by some amount d_2 such that

$$\frac{\text{len}(s_b) - \text{len}(s_\ell) + h}{\text{len}(s_\ell)} = \frac{(\text{len}(s_b)+d_1) - (\text{len}(s_\ell)+d_2) + h}{\text{len}(s_\ell)+d_2}$$

so that $g_{b\ell}$ does not change. Solving this equation for d_2 yields

$$d_2 = d_1 \frac{\text{len}(s_\ell)}{\text{len}(s_b)+h}.$$

Because s_ℓ is not in contact with the bottom of R, $\text{len}(s_\ell) < \text{len}(s_b) + h$. Thus, $d_2 < d_1$.

Let w be the width of the central gap. Then,

$$g_{tl} = \frac{\text{len}(s_t)}{\text{len}(s_\ell) - \text{len}(s_t) + w}.$$

After increasing the length of s_ℓ by d_2, to maintain $g_{t\ell}$, we must increase the length of s_t by some amount d_3 such that

$$\frac{\text{len}(s_t)}{\text{len}(s_\ell) - \text{len}(s_t) + w} = \frac{\text{len}(s_t)+d_3}{(\text{len}(s_\ell)+d_2) - (\text{len}(s_t)+d_3) + w}.$$

Solving for d_3 gives

$$d_3 = d_2 \frac{\text{len}(s_t)}{\text{len}(s_\ell)+w}.$$

Because s_t is not in contact with the left side of R, $\text{len}(s_t) < \text{len}(s_\ell) + w$. Thus, $d_3 < d_2$.

After increasing the length of s_t by d_3, we must increase the length of s_r by some amount d_4 to maintain g_{tr}. Similarly to the argument above, we obtain $d_4 < d_3$, and thus, $d_4 < d_1$.

So, this series of transformations, preserving g_{tr}, $g_{t\ell}$, $g_{b\ell}$, and the central gap, increases the length of s_b by d_1, which is more than the amount it increases the length of s_r, d_4. Specifically,

$$d_4 = \frac{d_1 \text{len}(s_\ell)\text{len}(s_t)\text{len}(s_r)}{(\text{len}(s_b)+h)(\text{len}(s_\ell)+w)(\text{len}(s_t)+h)} < d_1.$$

Before the transformations, the top boundary of s_b overlapped the bottom boundary of s_r by some amount x. After the transformations, it overlaps by $x+d_1$, because s_b has been scaled up from its top-left corner.

The width of the bottom-right gap equals $\text{len}(s_r)$ minus the length of the common boundary between s_r and s_b. Because the length of that common boundary increases by d_1, but $\text{len}(s_r)$ increases only by $d_4 < d_1$, the width decreases. Consequently, the width of the bottom-right gap decreases and its height increases linearly in d_1. Overall, g_{br} monotonically increases in d_1. We have constructed a series of transformations that can increase g_{br} to any $\gamma > g_{br}$ with a suitable d_1. □

Lemma 6 *A vertical (resp., horizontal) stacked configuration with a point contact between two of the squares can be transformed such that the aspect ratio of the outer gap between those squares increases (resp., decreases) to any amount $\gamma > 0$ while all other aspect ratios remain the same.*

Proof. Assume w.l.o.g. that we are given a vertical stacked configuration in which s_r and s_b have a point contact, and we wish to increase the aspect ratio g_{br} to $\gamma > g_{br}$.

If there is not a point contact between s_ℓ and s_t, then the following transformation can be applied. Scale up s_b from its top-left corner to increase g_{br}. To account for the resulting change in $g_{b\ell}$, scale up s_ℓ and translate it downward while maintaining $g_{t\ell}$, as described in Lemma 4. This transformation will either increase g_{br} to γ, or it will result in a point contact between s_ℓ and s_t.

If there is a point contact between s_ℓ and s_t, then the squares are arranged in a pinwheel configuration, and by Lemma 5 we can increase g_{br} to γ while maintaining all other aspect ratios. □

Lemma 7 *An upward or downward (resp., rightward or leftward) arrow configuration, with a point contact between the directional square and one of its neighbors, can be transformed such that the aspect ratio of the outer gap between those squares increases (resp., decreases) to* *any amount $\gamma > 0$ while all other aspect ratios remain the same.*

Proof. Assume w.l.o.g. that we are given a downward arrow configuration in which s_r and s_b have a point contact, and we wish to increase the aspect ratio g_{br} to γ.

If s_b and s_ℓ do not have a point contact, translate s_b to the right while scaling it up in order to maintain $g_{b\ell}$ (as described in Lemma 4) while increasing g_{br} until $g_{br} = \gamma$, or until there is a point contact between s_b and s_ℓ.

If s_b and s_ℓ have a point contact, then scale up s_b from its top-left corner to increase g_{br}. To account for the corresponding change in $g_{b\ell}$, translate s_ℓ downward while scaling it up to maintain $g_{t\ell}$ (as described in Lemma 4) until $g_{br} = \gamma$, or until there is a point contact between s_ℓ and s_t.

If s_ℓ and s_t have a point contact, then the squares are arranged in a pinwheel configuration, and by Lemma 5 we can increase g_{br} to γ while maintaining all other aspect ratios. □

Lemma 8 *A near-pinwheel configuration can be transformed such that the aspect ratio of the outer gap in the direction of the near-pinwheel (clockwise or counterclockwise) from the reversed contact increases to any amount $\gamma > 0$ if its left side is the side of a square, or decreases to any amount $\gamma > 0$ if its top side is the side of a square, while all other aspect ratios remain the same.*

Proof. Assume w.l.o.g. that we are given a clockwise near-pinwheel with a reversed top-right contact (as in Figure 6d), and we wish to increase the aspect ratio g_{br} to γ.

Perform the following transformation until s_t and s_r have a point-contact or until g_{br} has been increased to γ. Scale up s_b from its top-left corner by some amount. To account for the corresponding change in $g_{b\ell}$, scale up s_ℓ from its top-right corner. To account for the corresponding change in $g_{t\ell}$, scale up s_t and translate it to the left while maintaining g_{tr} as described in Lemma 4.

If g_{br} does not reach its target value once s_t and s_r have a point contact, then the configuration is a pinwheel, and by Lemma 5 we can increase g_{br} to γ. □

We now have everything needed to prove Lemma 3.

Proof. [Proof of Lemma 3] Let α_c, $\alpha_{t\ell}$, α_{tr}, α_{br}, and $\alpha_{b\ell}$ be given. Start with the initial configuration (cf. Fig. 5). If the target aspect ratios of all four outer gaps are α_c^{-1}, then R can be drawn now with aspect ratio α_c. Otherwise, one or more of the outer gaps must have their aspect ratios changed, either by increasing or decreasing them.

Rotate and reflect the initial configuration if necessary such that at least one gap needs to be made wider (i.e., $\alpha < g$), and the ratio g/α is maximal for the top-right gap. In order to change g_{tr} to α_{tr}, we can scale up s_r from its bottom-left corner until $g_{tr} = \alpha_{tr}$. This scaling will not affect $g_{t\ell}$ or $g_{b\ell}$, but it will decrease g_{br}. After the scaling, the bottom-right gap will either have the target aspect ratio already, need to be wider yet, or need to be narrower. From now on, we will not mention the case where a gap has reached its target aspect ratio already, because it just means that the next step can be skipped.

If the bottom-right gap needs to be wider yet, then by Lemma 4 we can scale up s_r and translate it downward until $g_{br} = \alpha_{br}$ without changing g_{tr}. As g/α is assumed to be maximal for the top-right gap, if this transformation results in a point contact between s_r and s_t, it also achieves $g_{br} = \alpha_{br}$ (because otherwise, $g_{br} > g_{tr} = \alpha_{tr}$).

If the bottom-right gap needs to be narrower, then we can scale up s_b from its top-left corner until $g_{br} = \alpha_{br}$. This will increase $g_{b\ell}$.

Now, we can assume that $g_{tr} = \alpha_{tr}$ and $g_{br} = \alpha_{br}$. We distinguish between four cases:

1. s_b has not been scaled, and either $\alpha_{b\ell} \leq \alpha_c^{-1}$ or $\alpha_{t\ell} \leq \alpha_c^{-1}$.

2. s_b has been scaled up from its top-left corner, $\alpha_{b\ell} \leq \alpha_{t\ell}$, and $\alpha_{b\ell} \leq \alpha_c^{-1}$.

3. s_b has been scaled up from its top-left corner, $\alpha_{t\ell} \leq \alpha_{b\ell}$, and $\alpha_{t\ell} \leq \alpha_c^{-1}$.

4. $\alpha_{t\ell} > \alpha_c^{-1}$ and $\alpha_{b\ell} > \alpha_c^{-1}$.

Case 1: s_b has not been scaled, and either $\alpha_{b\ell} \leq \alpha_c^{-1}$ or $\alpha_{t\ell} \leq \alpha_c^{-1}$. Reflect the configuration, if necessary, such that $\alpha_{b\ell} \leq \alpha_{t\ell}$. Scale up s_ℓ from its top-right corner until $g_{b\ell} = \alpha_{b\ell}$ (making the top-left gap wider). Then, if $g_{t\ell}$ needs to decrease further, by Lemma 4 we can scale up and translate s_ℓ until $g_{t\ell} = \alpha_{t\ell}$ to achieve all target aspect ratios (once again, this transformation guarantees $g_{t\ell} = \alpha_{t\ell}$ even if it results in a point contact, because we assume $\alpha_{b\ell} \leq \alpha_{t\ell}$). Otherwise, the top-left gap needs to be narrower. Since the configuration is a horizontal stacked configuration, and by Lemma 6 we can apply a series of transformations to achieve all target aspect ratios.

Case 2: s_b has been scaled up from its top-left corner, $\alpha_{b\ell} \leq \alpha_{t\ell}$, and $\alpha_{b\ell} \leq \alpha_c^{-1}$. Scale up s_ℓ from its top-right corner until $g_{b\ell} = \alpha_{b\ell}$. This transformation decreases $g_{t\ell}$. Then, if $g_{t\ell}$ needs to decrease further, by Lemma 4 we can scale up and translate s_ℓ until $g_{t\ell} = \alpha_{t\ell}$ to achieve all target aspect ratios (once again guaranteed because $\alpha_{b\ell} \leq \alpha_{t\ell}$). Otherwise the top-left gap needs to be narrower. Since the squares are arranged in a pinwheel configuration, Lemma 5 completes the proof.

Case 3: s_b has been scaled up from its top-left corner, $\alpha_{t\ell} \leq \alpha_{b\ell}$, and $\alpha_{t\ell} \leq \alpha_c^{-1}$. Scale up s_ℓ from its bottom-right corner until $g_{t\ell} = \alpha_{t\ell}$. This transformation decreases $g_{b\ell}$. Then, if $g_{b\ell}$ needs to decrease further, by Lemma 4 we can scale up s_ℓ and translate it downward, maintaining all other aspect ratios, until $g_{b\ell} = \alpha_{b\ell}$ or s_ℓ and s_t have a point contact. If s_ℓ and s_t have a point contact, then the squares are arranged in a pinwheel configuration, and Lemma 5 completes the proof. Otherwise, $g_{b\ell}$ needs to increase. Since the squares form a downward arrow configuration in this case, with a point contact between s_b and s_ℓ, Lemma 7 completes the proof.

Case 4: $\alpha_{t\ell} > \alpha_c^{-1}$ and $\alpha_{b\ell} > \alpha_c^{-1}$. We distinguish between two subcases.

Case 4.1: If the top-right corner of s_b lies on the bottom side of s_r, then by Lemma 4, we can translate s_b to the left while scaling it up until $g_{b\ell} = \alpha_{b\ell}$ or s_b and s_r have a point-contact, while maintaining all other aspect ratios. If $g_{b\ell} = \alpha_{b\ell}$, then the configuration is a near-pinwheel and Lemma 8 completes the proof. Otherwise, if s_b and s_r have a point-contact, then the conditions of Case 4.2 below are satisfied and we proceed as follows.

Case 4.2: If the top-right corner of s_b lies on the left side of s_r, then scale up s_t from its bottom-right corner until $g_{t\ell} = \alpha_{t\ell}$ and scale up s_b from its top-right corner until $g_{b\ell} = \alpha_{b\ell}$. Now, g_{tr} and g_{br} (which were previously at their target values) both need to decrease. Reflect the configuration, if necessary, so that the width of the bottom-right gap needs to be increased by a larger amount than the top-right gap. Scale up s_r from its bottom-left corner until $g_{tr} = \alpha_{tr}$. Then, because the width of the bottom-right gap needed to be increased by a larger amount of the two, it still needs to be wider. The configuration is a rightward arrow, so by Lemma 7, we can decrease g_{br} arbitrarily while maintaining the other aspect ratios. □

The following lemma, Lemma 9, shows that all improper contacts can be replaced by proper contacts at the expense of allowing the five aspect ratios to vary within a given threshold. Using exact values of the aspect ratios, Lemma 3 can only guarantee single-point contacts. However, it is easy to extend Lemma 3 to Lemma 9 by changing any improper contacts among adjacent squares in the 4-cycle into proper contacts.

Lemma 9 *For every $\alpha_1, \ldots, \alpha_5 > 0$ and $\varepsilon > 0$, there exists a $\lambda > 0$ and a $\delta > 0$ such that every axis-aligned rectangle R of aspect ratio λ', $|\lambda - \lambda'| < \delta$, can be subdivided into four squares and five gaps of aspect ratios α'_i, with $|\alpha'_i - \alpha_i| < \varepsilon$, for $i = 1, \ldots, 5$ such that*

- *the four squares are each in contact with a side of R, and their contact graph is a 4-cycle, and all contacts are proper;*
- *the first four gaps are each incident to the top-left, bottom-left, bottom-right, and top-right corner of R, respectively, and the fifth gap lies in the interior of R.*

Proof. Let α_c, $\alpha_{t\ell}$, α_{tr}, α_{br}, $\alpha_{b\ell}$, and $\varepsilon > 0$ be given. By Lemma 3, there is a rectangle R with some aspect ratio λ that can be subdivided into five gaps and four squares s_b, s_t, s_ℓ, and s_r whose contact graph is a cycle.

Case 1. Assume first that all four contacts in the cycle are proper. Then Lemma 9 holds with the same λ. In each case, there exists a square that can be scaled up or down while maintaining proper contacts in the cycle. When scaling a single square, the aspect ratio of the bounding box R and some of the gaps change continuously. By continuity, there exists a $\delta > 0$ such that if the aspect ratios of the bounding box is λ' with $|\lambda' - \lambda| < \delta$, then all five gaps are at most ε from their target values.

Case 2. Next assume that one or more contacts in the cycle are improper, i.e., two squares intersect in a common corner. For each improper contact, we can successively scale up one of the two squares to establish a proper contact. We scale up each square by a sufficiently small amount such that the aspect ratios of the five gaps change by less than $\varepsilon/2$. Let λ' be the aspect ratio of the new bounding box. We can show, similarly to Case 1, that Lemma 9 holds with $\lambda = \lambda'$ and some $\delta > 0$ by continuity. □

3 Proof of Theorem 2

Finally, we have all the tools needed to prove Theorem 2. We restate it for convenience:

Theorem 2 *Let $G \in \mathcal{G}$ be a graph with n vertices and $n-3$ bounded faces $f_1, \ldots, f_{n-3}$. For all $\alpha_1, \ldots, \alpha_{n-3} > 0$ and for all $\varepsilon > 0$, the graph G admits a proper square contact representation such that the aspect ratio of the gap corresponding to f_i is α'_i, with $|\alpha_i - \alpha'_i| < \varepsilon$, for all $i = 1, \ldots, n-3$.*

Proof. We proceed by induction on n, the number of vertices of G.

Basis step. Assume that $G = C_4$ is a 4-cycle with a single bounded face f_1. It is clear that for any $\alpha_1 > 0$, C_4 has a proper square contact representation as a pinwheel configuration in which the gap corresponding to f_1 has aspect ratio α_1.

Induction step. Let $G \in \mathcal{G}$ be a graph with $n \geq 5$ vertices, and assume that the claim holds for all graphs in $\mathcal{G}$ with fewer than n vertices. Then G was constructed from a graph $G_0 \in \mathcal{G}$ with operation (a) or (b) that inserts one or four vertices into a 4-face $f_0 = (v_1, \ldots, v_4)$. We may assume w.l.o.g. that v_1 and v_3 correspond to squares that lie on the vertical sides of the gap corresponding to f_0 in any square contact representation. We distinguish between two cases.

Case (a). Assume that G was obtained from G_0 by inserting a vertex u into f_0 and connecting it to v_1 and v_3. This operation subdivides f_0 into f_1 and f_2; and all other faces are present in both G and G_0. Let $\alpha_0 = \alpha_1+\alpha_2+1$. By Lemma 2, there exists a $\delta > 0$ such that any rectangle of aspect ratio α'_0 with $|\alpha'_0 - \alpha_0| < \delta$ can be subdivided by two horizontal lines into rectangles of aspect ratios α'_1, 1, and α'_2 such that $|\alpha'_1 - \alpha_1| < \varepsilon$ and $|\alpha'_2 - \alpha_2| < \varepsilon$. The induction hypothesis with $\varepsilon_0 = \min\{\varepsilon, \delta\}$ implies that G_0 admits a proper square contact representation such that the gap corresponding to f_0 has aspect ratio α'_0, where $|\alpha'_0 - \alpha_0| < \varepsilon_0 \leq \delta$, and all other gaps are at most $\varepsilon_0 \leq \varepsilon$ off from their target aspect ratios. Lemma 2 now yields a subdivision of the gap corresponding to f_0 into a square in proper contact with the squares corresponding to v_1 and v_3, and two gaps of aspect ratios α'_1 and α'_2 with $|\alpha'_1 - \alpha_1| < \varepsilon$ and $|\alpha'_2 - \alpha_2| < \varepsilon$.

Case (b). Assume that G was obtained from G_0 by inserting a 4-cycle (u_1, u_2, u_3, u_4) into f_0 and adding the edges u_iv_i for $i = 1, \ldots, 4$. This operation subdivides f_0 into five faces $f_1, \ldots, f_5$ of G; and all other faces are present in both G and G_0.

By Lemma 9, there exists an $\alpha_0 > 0$ and a $\delta > 0$ such that any rectangle of aspect ratio α'_0 with $|\alpha'_0 - \alpha_0| < \delta$ can be subdivided into four squares and five gaps corresponding to $f_1, \ldots, f_5$, of aspect ratios $\alpha'_1, \ldots, \alpha'_5$, respectively, such that $|\alpha'_i - \alpha_i| < \varepsilon$ for $i = 1, \ldots, 5$. The induction hypothesis with $\varepsilon_0 = \min\{\varepsilon, \delta\}$ implies that G_0 admits a proper square contact representation such that the gap corresponding to f_0 has aspect ratio α'_0, where $|\alpha'_0 - \alpha_0| < \varepsilon_0 \leq \delta$, and all other gaps are at most $\varepsilon_0 \leq \varepsilon$ off from their target aspect ratios. Lemma 9 now yields a subdivision of the gap corresponding to f_0 into four squares, each in contact with a unique one of $v_1, \ldots, v_4$ and cyclically in contact with one another, and five gaps of aspect ratios $\alpha_1, \ldots, \alpha_5$ with $|\alpha'_i - \alpha_i| < \varepsilon$ for $i = 1, \ldots, 5$. □

4 Proof of Theorem 3

Lemma 10 *For every integer $n > 2$, $K_{2,n} \in \mathcal{G}$; and in any SCR of $K_{2,n}$, if the squares corresponding to the partite set of size two have side lengths ℓ_1 and ℓ_2, then the distance between these squares is less than $\frac{\min(\ell_1,\ell_2)}{n-2}$.*

Proof. Let s_1 and s_2 be the squares of side lengths ℓ_1 and ℓ_2, respectively, in some SCR of $K_{2,n}$. W.l.o.g., we may assume that $\ell_1 \geq \ell_2$ and that s_1 is below s_2. There exists a rectangle between s_1 and s_2 whose top side is the side of s_2 and whose height is the distance between s_1 and s_2. It is clear that at most two of the n squares corresponding to the other partite set can be anything but fully contained in this rectangle (see Figure 7). Thus, the other $n-2$ squares must be inside this rectangle, and each must have the same side length because they contact the top and bottom of this rectangle. Furthermore, the sum of their side lengths is less than ℓ_2, because the squares don't overlap. Thus, each of these squares has height less than $\frac{\ell_2}{n-2}$, and the distance between s_1 and s_2 is less than $\frac{\ell_2}{n-2}$. □

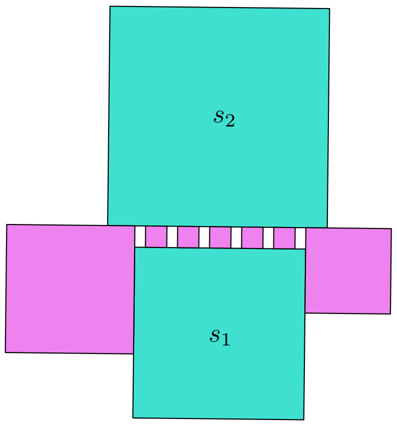


Figure 7: At most two squares are outside of the rectangle between s_1 and s_2.

Theorem 3 *For every $r, \delta > 0$, there exists a bipartite plane graph $G \in \mathcal{G}$ with a 4-cycle as its outer face such that in every SCR of G, the aspect ratio of the central gap between the four squares corresponding to that 4-cycle is confined to the interval $(r - \delta, r + \delta)$.*

Proof. Let $r, \delta > 0$ be given. By applying a 90° rotation, if necessary, we may assume that $r \in [1, \infty)$.

To construct G, we first construct its SCR. We start with the 4-cycle, and successively insert squares into a remaining gap (defined below). After i iterations, we obtain a graph G_i. We also maintain an interval I_i such that $r \in I_i$ and in *every* SCR of G_i, the aspect ratio of the central gap must be in I_i. Initially, we set $I_0 = (0, \infty)$. We show that $I_{i+1} \subset I_i$ and $|I_i| < 2^{-i}$ for all $i \in \mathbb{N}$. Consequently, $I_i \subset (r-\delta, r+\delta)$ when $2^{-i} < \delta$, and we can return $G = G_i$.

In each iteration, we repeatedly insert a square into a gap in the SCR contacting either the top and bottom of the gap, or the left and right. Clearly, the contact graph corresponding to the resulting SCR will be a 2-degenerate plane bipartite graph. Whenever we insert a square into a gap, we will also assume that it contacts one additional side of the gap; however, instead of an actual contact, we can only guarantee that in any SCR, they are sufficiently close to that side (cf. Lemma 10): If the square contacts the left and right, then it must be very close to the bottom; if it contacts the top and bottom, then it must be very close to the left. Specifically, if m is the total number of squares used in the rest of the construction, and ℓ is the side length of the largest square used in the construction, we can insert $\lceil \frac{2m\ell}{\delta} + 2 \rceil$ squares in between each square and the side it is supposed to be close to. This will ensure that each square is at most $\frac{\delta}{2m}$ apart from the side it is supposed to be close to, and thus that the aspect ratio differs from what the aspect ratio would be if these contacts actually existed by less than $\frac{\delta}{2}$. We can carry out the rest of the proof under the assumption that these squares in fact contact that side, and that the interval for the target aspect ratio is $(r - \frac{\delta}{2}, r + \frac{\delta}{2})$.

Because of these assumed additional contacts, there is always only one remaining gap in the course of the recursive construction. We will call this the *remaining* gap.

Let the aspect ratio of the central gap (of the outer 4-cycle) be constrained to the interval I_i. When we insert a square into the remaining gap, either the lower or upper bound on this aspect ratio will become constrained to some $c \in I_i$. Specifically, after inserting a square which contacts the left and right (and the bottom, as an additional contact) of the remaining gap, the lower bound increases to c and the upper bound is unchanged. This follows because the inserted square must be at least the width of the remaining gap, so the remaining gap's aspect ratio must be at least 1. However, it does not impose any constraint on the maximum height of the remaining gap, since the top of the square does not contact the top of the gap. Similarly, after inserting a square which contacts the top and bottom (and left) of the remaining gap, the height of the gap is limited to the height of the square, so the remaining gap's aspect ratio must be at most 1, while the width of the gap is no further constrained. As we will show later, the central gap's aspect ratio varies monotonically in the aspect ratio of the remaining gap. Thus, we know that some c must exist because inserting a square which contacts the top and bottom and inserting a square which contacts the left and right will each change a different one of the bounds of the aspect ratio of the remaining gap, and hence the central gap, to the same value.

So, one can always insert a sequence of squares contacting either the top and bottom or the left and right of the remaining gap, and it will either increase the lower bound or decrease the upper bound of the interval I_i, while containing the target aspect ratio r. In the remainder of the proof, we choose a specific sequence of insertions and show that both the upper and lower bounds converge to r.

Phases. Each iteration of the construction will consist of inserting squares into the remaining gap, g, in two *phases.* In each phase, we will either insert some number of squares which contact the left and right edges of the gap (a *vertical* phase) or some number of squares with contact the top and bottom (a *horizontal* phase). The number of squares inserted is the *size* of that phase. Because the squares in each phase contact the same two sides of the gap, each phase will either increase the lower bound or decrease the upper bound of the interval I_i. W.l.o.g., let the next phase to insert be horizontal, setting some upper bound on the aspect ratio of g. Then, by Lemma 10, we can insert a sufficiently large phase to reduce the distance between the last square in this horizontal phase and the side of g to an arbitrarily small value, bringing the lower bound of the aspect ratio of g arbitrarily close to the upper bound. Because the central gap's aspect ratio varies monotonically in the aspect ratio of g, for any vertical (resp., horizontal) phase, there exists a k for which inserting a phase of size $k+1$ would bring the lower (resp., upper) bound of I_i above (resp., below) r.

We will use the following process to construct a SCR whose central gap's aspect ratio is constrained to $(r - \delta, r + \delta)$, assuming $r \geq 1$. Let the interval which is the bounds of the central gap's aspect ratio be $I_i = (a_i, b_i)$. Starting with the four outer squares, while $|I_i| \geq \frac{\delta}{2}$:

1. Insert a vertical phase whose size is the largest possible such that $a_i \leq r$.
2. Insert a horizontal phase whose size is the largest possible such that $b_i \geq r$.

Let n be the total number of iterations.

Convergence. It is clear from the construction that $r \in I_{i+1} \subset I_i$ for all $i \in \mathbb{N}$. It remains to show that $|I_i| \leq 2^{-i}$. To prove the convergence, we will construct the same SCR from the inside-out. We start with an arrangement which is just the remaining gap, a rectangle, and add phases of squares alternatively contacting the left and bottom of this arrangement, as shown in Figure 8. After adding phases in this way, the four outer squares can be added so that this construction ends with the same SCR as we constructed with the above process.

Let the width of the configuration after adding i vertical phases be 1, and the height h (and thus, the aspect ratio), be in some interval $J_i = (c, d)$. In particular, note that $J_0 = (0, \infty)$ and $J_n = I_n$. We will then add a horizontal phase of size k, then a vertical phase of size ℓ. Note that each square in the horizontal phase has side length h, and each square in the vertical phase has side length $kh+1$. Thus, the aspect ratio of the arrangement after inserting these phases is now

$$\frac{(kh+1)\ell + h}{kh+1}.$$

This expression shows that $|J_i| < \infty$ for all $i \geq 1$. By adding an extra iteration with $k_0, \ell_0 = 1$ we can also guarantee that $J_0 < 1$. We can transform this expression as follows:

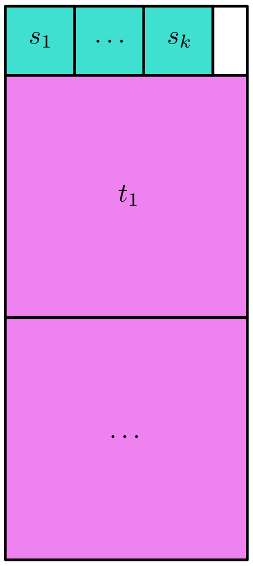


Figure 8: Starting with the remaining gap, we add a horizontal phase of squares (labeled $s_1, \dots, s_k$ in this figure), then a vertical phase ($t_1, \dots$), and will continue with alternating phases.

$$\frac{(kh+1)\ell + h}{kh+1} = \frac{(k\ell+1)h + \ell}{kh+1} = \frac{k\ell+1}{k} - \frac{\frac{1}{k}}{kh+1}.$$

This shows that the aspect ratio of the central gap varies monotonically in the aspect ratio of the remaining gap (as noted earlier).

As $\frac{k\ell+1}{k}$ is a positive constant, it does not affect the length of the interval J_{i+1}. Thus, we can say now that

$$|J_{i+1}| < \frac{1}{k} \left| \frac{1}{kd+1} - \frac{1}{kc+1} \right| < \frac{|c-d|}{(kc+1)(kd+1)}.$$

We know that d is at least 1, because $r \geq 1$, and k is at least 1 as well. Thus, the denominator is at least 2, and since $|J_i| = d - c$,

$$|J_{i+1}| < \frac{|J_i|}{2}.$$

Combined with $|J_0| < 1$, this implies $|I_n| = |J_n| < 2^{-n}$, and so $|I_n| < \delta$ if $2^{-n} < \delta$, or equivalently, $n > \log \delta^{-1}$. □

Acknowledgements

The author would like to thank Stefan Felsner for bringing to his attention the connection to Schramm's square tiling result. Additionally, he would like to thank Csaba Toth for all of his help and support in presenting these results.

References

[1] Melanie Badent, Carla Binucci, Emilio Di Giacomo, Walter Didimo, Stefan Felsner, Francesco Giordano, Jan Kratochvíl, Pietro Palladino, Maurizio Patrignani, and Francesco Trotta. Homothetic triangle contact representations of planar graphs. In *Proc. 19th Canadian Conference on Computational Geometry (CCCG)*, pages 233–236, Ottawa, ON, Canada, 2007. Carleton University. URL: http://cccg.ca/proceedings/2007/09b4.pdf.

[2] Giordano Da Lozzo, William E. Devanny, David Eppstein, and Timothy Johnson. Square-contact representations of partial 2-trees and triconnected simply-nested graphs. In *Proc. 28th Symposium on Algorithms and Computation (ISAAC)*, volume 92 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl, 2017. doi:10.4230/LIPIcs.ISAAC.2017.24.

[3] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. On triangle contact graphs. *Comb. Probab. Comput.*, 3:233–246, 1994. doi:10.1017/S0963548300001139.

[4] David Eppstein. Square contact graphs, 2017. URL: https://11011110.github.io/blog/2017/10/03/square-contact-graphs.html.

[5] Stefan Felsner and Mathew C. Francis. Contact representations of planar graphs with cubes. In *Proc. 27th Symposium on Computational Geometry (SoCG)*, pages 315–320. ACM Press, 2011. doi:10.1145/1998196.1998250.

[6] Daniel Gonçalves, Benjamin Lévêque, and Alexandre Pinlou. Triangle contact representations and duality. *Discret. Comput. Geom.*, 48(1):239–254, 2012. doi:10.1007/s00454-012-9400-1.

[7] Jonathan Klawitter, Martin Nöllenburg, and Torsten Ueckerdt. Combinatorial properties of triangle-free rectangle arrangements and the squarability problem. In *Proc. 23rd Symposium on Graph Drawing and Network Visualization (GD)*, volume 9411 of *LNCS*, pages 231–244. Springer, 2015. doi:10.1007/978-3-319-27261-0_20.

[8] Paul Koebe. Kontaktprobleme der Konformen Abbildung. *Ber. Sächs. Akad. Wiss. Leipzig, Math.-Phys. Kl.*, 88:141–164, 1936.

[9] László Lovász. *Graphs and Geometry*, volume 65 of *Colloquium Publications*. AMS, Providence, RI, 2019.

[10] Oded Schramm. Existence and uniqueness of packings with specified combinatorics. *Israel Journal of Mathematics*, 73:321–341, 1991. doi:10.1007/BF02773845.

[11] Oded Schramm. Square tilings with prescribed combinatorics. *Israel Journal of Mathematics*, 84:97–118, 1993. `doi:10.1007/BF02761693`.

Folding Polyiamonds into Octahedra

Eva Bolle[*] Linda Kleist[*]

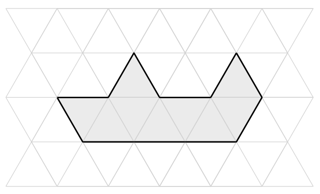
Abstract

We study polyiamonds (polygons arising from the triangular grid) that fold into the smallest yet unstudied platonic solid – the octahedron. Our results are threefold. Firstly, we characterize foldable polyiamonds containing a hole of positive area, namely each but one polyiamond is foldable. Secondly, we show that a convex polyiamond folds into the octahedron if and only if it contains one of five polyiamonds. Finally, we present a sharp size bound: While there exist unfoldable polyiamonds of size 14, every polyiamond of size at least 15 folds into the octahedron.

1 Introduction

Algorithmic origami is a comparatively young branch of computer science that studies the algorithmic aspects of folding various materials. The construction of three-dimensional objects from two-dimensional raw materials is of particular interest and has applications in robotics in general [10, 12], and also in the construction of objects in space [9].

While foldings of polycubes and tetrahedra have already been studied, we take the next step and focus on the question of whether a given polyiamond folds into the octahedron, e.g., does the polyiamond in Figure 1 fold into the octahedron?


(a) A polyiamond P.

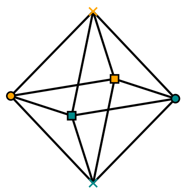

(b) The octahedron $\mathcal{O}$.

Figure 1: Does the polyiamond P fold into the octahedron $\mathcal{O}$?

Terminology. By the *octahedron* $\mathcal{O}$, we refer to the regular octahedron composed of eight equilateral (unit) triangles; for an illustration consider Figure 1(b). Note that four triangles meet in each of the six corners of the octahedron. Because all faces of the octahedron are triangles, our pieces of paper are polygons arising from the triangular grid. A *polyiamond* of size n is a connected polygon in the plane formed by joining n triangles from the triangular grid by identifying some of their common sides; for an example consider Figure 1(a). To avoid confusion with the corners of the octahedron, we refer to the vertices of the triangles forming P as the *vertices* of P; note that these vertices may also lie inside P.

We view P as a set which includes the n open triangles and a subset of the shared unit-length boundary edges; the existence of such an edge models the fact that the triangles are glued along this side. Because we only want robust connections between triangles via their sides, we do not specify the existence or non-existence of vertices which do not influence the foldability. However, for the upcoming definitions of slits and holes, we assume that the vertices do not belong to the polyiamond. If a shared edge does not belong to P, we call it a *slit edge*. We also allow the polyiamonds to have holes; a *hole* of a polyiamond is a bounded connected component of its complement, which is different from a single vertex. We call a hole a *slit* if it has area zero and consists of one or more slit edges. We consider two polyiamonds to be *the same* if they are congruent, i.e., if they can be transformed into one another by a set of translations, rotations and reflections. Moreover, a polyiamond is *convex* if it forms a convex set in the plane (after adding a finite set of points corresponding to vertices).

Folding model. We consider foldings in the *grid folding model*, where folds along the grid lines are allowed such that in the final state every triangle covers a face of the octahedron, i.e., we forbid folding material strictly outside or inside the octahedron. Consequently, in the final state the folding angles are $\pm\beta := \arccos(1/3)$ or $\pm 180°$. Moreover, a folding of a polyiamond P into the octahedron $\mathcal{O}$ induces a *triangle-face-map*, i.e., a mapping of the triangles of P to the faces of $\mathcal{O}$. We say *P folds into $\mathcal{O}$* (or *P is foldable*), if P can be transformed by folds along the grid lines into a folded state such that the induced triangle-face-map is surjective, i.e., each face of $\mathcal{O}$ is covered by at least one triangle. In order to study non-foldable polyiamonds, we also consider partial foldings, i.e., foldings where potentially not all faces of $\mathcal{O}$ are covered. Note that partial foldings induce triangle-face-maps that are not necessarily surjective.

[*]Technische Universität Braunschweig, Germany {eva.bolle,l.kleist}@tu-bs.de

1.1 Related work

Past research has particularly focused on folding polyominoes into *polycubes*. Allowing for folds along the *box-pleat grid* (consisting of square grid lines and alternating diagonals), Benbernou et al. (with differing co-authors) show that every polycube Q of size n can be folded from a sufficiently large square polyomino [6] or from a $2n \times 1$ strip-like polyomino [5]. Moreover, common unfoldings of polycubes have been investigated in the grid model. The *(square) grid model* allows folds along the grid lines of a polyomino with fold angles of $\pm 90°$ and $\pm 180°$, and allows material only on the faces of the polyhedron. Benbernou et al. show that there exist polyominoes that fold into all polycubes with bounded surface area [5] and Aloupis et al. study common unfoldings of various classes of polycubes [4]. Moreover, there exist polyominoes that fold into several different boxes [1, 11, 13, 14, 15].

Decision questions for folding (unit) cubes are studied by Aichholzer et al. [2, 3]. The *half-grid model* allows folds of all degrees along the grid lines, the diagonals, as well as along the horizontal and vertical halving lines of the squares. In this model, every polyomino of size at least 10 folds into the cube [3]. The remaining polyominoes of smaller size are explored by Czajkowski et al. [8]. In the grid model, Aichholzer et al. [3] characterized the foldable tree-shaped polyominoes that fit within a $3 \times n$ strip. Investigating polyominoes with holes, Aichholzer et al. [2] show that all but five *basic* holes (a single unit square, a slit of length 1, a straight slit of length 2, a corner slit of length 2 and a U-shaped slit of length 3) guarantee that the polyomino folds in the grid model into the cube.

In the context of polyiamonds, Aichholzer et al. [3] present a nice and simple characterization of polyiamonds that fold into the smallest platonic solid: Even when restricting to folds along the grid lines, a polyiamond folds into the tetrahedron if and only if it contains one of the two tetrahedral nets.

Results and organization. Our main results are as follows:

- In Section 2, we identify some sufficient and necessary conditions for foldability and take a closer look at polyiamonds with slits and holes.
- Among our findings in Section 3, we characterize foldable polyiamonds containing a hole of positive area: each but one polyiamond is foldable.
- In Section 4, we characterize the convex foldable polyiamonds: A convex polyiamond folds into $\mathcal{O}$ if and only if it contains one of five polyiamonds.
- Lastly, in Section 5, we show that every polyiamond of size ≥ 15 is foldable. An non-foldable polyiamond of size 14 proves that this bound is best possible.

2 Some Tools

In this section, we present tools for proving or disproving the foldability of a polyiamond into an octahedron.

2.1 Foldability

A polyiamond P *contains* a polyiamond P' if P' can be translated, rotated, and reflected such that all triangles and triangle sides of P' also belong to P. Restricting our attention to the triangles, a polyiamond P $\triangle$-*contains* a polyiamond P' if all triangles of P' belong to P. For example, the polyiamond in Figure 7(b) does not contain but $\triangle$-contains the polyiamond in Figure 7(a). As we will see in Observation 1, neither containment nor $\triangle$-containment of a foldable polyiamond is a sufficient folding criterion. Nevertheless, we are able to show two sufficient criteria based on $\triangle$-containment of foldable polyiamonds. By zig-zag-folding as indicated in Figure 2, every polyiamond can be reduced to a contained convex polyiamond.

Lemma 1 *A polyiamond P is foldable if it $\triangle$-contains a convex foldable polyiamond C.*

Proof. Firstly, we reduce P to C: For every boundary side s of C, we fold the triangles of P outside C in a zig-zag-manner. To this end, we fold along the grid lines parallel to s with $+180°$ and $-180°$ folds alternatingly, as illustrated in Figure 2. As a result, the supporting line of s bounds the folded polyiamond. Because C is convex and contained in P, P can be transformed to C with the above procedure. Secondly, we use the fact that C folds into $\mathcal{O}$. $\square$

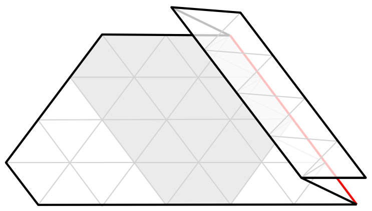


Figure 2: Folding strategy to reduce a polyiamond to a convex subpolyiamond by zig-zag-folding the outside.

A *net* of a polyhedron is formed by cutting along certain edges and unfolding the resulting connected set to lie flat. There exist two interesting facts for nets of 3-dimensional regular convex polyhedra [7]: Firstly, each net is uniquely determined by a spanning tree of the 1-skeleton of the polyhedron, i.e., the cut edges form a spanning tree of the vertex-edge graph. Secondly, dual polyhedra (e.g., the cube and the octahedron) have the same number of nets. Consequently, there exist eleven octahedron nets. They are depicted in Figure 3.

We show that $\triangle$-containing a net is a sufficient folding criterion for a polyiamond.

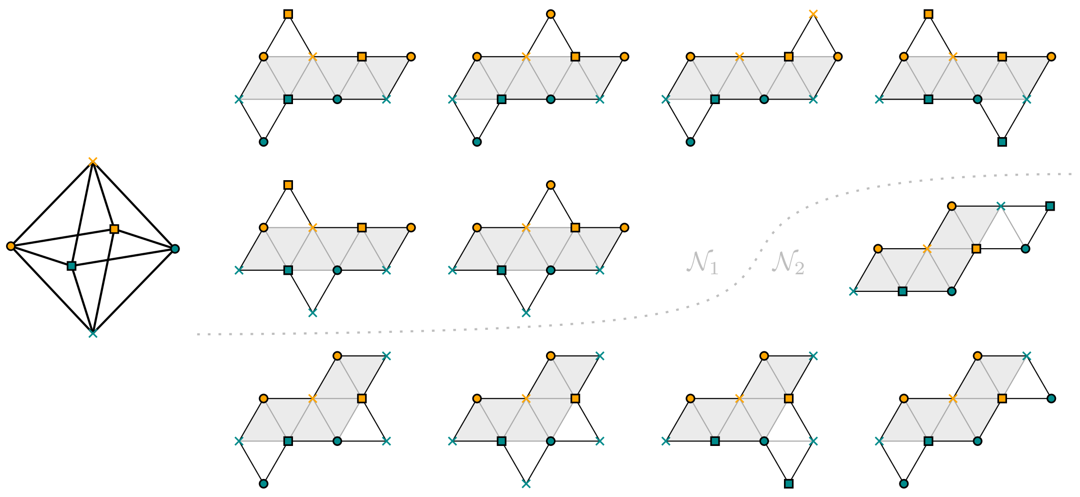


Figure 3: The eleven nets of the octahedron split into two groups $\mathcal{N}_1$ and $\mathcal{N}_2$.

Lemma 2 *A polyiamond is foldable if it △-contains an octahedron net.*

Proof. We partition the set of nets into two groups $\mathcal{N}_1$ and $\mathcal{N}_2$ as illustrated in Figure 3. Note that within each group, the vertex-corner-maps (can be shifted such that they) are consistent on common triangles. For $i = 1, 2$, we consider the smallest convex polyiamond S_i containing the nets of $\mathcal{N}_i$ as depicted in Figure 4. The coloring of the vertices gives a mapping to the corners of $\mathcal{O}$ (where antipodal corners of $\mathcal{O}$ have the same shape but different colors as in Figure 3) and thus describes a folding of S_i into $\mathcal{O}$.

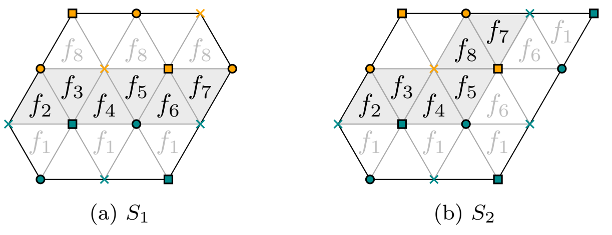


Figure 4: Illustration for the proof of Lemma 2.

By Lemma 1, all polyiamonds that contain a convex polyiamond can be reduced to the convex polyiamond. By construction, not all triangles of S_i are present in each net of $\mathcal{N}_i$. However, each net has at least eight triangles with pairwise different labels. The non-existence of a triangle harms the foldability only if it is essential to cover a face. □

2.2 Non-foldability

As indicated in Figure 5, the triangular grid graph allows for a proper 3-coloring. Because every (connected) inner triangulation has at most one 3-coloring (up to exchange of the colors), every polyiamond has a unique 3-coloring which is induced by the triangular grid. If there exists a slit edge along a grid line, the polyiamond graph may have several vertices corresponding to one grid vertex, see also Figure 7(b). Note that each corner of $\mathcal{O}$ has a unique non-adjacent corner which we call its *antipodal*.

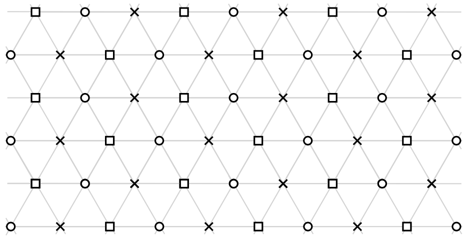


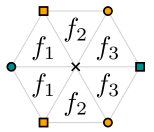
Figure 5: A 3-coloring of the triangular grid.

In order to study the non-foldability, we also consider partial foldings. In particular, when relaxing the condition that all faces are covered, we say a polyiamond is *partially* folded into the octahedron.

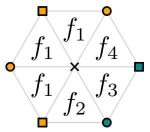
Lemma 3 *Let P be a polyiamond with a 3-coloring of its vertices. In every (partial) folding of P to the octahedron $\mathcal{O}$, the vertices of each color class are mapped to (one corner or a pair of) antipodal corners of $\mathcal{O}$.*

Proof. Consider two neighboring triangles of P and note that their two private vertices have the same color. If their common side is folded by $\pm\beta$, these two vertices are mapped to antipodal corners of $\mathcal{O}$; otherwise the edge is folded by $\pm 180°$ and the two vertices are mapped to the same corner of $\mathcal{O}$. The fact that P is connected implies that every color class is mapped to a different set of antipodal corners. □

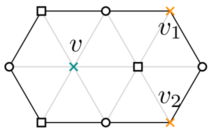
Let C_6 and C_{10} denote the polyiamonds depicted in Figures 6(a) and 6(c), respectively. The following lemma is a crucial tool to disprove foldability.



(a) A triangle-face-map of C_6 covering 3 faces of $\mathcal{O}$.



(b) A triangle-face-map of C_6 covering 4 faces of $\mathcal{O}$.



(c) Any triangle-face-map of C_{10} covers at most 6 faces of $\mathcal{O}$.

Figure 6: Illustration of Lemma 4 and its proof.

Lemma 4 *Let P be a polyiamond (partially) folded into the octahedron $\mathcal{O}$.*

(i) Every C_6 $\triangle$-contained in P covers at most 4 different faces of $\mathcal{O}$.

(ii) If a C_6 in P covers exactly 3 or 4 faces, then the induced triangle-face-mapping is unique (up to symmetry) and as depicted in Figures 6(a) and 6(b), respectively.

(iii) Every C_{10} contained in P covers at most 6 different faces of $\mathcal{O}$.

Proof. Let v denote the central vertex of C_6. In the folded state, v is mapped to a corner c of the octahedron which is (like every corner) incident to 4 faces.

(i) Because every triangle of C_6 is incident to vertex v, these triangles cover a subset of the 4 faces incident to c.

(ii) We consider a 3-coloring as indicated in Figure 6(a). If all circle or all square vertices map to a same corner, then C_6 covers at most two faces of $\mathcal{O}$, namely the ones incident to the cross and circle vertex. Hence, if P covers 3 or 4 faces, then exactly two vertices of each class map to the same corner and the third vertex of each class maps to its antipodal, we call this vertex *lonely*. We distinguish whether the two lonely vertices are a) adjacent or b) opposite in C_6, see Figures 6(a) and 6(b). It follows that the number of covered faces is 3 and 4, respectively.

(iii) We consider a 3-coloring of C_{10} as illustrated in Figure 6(c) and use the fact that each color class is mapped to antipodal corners by Lemma 3. We denote the three cross vertices by v, v_1, v_2 as illustrated in Figure 6(c); similarly, we denote the corner of $\mathcal{O}$ to which v is mapped by c. If at most one v_i (which are both incident to only two triangles) is mapped to the antipodal corner $\overline{c}$ of c, then at most 2 faces incident to $\overline{c}$ can be covered. If both v_1 and v_2 are mapped to $\overline{c}$, then the 4 incident triangles of $\overline{c}$ share a common edge. Consequently, they may cover at most 2 incident faces. In other words, in both cases at least 2 faces (incident to $\overline{c}$) remain uncovered.

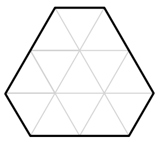
This completes the proof. □

3 On Slits and Holes

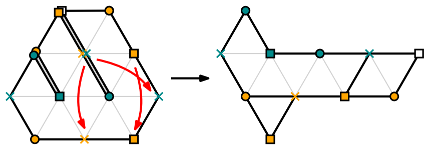
In this section, we consider polyiamonds with slits and holes. First of all, we remark that removing individual edges from a foldable polyiamond does not destroy its foldability as long as connectivity is maintained. This allows us to focus on polyiamonds without slit edges, i.e., *sealing* slit edges may only increase the level of difficulty to prove foldability. On the other hand, we note that slits may in fact enable foldability.

Observation 1 *Let P be a polyiamond ($\triangle$-)containing a foldable polyiamond P'. Then, the polyiamond P may not be foldable.*

As we show in Theorem 6, the polyiamond P depicted in Figure 7(a) does not fold into $\mathcal{O}$, while the polyiamond P' with additional slit edges in Figure 7(b) can be transformed into a polyiamond containing a net. Hence, P' is foldable by Lemma 2.



(a) By Theorem 6, this polyiamond does not fold into $\mathcal{O}$.



(b) With additional slit edges, the polyiamond folds into $\mathcal{O}$.

Figure 7: Illustration for Observation 1.

We now characterize foldable polyiamonds with holes of positive area. Let O denote the polyiamond illustrated in Figure 8.

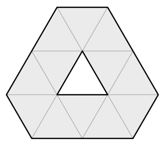


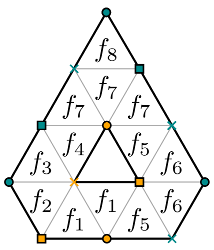
Figure 8: Polyiamond O

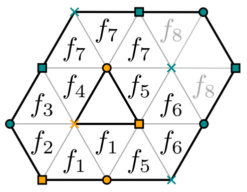
Theorem 5 *Let P be a polyiamond containing a hole h of positive area. Then P folds into $\mathcal{O}$ if and only if it is not the polyiamond O.*

Proof. The non-foldability of O is analogous to the proof that the polyiamond depicted in Figure 7(a) is non-foldable, see Claim 4 below.

For the reverse direction, we focus on a largest hole h with positive area and distinguish two cases:

If h contains two neighboring triangles, then we reduce P to the polyiamond P_c depicted in Figure 9(c) as follows: we choose two neighboring triangles of h which form a (potentially smaller) hole h' in the form of a parallelogram. Then, we fold all triangles that do not touch h' with a vertex or edge by zig-zag-folding the outside as in Figure 2. This results in the polyiamond P_c because h and thus h' are enclosed by a cycle of triangles of P. Moreover, it is easy to check that P_c is foldable, e.g., when inducing the triangle-face-map depicted in Figure 9(c).


(a) Polyiamond P_a


(b) Polyiamond P_b

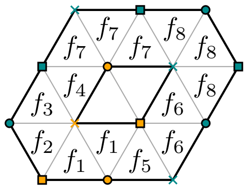

(c) Polyiamond P_c

Figure 9: llustration for the proof of Theorem 5.

It remains to consider the case that h contains a triangle and P is not O. If P can be reduced (by zig-zag-folding) to the polyiamond P_a depicted in Figure 9(a), then P folds into $\mathcal{O}$. Otherwise, we use zig-zag-folds to obtain a subpolyiamond P' of P_b depicted in Figure 9(b). Because P is different from O and cannot be reduced to P_a, this ensures that P' has at least one triangle with label f_8. Because P_b folds into $\mathcal{O}$, so does P. ☐

4 Characterization for Convex Polyiamonds

In this section, we characterize convex foldable polyiamonds. Let $\mathcal{C}$ denote the set of five convex polyiamonds depicted in Figure 10.

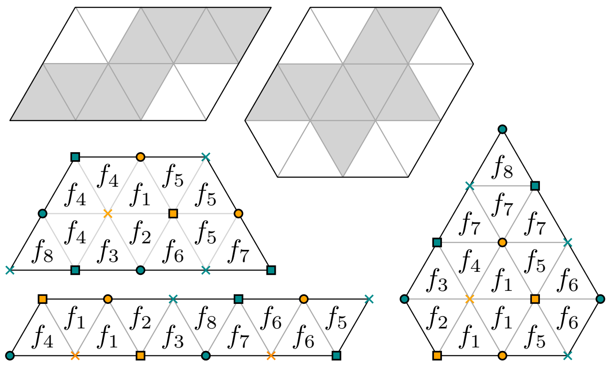


Figure 10: Illustration for Theorem 6; the set $\mathcal{C}$ of foldable polyiamonds and their foldings.

Theorem 6 *A convex polyiamond P folds into $\mathcal{O}$ if and only if it contains one of the five polyiamonds in $\mathcal{C}$.*

Proof. First, we show that a convex polyiamond P folds into $\mathcal{O}$ if it ($\triangle$-)contains a polyiamond in $\mathcal{C}$. Note that each polyiamond in $\mathcal{C}$ is convex. Hence, by Lemma 1, it suffices to present folding strategies for the polyiamonds in $\mathcal{C}$, see Figure 10. While two polyiamonds contain a octahedral net, we present explicit strategies for the remaining three.

Second, we show that every convex polyiamond that folds into $\mathcal{O}$ contains a polyiamond from $\mathcal{C}$. To do so, we construct all convex $\mathcal{C}$*-free* polyiamonds, i.e., all convex polyiamonds that contain none of the five polyiamonds in $\mathcal{C}$. The construction is as follows, for an illustration consider Figure 12: We start with the unique polyiamond of size 1. Then, we consider all possibilities to enlarge every constructed polyiamond by one triangle and extend it to the smallest convex polyiamond containing it, i.e., we add just enough triangles such that the resulting polyiamond is convex again. We stop when we encounter a polyiamond from $\mathcal{C}$ or a polyiamond containing one of them.

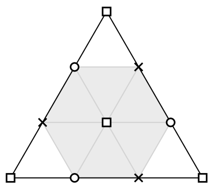
By their convexity and Lemma 1, it suffices to show the non-foldability of the inclusion-wise maximal $\mathcal{C}$-free polyiamonds. The construction shows that the set $\overline{\mathcal{C}}$ of inclusion-wise maximal $\mathcal{C}$-free polyiamonds consists of the four polyiamonds $\overline{C}_1 :=$ o, $\overline{C}_2 :=$ w, $\overline{C}_3 :=$ s, and $\overline{C}_4 :=$ p, i.e., each $\mathcal{C}$-free polyiamond is contained in some polyiamond in $\overline{\mathcal{C}}$. It remains to show that all of these do not fold into $\mathcal{O}$.

Claim 1 *The polyiamond $\overline{C}_1$ does not fold into $\mathcal{O}$.*

The polyiamond $\overline{C}_1$ contains a C_6, see Figure 11(a). By Lemma 4(i), the contained C_6 covers at most 4 faces of the octahedron $\mathcal{O}$. Hence, in every partial folding of $\overline{C}_1$ into $\mathcal{O}$, $\overline{C}_1$ covers at most 7 faces of $\mathcal{O}$. Consequently, it does not fold into $\mathcal{O}$.

Claim 2 *The polyiamond $\overline{C}_2$ does not fold into $\mathcal{O}$.*

For the purpose of a contradiction, we assume that $\overline{C}_2$ does fold into $\mathcal{O}$. We consider a 3-coloring as illustrated


(a) A C_6 in $\overline{C}_1$.

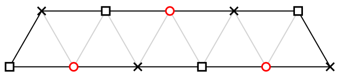

(b) A 3-coloring of $\overline{C}_2$.

Figure 11: Illustration for the proof that $\overline{C}_1$ and $\overline{C}_2$ do not fold into $\mathcal{O}$.

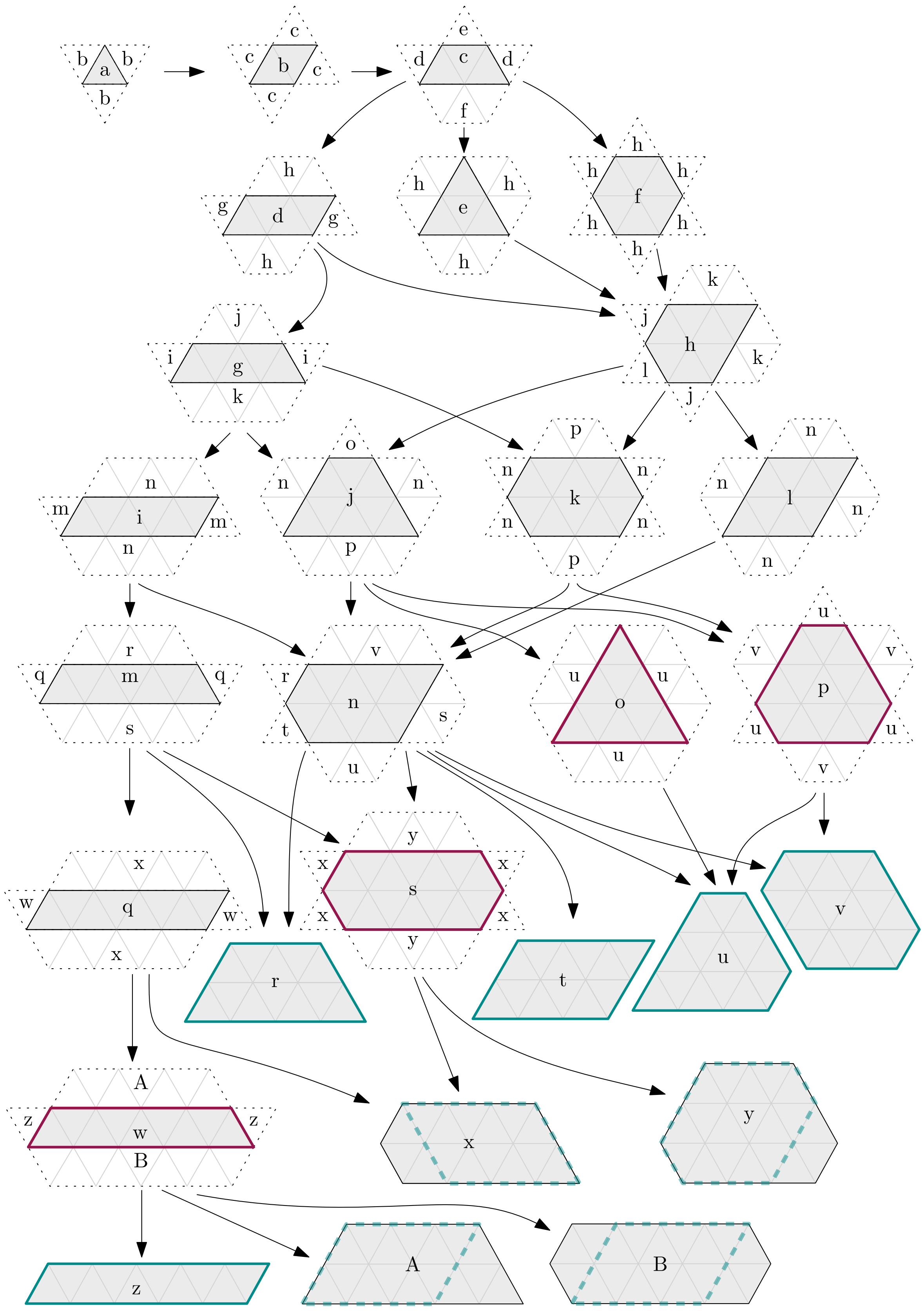
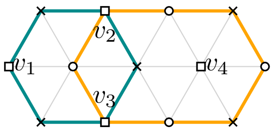


Figure 12: Construction of all $\mathcal{C}$-free polyiamonds; the inclusion-wise maximal $\mathcal{C}$-free polyiamonds o, p, s, and w are highlighted in red.

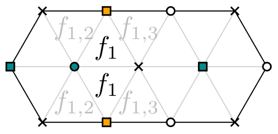
in Figure 11(b) and use the fact that each color class is mapped to a pair of antipodal pairs by Lemma 3. Note that there exist only three circle vertices, each of which is adjacent to three faces of $\overline{C}_2$. Hence, one (of the two antipodal) corner of $\mathcal{O}$ is covered by only one circle vertex implying that not all of its incident faces are covered.

Claim 3 *The polyiamond $\overline{C}_3$ does not fold into $\mathcal{O}$.*

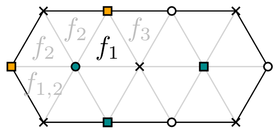
The polyiamond $\overline{C}_3$ can be viewed as copies of C_6 and C_{10} overlapping in two triangles, see Figure 13(a). For the purpose of a contradiction, we consider a 3-coloring of $\overline{C}_3$ as illustrated in Figure 13(b). Note that there are four square vertices in total; we denote them by v_1, v_2, v_3, v_4. Moreover, the three leftmost square vertices cannot all map to the same corner c; otherwise the left C_6 maps to at most 2 faces and $\overline{C}_3$ covers at most $2+6-1=7$ faces. We distinguish two cases.



(a) $\overline{C}_3$ consists of a C_6 and a C_{10} overlapping in 2 triangles.



(b) Case: v_2 and v_3 are mapped to same corner.



(c) Case: v_2 and v_3 are mapped to antipodal corners.

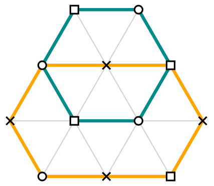
Figure 13: Illustration for the non-foldability of $\overline{C}_3$.

If v_2 and v_3 are mapped to c (and v_1 to the antipodal corner $\bar{c}$), then all of their 6 incident triangles contain one of two neighboring edges of c; for an illustration consider Figure 13(b). Hence, they cover at most three faces incident to c. Moreover, v_4 must map to $\bar{c}$; otherwise the two triangles incident to v_1 are the only ones mapping to any of the four faces incident to $\bar{c}$. Consequently, all remaining triangles map to a face incident to $\bar{c}$ and thus, they are not able to cover the remaining face incident to c. A contradiction.

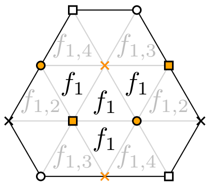
It remains to consider the case that v_2 and v_3 are mapped to two antipodal corners c and $\bar{c}$, respectively. We may assume without loss of generality that v_1 is mapped to the corner c as illustrated in Figure 13(c). Then v_4 is mapped to the antipodal $\bar{c}$; otherwise not all faces of $\bar{c}$ are covered. Consequently, all triangles incident to c are incident to v_1 and v_2. However, four (of the five) triangles incident to v_1 and v_2 share one edge of $\mathcal{O}$. Hence the five triangles of v_1 and v_2 cover at most 3 faces incident to c. A contradiction to the foldability of $\overline{C}_3$.

Claim 4 *The polyiamond $\overline{C}_4$ does not fold into $\mathcal{O}$.*

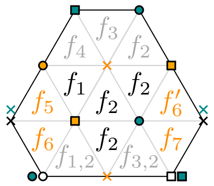
The polyiamond $\overline{C}_4$ consists of a C_{10} and a C_6 overlapping in three triangles as illustrated in Figure 14(a). If their intersection is mapped to three different faces of $\mathcal{O}$, then by Lemma 4(i) and (iii), $\overline{C}_4$ covers at most $4+6-3=7$ faces of $\mathcal{O}$. Consequently, in every folding of $\overline{C}_4$ into $\mathcal{O}$, the triangles in the considered intersection map to at most 2 distinct faces. In the following, we focus on the four central triangles of $\overline{C}_4$. By the above observation and the rotational symmetry of $\overline{C}_4$, the triangles of each 'line' are mapped to at most 2 distinct faces. We distinguish two cases.



(a) $\overline{C}_4$ consists of a C_6 and a C_{10} overlapping in 3 triangles.



(b) Case: the four central triangles map to the same face.



(c) Case: the four central triangles map to two faces.

Figure 14: Illustration for the non-foldability of $\overline{C}_4$.

If all four triangles map to the same face, denoted by f_1, then consider Figure 14(b). By their common edge incident to f_1, each of the two triangles with label $f_{1,i}$ Figure 14(b), $i \in \{2,3,4\}$, cover at most one face different from f_1. Consequently, at most 7 faces can be covered in total and this case does not yield a folding of $\overline{C}_4$ into $\mathcal{O}$.

If the four central triangles map to 2 different faces, then by Lemma 3, the map is as illustrated in Figure 14(c). By Lemma 4(i) and (iii), the copy of C_6 covers at most 4 faces and the copy of C_{10} covers at most 6 faces. Subtracting the double count of the intersection, the triangles of $\overline{C}_4$ cover at most $4+6-2=8$ faces. Hence, by Lemma 4(ii), the top copy of C_6 covers exactly 4 faces and is consistent with the triangle-face-map of Figure 6(b). Note that the two triangles with label $f_{i,2}$, $i \in \{1,3\}$, in Figure 14(c) contain the common edge of f_i and f_2 and thus they may not cover new faces of $\mathcal{O}$. It follows that the remaining four triangles cover distinct and new faces of $\mathcal{O}$. However, this implies that two triangles f_6 and f'_6 are mapped to the same face. A contradiction. Hence, $\overline{C}_4$ does not fold into the octahedron. □

5 A Sharp Size Bound

As shown in Claim 3, the polyiamond $\overline{C}_3$ is not foldable, i.e. there exist polyiamonds of size 14 that do not fold into $\mathcal{O}$. In this section, we show the following complementing theorem.

Theorem 7 *Every polyiamond P of size $\geq$ 15 folds into $\mathcal{O}$.*

To present an idea of the proof, we give some useful sufficient conditions and a simple upper bound. Let P be a polyiamond and ℓ some grid line. The *ℓ-width* of P denotes the size of the polyiamond obtained by folding all edges parallel to ℓ in a zig-zag-manner as indicated in Figure 2. The *width* of P is the maximum of the three different ℓ-widths. Because the convex polyiamond $P_- := z$, depicted in Figure 12, folds into $\mathcal{O}$, we obtain the following.

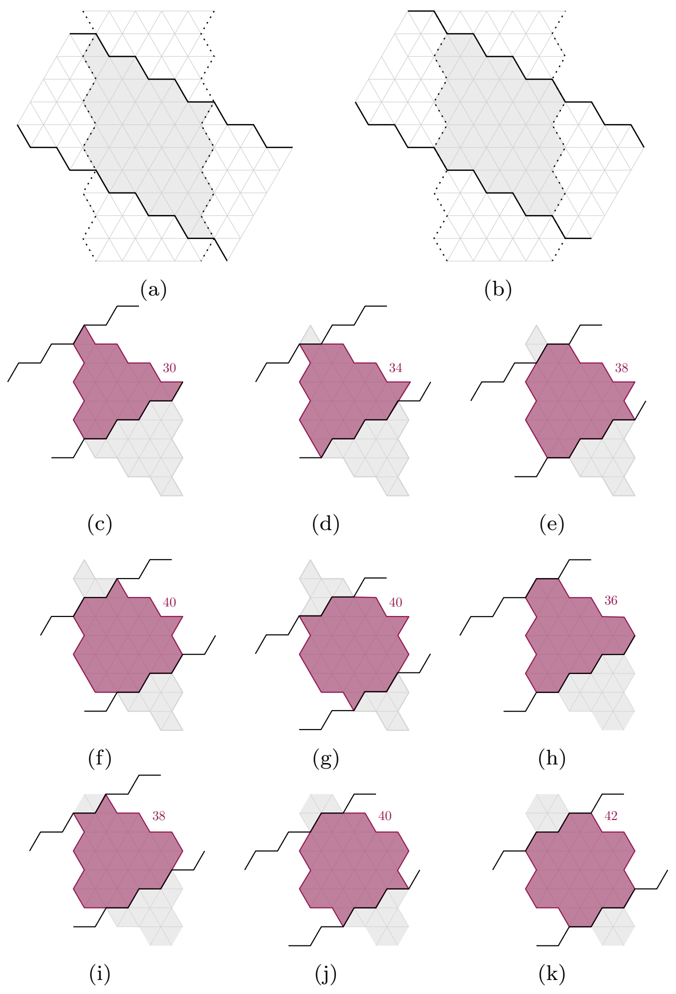


Figure 15: Illustration for the proof of Corollary 9. Construction of the maximal polyiamonds of width $\leq$ 9; their sizes are indicated by numbers.

Lemma 8 *Every polyiamond P of width at least 10 folds into $\mathcal{O}$.*

Proof. Because P has width 10, it can be folded into the polyiamond P_- by zig-zag-folds. Then, by Theorem 6, P_- can be folded into $\mathcal{O}$. □

Moreover, we determine an upper bound on the size of polyiamonds of width $\leq$ 9. In particular, they have size $\leq$ 42 which yields a nice and simple upper bound.

Corollary 9 *Every polyiamond of size > 42 folds into $\mathcal{O}$.*

Proof. Let P be a polyiamond that does not fold into $\mathcal{O}$. Then, by Lemma 8, P has width $\leq$ 9. Consequently, P is contained in the intersection of three strips of width 9 with different rotation. As illustrated in Figures 15(a) and 15(b), two of these infinite strips may intersect in two distinct ways. The intersection with a third strip results in nine polyiamonds (six of which are pairwise different), see Figure 15. The largest of these polyiamonds has size 42 and is depicted in Figure 15(k). Because P is contained in one of them, P has size at most 42. Consequently, any larger polyiamond is foldable. □

To show the sharp bound, we need to work a little harder. In particular, the proof is computer-aided.

Proof sketch of the sharp upper bound. Theorem 7 is based on a strong sufficient criterion. Let P_X, P_U, P_Z, and P_L denote the polyiamonds depicted in Figures 16(a) to 16(d), respectively. In a first step, we show that polyiamonds that are large enough and do contain one of the four polyiamonds fold into $\mathcal{O}$.

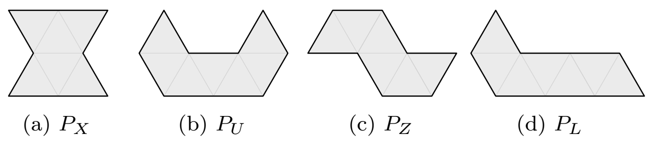


Figure 16: Illustration of the four polyiamonds used in Proposition 10.

Proposition 10 *Every polyiamond P that $\triangle$-contains P_X, P_U, P_Z, or P_L and has size $\geq$ 15 folds into $\mathcal{O}$.*

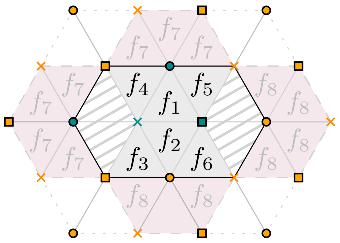
We call a polyiamond $\mathcal{P}$*-free* if it does not $\triangle$-contain any of the polyiamonds P_-, P_X, P_U, P_Z, or P_L. By Theorem 6 and Proposition 10, it remains to show that no $\mathcal{P}$-free polyiamond of size $\geq$ 15 exists. To do so, we construct all $\mathcal{P}$-free polyiamonds bottom-up with computer assistance and observe that indeed no such polyiamond exists.

In the following, we present the remaining details. We start with four lemmas proving Proposition 10.

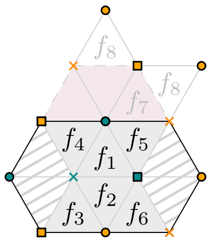
Lemma 11 *Every polyiamond P that $\triangle$-contains P_X and has size ≥ 15 folds into $\mathcal{O}$.*

Proof. We consider the C_{10}-frame containing P_X as illustrated in Figure 17(a) and call each connected group of rose triangles a *flap* of P_X. We consider the following cases:

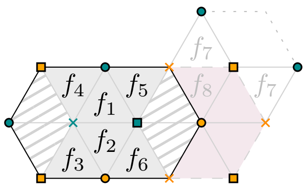
If triangles exist in two distinct flaps, then there exists a triangle-face-map such that some triangles are mapped to (the two missing faces) f_7 and f_8, see Figure 17(a) (or its mirror image). First, we fold away all (but at most 2) triangles that are not contained in the two flaps. The two corner triangles between two flaps may remain. Its foldability is implied by the fact that the polyiamond in Figure 17(a) folds into $\mathcal{O}$.



(a) two flaps



(b) top or bottom flap



(c) left or right flap

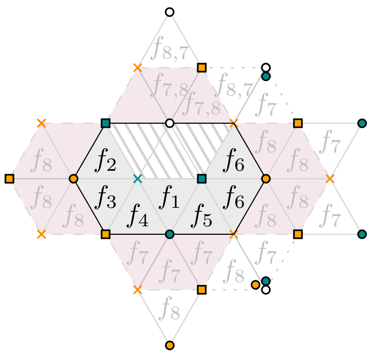
Figure 17: Illustration of the proof of Lemma 11.

It remains to consider the case that P without C_{10} is attached via only one flap. Because P has size at least 15 and each flap has size at most 4, there exists a triangle outside the flap. Figures 17(b) and 17(c) shows that this guarantees foldability in all cases. □

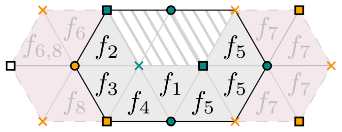
Lemma 12 *Every polyiamond P that $\triangle$-contains P_U and has size ≥ 15 folds into $\mathcal{O}$.*

Proof. We may assume that P does not $\triangle$-contain C_{10} (which $\triangle$-contains P_X); otherwise Lemma 11 implies the claim. Consequently, P has 6 triangles outside the C_{10}-frame containing P, see Figure 18(a). We distinguish two cases: a) there exists a triangle in some flap with a neighboring triangle outside the flap or b) all triangles of P lie within the flaps.

In case a), the map in Figure 18(a) can be reflected (horizontally) such that there exist triangles with labels f_7 and f_8. The label $f_{7,8}$ indicates that it can be adjusted as wished. Moreover, P folds into $\mathcal{O}$ by some strategy



(a)



(b)

Figure 18: Illustration of the proof of Lemma 12.

presented in Figure 18(a) (after reducing to a crucial convex subpolyiamond).

In case b), unless all triangles lie within the left and right flap, the map in Figure 18(a) can be reflected such that there exist faces with labels f_7 and f_8. Moreover, P folds into $\mathcal{O}$ by the strategy presented in Figure 18(a).

If all triangles lie within the left and right flap, we consider the strategy indicated in Figure 18(b). By symmetry, we may assume that the left flap contains at least 3 triangles. Hence, there exist faces with label f_6 and f_8; moreover, a triangle with label f_7 exists in the right flap. As the polyiamond in Figure 18(b) folds into $\mathcal{O}$, P does as well. This completes the proof. □

Lemma 13 *Every polyiamond P that $\triangle$-contains P_Z and has size ≥ 15 folds into $\mathcal{O}$.*

Proof. We may assume that P does not $\triangle$-contain P_X nor P_U; otherwise Lemmas 11 and 12 imply the claim. Consequently, P has seven triangles outside the C_{10}-frame depicted in Figure 19(a).

Similar as above, we consider the case that there exists a flap with a neighboring triangle outside the flap. Then, the map in Figure 19(a) induces triangles with labels f_7 and f_8. Moreover, the depicted polyiamonds folds into $\mathcal{O}$ and $\triangle$-contains P. The same argument can be applied for the case that there exist triangles in flaps with different labels.

It remains to consider the case that all triangles are contained in neighboring flaps with the same labels. By the rotational symmetry, we may assume that all triangles are contained in the top and right flap as illustrated in Figure 19(b); moreover, we know that all of these triangles are present because at least 7 triangles exist outside the C_{10}-frame. The illustrated polyiamond folds into $\mathcal{O}$ and $\triangle$-contains P. Thus P folds into $\mathcal{O}$. □

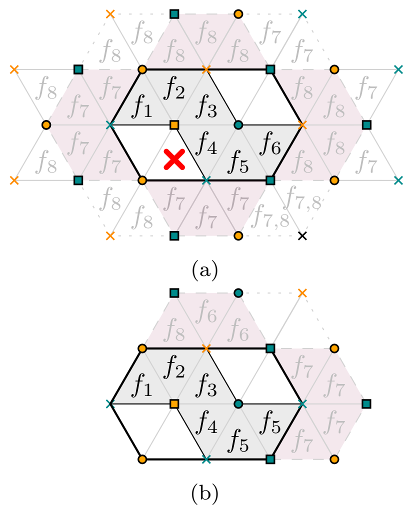


Figure 19: Illustration for the proof of Lemma 13.

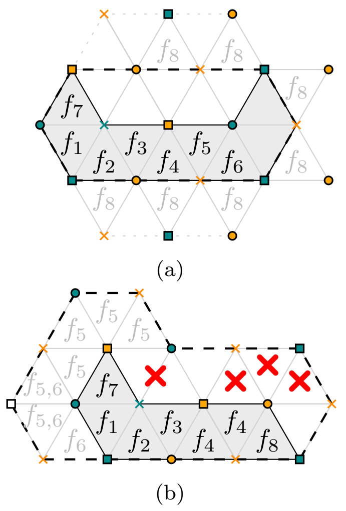


Figure 21: Illustration for the proof of Lemma 14.

Lemma 14 *Every polyiamond P that $\triangle$-contains P_L and has size ≥ 15 folds into $\mathcal{O}$.*

Proof. Observe that every triangle outside the dashed frame in Figure 20 yields a triangle with the missing label f_8. Hence, we may assume that P is contained in the frame. Moreover, we may assume that P does not contain P_U nor P_Z; otherwise Lemmas 12 and 13 imply the claim. Consequently, at least 3 triangles are missing within the frame as indicated, where crosses on an edge indicate that at most one of the incident triangles exist.

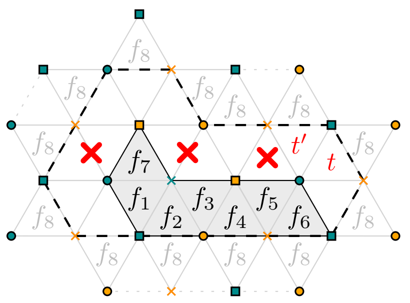


Figure 20: Illustration for the proof of Lemma 14.

We distinguish two cases: If P contains the triangle t, then it contains the polyiamond depicted in Figure 21(a). Because the frame contains only 14 triangles, there exists a triangle f outside the frame. Together with the depicted map (or its mirror image), f ensures a triangle with label f_8.

If P does not contain the triangle t, then its left neighboring triangle t' does not belong to P because P is contained in the dashed frame, see Figure 21(b). The frame-polyiamond depicted in Figure 21(b) has four triangles with label f_5, one with f_6 and two with joker label $f_{5,6}$ (indicating that these can be adjusted as wished). Because P does not $\triangle$-contain P_Z nor P_U, at most 3 of the remaining triangles without labels exist. It follows that for any choice of 8 triangle, triangles with different labels exist. Moreover, the depicted polyiamonds folds into $\mathcal{O}$. $\square$

Together, Lemmas 11 to 14 imply Proposition 10. We are now ready to prove the sharp upper bound, namely Theorem 7.

Theorem 7 *Every polyiamond P of size ≥ 15 folds into $\mathcal{O}$.*

Proof. We call a polyiamond $\mathcal{P}$*-free* if it does not $\triangle$-contain any of the polyiamonds P_-, P_X, P_U, P_Z, or P_L. By Theorem 6 and Proposition 10, it remains to show that no $\mathcal{P}$-free polyiamond of size ≥ 15 exists. To do so, we construct all $\mathcal{P}$-free polyiamonds bottom-up and show that indeed there exists no such polyiamond. The construction is similar to the one in the proof of Theorem 6: We start with the polyiamond of size 1. Then, we enlarge every $\mathcal{P}$-free polyiamond of size k by individual triangles and check if the resulting polyiamonds remain $\mathcal{P}$-free. In this way, we obtain a list of $\mathcal{P}$-free polyiamonds of size $k+1$. Table 1 presents the numbers $p(n)$ of $\mathcal{P}$-free polyiamonds with size n; these numbers have been generated by computer-search.

Table 1: The number $p(n)$ of $\mathcal{P}$-free polyiamonds of size n.

n	2	3	4	5	6	7	8
$p(n)$	1	1	3	4	10	16	22

n	9	10	11	12	13	14	15
$p(n)$	22	16	9	3	1	0	0

The code is available at `https://github.com/dasnessie/folding-polyiamonds/`. $\square$

References

[1] Z. Abel, E. Demaine, M. Demaine, H. Matsui, G. Rote, and R. Uehara. Common developments of several different orthogonal boxes. In *Canadian Conference on Computational Geometry (CCCG '11)*, 2011.

[2] O. Aichholzer, H. Akitaya, K. Cheung, E. Demaine, M. Demaine, S. P. Fekete, L. Kleist, I. Kostitsyna, M. Löffler, Z. Masárová, and K. Mundilova. Folding polyominoes with holes into a cube. *Computational Geometry (CGTA)*, 93:101700, 2020.

[3] O. Aichholzer, M. Biro, E. D. Demaine, M. L. Demaine, D. Eppstein, S. P. Fekete, A. Hesterberg, I. Kostitsyna, and C. Schmidt. Folding polyominoes into (poly)cubes. *International Journal of Computational Geometry & Applications (IJGCA)*, 28:197–226, 2018.

[4] G. Aloupis, P. K. Bose, S. Collette, E. D. Demaine, M. L. Demaine, K. Douïeb, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Common unfoldings of polyominoes and polycubes. In *Computational Geometry, Graphs and Applications (CGGA '10)*, pages 44–54. Springer, 2010.

[5] N. M. Benbernou, E. D. Demaine, M. L. Demaine, and A. Lubiw. Universal hinge patterns for folding strips efficiently into any grid polyhedron. *Computational Geometry*, page 101633, 2020.

[6] N. M. Benbernou, E. D. Demaine, M. L. Demaine, and A. Ovadya. Universal hinge patterns for folding orthogonal shapes. *Origami^5: Proceedings of the 5th International Conference on Origami in Science, Mathematics and Education (OSME '11)*, pages 405–419, 2011.

[7] F. Buekenhout and M. Parker. The number of nets of the regular convex polytopes in dimension ≤ 4. *Discrete Mathematics*, 186(1):69 – 94, 1998.

[8] K. Czajkowski, E. D. Demaine, M. L. Demaine, K. Eppling, R. Kraft, K. Mundilova, and L. Smith. Folding small polyominoes into a unit cube. In *Canadian Conference on Computational Geometry (CCCG '20)*, 2020.

[9] E. Hawkes, B. An, N. M. Benbernou, H. Tanaka, S. Kim, E. D. Demaine, D. Rus, and R. J. Wood. Programmable matter by folding. *Proceedings of the National Academy of Sciences*, 107(28):12441–12445, 2010.

[10] K. Kuribayashi, K. Tsuchiya, Z. You, D. Tomus, M. Umemoto, T. Ito, and M. Sasaki. Self-deployable origami stent grafts as a biomedical application of ni-rich tini shape memory alloy foil. *Materials Science and Engineering: A*, 419:131–137, 03 2006.

[11] J. Mitani and R. Uehara. Polygons folding to plural incongruent orthogonal boxes. In *Canadian Conference on Computational Geometry (CCCG '08)*, pages 39–42, 2008.

[12] A. Qattawi, M. Abdelhamid, A. Mayyas, and M. Omar. Design analysis for origami-based folded sheet metal parts. *SAE International Journal of Materials and Manufacturing*, 7(2):488–498, 2014.

[13] T. Shirakawa and R. Uehara. Common developments of three incongruent orthogonal boxes. *International Journal of Computational Geometry & Applications (IJGCA)*, 23(01):65–71, 2013.

[14] R. Uehara. A survey and recent results about common developments of two or more boxes. In *Origami^6: Proceedings of the 6th International Meeting on Origami in Science, Mathematics and Education (OSME '14)*, volume 1, pages 77–84, 2014.

[15] D. Xu, T. Horiyama, T. Shirakawa, and R. Uehara. Common developments of three incongruent boxes of area 30. *Computational Geometry (CGTA)*, 64:1–12, 2017.

Folding Points to a Point and Lines to a Line

Hugo A. Akitaya[*] Brad Ballinger[†] Erik D. Demaine[‡] Thomas C. Hull[§] Christiane Schmidt[¶]

Abstract

We introduce basic, but heretofore generally unexplored, problems in computational origami that are similar in style to classic problems from discrete and computational geometry.

We consider the problems of folding each corner of a polygon P to a point p and folding each edge of a polygon P onto a line segment ℓ that connects two boundary points of P and compute the number of edges of the polygon containing p or ℓ limited by crease lines and boundary edges.

1 Introduction

Many classic problems from discrete and computational geometry that concern simple statements about points and lines in the plane, such as, “Given n points in the plane, how do we determine their convex hull?” or “Into how many regions do n lines in general position divide the plane?” Similarly basic questions can be asked about origami, but none seem to have been fully explored in the literature. In this paper we investigate two such questions in computational origami. Both involve starting with a convex-polygon piece of paper P.

1. Fold and unfold each corner of P, in turn, to a chosen point $p \in P$ so that p will be contained in a polygon Q_p whose interior is uncreased and sides are either the crease lines or the boundary edges of P; see Figure 1(a). How many sides can Q_p have?

2. Let $a, b \in \partial P$ (the boundary of P), and let ℓ' be the line that contains the line segment $\ell = \overline{ab}$. Fold each side of P onto ℓ', and let Q_ℓ be the polygon limited by the crease lines that contains ℓ; see Figure 1(b). How many sides can Q_ℓ have?

In these problems the point p and line ℓ may be chosen to lie on the boundary of P. Our aim is to find straightforward methods for calculating $|Q_p|$ and $|Q_\ell|$ so as to create visualizations for which choices of p and ℓ will give us different answers.

[*]Department of Computer Science, University of Massachusetts Lowell, hugo_akitaya@uml.edu

[†]Humbolt State University, bradley.ballinger@humboldt.edu

[‡]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, edemaine@mit.edu

[§]Department of Mathematics, Western New England University, thull@wne.edu

[¶]Communications and Transport Systems, ITN, Linköping University, christiane.schmidt@liu.se

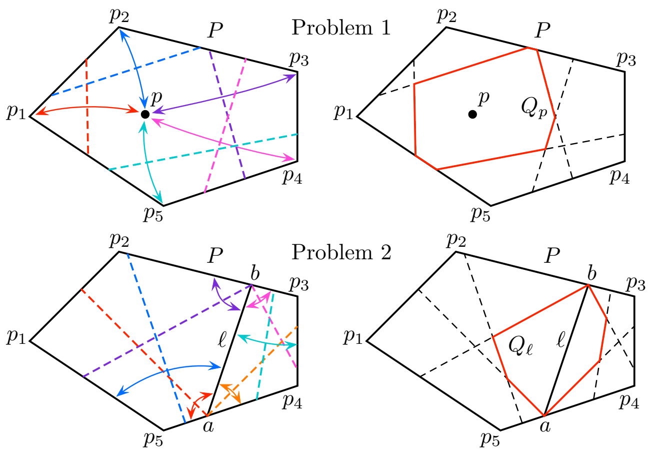


Figure 1: Illustrations of Problems 1 and 2.

Problem 1 was first investigated by Kazuo Haga [6, 7, 8], but only in the case where P is a square. We will see that the full version of Problem 1, including the case where P is the whole plane, is a natural application of Voronoi diagrams and Delaunay triangulations. Problem 2 is solved using event circles of the straight skeleton of P.

2 Folding Points to a Point

To first simplify Problem 1, let $S = \{p_1, \ldots, p_n\}$ be the vertices of the polygon P and let us, for now, ignore the sides of P, focusing on only folding an arbitrarily-chosen point p to the points in S; call this *Problem 1a.* We introduce notation for Voronoi diagrams (see [2, 4]). Given $a, b \in \mathbb{R}^2$, define the *halfplane determined by a and b that contains a* to be

$$h(a, b) = \{x \in \mathbb{R}^2 \mid ||x - a|| \leq ||x - b||\},$$

where $||.||$ denotes Euclidean norm. The *Voronoi region* $\mathrm{Vor}(p, A)$ containing point p relative to a finite point set A is defined as

$$\mathrm{Vor}(p, A) = \{x \in \mathbb{R}^2 \mid ||x-p|| \leq ||x-a|| \text{ for all } a \in A\}.$$

A standard result (see [4, Theorem 4.1]) is that $\mathrm{Vor}(p, A)$ is equal to the intersection of halfplanes determined by p and the points in A:

Theorem 1

$$\mathrm{Vor}(p, A) = \bigcap_{a \in A} h(p, a).$$

The *Voronoi diagram* for a finite point set $S = \{p_1, \ldots, p_n\}$ is then the collection of Voronoi regions $\mathrm{Vor}(p_i, S \setminus \{p_i\})$ for $i = 1, \ldots, n$.

Theorem 2 *Let Q_p be the polygon containing p limited by the crease lines made from folding p to each point in S. Then $Q_p = \mathrm{Vor}(p, S)$.*

Proof. This follows almost immediately from the fact that when we fold a point $p_i \in S$ to the point p, the crease line L that is made is the perpendicular bisector of the segment joining p_i and p. But L is also the boundary of the halfplane $h(p, p_i)$, and Q_p will be contained in all such halfplanes. Furthermore, any point that is contained in all the halfplanes $h(p, p_i)$ will, by definition, also be in Q_p. This proves the result. □

This connection between origami and Vonoroi diagrams is known to origami artists. In fact, one of the easiest ways to construct a Voronoi diagram is to draw the point set S on a piece of paper and carefully fold pairs of points in S together, creasing the various halfplane boundaries. See [11] for more details.

We let $\mathrm{conv}(S)$ denote the convex hull of the set S. The definition of Voronoi region implies that $\mathrm{Vor}(p, S)$ will be unbounded if p is not in the interior of $\mathrm{conv}(S)$. Thus Theorem 2 implies the following:

Corollary 3 *If $p \notin \mathrm{int}(\mathrm{conv}(S))$, then Q_p is unbounded.*

Now let $\mathrm{Del}(S)$ denote the Delaunay triangulation of a finite point set S. For details on $\mathrm{Del}(S)$, see [2, 4]; we remind the reader of three key properties of $\mathrm{Del}(S)$:

1. The interior of the circumcircle of any triangle in $\mathrm{Del}(S)$ contains no points of S.

2. If there exists a circle containing two points $p_1, p_2 \in S$ whose interior contains no points of S, then p_1p_2 is an edge of $\mathrm{Del}(S)$.

3. $\mathrm{Del}(S)$ is the planar dual graph of the Voronoi diagram graph of S (i.e., the dual of the planar graph obtained by the boundaries of the Voronoi regions and ignoring the outside region of this graph).

Property 3 gives us a solution to Problem 1a. Let $|Q_p|$ denote the number of sides of the polygon Q_p.

Theorem 4 *Given a finite point set $S \subset \mathbb{R}^2$ and a point p, a solution to Problem 1a is $|Q_p| = \deg(p)$, the degree (i.e., valency) of p in $\mathrm{Del}(S \cup \{p\})$.*

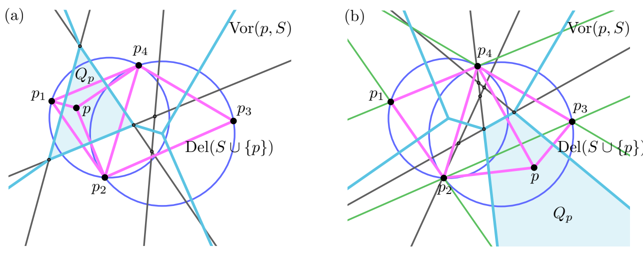


Figure 2: The region Q_p for a point set $S = \{p_1, \ldots, p_4\}$ where (a) $p \in \mathrm{conv}(S)$ and (b) $p \notin \mathrm{conv}(S)$. Black lines are the creases made when folding p to points in S. Circles are the circumcircles of the triangles in $\mathrm{Del}(S)$. Green lines are supporting hyperplanes of $\mathrm{conv}(S)$.

See Figure 2 for illustrations of this Theorem.

A more complete solution to Problem 1a would be to partition the plane into regions of constant $|Q_p|$. Since computing $\mathrm{Del}(S \cup \{p\})$ for many choices of p is cumbersome, we seek a more computationally direct way of solving Problem 1a. For example, an algorithm that relies only on $\mathrm{Del}(S)$ instead of $\mathrm{Del}(S \cup \{p\})$ would be preferable. We achieve this via a sequence of Lemmas which, while tailored to our specific problem, follow from basic facts about Delaunay triangulations and Voronoi diagrams (see [2, 4]).

For $x \in \mathbb{R}^2$, let us define $T_S(x)$ as the set of triangles in $\mathrm{Del}(S)$ whose circumcircles contain x in their interior.

Lemma 5 *If $p_i \in S$ is in the interior of $\mathrm{conv}(S)$, the edge pp_i is in $\mathrm{Del}(S \cup \{p\})$ if and only if $T_S(p)$ contains a triangle that has p_i as a corner.*

Proof. First assume that pp_i is in $\mathrm{Del}(S \cup \{p\})$. The edge must be part of at least one triangle p_ip_jp. Then, by property 1, there is an interior-empty circle containing p, p_i and p_j. Ignoring p for the moment, grow this circle maintaining p_ip_j as a chord and so that p remains inside this circle until the circle contains another point in S; this must happen because p_ip_j is not in the boundary of $\mathrm{conv}(S)$. The obtained circle then contains three points in S and its interior contains only p, by construction. The triangle defined by the points on this circle is in $T_S(p)$.

Now assume that a triangle $T \in T_S(p)$ has p_i as a corner. By property 1, there is a circle C containing p_i whose interior contain only p and no other point. We shrink C along its diameter that contains p_i, anchored at p_i until we obtain a circle C' containing p. By construction, the interior of C' is empty. By property 2, pp_i is an edge in $\mathrm{Del}(S \cup \{p\})$. □

Lemma 6 *Let $T_S(p) = \{T_1, \ldots, T_k\}$ and G be the graph whose vertex set is $T_S(p)$, and with an edge T_iT_j if T_i and T_j share a side. So long as $p \notin S$, then G is a tree.*

Proof. We first claim that a triangle T in $\mathrm{Del}(S \cup \{p\})$ that does not have p as a corner is also in $\mathrm{Del}(S)$. By property 1, the interior of the circumcircle of T is empty and deleting p does not change that. Hence, T is also in $\mathrm{Del}(S)$ as claimed.

If we delete p from $\mathrm{Del}(S \cup \{p\})$ (and all edges adjacent to p) we either get a single polygonal star-shaped hole or a pocket polygon (a cavity on the boundary of $\mathrm{conv}(S)$). Since $p \notin S$, in order to obtain $\mathrm{Del}(S)$ we can simply triangulate the hole or pocket by the above claim. By Lemma 5, the new triangles are exactly $T_S(p)$. The dual graph of a triangulation of a simple polygon is a tree. $\square$

Lemma 7 *If $p \in \mathrm{conv}(S)$ and $p \notin S$, then* $|Q_p| = |T_S(p)| + 2$.

Proof. Let $T_S(p) = \{T_1, \ldots, T_k\}$. One of these triangles, say T_a, contains p because $p \in \mathrm{conv}(S)$. By Theorem 4, $|Q_p| = \deg(p)$ in $\mathrm{Del}(S \cup \{p\})$, which equals the number of distinct vertices among the triangles $T_1, \ldots, T_k$, by Lemma 5. By Lemma 6, the edge-adjacency graph of $T_S(p)$ is a tree, which we can root at T_a. If we first count the three vertices in T_a, then we traverse the tree and for each additional triangle that we discover in the traversal we add only one vertex to our count. This gives us $k + 2$ distinct vertices among the triangles in $T_S(p)$, and so $\deg(p) = k+2$, as claimed. $\square$

Note that Lemma 7 allows p to be on the boundary of $\mathrm{conv}(S)$ (but not in S), in which case Q_p will be unbounded. In fact, for some areas nearby but outside $\mathrm{conv}(S)$, Lemma 7 will still apply. But to handle any choice of $p \in \mathbb{R}^2$, we need to consider certain supporting halfplanes of $\mathrm{conv}(S)$.

The boundary of $\mathrm{conv}(S)$ is a polygon; denote its vertices by $q_1, \ldots, q_m$. Let $H(q_i, q_{i+1})$ denote the supporting halfplane of $\mathrm{conv}(S)$ that contains the segment $q_i q_{i+1}$ on its boundary, and assume that the indices are cyclic so that this notation includes $H(q_m, q_1)$.

Let $\overline{H}(p)$ denote the number of halfplanes $H(q_i, q_{i+1})$ that *do not* contain the point p.

Lemma 8 *If $p \notin \mathrm{conv}(S)$ or $p \in S$ then* $|Q_p| = \overline{H}(p) + |T_S(p)| + 1$.

Proof. Suppose p is far enough away from $\mathrm{conv}(S)$ so that none of the circumcircles of the triangles in $\mathrm{Del}(S)$ contain it. If $H(q_i, q_{i+1})$ does not contain p, then p will be able to "see" both q_i and q_{i+1}. That is, a straight line segment from p to either q_i or q_{i+1} will not intersect $\mathrm{conv}(S)$. Thus, in $\mathrm{Del}(S \cup \{p\})$, p will be adjacent to q_i and q_{i+1}. For all supporting halfplanes that do not contain p, their corresponding vertices q_i will form a path on the boundary of $\mathrm{conv}(S)$, and we have that $\deg(p) = \overline{H}(p) + 1$.

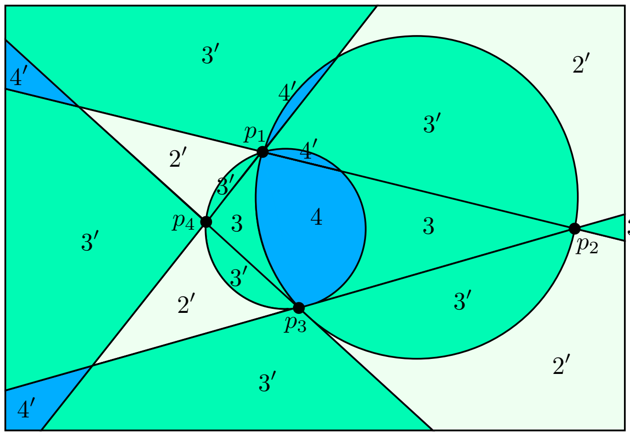


Figure 3: The plane partitioned into regions of constant $|Q_p|$ for a point set $S = \{p_1, \ldots, p_4\}$ in Problem 1a. Numbers marked with a prime symbol indicate regions where p gives an unbounded set Q_p.

If p is also in a circumcircle of a triangle T in $\mathrm{Del}(S)$ (an example of this is shown in Figure 2(b)), then by Lemma 5, p will be connected with new edges to every corner of T. Similar to the proof of Lemma 7, we can use Lemma 6 to show that there are $|T_S(p)| + 2$ such edges. However, two of these edges are double counted because two endpoints are on the boundary of $\mathrm{conv}(S)$. Thus the "+2" in Lemma 7 is not needed, and we arrive at $|Q_p| = \overline{H}(p) + |T_S(p)| + 1$.

If $p \in S$ then p is contained in all of the supporting halfplanes of $\mathrm{conv}(S)$. In fact, the situation is like that of Lemma 7 and its proof, except that Q_p will be unbounded and have one less side than the cases of Lemma 7 since we cannot fold p to itself. Thus $|Q_p| = |T_S(p)| + 1$ and we are done. $\square$

When $p \in \mathrm{conv}(S)$, we have that $\overline{H}(p) = 0$, so we can combine Lemmas 7 and 8 if we add an indicator function $I(p)$ which equals 1 if $p \in (\mathrm{conv}(S) \setminus S)$ and 0 otherwise.

Theorem 9 *Using the notation of Problem 1a, we have*

$$|Q_p| = \overline{H}(p) + |T_S(p)| + I(p) + 1.$$

Figure 3 illustrates how Theorem 9 can be used to partition the plane into regions of constant $|Q_p|$.

Remark on general position: Notice that no requirement has been made for the points in S to be in general position (no three points on a line). This is because no such requirement is necessary, but some care must be taken. If three points $p_1, p_2, p_3 \in S$ lie on a line, then those points cannot make a triangle in $\mathrm{Del}(S)$, and neither can they make a circumcircle. However, if these three collinear points lie on the boundary of $\mathrm{conv}(S)$,

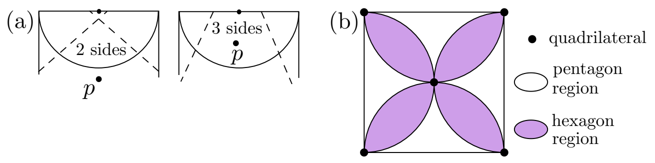


Figure 4: (a) Folding two corners to p with p outside, then inside, the side's midpoint circle. (b) The full solution to Problem 1 on a square.

say in order p_1, p_2, then p_3, then they will form two supporting halfplanes, one for the segment p_1p_2 and one for p_2p_3. This will affect the count of $\overline{H}(p)$; if p is not in the halfplane $H(p_1, p_2)$ then it will also not be in $H(p_2, p_3)$, and so the side of $\text{conv}(S)$ made by p_1, p_2, and p_3 will "count twice" in $\overline{H}(p)$ if p is on the other side of it.

Remark on computation time: The Voronoi diagram can be computed in $O(n \log n)$ time [5] and, thus, we can compute Q_p in the same asymptotic time. Note that Q_p may have $\Omega(n)$ sides and that the boundary of Q_p encodes the sorted cyclic order of the points adjacent to p in $\text{Del}(S \cup \{p\})$. Then, this is the best possible bound in the decision tree computation model. By Theorem 9, the partition of the plane into regions of constant $|Q_p|$ is given by an arrangement of circles and lines. Note that such arrangement can be of size $\Theta(n^2)$. Since each pair of these curves can only intersect at most twice, a sweep line algorithm can compute the arrangement in $O(n^2 \log n)$ time. Then, with $O(n^2 \log n)$ preprocessing time, given a query point p, Problem 1a can be solved in $O(\log n)$ time using a point-location data structure such as Kirkpatrick's [10].

3 Folding Corners of a Polygon to a Point

We return to Problem 1, folding the corners of a polygon P to a point p, so that the sides of Q_p may now be crease lines or edges of P.

As Haga discusses in his solution of the case where P is a square [6], the key is to see how close p needs to be to a boundary side of the paper in order to make that boundary add a side to Q_p. Imagine a semicircle C drawn on the paper whose center is the midpoint of a boundary side of P and with diameter equal to the length of that side. Then the endpoints of C equal two corners of P. If p is inside C, then when these two corners are folded to p their creases will not intersect at a point inside P, making the boundary edge between them add a side to Q_p. But if p lies on C or outside of C, then the two crease lines will intersect on P's boundary or inside it, respectively, implying that the boundary side in question will not contribute a side to Q_p. See Figure 4(a) for an illustration of this in the case where P is a square.

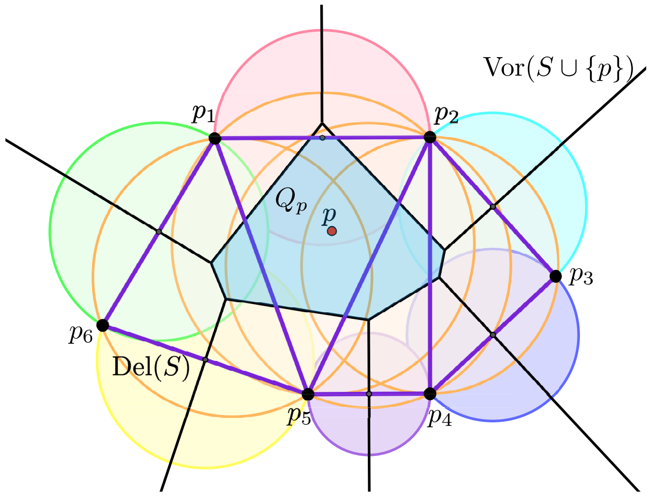


Figure 5: A generalized Problem 1 example on an irregular hexagon-shaped piece of paper $S = \{p_1, \ldots, p_6\}$. p is located in four circumcircles of $\text{Del}(S)$ and one midpoint circle, giving us $|Q_p| = 7$.

Thus, Haga's original solution is to draw four semicircles on our square, each centered at a different midpoint of the sides and with diameter equal to that side. We may then determine $|Q_p|$ by counting how many of the semicircle interiors contain p. If p is in the interior of only one semicircle, then only one boundary edge will contribute a side to Q_p, giving us $|Q_p| = 5$. If p is in two semicircle interiors, then two boundary sides will contribute, giving $|Q_p| = 6$. As can be seen in Figure 4(b), p can be in at most two of these semicircle interiors, so this solves the problem and gives us a partition of the square into regions of constant $|Q_p|$.

What Haga's original problem solution hides is the influence of Voronoi diagrams and Delaunay triangulations, because in the case where our paper is a square, the circumcircles of $\text{Del}(S)$ are equal and merely circumscribe the square. Thus, in our full Problem 1 statement we can combine our result from Problem 1a to obtain a full solution.

Let $M(p)$ denote the number of *midpoint circles* (circles centered at a midpoint of the polygon-shaped paper's edge and with diameter equal to that edge length) that contain p in their interior.

Theorem 10 *Given a convex polygon of paper P and a point $p \in P$, we have $|Q_p| = M(p) + |T_S(p)| + 2$.*

An illustration of this solution to Problem 1 is shown in Figure 5.

4 Folding Lines to a Line

To recap Problem 2, we take two points a, b on the boundary of a convex polygon P, let ℓ' be the line containing the segment $\ell = \overline{ab}$, and then fold and unfold

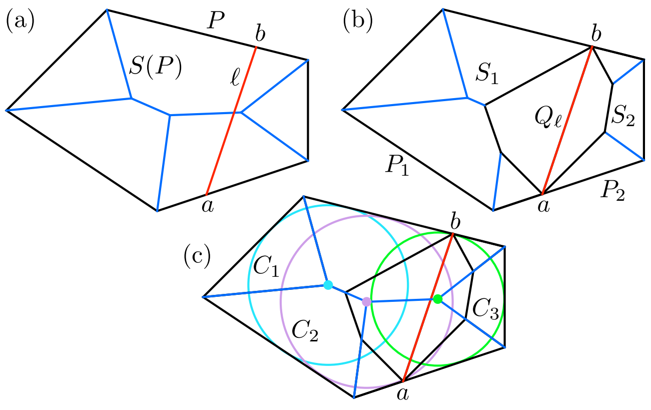


Figure 6: Analyzing Problem 2. (a) The polygon P, line ℓ, and the straight skeleton/medial axis $S(P)$. (b) The skeletons S_1, S_2 and the polygon Q_ℓ containing ℓ. (c) The event circles C_1, C_2, and C_3 of $S(P)$; here two of them intersect ℓ.

each side of P onto ℓ' in turn. We wish to describe the number of sides of the polygon Q_ℓ that contains ℓ and is limited by the crease lines we made. Let $V(P)$ and $E(P)$ denote the vertex and edge sets of P.

While Problem 1 was determined by the Voronoi diagram of $V(P)$, Problem 2 is governed by the straight skeleton (or medial axis, as these are equivalent on convex polygons) of $E(P)$. Towards that end, we will establish some notation, based on that of [1]. Let $S(P)$ denote the straight skeleton/medial axis of P, which we may think of the set of points x inside (or on the boundary) of P such that x is equidistant between at least two points of ∂P. Since P is convex, $S(P)$ will be a straight-line graph drawn on P that will include segments bisecting the angles of P (see Figure 6(a)).

The non-leaf vertices of $S(P)$ are called *event points* and the circles centered at these points that are tangent to their nearest sides of P are called the *event circles* of $S(P)$. Label these event circles $C_1, \ldots, C_k$ and let $\mathcal{C}^\ell$ be the subset of these circles that intersect ℓ (see Figure 6(c)).

Since ℓ splits P into two polygons, P_1 and P_2, we may find their straight skeletons as well; call them S_1 and S_2 respectively (see Figure 6(b)). One of each pair will be degenerate if a and b are on the same edge of P.

We define $b(e, \ell)$ to be the angle bisector between the line containing the edge $e \in E(P)$ and ℓ' (or, if these lines are parallel, let $b(e, \ell)$ be the line that is equidistant between these lines). Let $h(\ell, e)$ and $h(e, \ell)$ be the halfplane induced by $b(e, \ell)$ that contains ℓ and e, respectively.

Lemma 11 $V(Q_\ell) \setminus \{a, b\} \subset S(P)$.

Proof. We create the sides of Q_ℓ by folding a side $e \in E(P)$ to ℓ, making a crease that is a segment along the bisector $b(e, \ell)$. Thus the sides of Q_ℓ will lie along the edges of the skeletons S_1 and S_2, and the vertices of Q_ℓ (aside from a and b) will be event points of S_1 and S_2, which must lie on $S(P)$. ☐

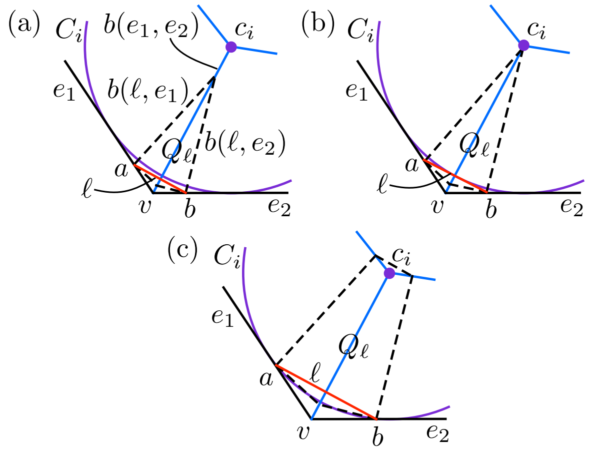


Figure 7: For Theorem 14's proof. (a) For ℓ close enough to a vertex $v \in V(P)$, Q_ℓ will be a quadrilateral. (b) If ℓ is tangent to the event circle C_i closest to v, Q_ℓ remains a quadrilateral, but critically. (c) When ℓ intersects the interior of C_i, a side is added to Q_ℓ.

Lemma 12 *Let $C_\ell(q)$ be the circle centered at q that is tangent to ℓ. Then $q \in \mathrm{int}(Q_\ell)$ if and only if $e \cap C_\ell(q) = \emptyset$ for all $e \in E(P)$.*

Proof. Let $q \in \mathrm{int}(Q_\ell)$ and consider any edge $e \in E(P)$. Then $q \in h(\ell, e)$ and, hence, $e \cap C_\ell(q) = \emptyset$.

Consider $q \notin \mathrm{int}(Q_\ell)$. Then there exists an edge $e \in E(P)$ with $q \in h(e, \ell)$, whereby $e \cap C_\ell(q) \neq \emptyset$. ☐

Lemma 13 *Let C be an event circle of $S(P)$ that does not intersect ℓ. Then the center of C will not be a vertex of Q_ℓ.*

Proof. Clearly the center of such a circle C cannot be a or b. Any other vertex of Q_ℓ lies on $S(P)$ by Lemma 11 as well as on either S_1 or S_2. Thus if C were centered at a point in $V(Q_\ell) \setminus \{a, b\}$ then C would be tangent to ℓ, which we forbid in this Lemma. ☐

As our solution to Problem 2, we claim that the number of sides of Q_ℓ will equal the number of event circles that intersect ℓ plus four, unless one or more of $\{a, b\}$ are also vertices of P, in which case those points must be subtracted.

Theorem 14 $|V(Q_\ell)| = |\mathcal{C}^\ell| + 4 - |\{x \in \{a, b\} : x \in V(P)\}|$.

Proof. We let ℓ sweep over P, starting at a vertex $v \in V(P)$ that is adjacent to sides $e_1, e_2 \in E(P)$. The bisector $b(e_1, e_2)$ at v will lie along the edge $\overline{vc_i}$ of the

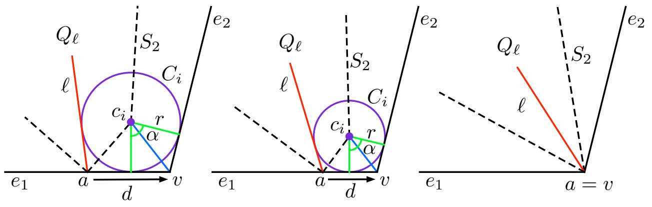


Figure 8: Showing, for the proof of Theorem 14, how if an endpoint a of ℓ equals a vertex v of P, then Q_ℓ will lose a side.

straight skeleton $S(P)$ at v, where c_i is the center of an event circle C_i that is tangent to the sides e_1 and e_2. Now, if ℓ is drawn close enough to v so that ℓ lies outside or tangent to C_i, then Q_ℓ will be a quadrilateral. This is because if ℓ is outside of C_i then the bisectors $b(\ell, e_1)$ and $b(\ell, e_2)$ will intersect on $S(P)$ between v and c_i, and if ℓ is tangent to C_i this intersection will occur at c_i, as we have $c_i \notin Q_\ell$ by Lemma 12. See Figures 7(a) and (b).

If, however, we sweep ℓ so that it intersects the interior of C_i, then the bisectors $b(\ell, e_1)$ and $b(\ell, e_2)$ will intersect along $b(e_1, e_2)$ beyond the point c_i, meaning that they will cross other segments of $S(P)$ before such an intersection. By Lemma 11, in order for the vertices of Q_ℓ to remain on $S(P)$ we must have that a side was added to Q_ℓ, as demonstrated in Figure 7(c).

We then sweep ℓ, continuously moving a and b clockwise and counterclockwise, respectively, from v along ∂P. When ℓ intersects the interior of an event circle C_i in $S(P)$, thus adding a circle to $\mathcal{C}^\ell$, an additional side will be added to Q_ℓ. All event circles that do not intersect ℓ will not contribute sides to Q_ℓ by Lemma 13.

If $a \in V(P)$ one event circle of either S_1 or S_2 will disappear at this vertex: Let a be close to a vertex $v \in V(P)$, say with $d(a, v) = d$, see Figure 8. We consider the circle C_i tangent to ℓ and the two edges incident to v, e_1 and e_2. When we move a towards v, the angle α between the two radii from the center of C_i to e_1 and e_2 is constant (since c_i will move closer to v along the bisector $b(e_1, e_2)$ segment of $S(P)$ as a approaches v). Hence, the ratio between d and the radius r of C_i is constant. Consequently, d approaches zero when r goes to zero, and C_i disappears at the limit. The same holds true for $b \in V(P)$. □

Remark on computation time: The medial axis of a simple polygon P with n vertices can be computed in $O(n)$ time [3] and, thus, we can compute Q_ℓ in the same asymptotic time by computing the medial axis of the two convex polygons obtained by splitting P with ℓ. Note that the complexity of Q_ℓ is $\Omega(n)$ in the worst case and thus this time bound is optimal.

5 Bounds and Visualization for Problem 2

When we follow the computation from Theorem 14 to determine the number of sides/edges of our polygon Q_ℓ, we need to compute the event circles of the straight skeleton intersecting the line segment ℓ. In this section, we do not aim for an exact computation of $|V(Q_\ell)|$, instead giving simple bounds.

The line segment ℓ is determined by two points, a, b, on P's boundary. The maximum number of vertices of Q_ℓ depends on the location of a and b—we distinguish whether both, one of, or none of a and b are vertices of P:

Lemma 15 *With $n = |V(P)|$, we have:*

1. *If $a, b \in \partial P \setminus V(P)$, then $|V(Q_\ell)| \leq n + 2$.*
2. *If $a \in V(P), b \in \partial P \setminus V$, then $|V(Q_\ell)| \leq n + 1$.*
3. *If $a, b \in V(P)$, then $|V(Q_\ell)| \leq n$.*

Proof. For the number of event circles $C_1, \ldots, C_k$ in P (or vertices of $S(P)$), we have $|\{C_1, \ldots, C_k\}| = n-2$ (see Aichholzer and Aurenhammer [1]). In $\mathcal{C}^\ell$, we consider only the subset of these event circles that intersect ℓ, hence, $|\mathcal{C}^\ell| \leq |\{C_1, \ldots, C_k\}| = n-2$. With $|\{x \in \{a, b\} : x \in V(P)\}| = 0$ for $a, b \in \partial P \setminus V(P)$, $|\{x \in \{a, b\} : x \in V(P)\}| = 1$ for $a \in V(P), b \in \partial P \setminus V(P)$, and $|\{x \in \{a, b\} : x \in V(P)\}| = 2$ for $a, b \in V(P)$, the claim follows directly from Theorem 14. □

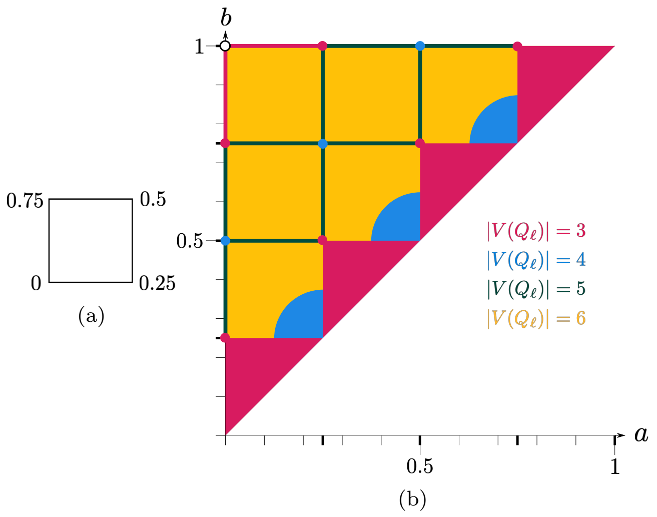


Figure 9: Example for the visualization of the exact value of $|V(Q_\ell)|$ for a square: (a) parameterization and (b) configuration space.

Visualization. To visualize upper bounds, lower bounds, or the exact value of $|V(Q_\ell)|$, we parameterize ∂P starting from and ending at an arbitrary vertex

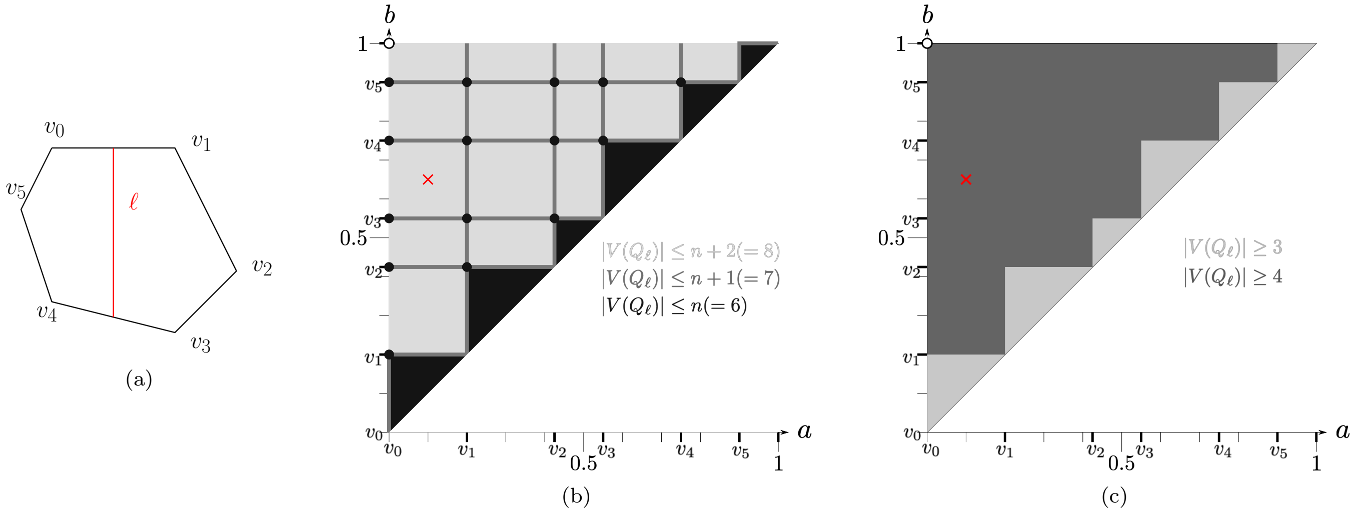


Figure 10: An example of Problem 2 visualization. (a) A 6-gon with a possible line segment ℓ (in red) with $a, b \in \partial P \setminus V$. (b) Upper bounds from Lemma 15, (c) lower bounds. The red cross in (b) and (c) represents the line segment ℓ from (a).

$v \in V(P) = \{v_0, \ldots, v_{n-1}\}$ (w.l.o.g., we choose $v = v_0$). We consider a unit square, with the location of a on the x-axis and the location of b on the y-axis. We color all points of the triangle above the line segment $\overline{(0,0),(1,1)}$ according to the valid upper bound, lower bound, or the value of $|V(Q_\ell)|$. For an example with exact values of $|V(Q_\ell)|$ when P is a square see Figure 9, for an example of upper and lower bounds when P is a 6-gon see Figure 10.

6 Conclusion

The problems presented here are fairly abstract and seem divorced from computational origami problems that have been previously studied. Connections may exist, however. For example, the straight skeleton is useful in algorithms for origami design [12], and so the line ℓ in Problem 2 could be interpreted as separating our polygon P of paper into two regions, each of which could then be folded into different origami bases via their straight skeletons. Q_ℓ would then represent the polygon of paper that, when folded along ℓ, links the two bases together. Knowing $|Q_\ell|$ in advance might help the origami designer plan how to sink the flaps of paper adjoining Q_ℓ, a step that is often useful when turning an algorithmicly-designed origami base into a representational model.

At first glance, one might think that Problems 1 and 2 would be duals to each other in the projective geometry sense, since the first concerns folding points to points and the second lines to lines. However, this is incorrect, mainly because the operation of folding a point p_1 to another point p_2, which makes a crease line that is the perpendicular bisector of $\overline{p_1p_2}$, is not dual to the operation of folding a line l_1 to l_2, which forms the bisector of l_1 and l_2. This is why our solutions to these two problems, while similar in flavor, are not reducible from one another.

We hope that this work will inspire others to consider similar folding problems in computational origami. Indeed, the list of basic origami operations (see [9, Chapter 1]) would be a good place to start to investigate what other simple folding problems are possible. Also, the work of Kazuo Haga, which is characterized by playful investigations of simple folds (what he calls *origamics*) led directly to our investigations in the present paper. Haga's work, in particular [7], certainly contains avenues for further study.

Acknowledgments

This work was conducted at the 2018 and 2019 Bellairs Workshops on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint. We thank the other participants of the workshop for helpful discussions, especially Klara Mundilova and Tomohiro Tachi. H. A. A. was partially supported by NSF grants CCF-1422311, CCF-1423615. T. C. H. was partially supported by NSF grant DMS-1906202 and C. S. was partially supported by Jubileumsanslaget från Knut och Alice Wallenbergs Stiftelse and Vinnova grant 2018-04101.

References

[1] O. Aichholzer and F. Aurenhammer, *Straight Skeletons for General Polygonal Figures in the Plane*, Proc. 2nd Ann. Int'l. Computing and Combinatorics Conf. COCOON'96, Lecture Notes in Computer Science, volume 1090, pages 117-126, Hong Kong, 1996. Springer Verlag.

[2] F. Aurenhammer, R. Klein, and D. Lee, *Voronoi Diagrams and Delaunay Triangulations*, World Scientific, Singapore, 2013.

[3] F. Chin, J. Snoeyink, and C. A. Wang, Finding the medial axis of a simple polygon in linear time, *Discrete & Computational Geometry*, 1999, 21(3), 405–20.

[4] S. Devadoss and J. O'Rourke, *Discrete and Computational Geometry*, Princeton University Press, Princeton, NJ, 2011.

[5] S. Fortune, A sweepline algorithm for Voronoi diagrams, *Algorithmica*, 1987, 2(1), 153–174.

[6] K. Haga, Proposal of a term origamics for plastic origami–workless scientific origami, in *Second International Meeting of Origami Science and Scientific Origami Abstracts*, Seian University of Art and Design, Otsu, Japan, 1994, 29–30.

[7] K. Haga, *Origamics: Mathematical Explorations Through Paper Folding*, World Scientific Publishing Co., River Edge, NJ, 2008.

[8] T. Hull, *Project Origami: Activities for Exploring Mathematics, 2nd ed.*, CRC Press/A K Peters, Boca Raton, FL, 2012.

[9] T. Hull, *Origametry: Mathematical Methods in Paper Folding*, Cambridge University Press, Cambridge, UK, 2020.

[10] D. Kirkpatrick, Optimal search in planar subdivisions, *SIAM Journal on Computing*, 1983, 12(1), 28–35.

[11] R. Kraft, Orthoginal Voronoi molecules, in Lang et al. ed., *Origami^7: The Proceedings from the 7th International Meeting on Origami in Science, Mathematics, and Education*, Tarquin, St. Albans, UK, 2018, 607–621.

[12] R. J. Lang, A computational algorithm for origami design, in *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, ACM, 1996, 98–105.

Cut Locus Realizations on Convex Polyhedra

Joseph O'Rourke[*] Costin Vîlcu[†]

Abstract

We prove that every positively-weighted tree T can be realized as the cut locus $C(x)$ of a point x on a convex polyhedron P, with T weights matching $C(x)$ lengths. If T has n leaves, P has (in general) $n+1$ vertices. We show there are in fact a continuum of polyhedra P each realizing T for some $x \in P$. Three main tools in the proof are properties of the star unfolding of P, Alexandrov's gluing theorem, and a cut-locus partition lemma. The construction of P from T is surprisingly simple.

1 Introduction

There is a long tradition of reversing, in some sense, the construction of a graph G from a geometric set. The geometric set may be a point set, a polygon, or a polyhedron, and the graph G could be the Voronoi Diagram, the straight skeleton, or the cut locus, respectively. Reversing would start with, say, the straight skeleton, and reconstruct a polygon with that skeleton. One might even view Steinitz's classic theorem (e.g., [17]) as starting with a planar 3-connected graph and "reconstructing" it as the 1-skeleton of a convex polyhedron. Here we start with a positively-weighted tree T and construct polyhedra P on which T is realized as the cut locus for a point $x \in P$. (The cut locus is defined in Section 2.1 below.)

The literature has primarily examined three models for the graph G, often specialized (as here) to trees T:

(1) *Unweighted tree*: The combinatorial structure of T, without further information.

(2) *Length tree*: T with positive edge weights representing Euclidean lengths, and with given circular order of the edges incident to each node of T. Called "ribbon trees" in [6], and "ordered trees" in [5].

(3) *Geometric tree*: Given by a planar drawing, i.e., coordinates of nodes, determining lengths and angles.

Our main result is this:

Theorem 1 *Given a length tree T of n leaves, we can construct a continuum of star-unfoldings of convex polyhedra P of $n+1$ vertices, each of which, when folded, realizes T as the cut locus $C(x)$ for a point $x \in P$. Each star-unfolding can be constructed in $O(n)$ time.*

Thus, every length tree is isometric to a cut locus on a convex polyhedron.

1.1 Related Results

The computer science literature is extensive, and we cite just a few results:

- Every unweighted tree can be realized as the Voronoi diagram of a set of points in convex position [12].
- Every length tree can be realized as the furthest-point Voronoi diagram of a set of points [5].
- Every unweighted tree can be realized as the straight skeleton of a convex polygon, and conditions for length-tree realization are known [6] [2] [5].

In all cases, the reconstruction algorithms are efficient: either $O(n)$ or $O(n \log n)$ for trees of n nodes. Although all these results can be viewed as variations on realizing Voronoi diagrams, and a cut locus is a subgraph of a Voronoi diagram, it appears that prior work does not imply our results.

Our inspiration derives from two results in the geometry literature:

- Every length graph can be realized as a cut locus on a Riemannian surface [10]. The result is nonconstructive.
- Every unweighted tree can be realized as a cut locus on a doubly covered convex polygon, and length trees can be realized on such polygons when several technical conditions are satisfied [9].

2 Background Tools

In this section we describe the tools needed to prove our main theorem, drawing heavily on our [13].

[*]Smith College, jorourke@smith.edu.

[†]*Simion Stoilow* Institute of Mathematics of the Romanian Academy, Costin.Vilcu@imar.ro.

2.1 Cut Locus

The *cut locus* $C(x)$ of a point x on (the surface of) a convex polyhedron P is the closure of the set of points to which there are more than one shortest path from x. This concept goes back to Poincaré [14], and has been studied algorithmically since [15] (under the name "ridge tree"). Some basic properties and terminology:

- $C(x)$ is a tree whose leaves are vertices of P, and all vertices of P are in $C(x)$.
- Points interior to $C(x)$ of tree-degree 3 or more we will call *ramification points*.
- The edges of $C(x)$ are *geodesic segments* on P, geodesic shortest paths between their endpoints [1].

2.2 Star Unfolding

The *star unfolding* $S_P(x)$ of P with respect to x is formed by cutting a shortest path from x to every vertex of P [4] [1]. This unfolds P to a simple non-overlapping planar polygon $S = S_P(x)$ of $2n$ vertices: n images x_i of x, and n images of the vertices v_i of P. The connection between the cut locus and the star unfolding is that the image of $C(x)$ in S is the restriction to S of the Voronoi diagram of the images of x [4]. See Fig. 1.

2.3 Alexandrov's Gluing Theorem

We rely on Alexandrov's celebrated "Gluing" Theorem [3, p.100], which we will abbreviate as Theorem AGT.

Theorem AGT *If the boundaries of planar polygons are glued together (by identifying portions of the same length) such that*

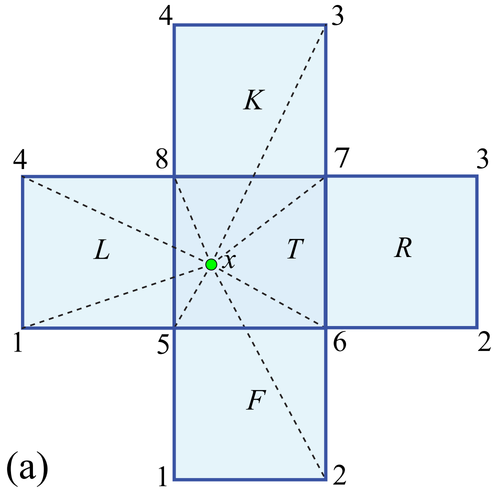
(1) The perimeters of all polygons are matched (no gaps, no overlaps).

(2) The resulting surface is a topological sphere.

(3) At most 2π surface angle is glued at any point.

Then the result is isometric to a convex polyhedron P, possibly degenerated to a doubly-covered convex polygon. Moreover, P is unique up to rigid motion and reflection.

A doubly-covered polygon is a flat, two-sided, zero-volume polyhedron, sometimes called a *dihedron* or *bihedron*. The proof of Alexandrov's theorem is nonconstructive, and there remains no effective procedure for constructing the polyhedron guaranteed to exist by this theorem. See [11].



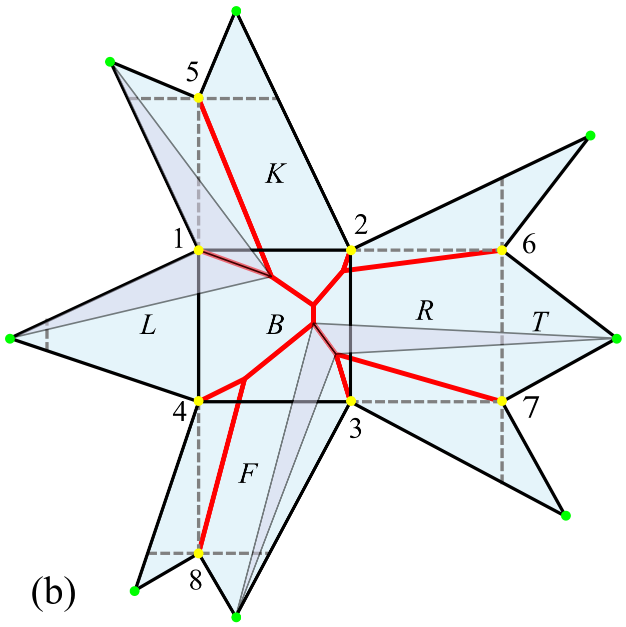


Figure 1: (a) Cut segments to the 8 vertices of a cube from a point x on the top face. (No cut is interior to the bottom face.) T, F, R, K, L, B = Top, Front, Right, Back, Left, Bottom. (b) The star-unfolding from x. The cut locus $C(x)$ (red) is the Voronoi diagram of the 8 images of x (green). Two pairs of fundamental triangles are shaded.

2.4 Fundamental Triangles

The following lemma is one key to our proof. See Fig. 1(b).

Lemma 2 (Fundamental Triangles [7]) *For any point $x \in P$, P can be partitioned into flat triangles whose bases are edges of $C(x)$, and whose lateral edges are geodesic segments from x to the ramification points or leaves of $C(x)$. Moreover, those triangles are isometric to plane triangles, congruent by pairs.*

The overall form of our proof of Theorem 1 is to create a star unfolding S by pasting together x_i-apexed fundamental triangles straddling each edge of T, and then applying Alexandrov's theorem to conclude that the folding of S yields a convex polyhedron P that realizes $C(x)$.

2.5 Cut Locus Partition

The last tool we need is a generalization of lemmas in [7]. On a polyhedron P, connect a point x to a point $y \in C(x)$ by two geodesic segments γ, γ'. This partitions P into two "half-surface" digons H_1 and H_2, each bound by $\gamma \cup \gamma'$. If we now zip each digon separately closed by joining γ and γ', AGT leads to two convex polyhedra P_1 and P_2. The lemma says that the cut locus on P is the "join" of the cut loci on P_i. See Fig. 2.

Lemma 3 (Cut Locus Partition) *Under the above circumstances, the cut locus $C(x,P)$ of x on P is the* join *of the cut loci on P_i: $C(x,P) = C(x,P_1) \sqcup_y C(x,P_2)$, where $\sqcup_y$ joins the two cut loci at y. And starting instead from P_1 and P_2, the natural converse holds as well.*

Proof. Induction and Lemma 2 shows that the cut locus of x on P_i is indeed the truncation of $C(x,P)$. Therefore, $C(x,P) = C(x,P_1) \sqcup_y C(x,P_2)$.

Assume we start now from P_1 and P_2, having vertices $x_1, y_1 \in P_1$ and $x_2, y_2 \in P_2$ such that

- $\rho_{P_1}(x_1,y_1) = \rho_{P_2}(x_2,y_2)$, where $\rho_{P_i}()$ is the geodesic distance between the indicated points on P_i.
- $\theta_{x_1} + \theta_{x_2} \le 2\pi$, where θ_x is the total surface angle incident to x, and
- $\theta_{y_1} + \theta_{y_2} \le 2\pi$.

Then we can cut open P_i along a geodesic segment γ_i from x_i to y_i and join the the two halves by AGT, such that x_1, x_2 have a common image x, and y_1, y_2 have a common image y.

Now all geodesic segments starting at x into H_i remain in H_i, because geodesic segments do not branch. Therefore, H_1 has no influence on $C(x,P_2)$ and H_2 has no influence on $C(x,P_1)$. □

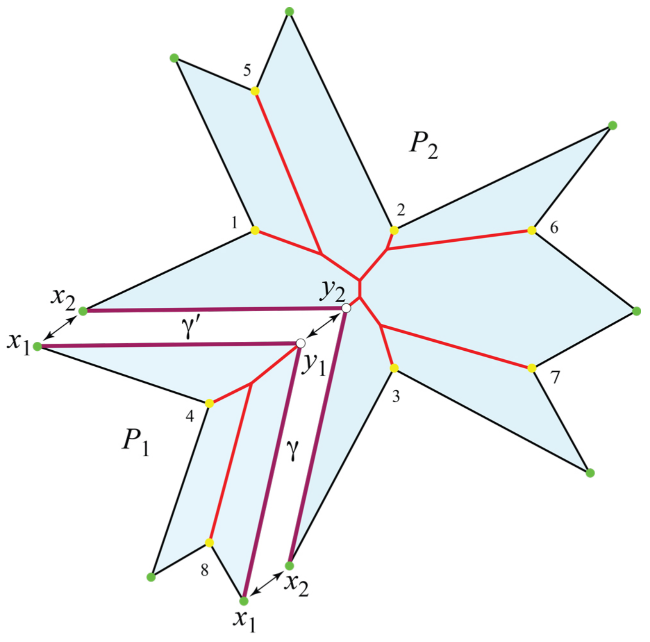


Figure 2: Geodesic segments γ and γ' (purple) connect $x{=}x_1{=}x_2$ to $y{=}y_1{=}y_2$. P_1 folds to a tetrahedron, and P_2 to an 8-vertex polyhedron, with x and y vertices in each. P_1 and P_2 are cut open along geodesic segments from x_i to y_i and glued together to form P. Based on the cube unfolding in Fig. 1(b).

3 Star-Tree

We start with T a star-tree: one central node u of degree-m with edges to nodes $u_1, u_2, \ldots, u_m$.

A *cone* in the plane is the unbounded region between two rays from its apex. Set $\lambda > L$ to be longer than L, the length of the longest edge of T.

We realize T within a cone of apex angle α, with $0 < \alpha \le 2\pi$.[1] See Fig. 3. Identify points x_1 and x_{m+1} on the cone boundary, with $|ux_i| = \lambda$. Inside the cone, place x images $x_2, \ldots, x_m$, with each $|ux_i| = \lambda$. Finally place u_i so that uu_i bisects $\angle(x_i, u, x_{i+1})$, $i = 1, \ldots, m$. Chose u_i so that $|uu_i|$ matches T's given edge weights. Finally, connect

$$(u, x_1, u_2, x_2, \ldots, x_m, u_m, x_{m+1})$$

into a simple polygon.[2]

Before we proceed with the proof, we emphasize that there are several free choices in this construction, illustrated in Fig. 4:

- The angle α at the root is arbitrary.
- The angular distribution of the ux_i segments is arbitrary.

[1]Note that we allow $\alpha > \pi$; $\alpha = 2\pi$ represents the whole plane.
[2]If $\alpha = 2\pi$, $x_1 = x_{m+1}$ and u is interior to the polygon.

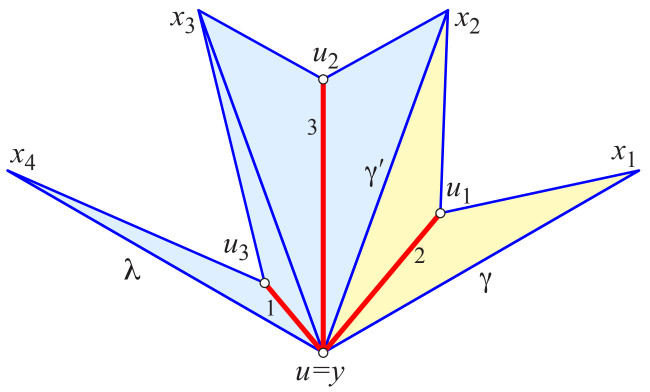


Figure 3: $\alpha = 120°$, $m = 3$, T edge lengths $(2, 3, 1)$ (red), $\lambda = 4$. In the induction proof of Lemma 4, $\bar{P}(T_1)$ (yellow) is joined to $\bar{P}(T_2)$ (blue).

- $\lambda > L$ needs to be "sufficiently large" in a sense we will quantify, but otherwise is arbitrary.

In general we will distribute x_i equi-angularly. Choosing $\alpha = 2\pi$ results in a polyhedron of $n + 1$ vertices; for $\alpha < 2\pi$, u is an additional vertex.

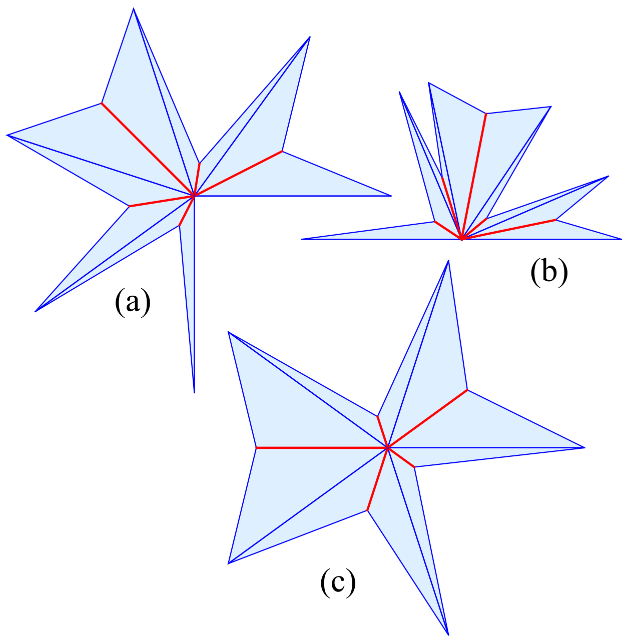


Figure 4: Star-T with edge lengths $(3, 1, 4, 2, 1)$. (a) $\alpha = 270°$, equiangular x_i, $\lambda = 6$. (b) $\alpha = 180°$, random x_i, $\lambda = 5$. (c) $\alpha = 360°$, equiangular x_i, $\lambda = 6$.

We call the described star-T construction a *triangle packing*.

Lemma 4 *A triangle packing of star-T, for sufficiently large λ, is the star-unfolding of a polyhedron P with respect to a point x such that $T = C(x)$.*

Proof. The proof is by induction on the number m of edges of T, which is the degree of the root node u. If T is a single edge $e = uu_1$, then folding the twin triangles by creasing e and joining x_1 and x_2, the two images of x, leads to a doubly covered triangle with x at the corner opposite e. (See the pair of yellow triangles in Fig. 3.) Clearly $e = C(x)$ is the restriction of the Voronoi diagram of x_1 and x_2, the bisector of x_1 and x_2. Concerning λ, we need that the angle θ_x incident to x is at most 2π (because x is a point on a convex surface). In this base case, $\theta_1 + \theta_2 \leq 2\pi$ is immediate, so λ is sufficiently large.

Now let $m > 1$, and partition T into $T_1 \cup_u T_2$ with root degrees m_1 and m_2 respectively, where $\cup_u$ indicates joining the trees at the root u. We will use $\bar{P}(T)$ for the planar triangle packing for T, and $P(T)$ for the folded polyhedron.

We first address λ. In order to apply AGT, we need that θ_x, the sum of the angles at the tips of the triangles—the images of x—is at most 2π.

Let λ be the larger of λ_1 and λ_2 for $\bar{P}(T_1)$ and $\bar{P}(T_2)$ respectively, and stretch the smaller λ_i so that they both share the same λ. Form $\bar{P}(T)$ by joining the two now-compatible triangle packings. Fixing α and the sector angles, it is clear that the angle at each triangle tip decreases monotonically as λ increases. So increase λ as needed so that $\theta_x \leq 2\pi$. Somewhat abusing notation, call these possibly enlarged packings $\bar{P}(T_1)$, $\bar{P}(T_2)$ and $\bar{P}(T)$.

Now we aim to show that $\bar{P}(T)$ is the star-unfolding of a polyhedron with $T = C(x)$. Certainly $\bar{P}(T)$ folds to a polyhedron P, because (a) by construction the edges incident to $u_1, \ldots, u_k$ match in length, and (b) we have ensured that $\theta_x \leq 2\pi$. So Alexandrov's AGT theorem applies. Now identify γ and γ' on P from x to $y = u$ separating the surface into pieces corresponding to $\bar{P}(T_1)$ and $\bar{P}(T_2)$, which fold to P_1 and P_2 respectively. (Refer again to Fig. 3.) By the induction hypothesis, $T_1 = C(x, P_1)$ and $T_2 = C(x, P_2)$. Applying Lemma 3, we have $C(x, P) = C(x, P_1) \sqcup_y C(x, P_2)$, where the two cut loci are joined at $y = u$. And so indeed $T_1 \cup_u T_2 = T = C(x)$. □

In Section 4.1 we will calculate the needed λ explicitly.

4 General Length-Trees T

We now generalize the above to arbitrary length-trees T, using the example in Fig. 5 as illustration.

Given T, select any node u to serve as the root. Fix any α, and choose λ to exceed the length L of the longest path from u to a leaf in T. Now create a triangle packing for T as follows.

First create a triangle packing for u and its immediate children $u_1, \ldots, u_m$, just as previously described. With $\lambda > L$, the external angle at u_i is strictly less than

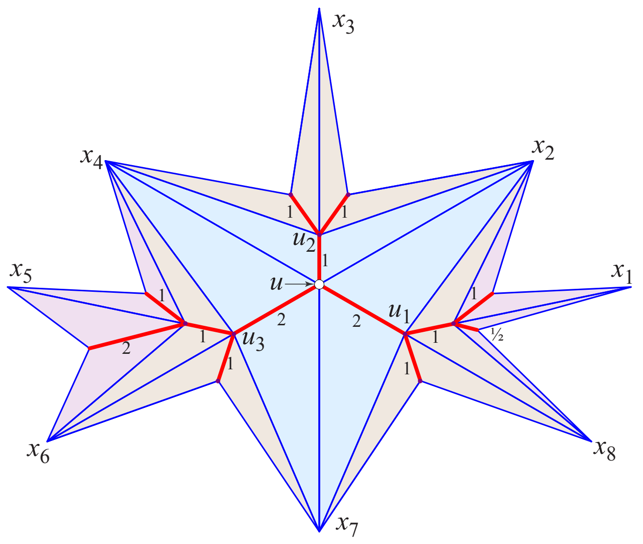


Figure 5: Realization of a length-tree T of height 3, using $\alpha = 2\pi$ and $\lambda = L = 5$. This polygon folds to a polyhedron of 9 vertices: the 8 leaves of T, and x.

π, i.e., it forms a "V-shape" there. Call this a *cup*, $c_i = (x_i, u_i, x_{i+1})$ with α_i external angle at u_i. Let u_i have children $u_{i1}, u_{i2}, \ldots, u_{il}$. So $u_i, (u_{i1}, \ldots, u_{il})$ is a star-tree. Fill in the cup c_i by inserting a triangle packing for this sub-star-tree, with apex at u_i, angle α_i, and λ-length $|u_i x_i|$, the distance from u_i to the tips of the cup.

After filling the c_i cups for all the u_i at level-2 of T, repeat the process with level-3 of T, and so on. Throughout the construction, the locations for x_i remain fixed after their initial placement. And with sufficiently large λ, all the cups form V-shapes.

Note that the triangles incident to an internal node u_i of T (neither the root nor a leaf) leave no gaps: they cover the 2π surrounding u_i.

Lemma 5 *A triangle packing for any T, as just described, for sufficiently large λ, is the star-unfolding of a polyhedron P with respect to a point x such that $T = C(x)$.*

Proof. The proof is by induction, and parallels the proof of Lemma 4 closely. Consequently, we only sketch the proof.

The base of the induction is a star-graph, settled by Lemma 4. Let T be an arbitrary length-tree, and partition T into two smaller trees T_1 and T_2 sharing the root u, so $T = T_1 \cup_u T_2$. Select an α for T and α_i for T_i, $i = 1, 2$, so that $\alpha = \alpha_1 + \alpha_2$.

By the induction hypothesis, T_i can be realized in cups of angle α_i. Moreover, $\bar{P}(T_i)$ folds to P_i and $T_i = C(x, P_i)$. Stretch λ_i as needed to allow $\bar{P}(T_1)$ to share λ with $\bar{P}(T_2)$ at u, and stretch again so that $\theta_x \leq 2\pi$.

Form $\bar{P}(T)$ by adjoining $\bar{P}(T_1)$ and $\bar{P}(T_2)$ at u, with cup apex α. Fold $\bar{P}(T)$ to P by AGT. Apply Lemma 3 to conclude $C(x, P) = C(x, P_1) \sqcup_y C(x, P_2)$, where $y = u$. And so $T_1 \cup_u T_2 = T = C(x)$. $\square$

Note that all ramification points of $C(x)$ are flat on P, with 2π incident surface angle. If θ_x is strictly less than 2π, then the source x is a vertex on P. If at the root, $\alpha < 2\pi$, then in addition u is a vertex on P. So P has n, $n+1$, or $n+2$ vertices.

Lemmas 4 and 5, together with Lemma 6 (below) prove Theorem 1. The construction of the triangle packing can be achieved in $O(n)$ time: we are given the cyclic ordering of the edges incident to each node, so sorting is not necessary, and the level-by-level packing construction is proportional to the the number of edges.

4.1 Total angle at x

We derive here a sufficient condition on the value of the parameter λ for the root cup to guarantee that $\theta_x \leq 2\pi$.

Lemma 6 *If T has m edges, and L is the longest path from the root in T, then $\theta_x \leq 2\pi$ when*

$$\lambda \geq L\left[1 + \cot\left(\frac{\pi}{m}\right)\right] .$$

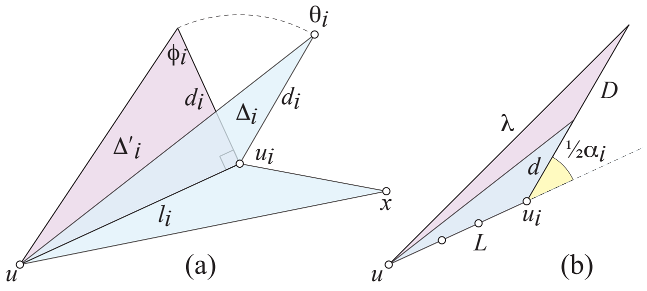


Figure 6: (a) $\theta_i \leq \phi_i$. (b) Increasing λ increases d. L is the length of the longest path in T, D the target bound for $\lambda \geq L + D$.

Proof. First we establish notation, illustrated in Fig. 6(a). As before, u is the root of T, and let u_i be a leaf of T, with edge uu_i shared by twin triangles, one of which is $\triangle_i$. Let $\ell_i = |uu_i|$ and d_i the distance from the tip of $\triangle_i$ to u_i, and θ_i the angle at that tip, an image of x. Because the fundamental triangles come in pairs, and there are m edges, we have that the total angle at x satisfies $\theta_x = 2\sum \theta_i$.

Consider now a right triangle $\triangle'_i$ having the same base as $\triangle_i$ and height d_i, and denote by ϕ_i its angle opposite to the base uu_i. Then $\phi_i = \arctan\left(\frac{\ell_i}{d_i}\right)$ and $\theta_i \leq \phi_i$; see again Fig. 6.

Because arctan is an increasing function, we can obtain an upper bound by replacing ℓ_i with the longest edge length ℓ, and replacing d_i by the shortest of the xu_j diagonals, call it d: $\ell = \max_i \ell_i$, $d = \min_i d_i$. So $\phi_i \leq \arctan\left(\frac{\ell}{d}\right)$, and therefore

$$\theta_x \ = \ 2\sum \theta_i \ \leq \ 2\sum \phi_i \ \leq \ 2m \arctan\left(\frac{\ell}{d}\right) \ .$$

The expression $2m \arctan\left(\frac{\ell}{d}\right)$ decreases as d increases. For it to evaluate to at most 2π, calculation shows the following must hold:

$$d \ \geq \ \ell \cot\left(\frac{\pi}{m}\right) \ .$$

The longest path L from root to leaf is at least as long as the longest edge, $L \geq \ell$, so this bound will more than suffice:

$$d \ \geq \ L \cot\left(\frac{\pi}{m}\right) = D \ .$$

Now we show that if λ is long enough, then $d \geq D$. Consider Fig. 6(b), where L is the longest path from u to a leaf u_i. Let the external angle at u_i be α_i. By definition, there is some index j such that $d = d_j$. The triangle inequality directly implies $\lambda \leq \ell_j + d_j \leq L + d$.

With L and α_i fixed, increasing λ increases d. If we substitute the needed lower bound D for d in the expression (see Fig. 6(b)), then $\lambda \geq L + D$ forces $d \geq D$. Explicitly,

$$\lambda \geq L\left[1 + \cot\left(\frac{\pi}{m}\right)\right]$$

suffices to guarantee that $\theta_x \leq 2\pi$. □

The bound—approximately $L(1 + m/\pi)$—is far from tight. For example, in Fig. 5, $m = 13$ and $L = 5$ leads to $\lambda \geq 26$, but $\lambda = 5$ (illustrated) leads to $\theta_x \approx 167°$, and so easily suffices.

5 Remarks

One further example is shown in Fig. 7. It is the star unfolding of a polyhedron of 49 vertices, whose resemblance to a fractal suggests there might at a deeper connection. We will only mention that fractals play a role in the folding to specific convex polyhedra in [16], and fractal cut loci on k-differentiable Riemannian and Finslerian spheres are shown in [8] to exist for any $k \geq 2$.

A natural question is whether geometric trees—drawings embedded in the plane, and so providing angles between adjacent edges—can be realized as cut loci on convex polyhedra. Certainly not all geometric trees are realizable, for there are constraints on the angles: around a ramification point, no angle can exceed π, and

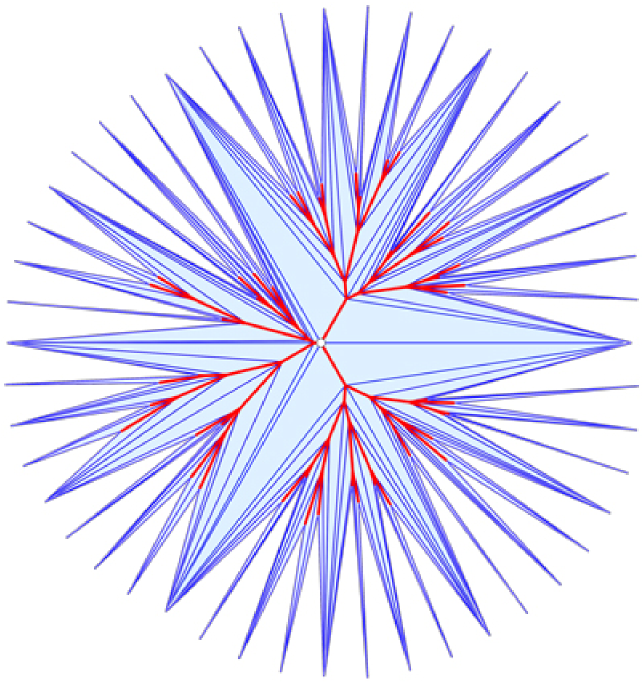


Figure 7: Regular degree-3 tree, random edge lengths, $\alpha = 2\pi$, $n = 48$, $\theta_x \approx 317°$.

the angles must sum to $\leq 2\pi$. And the sum of the curvatures at the u_i and at x must be 4π to satisfy the Gauss-Bonnet theorem.

We leave it as an open problem to determine whether or not a given geometric tree is realizable as a cut locus on a convex polyhedron.

Acknowledgements. We benefitted from the suggestions of four anonymous reviewers. The second author's research was partially supported by UEFISCDI, project no. PN-III-P4-ID-PCE-2020-0794.

References

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM J. Comput.*, 26:1689–1713, 1997.

[2] O. Aichholzer, T. Biedl, T. Hackl, M. Held, S. Huber, P. Palfrader, and B. Vogtenhuber. Representing directed trees as straight skeletons. In *Internat. Symp. Graph Drawing*, pages 335–347. Springer, 2015.

[3] A. D. Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005. Monographs in Mathematics. Translation of the 1950 Russian edition by N. S. Dairbekov, S. S. Kutateladze, and A. B. Sossinsky.

[4] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. *Discrete Comput. Geom.*, 8:219–250, 1992.

[5] T. C. Biedl, C. Grimm, L. Palios, J. R. Shewchuk, and S. Verdonschot. Realizing farthest-point Voronoi diagrams. In *Proc. 28th Canad. Conf. Comput. Geom.*, 2016.

[6] H. Cheng, S. L. Devadoss, B. Li, and A. Risteski. Skeletal configurations of ribbon trees. *Discrete Applied Math.*, 170:46–54, 2014.

[7] J.-i. Itoh, C. Nara, and C. Vîlcu. Continuous flattening of convex polyhedra. In *Computational Geometry*, volume 7579, pages 85–97. Springer LNCS, 2012.

[8] J.-i. Itoh and S. V. Sabau. Riemannian and Finslerian spheres with fractal cut loci. *Differential Geom. Applications*, 49:43–64, 2016.

[9] J.-i. Itoh and C. Vîlcu. Farthest points and cut loci on some degenerate convex surfaces. *J. Geometry*, 80(1-2):106–120, 2004.

[10] J.-i. Itoh and C. Vîlcu. Every graph is a cut locus. *J. Math. Society Japan*, 67(3):1227–1238, 2015.

[11] D. Kane, G. N. Price, and E. D. Demaine. A pseudopolynomial algorithm for Alexandrov's theorem. In *Workshop Algorithms Data Struct.*, pages 435–446. Springer, 2009.

[12] G. Liotta and H. Meijer. Voronoi drawings of trees. *Computational Geometry*, 24(3):147–178, 2003.

[13] J. O'Rourke and C. Vîlcu. Tailoring for every body: Reshaping convex polyhedra. `https://arxiv.org/abs/2008.01759`, Aug. 2020.

[14] H. Poincaré. Sur les lignes géodésiques des surfaces convexes. *Trans. Amer. Math. Soc.*, 6:237–274, 1905.

[15] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Computing*, 15(1):193–215, 1986.

[16] R. Uehara. Common nets of (regular) polyhedra. In *Introduction to Computational Origami*, pages 59–76. Springer, 2020.

[17] G. M. Ziegler. *Lectures on Polytopes*, volume 152. Springer Science & Business Media, 2012.

Unfolding a New Class of Orthographs of Arbitrary Genus

Mirela Damian[*] Robin Flatland[†]

Abstract

We show that every 3-separated orthograph (in which no two boxes of degree 3 of higher are adjacent, and no grid edge is entirely surrounded by boxes) of degree at most 3 can be unfolded with a 7×7 refinement of the grid faces. This result extends the class of well-separated orthographs known to have an unfolding by allowing boxes of degree 2 to be adjacent to each other and to higher degree boxes.

1 Introduction

An *unfolding* of a polydedron is obtained by cutting its surface and flattening it out into the plane as a single, non-overlapping piece known as a *net*. In an *edge unfolding*, cuts are only allowed along the polyhedron's edges, whereas in a *general unfolding*, cuts are allowed anywhere, including across the interior of faces. It is known that every convex polyhedron has a general unfolding [9, Sec. 24.1.1], but whether they have an edge unfolding or not is a long standing open question [9, Ch. 22]. Conversely, it is known that not all non-convex polyhedra have an edge unfolding [1], but whether they have a general unfolding or not is an open question.

Little progress has been made on algorithms for finding general unfoldings of non-convex polyhedra, with the notable exception of orthogonal polyhedra. An *orthogonal polyhedron* is a polyhedron with each face perpendicular to one of the three coordinate axes. Although not all orthogonal polyhedra have edge unfoldings [2], there are algorithms that produce a general unfolding for any orthogonal polyhedron of genus zero [8, 4, 3] and up to genus two [5]. These unfoldings make cuts across faces, but the cuts are limited to a defined set of added edges. Specifically, the polyhedron's faces are first subdivided into rectangular *grid faces* by adding edges at the intersection between the polyhedron's surface and axis-perpendicular planes passing through each vertex. Then each rectangular face is further refined by subdividing it into an $(a \times b)$ orthogonal grid of edges, for some positive integers $a, b \geq 1$. A progression of results has reduced the amount of refinement needed from $(O(2^n) \times O(2^n))$ [8], to $(O(n^2) \times O(n^2))$ [4], to $(O(n) \times O(n))$ [3, 5]. For several specialized orthogonal shape classes, it has been shown that only a constant amount of refinement is needed. For example, there exist algorithms for unfolding orthostacks using a (1×2) refinement [2], Manhattan Towers using a (4×5) refinement [7], and low-degree orthotrees using a (4×4) refinement [6].

Unfolding orthogonal polyhedra of genus greater than 2 has proved challenging. In [11], it was shown that extrusions of orthogonal polygons with any number of square holes admit unfoldings. Chang and Yen [3] generalized this to extrusions of orthogonal polygons with arbitrarily shaped (orthogonal) holes. A less restrictive subclass of orthogonal polyhedra of arbitrary genus shown to be unfoldable is the class of well-separated orthographs [10]. An orthograph $\mathcal{O}$ is a polyhedron composed of rectangular boxes glued together along congruent faces, whose surface is a 2-manifold. The degree of a box is the number of boxes glued to it. A box is said to be a *bend* if its degree is at least 2 and it has two boxes glued to it along perpendicular faces (note that, by this definition, all boxes of degree 3 or more are bends; for bends of degree 2, see boxes $C_1 \dots C_9$ from Figure 1). An orthograph is *well-separated* if no two bends are adjacent. In [10], Chang *et al.* showed that every well-separated orthograph has a (2×1) unfolding.

The work presented here takes a first step towards loosening the well-separated requirement from [10]. Specifically, we say that an orthograph is *3-separated* if no two boxes of degree 3 or higher are adjacent, and no grid edge is completely surrounded by boxes (i.e, there is no cycle of length 4). The class of 3-separated orthographs subsumes the class of well-separated orthographs by allowing bends of degree two to be adjacent to each other and to higher degree boxes. We show that all 3-separated orthographs of degree at most 3 can be unfolded using a (7×7) refinement. Although this class of polyhedra may seem restrictive, ideas used in this paper show promise of extending to orthographs of arbitrary degree. Our unfolding algorithm is an adaptation of the Euler path algorithm from [10].

2 Terminology

An *orthograph* $\mathcal{O}$ is a polyhedron composed of rectangular boxes glued together along congruent faces, whose surface is a 2-manifold. A face on the surface of $\mathcal{O}$ is

[*]Department of Computer Science, Villanova University, Villanova, PA, mirela.damian@villanova.edu

[†]Department of Computer Science, Siena College, Loudonville, NY, flatland@siena.edu

called *open*; a face shared by two boxes of $\mathcal{O}$ is *closed*. The *dual graph* $G = (V, E)$ of $\mathcal{O}$ has a vertex for each box in $\mathcal{O}$ and an edge for each pair of boxes glued together in $\mathcal{O}$, whose endpoints are the corresponding dual vertices. (Throughout this paper, we will use the terms *box* and *vertex* interchangeably.) An orthograph is *3-separated* if no two boxes of degree 3 or higher are adjacent, and no grid edge is entirely surrounded by cubes (i.e, its dual graph does not contain a cycle of length 4). For an example of a 3-separated orthograph and its dual graph, see Figure 1(a,b).

A box of degree one in G is a *leaf*; a box of degree 2 is a *connector*; any other box is a *junction*. A connector C is of *type*-1 if its two neighbors are attached to opposite (parallel) faces of C; otherwise, C is of *type*-2. Figure 1a shows an example of an orthograph with five degree-3 junctions $J_1 \dots J_5$, nine type-1 connectors $A_1 \dots A_9$, nine type-2 connectors $C_1 \dots C_9$, and three leaves $D_1 \dots D_3$.

A type-2 connector is in *standard position* if its two neighbors are attached to its front and right faces (from a viewpoint at $z = \infty$). For example, the type-2 connector C_1 from Figure 1a is in standard position. If no two neighbors of a degree-3 junction J are attached to opposite faces of J, then J is called *orthogonal*; otherwise, J is called *T-shaped*. The orthograph from Figure 1a contains two orthogonal junctions J_2 and J_3 and three T-shaped junctions J_1, J_4 and J_5.

An orthograph composed of cubes is called a *polycube graph*. For simplicity, we describe our unfolding algorithm on polycube graphs composed of cubes that are 7 units long (so a 7×7 refinement produces unit square grid faces), then show how it extends to orthographs. So throughout the rest of the paper, $\mathcal{O}$ refers to a 3-separated polycube graph composed of cubes of side length 7, unless otherwise specified.

3 Ports and Links

Similar to the approach in [10], we select certain edges to serve as *ports* and link them together to guide the unfolding. For each closed face f of box A in $\mathcal{O}$, we assign two edges of f to be *ports*. Ports on different closed faces of A are paired together by a set of *links*. A link between ports p and q is denoted by $p \sim q$. If $p = q$ (i.e, p and q refer to the same edge shared by two of A's closed faces), then the link $p \sim q$ is called *null*; otherwise, the link $p \sim q$ is called *non-null*. For example, the link $u \sim u$ from Figure 3a that pairs port u on J's left face with port u on J's bottom face is a null link, whereas the link $a \sim x$ that pairs port a on J's left face with port x on J's back face is a non-null link.

In the rest of this section, we show how to select two ports for each closed face of A. Although similar to the construction used in [10], our selection of ports is slightly different due to the restriction that they satisfy the following property.

Property 1 *No port (edge) is shared by the two neighbors of a type-2 connector.*

Property 1 will facilitate unfolding a type-2 connector, as discussed later (Section 6.2).

3.1 Assigning Ports to Connectors

Let C be a (type-1 or type-2) connector. If a closed face f of C is shared with a junction J, then the two ports on f are selected according to the junction rules applied to J (as discussed later in Section 3.2). Otherwise, f is shared with a leaf or another connector A. In this case we select the ports a, b on f to be any two edges of f adjacent to open faces of A and C (see Figure 2 for some examples). Since neither A nor C is a junction, such edges always exist.

In the case of connectors, pairing ports with links will be done later, during the unfolding procedure.

3.2 Assigning Ports and Links to Junctions

Next we describe a method for assigning ports to junctions and pairing them with links. In this paper we handle junctions of degree 3 only.

Let J be a junction of degree 3. Note that each of J's neighbors is a connector (by the 3-separation condition), and at least one of J's neighbors shares an edge with each of the other two neighbors of J. Let B be such a neighbor of J. We start by selecting the first two ports of J to be the two edges u, v shared by B with the other two neighbors of J (see Figure 3.) The shared edges u, v are associated with one closed face f of J, and contribute one port to each of the other two closed faces of J. For any other closed face $f' \neq f$ shared by J with a neighbor C, we select the second port to be an edge of f' adjacent to open faces of J and C (see the ports labeled a, x in Figure 3). Since C is a connector, such an edge always exist. Note that all ports associated with J satisfy Property 1. In other words, if one of J's neighbors is a type-2 connector sharing one of J's ports, then the shared port satisfies Property 1.

We pair the ports $u \sim u$ and $v \sim v$ by null links (marked in a dashed line in Figure 3), and $a \sim x$ by a non-null link (marked in a thick solid line in Figure 3).

4 Rings, Gates and Wings

For each box $A \in \mathcal{O}$ and each closed face f of A, we introduce the following definitions. The *ring* $r \in A$ *associated with* f is the set of all points on the surface of A (not necessarily on the surface of $\mathcal{O}$) within unit distance from f. (Note that rings are associated with

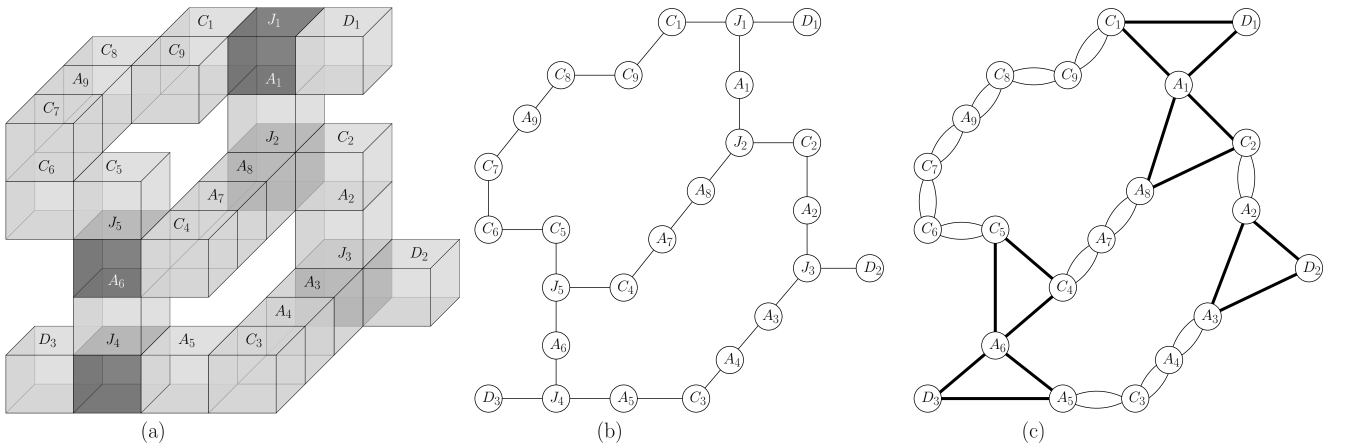


Figure 1: (a) Orthograph $\mathcal{O}$ with degree-3 junctions $J_1 \dots J_5$ (labels attached to top faces), type-1 connectors $A_1 \dots A_9$, type-2 connectors $C_1 \dots C_9$, and leaves $D_1 \dots D_3$. (b) Dual graph of $\mathcal{O}$ (c) Modified dual graph, with connector (junction) edges marked with thin (thick) lines.

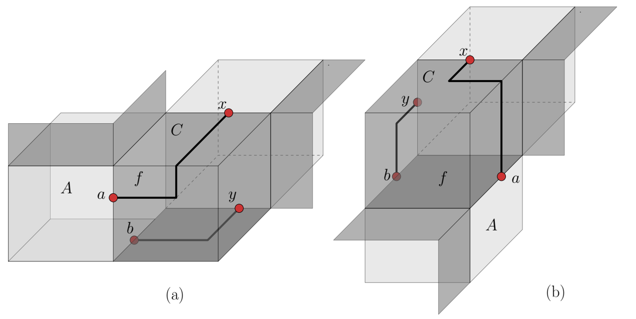


Figure 2: Associating ports (a, b, x, y) with a type-2 connector C. Links $a \sim x$ and $b \sim y$ (arbitrarily picked here) are marked with thick lines.

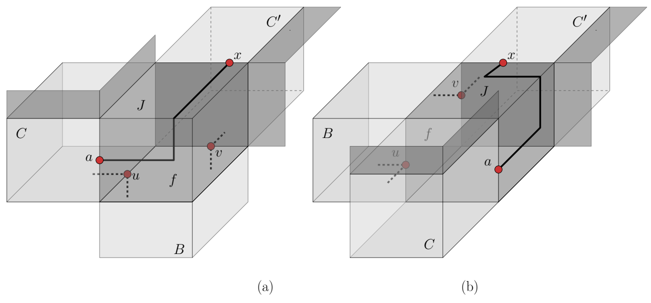


Figure 3: Associating ports and links with a degree-3 junction J (a) orthogonal (b) T-shaped.

closed faces only.) A *face* of r (which we refer to as a *ring face*) is a maximal set of ring points that lie on a face of A. Note that each ring face is 7 units long and 1 unit wide (since each face of $\mathcal{O}$ is 7 units long and 7 units wide). We view each ring face $g \in r$ as the union of three pieces: the middle 1×1 piece of g, which we refer to as a *gate*, and two 3×1 strips of g attached to either side of the gate, which we refer to as *wings*. Thus, each ring includes four gates (one per face) and eight wings (two per face). Refer to Figure 4.

A gate unattached to a port is called *free*; otherwise, the gate is *locked*. Note that each wing w is incident to two faces of A (one containing w, and one including just a short edge of w). If both faces of A incident to a wing are open, we say that the wing is *free*; otherwise, the wing is *locked*. We say that a gate (wing) of A *extends* (or is an *extension* of) the box B sharing the closed face f with A.

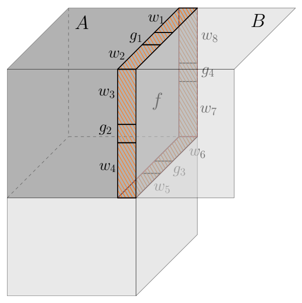


Figure 4: Rings, gates and wings: A's ring associated with its closed face f is marked with stripes; g_1, g_2, g_3 and g_4 are gates; w_1 through w_8 are wings; wings w_1, w_2, w_3 and w_8 are free, and the rest are locked; all gates and rings are *extensions* of B.

Property 2 *A wing/gate extends a unique box, and any two free wings are interior-disjoint.*

5 Modified Dual Graph

Similar to the approach presented in [10], we seek an unfolding path in a multigraph that is a modified version of the dual graph G of $\mathcal{O}$. The requirement for the multigraph, which may not be met by the original dual graph G, is that it must admit an Euler cycle that yields a valid unfolding. We modify the dual graph G to satisfy this requirement as follows:

1. For each edge (B, C), such that neither B nor C is a junction, replace the edge (B, C) by parallel edges $\{(B, C), (B, C)\}$. We call these edges *connector* edges.

2. For each degree-3 junction J with neighbors A, B and C, replace the edges $\{(J, A), (J, B), (J, C)\}$ with $\{(A, B), (A, C), (B, C)\}$, then remove J from G. We call these new edges in G *junction* edges associated with J. Note that there is a one-to-one correspondence between these junction edges and the links associated with J.

Note that the first rule above differs from the one from [10] for the case when at least one of B and C is a type-2 connector, to accommodate the case of adjacent type-2 connectors. Figure 1b shows the dual graph corresponding to the orthograph from Figure 1a, and Figure 1c shows the modified dual graph.

These modification rules, along with the 3-separateness condition, yield the following property.

Property 3 *Each leaf (non-leaf) vertex of the modified dual graph G has degree 2 (4) and is incident to an even number of junction edges.*

Definition 4 *A cycle ξ in G is an* Euler unfolding cycle *if it satisfies the following properties:*

(E1) ξ passes through all edges of G (so it's an Euler cycle).

(E2) If (A, B, C) is a subpath in ξ and (A, B), (B, C) are both junction edges, then either (i) B is a leaf, or (ii) (A, B) and (B, C) are associated with different junctions; in other words, (A, B, C) is not a cycle in G.

(E3) If (A, B, A) is a subpath in ξ, then B is a leaf.

Referring to the modified dual graph G from Figure 1a, (E_2) of Definition 4 allows (A_2, D_2, A_3) to be a subpath of ξ (since D_2 is a leaf), however ξ may not contain the subpath (A_1, C_1, D_1), for instance. The reason for this restriction is that junction edges in G associated with the same junction correspond to disjoint links (that do not share ports) and the unfolding path can only transition from one link to another by means of a shared port. For a specific example, refer to Figure 3b: the junction edge (B, C) is associated with the null link $u \sim u$, and the junction edge (C, C') is associated with the non-null link $a \sim c$; if (B, C, C') were part of unfolding path, then the unfolding would have to somehow transition from u to a, which may not be possible; this is precisely what we are trying to avoid.

Lemma 5 *The multigraph G contains an Euler unfolding cycle.*

Proof. First note that all the vertices of G are of even degree, therefore G contains an Euler cycle ξ. Suppose that (A, B, C) is a subpath of ξ, and (A, B) and (B, C) are both junction edges. If B is a leaf, or (A, B) and (B, C) are associated with different junctions, then ξ satisfies condition $(E2)$ of Definition 4. Otherwise, the degree of B is 4, and (A, B) and (B, C) are both associated with a junction J. Since ξ is an Euler cycle, ξ must pass through B again along different edges (X, B) and (B, Y). By Property 3, these are both either connector edges or junction edges associated with a different junction $J' \neq J$. Let $\xi = (A, B, C) \oplus \xi_1 \oplus (X, B, Y) \oplus \xi_2$, where ξ_1 and ξ_2 are subpaths of ξ, with ξ_2 ending at A (to close the cycle). Here $\oplus$ denotes concatenation. In this case we rearrange the cycle into $\xi' = (A, B, X) \oplus \overleftarrow{\xi_1} \oplus (C, B, Y) \oplus \xi_2$, where $\overleftarrow{\xi_1}$ is ξ_1 traversed in reverse. Note that ξ' avoids traversing edges of the same junction consecutively. Using this approach (applied as many times as necessary) we construct an Euler cycle that satisfies condition $(E2)$ of Definition 4.

A similar rearrangement can be used to ensure that ξ avoids traversing parallel connector edges (A, B) and (B, A) consecutively, unless B is a leaf, so condition $(E3)$ of Definition 4 is also met. $\square$

6 Nets Realizing Links

The discussion in this section is based on the assumption that ports associated with each box $A \in \mathcal{O}$ have been paired up with links. The pairing rules for the case when A is a junction are described in Section 3.2. If A is a leaf or a connector, the pairing will be implicitly done by the unfolding path, as described later in Section 7. We will later see that the unfolding procedure simply follows a series of links $a_1 \sim a_2 \sim a_3 \sim \ldots a_1$ and realizes them by nets that extend horizontally without overlap. For every link $a_i \sim a_{i+1}$ associated with a box A, we refer to a_i (a_{i+1}) as the *entry*(*exit*) port for A. Note that the same port a_i may serve as an entry port for a box A and an exit port for a different box B, if a_i is shared by A and B.

For any box $A \in \mathcal{O}$, L_A and R_A are the *left* and *right* faces of A (orthogonal to the x-axis); F_A and K_A are the *front* and *back* faces of A (orthogonal to the z-axis); and T_A and B_A are the *top* and *bottom* faces of A (orthogonal to the y-axis).

Definition 6 (Proper Net) *Let $p \sim q$ be a link associated with a box A. We say that a net $\mathcal{N}$ realizing the link $p \sim q$ is* proper *if the following conditions hold:*

(P1) *$\mathcal{N}$ includes the locked gates on p and q and lies between vertical lines passing through them.*

(P2) *Free wings/gates of A can be removed from $\mathcal{N}$ without disconnecting $\mathcal{N}$.*

6.1 Unfolding a Leaf

Theorem 7 *Let $p \sim q$ be a link associated with a leaf A. Then the surface of A can be unfolded into a proper net $\mathcal{N}$ that realizes the link $p \sim q$ and covers the surface of A.*

Proof. Let a be the entry port for A, and assume without loss of generality that a is the top front edge of A. Possible exit ports for A are labeled x, y and z in Figure 5a. Mapped onto the surface of A is an unfolding multipath Υ that spans these entry and exit ports of A.

The unfolding net $\mathcal{N}$ realizing the link $p \sim q$ is shown in Figure 5b. It can be easily verified that $\mathcal{N}$ covers the surface of A and satisfies property (*P1*) of Definition 6. Also note that the free wings of A (dark-shaded in Figure 5b) can be removed from $\mathcal{N}$ without disconnecting $\mathcal{N}$, and similarly for A's free gates (so any gate other than p and q). Thus property (*P2*) of Definition 6 is also satisfied, so $\mathcal{N}$ is a proper net. □

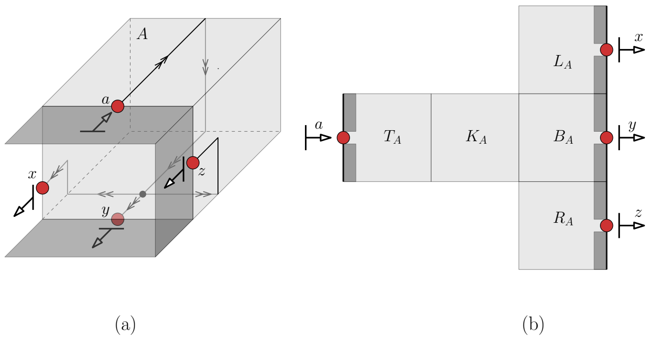


Figure 5: Leaf A with a front neighbor. (a) Unfolding multipath that spans entry port a and exit ports $\{x, y, z\}$. (b) Unfolding net $\mathcal{N}$ that realizes the link $a \sim q$, for any $q \in \{x, y, z\}$; the dark-shaded pieces are the free wings.

6.2 Unfolding Connectors

The proof of the following theorem spans sections Sections 6.2.1 and 6.2.2.

Theorem 8 *Let $p \sim q$ and $p' \sim q'$ be a pair of links associated with a connector C, with $p \neq p'$ and $q \neq q'$. Then the surface of C can be unfolded into two proper nets $\mathcal{N}$ and $\mathcal{N}'$, such that $\mathcal{N}$ realizes the link $p \sim q$ and N' realizes the link $p' \sim q'$, and $\mathcal{N} \cup \mathcal{N}'$ covers the surface of C.*

6.2.1 Unfolding a Type-1 Connector

We first prove the result of Theorem 8 for the case when C is a type-1 connector. Assume without loss of generality that C has front and back neighbors, and p is the front top edge of C (if this is not the case, we can always reorient C to meet this restriction). We further assume without loss of generality that $p' \in \{b, c\}$, where b is the front right edge of C, and c is the front bottom edge of C; the case when p' is the front left edge of C is symmetric to the case $p' = b$. Refer to Figure 6. Let x, y, z and t be the exit ports of C in clockwise order, starting with the back top edge x of C. Mapped onto the surface of C in Figure 6a is an unfolding multipath that maps the entry port $p = a$ to these exit ports; for clarity purposes, we do not overlay this multipath with unfolding paths from p' to the exit ports. We consider

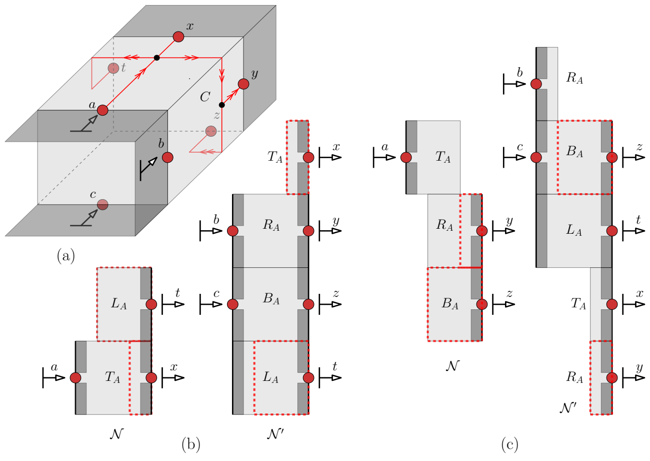


Figure 6: Type-1 connector C with front and back neighbors. (a) Unfolding multipath that spans entry port a and exit ports $\{x, y, z, t\}$; to keep it clear, paths from entry ports b, c to exit ports are not shown here. (b) Unfolding nets $\mathcal{N}$ and $\mathcal{N}'$ that realize the links $p \sim q$ and $p' \sim q'$ respectively, with $p = a$, $q \in \{x, t\}$, $p' \in \{b, c\}$ and $q' \in \{x, y, z, t\}$; the dash-outlined piece of L_A (T_A) gets attached to $\mathcal{N}$ if $q = t$ ($q = x$), otherwise it gets attached to $\mathcal{N}'$; the dark-shaded pieces are the free wings (c) Unfolding nets $\mathcal{N}$ and $\mathcal{N}'$ that realize the links $p \sim q$ and $p' \sim q'$ respectively, with $p = a$, $q \in \{y, z\}$, $p' \in \{b, c\}$ and $q' \in \{x, y, z, t\}$; the dash-outlined piece of R_A (B_A) gets attached to $\mathcal{N}$ if $q = y$ ($q = z$), otherwise it gets attached to $\mathcal{N}'$; the dark-shaded pieces are the free wings.

four cases:

- If $q = x$, then C can be unfolded as in Figure 6b, with the dash-outlined piece of T_A attached to $\mathcal{N}$ (and not to $\mathcal{N}'$) the dash-outlined piece of L_A attached to $\mathcal{N}'$ (and not to $\mathcal{N}$).
- If $q = t$, then C can be unfolded as in Figure 6b, with the dash-outlined piece of T_A attached to $\mathcal{N}'$

(and not to $\mathcal{N}$) the dash-outlined piece of L_A attached to $\mathcal{N}$ (and not to $\mathcal{N}'$).

- If $q = y$, then C can be unfolded as in Figure 6c, with the dash-outlined piece of R_A attached to $\mathcal{N}$ (and not to $\mathcal{N}'$) the dash-outlined piece of B_A attached to $\mathcal{N}'$ (and not to $\mathcal{N}$).
- If $q = z$, then C can be unfolded as in Figure 6c, with the dash-outlined piece of R_A attached to $\mathcal{N}'$ (and not to $\mathcal{N}$) the dash-outlined piece of B_A attached to $\mathcal{N}$ (and not to $\mathcal{N}'$).

In all cases, it can be easily verified that $\mathcal{N} \cup \mathcal{N}'$ covers the surface of C, and $\mathcal{N}$ and $\mathcal{N}'$ satisfy property (*P1*) of Definition 6. Note that the free wings of C, which are dark-shaded in Figure 6(b,c), can be removed from $\mathcal{N}$ ($\mathcal{N}'$) without disconnecting $\mathcal{N}$ ($\mathcal{N}'$). This is also true for any of C's free gates (so any gate other than $p = a$, q, p', q'). Thus property (*P2*) of Definition 6 is also satisfied. It follows that $\mathcal{N}$ and $\mathcal{N}'$ are proper nets.

6.2.2 Unfolding a Type-2 Connector

In this section we prove the result of Theorem 8 for the case when C is a type-2 connector. Assume without loss of generality that C is in standard position, and A and D are the front and right neighbors of C, respectively (refer to Figure 7a). Property 1 guarantees that no port is associated with the edge shared by A and D. Possible entry (exit) ports for C are labeled a, b and c (x, y and z) in Figure 7a. Mapped onto the surface of C is an unfolding multipath that spans these entry and exit ports.

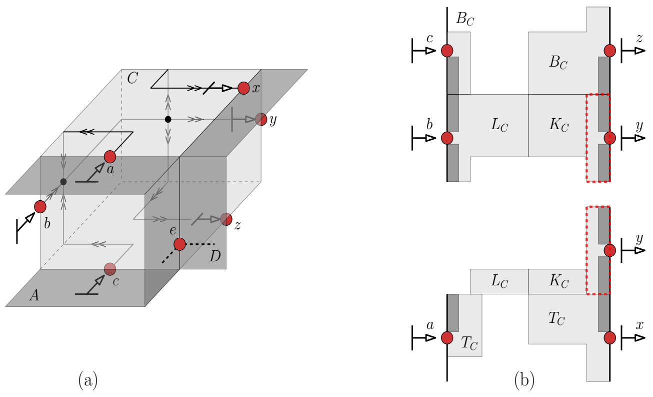


Figure 7: (a) Type-2 connector C in standard position and unfolding multipath that spans entry ports $\{a, b, c\}$ and exit ports $\{x, y, z\}$. (b) Unfolding nets $\mathcal{N}$ (bottom) and $\mathcal{N}'$ (top) that realize the links $p \sim q$ and $p' \sim q'$ respectively, with $p = a$, $q \in \{x, y\}$, $p' \in \{b, c\}$ and $q' \in \{y, z\}$; the dash-outlined piece of K_C gets attached to the bottom net if $q = y$, otherwise it gets attached to the top net; the dark-shaded pieces are the free wings.

Assume without loss of generality that one of the ports, say p, is equal to a. Note that the case $p = c$ is vertically symmetric, and if $p = b$, then p' is either a or c, so this case reduces to the general case as well.

Assume first that the two links correspond to *non-crossing* paths on C's surface, meaning that one path is always on the same side of the other path (although the two paths may share vertices and edges). We consider two cases:

- If the first link is $a \sim x$, then possible choices for the second link are $b \sim y$, $b \sim z$, $c \sim y$ and $c \sim z$. The unfolding for this case is depicted in Figure 7b, with the dash-outlined piece of K_C attached to the top net $\mathcal{N}'$ (and not to $\mathcal{N}$).
- If the first link is $a \sim y$, then the only possible choices for the second link are $b \sim z$ and $c \sim z$ (any other choice would correspond to a path crossing the path from a to x in Υ). The unfolding for this case is the same as before (see Figure 7b), but with the dash-outlined piece of K_C attached to the bottom net $\mathcal{N}$ (and not to $\mathcal{N}'$).

In both cases, it can be easily verified that $\mathcal{N} \cup \mathcal{N}'$ covers the entire surface of C, and $\mathcal{N}$ and $\mathcal{N}'$ satisfy property (*P1*) of Definition 6. The free wings of C, which are dark-shaded in Figure 7b, can be removed from $\mathcal{N}$ ($\mathcal{N}'$) without disconnecting $\mathcal{N}$ ($\mathcal{N}'$). This is also true for any of C's free gates (so any gate other than p, q, p', q'). Thus property (*P2*) of Definition 6 is also satisfied, so $\mathcal{N}$ and $\mathcal{N}'$ are proper nets.

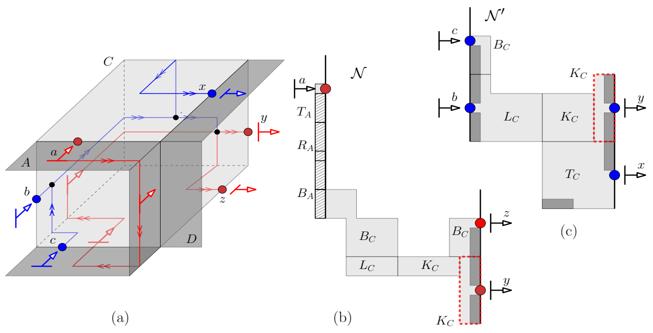


Figure 8: Type-2 connector C with front neighbor A and right neighbor D, case when B_A and T_A are open. (a) Unfolding multipath that spans entry ports $\{a, b, c\}$ and exit ports $\{x, y, z\}$. (b) Unfolding net $\mathcal{N}$ that realizes the link $a \sim q$, for any $q \in \{y, z\}$; the dash-outlined piece of K_C gets attached here only if $q = y$; otherwise, it gets attached to $\mathcal{N}'$. (c) Unfolding net $\mathcal{N}'$ that realizes the link $p' \sim q'$, for any $p' \in \{b, c\}$ and $q' \in \{x, y\}$; the dark-shaded pieces are the free wings of C.

Assume now that the two links correspond to *crossing* paths on C's surface. In this case, since $p = a$, the only cases leading to crossing paths are $q \in \{y, z\}$, $p' \in \{b, c\}$ and $q' \in \{x, y\}$, with $q' \neq q$. First note that B_A and

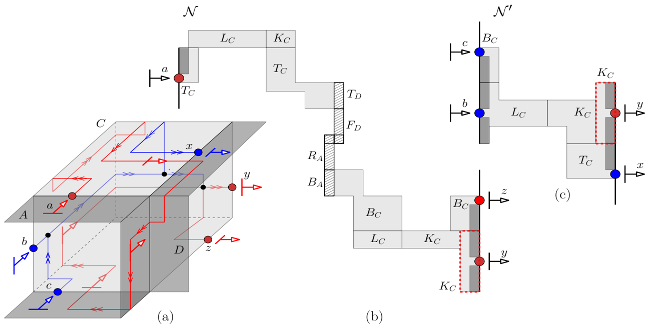


Figure 9: Type-2 connector C with front neighbor A and right neighbor D, case when B_A and T_D are open. (a) Unfolding multipath that spans entry ports $\{a,b,c\}$ and exit ports $\{x,y,z\}$. (b) Unfolding net $\mathcal{N}$ that realizes the link $a \sim q$, for any $q \in \{y,z\}$; the dash-outlined piece of K_C gets attached here only if $q = y$; otherwise, it gets attached to $\mathcal{N}'$. (c) Unfolding net $\mathcal{N}'$ that realizes the link $p' \sim q'$, for any $p' \in \{b,c\}$ and $q' \in \{x,y\}$; the dark-shaded pieces are the free wings of C.

B_D cannot both be closed, since the surface of $\mathcal{O}$ is a 2-manifold. Assume without loss of generality that B_A is open. If T_A is also open, then C's unfolding for this case is shown in Figure 8. Mapped onto the surface of $C \cup A$ is an unfolding multipath that spans all entry and exit ports of C (see Figure 8a). The unfolding net $\mathcal{N}$ shown in Figure 8b realizes any of the links $a \sim y$ and $a \sim z$. The unfolding net $\mathcal{N}'$ shown in Figure 8c realizes the link $p \sim q$, for any $p \in \{b,c\}$ and $q \in \{x,y\}$. If the link $a \sim y$ exists, then the dash-outlined piece of K_C is attached to $\mathcal{N}$; otherwise, it is attached to $\mathcal{N}'$. In either case, if y is an exit port, the dash-outlined piece lies along y.

Note that $\mathcal{N}$ and $\mathcal{N}'$ satisfy property (*P1*) of Definition 6, and $\mathcal{N} \cup \mathcal{N}'$ covers the entire surface of C. One observation here is that $\mathcal{N}$ borrows four free extension wings and one free extension gate from the front box A (see the striped pieces from Figure 8b). By Property 2, these pieces are extensions of C only, therefore there is no conflict over their use. By property (*P2*) of Definition 6 applied to A, removal of these wings and gates from A's net does not disconnect A's net. Now notice that the free wings of C (dark-shaded in Figure 8b,c) can be removed from $\mathcal{N}$ ($\mathcal{N}'$) without disconnecting $\mathcal{N}$ ($\mathcal{N}'$). This is also true for any of C's free gates (so any gate other than p, q, p', q'). Thus property (*P2*) of Definition 6 is also satisfied, so $\mathcal{N}$ and $\mathcal{N}'$ are proper nets.

If T_A is closed, then T_D must be open (this is the more difficult case). C's unfolding for this case is shown in Figure 9. Mapped onto the surface of $C \cup A \cup D$ is an unfolding multipath that spans all entry and exit ports of C (see Figure 9a). The unfolding net $\mathcal{N}$ shown in Figure 9a realizes any of the links $a \sim y$ and $a \sim z$. The unfolding net $\mathcal{N}'$ shown in Figure 9c realizes the link $p' \sim q'$, for any $p' \in \{b,c\}$ and any $q' \in \{x,y\}$. As in the previous case, if the link $a \sim y$ exists, then the dash-outlined piece of K_C is attached to $\mathcal{N}$; otherwise, it is attached to $\mathcal{N}'$. It can be verified that $\mathcal{N}$ and $\mathcal{N}'$ satisfy property (*P1*) of Definition 6. Note that $\mathcal{N}$ borrows four free extension wings and two free extension gates from neighbors A and D (see the striped pieces from Figure 9b). Arguments similar to the ones above show that only C's nets can use these pieces (by Property 2) and their removal from A's and D's nets does not disconnect these nets (by property (*P2*) of Definition 6 applied to A and D). All other pieces in $\mathcal{N} \cup \mathcal{N}'$ belong to the surface of C and cover the entire surface of C. Finally, notice that the free wings of C (dark-shaded in Figure 9b,c) can be removed from $\mathcal{N}$ ($\mathcal{N}'$) without disconnecting $\mathcal{N}$ ($\mathcal{N}'$). This is also true for any of the free gates, so property (*P2*) of Definition 6 is satisfied. It follows that $\mathcal{N}$ and $\mathcal{N}'$ are proper nets. This concludes the proof of Theorem 8.

6.3 Unfolding a Degree-3 Junction

As discussed in Section 3.2, each degree-3 junction J is associated with two null links and one non-null link. The null links are realized by empty unfolding nets, while the non-null link is realized by J's unfolding net according to the following theorem.

Theorem 9 *Let $p \sim q$ be a non-null link associated with a degree-3 junction J. Then the surface of J can be unfolded into a proper net $\mathcal{N}_J$ that realizes the link $p \sim q$ and covers the surface of J.*

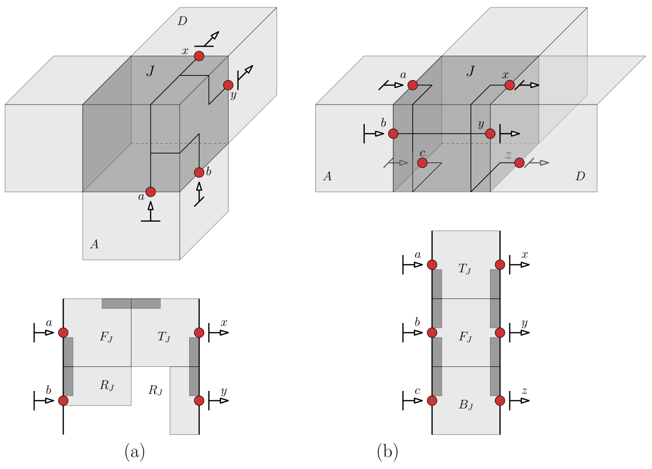


Figure 10: Unfolding degree-3 junction J. (a) Unfolding net $\mathcal{N}_J$ of ortogonal junction J that realizes the link $p \sim q$, for any $p \in \{a,b\}$ and $q \in \{x,y\}$. (b) Unfolding net $\mathcal{N}_J$ of T-shaped junction J that realizes the link $p \sim q$, for any $p \in \{a,b,c\}$ and $q \in \{x,y,z\}$. The dark-shaded pieces in the unfolding nets are the free wings.

Proof. Let A and D be J's neighbors connected by the non-null link. Consider first the case when J is orthogonal, as depicted in Figure 10a. Potential entry ports for J are labeled a and b, and potential exit ports are labeled x and y. Mapped onto the surface of J is an unfolding multipath that spans these entry and exit ports. The unfolding net $\mathcal{N}_J$ from Figure 10a realizes the link $p \sim q$, with $p \in \{a, b\}$ and $q \in \{x, y\}$.

Consider now the case when J is a T-shaped junction, as depicted in Figure 10b. Potential entry (exit) ports for J are labeled a, b and c (x, y and z). Marked on the surface of J is an unfolding multipath that spans these entry and exit ports. The unfolding net $\mathcal{N}_J$ from Figure 10b realizes the link $p \sim q$, with $p \in \{a, b, c\}$ and $q \in \{x, y, z\}$.

Note that each net N_J from Figure 10(a,b) covers the surface of J and trivially satisfies property (*P1*) of Definition 6. Also note that the free wings of J (dark-shaded in $\mathcal{N}_J$) can be removed from $\mathcal{N}_J$ without disconnecting $\mathcal{N}_J$, and the same is true for J's gates. Thus property (*P2*) of Definition 6 is also satisfied, so $\mathcal{N}_J$ is a proper net. This concludes the proof. □

7 Unfolding Algorithm

In this section, we use the notation $\mathcal{N}_A$ to refer to an unfolding net associated with a box A. If $\mathcal{N}_A$ and $\mathcal{N}_B$ share a locked gate segment (along the right vertical boundary line of $\mathcal{N}_A$, and left vertical boundary line of $\mathcal{N}_B$), then $\mathcal{N}_A \cup \mathcal{N}_B$ refers to the compound net obtained by gluing $\mathcal{N}_A$ and $\mathcal{N}_B$ along the shared locked gate (with potentially some free wings and free gates redistributed between the two). Table 1 outlines our unfolding algorithm.

Figure 11 shows an unfolding example. The EulerUnf algorithm (step 1) starts by visiting the junction edge $C_3 \to D_1$ associated with the null link $a \sim a$. Next along the Euler cycle (shown in Figure 11c) is the junction edge $D_1 \to A_1$ associated with the non-null link $b \sim c$. Step 2.2 of the EulerUnf algorithm attaches to the unfolding net $\mathcal{N}$ (initially empty) the net $\mathcal{N}_{D_1}$ that realizes the link $a \sim b$, followed by the net $\mathcal{N}_{J_1}$ that realizes the link $b \sim c$. Next along the Euler cycle is the junction edge $A_1 \to D_2$ associated with the null link $d \sim d$. Step 2.2 of the EulerUnf algorithm attaches to $\mathcal{N}$ the net $\mathcal{N}_{A_1}$ that realizes the link $c \sim d$ (the junction net realizing the null link is empty). The algorithm continues this way up to the last edge $A_4 \to C_3$ of the Euler cycle. At this point the last visited port is t and the next edge along the Euler cycle is the junction edge $C_3 \to D_1$ associated with the null link $a \sim a$. Finally, the EulerUnf algorithm attaches to $\mathcal{N}$ the net $\mathcal{N}_{C_3}$ that realizes the link $t \sim a$ and terminates (since all edges along the Euler cycle have been visited).

Lemma 10 *Let $\mathcal{N}^k$ be the partial unfolding net obtained by the* EulerUnf *algorithm upon visiting k edges of ξ, for some $k \geq 1$. Let q be the last port visited by ξ. Then $\mathcal{N}^k$ is a (connected) net that lies left of a vertical line passing though the locked gate on q.*

Proof. The proof is by induction on k. The base case corresponds to $k = 1$. In this case, $\mathcal{N}^k$ is empty (step 1 of the EulerUnf algorithm) and thus satisfies the conditions of this lemma.

The inductive hypothesis states that the theorem holds for $\mathcal{N}^{k-1}$, for some $k > 1$. Let $A \to B$ (p) be the last edge (port) visited by ξ up to this point. To prove the inductive step, let $B \to C$ be the next (k^{th}) edge visited by ξ. We discuss two cases, depending on whether $B \to C$ is a *connector* edge or a *junction* edge.

Assume first that $B \to C$ is a *connector* edge. This implies that B and C are adjacent in $\mathcal{O}$ and share two ports along their shared closed face. Note that at least one of these two ports has not yet been visited (since this is the first time ξ visits $B \to C$, and associated with each closed face are two ports and two edges in G). The algorithm picks one such port q and pairs it with p. Note that p and q are on different closed faces of B, so the link $p \sim q$ must be realized by one of B's nets.

Let's consider now the net $\mathcal{N}_B$ that realizes the link $p \sim q$. By Theorem 8, $\mathcal{N}_B$ is a proper net. Note that $\mathcal{N}_B$ may borrow from its neighbors A and C some free wings/gates extending B (as in Figures 8 and 9); by Property 2, such pieces are an extension of B only, so there is no contention over their use. Similarly, some of B's free wings/gates extending A may have been employed by a net $\mathcal{N}_A$ associated with A. By property (*P2*) of Definition 6 applied to A, removing such pieces from $\mathcal{N}_A$ does not disconnect $\mathcal{N}_A$ (and consequently N^{k-1}), and similarly for $\mathcal{N}_B$; the remaining N^{k-1} and $\mathcal{N}_B$ are glued together along the shared locked gate on p. The resulting net has no overlap, since $\mathcal{N}^{k-1}$ and $\mathcal{N}_B$ lie on either side of the vertical line passing through the locked gate on p (by the inductive hypothesis and the fact that $\mathcal{N}_B$ is a proper net. These together show that the lemma holds for $\mathcal{N}^k = \mathcal{N}^{k-1} \cup \mathcal{N}_B$.

Assume now that $B \to C$ is a *junction* edge, and let J be the junction adjacent to both B and C. Let $j \sim q$ be the link associated with the junction edge $B \to C$, according to the rules described in Section 3.2. Let $\mathcal{N}_J$ be the net that realizes the link $j \sim q$ (could be empty, if $j = q$). By Theorem 9, $\mathcal{N}_J$ is a proper net. Since J is a junction, each of B and C is either a connector or a leaf. Furthermore, p and j lie on different closed faces of B, so the link $p \sim j$ must be realized by one of B's nets, say $\mathcal{N}_B$. Arguments similar to the ones above show that $\mathcal{N}^k = \mathcal{N}^{k-1} \cup \mathcal{N}_B \cup \mathcal{N}_J$ satisfies the lemma. □

It can be verified that all unfoldings from Figures 5 to 10

```
Algorithm EulerUnf($\mathcal{O}$)

Let $G$ be the modified dual graph of $\mathcal{O}$ (Section 5).
Let $\xi$ be a directed Euler unfolding cycle in $G$ (Theorem 5).
1. If $\mathcal{O}$ contains at least one junction
       Let $A \to B$ be a junction edge in $\xi$ associated with a null link $p \sim p$ (always exists).
   Else (* $\mathcal{O}$ contains only connectors *):
       Let $A \to B$ be a connector edge in $\xi$ and let $p$ be an arbitrary port shared by $A$ and $B$.
   Mark $A \to B$ and $p$ as visited.
   Initialize the unfolding net $\mathcal{N} \leftarrow \emptyset$.
2. repeat
       Let $A \to B$ ($p$) be the last dual edge (port) visited by $\xi$.
       Let $B \to C$ be the next dual edge along $\xi$.
       2.1  If $B \to C$ is a connector edge:
                Let $q$ be an unvisited port shared by $B$ and $C$.
                Associate with $B$ the link $p \sim q$.
                Let $\mathcal{N}_B$ be the unfolding net that realizes the link $p \sim q$.
                Update $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}_B$.
       2.2  Else (* $B \to C$ is a junction edge *):
                Let $J$ be the junction box adjacent to both $B$ and $C$ in $\mathcal{O}$.
                Let $j \sim q$ be the link corresponding to the junction edge $B \to C$,
                    with $j$ on $B$ and $q$ on $C$ (Section 3.2).
                Associate with $B$ the link $p \sim j$.
                Let $\mathcal{N}_B$ be the unfolding net that realizes the link $p \sim j$.
                Let $\mathcal{N}_J$ be the unfolding net that realizes the link $j \sim q$ (could be null, if $j = q$).
                Update $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}_B \cup \mathcal{N}_J$.
       Mark $B \to C$ and $q$ as visited.
   until the cycle $\xi$ has been completed.
```

Table 1: Unfolding Algorithm.

can be realized with a (7×7) refinement. The requirement for 7 strips along one dimension arises from the unfolding from Figure 9, where 3 horizontal strips are needed in front of the C's locked gate adjacent to port x on C's top face. Counting 3 strips on each side of the gate, plus the gate strip, gives us 7 strips along one dimension. Our main result is stated by the theorem below.

Theorem 11 *The* EulerUnf *algorithm produces a connected net that covers the entire surface of $\mathcal{O}$, with cuts restricted to a (7×7) refinement.*

8 Conclusion

This paper introduces a new class of orthographs of arbitrary genus that can be unfolded using a (7×7) refinement of the grid faces. If we allow a non-uniform grid refinement, this result extends to orthographs composed of rectangular boxes of arbitrary sizes. The only modification to the current approach would be to fix the smaller dimension of each ring face to $d_{min}/7$, where d_{min} is the smallest of all box dimensions; the remaining refinement cuts could then be equally distributed across non-ring faces. This modification would prevent the vertical strip R_A from Figure 9 from extending too far left and cause overlap; this is the only instance where the sizes of different boxes matter.

Our unfolding algorithm currently handles polycube graphs of maximum degree 3, however it shows promise of extending to polycube graphs of arbitrary degree. The algorithm can be easily extended to handle junctions of degrees 5 and 6, for which the existence of ports that satisfy Property 1 is evident; the links are then constructed as in [10], and the algorithm works as is. The only obstacle appears to be in handling cross-shaped junctions of degree 4, such as the junction J from Figure 12 with two type-2 connectors A and C as neighbors. The approach from [10] would link two ports shared by an open face of J with the neighbors A and C, however one of these two ports would fail to satisfy Property 1 (since it would be shared by the two neighbors of either A or C). This is the only configuration that requires a different approach to selecting ports and links. One strategy would be to select each edge shared by two neighbors of J as a port (these would be J's four vertical edges in Figure 12) and link each port to itself (so the unfolding path doesn't cross any of J's open faces), then find a different placement for the two open faces of J (in the unfolding nets of J's neighbors).

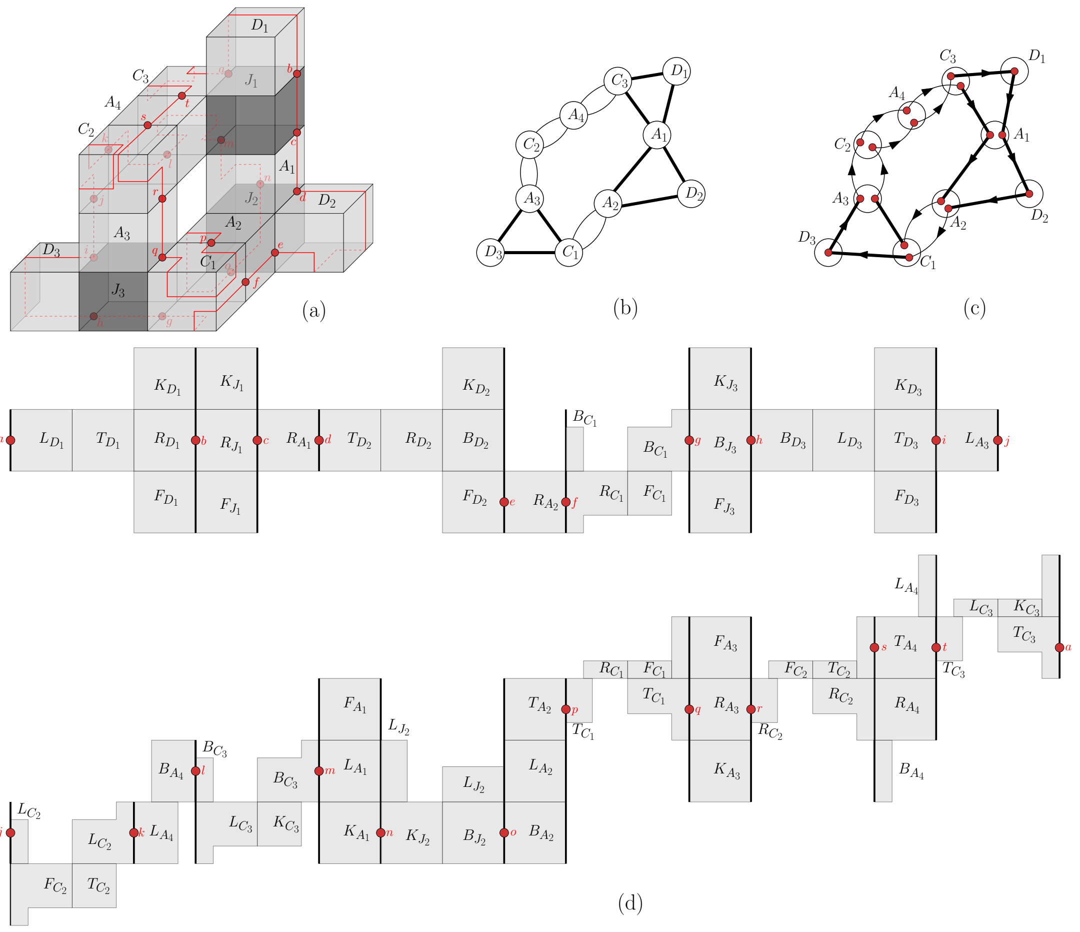


Figure 11: Unfolding example. (a) Polycube graph $\mathcal{O}$ with leaves $D_1 \ldots D_3$, type-1 connectors $A_1 \ldots A_4$, type-2 connectors $C_1 \ldots C_3$ and junctions $J_1 \ldots J_3$. (b) Modified dual graph G of $\mathcal{O}$. (c) Directed Euler unfolding path in G; inner circles corresponds to links (two links per connector and one link per leaf; adjacent links share a port). (d) Unfolding net of $\mathcal{O}$.

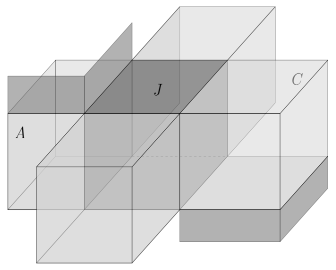


Figure 12: The approach from [10] for selecting ports and links does not work on this junction J of degree 4, since opposite neighbors of J must be linked together and at least one port shared by A or C fails to satisfy Property 1.

References

[1] M. Bern, E. Demaine, D. Eppstein, E. Kuo, A. Mantler, and J. Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory and Applications*, 24(2):51–62, February 2003.

[2] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, Montréal, Canada, August 1998.

[3] Y.-J. Chang and H.-C. Yen. Unfolding orthogonal polyhedra with linear refinement. In *Proceedings of the 26th International Symposium on Algorithms and Computation, ISAAC 2015, Nagoya, Japan*, pages 415–425. Springer Berlin Heidelberg, 2015.

[4] M. Damian, E. Demaine, and R. Flatland. Unfolding orthogonal polyhedra with quadratic refinement: the Delta-unfolding algorithm. *Graphs and Combinatorics*, 30(1):125–140, 2014.

[5] M. Damian, E. Demaine, R. Flatland, and J. O'Rourke. Unfolding genus-2 orthogonal polyhedra with linear refinement. *Graph. Comb.*, 33(5):1357–1379, Sept. 2017.

[6] M. Damian and R. Flatland. Unfolding low-degree orthotrees with constant refinement. In *Proceedings of the 30th Canadian Conference on Computational Geometry*, pages 189–208, Winnipeg, Canada, August 2018.

[7] M. Damian, R. Flatland, and J. O'Rourke. Unfolding Manhattan towers. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 211–214, Windsor, Canada, August 2005.

[8] M. Damian, R. Flatland, and J. O'Rourke. Epsilon-unfolding orthogonal polyhedra. *Graphs and Combinatorics*, 23(1):179–194, 2007.

[9] E. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[10] K.-Y. Ho, Y.-J. Chang, and H.-C. Yen. Unfolding some classes of orthogonal polyhedra of arbitrary genus. *Journal of Combinatorial Optimization*, 37(2):482–500, 2019.

[11] M.-H. Liou, S.-H. Poon, and Y.-J. Wei. On edge-unfolding one-layer lattice polyhedra with cubic holes. In *The 20th International Computing and Combinatorics Conference (COCOON) 2014*, pages 251–262, 2014.

Dispersion on Intervals

Tetsuya Araki [*] Hiroyuki Miyata [†] Shin-ichi Nakano [‡]

Abstract

Given a set of n disjoint intervals on a line and an integer k, we want to find k points in the intervals so that the minimum pairwise distance of the k points is maximized. Intuitively, given a set of n disjoint time intervals on a timeline, each of which is a time span we are allowed to check something, and an integer k, which is the number of times we will check something, we plan the k checking times so that the checks occur at equal time intervals as much as possible, that is, we want to maximize the minimum time interval between the k checking times. We call the problem the k-dispersion problem on intervals. If we need to choose exactly one point in each interval, so $k = n$, and the disjoint intervals are given in the sorted order on the line, then two $O(n)$ time algorithms to solve the problem are known.

In this paper we give the first $O(n)$ time algorithm to solve the problem for any constant k. Our algorithm works even if the disjoint intervals are given in any (not sorted) order. If the disjoint intervals are given in the sorted order on the line, then, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem. Also we show some results on the k-dispersion problem on disks, including a PTAS.

1 Introduction

The facility location problem and many of its variants have been studied [11, 12]. Typically, given a set of locations on which facilities can be placed and an integer k, we want to place k facilities on some locations so that a designated objective function is minimized. By contrast in the *dispersion problem*, we want to place facilities so that a designated objective function is maximized.

In this paper we consider the dispersion problem on intervals. Given a set of n disjoint intervals on a line and an integer k, we want to find k points in the intervals so that the minimum pairwise distance of the k points is maximized. See an example in Fig. 1.

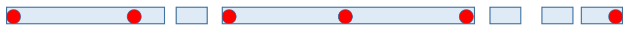


Figure 1: An example of the dispersion problem on intervals with $k = 6$.

Intuitively, given a set of n disjoint (non-overlapping) time intervals on a timeline, each of which is a time span we are allowed to check something, and an integer k, which is the number of times we will check something, we plan the k checking times so that the checks occur at equal time intervals as much as possible, that is, we want to maximize the minimum time interval between the k checking times. We call the problem *the k-dispersion problem on intervals.*

Let S be a set of optimal k points on the line (corresponding to the k checking times), and $cost(S) = \min_{\{s,t\} \subset S}\{d(s,t)\}$ the minimum pairwise distance of the k points in S.

If we need to choose exactly one point in each time interval, and so $k = n$, and the disjoint intervals are given in the sorted order on the line, two $O(n)$ time algorithms to solve the problem are known [5, 18].

Our result In this paper we give the first $O(n)$ time algorithm to solve the problem for any constant k. First we solve the problem where one can choose any number of points in each interval. Then we solve the problem where one can choose at most one point in each interval. Our algorithm works even if the disjoint intervals are given in any (unsorted) order. Our algorithms is based on the pigeonhole principle, and is a generalization of the algorithm in [3] to solve a similar dispersion problem.

If the disjoint intervals are given in the sorted order on the line, then, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem.

Related result Given a set P of n possible locations, and a distance function d for each pair of locations, and an integer k with $k \ll n$, the *max-min k-dispersion problem* computes a subset $S \subset P$ with $|S| = k$ such that the cost $cost(S) = \min_{\{u,v\} \subset S}\{d(u,v)\}$ is maximized. Several results are known for this max-min k-dispersion problem [1, 2, 14, 19, 21]. For the max-sum version several results are also known [4, 6, 8, 9, 10, 15, 17, 19]. For a variety of related problems, see [4, 10]. See more ap-

[*]Gunma University, Japan tetsuya.araki@gunma-u.ac.jp
[†]Gunma University, Japan hmiyata@gunma-u.ac.jp
[‡]Gunma University, Japan nakano@cs.gunma-u.ac.jp

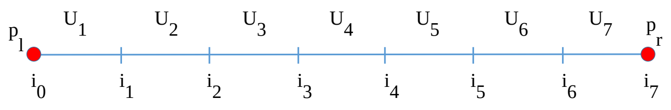


Figure 2: Illustration of $U_1, U_2, \cdots$ for $k = 8$.

plications, including *result diversification*, in [9, 19, 20].

Given a set of n disks on a plane, we want to choose k points, at most one point in each disks, so that the minimum distance among the points is maximized. The problem is called the *dispersion problem on disks*, and some results are known [7, 13, 16]. The k-dispersion problem on intervals is the 1D version of the dispersion problem on disks.

We show some results on the k-dispersion problem on disks. Also we show a variant of the problem where we can choose any number of points in each disk has a PTAS.

The remainder of this paper is organized as follows. In Section 2 we design an $O(n)$ time simple algorithm to solve the dispersion problem when intervals are given unsorted on a line. In Section 3 we give an $O(\log n)$ time algorithm to solve the dispersion problem when intervals are given sorted on a line. In Section 4 we show several results on the k-dispersion problem on disks. Finally Section 5 is a conclusion.

2 k-dispersion for unsorted intervals

In this section we design a simple $O(n)$ time algorithm to solve the k-dispersion problem on intervals when the disjoint n intervals are given unsorted on a line. The idea of our algorithm is a simple divide and conquer algorithm using the pigeonhole principle, as follows. Similar idea is used to solve a similar max-min dispersion problem on a line [3].

Let I be a set of disjoint intervals on a horizontal line and p_ℓ and p_r are the leftmost point and the rightmost point in I. One can find p_ℓ and p_r in $O(n)$ time.

If $k = 1$ then a solution S of the 1-dispersion problem is $\{p_\ell\}$.

If $k = 2$ then the solution S of the 2-dispersion problem is $\{p_\ell, p_r\}$.

If $k = 3$ then let the solution S be $\{p_\ell, p_s, p_r\}$. The solution S consists of p_ℓ and p_r and exactly one more point p_s in some interval in I. We can compute p_s as follows.

Let $i_0 = p_\ell$, $i_2 = p_r$, and let i_1 be the midpoint between p_ℓ and p_r. If some interval in I contains i_1 then $p_s = i_1$. Otherwise, let U_1 be the interval (i_0, i_1), and U_2 be the interval (i_1, i_2). Now p_s appears in either U_1 or U_2. So, by pigeonhole principle, p_s does not appear in U_1, or p_s does not appear in U_2. Thus we have two cases.

Case 1: p_s does not appear in U_1.

In this case, S consists of p_ℓ and the solution of the 2-dispersion problem on intervals in $(i_1, i_2]$, say $R \subset I$, which consists of (1) the leftmost point in R and (2) p_r.

Case 2: p_s does not appear in U_2.

In this case, S consists of p_r and the solution of the 2-dispersion problem on intervals in $[i_0, i_1)$, say $L \subset I$, which consists of (1) the rightmost point in L and (2) p_ℓ.

We can generalize this method for a constant $k > 3$, as follows.

Let $i_0 = p_\ell$, $i_{k-1} = p_r$ and let $i_1, i_2, \cdots, i_{k-2}$ be the points which evenly spaced on the line between p_ℓ and p_r. Clearly the cost of the solution is at most $d(i_0, i_1)$, where $d(i_0, i_1)$ is the distance between i_0 and i_1. If each of $i_1, i_2, \cdots, i_{k-2}$ is contained in some interval in I, then $\{i_0, i_1, \cdots, i_{k-1}\}$ is the solution, and the cost is $d(i_0, i_1)$. Assume otherwise. Let U_j be the interval $(i_{j-1}, i_j]$ for $j = 1, 2, \cdots, k-2$, and U_{k-1} be the interval (i_{k-2}, i_{k-1}). See an example in Fig. 2.

The solution for the k-dispersion problem consists of p_ℓ and p_r and exactly $k-2$ points in (i_0, i_{k-1}). So, by pigeonhole principle, S has no point in at least one of U_1, U_2, $\cdots$, U_{k-1}. Thus we have $k-1$ cases as follows.

Case 1: S has no point in U_1.

If there is an interval in I containing i_1, then replace its left end to i_1.

In this case, S consists of (1) p_ℓ and (2) the solution of $(k-1)$-dispersion problem for the intervals in $[i_1, i_{k-1}]$.

Case 2: S has no point in U_2.

In this case, for some s with $1 \le s \le k-1$, S consists of (1) the solution of s-dispersion problem for the intervals, say L, in $[i_0, i_1]$ (if there is an interval in I containing i_1 then replace its right end to i_1) and (2) the solution of $(k-s)$-dispersion problem for the intervals, say R, in $[i_2, i_{k-1}]$ (if there is an interval in I containing i_2 then replace its left end to i_2). Note that since S has no point in U_2 the solution for L does not affect the solution for R, so we can solve the two smaller subproblems independently. Also note that if there is an interval in I containing both i_1 and i_2, then two subintervals of the interval appear one in L and the other in R.

Case 3: S has no point in U_3.

Similar to Case 2.

$\cdots$

Case $k-2$**:** S has no point in U_{k-2}.

Similar to Case 2.

Case $k-1$**:** S has no point in U_{k-1}.

If there is an interval in I containing i_{k-2}, then replace its right end to i_{k-2}.

In this case, S consists of (1) the solution of $(k-1)$-dispersion problem for the intervals in $[i_0, i_{k-2}]$ and (2)

Algorithm Find-dispersion-on-Intervals(I, k)

```
/* $p_\ell$ and $p_r$ are the leftmost point and the rightmost point in $I$ */
if $k = 1$ then
    $S = \{p_\ell\}$
    return $S$
end if
if $k = 2$ then
    $S = \{p_\ell, p_r\}$
    return $S$
end if
/* $i_0 = p_\ell$, $i_{k-1} = p_r$ and let $i_1, i_2, \cdots, i_{k-2}$ be the points which evenly spaced on the line between $p_\ell$ and $p_r$ */
/* $k \geq 3$ */
if each of $i_1, i_2, \cdots, i_{k-2}$ is contained in some interval in $I$ then
    $S = \{i_0, i_1, \cdots, i_{k-1}\}$
    return $S$
end if
/* Case: $S$ has no point in $U_1 = (i_0, i_1]$ */
Let $R$ be the set of intervals in $(i_1, i_{k-1}]$.
(if there is an interval in $I$ containing $i_1$ then replace its left end to $i_1$ )
$S_L = \{p_\ell\}$
$S_R$ =Find-dispersion-on-a-line($R, k-1$)
$S = S_L \cup S_R$
/* Case: $S$ has no point in $U_j = (i_{j-1}, i_j]$ for $j = 2, 3, \cdots, k-2$ */
for $j = 2$ to $k-2$ do
    Let $L$ be the intervals in $[i_0, i_{j-1}]$.
    (If there is an interval in $I$ containing $i_{j-1}$ then replace its right end to $i_{j-1}$)
    Let $R$ be the intervals in $(i_j, i_{k-1}]$.
    (If there is an interval in $I$ containing $i_j$ then replace its left end to $i_j$)
    for $s = 1$ to $k-1$ do
        $S_L$ =Find-dispersion-on-a-line($L, s$)
        $S_R$ =Find-dispersion-on-a-line($R, k-s$)
        if $cost(S_L \cup S_R) > cost(S)$ then
            $S = S_L \cup S_R$
        end if
    end for
end for
/* Case: $S$ has no point in $U_{k-1} = (i_{k-2}, i_{k-1})$ */
Let $L$ be the set of intervals in $[i_0, i_{k-2}]$.
(If there is an interval in $I$ containing $i_{k-2}$ then replace its right end to $i_{k-2}$)
$S_L$ =Find-dispersion-on-a-line($L, k-1$)
$S_R = \{p_r\}$
if $cost(S_L \cup S_R) > cost(S)$ then
    $S = S_L \cup S_R$
end if
return $S$
```

p_r.

We (recursively) check all possible cases and choose the best one. See algorithm **Find-dispersion-on-intervals**.

Thus if we have the solution of at most $2k^2$ smaller dispersion problems then we can solve the original k-dispersion problem.

We have the following theorem.

Theorem 1 *One can solve the k-dispersion problem on intervals in $O(n)$ time even when the intervals are given unsorted on a line.*

Proof. Consider the tree structure of the recursive calls. Each inner node has at most $2k^2$ children and the height of the tree is at most k, so the number of inner node is at most $(2k^2)^k$. Before calling the children one needs to compute p_ℓ, p_r, L and R by scanning the list of unsorted intervals with buckets L and R. So it needs $O(n)$ time, where n is the number of intervals. Thus each inner node needs $O(n)$ time except for the calls for its children. Therefore the total running time of the algorithm is $O((2k^2)^k n)$. Since k is a constant it is $O(n)$. □

By slightly modifying the algorithm we can solve the similar dispersion problem on intervals where we can choose at most one point in each interval, as follows. (We cannot choose two or more points in an interval in I.) If there is an interval, say $I' \in I$, containing both endpoints of an empty interval (i_{j-1}, i_j) in **Case** j, then we need to consider the following two more subcases. Case (L): The subinterval of I' appears only in L, with its right endpoint (i_{j-1}). Case (R): the subinterval of I' appears only in R, with its left endpoint (i_j) . Now the number of subproblems is at most $(4k^2)^k$, and the total running time of the algorithm is $O((4k^2)^k n)$. Since k is a constant it is $O(n)$.

Theorem 2 *One can solve the k-dispersion problem on intervals with the constraint that we can choose at most one point in each interval in $O(n)$ time even when the intervals are given unsorted on a line.*

3 k-dispersion for sorted intervals

If I is a set of sorted disjoint intervals on a line, and the coordinates of the endpoints of intervals are given as an array in which the coordinates are stored in the sorted order, then by slightly modifying the algorithm we can solve the dispersion problem in $O(\log n)$ time.

We can compute p_ℓ and p_r in $O(\log n)$ time using the array. We can also decide whether some point i is contained in some interval or not in $O(\log n)$ time by binary search on the array. Also instead of computing L

explisitly, we can compute the leftmost interval in L and the rightmost interval in L in $O(\log n)$ time by binary search, and we can regard L as the intervals in I between those two intervals. Similar for R. Thus we can call each child with those information as arguments, instead of L and R. Now the running time is $O((2k^2)^k \log n)$, which is $O(\log n)$ since k is a constant.

Theorem 3 *One can solve the k-dispersion problem on intervals in $O(\log n)$ time when the intervals are given sorted on a line.*

4 Dispersion on Disks

Given a set D of n disjoint disks on a plane and an integer $k \leq |D|$, we wish to find k points in those disks so that the minimum distance between them is maximized. We can choose at most one point in each disk. (Later we consider a similar problem where we can choose any number of points in each disk in Theorem 7.) We call the problem the dispersion problem on disks. Note that the 1D version of the problem is the k-dispersion problem on intervals, which we have discussed in Section 2 and Section 3.

We need some notations. For a set S of points let $cost(S)$ be the minimum pairwise distance of the points in S.

Let C be the set of n center points of the disks in D, C^* a set C' of k points in C maximizing $cost(C')$.

Let S^* be a set S' of k points in D maximizing $cost(S')$, $D(S^*)$ the set of k disks containing S^*, and $C(S^*)$ the set of k center points of $D(S^*)$.

For $k = n$ the problem is NP-hard, APX-hard, and polynomial time 0.707-approximate algorithm is known [13]. For $k < n$ no results are known for the problem.

We have the following lemma and two theorems. Note that we can choose at most one point in each disk since we always choose points from C.

Lemma 4 *One can choose a set $C_A \subset C$ of k points in $O(n^2)$ time so that $cost(C_A) \geq cost(C^*)/2$.*

Proof. Similar to the proof of Theorem 2 in [19]. First we choose the two points having the maximum distance in C. Let initially C_A be the set of the two points. Then repeatedly we append to C_A a point in $C - C_A$ having the maximum distance to C_A so that C_A finally has k points. Thus the algorithm is a simple greedy algorithm.

Let $D(C^*)$ be the set of k disks having centers at C^* with radii $cost(C^*)/2$. Note that the proper inside of disks in $D(C^*)$ are disjoint. When we append a point in $C - C_A$ to C_A, $|C_A| < k$ holds, so there is a disk in $D(C^*)$ properly containing no point in C_A. Now the center of the disk has no point in C_A within distance $cost(C^*)/2$. Thus we can always find a point in $C - C_A$ having no point in C_A within distance $cost(C^*)/2$.

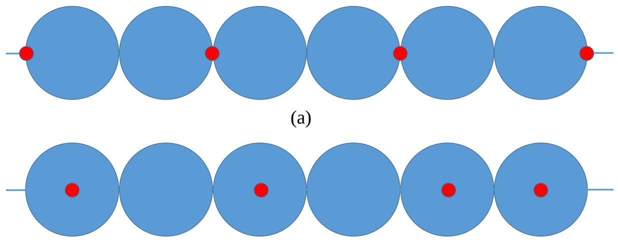


Figure 3: An example with $cost(C_A) = cost(S^*)/2$. The radii of disks are 0.5.

Therefore $cost(C_A) \geq cost(C^*)/2$ holds. □

Theorem 5 *When D is a set of n disjoint disks with arbitrary radii, given an integer $k \leq n$ one can find a set C_A of k points in $C \subset D$ in $O(n^2)$ time so that (1) no two points in C_A are contained in a disk in D and (2) $cost(C_A) \geq cost(S^*)/4$ holds.*

Proof. Since D is disjoint $cost(C(S^*)) \geq cost(S^*)/2$ holds. Also $cost(C^*) \geq cost(C(S^*))$ holds. If we find a set C_A by Lemma 4 we have $cost(C_A) \geq cost(C^*)/2 \geq cost(C(S^*))/2 \geq cost(S^*)/4$. □

Theorem 6 *When D is a set of n disjoint disks with uniform radii, say r, given an integer $k \leq n$ one can find a set C_A of k points in $C \subset D$ in $O(n^2)$ time so that (1) no two points in C_A are contained in a disk in D and (2) $cost(C_A) \geq cost(S^*)/3$ holds.*

Proof. Now $cost(S^*) \geq cost(C^*)$. Since D is disjoint $cost(S^*) - 2r \leq cost(C^*)$ holds. Thus $cost(S^*) \leq cost(C^*) + 2r$ holds. If we find a set C_A by Lemma 4 we have $cost(C_A) \geq cost(C^*)/2$ and so $cost(C^*) \leq 2cost(C_A)$. Now $cost(S^*) \leq 2cost(C_A) + 2r$. Therefore, since $cost(C_A) \geq 2r$, $cost(C_A)/cost(S^*) \geq cost(C_A)/(2cost(C_A)+2r) = 1/(2+2r/cost(C_A)) \geq 1/3$ holds. □

See Fig. 3. The cost of optimal $k = 4$ points S^* is 2 (See Fig. 3(a)), however the cost of $k = 4$ points C_A computed by the greedy algorithm in the proof of Lemma 4 is 1. (See Fig. 3(b)). Thus there is an example for which the greedy algorithm computes a set C_A with $cost(C_A) = cost(S^*)/2$.

Now we consider a variant of the problem where we can choose any number of points in each disk. We have the following theorem. Let S^* be a set S' of k points in D maximizing $cost(S')$.

Theorem 7 *When D is a set of n disjoint disks with arbitrary radii, given an integer $k \leq n$ and a positive real number $\epsilon << 1$ one can choose a set G_A of*

k points in D in $O(n^2/\epsilon^4)$ time so that $cost(G_A) \geq cost(S^*)/(2(1+\epsilon))$ holds. Also one can choose a set G^* of k points in D in $O((n/\epsilon^2)^k)$ time so that $cost(G^*) \geq cost(S^*)/(1+\epsilon)$ holds.

Proof. Let r be the radius of the largest disk in D, and (x', y') be the coordinate of the center of the largest disk. Since we can locate k points in the largest disk so that they evenly spaced on a line segment corresponding to the diameter, $cost(S^*) \geq 2r/(k-1) > 2r/k$ holds.

Now we define the grid points as follows. A point p located at (x, y) is a grid point iff $x = x' + (r\epsilon/ck)i$ and $y = y' + (r\epsilon/ck)j$ with some integers i and j. We will explain later the constant c which defines the size of the cell of the grid.

Let G be the set of points consists of (1) the centers of disks in D, and (2) the grid points contained in disks in D. Now $|G| \leq n + (2r/(r\epsilon/ck))^2 n = n + 4(ck/\epsilon)^2 n$ holds, so $|G|$ is $O(n/\epsilon^2)$.

Let $G(S^*)$ be the set of points derived from S^* by choosing a nearest point in G for each point in S^*. We choose c large enough so that (1) $cost(G(S^*)) \geq cost(S^*)/(1+\epsilon)$ holds and (2) no two points in S^* have the common nearest point in G.

Let $G^* \subset G$ be the set of k points maximizing $cost(G^*)$.

If we find a set $G_A \subset G$ of k points with $cost(G_A) \geq cost(G^*)/2$ in $O(|G|^2)$ time by the greedy algorithm in the proof of Lemma 4, we have $cost(G_A) \geq cost(G^*)/2 \geq cost(G(S^*))/2 \geq cost(S^*)/(2(1+\epsilon))$.

If we find a set G^* in $O(|G|^k)$ time by a brute force algorithm we have $cost(G^*) \geq cost(G(S^*)) \geq cost(S^*)/(1+\epsilon)$. □

Thus this version of the dispersion problem on disks has a PTAS.

5 Conclusion

In this paper we have designed a simple algorithm to solve the k-dispersion problem on intervals. This is the first $O(n)$ time algorithm to solve the problem for any constant k.

Then we have shown, if intervals are given sorted on a line, by slightly modifying the algorithm, one can solve the problem in $O(\log n)$ time. This is the first sublinear time algorithm to solve the problem.

If disjoint intervals on a circle are given sorted on the circle an $O(n)$ time algorithm to solve the n-dispersion problem is known [5, 18]. Can we apply the method in this paper for the k-dispersion problem on disjoint intervals on a circle for any constant k?

We have shown some results on the k-dispersion problem on disks. Also we have shown a variant of the problem where we can choose any number of points in each disk has a PTAS.

References

[1] T. Akagi and S. Nakano, Dispersion on the line, IPSJ SIG Technical Reports, 2016-AL-158-3 (2016).

[2] T. Akagi, T. Araki, T. Horiyama, S. Nakano, Y. Okamoto, Y. Otachi, T. Saitoh, R. Uehara, T. Uno, K. Wasa, Exact Algorithms for the Max-Min Dispersion Problem, Proc. of FAW 2018, LNCS 10823, pp.263-272 (2018).

[3] T. Araki and S. Nakano, Max-Min Dispersion on a Line, Journal of Combinatorial Optimization. Springer, (2020)

[4] C. Baur and S. P. Fekete, Approximation of geometric dispersion problems, Proc. of APPROX 1998, pp. 63–75 (1998).

[5] T. Biedl, A. Lubiw, A. M. Naredla, P. D. Ralbovsky and G. Stroud, Dispersion for Intervals: A Geometric Approach, Prc. of SOSA 2021 (2021)

[6] B. Birnbaum and K.J.Goldman, An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs, Algorithmica, 50, pp. 42–59 (2009).

[7] S. Cabello, Approximation algorithms for spreading points, Journal of Algorithms, 62, pp.49–73 (2007).

[8] A. Cevallos, F. Eisenbrand and R. Zenklusen, Max-sum diversity via convex programming, Proc. of SoCG 2016, pp. 26:1–26:14 (2016).

[9] A. Cevallos, F. Eisenbrand and R. Zenklusen, Local search for max-sum diversification, Proc. of SODA 2017, pp. 130–142 (2017).

[10] B. Chandra and M. M. Halldorsson, Approximation algorithms for dispersion problems, J. of Algorithms, 38, pp. 438-465 (2001).

[11] Z. Drezner, Facility location: A Survey of Applications and Methods, Springer (1995).

[12] Z. Drezner and H.W. Hamacher, Facility Location: Applications and Theory, Springer (2004).

[13] A. Dumitrescu and M. Jiang, Dispersion in Disks, Theory of Computing Systems volume 51, pp.125–142 (2012).

[14] E. Erkut, The discrete p-dispersion problem, European Journal of Operational Research, 46, pp. 48–60 (1990).

[15] S. P. Fekete and H. Meijer, Maximum dispersion and geometric maximum weight cliques, Algorithmica, 38, pp. 501-511 (2004).

[16] J. Fiala, J. Kratochvil, A. Proskurowski, Systems of distant representatives, Discrete Appl. Math., 145, pp.306–36 (2005).

[17] R. Hassin, S. Rubinstein and A. Tamir, Approximation algorithms for maximum dispersion, Operation Research Letters, 21, pp. 133–137 (1997).

[18] S. Li and H. Wang, Dispersing Points on Intervals, Discrete Applied Mathematics, 239, pp. 106-118 (2018).

[19] S. S. Ravi, D. J. Rosenkrantz and G. K. Tayi, Heuristic and special case algorithms for dispersion problems, Operations Research, 42, pp. 299–310 (1994).

[20] M. Sydow, Approximation guarantees for max sum and max min facility dispersion with parameterised triangle inequality and applications in result diversification, Mathematica Applicanda, 42, pp. 241–257 (2014).

[21] D. W. Wang and Y.-S. Kuo, A study on two geometric location problems, Information Processing Letters, 28, pp. 281–286 (1988).

Approximation Algorithms for the Euclidean Dispersion Problems

Pawan K. Mishra[*] Gautam K. Das[†]

Abstract

In this article, we consider the Euclidean dispersion problems. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$. For each point $p \in P$ and $S \subseteq P$, we define $cost_\gamma(p, S)$ as the sum of Euclidean distance from p to the nearest γ point in $S \setminus \{p\}$. We define $cost_\gamma(S) = \min_{p \in S}\{cost_\gamma(p, S)\}$ for $S \subseteq P$. In the γ-dispersion problem, a set P of n points in $\mathbb{R}^2$ and a positive integer $k \in [\gamma + 1, n]$ are given. The objective is to find a subset $S \subseteq P$ of size k such that $cost_\gamma(S)$ is maximized. We consider both 2-dispersion and 1-dispersion problem in $\mathbb{R}^2$. Along with these, we also consider 2-dispersion problem when points are placed on a line.

In this paper, we propose a simple polynomial time $(2\sqrt{3} + \epsilon)$-factor approximation algorithm for the 2-dispersion problem, for any $\epsilon > 0$, which is an improvement over the best known approximation factor $4\sqrt{3}$ [Amano, K. and Nakano, S. I., An approximation algorithm for the 2-dispersion problem, IEICE Transactions on Information and Systems, Vol. 103(3), pp. 506-508, 2020]. Next, we develop a common framework for designing an approximation algorithm for the Euclidean dispersion problem. With this common framework, we improve the approximation factor to $2\sqrt{3}$ for the 2-dispersion problem in $\mathbb{R}^2$. Using the same framework, we propose a polynomial time algorithm, which returns an optimal solution for the 2-dispersion problem when points are placed on a line. Moreover, to show the effectiveness of the framework, we also propose a 2-factor approximation algorithm for the 1-dispersion problem in $\mathbb{R}^2$.

1 Introduction

The facility location problem is one of the extensively studied optimization problems. Here, we are given a set of locations on which facilities can be placed and a positive integer k, and the goal is to place k facilities on those locations so that a specific objective is satisfied. For example, the objective is to place these facilities such that their closeness is undesirable. Often, this closeness measured as a function of the distances between a pair of facilities. We refer to such facility location problem as a dispersion problem. More specifically, we wish to minimize the interference between the placed facilities. The most studied dispersion problem is the *max-min dispersion problem.*

In the *max-min dispersion problem*, we are given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n locations, the non-negative distances between each pair of locations $p, q \in P$, and a positive integer k ($k \leq n$). Here, k refers to the number of facilities to be opened and distances are assumed to be symmetric. The objective is to find a k size subset $S \subseteq P$ of locations such that $cost(S) = \min\{d(p, q) \mid p, q \in S\}$ is maximized, where $d(p, q)$ denotes the distance between p and q. This problem is known as 1-dispersion problem in the literature. In this article, we consider a variant of the max-min dispersion problem. We refer to it as a 2-dispersion problem. Now, we define 2-dispersion problem as follows:

2-dispersion problem: *Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$. For each point $p \in P$ and $S \subseteq P$, we define $cost_2(p, S)$ as the sum of Euclidean distance from p to the first closest point in $S \setminus \{p\}$ and the second closest point in $S \setminus \{p\}$. We also define $cost_2(S) = \min_{p \in S}\{cost_2(p, S)\}$ for each $S \subseteq P$. In the 2-dispersion problem, a set P of n points in $\mathbb{R}^2$ and a positive integer $k \in [3, n]$ are given. The objective is to find a subset $S \subseteq P$ of size k such that $cost_2(S)$ is maximized.*

We find an immense number of applications for the dispersion problem in the real world. The situation in which we want to open chain stores in a community has generated our interest in the dispersion issue. In order to eliminate/prevent self-competition, we need to open stores far away from each other. Another situation in which the issue of dispersion occurs is installing hazardous structures, such as nuclear power plants and oil tanks. These facilities need to be dispersed to the fullest degree possible so that an accident at one of the facilities would not affect others. The dispersion problem also has its application in information retrieval where we need to find a small subset of data with some desired variety from an extensive data set such that a small subset is a reasonable sample to overview the large data set.

2 Related Work

In 1977, Shier [13] studied the k-center problem and the max-min dispersion problem on a tree network.

[*]Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, `pawan.mishra@iitg.ac.in`

[†]Department of Mathematics, Indian Institute of Technology Guwahati, `gkd@iitg.ac.in`

Shier showed that the max-min dispersion problem and $(k-1)$-center on a tree network are dual and established an equivalence between the max-min dispersion problem and $(k-1)$-center problem. In 1981, Chandrasekaran and Daughety [7] studied the max-min dispersion problem on a tree network. In 1982, Chandrasekaran and Tamir [8] also studied the max-min dispersion problem and k-center problem on a tree network, where set of locations is a finite subset. They also established an equivalence between the max-min dispersion problem and $(k-1)$-center problem on a tree network. So, using k-center algorithm on a tree (proposed in [10]), a linear time algorithm for the max-min dispersion problem on a tree can be devised. The max-min dispersion problem is NP-hard even when the distance function satisfies triangle inequality [9]. Wang and Kuo [15] introduced the geometric version of the max-min dispersion problem. They consider the problem in the d-dimensional space where the distance between two points is Euclidean. They proposed a dynamic programming that solves the problem for $d = 1$ in $O(kn)$ time. They also proved that the problem is NP-hard for $d = 2$. In [16], White studied the max-min dispersion problem and proposed a 3-factor approximation result. In 1991, Tamir [14] studied the max-min dispersion problem on a graph, where continuum set of points on the edges are considered as locations. Tamir showed that for a continuum set of locations on a graph, the max-min dispersion problem can not be approximated within a factor of $\frac{3}{2}$ unless $P = NP$. In [14], a heuristic is proposed that produces a 2-factor approximation result for the max-min dispersion problem on a graph. Later in 1994, Ravi et al. [12] studied the max-min dispersion problem on a complete graph, where each edge is associated with a non-negative weight (distance). They independently analyzed the same heuristic proposed in [14] (for the max-min dispersion problem on a complete graph), and showed that the same heuristic produces a 2-approximation result for the complete graph. Furthermore, they also demonstrated that unless $P = NP$, the max-min dispersion problem on complete graph cannot be approximated within a factor of 2 even if the distance function satisfies the triangle inequality.

Recently, Akagi et al. [1] established a relationship between the max-min dispersion problem and the maximum independent set problem, and using it they proposed an exact algorithm for the problem. The running time of the exact algorithm is $O(n^{wk/3} \log n)$, where $w < 2.373$. They also studied two special cases, namely, the input points are on a line and on a circle separately. They proposed a polynomial time exact algorithm for both special cases.

The other popular variant of the dispersion problem is *max-sum k-dispersion problem.* In the max-sum k-dispersion problem, the objective is to maximize the sum of distances between k facilities. Erkut [9] idea's can be adapted to show that the problem is NP-hard. Ravi et al. [12] gave a polynomial time exact algorithm when the points are placed on a line. They also proposed a 4-factor approximation algorithm if the distance function satisfies the triangle inequality. In [12], they also proposed a $(1.571 + \epsilon)$-factor approximation algorithm for 2-dimensional Euclidean space, where $\epsilon > 0$. In [5] and [11], the approximation factor of 4 was improved to 2. One can see [4] and [6] for other variations of the dispersion problems. In comparison with max-min dispersion (1-dispersion) problem, a handful amount of research has been done in 2-dispersion problem. Recently, in 2018, Amano and Nakano [2] proposed a greedy algorithm, which produces an 8- factor approximation result. In 2020, [3] they analyzed the same greedy algorithm proposed in [2] and proposed a $4\sqrt{3}(\approx 6.92)$-factor approximation result.

2.1 Our Contribution

In this article, we first consider the 2-dispersion problem in $\mathbb{R}^2$ and propose a simple polynomial time $(2\sqrt{3} + \epsilon)$-factor approximation algorithm for any $\epsilon > 0$. The best known result in the literature is $4\sqrt{3}$-factor approximation algorithm [3]. We also develop a common framework that improves the approximation factor to $2\sqrt{3}$ for the same problem. We present a polynomial time optimal algorithm for 2-dispersion problem if the input points lies on a line. Though a 2-factor approximation algorithm available in the literature for the 1-dispersion problem in $\mathbb{R}^2$ [12], but to show the effectiveness of the proposed common framework, we propose a 2-factor approximation algorithm for the 1-dispersion problem using the developed framework.

2.2 Organization of the Paper

The remainder of the paper is organized as follows. In Section 3, we propose a $(2\sqrt{3} + \epsilon)$-factor approximation algorithm for the 2-dispersion problem in $\mathbb{R}^2$, where $\epsilon > 0$. In Section 4, we propose a common framework for the dispersion problem. Using the framework, followed by $2\sqrt{3}$-factor approximation result for the 2-dispersion problem in $\mathbb{R}^2$, a polynomial time optimal algorithm for the 2-dispersion problem on a line and 2-factor approximation result for the 1-dispersion problem in $\mathbb{R}^2$. Finally, we conclude the paper in Section 5.

3 $(2\sqrt{3} + \epsilon)$-Factor Approximation Algorithm

In this section, we propose a $(2\sqrt{3} + \epsilon)$-factor approximation algorithm for the 2-dispersion problem, for any $\epsilon > 0$. Actually, we consider the same algorithm proposed in [3], but using different argument, we will show that for any $\epsilon > 0$, it is a $(2\sqrt{3}+\epsilon)$-factor approximation algorithm. For completeness of this article, we prefer to discuss the algorithm briefly as follows. Let $I = (P, k)$ be an arbitrary instance of the 2-dispersion problem, where $P = \{p_1, p_2, \dots p_n\}$ is the set of n points in $\mathbb{R}^2$ and $k \in [3, n]$ is a positive integer. Initially, we choose a subset $S_3 \subseteq S$ of size 3 such that $cost_2(S_3)$ is maximized. Next, we add one point $p \in P$ into S_3 to construct S_4, i.e., $S_4 = S_3 \cup \{p\}$, such that $cost_2(S_4)$ is maximized and continues this process up to the construction of S_k. The pseudo code of the algorithm is described in Algorithm 1.

Algorithm 1 GreedyDispersionAlgorithm(P, k)

Input: A set $P = \{p_1, p_2, \dots, p_n\}$ of n points, and a positive integer $k(3 \leq k \leq n)$.
Output: A subset $S_k \subseteq P$ of size k.

```
1: Compute $\{p_{i_1}, p_{i_2}, p_{i_3}\} \subseteq P$ such that $cost_2(S_3)$ is maximized.
2: $S_3 = \{p_{i_1}, p_{i_2}, p_{i_3}\}$
3: for $(j = 4, 5 \dots, k)$ do
4:     Let $p \in P \setminus S_{j-1}$ such that $cost_2(S_{j-1} \cup \{p\})$ is maximized.
5:     $S_j \leftarrow S_{j-1} \cup \{p\}$
6: end for
7: return $(S_k)$
```

Theorem 1 *For any $\epsilon > 0$, Algorithm 1 produces $(2\sqrt{3}+\epsilon)$-factor approximation result in polynomial time.*

Proof. Let $I = (P, k)$ be an arbitrary input instance of the 2-dispersion problem, where $P = \{p_1, p_2, \dots, p_n\}$ is the set of n points and k is a positive integer. Let S_k and OPT be the output of Algorithm 1 and optimum solution, respectively, for the instance I. To prove the theorem, we have to show that $\frac{cost_2(OPT)}{cost_2(S_k)} \leq 2\sqrt{3}+\epsilon$. Here we use induction to show that $cost_2(S_i) \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$ for each $i = 3, 4, \dots, k$. Since S_3 is an optimum solution for 3 points (see line number 1 of Algorithm 1), therefore $cost_2(S_3) \geq cost_2(OPT) \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$ holds. Now, assume that the condition holds for each i such that $3 \leq i < k$. We will prove that the condition holds for $(i+1)$ too.

Now, we define a disk D_i centered at each $p_i \in P$ as follows: $D_i = \{p_\ell \in \mathbb{R}^2 | d(p_i, p_\ell) \leq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}\}$. Let D^* be a set of disks corresponding to each point in OPT. A point p_j is *contained* in D_i, if $d(p_i, p_j) \leq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$.

Lemma 2 *For any point $p_i \in P$, $|D_i \cap OPT| \leq 2$.*

Proof. On the contrary, assume that three points $p_a, p_b, p_c \in D_i \cap OPT$. Let $S = \{p_a, p_b, p_c\}$. Without loss of generality assume that $cost_2(p_a, S) \leq cost_2(p_b, S)$ and $cost_2(p_a, S) \leq cost_2(p_c, S)$, i.e., $d(p_a, p_b) + d(p_a, p_c) \leq d(p_a, p_b) + d(p_b, p_c)$ and $d(p_a, p_b) + d(p_a, p_c) \leq d(p_a, p_c) + d(p_b, p_c)$, which leads to $d(p_a, p_b) \leq d(p_b, p_c)$ and $d(p_a, p_c) \leq d(p_b, p_c)$. We notice that maximizing $d(p_a, p_b) + d(p_a, p_c)$ results in minimizing $d(p_b, p_c)$(see Figure 1). The minimum value of $d(p_b, p_c)$ is $\sqrt{3}\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$ as both $d(p_a, p_b)$ and $d(p_a, p_c)$ is less than equal to $d(p_b, p_c)$. Therefore, from the packing argument inside a disk, $d(p_a, p_b) + d(p_a, p_c)$ is maximum if p_a, p_b, p_c are on an equilateral triangle and on the boundary of the disk D_i. Then, $cost_2(S) \leq d(p_a, p_b) + d(p_a, p_c) \leq \sqrt{3}\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon} + \sqrt{3}\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon} = 2\sqrt{3}\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon} < cost_2(OPT)$, which leads to a contradiction to the optimal value $cost_2(OPT)$. Therefore for any $p_i \in P$, D_i contains at most two points from the optimal set OPT.

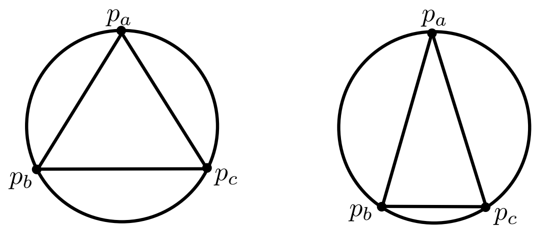


Figure 1: Points $p_a, p_b, p_c \in D_i$

□

Lemma 3 *For some $p_j \in OPT$, $|D_j \cap S_i| < 2$.*

Proof. On the contrary, assume that there does not exist any $j \in [1, k]$ such that $|D_j \cap S_i| < 2$. Let $D^* = \{D_i \mid p_i \in OPT\}$. Construct a bipartite graph $H(S_i \cup D^*, \mathcal{E})$ as follows: (i) S_i and $D^* = \{D_1, D_2, \dots, D_k\}$ are two partite vertex sets, and (ii) for $u'_i \in S_i$, $(u'_i, D_j) \in \mathcal{E}$ if and only if u'_i is contained in D_j. According to assumption, each disk D_j contains at least 2 points from S_i. Therefore, the total degree of the vertices in D^* in H is at least $2k$. Note that $|D^*| = k$. On the other hand, the total degree of the vertices in S_i in H is at most $2 \times |S_i|$ (see Lemma 2). Since $|S_i| < k$ (based on the assumption of the induction hypothesis), the total degree of the vertices in S_i in H is less than $2k$, which leads to a contradiction that the total degree of the vertices in D^* in H is at least $2k$. Thus, there exist at least one $p_j \in OPT$ such that $|D_j \cap S_i| < 2$. □

Without loss of generality, assume that disk $D_j \in D^*$ has at most one point from the set S_i. Suppose D_j contains only one point of the set S_i, then the distance of p_j to the second closest point in S_i is greater than $\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$(see Figure 2). Also, from triangle inequality $d(p_i, p_j) + d(p_i, p_\ell) > \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$ for each point $p_\ell \in S_i$. So, we can add the point $p_j \in OPT$ to the set S_i to construct set S_{i+1}. Here, $S_{i+1} = S_i \cup \{p_j\}$. Therefore, the cost of $S_{i+1} \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$.

Now, assume that D_j does not contain any point from the set S_i, then the distance of the point $p_j \in OPT$ to any point of S_i is greater than $\frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$. By adding the point p_j in the set S_i, we construct the set S_{i+1}, which leads to the $cost_2(S_{i+1}) \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$. Since our algorithm chooses a point (see line number 4 of Algorithm 1) that maximizes $cost_2(S_{i+1})$, therefore algorithm will always choose a point in the iteration $i+1$ such that $cost_2(S_{i+1}) \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$.

By the help of Lemma 2 and Lemma 3, we can conclude that the $cost_2(S_{i+1}) \geq \frac{cost_2(OPT)}{2\sqrt{3}+\epsilon}$ and thus condition holds for $(i+1)$ too.

Therefore, for any $\epsilon > 0$, Algorithm 1 produces $(2\sqrt{3}+\epsilon)$-factor approximation result in polynomial time. □

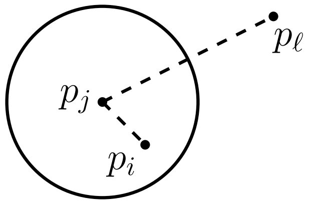


Figure 2: Points $p_j, p_i \in D_j$ and p_ℓ outside the disk D_j

4 An Algorithm for the Dispersion Problem

In this section, we propose an algorithm for the dispersion problem. It is a common algorithm for 1-dispersion, 2-dispersion problem in $\mathbb{R}^2$ and 1-dispersion/2-dispersion problem in $\mathbb{R}$. Input of the algorithm are (i) a set $P = \{p_1, p_2, \ldots p_n\}$ of n points, (ii) an integer $\gamma(= 1$or $2)$ for the γ-dispersion problem, and (iii) an integer $k(\gamma+1 \leq k \leq n)$. In the first line of the algorithm, we set the value of a constant λ. If $\gamma = 2$ and points are in $\mathbb{R}^2$ (resp. $\mathbb{R}$), then we set $\lambda = 2\sqrt{3}$ (resp. $\lambda = 1$), and if $\gamma = 1$ and points are in $\mathbb{R}^2$, then we set $\lambda = 2$. We prove that the algorithm is λ-factor approximation algorithm. We use $S_i(\subseteq P)$ to denote a set of size i. We start algorithm with $S_{\gamma+1} \subseteq P$ containing $\gamma+1$ points as a solution set. Next, iteratively we add one by one point from P into the solution set to get a final solution set, i.e., if we have a solution set S_i of size i, then we add one more point into S_i to get solution set S_{i+1} of size $i+1$. Let $\alpha = cost_\gamma(S_i)$. Now, we add a point from $P \setminus S_i$ into S_i to get S_{i+1} such that $cost_\gamma(S_{i+1}) \geq \frac{\alpha}{\lambda}$. We stop this iterative method if we have S_k or no more point addition is possible. We repeat the above process for each distinct $S_{\gamma+1} \subseteq P$ and report the solution for which the γ-dispersion cost value is maximum.

Algorithm 2 Dispersion Algorithm(P, k, γ)

```
Input : A set $P$ of $n$ points, a positive integer $\gamma$ and an integer $k$ such that $\gamma+1 \leq k \leq n$.
Output: A subset $S_k \subseteq P$ such that $|S_k| = k$ and $\beta = cost_\gamma(S_k)$.
1: If $\gamma = 2(\gamma = 1)$, then $\lambda \leftarrow 2\sqrt{3}$ ($\lambda \leftarrow 2$), and if points are on a line then $\lambda \leftarrow 1$ .
2: $\beta \leftarrow 0$ // Initially, $cost_\gamma(S_k) = 0$
3: for each subset $S_{\gamma+1} \subseteq P$ consisting of $\gamma+1$ points do
4:     Set $\alpha \leftarrow cost_\gamma(S_{\gamma+1})$
5:     Set $\rho \leftarrow \alpha/\lambda$
6:     if $\rho > \beta$ then
7:         $flag \leftarrow 1$, $i \leftarrow \gamma+1$
8:         while $i < k$ and $flag \neq 0$ do
9:             $flag \leftarrow 0$
10:            choose a point $p \in P \setminus S_i$ (if possible) such that $cost_\gamma(S_i \cup \{p\}) \geq \rho$ and $cost_\gamma(p, S_i) = \min_{q \in P \setminus S_i} cost_\gamma(q, S_i)$.
11:            if such point $p$ exists in step 10 then
12:                $S_{i+1} \leftarrow S_i \cup \{p\}$
13:                $i \leftarrow i+1$, $flag \leftarrow 1$
14:            end if
15:        end while
16:        if $i = k$ then
17:            $S_k \leftarrow S_i$ and $\beta \leftarrow \rho$
18:        end if
19:    end if
20: end for
21: return $(S_k, \beta)$
```

4.1 $2\sqrt{3}$-Factor Approximation Result for the 2-Dispersion Problem

Let $S^* \subseteq P = \{p_1, p_2, \ldots, p_n\}$ be an optimal solution for a given instance (P, k) of the 2-dispersion problem and $S_k \subseteq P$ be a solution returned by greedy Algorithm 2 for the given instance, provided $\gamma = 1$ as an additional input. A point $s_o^* \in S^*$ is said to be a solution point if $cost_2(S^*)$ is defined by s_o^*, i.e., $cost_2(S^*) = d(s_o^*, s_r^*) + d(s_o^*, s_t^*)$ such that (i) $s_r^*, s_t^* \in S^*$, and (ii) s_r^* and s_t^* are the first and second closest points of s_o^* in S^*, respectively. We call s_r^*, s_t^* as supporting points. Let $\alpha = cost_2(S^*)$. In this problem, the value of λ is $2\sqrt{3}$ (line number 1 of Algorithm 2).

Lemma 4 *The triangle formed by three points s_o^*, s_r^* and s_t^* does not contain any point in $S^* \setminus \{s_o^*, s_r^*, s_t^*\}$, where s_o^* is the solution point, and s_r^*, s_t^* are supporting points.*

Proof. Suppose there exist a point $s_m^* \in S^*$ inside the triangle formed by s_o^*, s_r^* and s_t^*. Now, if $d(s_o^*, s_r^*) \geq d(s_o^*, s_t^*)$ then $d(s_o^*, s_t^*) + d(s_o^*, s_m^*) < d(s_o^*, s_r^*) + d(s_o^*, s_t^*)$ which contradict the optimality of $cost_2(S^*)$. A similar argument will also work for $d(s_o^*, s_r^*) < d(s_o^*, s_t^*)$. □

In this problem, $\rho = \frac{\alpha}{\lambda} = \frac{cost_2(S^*)}{2\sqrt{3}}$. We define a disk D_i centered at $p_i \in P$ as follows: $D_i = \{p_j \in \mathbb{R}^2 | d(p_i, p_j) \leq \rho\}$. Let $D = \{D_i \mid p_i \in P\}$. Let D^* be the subsets of D corresponding to disks centered at points in S^*. A point p_j is *properly contained* in D_i, if $d(p_i, p_j) < \rho$, whereas if $d(p_i, p_j) \leq \rho$, then we say that point p_j is *contained* in D_i.

Lemma 5 *For any point $p \in P$, if $D^p = \{q \in \mathbb{R}^2 \mid d(p, q) \leq \rho\}$ then D^p properly contains at most two points of the optimal set S^*.*

Proof. On the contrary, assume that three points $p_a, p_b, p_c \in S^*$ such that p_a, p_b, p_c are properly contained in D^p. Using the similar arguments discussed in the proof of Lemma 2, $cost_2(\{p_a, p_b, p_c\})$ is maximum if p_a, p_b, p_c are on equilateral triangle inside D^p. Therefore, $d(p_a, p_b) = d(p_a, p_c) = d(p_b, p_c)$. Now, $cost_2(\{p_a, p_b, p_c\}) = d(p_a, p_b) + d(p_a, p_c) < \sqrt{3}\rho + \sqrt{3}\rho = 2\sqrt{3}\rho = cost_2(S^*)$. Therefore, $p_a, p_b, p_c \in S^*$ and $cost_2(\{p_a, p_b, p_c\}) < cost_2(S^*)$ leads to a contradiction. Thus, the lemma.

□

Lemma 6 *For any three points $\{p_a, p_b, p_c\} \in S^*$, there does not exist any point $s \in \mathbb{R}^2$ such that s is properly contained in $D_a \cap D_b \cap D_c$.*

Proof. On the contrary, assume that s is properly contained in $D_a \cap D_b \cap D_c$. This implies $d(p_a, s) < \rho$, $d(p_b, s) < \rho$ and $d(p_c, s) < \rho$. Therefore, the disk $D^s = \{q \in \mathbb{R}^2 \mid d(s, q) \leq \rho\}$ properly contains three points p_a, p_b and p_c, which is a contradiction to Lemma 5. Thus, the lemma.

□

Corollary 7 *For any point $p \in P$, if $D' \subseteq D^*$ is the subset of disks that contains p, then $|D'| \leq 3$ and p lies on the boundary of each disk in D'.*

Proof. Follows from Lemma 6. □

Corollary 8 *For any point $p \in P$, if $D'' \subseteq D^*$ is the subset of disks that properly contains point p, then $|D''| \leq 2$.*

Proof. Follows from Lemma 6 and Corollary 7. □

Lemma 9 *Let $S \subseteq P$ be a set of points such that $|S| < k$. If $cost_2(S) \geq \rho$, then there exists at least one disk $D_j \in D^* = \{D_1, D_2, \ldots, D_k\}$ that properly contains at most one point from the set S.*

Proof. On the contrary, assume that each $D_j \in D^*$ properly contains at least two points from the set S. Construct a bipartite graph $G(S \cup D^*, \mathcal{E})$ as follows: (i) S and D^* are two partite vertex sets, and (ii) for $u \in S$, $(u, D_j) \in \mathcal{E}$ if and only if u is properly contained in D_j. According to assumption, each disk D_j contains at least 2 points from the set S. Therefore, the total degree of the vertices in D^* in G is at least $2k$. Note that $|D^*| = k$. On the other hand, the total degree of the vertices in S in G is at most $2 \times |S|$ (see Corollary 8). Since $|S| < k$, the total degree of the vertices in S in G is less than $2k$, which leads to a contradiction that the total degree of the vertices in D^* in G is at least $2k$. Thus, there exist at least one disk $D_j \in D^*$ such that the disk D_j properly contains at most one point from the set S.

□

Theorem 10 *Algorithm 2 produces a $2\sqrt{3}$-factor approximation result for the 2-dispersion problem in $\mathbb{R}^2$.*

Proof. Since it is a 2-dispersion problem, so $\gamma = 2$ and set $\lambda = 2\sqrt{3}$ in line number 1 of Algorithm 2. Now, assume $\alpha = cost_2(S^*)$ and $\rho = \frac{\alpha}{\gamma} = \frac{cost_2(S^*)}{2\sqrt{3}}$, where S^* is an optimum solution. Here, we show that Algorithm 2 returns a solution set S_k of size k such that $cost_2(S_k) \geq \rho = \frac{cost_2(S^*)}{2\sqrt{3}}$. More precisely, we show that Algorithm 2 returns a solution S_k of size k such that $cost_2(S_k) \geq \frac{cost_2(S^*)}{2\sqrt{3}}$ and $S_k \supseteq \{s^*_o, s^*_r, s^*_t\}$, where s^*_o is the solution point and s^*_r and s^*_t are supporting points, i.e., $cost_2(S^*) = d(s^*_o, s^*_r) + d(s^*_o, s^*_t)$. Let $S^*_3 = \{s^*_o, s^*_r, s^*_t\}$. Now, consider the case when $S_3 = \{s^*_o, s^*_r, s^*_t\}$ in line number 3 of Algorithm 2. Our objective is to show that if $S_3 = \{s^*_o, s^*_r, s^*_t\}$ in line number 3 of Algorithm 2, then it computes a solution S_k of size k such that $cost_2(S_k) \geq \frac{cost_2(S^*)}{2\sqrt{3}}$. Note that any other solution returned by Algorithm 2 has a 2-dispersion cost better than $\frac{cost_2(S^*)}{2\sqrt{3}}$. Therefore, it is sufficient to prove that if $S_3 = \{s^*_o, s^*_r, s^*_t\}$ in line number 3 of Algorithm 2, then the size of S_k (updated) in line number 17 of Algorithm 2 is k as every time Algorithm 2 added a point (see line number 12) into the set with the property that 2-dispersion cost of the updated set is greater than or equal to $\frac{cost_2(S^*)}{2\sqrt{3}}$. Therefore, we consider $S_3 = \{s^*_o, s^*_r, s^*_t\}$ in line number 3 of Algorithm 2.

We use induction to establish the condition $cost_2(S_i) \geq \rho$ for each $i = 3, 4, \ldots k$. Since $S_3 = S^*_3$, therefore $cost_2(S_3) = cost_2(S^*_3) = \alpha > \rho$ holds. Now, assume that the condition $cost_2(S_i) \geq \rho$ holds for each i such that $3 \leq i < k$. We will prove that the condition $cost_2(S_{i+1}) \geq \rho$ holds for $(i+1)$ too.

Let D^* be the set of disks centered at the points in S^* such that the radius of each disk is ρ. Since $i < k$ and $S_i \subseteq P$ with condition $cost_2(S_i) \geq \rho$, there exist at least one disk, say $D_j \in D^*$ that properly contains at most one point in S_i (see Lemma 9). We will show that $cost_2(S_{i+1}) = cost_2(S_i \cup \{p_j\}) \geq \rho$, where p_j is the center of the disk D_j. Suppose, D_j contains only one point $p_x \in S_i$, then p_x is the first closest point of p_j in the set S_i. Now, by Corollary 7 and by Lemma 9, we claim the second closest point p_ℓ of p_j in the set S_i may lie (i) on the boundary of the disk D_j (see Figure 3(a)) or (ii) outside of the disk D_j(see Figure 3(b)).

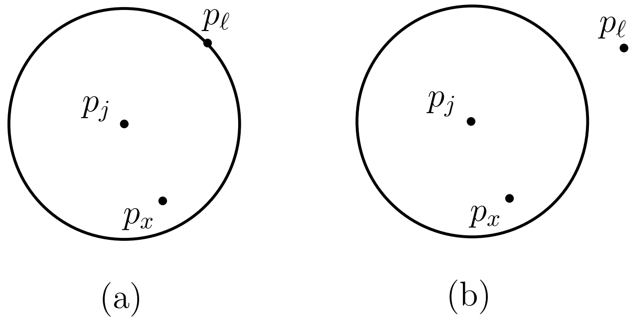


Figure 3: (a) p_ℓ lies on the boundary of the disk D_j and (b) p_ℓ lies outside of the disk D_j

Since $d(p_j, p_\ell) \geq \rho$ for both the above mentioned cases, therefore $cost_2(p_j, S_i) \geq \rho$. Also, from triangle inequality $d(p_x, p_j) + d(p_x, p_\ell) \geq d(p_j, p_\ell) \geq \rho$ for each point $p_\ell \in S_i$. So, we can add the point p_j to the set S_i to construct set S_{i+1}. Here, $S_{i+1} = S_i \cup \{p_j\}$. Therefore, $cost_2(S_{i+1}) \geq \rho$.

Now, if D_j does not properly contain any point from the set S_i, then the distance of p_j to any point of the set S_i is greater than or equal to ρ. Since there exists at least one point $p_j \in P \setminus S_i$ such that $cost_2(S_{i+1}) = cost_2(S_i \cup \{p_j\}) \geq \rho$, therefore Algorithm 2 will always choose a point (see line number 10 of Algorithm 2) in the iteration $i+1$ such that $cost_2(S_{i+1}) \geq \rho$.

So, we can conclude that $cost_2(S_{i+1}) \geq \rho$ and thus condition holds for $(i+1)$ too.

Therefore, Algorithm 2 produces a set S_k of size k such that $cost_2(S_k) \geq \rho$. Since $\rho \geq \frac{cost_2(S^*)}{2\sqrt{3}}$, Algorithm 2 produces $2\sqrt{3}$-factor approximation result for the 2-dispersion problem.

$\square$

4.2 2-Dispersion Problem on a Line

In this section, we discuss the 2-dispersion problem on a line L. Let the point set $P = \{p_1, p_2, \dots p_n\}$ be on a horizontal line arranged from left to right. Let $S_k \subseteq P$ be a solution returned by Algorithm 2 and $S^* \subseteq P$ be an optimal solution. Note that, the value of γ is 2 and the value of λ (line number 1 of Algorithm 2) is 1 in this problem. Let s_o^* be a solution point and s_r^*, s_t^* be supporting points, i.e., $cost_2(S^*) = d(s_o^*, s_r^*) + d(s_o^*, s_t^*)$. Let $S_3^* = \{s_o^*, s_r^*, s_t^*\}$. We show that if $S_3 = S_3^*$ in line number 3 of Algorithm 2, then $cost_2(S_3) = cost_2(S^*)$. Let $S^* = \{s_1^*, s_2^*, \dots s_k^*\}$ are arranged from left to right.

Lemma 11 *Let S^* be an optimal solution. If s_o^* is the solution point and s_r^*, s_t^* are supporting points, then both points s_r^* and s_t^* cannot be on the same side on the line L with respect to s_o^* and three points s_r^*, s_o^*, s_t^* are consecutive on the line L in S^*.*

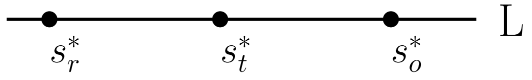


Figure 4: s_r^* and s_t^* on left side of s_o^*

Proof. On the contrary, assume that both s_r^* and s_t^* are on the left side of s_o^*, and s_t^* lies between s_r^* and s_o^* (see Figure 4). Now, $d(s_t^*, s_o^*) + d(s_t^*, s_r^*) < d(s_o^*, s_t^*) + d(s_o^*, s_r^*)$ which leads to a contradiction that s_o^* is a solution point, i.e., $cost_2(S^*) = d(s_o^*, s_r^*) + d(s_o^*, s_t^*)$. Now, suppose s_r^*, s_o^*, s_t^* are not consecutive in S^*. Let s^* be the point in S^* such that either $s^* \in (s_r^*, s_o^*)$ or $s^* \in (s_o^*, s_t^*)$. If $s^* \in (s_r^*, s_o^*)$, then $d(s_o^*, s_r^*) + d(s_o^* + s_t^*) > d(s_o^*, s^*) + d(s_o^* + s_t^*)$, which leads to a contradiction that s_r^* is a supporting point. Similarly, we can show that if $s^* \in (s_o^*, s_t^*)$, then s_t^* is not a supporting point. Thus, s_r^*, s_o^*, s_t^* are consecutive points on the line L in S^*.

$\square$

Lemma 11 says that if s_o^* is a solution point, then s_{o-1}^* and s_{o+1}^* are supporting points as $s_1^*, s_2^*, \dots, s_k^*$ are arranged from left to right.

Lemma 12 *Let $S_3 = \{s_o^*, s_r^*, s_t^*\}$ and $\alpha = cost_2(S_3)$. Now, if $S_i = S_{i-1} \cup \{p_i\}$ constructed in line number 12 of Algorithm 2, then $cost_2(S_i) = \alpha$.*

Proof. We use induction to prove $cost_2(S_i) = \alpha$ for $i = 4, 5 \dots, k$.

Base Case: Consider the set $S_4 = S_3 \cup \{p_4\}$ constructed in line number 12 of Algorithm 2. If s_o^* is a solution points, and s_r^*, s_t^* are supporting points and $cost_2(p_4, S_3) \geq \alpha$, therefore $p_4 \notin [s_{o-1}^*, s_{o+1}^*]$ (otherwise one of s_{o-1}^* and s_{o+1}^* will not be supporting point). This implies p_4 either lies in $[p_1, s_{o-1}^*)$ or $(s_{o+1}^*, p_n]$. Assume $p_4 \in (s_{o+1}^*, p_n]$. In Algorithm 2, we choose p_4 such that $cost_2(p_4, S_4) \geq \alpha$ (see line number 10 of Algorithm 2) and $cost_2(p_4, S_4) = \min_{q \in P \setminus S_3} cost_2(q, S_4)$. Therefore, $p_4 \in (s_{o+1}^*, s_{o+2}^*]$. Let $S_4' = \{s_1^*, s_2^*, \dots, s_{o-2}^*\} \cup S_4 \cup \{s_{o+3}^*, s_{o+4}^*, \dots, s_k^*\}$. Suppose $p_4 = s_{o+2}^*$ and we know that $S_3 = S_3^*$ then $S_4' = S^*$. So, $cost_2(S_4') = cost_2(S^*) = \alpha$. This implies $cost_2(S_4) = \alpha$. Now assume that $p_4 \in (s_{o+1}^*, s_{o+2}^*)$, then also we will show that $cost_2(S_4') = \alpha$. We calculate $cost_2(p_4, S_4') = d(p_4, s_{o+1}^*) + d(p_4, s_{o+3}^*) = d(s_{o+2}^*, s_{o+1}^*) + d(s_{o+2}^*, s_{o+3}^*) \geq \alpha$ and $cost_2(s_{o+3}^*, S_4') = d(s_{o+3}^*, p_4) + d(s_{o+3}^*, s_{o+4}^*) \geq d(s_{o+3}^*, s_{o+2}^*) + d(s_{o+3}^*, s_{o+4}^*) \geq \alpha$ (see Figure 5). Thus if $p_4 \in (s_{o+1}^*, s_{o+2}^*)$, then $cost_2(S_4') = \alpha$. Therefore, if

$k \geq 4$, then p_4 exists and $cost_2(S_4) = \alpha$. Similarly, we can prove that if $p_4 \in [p_1, s^*_{o-1})$, then $cost_2(S'_4) = \alpha$, where $S'_4 = \{s^*_1, s^*_2, \ldots, s^*_{o-3}\} \cup S_4 \cup \{s^*_{o+2}, s^*_{o+4}, \ldots, s^*_k\}$. In this case also p_4 exists and $cost_2(S_4) = \alpha$.

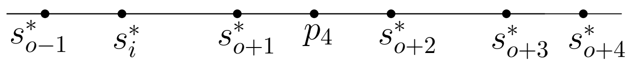


Figure 5: Snippet of S'_4

Now, assume that $S_i = S_{i-1} \cup \{p_i\}$ for $i < k$ such that $cost_2(S'_i) = \alpha$ and $cost_2(S_i) = \alpha$ where $S'_i = \{s^*_1, s^*_2, \ldots, s^*_u\} \cup S_i \cup \{s^*_v, s^*_{v+1}, \ldots, s^*_k\}$. If $p_i \in (s^*_o, p_n]$, then $s^*_{v-1} \in S^*$ is the left most point in the right of p_i and $u \geq k - (i + k - v + 1) = v - i - 1$ with each point of S_i are on the right side of s^*_u (see Figure 6(a)) and if $p_i \in [p_1, s^*_o)$, then $s^*_{u+1} \in S^*$ is the right most point in the left of p_i where $v \geq u + i + 1$ (see Figure 6(b)).

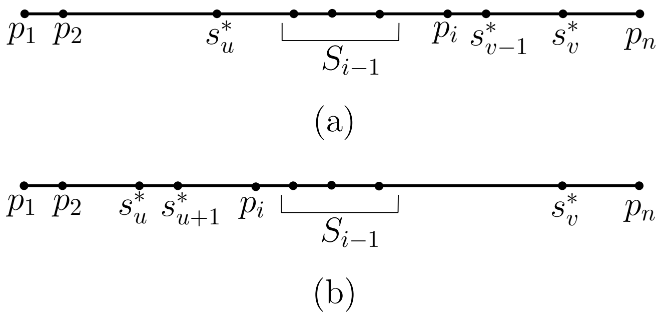


Figure 6: Placement of set $S_{i-1} \cup \{p_i\}$.

We prove that $cost_2(S_{i+1}) = \alpha$, where $S_{i+1} = S_i \cup \{p_i\}$. It follows from the fact that size of S_i is less than k, and the set $\{s^*_1, s^*_2, \ldots, s^*_{s_u}\} \cup \{s^*_v, s^*_{v+1}, \ldots, s^*_k\} \neq \phi$ and the similar arguments discussed in the base case. □

Lemma 13 *The running time of Algorithm 2 on line is $O(n^4)$.*

Proof. Since it is a 2-dispersion problem on a line, so algorithm starts by setting $\lambda = 1$ in line number 1 of Algorithm 2, and then compute solution set for each distinct $S_3 \subseteq P$ independently. Now, for each S_3, algorithm selects a point iteratively based on greedy choice (see line number 10 of Algorithm 2). Now, for choosing remaining $(k-3)$ points, the total amortize time taken by the algorithm is $O(n)$. So, the overall time complexity of Algorithm 2 on line consisting of n points is $O(n^4)$.

□

Theorem 14 *Algorithm 2 produces an optimal solution for the 2-dispersion problem on a line in polynomial time.*

Proof. Follows from Lemma 12 that $cost_2(S_i) = \alpha = cost_2(S^*_3)$ for $3 \leq i \leq k$, where $S_3 = \{s^*_o, s^*_r, s^*_t\}$. Therefore, $cost_2(S_k) = \alpha$. Also, Lemma 13 says that Algorithm 2 computes S_k in polynomial time. Thus, the theorem. □

4.3 1-Dispersion Problem in $\mathbb{R}^2$

In this section, we show the effectiveness of Algorithm 2 by showing 2-factor approximation result for the 1-dispersion problem in $\mathbb{R}^2$. Here, we set $\gamma = 1$ as input along with input P and k. We also set $\lambda = 2$ in line number 1 of the algorithm 2.

Let S^* be an optimal solution for a given instance (P, k) of 1-dispersion problem and $S_k \subseteq P$ be a solution returned by our greedy Algorithm 2 provided $\gamma = 1$ as an additional input. Let $s^*_o \in S^*$ a solution point, i.e., $cost_1(S^*) = d(s^*_o, s^*_r)$ such that s^*_r is the closest points of s^*_o in S^*. We call s^*_r as supporting point. Let $\alpha = d(s^*_o, s^*_r)$ and $\rho = \frac{\alpha}{2}$.

We define a disk D_i centered at $p_i \in P$ as follows: $D_i = \{p_j \in \mathbb{R}^2 | d(p_i, p_j) \leq \rho\}$. Let $D = \{D_i \mid p_i \in P\}$. Let D^* be the subsets of D corresponding to disks centered at points in S^*. If $d(p_i, p_j) < \rho$, then we say that p_j is *properly contained* in D_i and if $d(p_i, p_j) \leq \rho$, then we say that p_j is *contained* in D_i.

Lemma 15 *For any point $s \in P$, if $D^s = \{q \in \mathbb{R}^2 \mid d(s, q) \leq \rho\}$ then D^s properly contains at most one point of the optimal set S^*.*

Proof. On the contrary, assume that $p_a, p_b \in S^*$ such that p_a, p_b are properly contained in D^s. If two points p_a and p_b are properly contained in D^s, then $d(p_a, p_b) \leq d(p_a, s) + d(p_b, s) < \frac{\alpha}{2} + \frac{\alpha}{2} = \alpha$, which leads to a contradiction to the optimality of S^*. Thus, the lemma. □

Lemma 16 *For any two points $p_a, p_b \in S^*$, there does not exist any point $s \in \mathbb{R}^2$ that is properly contained in $D_a \cap D_b$.*

Proof. On the contrary, assume that s is properly contained in $D_a \cap D_b$. This implies $d(p_a, s) < \frac{\alpha}{2}$ and $d(p_b, s) < \frac{\alpha}{2}$. Therefore, the disk $D^s = \{q \in \mathbb{R}^2 \mid d(s, q) \leq \rho\}$ properly contains two points p_a and p_b, which is a contradiction to Lemma 15. Thus, the lemma.

□

Corollary 17 *For any point $s \in P$, if $D' \subseteq D^*$ is the set of disks that contain s, then $|D'| \leq 2$ and s lies on the boundary of both the disk in D'.*

Proof. Follows from Lemma 16. □

Corollary 18 *For any point $s \in P$, if $D'' \subseteq D^*$ be a subset of disks that properly contains point s, then $|D''| \leq 1$.*

Proof. Follows from Lemma 16 and Corollary 17. □

Lemma 19 *Let $M \subseteq P$ be a set of points such that $|M| < k$. If $cost_2(M) \geq \frac{\alpha}{2}$, then there exists at least one disk $D_j \in D^* = \{D_1, D_2, \ldots, D_k\}$ that does not properly contain any point from the set M.*

Proof. On the contrary, assume that each $D_j \in D^*$ properly contains at least one point from the set M. Construct a bipartite graph $G(M \cup D^*, \mathcal{E})$ as follows: (i) M and D^* are two partite vertex sets, and (ii) for $u \in M$, $(u, D_j) \in \mathcal{E}$ if and only if u is properly contained in D_j. According to assumption, each disk D_j contains at least 1 points from the set M. Therefore, the total degree of the vertices in D^* in G is at least k. Note that $|D^*| = k$. On the other hand, the total degree of the vertices in M in G is at most $|M|$ (see Corollary 18). Since $|M| < k$, the total degree of the vertices in M in G is less than k, which leads to a contradiction that the total degree of the vertices in D^* in G is at least k. Thus, there exist at least one disk $D_j \in D^*$ such that D_j does not properly contain any point from the set M. □

Theorem 20 *Algorithm 2 produces a 2-factor approximation result for the 1-dispersion problem in $\mathbb{R}^2$.*

Proof. Since it is a 1-dispersion problem, so $\gamma = 1$ and set $\lambda = 2$ in line number 1 of the algorithm. Now, assume $\alpha = cost_1(S^*)$ and $\rho = \frac{\alpha}{\lambda} = \frac{cost_1(S^*)}{2}$, where S^* is the optimum solution. Here, we show that Algorithm 2 returns a solution set S_k of size k such that $cost_1(S_k) \geq \rho$. More precisely, we show that Algorithm 2 returns a solution S_k of size k such that $cost_1(S_k) \geq \rho$ and $S_k \supseteq \{s_o^*, s_r^*\}$, where s_o^* is the solution point and s_r^* is the supporting point. Our objective is to show that if $S_2 = \{s_o^*, s_r^*\}$ in line number 3 of Algorithm 2, then it computes a solution S_k of size k such that $cost_1(S_k) \geq \rho$. Note that any other solution returned by Algorithm 2 has a 1-dispersion cost better than $\frac{cost_1(S^*)}{2}$. Therefore, it is sufficient to prove that if $S_2 = \{s_o^*, s_r^*\}$ in line number 3 of Algorithm 2, then the size of S_k (updated) in line number 17 of Algorithm 2 is k as every time Algorithm 2 added a point (see line number 12) into the set with the property that 1-dispersion cost of the updated set is greater than or equal to $\rho = \frac{cost_1(S^*)}{2}$. Therefore, we consider $S_2 = \{s_o^*, s_r^*\}$ in line number 3 of Algorithm 2.

We use induction to establish the condition $cost_1(S_i) \geq \rho$ for each $i = 3, 4, \ldots k$. Since $S_2 = S_2^*$, therefore $cost_1(S_2) = cost_1(S_2^*) = \alpha$ holds. Now, assume that the condition $cost_1(S_i) \geq \rho$ holds for each i such that $3 \leq i < k$. We will prove that the condition $cost_1(S_{i+1}) \geq \rho$ holds for $(i+1)$ too.

Let D^* be the set of disks centered at the points in S^* such that the radius of each disk be $\rho = \frac{\alpha}{2}$. Since $i < k$ and $S_i \subseteq P$ with condition $cost_1(S_i) \geq \rho = \frac{\alpha}{2}$, there exist at least one disk, say $D_j \in D^*$ that does not contain any point from the set S_i (see Lemma 19). We will show that $cost_1(S_{i+1}) = cost_1(S_i \cup \{p_j\}) \geq \rho$, where p_j is the center of the disk D_j.

Now, if D_j does not properly contain any point from the set S_i, then the closest point of $p_j \in S^*$ may lie on the boundary of the disk D_j (by Corollary 17) or outside the disk D_j (by Lemma 19). In both the cases, distance of p_j to any point of the set S_i is greater than or equal to ρ (see Figure 7). Since there exists at least one point $p_j \in P \backslash S_i$ such that $cost_1(S_{i+1}) = cost_1(S_i \cup \{p_j\}) \geq \rho$, therefore Algorithm 2 will always choose a point (see line number 10 of Algorithm 2) in the iteration $i+1$ such that $cost_1(S_{i+1}) \geq \rho$.

So, we can conclude that $cost_1(S_{i+1}) \geq \rho$ and thus condition holds for $(i+1)$ too.

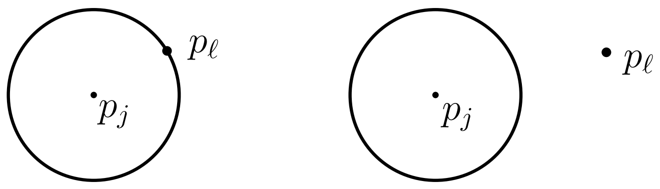


Figure 7: Second closest point of p_j lies on boundary of D_j or outside of D_j.

Therefore, Algorithm 2 produces a 2-factor approximation result for the 1-dispersion problem in $\mathbb{R}^2$. □

5 Conclusion

In this article, we proposed a $(2\sqrt{3} + \epsilon)$-factor approximation algorithm for the 2-dispersion problem in $\mathbb{R}^2$, where $\epsilon > 0$. The best known approximation factor available in the literature is $4\sqrt{3}$ [3]. Next, we proposed a common framework for the dispersion problem. Using the framework, we further improved the approximation factor to $2\sqrt{3}$ for the 2-dispersion problem in $\mathbb{R}^2$. We studied the 2-dispersion problem on a line and proposed a polynomial time algorithm that returns an optimal solution using the developed framework. Note that, for the 2-dispersion problem on a line, one can propose a polynomial time algorithm that returns an optimal value in relatively low time complexity, but to show the adaptability and flexibility of our proposed framework, we presented an algorithm for the same problem using the developed framework. We also proposed a 2-factor approximation algorithm for the 1-dispersion problem using the proposed common framework to show the effectiveness of the framework.

Acknowledgment

We thank Prof. Arie Tamir (Tel Aviv University, Tel Aviv, Israel) for his useful suggestions on the literature of the problem.

References

[1] Akagi, Toshihiro and Araki, Tetsuya and Horiyama, Takashi and Nakano, Shin-ichi and Okamoto, Yoshio and Otachi, Yota and Saitoh, Toshiki and Uehara, Ryuhei and Uno, Takeaki and Wasa, Kunihiro. Exact algorithms for the max-min dispersion problem. *International Workshop on Frontiers in Algorithmics*, pp. 263–272, 2018.

[2] Amano, Kazuyuki and Nakano, Shin-Ichi. Away from Rivals. *CCCG*, pp. 68–71, 2018.

[3] Amano, Kazuyuki and Nakano, Shin-Ichi. An Approximation Algorithm for the 2-Dispersion Problem. *IEICE Transactions on Information and Systems*, 103(3): 506–508, 2020.

[4] Baur, Christoph and Fekete, Sándor P. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

[5] Birnbaum, Benjamin and Goldman, Kenneth J. An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs. *Algorithmica*, 55(1):42–59, 2009.

[6] Chandra, Barun and Halldórsson, Magnús M. Approximation algorithms for dispersion problems. *Journal of algorithms*, 38(2):438–465, 2001.

[7] Chandrasekaran, R and Daughety, Andrew. Location on tree networks: p-centre and n-dispersion problems. *Mathematics of Operations Research*, 6(1):50–57, 1981.

[8] Chandrasekaran, R and Tamir, Arie. Polynomially bounded algorithms for locating p-centers on a tree. *Mathematical Programming*, 22(1): 304–315, 1982

[9] Erkut, Erhan. The discrete p-dispersion problem. *European Journal of Operational Research*, 46(1):48–60, 1990.

[10] Frederickson, Greg N. Optimal algorithms for tree partitioning. *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 168–177, 1991.

[11] Hassin, Refael and Rubinstein, Shlomi and Tamir, Arie. Approximation algorithms for maximum dispersion. *Operations research letters*, 21(3):133–137, 1997.

[12] Ravi, Sekharipuram S and Rosenkrantz, Daniel J and Tayi, Giri Kumar. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[13] Shier, Douglas R. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11(3):243–252, 1977.

[14] Tamir, Arie. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4(4): 550-567, 1991.

[15] Wang, DW and Kuo, Yue-Sun. A study on two geometric location problems. *Information processing letters*, 28(6):281–286, 1988.

[16] White, Douglas J. The maximal-dispersion problem. *IMA Journal of Mathematics Applied in Business and Industry*, 3(2):131–140, 1991.

The Discrete Median and Center Line Segment Problems in the Plane

Ovidiu Daescu* Ka Yaw Teo†

Abstract

Let P be a set of n points in the plane. The *discrete median line segment* of P is the line segment with both its endpoints in P such that the sum of the distances from P to the line segment is minimized. Similarly, the *discrete center line segment* of P is the line segment bounded by two points of P such that the maximum of the distances from P to the line segment is minimized. We present exact algorithms for computing the discrete median and center line segments of P. Our algorithms run in $O(n^2)$ time and use $O(n^2)$ space.

1 Introduction

In this paper, we consider the following two problems.

Discrete median line segment problem *Given a set P of n points in $\mathbb{R}^2$, locate a line segment bounded by two points of P such that the* ***sum*** *of the Euclidean distances from P to the line segment is minimized.*

Discrete center line segment problem *Given a set P of n points in $\mathbb{R}^2$, locate a line segment bounded by two points of P such that the* ***maximum*** *of the Euclidean distances from P to the line segment is minimized.*

It appears, as far as the authors are aware, that the problems defined above, albeit interesting and geometric in nature, have been seemingly overlooked in facility location theory. The proposed problems are closely related to a class of "discrete" problems in facility location theory, where the goal is to select one point (or several) from a given set of points P so as to minimize an objective function that is distance-dependent with respect to P.

In general, there are two types of problems in facility location theory depending on the objective function used – i) center (minimax) and ii) median (minsum). In regards to the discrete point facility location problems aforementioned, the discrete center problem asks to locate a point in P that minimizes the maximum of the distances between the points of P and the located point. This is analogous to finding the smallest disk centered at a point of P and containing P. The discrete center problem can be solved in $O(n \log n)$ time using the farthest-neighbor Voronoi diagram of P [5, Chapter 7]. The discrete median, which is commonly known as the medoid, is a point in P that has the minimal sum of distances to P. One can find the medoid of P by simply computing all $O(n^2)$ pairwise distances. However, it has been argued that no exact algorithm exists for solving the medoid problem in $o(n^2)$ time [8].

2 Our results

We begin in Section 3 by addressing the discrete median line segment problem. First, we solve the problem in $O(n^2 \log n)$ time by using enumeration enhanced with logarithmic query-time data structures (Section 3.1). We then improve the time complexity of our algorithm to $O(n^2)$ by reducing the query time of our data structures to amortized $O(1)$ (Section 3.2).

In the process of deriving our solution to the discrete median line segment problem, we develop efficient data structures supporting half-plane distance-sum queries (see Subproblems 1 and 2 for detailed definitions), which happen to be more general than required for solving our problem.

We proceed to solve the discrete center line segment problem in Section 4 along similar lines. We obtain an $O(n^2 \log n)$-time algorithm for the problem based on the query data structures proposed in [1, 4] (Section 4.1). We then follow by deriving an $O(n^2)$-time algorithm, which requires a data preprocessing approach different from that in the prior algorithm (Section 4.2).

In Section 5, we show that, against intuition, the discrete median or center line segment of P does not necessarily have an endpoint at a vertex on the convex hull of P. This rules out any algorithm whose time complexity depends on the number of vertices on the convex hull of P, which is typically smaller than n for random (and many practical) sets of points.

We end the paper with some brief concluding remarks in Section 6.

3 Discrete median line segment

One can find the discrete median line segment in $O(n^3)$ time using a brute-force method – namely, by enumerating all $\binom{n}{2}$ candidate line segments (i.e., distinct pairs

*Department of Computer Science, University of Texas at Dallas, ovidiu.daescu@utdallas.edu

†Department of Computer Science, University of Texas at Dallas, ka.teo@utdallas.edu

of points in P) and computing the corresponding sum of $n-1$ distances for each candidate line segment.

3.1 An $O(n^2 \log n)$-time algorithm

In this section, we derive an $O(n^2 \log n)$-time algorithm for the discrete median line segment problem. The idea is to preprocess P into some data structures of logarithmic query time for use in computing the sum of distances for each candidate line segment. We begin by addressing the required query data structures, which are derived from solving the following two subproblems (refer to Figure 1).

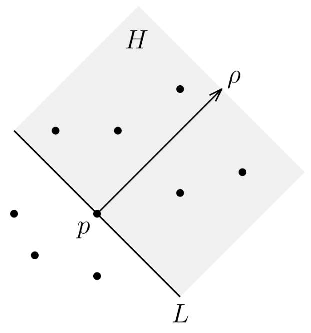


Figure 1: Illustration for Subproblems 1 and 2.

Subproblem 1 *Given a set P of n points in the plane, let H be a query half-plane bounded by a line L containing a point $p \in P$. Preprocess P so that, for a point $p \in P$ and a half-plane H given at query time, one can efficiently report the sum of the distances from $P \cap H$ to p.*

Subproblem 2 *Given a set P of n points in the plane, let H be a query half-plane bounded by a line L containing a point $p \in P$. Let ρ be the ray emanating from p, perpendicular to L, and contained in H. Preprocess P so that, for a point $p \in P$ and a half-plane H given at query time, one can efficiently report the sum of the orthogonal distances from $P \cap H$ to ρ.*

Here we give a brief description of the preprocessing procedure for solving the subproblems. For each point $p \in P$, we i) sort the points of $P \setminus \{p\}$ around p in $O(n \log n)$ time, ii) define a sequence of $O(n)$ intervals in the sorted order such that $P \cap H$ remains constant within each interval, iii) enumerate the intervals in the sorted order so that it takes $O(1)$ time to evaluate the sum of distances in each interval, and iv) store the distance sums computed for the intervals in an $O(\log n)$-query time data structure. The details of the solutions to Subproblems 1 and 2 are presented in the following two subsections.

Subproblem 1

Recall that L denotes a line passing through a point $p \in P$. Without loss of generality, let H be one of the two half-planes bounded by L (the other half-plane can be handled similarly due to symmetry). As line L rotates around point p, a point $q \in P \setminus \{p\}$ may enter and leave H. These point-entering and -leaving events can be determined in $O(1)$ time each by computing the line passing through p and each point $q \in P \setminus \{p\}$. Since a point $q \in P \setminus \{p\}$ can enter and leave H at most once during a full rotation of line L around point p, the total number of point-entering and -leaving events is bounded by $2n-2$. These events can be sorted in counterclockwise order, according to the slopes of their corresponding lines L, in $O(n \log n)$ time.

Let q_i and q_j be the points in $P \setminus \{p\}$ associated with any two consecutive events in the sorted order. Note that, within the event interval bounded by q_i and q_j, the subset of points $P \setminus \{p\}$ contained in H remains constant, and so does the sum of their distances to p. The said distance sum for the set of points $(P \cap H) \setminus \{p\}$ for each of these event intervals can be computed as follows. We begin with the event interval with the smallest slope of L (in the sorted order), for which we determine the set of points $(P \cap H) \setminus \{p\}$ and calculate the sum of their distances to p in $O(n)$ time. For each subsequent event interval in the sorted order, the distance sum can be computed in constant time using that of the preceding event interval – that is, by adding to (or subtracting from) the current distance sum the distance between p and the point entering (or leaving) H. Therefore, for a given point $p \in P$, the total time for the computation of the distance sums for all event intervals is bounded by $O(n)$.

A data structure $\mathcal{D}_1$ can then be built as follows. For each point $p \in P$, we construct a simple logarithmic query-time data structure to store the distance sums computed for the event intervals. Given a query point $p \in P$ and a query half-plane H defined by a line L passing through p, we can use data structure $\mathcal{D}_1$ to look up the event interval that contains the slope of line L, and report the sum of the distances from the points of $P \cap H$ to p in $O(\log n)$ time.

Lemma 1 *In Subproblem 1, a set P of n points can be preprocessed in $O(n^2 \log n)$ time into a data structure of size $O(n^2)$ so that, given a query point $p \in P$ and a query half-plane H, one can report the sum of the distances from $P \cap H$ to p in $O(\log n)$ time.*

Subproblem 2

The line containing ray ρ can be described by $y = mx + c$. Let x_i and y_i be the x- and y-coordinates, respectively, of a point $p_i \in P$, where $1 \leq i \leq n$. Define $T^+ = \{i : y_i - mx_i > c\}$ and $T^- = \{i : y_i - mx_i < c\}$.

The sum of the distances from $P \cap H$ to ρ can then be expressed as

$$\begin{aligned}
&\sum_{i \in T^+ \cup T^-} \frac{|y_i - mx_i - c|}{(m^2+1)^{1/2}} \\
&= (m^2+1)^{-1/2} \left[\sum_{i \in T^+} (y_i - mx_i - c) \right. \\
&\quad \left. - \sum_{i \in T^-} (y_i - mx_i - c) \right] \\
&= (m^2+1)^{-1/2} \left[\sum_{i \in T^+} y_i - m \sum_{i \in T^+} x_i - \sum_{i \in T^+} c \right. \\
&\quad \left. - \sum_{i \in T^-} y_i + m \sum_{i \in T^-} x_i + \sum_{i \in T^-} c \right] \qquad (1)
\end{aligned}$$

In order to solve Subproblem 2, we will follow a strategy similar to that in Subproblem 1. Assume, without loss of generality, that H is one of the two half-plane bounded by L (the other case can be handled symmetrically). Observe that, when L rotates counterclockwise around p, a point $q \in P \setminus \{p\}$ may enter H, leave H, or move from set T^+ to T^-. Each of these point-entering and -leaving events can be determined in constant time by computing i) the line L passing through p and each point $q \in P \setminus \{p\}$, and ii) the ray ρ emanating from p and passing through each point $q \in P \setminus \{p\}$. The total number of events is at most $3n-3$. These events can be sorted in counterclockwise order, according to the slopes of their respective lines L, in $O(n \log n)$ time.

Notice that, within any interval delimited by two consecutive events in the sorted order, sets T^+ and T^- remain constant. We will determine, for each event interval, the following set Q of values (from Equation 1)

$$\sum_{i \in T^+} x_i, \sum_{i \in T^-} x_i, \sum_{i \in T^+} y_i, \sum_{i \in T^-} y_i, \sum_{i \in T^+} 1, \sum_{i \in T^-} 1$$

by keeping track of T^+ and T^- as we sweep the event intervals in the counterclockwise order. We begin with the first event interval in the ordering by determining its corresponding sets T^+ and T^-, as well as the respective set of values Q, in $O(n)$ time. For each subsequent event interval, we can determine the corresponding set of values Q by updating those of the preceding event interval in $O(1)$ time. Hence, for a point $p \in P$, it takes $O(n)$ time to compute Q for all event intervals.

We can then built a query data structure $\mathcal{D}_2$ as follows. For each point $p \in P$, we construct a linear-size data structure with a logarithmic query time to store the sets of values Q computed for the event intervals. Given a query point $p \in P$ and a query half-plane H bounded by a line L passing through p (along with the calculated values of parameters m and c associated with the line supporting ray ρ), by employing data structure $\mathcal{D}_2$, we retrieve in $O(\log n)$ time the corresponding set of values Q, from which we can calculate the sum of the distances from the points of $P \cap H$ to ρ using Equation 1 in constant time.

Lemma 2 *In Subproblem 2, a set P of n points can be preprocessed in $O(n^2 \log n)$ time into a data structure of size $O(n^2)$ so that, given a query point $p \in P$ and a query half-plane H, one can report the sum of the orthogonal distances from $P \cap H$ to ρ in $O(\log n)$ time.*

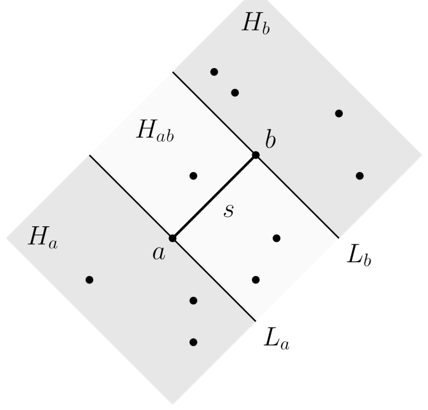


Figure 2: Computing the distances from P to s.

We are now in position to describe an $O(n^2 \log n)$-time algorithm for the discrete median line segment problem. Let a and b denote the two endpoints of a candidate line segment s, where $a, b \in P$ and $a \neq b$ (see Figure 2). Let L_a (resp. L_b) be the line passing through a (resp. b) and perpendicular to s. Let H_a (resp. H_b) be the half-plane bounded by L_a (resp. L_b) and not containing s. Define $H_{ab} = \mathbb{R}^2 \setminus (H_a \cup H_b)$. Recall that $\mathcal{D}_1$ and $\mathcal{D}_2$ denote the query data structures derived from Subproblems 1 and 2, respectively. We can compute the sum of the distances from $P \setminus \{a, b\}$ to s using $\mathcal{D}_1$ and $\mathcal{D}_2$ as follows.

We denote by i) Σ_a the sum of the distances from $P \cap H_a$ to s, ii) Σ_b the sum of the distances from $P \cap H_b$ to s, and iii) Σ_{ab} the sum of the distances from $P \cap H_{ab}$ to s. We can determine Σ_a by querying $\mathcal{D}_1$ using a and H_a as the query inputs. Similarly, Σ_b can be found using $\mathcal{D}_1$ with b and H_b as inputs for the query. In order to calculate Σ_{ab}, we execute the following queries. Define $H'_a = \mathbb{R}^2 \setminus H_a$. Let Υ_a be the distance sum reported by querying $\mathcal{D}_2$ using a and H'_a as the query inputs. Likewise, let Υ_b be the distance sum retrieved from a query of $\mathcal{D}_2$ with b and H_b as the query inputs. Then, $\Sigma_{ab} = \Upsilon_a - \Upsilon_b$. Finally, the sum of the distances from $P \setminus \{a, b\}$ to s is given by $\Sigma_a + \Sigma_b + \Sigma_{ab}$. Overall, we perform four $O(\log n)$-time queries for each of the $O(n^2)$ candidate line segments. We hence reach the following conclusion.

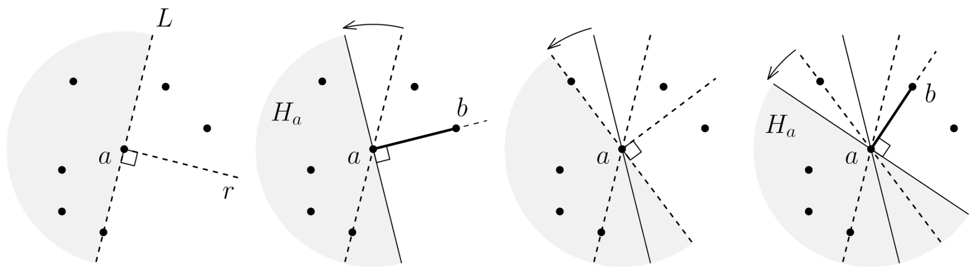


Figure 3: Keeping track of query half-plane H_a in Subproblem 1 as line L and its perpendicular ray r rotate around a point $a \in P$ in a counterclockwise direction (illustrations from left to right).

Theorem 3 *The discrete median line segment problem can be solved in $O(n^2 \log n)$ time using $O(n^2)$ space.*

Remark 1 *It is worth noting that the logarithmic query-time data structures given in Lemmas 1 and 2 are in fact more general than necessary to solve our problem. Namely, using those data structures, the distance sum can be reported for* any *query half-plane defined by a line passing through a point $p \in P$, whereas each query half-plane in our problem is always associated with a line passing through a pair of points in P. This observation would become one of the keys in improving the time bound of our algorithm to $O(n^2)$, as detailed in the next section.*

3.2 An $O(n^2)$-time algorithm

We can reduce the running time of the algorithm above by an $O(\log n)$ factor as follows.

Using the point-line duality transform, point set P can be mapped into a set of n lines, whose arrangement can be constructed in $O(n^2)$ time using $O(n^2)$ space [2, 6]. For any point $p \in P$ in the primal plane, let p^* denote its corresponding line in the dual plane. Observe that the line containing p and a point $q \in P \setminus \{p\}$ in the primal plane corresponds to the intersection point of lines p^* and q^* in the dual plane. By the properties of the duality transform, the ordering of slopes of the lines passing through p and every point $q \in P \setminus \{p\}$ in the primal plane is equivalent to the ordering of x-coordinates of the intersections between line p^* and each line q^* in the dual plane. Using this trait, for any point $p \in P$, the set of points $P \setminus \{p\}$ can be obtained in sorted order around p by simply traversing the vertices along the dual line of point p in $O(n)$ time. Notice that these sorted points correspond to the endpoints of the sorted event intervals in Subproblems 1 and 2. Consequently, the preprocessing time of the query data structures given in Lemmas 1 and 2 is reduced to $O(n^2)$.

In addition, we note that each query half-plane in our problem is associated with a pair of points in P. We can predetermine and index each of these query half-planes, and use the static indices to create perfect hash tables [3, Chapter 11] for constant-time look-ups (in the worst case) in place of the current logarithmic-time query data structures $\mathcal{D}_1$ and $\mathcal{D}_2$.

Specifically, for a candidate line segment s bounded by a pair of points a and b in P, the respective set of query half-planes consists of H_a, H_b, and $H'_a = \mathbb{R} \setminus H_a$ (Figure 2). In the ensuing discussion, we will give an argument for query half-plane H_a, and an analogues argument can be made about query half-planes H_b and H'_a.

Recall that a query half-plane H_a is uniquely defined by a and b (i.e., a pair of points in P) as being i) delimited by the line passing through a and orthogonal to s, and ii) not containing b. For a given point $a \in P$, let L denote any line passing through a, and let H be one of the two half-planes bounded by L. In Subproblem 1, a point-entering or leaving event is indicated by line L passing through a point $q \in P \setminus \{a\}$. Let r be the ray emanating from a, perpendicular to L, and not contained in H (see Figure 3). Notice that a query half-plane H_a is equivalent to H when ray r passes through a point $b \in P \setminus \{a\}$. Thus, for a candidate point $a \in P$, there exists a set of $n-1$ query half-planes H_a, due to $n-1$ other possible points $b \in P \setminus \{a\}$. For each point $a \in P$, by computing the ray r passing through each point $b \in P \setminus \{a\}$, we can obtain in advance the set of all $O(n)$ possible query half-planes H_a according to the counterclockwise ordering of points $b \in P \setminus \{a\}$ around point a.

In Subproblem 1, when we perform the sweep procedure for a point $p = a \in P$ by rotating line L counterclockwise around a, in addition to processing each point-entering or leaving event due to line L passing through a point of $P \setminus \{a\}$, we record the distance sum for each query half-planes H_a as ray r passes through a point $b \in P \setminus \{a\}$. For each point $a \in P$, we create a linear-size (perfect) hash table that maps each query half-plane H_a to its corresponding sum of distances. As a result, we can perform each query in $O(1)$ time.

Theorem 4 *The discrete median line segment problem can be solved in $O(n^2)$ time using $O(n^2)$ space.*

4 Discrete center line segment

Naively, we can find the discrete center line segment for point set P in $O(n^3)$ time (with a linear space usage) by simply enumerating all $O(n^2)$ candidate line segments and determining the farthest point of P from each candidate line segment in $O(n)$ time.

Before proceeding any further, we denote by i) $[P]$ the convex hull of P, ii) $bd[P]$ the boundary of $[P]$, and iii) $int[P]$ the interior of $[P]$.

4.1 An $O(n^2 \log n)$-time algorithm

We can derive an $O(n^2 \log n)$-time algorithm for the discrete center line segment problem based on the previous results given by Aronov et al. [1] and Daescu et al. [4].

According to [4, Theorem 6], after a preprocessing of P that takes $O(n \log n)$ time and space, the farthest point of P from a query line segment can be determined in $O(\log^2 n)$ time. As a direct consequence, we can compute the discrete center line segment for P in $O(n^2 \log^2 n)$ time.

As shown in [4, Theorems 5 and 6], the time and space complexities of finding the farthest point of P from a query line segment are dominated by those of solving the following key subproblem (half-plane farthest point queries):

Preprocess $[P]$ into a data structure so that, given a point q and a directed line L, report the point $p \in [P]$ farthest from q among those located to the left of L.

So, if we use the $O(n \log^3 n)$-space data structure proposed by Aronov et al. [1, Corollary 11] for answering half-plane farthest point queries in $O(\log n)$ time, we can then obtain the following result.

Theorem 5 *The discrete center line segment problem can be solved in $O(n^2 \log n)$ time using $O(n \log^3 n)$ space.*

4.2 An $O(n^2)$-time algorithm

In this section, we derive an $O(n^2)$-time algorithm for computing the discrete center line segment of P.

Let s denote a line segment bounded by two points of P. We begin with the following observation.

Observation 1 *The point of P farthest from s is a vertex of $[P]$.*

Proof. Since s is a line segment with both its endpoints in P, s must lie within $[P]$; that is, $s \subset bd[P] \cup int[P]$. Note that, for any point $p \in bd[P] \cup int[P]$, the point of P farthest from p is a vertex of $[P]$. Thus, the farthest point of P from any point on s is a vertex of $[P]$. We conclude that the point of P farthest from s must be a vertex of $[P]$. □

Let α be the point of P farthest from s. Let β be the closest point on s to α. The Euclidean distance between α and s is defined as the distance between α and β.

Let a and b denote the two endpoints of s. Recall that H_a (resp. H_b) is the half-plane i) bounded by the line passing through a (resp. b) and perpendicular to s, and ii) not containing s. In addition, $H_{ab} = \mathbb{R}^2 \setminus (H_a \cup H_b)$.

If $\alpha \in H_a$, then the closest point on s to α is a. Similarly, if $\alpha \in H_b$, then the closest point on s to α is b. If $\alpha \in H_{ab}$, then the closest point on s to α is an interior point of s. Furthermore, if $\alpha \in H_{ab}$, then α is also the point of P farthest from the line containing s.

Based on the observations above, for line segment s, we can find the farthest point of P from s using the following approach. We determine

I. the point of $[P] \cap H_a$ farthest from a,

II. the point of $[P] \cap H_b$ farthest from b, and

III. the point of $[P]$ farthest from the line containing s.

Then, the farthest of the three is the farthest point of P from s. Since there are $O(n^2)$ candidate line segments, in order to obtain an $O(n^2)$-time algorithm, we have to address each of parts I, II, and III above in constant time (on average) for each candidate line segment.

Parts I and II

The subproblem of interest associated with parts I and II can be stated as follows.

Subproblem 3 *Preprocess P into a data structure so that, given a point $p \in P$ and a directed line L passing through p, one can efficiently report the point of P farthest from p among those located to the left of L.*

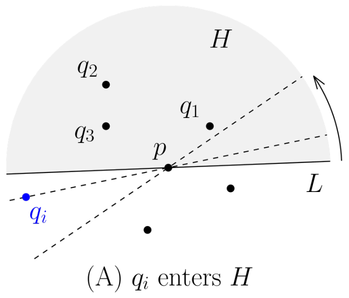
Consider the following approach for solving Subproblem 3.

Let H denote the half-plane to the left of directed line L. As line L rotates around point p, a point $q \in P \setminus \{p\}$ may enter and leave half-plane H. Each of these point-entering and -leaving events can be determined in $O(1)$ time by computing the line passing through p and each point $q \in P \setminus \{p\}$. Given that a point $q \in P \setminus \{p\}$ can enter and leave H only once, there exist $2n - 2$ point-entering and -leaving events. These events can be obtained in counterclockwise order in $O(n)$ time by employing the point-line duality transform – that is, mapping P, using $O(n^2)$ time and space, into an arrangement of n lines, through which the set of points $P \setminus \{p\}$ can be determined in sorted order around p in $O(n)$ time.

Within an interval bounded by any two consecutive events in the sorted order, the subset of points $P \setminus \{p\}$ contained in H remains unchanged, and so does the maximum of the distances from points $(P \cap H) \setminus \{p\}$

to p. As we sweep the event intervals in the sorted (counterclockwise) order, we can keep track of the maximum of the distances from points $(P \cap H) \setminus \{p\}$ to p by maintaining a monotonic double-ended queue Q, whose elements are a subset of $P \setminus \{p\}$, as follows.

For a point $q_i \in P \setminus \{p\}$, let d_i be the distance from q_i to p. Let q_f denote the first element in Q, and q_ℓ be the last element of Q. Upon a point-entering event, in which a point q_i enters half-plane H, if $d_i < d_\ell$, then q_i is added to the back of Q; otherwise, we keep removing the last element of Q until the condition $d_i < d_\ell$ is satisfied, and q_i is appended to the back of Q. For a point-leaving event, where a point q_i leaves half-plane H, if $q_i = q_f$ (i.e., q_i is the first element in Q), then we remove q_f from the front of Q; otherwise, no update is made to Q (see Figure 4 for an illustrative example).



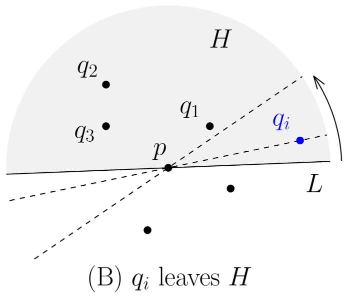


Figure 4: Illustrative example of point-entering and -leaving events in Subproblem 3. Note that $d_1 < d_3 < d_2 < d_i$. (A) Q is updated from (q_2, q_3) to (q_i) upon the entering of q_i into H. (B) Q is updated from (q_i, q_2, q_3) to (q_2, q_3) upon the leaving of q_i from H.

For the first event interval in the sorted order, we simply treat each of the points $(P \cap H) \setminus \{p\}$ as a point entering H, and process the points in counterclockwise order according to the rules above. Note that, within an event interval, Q always contains a subset of $(P \cap H) \setminus \{p\}$ such that the points, from the front to the back of Q, form a monotonic sequence with strictly decreasing distances from p. Thus, the first element of Q always corresponds to the point of $(P \cap H) \setminus \{p\}$ with the farthest distance from p for a given event interval. Since there are $O(n)$ point-entering and -leaving events, and each point of $P \setminus \{p\}$ can only be inserted into or removed from Q at most once, it takes a total of $O(n)$ time to determine the point of $(P \cap H) \setminus \{p\}$ farthest from p in all event intervals.

For each point $p \in P$, we then construct an $O(n)$-size data structure with a logarithmic query time to store the farthest point of $(P \cap H) \setminus \{p\}$ from p for each event interval. Given a point $p \in P$ and a directed line L passing through p, by using the associated query data structure, we can retrieve in $O(\log n)$ time the farthest point of P from p among those situated to the left of L.

Lemma 6 *In Subproblem 3, a set P of n points can be preprocessed using $O(n^2)$ time and space so that, given a point $p \in P$ and a directed line L passing through p, one can report, in $O(\log n)$ time, the point of P farthest from p among those located to the left of L.*

As with the queries required in solving the discrete median line segment problem (Section 3.2), for any point $p \in P$, a query line L passing through p must be perpendicular to the line passing through p and another point $q \in P$. Thus, we can pre-compute and index these $O(n)$ query lines, and use the indices to create a perfect hash map [3, Chapter 11] for $O(1)$-time searches instead of constructing the logarithmic query-time data structure above for each point $p \in P$.

Ergo, with respect to parts I and II of our current approach, it takes $O(n^2)$ time total to find i) the point of $[P] \cap H_a$ farthest from a, and ii) the point of $[P] \cap H_b$ farthest from b, for all $O(n^2)$ candidate line segments.

Part III

First, we note that the convex hull $[P]$ of P can be computed in $O(n \log n)$ time [5, Chapter 11]. Let L be a line passing through any two points of P, and let H denote either of the two half-planes bounded by L. Since the orthogonal distances from the vertices of $[P] \cap H$ to line L are unimodal (as the vertices are traversed in order) [9, Theorem 1], we can find the farthest point of $[P]$ to line L using two binary searches in $O(\log n)$ time. Thus, we can compute, for all lines L in $O(n^2 \log n)$ time total, the farthest point of $[P]$ from line L.

In order to achieve $O(n^2)$ time, we propose the following approach.

We begin by using the point-line duality transform to map P into a set P^* of n lines, whose arrangement can be computed in $O(n^2)$ time [2, 6]. Specifically, a point $p = (x_p, y_p) \in P$ in the primal plane is transformed into a line p^* represented by $y = x_p x - y_p$ in the dual plane. Notice that the x-coordinate x_p of point p in the primal plane is equivalent to the slope of line p^* in the dual plane.

Let V_u and V_ℓ denote the upper and lower envelopes of P^* in the dual plane. Note that V_u and V_ℓ correspond to the lower and upper hulls of $[P]$, respectively, in the primal plane. We can compute V_u and V_ℓ in $O(n \log n)$ time using $O(n)$ space [5, Chapter 11].

For a point $p \in P$, we can obtain the sequence of $n-1$ lines, each of which passes through p and a point $q \in P \setminus \{p\}$, in increasing order of their slopes in $O(n)$ time by simply traversing the vertices along the dual line p^*. We denote the sequence of lines as $S = (L_1, L_2, ..., L_{n-1})$. For each line L_i in the primal plane, we designate its dual point as L_i^*. Note that the slope of line L_i in the primal plane corresponds to the x-coordinate of point L_i^* in dual plane.

In the ensuing description, for conciseness, we present the arguments only for upper envelope V_u (i.e., the lower hull of $[P]$), and the same arguments can be similarly applied to lower envelope V_ℓ (i.e., the upper hull of $[P]$) due to symmetry.

First, we observe the following. For a line $L_i \in S$, let ρ_i be the vertical upward ray emanating from L_i^* in the dual plane. The line containing the edge of V_u intersected by ρ_i corresponds to the farthest point of the lower hull of $[P]$ from L_i in the primal plane.

Consequently, we can find the farthest point in the lower hull of $[P]$ from each line $L_i \in S$ – that is, L_1, L_2, ..., L_{n-1} in increasing order of their slopes – by simply traversing along V_u in the positive x-direction, while keeping track of the edge of V_u intersected by each ρ_i (see Figure 5).

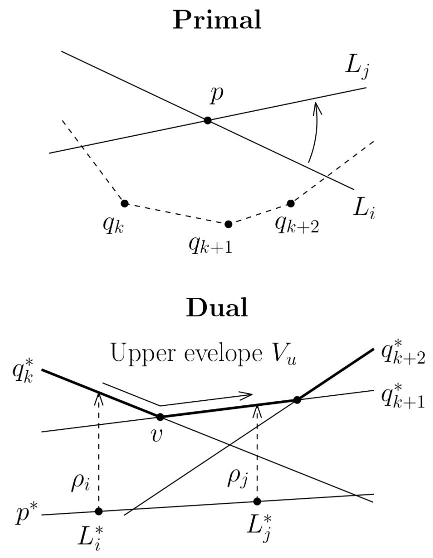


Figure 5: Traversal of upper envelope V_u in part III. The polygonal chain q_k, q_{k+1}, q_{k+2} belongs to the lower hull of $[P]$ in the primal plane. The dual lines of q_k, q_{k+1}, and q_{k+2} are denoted by q_k^*, q_{k+1}^*, and q_{k+2}^*, respectively. The farthest point in the lower hull of $[P]$ from L_i in the primal plane is given by line q_k^* containing the edge of V_u bounded to the right by v in the dual plane.

Specifically, let u and v be any pair of adjacent vertices in V_u, where the x-coordinate of u is smaller than that of v. For every line $L_i \in S$ whose slope lies between the x-coordinates of u and v, the farthest point in the lower hull of $[P]$ from L_i is given by (the line supporting) the edge connecting u and v. Thus, we only have to keep track of each vertex encountered in our traversal of V_u and process the sequence of lines S in order accordingly. That is, upon encountering vertex v in the traversal of V_u, for the contiguous subsequence of lines in S whose slopes fall between the x-coordinates of u and v, we record, as the farthest point in lower hull of $[P]$ from each of those lines, the point dual to the line containing the edge bounded on the right by v.

Note that, for the first line L_1 in S, it takes $O(\log n)$ time (i.e., a binary search on V_u) to locate the edge intersected by ρ_1. From there on, by traversing along V_u (and S in order), for each successive line $L_i \in S$ (in increasing order of slope), where $2 \leq i \leq n-1$, it requires only a constant number of operations to determine the edge of V_u intersected by ρ_i and thus the farthest point in the lower hull of $[P]$ from L_i.

Since S and V_u are bounded by $O(n)$ in size, the entire process above takes $O(n)$ time. In other words, for each point $p \in P$, it takes $O(n)$ time to determine the farthest point of the lower hull of $[P]$ from L_i for all $i \in [1, n-1]$.

The same traversal procedure can be performed on lower envelope V_ℓ to find the farthest point of the upper hull of $[P]$ from each line L_i, where $1 \leq i \leq n-1$.

Recall that L denotes any line containing two points of P. Since $|P| = n$, we conclude that, in $O(n^2)$ time total, we can find the farthest point of $[P]$ from each of the $O(n^2)$ lines L. That is, in part III, it takes a total of $O(n^2)$ time to find the point of $[P]$ farthest from the line containing a candidate line segment s for all $O(n^2)$ candidates.

Finally, based on the results obtained for parts I, II, and III, we arrive at the following conclusion.

Theorem 7 *The discrete center line segment problem can be solved in $O(n^2)$ time using $O(n^2)$ space.*

5 A remark on discrete median and center line segments and convex hull of points

Recall that, for any given set P of points, $[P]$ denotes the convex hull of P.

Observation 2 *The discrete median line segment of P does not **necessarily** have an endpoint at a vertex of $[P]$.*

Proof. It suffices to disprove the statement "if s is a discrete median line segment of P, then s has an endpoint at a vertex of $[P]$" by giving a counterexample. Consider a set of six points $P = \{p_1, p_2, ..., p_6\}$ in the plane with the following coordinates (see Figure 6): $p_1 = (0, 0)$, $p_2 = (1, 0)$, $p_3 = (1, 1)$, $p_4 = (0, 1)$, $p_5 = (x, \frac{1}{2})$, and $p_6 = (1 - x, \frac{1}{2})$.

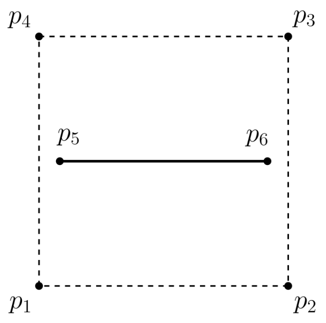


Figure 6: Illustrative example used in proving Observations 2 and 3.

It can be verified, using algebraic geometry, that p_5p_6 is the discrete median line segment of P for any $0 < x < \frac{1}{14}$ (note that, when $\frac{1}{14} < x < \frac{1}{2}$, either p_1p_3 or p_2p_4 is the discrete median line segment of P). Since the vertices of $[P]$ consist of p_1, p_2, p_3, and p_4, the discrete median line segment of P (i.e., p_5p_6) does not have an endpoint at a vertex of $[P]$. □

Observation 3 *The discrete center line segment of P does not **necessarily** have an endpoint at a vertex of $[P]$.*

Proof. Using the same counterexample as in the proof of Observation 2, with $0 < x < \frac{1}{2}$, we can show that the statement "if s is a discrete center line segment of P, then s has an endpoint at a vertex of $[P]$" is false, thus proving Observation 3. □

With respect to solving the discrete median and center line segment problems, Observations 2 and 3 essentially preclude the potential of finding efficient algorithms with a time complexity dependent on the number of vertices on the convex hull of P, which may be significantly smaller than n in practice.

6 Concluding remarks

We have described an $O(n^2)$-time algorithm for computing the discrete median line segment as well as the discrete center line segment for a set P of n points in the plane.

Our $O(n^2)$-time algorithm for solving the discrete median line segment problem matches in time complexity the lower bound of the medoid problem, as well as the fastest known algorithm for finding the median line in $\mathbb{R}^2$ (i.e., the line having the minimal sum of distances from P) [7]. Hence, we conjecture that our algorithm for the discrete median line segment problem is optimal. Nevertheless, we have no reason to believe that our $O(n^2)$ time bound is tight for the discrete center line segment problem.

By allowing each point of P to be associated with a positive weight, we can generalize our problems to those of minimizing the sum and maximum of weighted distances. The algorithms proposed herein can be directly extended to solve the weighted problems with the same time and space bound.

We end our paper with the following open questions. Can we reduce the quadratic space usage of our $O(n^2)$-time algorithms to $O(n)$? Can we obtain efficient (subcubic-time) algorithms for solving the discrete median and center line segment problems in $\mathbb{R}^d$ for $d \geq 3$?

References

[1] B. Aronov, P. Bose, E. D. Demaine, J. Gudmundsson, J. Iacono, S. Langerman, and M. Smid. Data structures for halfplane proximity queries and incremental voronoi diagrams. *Algorithmica*, 80(11):3316–3334, 2018.

[2] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2009.

[4] O. Daescu, N. Mi, C. S. Shin, and A. Wolff. Farthest-point queries with geometric and combinatorial constraints. *Computational Geometry*, 33(3):174–185, 2006.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 2008.

[6] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

[7] N. M. Korneenko and H. Martini. Approximating finite weighted point sets by hyperplanes. In *Scandinavian Workshop on Algorithm Theory*, pages 276–286. Springer, 1990.

[8] J. Newling and F. Fleuret. A sub-quadratic exact medoid algorithm. In *Artificial Intelligence and Statistics*, pages 185–193. PMLR, 2017.

[9] G. T. Toussaint. Complexity, convexity, and unimodality. *International journal of computer & information sciences*, 13(3):197–217, 1984.

Oblivious Median Slope Selection

Thore Thießen[*] Jan Vahrenhold[*]

Abstract

We study the median slope selection problem in the oblivious RAM model. In this model memory accesses have to be independent of the data processed, i. e., an adversary cannot use observed access patterns to derive additional information about the input. We show how to modify the randomized algorithm of Matoušek [26] to obtain an oblivious version with $\mathcal{O}(n \log^2 n)$ expected time for n points in $\mathbb{R}^2$. This complexity matches a theoretical upper bound that can be obtained through general oblivious transformation. In addition, results from a proof-of-concept implementation show that our algorithm is also practically efficient.

1 Introduction

Data collected for statistical analysis is often sensitive in nature. Given the increasing reliance on cloud-based solutions for data processing, there is a demand for data-processing techniques that provide privacy guarantees. One such guarantee is *obliviousness*, i. e., an algorithm's property to have externally observable runtime behavior that is independent of the data being processed. Depending on the runtime behavior observed, oblivious algorithms can be used to perform privacy-preserving computations on externally stored data or mitigate side channel attacks on shared resources [32, 25].

In the oblivious RAM model of computation [13, 14] algorithms need to be oblivious with respect to the memory access patterns; we refer to *memory-access obliviousness* as *obliviousness*. In general this leads to an $\Omega(\log m)$ overhead compared to RAM algorithms when operating on m memory cells [14, 21, 17]. A transformation approach matches this lower bound asymptotically [4], but is known to result in prohibitively large constant runtime overhead.

The median slope, know as the *Theil-Sen estimator*, is a linear point estimator that is robust against outliers [30]. The randomized algorithm of Matoušek [26] computes the median slope of n points in $\mathbb{R}^2$ with expected runtime $\mathcal{O}(n \log n)$ and is fast in practice. We derive an oblivious version of Matoušek's algorithm that is slower by a logarithmic factor — matching the complexity obtainable through general transformation — but still fast in practice.

[*]Westfälische Wilhelms-Universität Münster, Dept. of Computer Science `{t.thiessen,jan.vahrenhold}@uni-muenster.de`

1.1 Median slope selection problem

Median slope selection is a special case of the general *slope selection problem*: Given a set of points P in the plane, the slope selection problem for an integer k is to select a line with k-th smallest slope among all lines through points in P [9]. Formally, given a set of n points $P \subset \mathbb{R}^2$ let $L \coloneqq \{\{p,q\} \subseteq P \mid p_x \neq q_x\}$ be the set of all pairs of points from P with distinct x-coordinates. We use $\ell_{pq} \in L$ to denote the line through points $\{p,q\} \in L$. No line in L is vertical by definition,[1] so the slope $m(\ell_{pq})$ is well-defined for all $\ell_{pq} \in L$. Let k be an integer with $k \in [|L|] \coloneqq \{0, \ldots, |L|-1\}$. The slope selection problem for k then is to select points $\{p,q\} \in L$ such that ℓ_{pq} has a k-th smallest slope in L.

Unless noted otherwise and in line with Matoušek [26] our exposition assumes that the points P are in general position: All x-coordinates of points $\{p,q\} \subseteq P$ are distinct and all lines through different pairs of points have different slopes. For simplicity we also assume that $|L|$ is odd, so that the median slope can be determined by solving the slope selection problem for $k = \frac{|L|-1}{2}$. In Section 3, we discuss how to lift these restrictions.

Matoušek's algorithm approaches the slope selection problem by considering the dual *intersection selection problem* [26]: Each point $p = \langle p_x, p_y \rangle \in P$ can be mapped to dual non-vertical line $\overline{p}\colon x \mapsto (p_x x - p_y)$ and vice versa. Since we have $\overline{p}(x) = \overline{q}(x) = y \iff \overline{\langle x, y \rangle} = \ell_{pq}$, a point in the set $\overline{L}$ of (dual) intersection points with k-th smallest x-coordinate is dual to a line in L with k-th smallest slope [26, 10].

We thus restrict ourselves to finding an intersection of dual lines $\overline{P}$ with k-th smallest x-coordinate. By the above assumption regarding the general position of the points in P, the lines in $\overline{P}$ have distinct slopes and all intersection points have distinct x-coordinates.

1.2 Oblivious RAM model

We work in the *oblivious RAM* (*ORAM*) model [13, 14]. This model is concerned with what can be derived by an adversary observing the *memory access patterns* during the execution of a program. The general requirement is that memory accesses are (data-)*oblivious*, i. e., that the

[1]Cole et al. [9] allow the selection of vertical lines and thus points with identical x-coordinates, but we exclude these as the Theil-Sen estimator is defined for non-vertical lines only.

adversary can learn nothing about the input (or output) from the memory access pattern.

In line with standard assumptions, we assume a probabilistic word RAM with word length w, a constant number of registers in the processing unit and access to $m \leq 2^w$ memory cells with w bits each in the memory unit [17]. The constant number of registers in the processing unit are called *private memory* and do not have to be accessed in an oblivious manner.

Whether a given probabilistic RAM program R operating on inputs X is oblivious depends on the way memory is accessed. Let $\mathbb{A} \coloneqq \{\texttt{read}, \texttt{write}\} \times [m]$ be the set of memory *probes* observable by the adversary. Each probe is identified by the memory operation and the access location $i \in [m]$. The random variable $\mathcal{A}_{\texttt{R}(x)} \colon \Omega \to \mathbb{A}^*$ denotes the *probe sequence* performed by R for an input $x \in X$ where $\Omega \coloneqq \{0,1\}^{l \cdot w}$ is the set of possible random tape contents. The program R is *secure* if no adversary, given inputs $x, x' \in X$ of equal length and a probe sequence $A \in \mathbb{A}^*$, can reliably decide whether A was induced by x or x'. For a program with an output determined by the input this implies that no adversary can decide between given outputs [3].

We operationalize obliviousness by restricting the definition of Chan et al. [8] to perfect security, determined programs, and perfect correctness. The definition also generalizes the allowed dependence of the probe sequence on the length of the input to a general *leakage*; the leakage determines what information the adversary may be able to derive from the memory access patterns.

Definition 1: Oblivious simulation. *Let $f \colon X \to Y$ be a computable function and let* R *be a probabilistic RAM program.* R *obliviously simulates f with regard to leakage* $\texttt{leak} \colon X \to \{0,1\}^*$ *if* R *is* ***correct****, i. e., for all inputs $x \in X$ the equality $\Pr[\texttt{R}(x) = f(x)] = 1$ holds, and if* R *is* ***secure****, i. e., for all inputs $x, x' \in X$ with $\texttt{leak}(x) = \texttt{leak}(x')$ the equality $\sum_{A \in \mathbb{A}^*} \left|\Pr[\mathcal{A}_{\texttt{R}(x)} = A] - \Pr[\mathcal{A}_{\texttt{R}(x')} = A]\right| = 0$ holds.*

The composition of oblivious programs is also oblivious if the sub-procedures invoke each other in an oblivious manner. Here relaxing the leakage allows us to place fewer restrictions on sub-procedures while maintaining obliviousness of the complete program.

For the specific problem in this paper, the algorithm is only allowed to leak the number of given lines, or, for subroutines, the length of each given input array. We will prove the obliviousness of our algorithm by composability, so we will consider the obliviousness of sub-procedures individually. In line with Definition 1 we will show the obliviousness of each procedure in relation to the input. Since we only consider sub-procedures with determined result this implies the obliviousness in relation to the output.

1.3 Related work

There exists a breadth of research on the slope selection problem. Cole et al. [9] prove a lower bound of $\Omega(n \log n)$ for the general slope selection problem in the algebraic decision tree model that also holds in our setting. Both deterministic algorithms [9, 18, 7] and randomized [26, 10] algorithms have been proposed that achieve an $\mathcal{O}(n \log n)$ (expected) runtime. The problem has also been considered in other models, see, e. g., [6].

Asharov et al. [4] recently proposed an asymptotically optimal ORAM construction that matches the overhead factor of $\Omega(\log m)$ per memory access. This construction provides a general way to transform RAM programs into oblivious variants with no more than logarithmic overhead per memory operation. Due to large constants this optimal oblivious transformation is not viable in practice, though practically efficient (yet asymptotically suboptimal) constructions are available, see, e. g., [33]. Our algorithm matches the asymptotic runtime of an optimal transformation while maintaining practical efficiency and perfect security.

A different approach is the design of problem-specific algorithms without providing general program transformations. Oblivious algorithms for fundamental problems have been considered, e. g., for sorting [16, 3], sampling [28, 31], database joins [1, 22, 20], and some geometric problems [12]. To the best of our knowledge neither the slope selection problem nor the related inversion counting problem have been considered in the oblivious setting before.

2 A simple algorithm

As mentioned above, our approach is to modify the randomized algorithm proposed by Matoušek [26]. For this, we replace all non-trivial building blocks of the original algorithm — most notably intersection counting and intersection sampling — by oblivious counterparts.

2.1 The original algorithm

Algorithm 1 shows the original algorithm as described by Matoušek [26]. In a nutshell the algorithm works by maintaining intersections a and b as lower and upper bounds for the intersection p_k with k-th smallest x-coordinate to be identified.[2]

In the main loop a *randomized interpolating search* is performed, tightening the bounds a and b until only $N \in \mathcal{O}(n)$ intersections remain in between. For this, a multiset R of n intersections is sampled from the remaining intersections (with replacement) in each iteration. Then new bounds a' and b' are selected from R

[2]Generalizing the description of the algorithm in [26] we maintain the intersections a, b instead of only their x-coordinates.

Algorithm 1 Randomized intersection selection algorithm [26].

```
1:  function IntSelection($\overline{P}, k$)                                     ▷ $k \in [$IntCount$(\overline{P}, -\infty, +\infty))$
2:      $n \leftarrow |\overline{P}|; N \leftarrow$ IntCount$(\overline{P}, a, b)$          ▷ Number of input lines and of remaining intersections
3:      $a \leftarrow -\infty; b \leftarrow +\infty$
4:      do
5:          $j \leftarrow n \cdot (k -$ IntCount$(\overline{P}, -\infty, a) + 1) \,/\, N - 1$       ▷ Adjust $k$ relative to current boundaries
6:          $j_a \leftarrow \max\{0, \lfloor j - 3\sqrt{n} \rfloor\}; j_b \leftarrow \min\{n-1, \lceil j + 3\sqrt{n} \rceil\}$
7:          $R \leftarrow$ IntSample$(\overline{P}, a, b, n)$                             ▷ Sample intersection points
8:          $a' \leftarrow$ Select$_x(R, j_a)$; $b' \leftarrow$ Select$_x(R, j_b)$          ▷ Select candidate boundaries for next iteration
9:          $m_{a'} \leftarrow$ IntCount$(\overline{P}, -\infty, a')$; $m_{b'} \leftarrow$ IntCount$(\overline{P}, -\infty, b')$   ▷ Count intersections left of $a'$ and left of $b'$
10:         if $m_{a'} \le k < m_{b'} \wedge m_{b'} - m_{a'} \le 11N \,/\, \sqrt{n}$ then
11:             $a, b \leftarrow a', b'$                                            ▷ Update boundaries
12:             $N \leftarrow m_{b'} - m_{a'}$                                      ▷ Update remaining intersections
13:     while $N > n$
14:     $R \leftarrow$ IntEnumeration$(\overline{P}, a, b)$                         ▷ Enumerate all remaining intersections
15:     return Select$_x(R, k -$ IntCount$(\overline{P}, -\infty, a))$             ▷ Select correct intersection
```

based on the relative position of p_k among the remaining intersections. The check in Line 10 ensures that p_k lies within these new bounds and that the number of intersections has been sufficiently reduced. Matoušek proves that this check has a high probability to pass, implying that the number of remaining intersections is reduced by a factor of $\Omega(\sqrt{n})$ in an expected constant number of iterations. Thus only an expected constant number of loop iterations are required overall. With only $N \in \mathcal{O}(n)$ intersections remaining, the solution is computed by enumeration and selection.

The only non-standard building blocks required for the algorithm are intersection counting, the sampling of n intersections and the enumeration of intersections, all in a given range. Due to the composability of oblivious programs the use of oblivious replacements in Algorithm 1 leads to an oblivious algorithm; see Section 2.4.

2.2 Known oblivious building blocks

Sorting In the ORAM model, an array A of n elements can be sorted by a comparison-based algorithm in optimal $\Theta(n \log n)$ time [2, 16, 3]. We refer to this building block as `Sort`(A).

For the application in this paper we require a sorting algorithm which is fast in practice. To this end we can use *bucket oblivious sort* [3]: The algorithm works by performing an oblivious random permutation step with runtime $\mathcal{O}(n \log n)$, followed by a comparison-based sorting step (which is not necessarily oblivious). This leads to a total runtime of $\mathcal{O}(n \log n)$ when using an optimal comparison-based sorting algorithm. The random permutation ensures that the complete algorithm is oblivious, even if the sorting step is not (equal keys need to be handled separately). The permutation step in [3] has a bounded failure probability, but since failure of the random permutation leaks nothing about the input this step can be repeated until it succeeds. This leads to an $\mathcal{O}(n \log n)$ runtime in expectation.

Merging The building block `Merge`(A, B) takes two individually sorted arrays A and B and sorts the concatenation $A \parallel B$. There is a lower bound of $\Omega(n \log n)$ for merging in the indivisible oblivious RAM model.[3] *Odd-even merge* [5, 19] is an optimal merge algorithm (in the indivisible oblivious RAM model) with a good performance in practice.

Selection `Select`(A, k) denotes the selection of an element with rank k, i.e., a k-th smallest element, from an unordered array A. An optimal algorithm in the RAM model is Blum's linear-time selection algorithm. This problem can be solved by a near-linear oblivious algorithm [24], but current implementations suffer from high constant runtime factors due to the use of oblivious partitioning. For practical efficiency, we realize selection by sorting the given array A. Since for our application we may leak the index k, only one additional probe is required. We thus have leakage `leak`: $\langle A, k \rangle \mapsto \langle |A|, k \rangle$.

Filtering Filtering a field A with a predicate Pred ($\texttt{Filter}_{\text{Pred}}(A)$) extracts a sorted sub-list A' with all elements for which the predicate is true. The elements $a \in A'$ are stable swapped to the front of A and the number $|A'|$ of such elements is returned. Since filtering can be used to realize stable partitioning, the lower runtime bound of $\Omega(n \log n)$ for inputs of length n in the indivisible ORAM model of [24] applies. This operation can be implemented with runtime $\mathcal{O}(n \log n)$ using oblivious routing networks [15].

[3]Lin, Shi, and Xie [23] prove a lower bound of $\Omega(n \log n)$ for stable partition in the indivisible oblivious RAM model that also applies to merging. Due to [27] this bound applies even when restricting the input to arrays of (nearly) equal size.

Appending The building block $\texttt{Append}(A, B, i, k)$ is given two fields A and B as well as two indices i and k and appends the first k elements of B to the first i elements of A. This ensures that A' after the operation contains $A[0 : i] \parallel B[0 : k]$ in the first $i + k$ positions. All other positions may contain arbitrary elements. This operation can also be implemented with runtime $\mathcal{O}(n \log n)$ by using oblivious routing networks.

2.3 New oblivious building blocks

2.3.1 Inversion and intersection counting

The number of inversions in an array A is defined as the number of pairs of indices $i, j \in [|A|]$ with $A[i] > A[j]$ and $i < j$. In the RAM model, an optimal comparison-based approach to determine the number of inversions is a modified merge sort. Our oblivious merge-based inversion counting `Inversions` generalizes this to an arbitrary merge algorithm (with indivisible keys).

As noted by Cole et al. [9], inversion counting can be used to calculate the number of intersections of a set of lines in a given range $[a_x, b_x)$. This is by ordering the lines according to the y-coordinates at $x = a_x$ ($\leq_a$) and counting inversions relative to the order at $x = b_x$ ($\leq_b$). We use this to implement $\texttt{IntCount}(\overline{P}, a, b)$ for determining the number of intersections of lines $\overline{P}$:

Algorithm 2 Intersection counting.

```
1: function IntCount($\overline{P}, a, b$)
2:     $\overline{P}_a \leftarrow \texttt{Sort}_a(\overline{P})$            ▷ Sort according to $\leq_a$
3:     return $\texttt{Inversions}_b(\overline{P}_a)$       ▷ Count inversions
```

Given an array A of elements (in our case: lines sorted according to $\leq_a$), `Inversions` computes all inversions (in our case: corresponding to intersections in $[a_x, b_x)$) while at the same time sorting A. `Inversions` recursively computes all inversions in the first half A_{lo} and in the second half A_{hi} of the input. The inversions induced by lines from different halves, i. e., the number of pairs $\langle a, b\rangle \in A_{\text{lo}} \times A_{\text{hi}}$ with $a < b$, then is computed by $\texttt{BiInversions}(A_{\text{lo}}, A_{\text{hi}})$ which leverages that A_{lo} and A_{hi} may be assumed inductively to be sorted.

Algorithm 3 Merge-based inversion counting.

```
1: procedure BiInversions($A_{lo}, A_{hi}$)
2:     $\texttt{l}(e) \leftarrow 0,\ e \in A_{lo};\ \texttt{l}(e) \leftarrow 1,\ e \in A_{hi}$      ▷ Label
3:     $A \leftarrow \texttt{Merge}(A_{lo}, A_{hi})$    ▷ Permute labels as well
4:     $I \leftarrow 0; c \leftarrow 0$          ▷ No. of inversions / counter
5:     for $e \leftarrow A[0], \dots, A[|A| - 1]$ do
6:         if $\texttt{l}(e) = 0$ then
7:             $I \leftarrow I + c$                 ▷ Record inversions
8:         else                                   ▷ $\texttt{l}(e) = 1$
9:             $c \leftarrow c + 1$                ▷ Increase counter
10:    return $I$
```

To do this obliviously, `BiInversions` labels the elements according to which half they come from, then merges the labeled elements, and finally uses these labels to simulate the standard RAM merging algorithm. For this algorithm to work correctly, in general a stable merge algorithm is required, which sorts elements from the first half before elements from the second half if they are equal with regard to the order. We can drop this requirement since we only work on totally ordered inputs of unique elements.

The correctness of inversion counting follows from the correctness of `BiInversions`. Independent of the particular merge algorithm used, `BiInversions` is functionally equivalent to the merging step of the RAM algorithm. The runtime of `BiInversions` is dominated by merging, thus `Inversions` runs in time $\mathcal{O}(n \log^2 n)$. As merging has a lower bound of $\Omega(n \log n)$ in the indivisible ORAM model and, even without assuming indivisibility, no ORAM algorithm with runtime $o(n \log n)$ is known, any divide-and-conquer approach based on 2-way merges currently incurs a runtime of $\Omega(n \log^2 n)$.

Except for the invocation of `Merge`, all operations in `BiInversions` can be realized obliviously by a constant number of linear scans over the elements $A \coloneqq A_{\text{lo}} \parallel A_{\text{hi}}$ and their labels. Since `Merge` is oblivious, the obliviousness of `BiInversions` follows from the composability of oblivious programs. The obliviousness of `Inversions` again follows from composability. Finally, since the input is divided depending only on the size of the input, `Inversions` and `IntCount` only leak the input size.

Defining a suitable order Intuitively, the algorithm sorts the input (lines sorted according to $\leq_a$) according to $\leq_b$ while recording intersection points. At each such point, two lines adjacent in the underlying order exchange their position. In addition to handling boundary cases correctly, it is not immediately obvious how this approach can be modified to handle non-general positions, since there may be an arbitrary number of lines intersecting in a single point.

To be able to handle non-general positions obliviously, we do not explicitly use the y-coordinates to define $\leq_a$ and $\leq_b$. Instead, we — more generally — order the lines by their intersection points in relation to a given intersection p. For this, we use $p_{i\times j} \coloneqq \ell_i \cap \ell_j$ to denote the intersection point of two lines $\ell_i \neq \ell_j$.

Definition 2. *Let $P_\times \coloneqq \overline{L} \cup \{-\infty, +\infty\}$ be the set of all intersections formed by $\overline{P}$ with additional elements $-\infty$ and $+\infty$. Let also $\preceq$ be an order over $P_\times$ (with the corresponding strict order $\prec$). For each $p \in P_\times$, we define the binary relation $\leq_p$ over $\overline{P}$ as*

$$\ell_1 \leq_p \ell_2 :\Leftrightarrow \begin{cases} \top & \text{if } \ell_1 = \ell_2 \\ p \preceq p_{1\times 2} & \text{if } m(\ell_1) > m(\ell_2) \\ p_{1\times 2} \prec p & \text{if } m(\ell_1) < m(\ell_2) \end{cases}$$

For lines in general position, this definition essentially captures the ordering by y-coordinate: If the slope of ℓ_1 is larger than the slope of ℓ_2, ℓ_1 lies below ℓ_2 if their intersection point lies to the right of p; if the slope of ℓ_1 is smaller than the slope of ℓ_2, ℓ_1 lies above ℓ_2 if and only if their intersection point lies to the right.

Lemma 1: Correctness of `IntCount`. *Let $P_\times$, $\preceq$, $\prec$, and $\leq_p$ be as defined above. If*
(a) $\preceq$ is a total order over $P_\times$ with minimum $-\infty$ and maximum $+\infty$ and
(b) $\leq_p$ is a total order over $\overline{P}$ for all $p \in P_\times$,
then, given $a, b \in P_\times$ with $a \preceq b$, `IntCount` determines the number of intersections $p \in \overline{L}$ with $a \preceq p \prec b$.

Proof. `IntCount` sorts according to the order $\leq_a$ and then counts inversions according to the order $\leq_b$. The algorithm thus exactly counts the number of unique pairs $\{\ell_1, \ell_2\} \subseteq \overline{P}$ (assuming w.l.o.g. $m(\ell_1) > m(\ell_2)$) for which $(\ell_1 \leq_a \ell_2) \neq (\ell_1 \leq_b \ell_2)$. Since $\preceq$ is a total order and $a \preceq b$ this can only occur if $\ell_1 \leq_a \ell_2 \land \ell_1 \not\leq_b \ell_2$. Then $a \preceq p_{1\times 2} \prec b$ follows directly from the definition of $\leq_p$, thus `IntCount` counts exactly the number of intersections in the range $[a, b)$. $\square$

Since we want to identify the intersection with median x-coordinate, the intersections need to be ordered primarily by their x-coordinate. If all intersection points have distinct x-coordinates — which is the case for lines $\overline{P}$ in general position — we have:

Remark 1. *Let $\overline{P}$ be in general position and $\preceq$ be defined as $p \preceq q :\Leftrightarrow p_x \leq q_x$ for $p, q \in P_\times$ with special cases $-\infty \preceq p$ and $p \preceq +\infty$ for all $p \in P_\times$. Then both conditions in Theorem 1 are satisfied.*

We will prove this more generally in Section 3.

The intersection point of two given lines can be determined in constant time, so $\leq_p$ can be evaluated in constant time as well. As such the runtime of `IntCount` is dominated by `Inversions` and thus $\mathcal{O}(n \log^2 n)$ for n given lines. The method is oblivious by composability.

2.3.2 Intersection sampling and enumeration

The last building blocks to consider are the independent sampling as well as the enumeration of intersection points from a given range $[a, b)$. We need to avoid calculating all intersections explicitly, as this would result in a runtime of $\mathcal{O}(n^2)$. Recall that sampling can be done efficiently in the RAM model by modifying the standard intersection counting algorithm: First, a set K of k indices from the range $[\texttt{IntCount}(\overline{P}, a, b)]$ are sampled and then the intersection count is computed while iterating over the generated indices, reporting the corresponding intersections on the fly [26].

Unfortunately, this approach is not oblivious: First, synchronized iterations (such as over K and the set of intersections generated) are not oblivious in general as step widths depend on the data values encountered. Second, reporting an intersection on the fly leaks information about the lines inducing it.

We address these challenges in the following way. Just as we have done in `BiInversions`, we simulate a synchronized traversal over arrays A and B by first sorting the (labeled) elements and then iterating over their concatenation $A \| B$. For each element, we decide in private memory how to process the element based on its label.

To avoid leaking information about the two lines inducing a single intersection, we operate on batches producing partial results padded to their maximum possible length where needed. This way we do not leak the number of samples from a specific sub-range of the input.

We combine sampling and enumerating into a single building block $\texttt{IntCollect}(\overline{P}, a, b, K)$; K contains the indices of the intersections to sample in ascending order.

Algorithm 4 Enumerating specified intersections.

```
1: function IntCollect($\overline{P}, a, b, K$)          ▷ $a \prec b$, $|K| > 0$
2:     $k' \leftarrow 0$; $K' \leftarrow$ array[$|K|$]       ▷ Intersection storage
3:     $\overline{P}_a \leftarrow$ Sort$_a(\overline{P})$              ▷ Sort according to $\leq_a$
4:     $I \leftarrow 0$                                ▷ Intersection counter
5:     for $l \leftarrow 0, \ldots, \lceil \log_2 |\overline{P}| \rceil - 1$ do        ▷ All layers
6:         DetermineLineIndices$_b(\overline{P}_a, I, l)$   ▷ Upd. $I$
7:         $X \leftarrow$ MatchAgainstLines$(\overline{P}_a, K, l)$
8:         StoreIntersections$(X, K', k')$     ▷ Upd. $k'$
9:     return $K'$
```

From a high-level perspective, the algorithm first sorts the input according to $\leq_a$ and then iteratively implements a bottom-up divide-and-conquer strategy: As in the RAM algorithm sketched before, unique consecutive indices are (implicitly) assigned to all encountered intersection points. Note that, as we randomly sample/enumerate intersections, we may assign indices to the intersections arbitrarily. All lines are explicitly labeled with indices so that — given the index for an intersection — the lines inducing that intersection can easily be identified.

The intersection indices K are then matched against the lines, determining the inducing lines of each intersection. Finally, we store the pair of inducing lines as intersection in K'. These three steps are repeated for each layer l so that after processing all layers the inducing lines of all specified intersections are known.

We now discuss the routines called for each layer l.

Assigning indices to lines The first sub-routine called for each layer l is `DetermineLineIndices`. Building on the general ideas used in Algorithm 3, it iterates over pairs of subarrays of 2^l lines each, updates the intersection counter I, and assigns to each line in $\overline{P}$ four indices defined below that guide the oblivious sampling.

	before merge									*after merge*							
$\underline{\text{i}}$	0				1					0				1			
$\underline{\text{half}}$	0	0	1	1	0	0	1	1		0	1	1	0	1	1	0	0
$e \in L$	ℓ_0	ℓ_6	ℓ_1	ℓ_5	ℓ_4	ℓ_7	ℓ_2	ℓ_3	$\rightarrow$	ℓ_0	ℓ_1	ℓ_5	ℓ_6	ℓ_2	ℓ_3	ℓ_4	ℓ_7
$\underline{\text{0-index}}$										3			3			5	7
$\underline{\text{1-index}}$										0	0	1	2	2	3	2	2

Table 1: Labels assigned by `DetermineLineIndices` in layer $l = 1$ for an input of 8 lines $\ell_0, \dots, \ell_7$, numbered according to their $\leq_b$-order. In layer 0, $I = 3$ inversions have been counted. Layer 1 contains 6 inversions.

assigned index	3	4	5	6	7	8
intersection point	$p_{6\times 1}$	$p_{6\times 5}$	$p_{4\times 2}$	$p_{4\times 3}$	$p_{7\times 2}$	$p_{7\times 3}$
$\underline{\text{i}}$	0	0	1	1	1	1
$\underline{\text{0-index}}$ (index of inducing 0-line)	3	3	5	5	7	7
$\underline{\text{1-index}}$ (index of inducing 1-line)	0	1	2	3	2	3

Table 2: Result of the merging step shown in Table 1. Note that the indices are only assigned conceptually and the intersections are not computed explicitly. The assigned index is equal to $\underline{\text{0-index}} + (\underline{\text{1-index}} - \underline{\text{i}} \cdot 2^l)$.

Algorithm 5 Assigning indices to lines.

```
procedure DetermineLineIndices($\overline{P}_a, I, l$)
    $c^* \leftarrow 0$                                   ▷ Ctr. for 1-lines on level $l$
    for $i \leftarrow 0, \dots, \left\lceil \frac{|\overline{P}_a|}{2 \cdot 2^l} \right\rceil$ do                 ▷ Pairs of subarrays
        $\overline{P}_{\text{lo}} \leftarrow \overline{P}_a[2 \cdot i \cdot 2^l : 2 \cdot (i+1) \cdot 2^l - 1]$
        $\overline{P}_{\text{hi}} \leftarrow \overline{P}_a[2 \cdot (i+1) \cdot 2^l : 2 \cdot (i+2) \cdot 2^l - 1]$
        $\texttt{l}(\ell, \underline{\text{i}}) \leftarrow i$ for all $\ell \in \overline{P}_{\text{lo}} \parallel \overline{P}_{\text{hi}}$
        BiInversions'$_b$($\overline{P}_{\text{lo}}, \overline{P}_{\text{hi}}, I, c^*$)

procedure BiInversions'$_b$($\overline{P}_{\text{lo}}, \overline{P}_{\text{hi}}, I, c^*$)
    $\texttt{l}(\ell, \underline{\text{half}}) \leftarrow 0,\ \ell \in \overline{P}_{\text{lo}}$; $\texttt{l}(\ell, \underline{\text{half}}) \leftarrow 1,\ \ell \in \overline{P}_{\text{hi}}$
    $A \leftarrow$ Merge($\overline{P}_{\text{lo}}, \overline{P}_{\text{hi}}$)          ▷ Permute labels as well
    $c \leftarrow 0$                                    ▷ Local counter for 1-lines
    for $\ell \leftarrow A[0], \dots, A[|A| - 1]$ do
        $\texttt{l}(\ell, \underline{\text{0-index}}) \leftarrow I$
        if $\texttt{l}(\ell, \underline{\text{half}}) = 0$ then
            $\texttt{l}(\ell, \underline{\text{1-index}}) \leftarrow c$      ▷ Number of int. for $\ell$
            $I \leftarrow I + c$          ▷ Update intersection count
        else                                    ▷ $\texttt{l}(\ell, \underline{\text{half}}) = 1$
            $\texttt{l}(\ell, \underline{\text{1-index}}) \leftarrow c^*$          ▷ Record offset
            $c \leftarrow c + 1$               ▷ Update local counter
            $c^* \leftarrow c^* + 1$           ▷ Update level counter
```

- The index $\underline{\text{i}}$ of a line ℓ (or: $\texttt{l}(\ell, \underline{\text{i}})$) denotes the pair of blocks (on the current layer) containing ℓ. On each layer, we process only intersections of lines with the same index $\underline{\text{i}}$.
- The index $\underline{\text{half}}$ of a line ℓ indicates whether ℓ was stored in the first subarray $\overline{P}_{\text{lo}}$ ($\texttt{l}(\ell, \underline{\text{half}}) = 0$, "0-line") or in the second subarray $\overline{P}_{\text{hi}}$ ($\texttt{l}(\ell, \underline{\text{half}}) = 1$, "1-line"). For each pair of subarrays, we process only intersections of lines with different indices $\underline{\text{half}}$.
- For a 0-line ℓ_0, the index $\underline{\text{0-index}}$ is the offset of the first intersection induced by ℓ_0. By construction all intersections induced by ℓ_0 in this layer have consecutive indices. For a 1-line, $\underline{\text{0-index}}$ stores the number of intersections counted thus far, i. e., all lines are sorted by their values of $\underline{\text{0-index}}$ after merging.
- For a 1-line ℓ_1, $\underline{\text{1-index}}$ is the offset among all 1-lines in this layer. For a 0-line ℓ_0, $\underline{\text{1-index}}$ stores the number of intersection points induced by ℓ_0.

The resulting algorithm is given as Algorithm 5. Table 1 shows the labels assigned by Algorithm 5 in layer $l = 1$ when processing a sample input. The labels correspond to the indices implicitly assigned to the intersection points shown in Table 2. The indices are assigned to the lines so that an intersection with index $i \in K$ is induced by a 0-line ℓ_0 with next lower $\underline{\text{0-index}}$ relative to i. The $\underline{\text{1-index}}$ of the inducing 1-line ℓ_1 then is

$$\texttt{l}(\ell_1, \underline{\text{1-index}}) = \underbrace{i - \texttt{l}(\ell_0, \underline{\text{0-index}})}_{\text{relative index of } \ell_1 \text{ in the current pair of blocks}} + \texttt{l}(\ell_0, \underline{\text{i}}) \cdot 2^l$$

Like `BiInversions` the runtime of `BiInversions`$'_b$ is dominated by the call to `Merge` and thus $\mathcal{O}(s \log s)$ for sorted $\overline{P}_{\text{lo}}, \overline{P}_{\text{hi}}$ of size s. This means that the runtime of `DetermineLineIndices` is in $\mathcal{O}\left(\frac{n}{s} \cdot s \log s\right) \subseteq \mathcal{O}(n \log n)$. `BiInversions`$'_b$ is oblivious like `BiInversions` is. Since the main loop for `DetermineLineIndices` only depends on n and l, as does the size of the input to `BiInversions`$'_b$, the procedure is oblivious by composability with regard to leakage `leak`: $\langle \overline{P}_a, I, l \rangle \mapsto \langle |\overline{P}_a|, l \rangle$.

Matching lines and indices The second sub-routine, `MatchAgainstLines` (Algorithm 6), pairs the lines inducing intersection points encountered in this layer that correspond to indices in K.

First, the indices are matched against the 0-lines. This is done by assigning each index $i \in K$ the $\underline{\text{0-index}}$ $i + 0.5$ and then merging them with the lines (by

Algorithm 6 Method for matching intersection indices against lines.

```
function MatchAgainstLines($\overline{P}_a, K, l$)                    ▷ Lines $\overline{P}_a$ already have the appropriate labels
    $\mathtt{l}(i, \underline{\text{K}}) \leftarrow \top$ for all $i \in K$                                        ▷ Mark intersection indices
    $\mathtt{l}(i, \underline{\text{0-index}}) \leftarrow i + 0.5$ for all $i \in K$
    $X \leftarrow \mathtt{Merge}_{\underline{\text{0-index}}}(\overline{P}_a, K)$                       ▷ Merge lines and intersection indices
    $\ell_0 \leftarrow \bot$; $\ell_1 \leftarrow \bot$                              ▷ $\mathtt{l}(\bot, \underline{\text{0-index}}) := \mathtt{l}(\bot, \underline{\text{1-index}}) := 0$
    for $e \leftarrow X[0], \dots, X[|X| - 1]$ do                  ▷ Iterate over lines and indices, ignoring 1-lines
        if $\neg\mathtt{l}(e, \underline{\text{K}}) \wedge \mathtt{l}(e, \underline{\text{half}}) = 0 \wedge \mathtt{l}(e, \underline{\text{1-index}}) > 0$ then        ▷ Found 0-line inducing intersections
            $\ell_0 \leftarrow e$
        else if $\mathtt{l}(e, \underline{\text{K}}) \wedge \mathtt{l}(e, \underline{\text{0-index}}) < \mathtt{l}(\ell_0, \underline{\text{0-index}}) + \mathtt{l}(\ell_0, \underline{\text{1-index}})$ then    ▷ Found intersection index
            $\mathtt{l}(e, \underline{\text{0-line}}) \leftarrow \ell_0$                                  ▷ Mark index with inducing 0-line
            $\mathtt{l}(e, \underline{\text{1-index}}) \leftarrow \mathtt{l}(\ell_0, \underline{\text{i}}) \cdot 2^l + \mathtt{l}(e, \underline{\text{0-index}}) - \mathtt{l}(\ell_0, \underline{\text{0-index}})$   ▷ Calculate offset of the inducing 1-line
    $\mathtt{Sort}_{\underline{\text{1-index}}}(X)$
    for $e \leftarrow X[0], \dots, X[|X| - 1]$ do                  ▷ Iterate over lines and indices, ignoring 0-lines
        if $\neg\mathtt{l}(e, \underline{\text{K}}) \wedge \mathtt{l}(e, \underline{\text{half}}) = 1$ then                                   ▷ Found 1-line
            $\ell_1 \leftarrow e$
        else if $\mathtt{l}(e, \underline{\text{K}}) \wedge \mathtt{l}(e, \underline{\text{1-index}}) = \mathtt{l}(\ell_1, \underline{\text{1-index}}) + 0.5$ then           ▷ Found intersection index
            $\mathtt{l}(e, \underline{\text{1-line}}) \leftarrow \ell_1$                                  ▷ Mark index with inducing 1-line
    return $X$
```

<u>0-index</u>). When iterating over the merged sequence X, the 0-line inducing an intersection from this layer is exactly the last 0-line encountered before the index (that induces at least one intersection). Each index $i \in K$ is labeled with the inducing 0-line ℓ_0 as <u>0-line</u> and with the index of the corresponding 1-line as <u>1-index</u>; the <u>1-index</u> can be determined from the indices assigned to ℓ_0.

Similarly, the indices are matched against the 1-lines by sorting the array X of lines and indices according to the <u>1-index</u>. When iterating over the sorted sequence, the previous 1-line before each intersection index is the second line inducing the intersection. Each $i \in K$ already assigned a <u>0-line</u> can thus be labeled with the inducing 1-line ℓ_1 as <u>1-line</u>.

The runtime is dominated by the runtime for merging and sorting and thus is in $\mathcal{O}((n+k)\log(n+k))$ for $k := |K|$ and $n := |\overline{P}_a|$. The algorithm is oblivious since, in addition to merging and sorting, it only consists of linear scans over the array X. The input size for merging and sorting is at most $n + k$. Although not explicitly shown it is trivial to implement the loop bodies obliviously with respect to both memory access and memory trace-obliviousness. By composability, `MatchAgainstLines` is oblivious with regard to leakage `leak:` $\langle \overline{P}_a, K, l\rangle \mapsto \langle |\overline{P}_a|, |K|\rangle$.

Storing intersection The third subroutine called for each layer, `StoreIntersections` (Algorithm 7), stores the intersections (consisting of the pairs of lines matched in the previous step) in K'. Exactly the indices with an assigned <u>0-line</u> (and thus also <u>1-line</u>) have been found in this layer. For storing, the building block `Append` is used where k' is the number of indices already stored in K'. `Append` is oblivious and thus does not leak the number of intersections k_Δ from this layer. The runtime of this last step is dominated by the filtering and appending steps and thus realizable with runtime $\mathcal{O}(n \log n)$ where $n := |X|$. The obliviousness follows from composability with regard to leakage `leak:` $\langle X, K', k'\rangle \mapsto \langle |X|, |K|\rangle$.

Algorithm 7 Storing the sampled indices

```
procedure StoreSampledIntersections($X, K', k'$)
    $k_\Delta \leftarrow \mathtt{Filter}_{\underline{\text{K}} \wedge \underline{\text{0-line}}}(X)$        ▷ Matched indices
    $\mathtt{Append}(K', X, k', k_\Delta)$   ▷ Append (pairs of) lines
    $k' \leftarrow k' + k_\Delta$
```

Runtime and obliviousness Let $n := |\overline{P}|$ be the number of lines and $k := |K|$. The runtime of `IntCollect` is dominated by the main loop. This results in a total runtime of $\mathcal{O}(\log n(n+k)\log(n+k)) \overset{k \in \mathcal{O}(n)}{=} \mathcal{O}(n \log^2 n)$. The number of iterations and the sequence of values for l only depends on n and sub-routines only leak n, k, $n+k$, or l. Thus, `IntSample` is oblivious by composability with regard to leakage `leak:` $\langle \overline{P}, a, b, K\rangle \mapsto \langle |\overline{P}|, |K|\rangle$.

2.4 Analysis

Since our implementation of Matoušek's algorithm replaces only the building blocks used internally, the correctness and runtime properties follow from the respective analyses of the building blocks. We thus have:

Lemma 2: Correctness and runtime. *Let* `IntSelection` *be Algorithm 1 instantiated with the oblivious building blocks described above. Then, given a set* $\overline{P}$ *of* n *lines in general position and an integer* $k \in \left[\binom{n}{2}\right]$, `IntSelection`$(\overline{P}, k)$ *determines the in-*

tersection with k-th smallest x-coordinate in expected $\mathcal{O}(n \log^2 n)$ time.

We now turn our attention to the analysis of the proposed algorithm's obliviousness. Since oblivious programs are composable, we can prove the security by considering the leakage of each oblivious building block.

Lemma 3: Obliviousness. *Let $\overline{P}$ be a set of n lines in general position such that $\binom{|\overline{P}|}{2}$ is odd. If Algorithm 1 is instantiated with the oblivious building blocks described above,* $\texttt{MedianSelection}(\overline{P}) := \texttt{IntSelection}(\overline{P}, k)$ *with* $k := \frac{\binom{|\overline{P}|}{2}-1}{2}$ *obliviously realizes the median intersection selection with respect to leakage* $\texttt{leak}(\overline{P}) := |\overline{P}|$.

Proof. For the proof, we need to show both the correctness and the security of the algorithm for the specified inputs. The requirements above imply that k is an integer, so correctness follows from Theorem 2. It remains to show the security.

The oblivious algorithm directly uses the building blocks $\mathcal{S} := \langle \texttt{Sort}, \texttt{Select}, \texttt{IntCount}, \texttt{IntCollect} \rangle$ The building block $\texttt{IntCollect}$ is used to realize $\texttt{IntSample}(\overline{P}, a, b, k)$ by first determining the number of inversions $i := \texttt{IntCount}(\overline{P}, a, b)$ in range $[a, b)$, independently sampling k random indices $K \in [I]^k$, sorting the indices K and calling $\texttt{IntCollect}(\overline{P}, a, b, K)$. Similarly $\texttt{IntCollect}$ is used to realize $\texttt{IntEnumeration}(\overline{P}, a, b)$ by initializing an array $K := \langle 0, \ldots, i-1 \rangle$ and calling $\texttt{IntCollect}$. All building blocks are oblivious, with $\texttt{Select}$ additionally leaking the rank of the selected element, $\texttt{IntSample}$ leaking the number of samples via the size of K and $\texttt{IntEnumeration}$ leaking the number of intersections in the given range, also via the size of K. The arithmetic expressions and assignments operate on a constant number of memory cells and are trivially oblivious.

We first examine the values of n, k, N and $N' := \texttt{IntCount}(\overline{P}, -\infty, a)$ throughout the execution of the algorithm. The value of n remains constant and k is fixed relative to n, so we consider the sequence $B = \langle \langle N_0, N'_0 \rangle, \langle N_1, N'_1 \rangle, \ldots, \langle N_m, N'_m \rangle \rangle$ where N_i, N'_i are the values for N, N' after the i-th iteration of the main loop for a total of m loop iterations. In each iteration of the main loop, n intersections R are chosen uniformly at random from the range $[a, b)$. Since $\overline{P}$ is in general position, the intersections of distinct pairs of lines are distinct and all intersections are totally ordered. This implies that the random distribution of $\texttt{IntCount}(\overline{P}, -\infty, c)$ for an intersection c with fixed rank in R only depends on n, N and N'. Both j_a and j_b are fixed relative to n, N and N', so the random distribution of the next values for N and N' is solely determined by n and the previous values. Since initially $N_0 = \binom{n}{2}$ and $N'_0 = 0$ and the sequence ends with $N_m \leq n$, the random distribution of the complete sequence B is solely determined by n.

It can easily be seen that each sequence B of values for N, N' determines the sequence A of memory probes and sub-procedure invocations. This implies that any sequence A is equally likely for inputs of the same size and thus that $\texttt{MedianSelection}$ is secure by composability. □

3 Non-general positions

For simplicity of exposition, we assumed so far that the lines $\overline{P}$ are in general position, i. e., that all intersection points of two lines in $\overline{P}$ have distinct x-coordinates and that all lines in $\overline{P}$ have distinct slopes. We also assumed that the number of intersection points is odd, so that the median intersection point selection problem can always be solved by one call to a general intersection point selection algorithm; this latter assumption can be removed by computing both the element with rank $k_1 = \lfloor \frac{N-1}{2} \rfloor$ and with rank $k_2 = \lceil \frac{N-1}{2} \rceil$ (for $N = \binom{|\overline{P}|}{2}$) and returning their mean if there is an even number of intersections [30]. Since k_1 and k_2 differ by one at most by one, both intersections can be computed simultaneously with no significant impact on the runtime.

In RAM algorithms, degenerate configurations are a nuisance, but often can be handled by generic approaches — see, e. g., [11, 29, 34]. For our proposed algorithm, we must take care that these approaches do not affect the obliviousness. In particular, the runtime of the algorithm must not depend on the number of intersection points with identical x-coordinates; this rules out the problem-specific technique described by Dillencourt, Mount, and Netanyahu [10] to explicitly handle non-general position.

Regarding arithmetic precision, we note that the only arithmetic computation performed on the input values is the calculation of the x-coordinate of an intersection point. Thus, recall we are working in the word RAM model, for fixed-point input values with b bits of precision the use of $2(b+1)$ bits of precision suffices to perform all arithmetic computations exactly.

Parallel lines For technical reasons, we first discuss how to deal with inputs in which lines are parallel, i. e., for which we cannot assume distinctness of slopes.

Earlier on, we noted that our algorithm is allowed to leak the values of N and k.[4] This means that we cannot introduce data-dependency of these values and this, in turn, implies that (a) pairs of parallel lines cannot simply be excluded and that (b) k cannot be adjusted based on the number of pairs of parallel lines.

[4]Assuming the leakage of k allows us to treat the original algorithm of Matoušek as a black box. The author proves an expected lower bound on the reduction of N per loop iteration which is independent of k. This does not necessarily imply that the exact reduction of N is in fact independent of k.

We address this using a problem-specific, controlled version of the symbolic perturbation described in [11] and [34]. We perturb the lines $\overline{P}$ in such a way that each pair $\{\ell_1, \ell_2\}$ of lines intersects in a single intersection point $p_{1\times 2}$. Let V be the set of intersections induced by lines that were parallel previous to the perturbation. We ensure that V is partitioned into $V = V_- \cup V_+$ such that V_- and V_+ are (nearly) equally sized and each $v \in V_-$ has a x-coordinate less and each $v' \in V_+$: By equally distributing these "virtual" intersections to the left and to the right of all "real" intersections we maintain data-independent values of $N = \binom{n}{2}$ and $k = \frac{N-1}{2}$.

To realize this (symbolic) perturbation, we follow Edelsbrunner and Mücke [11] and introduce an infinitesimally small value $\varepsilon > 0$. We then identify each line $\ell\colon x \mapsto m(\ell) \cdot x + b(\ell)$ with the perturbed line $\ell'\colon x \mapsto m'(\ell) \cdot x + b'(\ell)$ where $m(\ell') \coloneqq m(\ell) + s_\ell \cdot \#_\ell \cdot \varepsilon^2$, $b(\ell') \coloneqq b(\ell) + \#_\ell \cdot \varepsilon$, $s_\ell \in \{-1, +1\}$ is a factor to achieve the distribution into V_- and V_+, and $\#_\ell \in \mathbb{N}_0$ is a unique index given to each line with respect to the order of the line offsets, i. e. $\forall \ell_1, \ell_2 \in \overline{P}\colon b(\ell_1) < b(\ell_2) \implies \#_{\ell_1} < \#_{\ell_2}$. We obtain the set $\overline{P}'$ of perturbed lines.

Due to space constraints, we omit the details of how to compute $\#_\ell$ and s_ℓ as well how to avoid leaking the number of "virtual" intersections.

Intersections with identical x-coordinates To handle intersections $p, q \in \overline{L}$ with identical x-coordinates without significantly affecting the runtime of the algorithm, we establish a total order $\preceq$ over all intersections, so that the lines $\overline{P}$ can be totally ordered relative to each intersection p as in Definition 2. For this, we characterize an intersection by its inducing pair of lines and define an order based on these lines' properties:

Definition 3. *Let $P_\times \coloneqq \overline{L} \cup \{-\infty, +\infty\}$ be the set of all intersections with additional elements $-\infty$ and $+\infty$. Let each $p \in \overline{L}$ be formed by lines $p_\uparrow$ and $p_\downarrow$ with $m(p_\uparrow) > m(p_\downarrow)$. We define a total order $\preceq$ over $P_\times$ via:*

$$p \preceq q :\Leftrightarrow \begin{cases} p_x < q_x & \text{if } p_x \neq q_x \\ m(p_\uparrow) < m(q_\uparrow) & \text{else if } p_\uparrow \neq q_\uparrow \\ m(p_\downarrow) \leq m(q_\downarrow) & \text{else} \end{cases}$$

for $p, q \in P_\times \setminus \{-\infty, +\infty\}$ and with special cases $-\infty \preceq p$ and $p \preceq +\infty$ for all $p \in P_\times$. Let $\prec$ denote the corresponding strict order over $P_\times$.

By construction, $\preceq$ is a (lexicographic) total order. This is ensured by the fact that all slopes are distinct. This order suffices to construct the total order over the lines in $\overline{P}$. To show this, we need the following lemma:

Lemma 4. *Let ℓ_1, ℓ_2, ℓ_3 be non-vertical lines with $m(\ell_1) < m(\ell_2) < m(\ell_3)$. Of the three intersections induced by these lines, the intersection $p_{1\times 3}$ of the two lines with extremal slopes is the median with respect to the order $\preceq$ defined above.*

Assuming only the distinctness of slopes (which, as discussed above, may be assumed w.l.o.g.), we have:

Lemma 5. *Let $P_\times$, $\preceq$, and $\prec$ be as in Definition 3. For each $p \in P_\times$, we have a total order $\leq_p$ over $\overline{P}$:*

$$\ell_1 \leq_p \ell_2 :\Leftrightarrow \begin{cases} \top & \text{if } \ell_1 = \ell_2 \\ p \preceq p_{1\times 2} & \text{if } m(\ell_1) > m(\ell_2) \\ p_{1\times 2} \prec p & \text{if } m(\ell_1) < m(\ell_2) \end{cases}$$

With the above definition, we can impose a total order on the set of lines irrespective of whether or not their intersection points' x-coordinates are distinct. Since the predicate $p \preceq q$ for intersections $p, q \in P_\times$ can still be evaluated in constant time, the asymptotic runtime of the algorithm remains unchanged.

Summary In conclusion, the two techniques sketched in this section generalize the algorithm not only to inputs $\overline{P}$ with parallel lines, but also to inputs with identical lines. The algorithm is thus applicable to arbitrary inputs. Since we can achieve the desired (symbolic) perturbation via pre-processing in $\mathcal{O}(n \log n)$ time for an input of n lines, our main theorem follows:

Theorem 6: Main result. *There exists a RAM program that obliviously realizes the median intersection selection in expected $\mathcal{O}(n \log^2 n)$ time for n non-vertical lines inducing at least one intersection.*

4 Implementation and evaluation

We developed a prototype of our oblivious algorithm in C++.[5] The goal of the implementation is to show that the algorithm is easily implementable and to provide an estimate of the algorithm's performance. For this we also implemented the baseline algorithm [26].

Limitations The primary limitation is that our prototype only accesses arrays of non-constant size in an oblivious manner. Code fragments such as inner loops and methods accessing only a constant number of memory cells do not necessarily probe memory obliviously. Even though it is conceptually trivial to transform those code fragments to achieve "full" obliviousness, we note that — without publicly available libraries providing low-level primitives for implementations of oblivious algorithms — the obliviousness eventually might depend on the compiler and platform used.

We believe that our implementation still provides a good estimate of the performance of a "fully" oblivious implementation: The loops in our runtime-intensive

[5] `http://go.wwu.de/ms6fz`

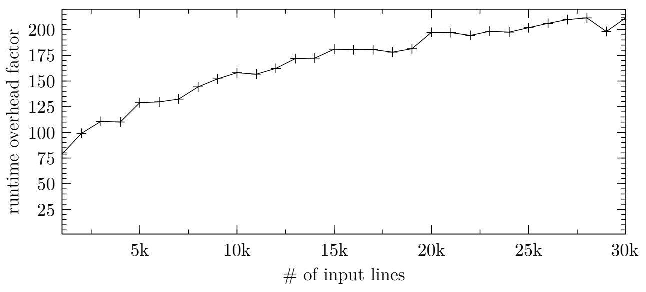


Figure 1: Runtime overhead factor (averaged over 10 random inputs) of the oblivious algorithm compared to the baseline algorithm with non-oblivious primitives.

primitives are all linear scans over arrays. As such "fully" oblivious loop bodies will not introduce a large overhead since they will likely not introduce cache misses. Also our oblivious primitives can also be implemented largely without data-dependent branches, thus potentially eliminating branch mispredictions.

The second main limitation is that we do not implement the handling of parallel lines (as described in Section 3). This would require an additional pre-processing step as well as extending both the slope and the offset with a symbolic perturbation. As mentioned above this would result in a low constant factor overhead in both runtime and memory space usage. Since this applies to both the oblivious and non-oblivious algorithm this has no direct implication for the performance evaluation below, although there might be a more efficient way to handle identical slopes in the non-oblivious case.

Finally our implementation resorts to a suboptimal, but easy-to-implement oblivious sorting primitive with $\mathcal{O}(n \log^2 n)$ and thus has an expected $\mathcal{O}(n \log^3 n)$ runtime in the oblivious setting. This leads to an additional $\mathcal{O}(\log n)$ overhead in runtime as compared to our non-oblivious implementation and thus underestimates the performance of the proposed algorithm.

Performance We used `libbenchmark`[6] to measure the runtime for inputs ranging from 1,000 to 30,000 lines. The input consists of shuffled sets of lines with non-uniformly increasing slope and a random offset, both represented by 64-bit integers. For all our experiments and independent of n, we fixed an interval $[m_{\min}, m_{\max}]$ and an interval $[b_{\min}, b_{\max}]$. To generate a set of n random lines, we then set $r \coloneqq (m_{\max} - m_{\min}) \,/\, n$ and constructed each line $\ell_i = \langle m_i, b_i \rangle$ in turn by independently sampling a random slope m_i from $m_{\min} + i \cdot r \leq m_i < m_{\min} + (i+1) \cdot r$ (thus ensuring both spread and distinctness of slopes) and a random offset b_i from $b_{\min} \leq b_i \leq b_{\max}$. We then permuted the resulting set of lines using `std::ranges::shuffle`.

[6]`https://github.com/google/benchmark`

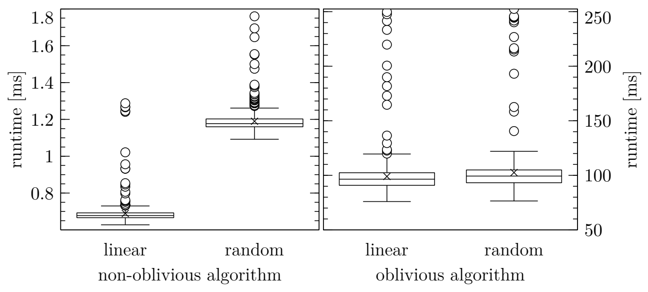


Figure 2: Runtime distribution over 500 runs of the oblivious and non-oblivious algorithms for $n = 1000$ lines. Left (in each subfigure): Data for a fixed, sorted input of lines intersecting in a single point. Right (in each subfigure): Data for a shuffled input of lines with non-uniformly increasing slopes and a random offset.

The performance evaluation results are shown in Fig. 1. For inputs of 10,000–30,000 random lines our algorithm is about 150–210 times slower than the baseline algorithm. While this is a significant slowdown, we remind the reader of both the logarithmic overhead incurred by choosing a suboptimal sorting algorithm and the fact that the baseline algorithm does not offer any obliviousness. The runtime was less than 10 seconds for all evaluated input sizes.

All experiments were performed on a Dell XPS 7390 with an Intel i7–10510U CPU and 16 GiB RAM running Ubuntu 20.04.

Obliviousness We assessed the obliviousness of our implementation of the building blocks by tracing memory accesses as part of unit testing. For this, we abstracted the memory sections as arrays of fixed but dynamic size. We assigned a fingerprint to each sequence of reads and writes by hashing both the memory operation and the access location. Since all building blocks used by the main algorithm are deterministic, we asserted their obliviousness by comparing fingerprints for different inputs with identical leakage.

Additionally, we evaluated the runtime of both our oblivious algorithm and the baseline algorithm when applied to two inputs of different characteristics. For this we compared the random lines described above with a sorted set of lines $\ell_i = \langle i, -i \rangle$, intersecting in the single point $p = \langle 1, 0 \rangle$. The baseline algorithm showed significantly different runtimes for different inputs (Fig. 2), making it abundantly clear that even without statistical analyses an adversary can distinguish these different kinds of input from the runtime alone. In contrast, there was only slight variation in the runtime of our proposed algorithm which we attribute to the presence of code processing constant-sized subproblems in a (currently) non-oblivious manner.

5 Conclusion

We presented a modification of the randomized algorithm of [26] for obliviously determining the median slope for a given set of n points. We also showed how to generalize the algorithm to arbitrary inputs — allowing both collinear points and multiple points with identical x-coordinate — while maintaining obliviousness. Our modified algorithm has an expected $\mathcal{O}(n \log^2 n)$ runtime, matching the general oblivious transformation bound of the original algorithm. We provide a proof-of-concept of the oblivious algorithm in C++, showing that the algorithm indeed can be implemented and has a runtime that make its application viable in practice.

References

[1] Rakesh Agrawal et al. "Sovereign Joins". In: *Proceedings of the 22nd International Conference on Data Engineering*. 2006. DOI: 10.1109/ICDE.2006.144.

[2] Miklós Ajtai, János Komlós, and Endre Szemerédi. "An O(n log n) Sorting Network". In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. 1983, pp. 1–9. DOI: 10.1145/800061.808726.

[3] Gilad Asharov et al. "Bucket Oblivious Sort: An Extremely Simple Oblivious Sort". In: *Proceedings of the 3rd SIAM Symposium on Simplicity in Algorithms*. 2020, pp. 8–14. DOI: 10.1137/1.9781611976014.2.

[4] Gilad Asharov et al. "OptORAMa: Optimal Oblivious RAM". In: *Advances in Cryptology – EUROCRYPT 2020*. Vol. 12106. Lecture Notes in Computer Science. 2020, pp. 403–432. DOI: 10.1007/978-3-030-45724-2_14.

[5] Ken E. Batcher. "Sorting Networks and Their Applications". In: *Proceedings of the April 30–May 2, 1968 Spring Joint Computer Conference*. 1968, pp. 307–314. DOI: 10.1145/1468075.1468121.

[6] Henrik Blunck and Jan Vahrenhold. "In-Place Randomized Slope Selection". In: *Algorithms and Complexity*. Vol. 3998. Lecture Notes in Computer Science. 2006, pp. 30–41. DOI: 10.1007/11758471_6.

[7] Hervé Brönnimann and Bernard Chazelle. "Optimal Slope Selection via Cuttings". In: *Computational Geometry* 10.1 (1998), pp. 23–29. DOI: 10.1016/S0925-7721(97)00025-4.

[8] T.-H. Hubert Chan et al. "Cache-Oblivious and Data-Oblivious Sorting and Applications". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. 2018, pp. 2201–2220. DOI: 10.1137/1.9781611975031.143.

[9] Richard Cole et al. "An Optimal-Time Algorithm for Slope Selection". In: *SIAM Journal on Computing* 18.4 (1989), pp. 792–810. DOI: 10.1137/0218055.

[10] Michael B. Dillencourt, David M. Mount, and Nathan S. Netanyahu. "A Randomized Algorithm for Slope Selection". In: *International Journal of Computational Geometry & Applications* 2.1 (1992), pp. 1–27. DOI: 10.1142/S0218195992000020.

[11] Herbert Edelsbrunner and Ernst Peter Mücke. "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms". In: *ACM Transactions on Graphics* 9.1 (1990), pp. 66–104. DOI: 10.1145/77635.77639.

[12] David Eppstein, Michael T. Goodrich, and Roberto Tamassia. "Privacy-Preserving Data-Oblivious Geometric Algorithms for Geographic Data". In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2010, pp. 13–22. DOI: 10.1145/1869790.1869796.

[13] Oded Goldreich. "Towards a Theory of Software Protection and Simulation by Oblivious RAMs". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. 1987, pp. 182–194. DOI: 10.1145/28395.28416.

[14] Oded Goldreich and Rafail Ostrovsky. "Software Protection and Simulation on Oblivious RAMs". In: *Journal of the ACM* 43.3 (1996), pp. 431–473. DOI: 10.1145/233551.233553.

[15] Michael T. Goodrich. "Data-Oblivious External-Memory Algorithms for the Compaction, Selection, and Sorting of Outsourced Data". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*. 2011, pp. 379–388. DOI: 10.1145/1989493.1989555.

[16] Michael T. Goodrich. "Randomized Shellsort: A Simple Oblivious Sorting Algorithm". In: *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. 2010, pp. 1262–1277. DOI: 10.1137/1.9781611973075.101.

[17] Pavel Hubáček et al. "Stronger Lower Bounds for Online ORAM". In: *Theory of Cryptography*. Vol. 11892. 2019, pp. 264–284. DOI: 10.1007/978-3-030-36033-7_10.

[18] Matthew J. Katz and Micha Sharir. "Optimal Slope Selection via Expanders". In: *Information Processing Letters* 47.3 (1993), pp. 115–122. DOI: 10.1016/0020-0190(93)90234-Z.

[19] Donald Ervin Knuth. *Sorting and Searching*. Vol. 3. The Art of Computer Programming. 1973.

[20] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. "Efficient Oblivious Database Joins". In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 2132–2145. DOI: 10.14778/3407790.3407814.

[21] Kasper Green Larsen and Jesper Buus Nielsen. "Yes, There Is an Oblivious RAM Lower Bound!" In: *Advances in Cryptology*. Vol. 10992. Lecture Notes in Computer Science. 2018, pp. 523–542. DOI: 10.1007/978-3-319-96881-0_18.

[22] Yaping Li and Minghua Chen. "Privacy Preserving Joins". In: *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. 2008, pp. 1352–1354. DOI: 10.1109/ICDE.2008.4497553.

[23] Wei-Kai Lin, Elaine Shi, and Tiancheng Xie. *Can We Overcome the n log n Barrier for Oblivious Sorting?* 2018/227. 2018. URL: https://eprint.iacr.org/2018/227.

[24] Wei-Kai Lin, Elaine Shi, and Tiancheng Xie. "Can We Overcome the n log n Barrier for Oblivious Sorting?" In: *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms*. 2019, pp. 2419–2438. DOI: 10.1137/1.9781611975482.148.

[25] Chang Liu, Michael Hicks, and Elaine Shi. "Memory Trace Oblivious Program Execution". In: *2013 IEEE 26th Computer Security Foundations Symposium*. 2013, pp. 51–65. DOI: 10.1109/CSF.2013.11.

[26] Jiří Matoušek. "Randomized Optimal Algorithm for Slope Selection". In: *Information Processing Letters* 39.4 (1991), pp. 183–187. DOI: 10.1016/0020-0190(91)90177-J.

[27] Peter Bro Miltersen, Mike Paterson, and Jun Tarui. "The Asymptotic Complexity of Merging Networks". In: *Journal of the ACM* 43.1 (1996), pp. 147–165. DOI: 10.1145/227595.227693.

[28] Sajin Sasy and Olga Ohrimenko. "Oblivious Sampling Algorithms for Private Data Analysis". In: *Advances in Neural Information Processing Systems 32*. 2019, pp. 6495–6506. URL: http://papers.nips.cc/paper/8877-oblivious-sampling-algorithms-for-private-data-analysis.

[29] Stefan Schirra. "Precision and Robustness in Geometric Computations". In: *Algorithmic Foundations of Geographic Information Systems*. Vol. 1340. Lecture Notes in Computer Science. 1996, pp. 255–287. ISBN: 978-3-540-69653-7.

[30] Pranab Kumar Sen. "Estimates of the Regression Coefficient Based on Kendall's Tau". In: *Journal of the American Statistical Association* 63.324 (1968), pp. 1379–1389. DOI: 10.1080/01621459.1968.10480934.

[31] Elaine Shi. "Path Oblivious Heap: Optimal and Practical Oblivious Priority Queue". In: *Proceedings of the 2020 IEEE Symposium on Security and Privacy*. 2020, pp. 842–858. DOI: 10.1109/SP40000.2020.00037.

[32] Emil Stefanov and Elaine Shi. "ObliviStore: High Performance Oblivious Distributed Cloud Data Store". In: *Proceedings of the 20th Annual Network & Distributed System Security Symposium*. 2013. URL: https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/oblivistore-high-performance-oblivious-distributed-cloud-data-store/.

[33] Emil Stefanov et al. "Path ORAM: An Extremely Simple Oblivious RAM Protocol". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. 2013, pp. 299–310. DOI: 10.1145/2508859.2516660.

[34] Chee-Keng Yap. "A Geometric Consistency Theorem for a Symbolic Perturbation Scheme". In: *Journal of Computer and System Sciences* 40.1 (1990), pp. 2–18. DOI: https://doi.org/10.1016/0022-0000(90)90016-E.

Any Regular Polyhedron Can Transform to Another by $O(1)$ Refoldings

Erik D. Demaine* Martin L. Demaine* Yevhenii Diomidov*
Tonan Kamata† Ryuhei Uehara† Hanyu Alice Zhang‡

Abstract

We show that several classes of polyhedra are joined by a sequence of $O(1)$ refolding steps, where each refolding step unfolds the current polyhedron (allowing cuts anywhere on the surface and allowing overlap) and folds that unfolding into exactly the next polyhedron; in other words, a polyhedron is refoldable into another polyhedron if they share a common unfolding. Specifically, assuming equal surface area, we prove that (1) any two tetramonohedra are refoldable to each other, (2) any doubly covered triangle is refoldable to a tetramonohedron, (3) any (augmented) regular prismatoid and doubly covered regular polygon is refoldable to a tetramonohedron, (4) any tetrahedron has a 3-step refolding sequence to a tetramonohedron, and (5) the regular dodecahedron has a 4-step refolding sequence to a tetramonohedron. In particular, we obtain a $\leq$ 6-step refolding sequence between any pair of Platonic solids, applying (5) for the dodecahedron and (1) and/or (2) for all other Platonic solids. As far as the authors know, this is the first result about common unfolding involving the regular dodecahedron.

1 Introduction

A polyhedron Q is ***refoldable*** to a polyhedron Q' if Q can be unfolded to a planar shape that folds into exactly the surface of Q', i.e., Q and Q' share a common unfolding/development, allowing cuts anywhere on the surfaces of Q and Q'. (Although it is probably not necessary for our refoldings, we also allow the common unfolding to self-overlap, as in [6].) The idea of refolding was proposed independently by M. Demaine, F. Hurtado, and E. Pegg [5, Open Problem 25.6], who specifically asked whether every regular polyhedron (Platonic solid) can be refolded into any other regular polyhedron. In this context, there exist some specific results: Araki et al. [2] found two Johnson-Zalgaller solids that are foldable to regular tetrahedra [2], and Shirakawa et al. [9] found an infinite sequence of polygons that can each fold into a cube and an approaching-regular tetrahedron.

More broadly, Demaine et al. [4] showed that any convex polyhedron can always be refolded to at least one other convex polyhedron. Xu et al. [11] and Biswas and Demaine [3] found common unfoldings of more than two (specific) polyhedra. On the negative side, Horiyama and Uehara [6] proved impossibility of certain refoldings when the common unfolding is restricted to cut along the edges of polyhedra.

In this paper, we consider the connectivity of polyhedra by the transitive closure of refolding, an idea suggested by Demaine and O'Rourke [5, Section 25.8.3]. Define a ***(k-step) refolding sequence*** from Q to Q' to be a sequence of convex polyhedra $Q = Q_0, Q_1, \ldots, Q_k = Q'$ where each Q_{i-1} is refoldable to Q_i. We refer to k as the ***length*** of the refolding sequence. To avoid confusion, we use "1-step refoldable" to refer to the previous notion of refoldability.

Our results. Do all pairs of convex polyhedra of the same surface area (a trivial necessary condition) have a finite-step refolding sequence? If so, how short of a sequence suffices? As mentioned in [5, Section 25.8.3], the regular polyhedron open problem mentioned above is equivalent to asking whether 1-step refolding sequences exist for all pairs of regular polyhedra. We solve a closely related problem, replacing "1" with "$O(1)$": for any pair of regular polyhedra Q and Q', we give a refolding sequence of length at most 6.

More generally, we give a series of results about $O(1)$-step refolding certain pairs of polyhedra of the same surface area:

1. In Section 3, we show that any two tetramonohedra are 1-step refoldable to each other, where a ***tetramonohedron*** is a tetrahedron that consists of four congruent acute triangles.

   This result offers a possible "canonical form" for finite-step refolding sequences between any two polyhedra: because a refolding from Q to Q' is also a refolding from Q' to Q, it suffices to show that any polyhedron has a finite-step refolding into some tetramonohedron.

*CSAIL, MIT, USA. {edemaine,mdemaine,diomidov}@mit.edu
†School of Information and Science, Japan Advanced Institute of Science and Technology, Japan. {kamata,uehara}@jaist.ac.jp
‡School of Applied and Engineering Physics, Cornell University, USA. hz496@cornell.edu

2. In Section 4, we show that every regular prismatoid and every augmented regular prismatoid are 1-step refoldable to a tetramonohedron.

   In particular, the regular tetrahedron is a tetramonohedron, the regular hexahedron (cube) is a regular prismatoid, and the regular octahedron and regular icosahedron are both augmented regular prismatoids. Therefore, the regular tetrahedron has a 2-step refolding sequence to the regular hexahedron, octahedron, and icosahedron (via an intermediate tetramonohedron); and every pair of polyhedra among the regular hexahedron, octahedron, and icosahedron have a 3-step refolding sequence (via two intermediate tetramonohedra).

3. In Section 5, we prove that a regular dodecahedron is refoldable to a tetramonohedron by a 4-step refolding sequence.

   As far as the authors know, there are no previous explicit refolding results for the regular dodecahedron, except the general results of [4].

   Combining the results above, any pair of regular polyhedra (Platonic solids) have a refolding sequence of length at most 6.

4. In addition, we prove that every doubly covered triangle (Section 3) and every doubly covered regular polygon (Section 4) are refoldable to a tetramonohedron, and that every tetrahedron has a 3-step refolding sequence to a tetramonohedron (Section 6).

   Therefore, every pair of polyhedra among the list above have an $O(1)$-step refolding sequence.

2 Preliminaries

For a polyhedron Q, $V(Q)$ denotes the set of vertices of Q. For $v \in V(Q)$, define the ***cocurvature*** $\sigma(v)$ of v on Q to be the sum of the angles incident to v on the facets of Q. The ***curvature*** $\kappa(v)$ of v is defined by $\kappa(v) = 2\pi - \sigma(v)$. In particular, if $\kappa(v) = \sigma(v) = \pi$, we call v a ***smooth vertex***. We define $\boldsymbol{\Pi_k}$ to be the class of polyhedra Q with exactly k smooth vertices. It is well-known that the total curvature of the vertices of any convex polyhedron is 4π, by the Gauss–Bonnet Theorem (see [5, Section 21.3]). Thus the number of smooth vertices of a convex polyhedron is at most 4. Therefore, the classes $\Pi_0, \Pi_1, \Pi_2, \Pi_3, \Pi_4$ give us a partition of all convex polyhedra.

An ***unfolding*** of a polyhedron is a (possibly self-overlapping) planar polygon obtained by cutting and developing the surface of the polyhedron (allowing cuts anywhere on the surface). ***Folding*** a polygon P is an operation to obtain a polyhedron Q by choosing crease lines on P and gluing the boundary of P properly. When the polyhedron Q is convex, the following result is crucial:

Lemma 1 (Alexandrov's Theorem [8, 5]) *If we fold a polygon P in a way that satisfies the following three **Alexandrov's conditions**, then there is a unique convex polyhedron Q realized by the folding.*

1. *Every point on the boundary of P is used in the gluing.*
2. *At any glued point, the summation of interior angles (cocurvature) is at most 2π.*
3. *The obtained surface is homeomorphic to a sphere.*

By this result, when we fold a polygon P to a polyhedron Q, it is enough to check that the gluing satisfies Alexandrov's conditions. (In this paper, it is easy to check that the conditions are satisfied by our (re)foldings, so we omit their proof.)

A polyhedron Q is ***(1-step) refoldable*** to a polyhedron Q' if Q can be unfolded to a polygon that folds to Q' (and thus they have the same surface area). A ***(k-step) refolding sequence*** of a polyhedron Q to a polyhedron Q' is a sequence of convex polyhedra $Q = Q_0, Q_1, \ldots, Q_k = Q'$ where Q_{i-1} is refoldable to Q_i for each $i \in \{1, \ldots, k\}$. To simplify some arguments that Q is refoldable to Q', we sometimes only partially unfold Q (cutting less than needed to make the surface unfold flat), and refold to Q' so that Alexandrov's conditions hold.

We introduce some key polyhedra. A tetrahedron is a ***tetramonohedron*** if its faces are four congruent acute triangles.[1] We consider a ***doubly covered polygon*** as a special polyhedron with two faces. Precisely, for a given n-gon P, we make a mirror image P' of P and glue corresponding edges. Then we obtain a ***doubly covered n-gon*** which has 2 faces, n edges, and zero volume.

3 Refoldabilty of Tetramonohedra and Doubly Covered Triangles

In this section, we first show that any pair of tetramonohedra can be refolded to each other. We note that a doubly covered rectangle is a (degenerate) tetramonohedron, by adding edges along two crossing diagonals (one on the front side and one on the back side). It is known that a polyhedron is a tetramonohedron if and only if it is in Π_4 [7]. In other words, Π_4 is the set of tetramonohedra.

Theorem 2 *For any $Q, Q' \in \Pi_4$, Q is 1-step refoldable to Q'.*

[1]This notion is also called ***isotetrahedron*** in some literature.

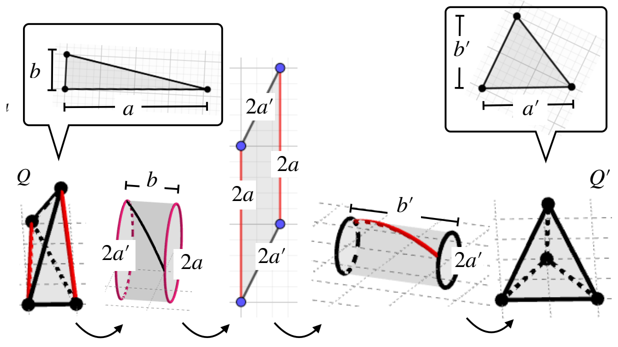


Figure 1: A refolding between two tetramonohedra

Proof. Let T be any triangular face of Q. Let a be the length of the longest edge of T and b the height of T for the base edge of length a. We define T', a', and b' in the same manner for Q'; refer to Figure 1. We assume $a > a'$ without loss of generality. Now we have $a' > b'$ because a' is the longest edge of T', and $a'b' = ab$ because T and T' are of the same area. Thus, $(a')^2 = a'b'\frac{a'}{b'} > a'b' = ab$, and $2a' > a' > b$ by $a > a'$.

We cut two edges of Q of length a, resulting in a cylinder of height b and circumference $2a$. Then we can cut the cylinder by a segment of length $2a'$ because $2a' > b$. The resulting polygon is a parallelogram such that two opposite sides have length $2a$ and the other two opposite sides have length $2a'$. Now we glue the sides of length $2a$ and obtain a cylinder of height b' and circumference $2a'$. Then we can obtain Q' by folding this cylinder suitably (the opposite of cutting two edges of Q' of length $2a'$). $\square$

To complement the doubly covered rectangles handled by Theorem 2, we give a related result for doubly covered triangles:

Theorem 3 *Any doubly covered triangle Q is 1-step refoldable into a doubly covered rectangle. Thus, Q has a refolding sequence to any doubly covered triangle Q' of length at most 3. If doubly covered triangles Q and Q' share at least one edge length, then the sequence has length at most 2.*

Proof. Let Q consist of a triangle T and its mirror image T'. We first cut Q along any two edges, and unfold along the remaining attached edge, resulting in a quadrilateral unfolding as shown in Figure 2. Let b be the length of the uncut edge, which we call the ***base***, and let h be the height of T with respect to the base. Let p and q be the midpoints of the two cut edges. Then the line segment pq is parallel to the base and of length $b/2$. In the unfolding of Q, let p' and q' be the mirrors of p and q, respectively. Then we can draw a grid based on the rectangle $pp'q'q$ as shown in Figure 2. By folding

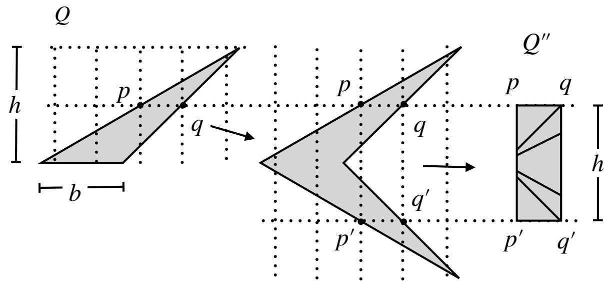


Figure 2: A refolding from a doubly covered triangle to a doubly covered rectangle

along the crease lines defined by the grid, we can obtain a doubly covered rectangle Q'' of size $b/2 \times h$ (matching the doubled surface area of Q). (Intuitively, this folding wraps T and T' on the surface of the rectangle $pp'q'q$.)

Because Q'' is also a tetramonohedron, the second claim follows from Theorem 2. When Q has an edge of the same length as an edge of Q', as in the third claim, we can cut the other two edges of Q and Q' to obtain the same doubly covered rectangle, resulting in a 2-step refolding sequence. $\square$

The technique in the proof of Theorem 3 works for any doubly covered triangle Q even if its faces are acute or obtuse triangles.

4 Refoldability of a Regular Prismatoid to a Tetramonohedron

In this section, we give a 1-step refolding of any regular prism or prismatoid to a tetramonohedron. We extend the approach of Horiyama and Uehara [6], who showed that the regular icosahedron, the regular octahedron, and the regular hexahedron (cube) can be 1-step refolded into a tetramonohedron. As an example, Figure 3 shows their common unfolding for the regular icosahedron.

A polygon $P = (p_0, c_1, p_1, c_2, p_2, \ldots, p_{2n}, c_{2n}, p_{2n+1}, p_0)$ is called a ***spine polygon*** if it satisfies the following two conditions (refer to Figure 4):

1. Vertex p_i is on the line segment p_0p_n for each $0 < i < n$; vertex p_i is on the line segment $p_{n+1}p_{2n+1}$ for each $n+1 < i < 2n+1$; and the polygon $B = (p_0, p_n, p_{n+1}, p_{2n+1}, p_0)$ is a parallelogram. We call B the ***base*** of P, and require it to have positive area.

2. The polygon $T_i = (p_i, c_{i+1}, p_{i+1}, p_i)$ is an isosceles triangle for each $0 \le i \le n-1$ and $n+1 \le i \le 2n$. The triangles $T_0, T_1, \ldots, T_{n-1}$ are congruent, and $T_{n+1}, T_{n+2}, \ldots, T_{2n}$ are also congruent. These triangles are called ***spikes***.

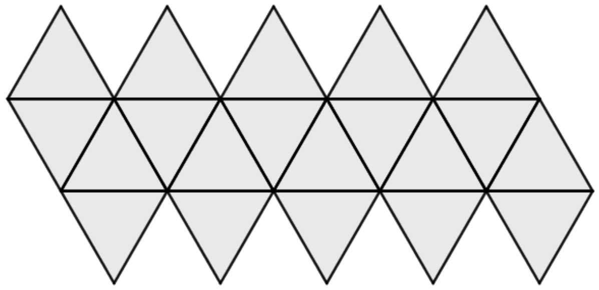


Figure 3: A common unfolding of a regular icosahedron and a tetramonohedron, from [6]

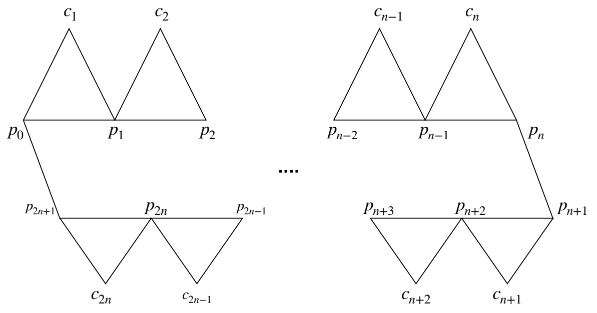


Figure 4: A spine polygon with $2n$ spikes

Lemma 4 *Any spine polygon P can be folded to a tetramonohedron.*

Proof. Akiyama and Matsunaga [1] prove that a polygon P can be folded into a tetramonohedron if the boundary of P can be divided into six parts, two of which are parallel and the other four of which are rotationally symmetric. We divide the boundary of a spine polygon P into $l_1 = (p_0, c_1, \ldots, c_n)$; $l_2 = (c_n, p_n)$, $l_3 = (p_n, p_{n+1})$; $l_4 = (p_{n+1}, c_{n+1}, \ldots, c_{2n})$, $l_5 = (c_{2n}, p_{2n+1})$; and $l_6 = (p_{2n+1}, p_0)$. Then l_3 and l_6 are parallel because the base of P is a parallelogram. Each of l_2 and l_5 is trivially rotationally symmetric. Each of l_1 and l_4 is rotationally symmetric because each spike of P is an isosceles triangle. ☐

Now we introduce some classes of polyhedra; refer to Figure 5.

A ***prismatoid*** is the convex hull of parallel ***base*** and ***top*** convex polygons. We sometimes call the base and the top ***roofs*** when they are not distinguished. We call a prismatoid ***regular*** if (1) its base P_1 and top P_2 are congruent regular polygons and (2) the line passing through the centers of P_1 and P_2 is perpendicular to P_1 and P_2. (Note that the side faces of a regular prismatoid do not need to be regular polygons.) The perpendicular distance between the planes containing P_1 and P_2 is the ***height*** of the prismatoid. The set of regular prismatoids contains ***prisms*** and ***antiprisms***, as well as doubly covered regular polygons (prisms of height zero).

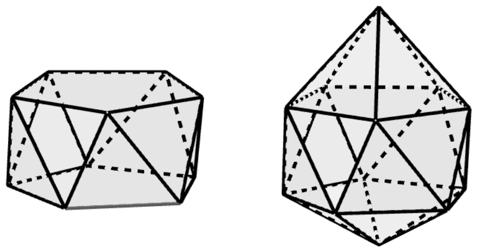


Figure 5: A regular prismatoid and an augmented regular prismatoid

A ***pyramid*** is the convex hull of a ***base*** convex polygon and an ***apex*** point. We call a pyramid ***regular*** if the base polygon is a regular polygon, and the line passing through the apex and the center of the base is perpendicular to the base. (Note that the side faces of a regular pyramid do not need to be regular polygons.) A polyhedron is an ***augmented regular prismatoid*** if it can be obtained by attaching two regular pyramids to a regular prismatoid base-to-roof, where the bases of the pyramids are congruent to the roofs of the prismatoid and each roof is covered by the base of one of the pyramids.

Theorem 5 *Any regular prismatoid or augmented regular prismatoid of positive volume can be unfolded to a spine polygon.*

Proof. Let Q be a regular prismatoid. Let c_1 and c_2 be the center points of two roofs P_1 and P_2, respectively. Cutting from c_i to all vertices of P_i for each $i = 1, 2$ and cutting along a line joining between any pair of vertices of P_1 and P_2, we obtain a spine polygon. For an augmented regular prismatoid Q, we can similarly cut from the apex c_i of each pyramid to the other vertices of the pyramid, which are the vertices of the roof P_i of the prismatoid. ☐

When the height of the regular prismatoid is zero (or it is a doubly covered regular polygon), the proof of Theorem 5 does not work because the resulting polygon is not connected. In this case, we need to add some twist.

Theorem 6 *Any doubly covered regular n-gon is 1-step refoldable to a tetramonohedron for $n > 2$.*

Proof. First suppose that n is an even number $2k$ for some positive integer $k > 1$. We consider a special spine polygon where the top angles are $\frac{2\pi}{k}$; the vertices p_0, p_{2n+1}, p_1 are on a circle centered at c_1; and the vertices p_{2n+1}, p_1, p_2 are on a circle centered at c_{2n}; see Figure 6. Then we can obtain a doubly covered n-gon by folding along the zig-zag path

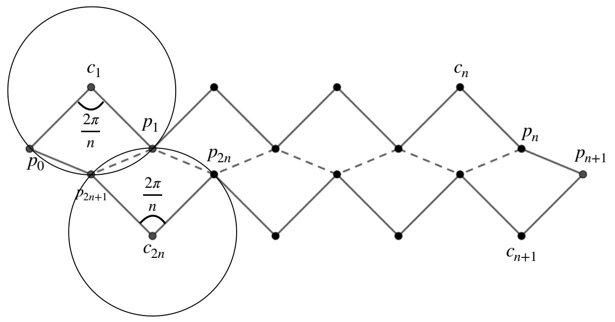


Figure 6: The case of a doubly covered regular 8-gon

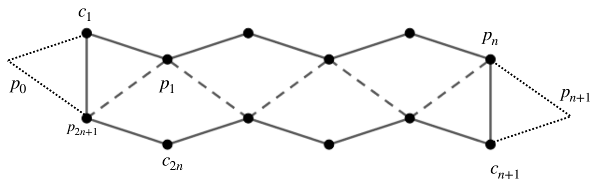


Figure 7: The case of a doubly covered regular 5-gon

$p_{2n+1}, p_1, p_{2n}, p_2, \ldots, p_{n+2}, p_n$ shown in Figure 6. Thus when $n = 2k$ for some positive integer k, we obtain the theorem.

Now suppose that n is an odd number $2k + 1$ for some positive integer k. We consider the spine polygon whose top angles are $\frac{4\pi}{2k+1}$; the vertices p_0, p_{2n+1}, p_1 are on a circle centered at c_1; and the vertices p_{2n+1}, p_1, p_2 are on a circle centered at c_{2n}. From this spine polygon, we cut off two triangles c_1, p_0, c_{2n+1} and c_{n+1}, p_{n+1}, p_n, as in Figure 7. Then we can obtain a doubly covered n-gon by folding along the zig-zag path $p_{2n+1}, p_1, p_{2n}, p_2, \ldots, p_{n+2}, p_n$ shown in Figure 7. Although the unfolding is no longer a spine polygon, it is easy to see that it can also fold into a tetramonohedron by letting $l'_1 = (c_1, p_1, \ldots, p_n)$, $l'_2 = (p_n, p_n)$, $l'_3 = (p_n, c_{n+1})$, $l'_4 = (c_{n+1}, p_{n+2} \ldots, p_{2n+1})$, $l'_5 = (p_{2n+1}, p_{2n+1})$, and $l'_6 = (p_{2n+1}, c_1)$ in the proof of Lemma 4. $\square$

The proof of Theorem 6 is effectively exploiting that a doubly covered regular $2k$-gon (with $k > 1$) can be viewed as a degenerate regular prismatoid with two k-gon roofs, where each of the side triangles of this prismatoid is on the plane of the roof sharing the base of the triangle.

Because the cube and the regular octahedron are regular prismatoids and the regular icosahedron is an augmented regular prismatoid, we obtain the following:

Corollary 7 *Let Q and Q' be regular polyhedra of the same area, neither of which is a regular dodecahedron. Then there exists a refolding sequence of length at most 3 from Q to Q'. When one of Q or Q' is a regular tetrahedron, the length of the sequence is at most 2.*

5 Refoldability of a Regular Dodecahedron to a Tetramonohedron

In this section, we show that there is a refolding sequence of the regular dodecahedron to a tetramonohedron of length 4. Combining this result with Corollary 7, we obtain refolding sequences between any two regular polyhedra of length at most 6.

Demaine et al. [4] mention that the regular dodecahedron can be refolded to another convex polyhedron. Indeed, they show that any convex polyhedron can be refolded to at least one other convex polyhedron using an idea called "flipping a Z-shape". We extend this idea.

Definition 1 For a convex polyhedron Q and $n, k \in \mathbb{N}$, let $p = (s_1, s_2, \ldots, s_{(2k+1)n})$ be a path that consists of isometric and non-intersecting $(2k + 1)n$ straight line segments s_i on Q. We cut the surface of Q along p. Then each line segment is divided into two line segments on the boundary of the cut. For each line segment s_i, let s^l_i and s^r_i correspond to the left and right sides on the boundary along the cut (Figure 8). Then p is a ***Z-flippable* (n, k)*-path*** on Q, and Q is ***Z-flippable*** by p, if the following gluing satisfies Alexandrov's conditions.

- Glue $s^l_1, s^l_2, \ldots, s^l_n$ to $s^l_{2n}, s^l_{2n-1}, \ldots, s^l_{n+1}$.

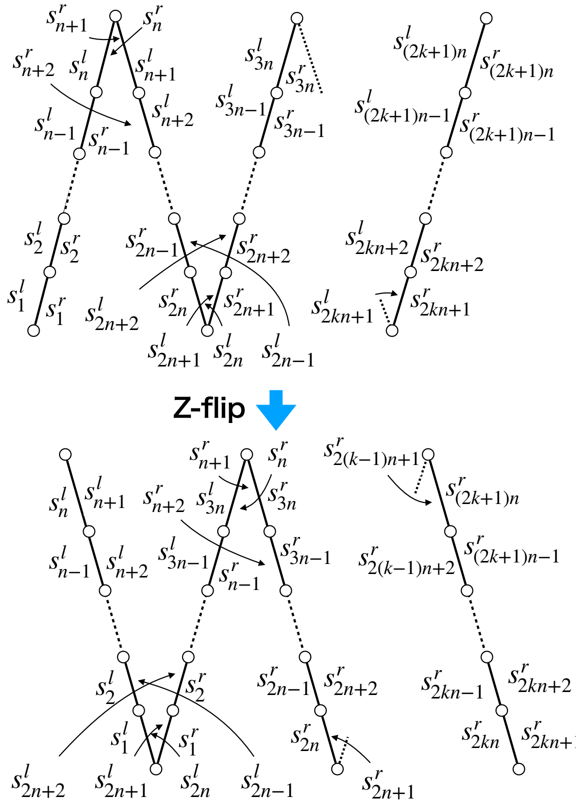


Figure 8: Z-flip

- Glue $s^r_1, s^r_2, \ldots, s^r_n$ to $s^l_{2n+1}, s^l_{2n+2}, \ldots, s^l_{3n}$.
- Glue $s^r_{n+1}, s^r_{n+2}, \ldots, s^r_{2n}$ to $s^r_{3n}, s^r_{3n-1}, \ldots, s^r_{2n+1}$.

$$\vdots$$

- Glue $s^r_{2(k-1)n+1}, s^r_{2(k-1)n+2}, \ldots, s^r_{2kn}$ to $s^r_{(2k+1)n}, s^r_{(2k+1)n-1}, \ldots, s^r_{2kn+1}$.

If there are Z-flippable paths $p^1, p^2, \ldots, p^m$ inducing a forest on Q, we can flip them all at the same time. Then we say that Q is ***Z-flippable*** by $p^1, p^2, \ldots, p^m$.

Theorem 8 *There exists a 4-step refolding sequence between a regular dodecahedron and a tetramonohedron.*

Proof. Let D be a regular dodecahedron. To simplify, we assume that each edge of a regular pentagon is of length 1. We show that there exists a refolding sequence D, Q_1, Q_2, Q_3, Q_4 of length 4 for a tetramonohedron Q_4.

All cocurvatures of the vertices of D are equal to $\frac{9\pi}{5}$. For any vertices v, there are 3 vertices of distance 1 from v and 6 vertices of distance $\phi = \frac{1+\sqrt{5}}{2}$ from v. Hereafter, in figures, each circle describes a non-flat vertex on a polyhedron and the number in the circle describes its cocurvature divided by $\frac{\pi}{5}$. Each pair of vertices of distance 1 is connected by a solid line, and each pair of vertices of distance ϕ is connected by a dotted line. Figure 9 shows the initial state of D in this notation. We note that solid and dotted lines do not necessarily imply edges (or crease lines) on the polyhedron.

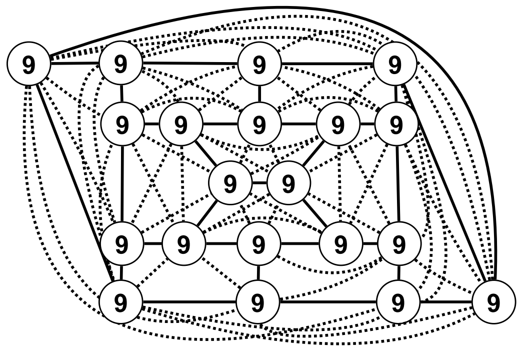


Figure 9: The initial regular dodecahedron

First, we choose $p^1 = (s^1_1, s^1_2, \ldots, s^1_6)$, $p^2 = (s^2_1, s^2_2, s^2_3)$, $p^3 = (s^3_1, s^3_2, \ldots, s^3_6)$, and $p^4 = (s^4_1, s^4_2, s^4_3)$ on the surface of D on the left of Figure 10. Then, p^1 and p^3 are Z-flippable $(2,1)$-paths and p^2 and p^4 are Z-flippable $(1,1)$-paths. Thus, D is Z-flippable by p^1, p^2, p^3, p^4 to the polyhedron on the right of Figure 10. Let Q_1 be the resulting polyhedron.

Second, we choose $p^1 = (s^1_1, s^1_2, \ldots, s^1_5)$ on the surface of Q_1 on the left of Figure 11. Then, p^1 is a Z-flippable $(1,3)$-path. Thus, Q_1 is Z-flippable by p^1 to the next polyhedron Q_2 on the right of Figure 11.

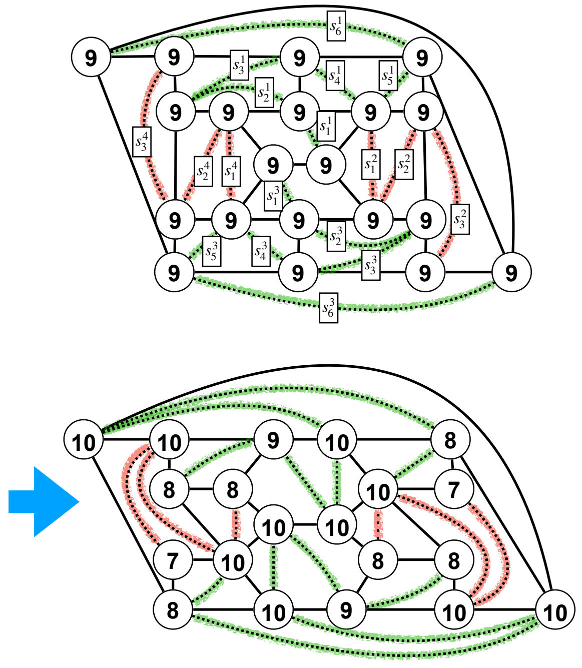

s^1_1 s^1_2 s^1_3 s^1_4 s^1_5 s^1_6 s^2_1 s^2_2 s^2_3 s^3_1 s^3_2 s^3_3 s^3_4 s^3_5 s^3_6 s^4_1 s^4_2 s^4_3

Figure 10: A refolding from D to Q_1

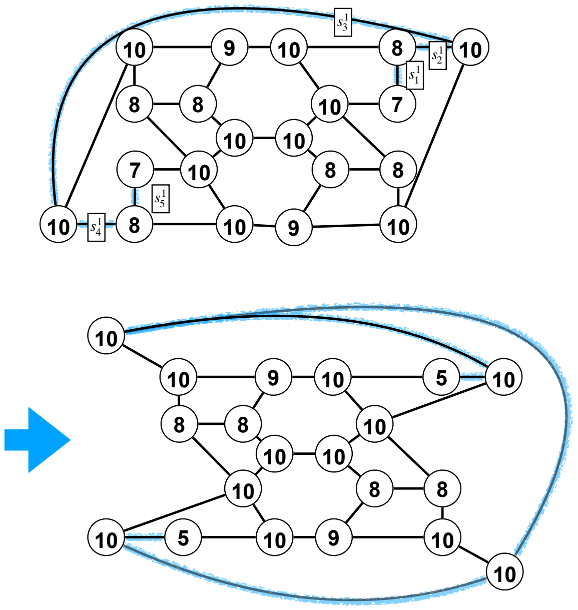


Figure 11: A refolding from Q_1 to Q_2

Third, we choose $p^1 = (s^1_1, s^1_2, s^1_3)$ and $p^2 = (s^2_1, s^2_2, s^2_3)$ on the surface of Q_2 on the left of Figure 12. Then, p^1 and p^2 are Z-flippable $(1,1)$-paths. Thus, Q_2 is Z-flippable by p^1 and p^2 to the polyhedron Q_3 on the right of Figure 12.

Fourth, we choose $p^1 = (s^1_1, s^1_2, \ldots, s^1_5)$, $p^2 = (s^2_1, s^2_2, \ldots, s^2_5)$, and $p^3 = (s^3_1, s^3_2, s^3_3)$ on the surface of

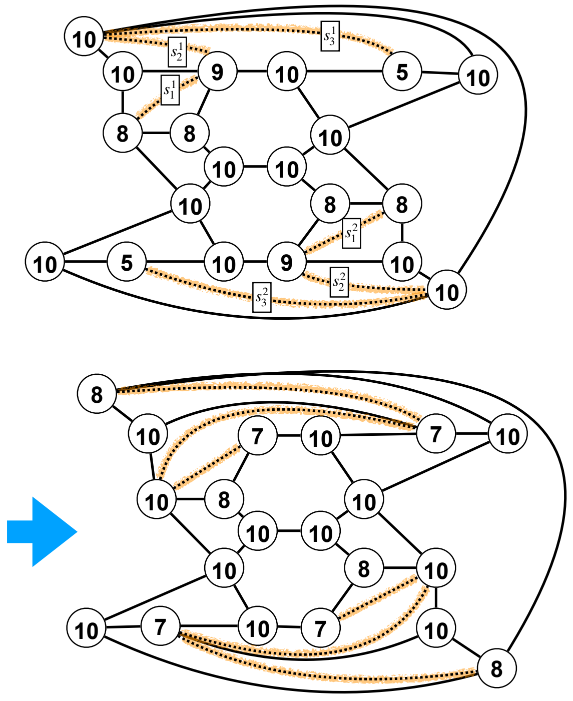


Figure 12: A refolding from Q_2 to Q_3

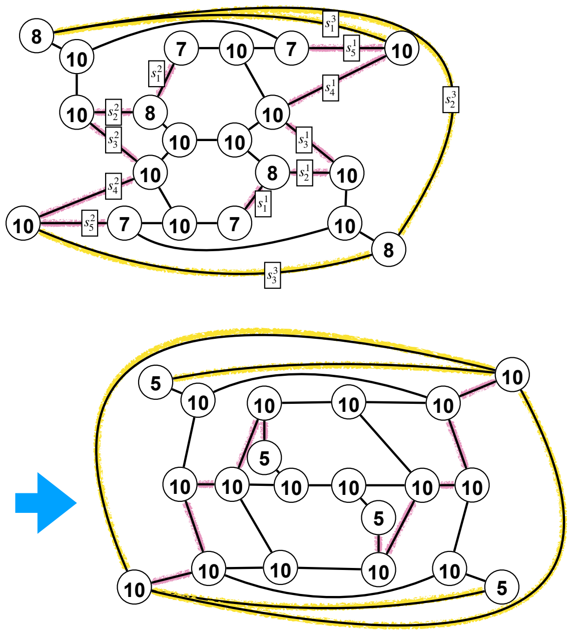


Figure 13: A refolding from Q_3 to Q_4

Q_3 on the left of Figure 13. Then, p^1 and p^2 are Z-flippable $(1,3)$-paths and p^3 is a Z-flippable $(1,1)$-path. Thus, Q_3 is Z-flippable by p^1, p^2, and p^3 to the polyhedron Q_4 on the right of Figure 13. Finally, we obtain a tetramonohedron Q_4 from a regular dodecahedron D by a 4-step refolding sequence.

In this proof, we used partial unfolding between pairs

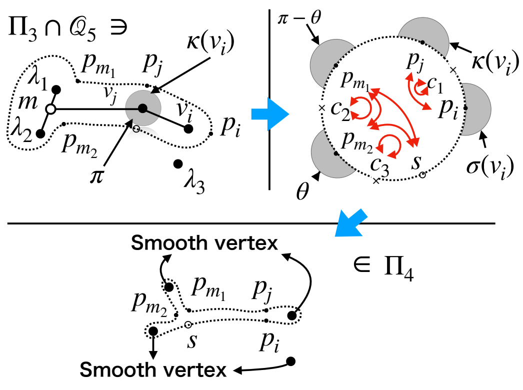


Figure 14: A refolding of a polyhedron in $\Pi_3 \cap \mathcal{Q}_5$ to Π_4

of polyhedra in the refolding sequence. Appendix A gives the (fully unfolded) common unfoldings. $\square$

6 Refoldability of a Tetrahedron to a Tetramonohedron

In this section, we prove that any tetrahedron can be refolded to a tetramonohedron. Let $\mathcal{Q}_k$ denote the class of polyhedra with exactly k vertices.

6.1 Refoldability of Π_3 to Π_4

First we show a technical lemma: any polyhedron Q in Π_3 can be refolded to a tetramonohedron by a refolding sequence of length linear in the number of vertices of Q.

Lemma 9 *For any $Q \in \Pi_3 \cap \mathcal{Q}_n$ with $n \geq 5$, there is a refolding sequence of length $2n-9$ from Q to some $Q' \in \Pi_4$.*

Proof. (Outline) We prove the claim by induction. As the base case, suppose $n = 5$; refer to Figure 14. Let $\lambda_1, \lambda_2, \lambda_3$ be the smooth vertices of Q and v_i, v_j be the other vertices. Take a point m on the segment $\lambda_1\lambda_2$ with $\angle v_iv_jm = \kappa(v_i)$ and cut the surface of Q along the segments $\lambda_1\lambda_2$, v_iv_j, and v_jm. Then $\sigma(v_j) - \kappa(v_i) = \pi$ because $\kappa(v_i) + \kappa(v_j) = \pi$. The point v_j is then divided into a point of degree $\kappa(v_i)$ and a point of degree π on the boundary. Trace the obtained boundary from v_i counterclockwise and denote points corresponding to v_i, v_j, m, m by $p_i, p_j, p_{m_1}, p_{m_2}$, respectively. Let c_1 and c_2 be the center points of the segments p_ip_j and $p_{m_1}p_{m_2}$, respectively. We take the point s which has the same distance with p_{m_1} from c_1. Let c_3 be the center of the segment $p_{m_2}s$. Now glue the segment sc_1 to $c_1p_{m_1}$, the segment $p_{m_2}c_2$ to $c_2p_{m_1}$, and the segment $p_{m_2}c_3$ to c_3, s. Let Q' be the obtained polyhedron after the gluing. Then Q' is in Π_4 because each curvature of every vertex of Q' is π.

Now we turn to the inductive step. Let Q be any polyhedron in $\Pi_3 \cap \mathcal{Q}_k$ with $k > 5$. We prove that there

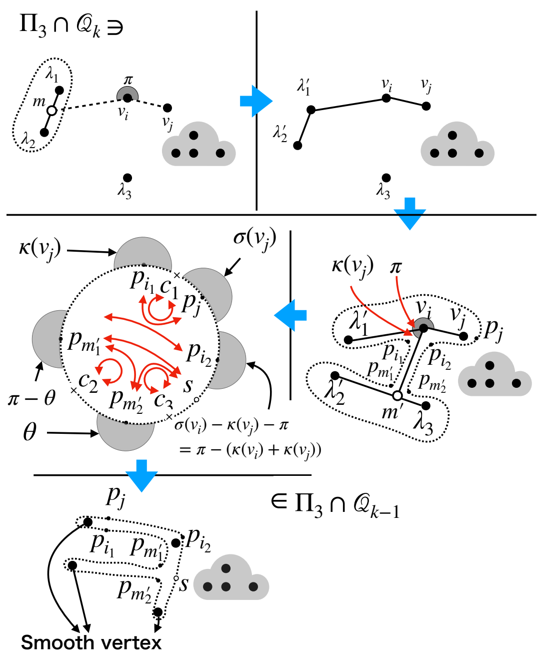

$\Pi_3 \cap \mathcal{Q}_k \ni$
Smooth vertex
$\in \Pi_3 \cap \mathcal{Q}_{k-1}$
$\sigma(v_i) - \kappa(v_j) - \pi$
$= \pi - (\kappa(v_i) + \kappa(v_j))$

Figure 15: A refolding sequence from $\Pi_3 \cap \mathcal{Q}_k$ to $\Pi_3 \cap \mathcal{Q}_{k-1}$

exists a refolding sequence (Q, Q', Q'') for two polyhedra Q' and Q'' with $Q'' \in \Pi_3 \cap \mathcal{Q}_{k-1}$; refer to Figure 15.

Let $\lambda_1, \lambda_2, \lambda_3$ be the smooth vertices of Q and v_i, v_j be any other vertices on Q. Note that $0 < \kappa(v_i)+\kappa(v_j) < \pi$ because $k > 5$. Take a point m on the segment $\lambda_1\lambda_2$ with $\angle v_i v_j m = \pi$, cut the surface of Q' along the segment $\lambda_1\lambda_2$, and glue it again so that m is an endpoint. (This can be done because the cut produces a "rolling belt" in terms of folding; see [5] for the details.) Let Q' be the obtained polyhedron, and let $\lambda'_1(= m), \lambda'_2, \lambda'_3(= \lambda_3)$ be the smooth vertices of Q'. Now take a point m' on the segment λ'_2, λ'_3 such that $\angle \lambda'_1 v_i m' = \kappa(v_j)$ and cut the surface of Q' along the segments $\lambda'_1 v_i$, $\lambda'_2\lambda'_3$, $v_i v_j$, and $v_i m'$. Trace the obtained boundary from v_j counterclockwise and denote points corresponding to v_j, v_i, m', m', v_i by $p_j, p_{i_1}, p_{m'_1}, p_{m'_2}, p_{i_2}$, respectively. Let c_1 and c_2 be the midpoints of $p_{i_1}p_j$ and $p_{m'_1}p_{m'_2}$, respectively. Take the point s which has the same distance with $p_{m'_1}$ from c_1. Let c_3 be the midpoint of the segment $p_{m'_2}s$. Now glue the segment $p_{m'_1}c_1$ to $c_1 s$, the segment $p_{m'_2}c_2$ to $c_2 p_{m'_1}$, and the segment $p_{m'_2}c_3$ to $c_3 s$. Let Q'' be the obtained polyhedron. Then Q'' is in $\Pi_3 \cap \mathcal{Q}_{k-1}$ because each curvature of points corresponding to c_1, c_2, c_3 is π and $0 < \sigma(v'_4) < \pi$ because $\sigma(v'_4) = \sigma(v_i) - \kappa(v_j) = \sigma(v_i) + \sigma(v_j) - 2\pi$ and $0 < \kappa(v_i) + \kappa(v_j) < \pi$. □

Theorem 10 *For any $Q \in \mathcal{Q}_4$, there is a 3-step refolding sequence from Q to some $Q''' \in \Pi_4$.*

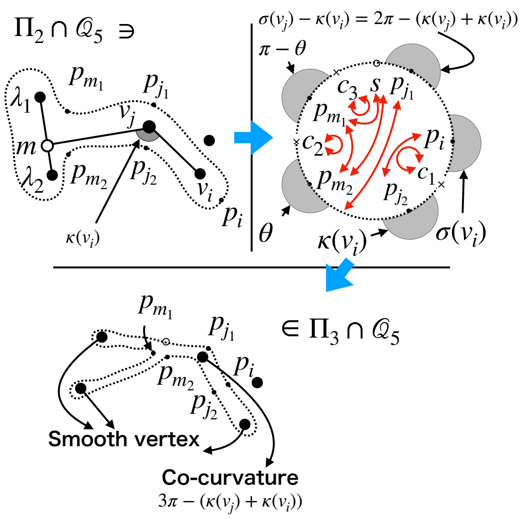

$\Pi_2 \cap \mathcal{Q}_5 \ni$
$\sigma(v_j) - \kappa(v_i) = 2\pi - (\kappa(v_j) + \kappa(v_i))$
$\in \Pi_3 \cap \mathcal{Q}_5$
Smooth vertex
Co-curvature
$3\pi - (\kappa(v_j) + \kappa(v_i))$

Figure 16: Refolding from any tetrahedron to a tetramonohedron

Proof. (Outline) Let v, v' be two vertices of Q with smallest cocurvature. (That is, $\sigma(v), \sigma(v') \leq \sigma(v'')$ for the other two vertices v'' of Q.) We cut along the segment vv' and glue the point v to v'. Let Q' be the resulting polyhedron. Then, because $\sigma(v) + \sigma(v') \leq 2\pi$ by the Gauss–Bonnet Theorem, Q' satisfies the Alexandrov's conditions. That is, Q' is a convex polyhedron in $\Pi_2 \cup \mathcal{Q}_5$. (We assume that the original Q has no smooth vertex to simplify the arguments.)

Let λ_1 and λ_2 be the two smooth vertices of Q' (which was generated by the gluing of v and v'), and v_i, v_j be two vertices of Q' of larger cocurvature than others with $\kappa(v_i) < \kappa(v_j)$. By the Gauss–Bonnet Theorem, $\frac{4\pi}{3} \leq \sigma(v_i) + \sigma(v_j) \leq 2\pi$. We take the point m on the segment $\lambda_1\lambda_2$ so that $\angle(v_i, v_j, m) = \kappa(v_i)$. Now we cut along the segments $\lambda_1\lambda_2$, $v_i v_j$, and $v_j m$; see Figure 16. Trace the obtained boundary from v_i counterclockwise and denote points corresponding to v_i, v_j, m, m, v_j by $p_i, p_{j_1}, p_{m_1}, p_{j_2}, p_{m_2}$, respectively. Let c_1 and c_2 be the midpoints of the segments $p_i p_{j_2}$ and $p_{m_1}p_{m_2}$, respectively. Furthermore, we take the point s which has the same distance with p_{m_2} from c_1, and let c_3 be the midpoint of sp_{m_1}.

Now glue segment $p_i c_1$ to $p_{j_2} c_1$, segment $p_i s$ to $p_{j_2} p_{m_2}$, segment $p_{m_1} c_2$ to $p_{m_2} c_2$, and segment sc_3 to $p_{m_1} c_3$. Let Q'' be the resulting polyhedron. Then the gluing to fold Q'' produces four vertices. Among them, three vertices produced by the points c_1, c_2, c_3 are smooth vertices of curvature π. The cocurvature of the vertex of Q'' generated by the gluing of p_{j_1} to the boundary is $3\pi - (\kappa(v_j) + \kappa(v_i))$, which is in $[\pi, \frac{5\pi}{3}]$ by $\frac{4\pi}{3} \leq \sigma(v_i)+\sigma(v_j) \leq 2\pi$. Therefore, Q'' satisfies Alexandrov's conditions, and hence we obtain $Q'' \in \Pi_3 \cup \mathcal{Q}$. By Lemma 9, there exists $Q''' \in \Pi_4$ such that there is a 3-step refolding sequence from Q to Q'''. □

7 Conclusion

In this paper, we give a partial answer to Open Problem 25.6 in [5]. For every pair of regular polyhedra, we obtain a refolding sequence of length at most 6. Although this is the first refolding result for the regular dodecahedron, the number of refolding steps to other regular polyhedra seems a bit large. Finding a shorter refolding sequence than Theorem 8 is an open problem.

The notion of refolding sequence raises many open problems. What pairs of convex polyhedra are connected by a refolding sequence of finite length? Is there any pair of convex polyhedra that are not connected by any refolding sequence?

At the center of our results is that the set of tetramonohedra induces a clique by the binary relation of refoldability. Is the regular dodecahedron refoldable to a tetramonohedron? Are all Archimedean and Johnson solids refoldable to tetramonohedra? Is there any convex polyhedron not refoldable to a tetramonohedron? (If not, we would obtain a 3-step refolding sequence between any pair of convex polyhedra.)

Another open problem is the extent to which allowing or forbidding overlap in the common unfoldings affects refoldability. While we have defined refoldability to allow overlap, in particular to follow [4] where it may be necessary, most of the results in this paper would still apply if we forbade overlap. For example, Appendix A confirms this for our refolding sequence from the regular dodecahedron to a tetramonohedron; while the general approach of Lemma 9 is likely harder to generalize. Are there two polyhedra that have a common unfolding but all such common unfoldings overlap? (If not, the two notions of refolding are equivalent.)

Acknowledgments

This work was initiated during MIT class 6.849: Geometric Folding Algorithms, Fall 2020.

Erik Demaine was partially supported by the Cornell Center for Materials Research with funding from the NSF MRSEC program (DMR-1120296). Hanyu Zhang was primarily supported by the Cornell Center for Materials Research with funding from the NSF MRSEC program (DMR-1719875). Ryuhei Uehara was partially supported by MEXT/JSPS Kakenhi Grant JP17H06287 and JP18H04091.

References

[1] Jin Akiyama and Kiyoko Matsunaga. An algorithm for folding a Conway tile into an isotetrahedron or a rectangle dihedron. *Journal of Information Processing*, 28:750–758, December 2020.

[2] Yoshiaki Araki, Takashi Horiyama, and Ryuhei Uehara. Common unfolding of regular tetrahedron and Johnson-Zalgaller solid. *Journal of Graph Algorithms and Applications*, 20(1):101–114, February 2016.

[3] Amartya Shankha Biswas and Erik D. Demaine. Common development of prisms, anti-prisms, tetrahedra, and wedge. In *Proceedings of the 29th Canadian Conference on Computational Geometry (CCCG 2017)*, pages 202–207, 2017.

[4] Erik D. Demaine, Martin L. Demaine, Jin-ichi Itoh, Anna Lubiw, Chie Nara, and Joseph O'Rourke. Refold rigidity of convex polyhedra. *Computational Geometry: Theory and Applications*, 46(8):979–989, October 2013.

[5] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007.

[6] Takashi Horiyama and Ryuhei Uehara. Nonexistence of common edge developments of regular tetrahedron and other Platonic solids. In *Proceedings of the China-Japan Joint Conference on Computational Geometry, Graphs and Applications (CGGA 2010)*, pages 56–57, 2010.

[7] Tonan Kamata, Akira Kadoguchi, Takashi Horiyama, and Ryuhei Uehara. Efficient folding algorithms for regular polyhedra. In *Proceedings of the 32nd Canadian Conference on Computational Geometry (CCCG 2020)*, pages 131–137, 2020.

[8] Yu. G. Reshetnyak and S. S. Kutateladze, editors. *A. D. Alexandrov: Selected Works*, chapter Existence of a convex polyhedron and a convex surface with a given metric, pages 169–173. Part I. Gordon and Breach, 1996.

[9] Toshihiro Shirakawa, Takashi Horiyama, and Ryuhei Uehara. On common unfolding of a regular tetrahedron and a cube (in Japanese). *Origami No Kagaku (Journal of Science Origami in Japanese)*, 4(1):45–54, 2015. (See [10] for details.).

[10] Ryuhei Uehara. *Introduction to Computational Origami: The World of New Computational Geometry*. Springer, 2020.

[11] Dawei Xu, Takashi Horiyama, Toshihiro Shirakawa, and Ryuhei Uehara. Common developments of three incongruent boxes of area 30. *Computational Geometry: Theory and Applications*, 64:1–17, August 2017.

A Common Unfoldings from Regular Dodecahedron to Tetramonohedron in Theorem 8

Figures 17, 18, 19, and 20 show the common unfoldings of each consecutive pair of polyhedra in the refolding sequence from the proof of Theorem 8.

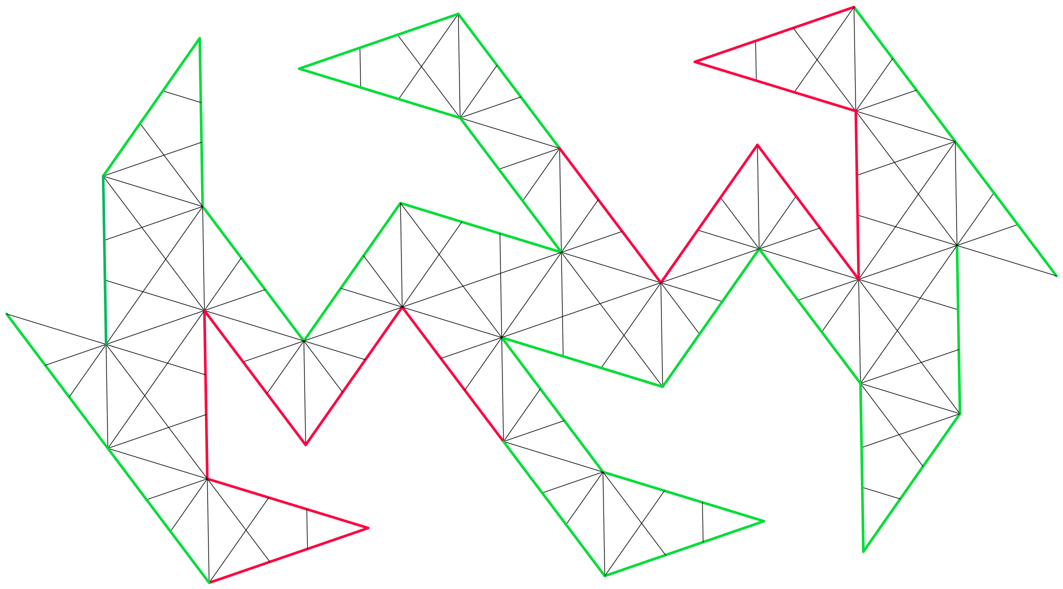


Figure 17: A common unfolding of D and Q_1

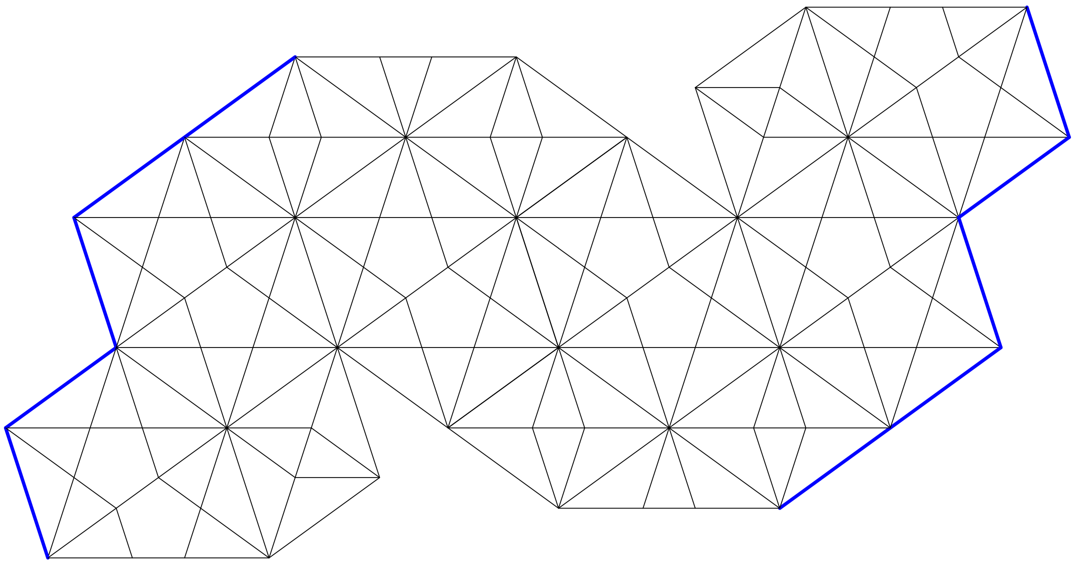


Figure 18: A common unfolding of Q_1 and Q_2

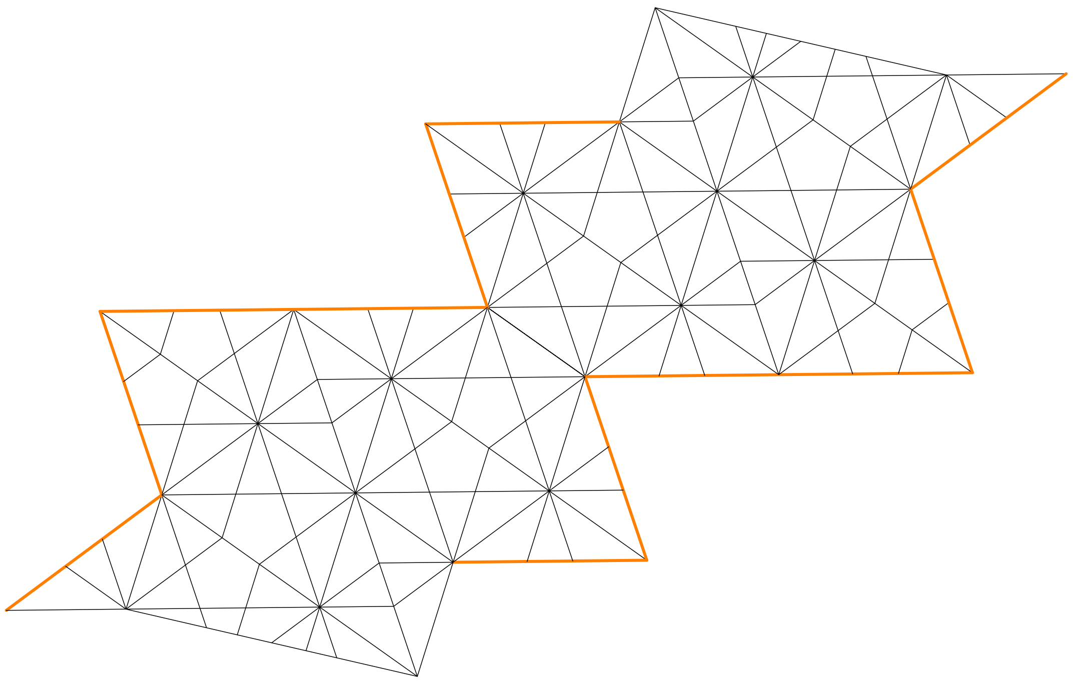


Figure 19: A common unfolding of Q_2 and Q_3

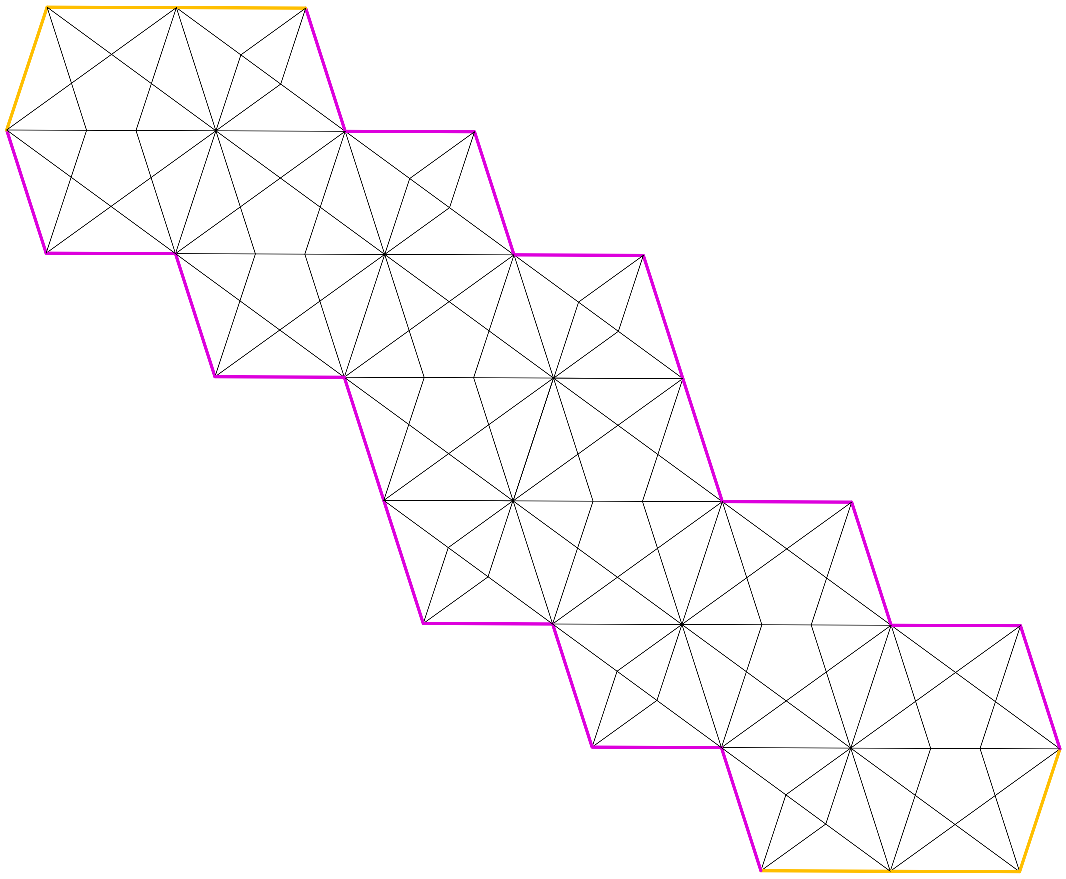


Figure 20: A common unfolding of Q_3 and Q_4

Edge-Unfolding Prismatoids: Tall or Rectangular Base

Vincent Bian* Erik D. Demaine† Rachana Madhukara*

Abstract

We show how to edge-unfold a new class of convex polyhedra, specifically a new class of prismatoids (the convex hull of two parallel convex polygons, called the top and base), by constructing a nonoverlapping "petal unfolding" in two new cases: (1) when the top and base are sufficiently far from each other; and (2) when the base is a rectangle and all other faces are nonobtuse triangles. The latter result extends a previous result by O'Rourke that the petal unfolding of a prismatoid avoids overlap when the base is a triangle (possibly obtuse) and all other faces are nonobtuse triangles. We also illustrate the difficulty of extending this result to a general quadrilateral base by giving a counterexample to our technique.

1 Introduction

A famous open problem known as Dürer's problem [2, Open Problem 21.11, p. 298] asks whether every convex polyhedron has an ***edge unfolding***, that is, a set of edges to cut such that the remaining surface unfolds into the plane without overlap. Despite the simple statement of the problem, a solution remains elusive. One approach to making partial progress on this problem is to prove that special classes of convex polyhedra have edge unfoldings.

One of the simplest yet still-open cases is ***prismatoids***, defined as the convex hull of two parallel convex polygons, called the ***top*** and ***base*** (bottom). Aloupis [1] showed that, if we omit the top and base, the resulting "band" of side faces has an edge unfolding. The challenge is thus to place the top and base without overlap; indeed, O'Rourke [3] showed that it is impossible to simply attach these polygons to an unfolded band without overlap (a "band unfolding").

A simpler goal is to unfold a prismatoid with the top removed, resulting in a polyhedron homeomorphic to a disk called a ***topless prismatoid***. At CCCG 2013, O'Rourke [4] constructed an edge unfolding for any topless prismatoid whose faces other than the base are triangles. Specifically, the edge unfolding has a strong property called ***petal unfolding***, meaning that it does not cut any of edges incident the base.

A topless prismatoid can be viewed as the local neighborhood of a single face on an arbitrary convex polyhedron; indeed, this work extends past petal unfoldings of a single face and its edge-adjacent faces ("edge-neighborhood patch") on a convex polyhedron [5] and of "domes" where all faces except a base share a single vertex [2, Section 22.5.2, p. 319] (which introduced petal unfoldings as "volcano unfoldings"). On the negative side, O'Rourke [4] showed that the larger neighborhood of faces sharing a vertex with a single face on a convex polyhedron ("vertex-neighborhood patch") does not always have a nonoverlapping petal unfolding. On the positive side, O'Rourke [4] showed that such a neighborhood has a nonoverlapping petal unfolding if the base is a triangle (possibly obtuse) and all other incident faces are nonobtuse triangles.

The latter result also leads to an edge unfolding of prismatoids with both the top and base, provided the base B is a triangle (possibly obtuse) and all other faces (including the top A) are nonobtuse triangles. In this setting, the definition of ***petal unfolding*** extends to mean that it does not cut any of edges incident to the base B, and cuts all but one of the edges incident to the top A. (Thus, in all cases, the side faces unfold by simple rotation around one edge of the base B.) O'Rourke [4] in fact showed that *all* petal unfoldings of such prismatoids avoid overlap.

1.1 Our Results

We expand O'Rourke's methods to encompass a broader family of prismatoids, showing that petal unfoldings never overlap in two new situations. Our first result is a step toward O'Rourke's conjecture that the base can be any convex polygon, provided the other faces are nonobtuse triangles:

Theorem 1.1 *For any prismatoid where the base B is a rectangle and all other faces are nonobtuse triangles, every petal unfolding avoids overlaps.*

Our second result takes a different approach, showing that "tall" prismatoids always petal unfold, and thus thin prismatoids form the remaining hard case:

Theorem 1.2 *For any prismatoid whose top A and base B are sufficiently far apart, every petal unfolding*

*MIT Department of Mathematics, Cambridge, MA, USA {vinvinb,rachanam}@mit.edu

†MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA, edemaine@mit.edu

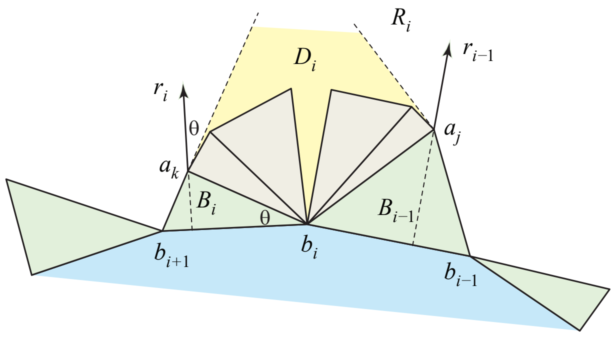


Figure 1: The diamond region D_i and the A-triangles it contains. [Based on Figure 12(a) of [4], used with permission.]

avoids overlaps. More precisely, petal unfolding avoids overlap if

$$z \geq \frac{3\pi P_A + 4d_{AB}}{2\Delta_B},$$

where

- *z is the distance between the planes containing the two bases A, B of the prismatoid;*
- *P_A is the perimeter of the top A;*
- *$\Delta_B = \pi - \max_i \angle_B(b_i)$ is the smallest turn angle in the base B (in radians);*
- *A' is the projection of A onto the plane of B; and*
- *d_{AB} is the diameter of the region $A' \cup B$.*

We prove these theorems in Sections 3 and 4 respectively, after covering the relevant background from [4] in Section 2. In Section 3.1, we give counterexamples for extending our technique of Theorem 1.1 to general quadrilaterals.

2 Background

We follow the notation given in O'Rourke's paper [4]. Let A and B be the top and base of the prismatoid, respectively. Let $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$ be the vertices of A and B respectively. Let B_i be the triangle with one vertex on A and two vertices at b_i and b_{i+1}, where indices are treated modulo n. Call these triangles ***B*-triangles**, and define ***A*-triangles** similarly.

Consider two consecutive B-triangles $B_{i-1} = b_{i-1}b_i a_j$ and $B_i = b_i b_{i+1} a_k$ in the unfolding, as in Figure 1. Define a diamond region D_i bounded by line segments $b_i a_j$ and $b_i a_k$, and by the rays through a_j and a_k perpendicular to $b_i a_j$ and $b_i a_k$ respectively. Because all the A-triangles are nonobtuse, all the A-triangles attached to edges $b_i a_j$ or $b_i a_k$ stay within the region D_i.

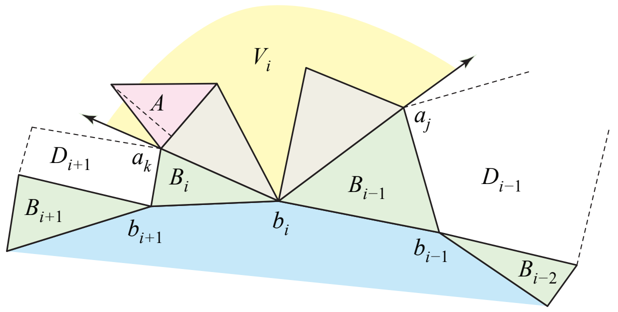


Figure 2: The region V_i containing A-triangles and the top A. [Based on Figure 13 of [4], used with permission.]

Define a larger wedge region V_i bounded by rays $\overrightarrow{b_i a_j}$ and $\overrightarrow{b_i a_k}$ (and disjoint from B, B_{i-1}, and B_i), as shown in Figure 2. Wedge V_i contains all the A-triangles attached to $b_i a_j$ or $b_i a_k$, as well as the top A, should it be attached to one of these A-triangles.

3 Unfolding Rectangular-Base Prismatoids (Proof of Theorem 1.1)

O'Rourke [4] showed that petal unfoldings never overlap for prismatoids with a convex base B and all other faces nonobtuse triangles *provided* that the region V_i does not intersect any B-triangles or any diamonds D_j for $j \neq i$ (which contain all other A-triangles). He showed that this property holds when the base B is a triangle (possibly obtuse). We extend this result to include the case where B is a rectangle, as in Figure 3.

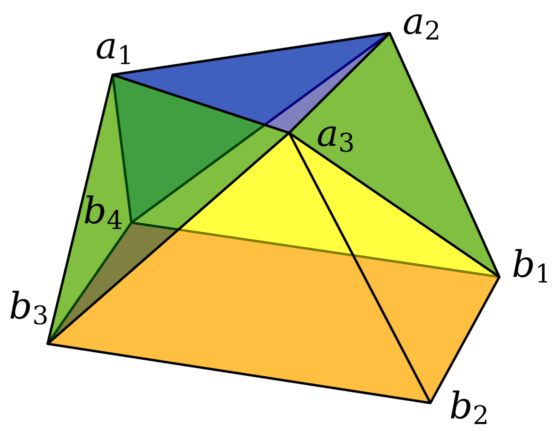


Figure 3: An acutely triangular prismatoid with a rectangular base.

Theorem 1.1 *For any prismatoid where the base B is a rectangle and all other faces are nonobtuse triangles, every petal unfolding avoids overlaps.*

Proof. O'Rourke [4] showed that it suffices to prove that V_i does not intersect any B-triangles or any diamonds D_j for $j \neq i$. He already showed that V_i does not intersect B_j, for $i-2 \leq j \leq i+1$. For a rectangle, this covers all four B-triangles. Because B_i and B_{i+1} are acute, the rays bounding D_{i+1} and D_{i-1} lie strictly outside V_i, so V_i cannot intersect those diamonds. Thus, all that remains is to show that V_i does not intersect D_{i+2}.

By symmetry, it suffices to show that V_1 does not intersect D_3, as shown in Figure 4. In fact, we claim that D_3 is contained within the region S bounded by rays $\overrightarrow{b_1b_2}$ and $\overrightarrow{b_1b_4}$ containing b_3. We will show that the line segments and rays bounding D_3 never leave S.

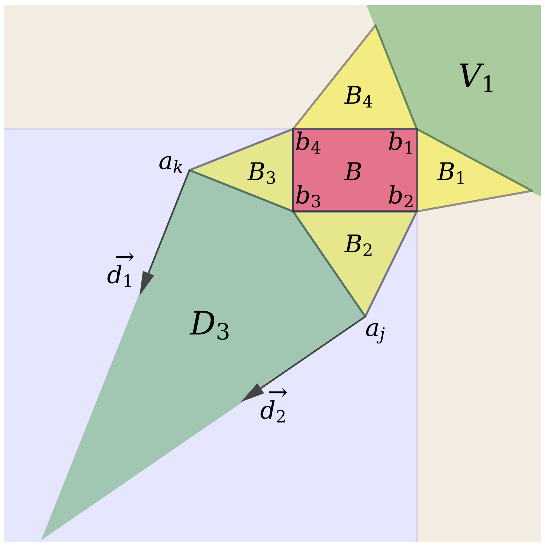


Figure 4: The regions V_1 and D_3 for the rectangular prismatoid in Figure 3. Note that D_3 always lies in the lower left quarter-plane, and V_1 always lies in the remaining three quarters of the plane.

Let a_j and a_k be the apices of triangles B_2 and B_3, so D_3 is bounded by the line segments b_3a_k and b_3a_j, and by the rays $\vec{d_1}$ and $\vec{d_2}$ perpendicular to b_3a_k and b_3a_j at a_k and a_j respectively.

First, if b_3a_k intersected line b_1b_4, then $\angle b_3b_4a_k$ of B_3 would be obtuse. Also, b_3a_k cannot intersect ray b_1b_2, as it is on the wrong side of line b_3b_4. Thus, b_3a_k is contained in S. Similarly, b_3a_j is contained in S.

Now consider ray $\vec{d_1}$. Suppose it intersected $\overrightarrow{b_1b_2}$ at some point x. Then, in quadrilateral $b_2b_3a_kx$, we have $\angle b_3a_kx = \angle xb_2b_3 = 90°$, meaning $\angle b_2b_3a_k = 180° - \angle a_kxb_2 < 180°$. However, this would make $\angle b_4b_3a_k = 360° - \angle b_2b_3b_4 - \angle b_2b_3a_k > 90°$, contradicting B_3 being nonobtuse.

Similarly, suppose that $\vec{d_1}$ intersects $\overrightarrow{b_1b_4}$ at some point y. Then, in triangle ya_kb_4, we have $\angle ya_kb_4 < 180°$, so $\angle b_4a_kb_3 = 360° - \angle ya_kb_4 - \angle b_3a_ky > 360° - 180° - 90° = 90°$, contradicting the assumption that B_3 is nonobtuse. Hence, $\vec{d_1}$ never intersects $\overrightarrow{b_1b_2}$ or $\overrightarrow{b_1b_4}$, and is thus contained in S. A similar argument shows that $\vec{d_2}$ is contained in S.

Finally, we show that V_1 intersects S only at point b_1. This claim holds because the two rays bounding V_1 only ever intersect $\overrightarrow{b_1b_2}$ and $\overrightarrow{b_1b_4}$ at b_1. Therefore, all petal unfoldings do not overlap. $\square$

3.1 Difficulty of Quadrilateral Bases

It is natural to hope that Theorem 1.1 can be extended to all quadrilateral bases, or any convex base. However, our technique above relies on the fact that each angle of B is nonobtuse. Specifically, showing that V_i and D_{i+2} do not intersect requires the assumption that $\angle b_1b_2b_3 \leq 90°$, and $\angle b_1b_4b_3 \leq 90°$. Every angle of polygon B is nonobtuse only when B is a rectangle or a nonobtuse triangle, so other quadrilaterals will require a more careful treatment.

Furthermore, the prismatoid $\mathcal{P}_c$, shown in Figure 5 and coordinatized in Table 1, is counterexample to the conjecture that the regions do not overlap when B is a general quadrilateral. Figure 6 shows the overlap.

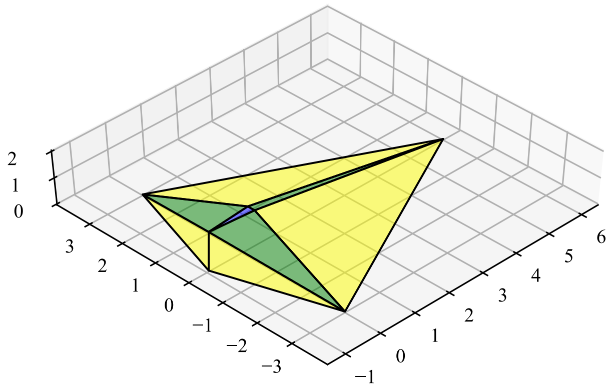


Figure 5: Prismatoid $\mathcal{P}_c$ has a quadrilateral base and all other faces nonobtuse triangles. The largest angle among the triangular faces is 89.7°.

The points of this prismatoid can be moved so that the base B is cyclic (vertices lie on a common circle), forming a new prismatoid $\mathcal{P}_{cyc}$ with coordinates given by Table 2. To find the coordinates of $\mathcal{P}_{cyc}$, we used a gradient descent method to minimize $|\angle b_1b_2b_3 - 90°|$ while maintaining that all triangles are nonobtuse. The overlap of the regions in $\mathcal{P}_{cyc}$ is much more difficult to see (refer to Figure 6): the angle formed at the intersection point is less than 0.003°.

These examples mean that extending the proof of Theorem 1.1, even to just cyclic quadrilaterals, requires a more precise treatment than considering the regions V_i and D_i. On the other hand, all petal unfoldings of $\mathcal{P}_c$ and $\mathcal{P}_{cyc}$ have no overlap, so O'Rourke's conjecture about petal unfoldings with an arbitrary convex base remains plausible.

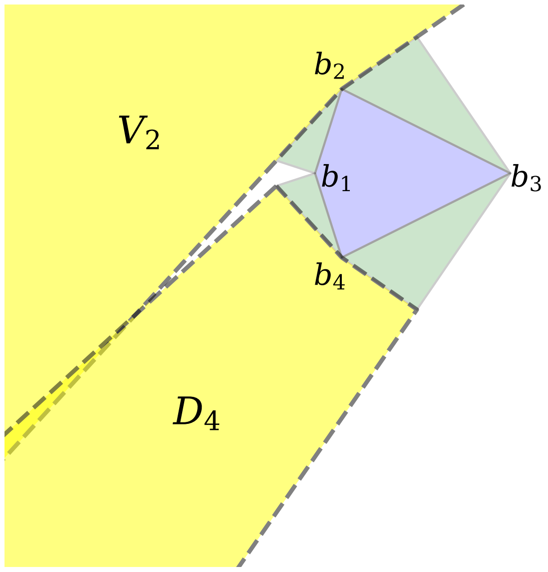


Figure 6: The regions V_2 and D_4 intersect.

Point(s)	Coordinates		
b_1	$(-0.95,$	$0.00,$	$0.00)$
b_2, b_4	$(0.00,$	$\pm 3.00,$	$0.00)$
b_3	$(6.00,$	$0.00,$	$0.00)$
a_1	$(-0.90,$	$0.00,$	$1.45)$
a_2, a_3	$(0.30,$	$\pm 0.10,$	$1.45)$

Table 1: The coordinates of the vertices of $\mathcal{P}_c$.

Point(s)	Coordinates		
b_1	$(-1.5633,$	$0.0000,$	$0.0000)$
b_2, b_4	$(0.0000,$	$\pm 3.7169,$	$0.0000)$
b_3	$(8.8372,$	$0.0000,$	$0.0000)$
a_1	$(-1.5581,$	$0.0000,$	$1.6435)$
a_2, a_3	$(0.2225,$	$\pm 0.0299,$	$1.6435)$

Table 2: The coordinates of the vertices of $\mathcal{P}_{cyc}$.

4 Unfolding Tall Prismatoids (Proof of Theorem 1.2)

For a given prismatoid, let z denote the distance between the planes of the top and base. We show that, for prismatoids with large enough z, all petal unfoldings avoid overlap.

Theorem 1.2 *For any prismatoid whose top A and base B are sufficiently far apart, every petal unfolding avoids overlaps. More precisely, petal unfolding avoids overlap if*

$$z \geq \frac{3\pi P_A + 4d_{AB}}{2\Delta_B},$$

where

- *z is the distance between the planes containing the two bases A, B of the prismatoid;*

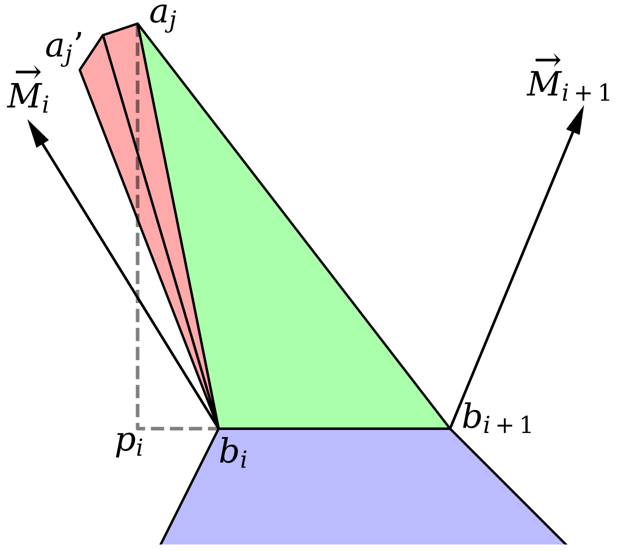


Figure 7: One of the B-triangles, along with some A-triangles attached to its left. In this configuration, p_i is on the opposite side of b_i as b_{i+1}, so $\angle b_{i+1} b_i a_j$ is obtuse.

- *P_A is the perimeter of the top A;*
- *$\Delta_B = \pi - \max_i \angle_B(b_i)$ is the smallest turn angle in the base B (in radians);*
- *A' is the projection of A onto the plane of B; and*
- *d_{AB} is the diameter of the region $A' \cup B$.*

Proof. We show that, in any petal unfolding, every face that gets attached to a B-face B_i will stay in a region S_i bounded by the edge $b_i b_{i+1}$ and the rays $\overrightarrow{M_i}$ and $\overrightarrow{M_{i+1}}$ bisecting the exterior angles of B at b_i and b_{i+1} respectively, as shown in Figure 7. Note that the angle between edge $b_i b_{i+1}$ and $\overrightarrow{M_i}$ is at least $\frac{\pi}{2} + \frac{\Delta_B}{2}$, and every edge of the form $b_i a_j$ has length at least z.

Let $0 < \ell < 1$ be a constant. Consider a B-face B_i with vertices $b_i b_{i+1} a_j$. First we claim that the angle $\angle b_{i+1} b_i a_j$ will be at most $\frac{\pi}{2} + \frac{\Delta_B}{2} \cdot \ell$, as long as $z \geq \frac{2d_{AB}}{\Delta_B \ell}$.

Consider the projection p_i of a_j onto $b_i b_{i+1}$. If it lies on the same side of b_i as b_{i+1}, then $\angle b_{i+1} b_i a_j$ is acute, and we are done. Otherwise, the angle is obtuse, but we can use the fact that the length of $b_i p_i$ is at most d_{AB}.

In this case, we know $\angle b_{i+1} b_i a_j = \frac{\pi}{2} + \arctan \frac{b_i p_i}{p_i a_j}$. Also $p_i a_j \geq z$, so

$$\angle b_{i+1} b_i a_j \leq \frac{\pi}{2} + \arctan \frac{d_{AB}}{z} \leq \frac{\pi}{2} + \frac{d_{AB}}{z}.$$

Substituting in our assumption that $z \geq \frac{2d_{AB}}{\Delta_B \ell}$, we get that $\angle b_{i+1} b_i a_j \leq \frac{\pi}{2} + \frac{\Delta_B}{2} \cdot \ell$, as desired.

Second, we show that, as long as $z \geq \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell}$, the angle $\angle a_j b_i a_j'$ subtended by the A-triangles attached to edge $a_j b_i$ is at most $\frac{\Delta_B}{3}(1-\ell)$. We start by bounding the measure of $\angle a_j b_i a_{j+1}$ for any edge $a_j a_{j+1}$ of A. By the Law of Sines, $\frac{\sin \angle a_j b_i a_{j+1}}{a_j a_{j+1}} = \frac{\sin \angle a_j a_{j+1} b_i}{a_j b_i}$, so

$$\begin{aligned}\angle a_j b_i a_{j+1} &= \arcsin \frac{a_j a_{j+1} \sin \angle a_j a_{j+1} b_i}{a_j b_i} \\ &\leq \arcsin \frac{a_j a_{j+1}}{z}.\end{aligned}$$

Because $\arcsin x \leq \frac{\pi}{2} x$ for $x \geq 0$, we obtain $\angle a_j b_i a_{j+1} \leq \frac{\pi}{2} \cdot \frac{a_j a_{j+1}}{z}$.

The sum of these lengths $a_j a_{j+1}$ over all A-triangles is P_A, so the sum of the angles over all A-triangles is at most $\frac{\pi P_A}{2z}$. Because the angle $\angle a_j b_i a'_j$ is the sum of $\angle a_j b_i a_{j+1}$ over some subset of the edges $a_j a_{j+1}$ of A, we can substitute $z \geq \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell}$ to get that $\angle a_j b_i a'_j \leq \frac{\Delta_B}{3}(1-\ell)$.

Third, we show that, if $z \geq \min\left(\frac{2d}{\Delta_B \ell}, \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell}\right)$, then no matter where the top face A is attached in the unfolding, it will not exit the region S_i. We accomplish this by proving that the shortest distance $d_{\min}$ between the point a'_j and the ray $\overrightarrow{M_i}$ is at least $\frac{P_A}{2}$. By the triangle inequality, this means that A cannot intersect $\overrightarrow{M_i}$. Note that this shortest distance is $d_{\min} = b_i a'_j \sin\left(\frac{\pi}{2} + \frac{\Delta_B}{2} - \angle b_{i+1} b_i a_j - \angle a_j b_i a'_j\right)$.

We know that $b_i a'_j \geq z$, and from our previous results, we know that

$$\begin{aligned}&\frac{\pi}{2} + \frac{\Delta_B}{2} - \angle b_{i+1} b_i a_j - \angle a_j b_i a'_j \\ &\geq \frac{\pi}{2} + \frac{\Delta_B}{2} - \frac{\pi}{2} - \frac{\Delta_B}{2} \cdot \ell - \frac{\Delta_B}{3}(1-\ell) \\ &= \frac{\Delta_B}{6}(1-\ell).\end{aligned}$$

Using the fact that $\sin x \geq \frac{2x}{\pi}$ for $0 \leq x \leq \frac{\pi}{2}$, we obtain

$$d_{\min} \geq \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell} \cdot \frac{2}{\pi} \cdot \frac{\Delta_B}{6}(1-\ell) = \frac{P_A}{2},$$

as desired.

Repeating this argument for every side $b_i a_j$ of every B-triangle, we obtain that, if

$$z \geq \min\left(\frac{2d}{\Delta_B \ell}, \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell}\right),$$

then no petal unfolding of $\mathcal{P}$ can overlap. This lower bound is minimized when the two inputs to the min are equal. This occurs when $\ell = \frac{4d}{4\pi P_A + 4d}$, which when substituted yields the desired $z \geq \frac{3\pi P_A + 4d_{AB}}{2\Delta B}$. □

The most room for improvement in this proof is the second step's bound $z \geq \frac{3\pi}{2\Delta_B} P_A \cdot \frac{1}{1-\ell}$, as it is impossible for all the A-triangles to be attached to a single point on A.

Acknowledgments

This work began as a final project in an MIT class on Geometric Folding Algorithms (6.849, Fall 2020). We thank Professor Joseph O'Rourke for his guidance on this project. We also thank Yevhenii Diomidov for helpful suggestions.

References

[1] Greg Aloupis. *Reconfigurations of Polygonal Structures.* PhD thesis, McGill University, 2005.

[2] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2007.

[3] Joseph O'Rourke. Band unfoldings and prismatoids: A counterexample. Technical Report 087, Smith College, October 2007. arXiv:0710.0811.

[4] Joseph O'Rourke. Unfolding prismatoids as convex patches: Counterexamples and positive results. In *Proceedings of the 25th Canadian Conference on Computational Geometry*, 2013. Full paper available as arXiv:1205.2048.

[5] Val Pinciu. On the fewest nets problem for convex polyhedra. *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 21–24, 2007.

On Guarding Polygons with Holes

Sharareh Alipour*

Abstract

There is an old conjecture by Shermer [7] that in a polygon with n vertices and h holes, $\lfloor \frac{n+h}{3} \rfloor$ vertex guards are sufficient to guard the entire polygon. The conjecture is proved for $h = 1$ by Shermer [7] and Aggarwal [5] seperately. In this paper, we prove a theorem similar to the Shermer's conjecture for a special case where the goal is to guard the vertices of the polygon (not the entire polygon) which is equivalent to finding a dominating set for the visibility graph of the polygon. Our proof also guarantees that the selected vertex guards also cover the entire outer boundary (outer perimeter of the polygon) as well.

1 Introduction

A set S of points is said to guard a polygon if, for every point p in the polygon, there is some $q \in S$ such that the line segment between p and q is inside the polygon.

The art gallery problem asks for the minimum number of guards that are sufficient to guard any polygon with n vertices. There are numerous variations of the original problem that are also referred to as the art gallery problem. In some versions guards are restricted to the perimeter, or even to the vertices of the polygon which are called vertex guards. Some versions require only the perimeter or a subset of the perimeter to be guarded. The version in which guards must be placed on vertices and only vertices need to be guarded is equivalent to the minimum dominating set problem for the visibility graph of the polygon.

In graph theory, for a given graph G with vertex set $V(G)$, $U \subseteq V(G)$ is a dominating set for G if every vertex $v \in V(G) \backslash U$ has a neighbor in U. Minimum dominating set problem is to find a dominating set $V^* \subseteq V(G)$ such that the size of V^* (denoted by $|V^*|$) is minimum among all dominating sets.

Related results

Chvátal's art gallery theorem [2] states that $\lfloor \frac{n}{3} \rfloor$ vertex guards are always sufficient and sometimes necessary to guard a simple polygon with n vertices. Later, Fisk [3] gave a short proof for Chvátal's art gallery theorem.

O'Rourke [6] proved that any polygon P with n vertices and h holes can be guarded with at most $\lfloor \frac{n+2h}{3} \rfloor$ vertex guards. Note that n is the total number of vertices of the polygon including the boundary and holes. But Shermer conjectured that any polygon P with n vertices and h holes can always be guarded with $\lfloor \frac{n+h}{3} \rfloor$ vertex guards. This conjecture has been proved by Shermer [7] and Aggarwal [5] independently for $h = 1$. For $h > 1$, the conjecture is still open for more than 35 years. However Hoffmann, Kaufmann and Kriegel in [4] and Bjorling-Sachs and Souvaine in [1] proved Shermer's conjecture for point guards (i.e. the guards can be chosen from any points inside or on the boundary of the polygon).

Our result

In this paper, we prove that every polygon with holes has a special kind of triangulation to be specified shortly. Next by using this theorem, we prove that $\lfloor \frac{n+h}{3} \rfloor$ vertex guards are sufficient to guard the boundary of a polygon with n vertices and h holes. By boundary of P we mean the outer perimeter of P. As far as we know this version has not been studied.

2 Special triangulation

In this section, we present some basic definitions and a theorem in order to prove our main result. It has been proved that every polygon with (or without) holes can be triangulated and this triangulation is not always unique.

Definition 1 *In a given polygon P with h holes, a triangle Δ in a triangulation of P is called a special triangle if one of its edges is an edge of a hole and the apex vertex is a vertex of the polygon not on that hole (see Figure 1).*

Theorem 2 *Every polygon with holes has a triangulation with a special triangle.*

Note that according to the definition of special triangle, one edge of a special triangle is always on a hole and its apex vertex is on the boundary or on a different hole. In the following we explain the proof of Theorem 2.

*School of Computer Science, Institute for Research in Fundamental Sciences, Iran, alipour@ipm.ir

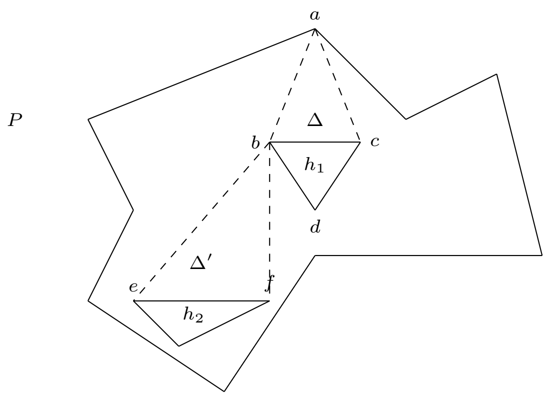


Figure 1: A polygon with 2 holes. In this example, Δ and Δ' are special triangles.

2.1 Proof of Theorem 2

Consider a triangulation of our polygon. If the number of holes is bigger than one, then there must exist an edge of this triangulation from a vertex of one of these holes to a vertex of the boundary of the polygon. If we make this edge into two parallel edges of very small distance from each other, we can make this hole into a part of the boundary, hence reduce the number of holes. A special triangle for this reduced polygon is also a special triangle for the original polygon. By induction on the number of holes, therefore it is enough to consider only the case when we have a polygon with one hole. Let us assume that this polygon with one hole has n vertices, and the existence of a special triangle for a polygon with one hole is proved for the case when the number of vertices is less than n. Note that the smallest possible n is 6 that happens when we have a triangle with a triangle as hole inside. Any triangulation for this particular polygon has a special triangle and in fact 3 special triangles. With this assumption, we can assume that non-adjacent vertices of the boundary can not see each other. Since if they do, the chord connecting them divide the boundary into two smaller parts where the hole is inside one of them. A special triangle for this smaller instance of a polygon with one hole, is also a special triangle for the original polygon. Hence a triangulation for the polygon does not have a triangle whose vertices are all on the boundary. This implies that any vertex of the boundary must see at least one vertex of the hole. Assume that we do not have a special triangle, we want to reach to a contradiction.

Let B_1 be a vertex of the boundary. According to the previous argument, it will see a vertex on the hole, say H_1. Without loss of generality, we may assume that the ray B_1H_1 in a counter clockwise rotational sweep, sees part of the edge H_1H_1' of the hole.

Since a special triangle does not exist, this rotating ray will hit an obstacle that prevents it from seeing the entire edge H_1H_1'. Let H_1'' be the point on the edge H_1H_1' obtained by rotating this ray until it hits an obstacle. This obstacle is either a vertex from the boundary or a vertex from the hole. Assume that it is a vertex B_2 from the boundary (we will discuss the second possibility shortly). Since the vertices on the boundary do not see non-adjacent vertices on the boundary, B_2 must be adjacent to B_1. Then B_2 also sees the portion H_1H_1'' of H_1H_1' and even more. Repeating the sweeping procedure with the ray B_2H_1'', we will hit another obstacle, unless $B_2H_1H_1'$ is a special triangle. The sequence of obstacles obtained this way can not be all the vertices of the boundary, because they keep seeing larger and larger portions of the edge H_1H_1' and hence they are different and we only have a finite number of vertices of the boundary. So we will reach a vertex H_2 of the hole after say $k_1 \geq 1$ steps. The vertices $B_1, \ldots, B_{k_1}$ are consecutive vertices on the boundary and the angles $B_{i-1}B_iB_{i+1}$ are all less than π, since they are obtained by counter clockwise sweeps. The vertex B_{k_1} will see a portion of the edge of the hole with the end-point H_2, say H_2H_2'. Since both of the edges of the hole with end-point H_2 are to the left of the ray $B_{k_1}H_2$, therefore we still need to rotate this ray counter clockwise along the edge H_2H_2' and hence if we hit a boundary vertex obstacle, say B_{k_1+1}, the angle $B_{k_1-1}B_{k_1}B_{k_1+1}$ is less than π.

Notice that the obstacles encountered for a vertex of the boundary B by rotating its corresponding ray counter-clockwise, can not all be among the vertices of the hole. In this case the rotating ray will make a full rotation of 2π, and this is impossible since the maximum rotational angle that this ray can have is less than the angle of the vertex B, which is definitely less than 2π degrees.

Now since we are assuming that no special triangle exists, the process of sweeping rays and hitting obstacles will be continued forever, producing a sequence of vertices of the boundary and hole. Since after reaching a boundary vertex, we can not get only vertices from the hole, the sequence of boundary vertices that are consecutive vertices of the polygon must come back to the starting point. Hence we go along all the vertices of the boundary, however the outer angles $B_{i-1}B_iB_{i+1}$ are all less than π. This is a contradiction.

3 Guarding vertices with vertex guards

Now as a result of Theorem 2, we present our main theorem.

Theorem 3 *For a given polygon P with n vertices and h holes, $\lfloor \frac{n+h}{3} \rfloor$ vertex guards are always sufficient to guard the vertices of P and also the entire boundary.*

Proof. The proof is by induction on the number of holes. Chvátal's theorem implies that when $h = 0$, $\lfloor \frac{n}{3} \rfloor$

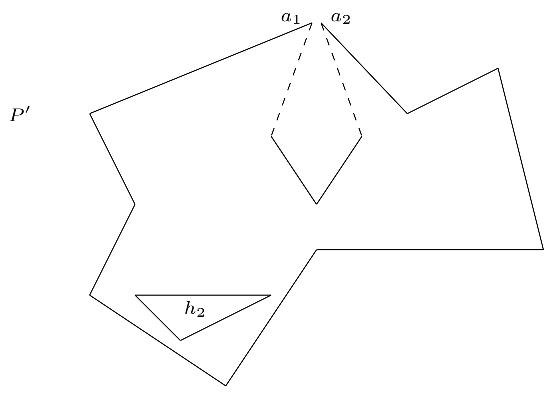


Figure 2: We split the vertex a into 2 vertices a_1 and a_2. Now the polygon has $1(h-1)$ hole and $16(n+1)$ vertices.

vertex guards are sufficient to guard the entire polygon. Suppose that the theorem is proved for $h-1$ holes. Now suppose that we are given a polygon P with n vertices and h holes. According to Theorem 2, there is a triangulation with a special triangle Δ. Suppose that Δ has a vertex a, on the boundary or a hole and another edge, bc on some other hole. We split a into two vertices a_1 and a_2. So, P is changed into a polygon P' with $h-1$ holes and $n+1$ vertices (See Figure 2). According to the induction assumption, we can choose $\lfloor \frac{n+1+h-1}{3} \rfloor$ vertex guards that guard the vertices of polygon P' and the boundary of P'. Since a_1 and a_2, are guarded in P', then all vertices of P are guarded by at most $\lfloor \frac{n+h}{3} \rfloor$ vertex guards of P. Also by induction the boundary of P' is guarded. On the other hand the boundary of P is a subset of the boundary of P', so the boundary of P is guarded too. Note that if we have two vertex guards on a_1 and a_2, in P they are combined into one vertex guard. □

Remark 4 *Note that in fact this proof, gives something slightly more. The guards will cover not only the entire outer perimeter of the polygon, but also the perimeter of holes with an exception of at most h segments on them. The reason is that the segment that is the base of the special triangle that was used in the proof above is not necessarily guarded, so a similar induction on the number of holes will prove our claim.*

4 Concluding remarks

In this paper, we have proved a theorem similar to the Shermer's conjecture for a special case of the Art Gallery problem where we only need to guard the vertices of the polygon. The proof is based on the existence of a special triangle in the polygon. The proposed algorithm is simple and easy to implement. In future work one can possibly extend this idea for the general case.

Also for a given connected graph G, it has been proved that the size of minimum dominating set of G is $\leq \frac{n}{2}$. So if we construct a polygon with minimum number of holes such that its visibility graph is isomorphic to G, our proof gives an upper bound of $\lfloor \frac{n+h}{3} \rfloor$ for the size of minimum dominating set of G.

References

[1] I. Bjorling-Sachs and D. L. Souvaine. An efficient algorithm for guard placement in polygons with holes. *Discrete & Computational Geometry*, 13:77–109, 1995.

[2] V. Chvatal. A combinatorial theorem in plane geometry. In *Journal of Combinatorial Theory, Series B*, volume 18, pages 39–41.

[3] S. Fisk. A short proof of chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24(3):374, 1978.

[4] F. Hoffmann, M. Kaufmann, and K. Kriegel. The art gallery theorem for polygons with holes. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 39–48, 1991.

[5] J. O'Rourke. *The art gallery theorem: its variations, applications, and algorithmic aspects.* Ph.D. thesis, Johns Hopkins Univ, 1984.

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

[7] T. Shermer. Triangulation graphs that require extra guards. *NYIT Computer Graphics Technical Report*, 3D-13, 1984.

An Acrophobic Guard Watchtower Problem on Terrains

Ritesh Seth* Anil Maheshwari† Subhas C Nandy‡

Abstract

In the *acrophobic guard watchtower problem* for a polyhedral terrain, a square axis-aligned platform is placed on the top of a tower whose bottom end-point lies on the terrain's surface. As in the standard watchtower problem, the objective is to minimize the tower's height such that every point on the surface of the terrain is weakly visible from the platform placed on the top of the tower. In this paper, we show that in $\mathbb{R}^2$, the problem can be solved in $O(n)$ time, and in $\mathbb{R}^3$, it takes $O(n^2)$ time, where n is the total number of vertices in the terrain.

1 Introduction

The problem of placing watchtowers on a polyhedral terrain in $\mathbb{R}^3$ is a matter of great interest to the computational geometers for a long time. The problem is to establish a given number (k) of watchtowers on a polyhedral terrain such that every point on the surface of the terrain becomes visible to at least one of the watchtowers, and the maximum height among these k watchtowers is minimized. It has many applications in surveillance and navigation in the context of defence and geographic information systems. Initial research on this topic was on the extension of the art-gallery problems in polyhedral terrain.

Cole and Sharir [3] proved that the problem of finding the minimum number of vertex guards on a polyhedral terrain in $\mathbb{R}^3$ is NP-hard. Bose et al. [2] obtained lower bounds on the number of vertex and edge guards for a polyhedral terrain. The first algorithm for the one watchtower placement problem in $\mathbb{R}^3$ was proposed by Sharir [5]; it runs in $O(n \log^2 n)$ time. Later the time complexity of the problem was improved to $O(n \log n)$ by Zhu [7]. Agarwal et al. [1] proposed a deterministic polynomial-time algorithm for the discrete version of the two watchtower problem where the watchtowers are allowed to be placed only at the vertices of the terrain. The time complexity of their algorithm is $O(n^{\frac{11}{3}} polylog(n))$, and it uses the parametric search technique of Meggido [4]; here n is the number of vertices of the polyhedral terrain. The general k-watchtower problem is studied recently in [6], and the proposed time complexity is $(k^3 n^{k+4})$. For the special case, where $k = 2$, their algorithm works in $O(n^4 \log n)$ worst-case time, which is slightly less efficient than the algorithm by [1]. The algorithm in [6] is simpler to implement as compared to that of [1].

This paper introduces a new variation of the watchtower problem, the *acrophobic guard watchtower problem.* Given a terrain $\mathcal{T}$ in $\mathbb{R}^3$ with x-y plane as its base, and a square horizontal (parallel to the x-y plane) platform $\mathcal{L}$ whose sides are parallel to the x and y axis of unit side-length, the watchtower is a vertical (parallel to the z-axis) line segment τ whose top end-point touches any point of $\mathcal{L}$ and the bottom end-point touches the surface of $\mathcal{T}$. Here, the objective is to find a point on the surface of $\mathcal{T}$ such that if the tower is placed at that point, then every point on the surface of $\mathcal{T}$ is weakly visible from $\mathcal{L}$. In other words, a guard can observe every point on the surface of the terrain while patrolling on the (transparent) *platform* $\mathcal{L}$, and the length of the tower τ is minimized.

We first study the problem for terrains in $\mathbb{R}^2$ where the platform is a unit length line segment parallel to the x-axis and propose a linear time algorithm. A terrain in $\mathbb{R}^2$ is a x-monotone polygonal chain consisting of n-vertices. Next, we extend the problem for terrains in $\mathbb{R}^3$ where the platform is an axis-aligned unit square parallel to the x-y plane. A terrain in $\mathbb{R}^3$ is typically visualized as a planar triangulation consisting of n-faces in the x-y plane, where each point also has an associated height, i.e., its z-coordinate.

2 Watchtower problem in $\mathbb{R}^2$

A terrain $\mathcal{T}$ in $\mathbb{R}^2$ is a x-monotone connected polygonal chain whose both the end-points touch the x-axis at two points, defining the base of the terrain. (In

*Indian Statistical Institute, Kolkata, India, ritesh1996seth2015@gmail.com

†Carleton University, Ottawa, Canada, anil@scs.carleton.ca

‡Indian Statistical Institute, Kolkata, India, nandysc@isical.ac.in

the literature, often this is referred to as a terrain in 1.5-dimension.) A watchtower τ at a point γ_t on an edge/vertex of $\mathcal{T}$ is a vertical line segment $[\gamma_t, \gamma_l]$, and a horizontal line segment $\mathcal{L} = [a, b]$ of unit length, called the platform, touching the point γ_l (see Figure 1). The point γ_t will be referred to as the base of the watchtower, and the height $h(\gamma_t)$ of the watchtower τ is the length of $[\gamma_t, \gamma_l]$. The objective is to locate the base $\gamma_t \in \mathcal{T}$ such that the height of the tower at γ_t, denoted by $h(\gamma_t)$, is minimum, and the entire terrain $\mathcal{T}$ is weakly visible[1] from the platform $\mathcal{L}$ placed at γ_t.

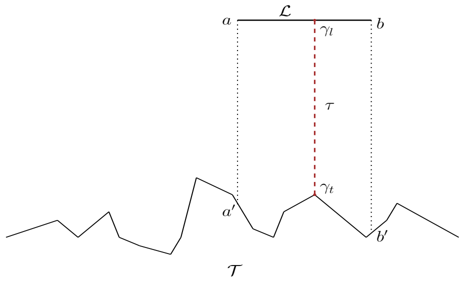


Figure 1: Demonstration of segment watchtower

Observe that for a platform $\mathcal{L} = [a, b]$, the height of the watchtower is defined as follows: take vertical projections a' and b' of the points a and b on the poly-chain $\mathcal{T}$. If γ_t is a point of maximum y-coordinate on the poly-chain $\mathcal{T}$ from a' to b', and γ_l is the vertical projection of γ_t on the line segment $[a, b]$, then the height of the watchtower is the length of the line segment $[\gamma_t, \gamma_l]$ (see Figure 1).

The objective is to identify an appropriate point γ_t on the surface of $\mathcal{T}$ and place the platform $\mathcal{L} = [a, b]$ on a point γ_l which is vertically above γ_t such that the entire $\mathcal{T}$ is weakly visible from $\mathcal{L}$, and the length of $[\gamma_t, \gamma_l]$ is minimum among every other point on the polyline $\mathcal{T}$ as the base of the watchtower. Note that the optimal height of the acrophobic guard watchtower is no larger than the height of the usual (point) watchtower [5, 7].

First let us fix some notations. We use V and E to denote the vertices and edges of $\mathcal{T}$; $|V| = n$, and hence $|E| = n - 1$. Let l_l and l_r be two vertical lines at the left and right end-points of $\mathcal{T}$. $\mathcal{T}$ splits the vertical strip bounded by l_l and l_r into two regions, namely the upper region and the lower region, respectively. An edge e of $\mathcal{T}$ is *completely visible* from a point α if for any point $\beta \in e$ the line segment $[\alpha, \beta]$ does not intersect the interior of the terrain $\mathcal{T}$. The line containing an edge $e \in E$ splits the plane into two regions (half-planes), namely *positive region* e^+ and *negative region* e^-; e is not at all visible from any point in the e^- region (assuming $e \notin e^-$).

[1] $\mathcal{T}$ is said to be weakly visible from $\mathcal{L}$ if every point of $\mathcal{T}$ is visible from some point of $\mathcal{L}$

Result 1 *[7] The terrain $\mathcal{T}$ is completely visible from every point of the region $\mathcal{S}$ bounded by the intersection of the positive regions defined by the lines containing all the edges in E.*

We further split the edge set of $\mathcal{T}$ into two subsets. The edges with positive (resp. negative) slope with respect to positive x axis are referred to as *acute* (resp. *obtuse*) edges. We assume that there is no edge perpendicular to the x-axis. We use $\mathcal{A}$ (resp. $\mathcal{O}$) to denote the set of all *acute* (resp. *obtuse*) edges; $E = \mathcal{A} \cup \mathcal{O}$ and $\mathcal{A} \cap \mathcal{O} = \phi$.

Definition 1 *With respect to the edges in $\mathcal{A}$ (resp. $\mathcal{O}$), the positive region $\mathcal{A}^+$ (resp. $\mathcal{O}^+$) is the region such that every point in $\mathcal{A}^+$ (resp. $\mathcal{O}^+$) lies in the* positive region *of all the edges in $\mathcal{A}$ (resp. $\mathcal{O}$).*

Please refer to Figures 2-4 for illustration of what follows. Note that all the edges in $\mathcal{A}$ (resp. $\mathcal{O}$) are visible from every point in $\mathcal{A}^+$ (resp. $\mathcal{O}^+$) region. The boundary of the region $\mathcal{A}^+$ (resp. $\mathcal{O}^+$) is the upper envelope of the lines containing all the edges $e \in \mathcal{A}$ (resp. $e \in \mathcal{O}$), which is a convex polychain such that as x increases, the y-coordinate on the curve monotonically increases (resp. decreases) (See Figure 2c). Let the boundary of $\mathcal{A}^+$ and $\mathcal{O}^+$ intersect at a point r, called the *critical point.* The horizontal (parallel to x-axis) line on which the *critical point* lies is referred to as the *threshold line.* The region $\mathcal{S} = \mathcal{A}^+ \cap \mathcal{O}^+$ is a convex region, which is unbounded above, and is formed by the part of those upper envelopes $\mathcal{A}^+$ and $\mathcal{O}^+$ up to the threshold line. The part of the boundary of $\mathcal{S}$ on $\mathcal{A}^+$ (resp. $\mathcal{O}^+$) is referred to as the right (resp. left) boundary of $\mathcal{S}$.

We compute $\mathcal{M} = (\mathcal{A}^+ \oplus \mathcal{L}) \cap (\mathcal{O}^+ \oplus \mathcal{L})$, where $\oplus$ is the Minkowski sum operator. It is a convex region bounded by two polychains from left and right where the left chain is parallel to the boundary of $\mathcal{O}^+$ and the right chain is parallel to the boundary of $\mathcal{A}^+$ (see Figure 3a). We define $\mathcal{M}^+$ as the region of $\mathcal{M}$ that lies above the threshold line (see Figure 3b). So $\mathcal{M}^+ \subseteq \mathcal{M}$. Observe that $\mathcal{M}^+ = \mathcal{S} \oplus \mathcal{L}$. Also we define $\mathcal{M}^- = \mathcal{M} \setminus \mathcal{M}^+$. Needless to say that $\mathcal{M}^-$ is the region of $\mathcal{M}$ below the threshold line. The region defined by $\mathcal{M}^+ \setminus \mathcal{S}$ and that lies to the left (resp. right) of $\mathcal{S}$ is referred as the *left annulus* (resp. the *right annulus*) (see Figure 3c). By Result 1, if a point of $\mathcal{L}$ lies in the region $\mathcal{S}$, then from that point, the entire terrain $\mathcal{T}$ is visible. This indicates that if $\mathcal{L}$ properly lies in the region $\mathcal{M}^+$, then $\mathcal{T}$ is entirely visible from $\mathcal{L}$ (see Figure 3c). We now show

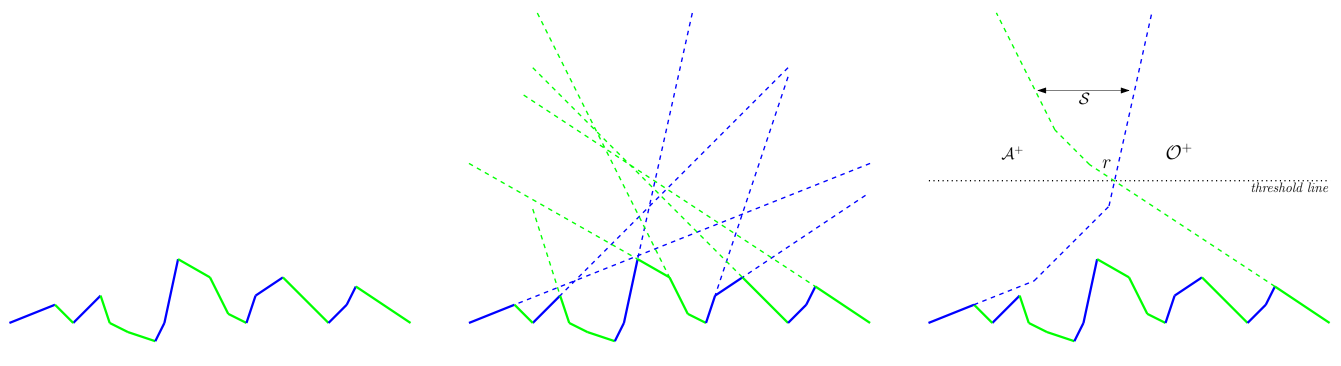


Figure 2: (a) Terrain $\mathcal{T}$ in 2-dimensions with acute (Blue) and obtuse (Green) edges (b) Extension of edges of terrain, (c) Demonstration of $\mathcal{A}^+$, $\mathcal{O}^+$, $\mathcal{S}$, critical point r and threshold line

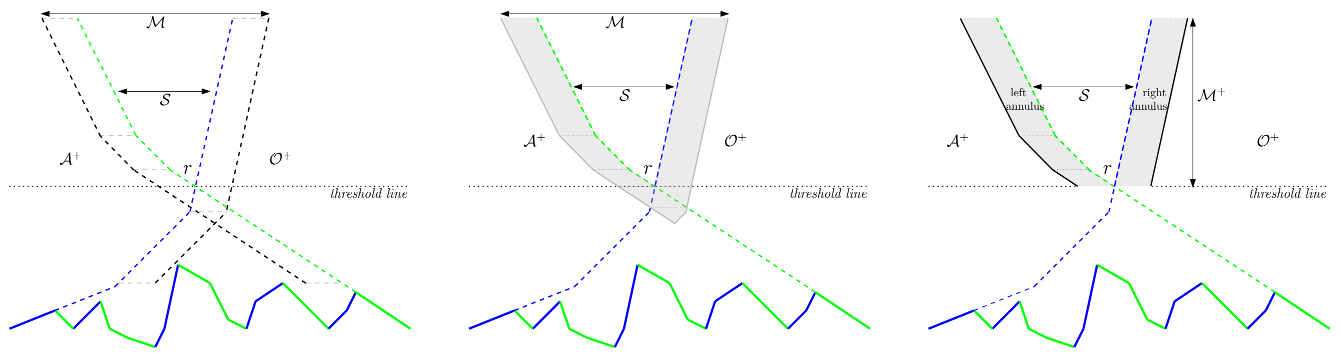


Figure 3: (a) Minkowski sum of $\mathcal{A}^+$ and $\mathcal{O}^+$ with the platform $\mathcal{L}$ (b) $\mathcal{M}$ (c) Left and right annulus and $\mathcal{M}^+$

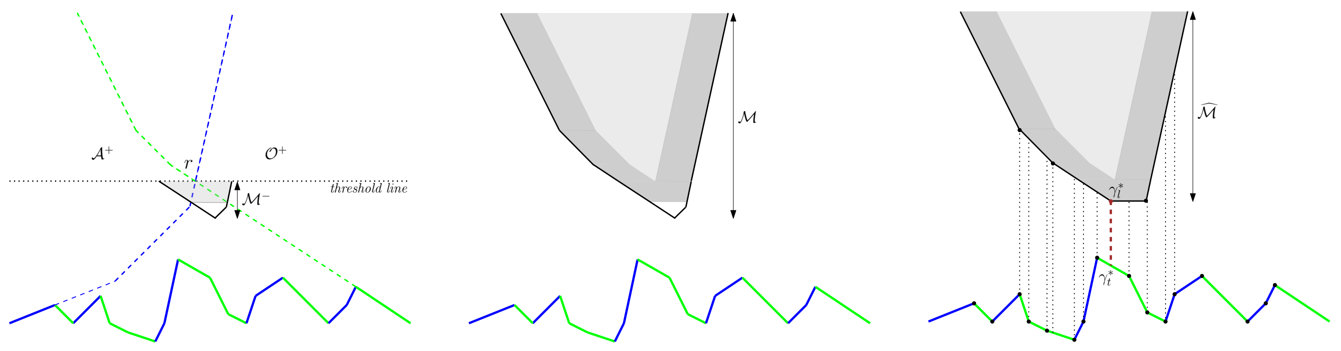


Figure 4: The feasible region: (a) $\mathcal{M}^-$ containing $\widehat{\mathcal{M}^-}$ (shaded), (b) $\mathcal{M}$ containing $\widehat{\mathcal{M}}$ (shaded), (c) Minimum height watchtower (dashed line) between $\widehat{\mathcal{M}}$ and $\mathcal{T}$ with respect to event points

that it is also possible to see the entire terrain $\mathcal{T}$ by placing $\mathcal{L}$ completely inside $\mathcal{M}^-$ below the threshold line. By the property of $\mathcal{A}^+$ and $\mathcal{O}^+$ regions, we have the following result:

Result 2 *If $\mathcal{L}$ intersects the boundary of both $\mathcal{A}^+$ and $\mathcal{O}^+$ regions, then for every point α on the surface of $\mathcal{T}$, there exists a point $\beta \in \mathcal{L}$ that can see the point α.*

Proof. $\mathcal{L}$ intersects the boundary of both $\mathcal{A}^+$ and O^+ regions implies that there exists a point on $\mathcal{L}$ that lies inside $\mathcal{A}^+$, and also there exists a point on $\mathcal{L}$ that lies inside $\mathcal{O}^+$. The result follows by the property of $\mathcal{A}^+$ and $\mathcal{O}^+$ regions. □

Observe that, below the threshold line, if the segment $\mathcal{L}$ entirely lies inside the region $\mathcal{M}^-$ then $\mathcal{L}$ intersects both $\mathcal{A}^+$ and $\mathcal{O}^+$ regions (see the shaded region in Figure 4a). Thus, Result 2 suggests the following:

Result 3 *Terrain $\mathcal{T}$ is weakly visible from $\mathcal{L}$ if and only if $\mathcal{L}$ lies completely inside the region $\mathcal{M}$.*

Proof. If part follows from the above discussion. To prove the only if part, suppose $\mathcal{L}$ properly intersects the boundary of $\mathcal{M}$ (i.e., part of $\mathcal{L}$ is outside $\mathcal{M}$). Without loss of generality, assume that $\mathcal{L}$ intersects the left boundary of $\mathcal{M}$. The left boundary of $\mathcal{M}$ is a poly-chain defined by the Minkowski sum of $\mathcal{L}$ and the boundary of $\mathcal{O}^+$ (see Figure 3a). If $\mathcal{L}$ intersects the left boundary, it cannot lie completely inside $(\mathcal{O}^+ \oplus \mathcal{L})$. So, $\mathcal{L}$ does not touch the boundary of $\mathcal{O}^+$. This implies that $\mathcal{L}$ does not intersect the e^+ region of at least one of the edges of $\mathcal{O}$; hence that edge is not visible from $\mathcal{L}$. $\square$

Definition 2 *Let a horizontal line at $y = y_0$ intersect the boundary of $\mathcal{M}$ at two points α and β. A cut on $\mathcal{M}$ at y_0 is defined as the line segment $[\alpha, \beta]$ and it is denoted by $\mathcal{W}(y_0)$.*

Definition 3 *A point $p \in \mathcal{M}$ is feasible if there exists a point $q \in \mathcal{L}$ such that if $\mathcal{L}$ is positioned coinciding the point q with the point p then entire $\mathcal{L}$ lies inside $\mathcal{M}$.*

Result 4 *Let y_p be the y-coordinate of a point $p \in \mathcal{M}$. Point p is feasible if and only if $|\mathcal{W}(y_p)| \geq 1$.*

Let y_r denote the y-coordinate of the critical point r. Observe that the length of $\mathcal{W}(y_r)$ is twice the length of $\mathcal{L}$ (see Figure 4a). Hence any point $p \in \mathcal{M}^+$ is feasible. Recall that $\mathcal{M}^-$ is a convex region bounded above by $\mathcal{W}(y_r)$. It may have an infeasible region at its bottom (see the unshaded region in Figure 4a). Observe that a point $p \in \mathcal{M}^-$ is infeasible if the length of $|\mathcal{W}(y_p)| < 1$. Since $\mathcal{M}^-$ is convex, the infeasible region lies at the bottom of $\mathcal{M}^-$. Thus we have the following result :

Result 5 *The bottom side of the feasible region of placing $\mathcal{L}$ in $\mathcal{M}^-$ is bounded by a horizontal line $\mathcal{W}(y^*)$ at $y = y^*$ such that the length of $\mathcal{W}(y)$ for any height $y \in [y^*, y_r]$ is at least ≥ 1 and there exists no $y < y^*$ such that $|\mathcal{W}(y)| \geq 1$.*

We remove the infeasible region from $\mathcal{M}^-$ by performing a linear scan of its boundary and let the feasible region thus obtained is denoted by $\widehat{\mathcal{M}^-}$. The feasible region of entire $\mathcal{M}$ is $\widehat{\mathcal{M}} = (\mathcal{M}^+ \cup \widehat{\mathcal{M}^-})$, see Figure 4b. Observe that $\widehat{\mathcal{M}}$ is a convex region that is unbounded from above and bounded below by a horizontal line segment (parallel to x-axis) of length ≥ 1.

Lemma 1 *A point $p \in \mathcal{M}$ is feasible if and only if $p \in \widehat{\mathcal{M}}$.*

Lemma 2 *Let τ^* be an optimal height watchtower, where γ_l^* is its top end point and γ_t^* is its bottom end point. Then, γ_l^* lies on the boundary of $\widehat{\mathcal{M}}$.*

Proof. From the above discussion we know that any point $p \in \widehat{\mathcal{M}}$ is feasible. Since the objective is to find the minimum vertical distance between $\mathcal{T}$ and the convex object $\widehat{\mathcal{M}}$, the height of τ^* is minimized if the end point γ_l^* is not in the interior of $\widehat{\mathcal{M}}$ (see Figure 4c). $\square$

To find an optimum placement of the watchtower, we vertically project the vertices of $\widehat{\mathcal{M}}$ on the boundary of the terrain $\mathcal{T}$ (see Figure 4c). These projected points and the vertices of $\mathcal{T}$ define the event points. We compute the height of τ at each event point. The minimum of these heights gives us the desired result. The platform $\mathcal{L}$ is placed with its one endpoint touching the boundary of $\widehat{\mathcal{M}}$ at that height. As the vertices of $\mathcal{T}$ are available in order, the boundaries of $\mathcal{A}^+$ and $\mathcal{O}^+$ are available in $O(n)$ time using a stack. Thus, the region $\mathcal{S}$, and hence the boundary of $\mathcal{M}^+$ (the portion of $\mathcal{M}$ above the threshold line) are also computable in $O(n)$ time. Similarly, the boundary of $\mathcal{M}^-$ is also computable in $O(n)$ time. By performing a linear scan of the boundary of $\mathcal{M}^-$, we can compute $\widehat{\mathcal{M}^-}$ in $O(n)$ time. Thus the computation of $\widehat{\mathcal{M}} = \mathcal{M}^+ \cup \widehat{\mathcal{M}^-}$ requires $O(n)$ time. As the complexity of $\mathcal{M}$ is $O(n)$, the complexity of $\widehat{\mathcal{M}}$ and the number of event points is $O(n)$. By processing the event points in order, we have the following.

Theorem 3 *The minimum height watchtower for a terrain in $\mathbb{R}^2$ can be computed in $O(n)$ time.*

3 Watchtower problem in $\mathbb{R}^3$

In $\mathbb{R}^3$, a terrain $\mathcal{T}$ with x-y plane as its base is a triangulated polyhedral surface such that any line vertical to the x-y plane hits the surface of $\mathcal{T}$ at most at one point. We use n to denote the number of vertices of $\mathcal{T}$; the number of edges and faces of $\mathcal{T}$ are both $O(n)$. Any plane parallel to the x-y plane will be referred to as the *horizontal plane* and any line perpendicular to the x-y plane will be referred to as a *vertical line*. We use $V_{\mathcal{T}}$, $E_{\mathcal{T}}$, $F_{\mathcal{T}}$ to denote the set of vertices, edges and faces of $\mathcal{T}$, respectively. Here the *platform* is a unit square plate $\mathcal{L}$ parallel to the x-y plane, and the two pairs of non-adjacent edges of $\mathcal{L}$ are parallel to x and y axis, respectively. The watchtower with acrophobic guard is formed by placing $\mathcal{L}$ on the top of a vertical line segment $\tau = [\gamma_t, \gamma_l]$, called the *tower*, whose bottom end-point γ_t touches the surface of $\mathcal{T}$ and the top end-point γ_l touches a point of $\mathcal{L}$. The problem is to identify the point γ_t on the surface of $\mathcal{T}$ and γ_l on the surface of $\mathcal{L}$ such that $\mathcal{T}$ is weakly visible from $\mathcal{L}$ and the height ($=|\gamma_t\gamma_l|$) of the tower τ is minimized.

The height of the of watchtower τ in $\mathbb{R}^3$ at a particular level (z-coordinate) is defined as follows: Vertically project the four edges of $\mathcal{L}$ on the surface of $\mathcal{T}$; thus, we have a region on the surface of $\mathcal{T}$ bounded by four polychains. Let θ be a point on the surface of $\mathcal{T}$ inside that region having maximum z-coordinate value. Now, vertically project θ at a point θ' on $\mathcal{L}$. Then the height of the tower τ is $(z(\theta') - z(\theta))$.

Next, we extend some of the concepts from the $\mathbb{R}^2$ case to $\mathbb{R}^3$.

Definition 4 *A face $f \in F_{\mathcal{T}}$ is said to be completely visible from a point $\alpha \in \mathbb{R}^3$ if for any point $\beta \in f$ the line segment $[\alpha, \beta]$ does not intersect the interior of $\mathcal{T}$.*

The plane containing a face $f \in F_{\mathcal{T}}$ splits $\mathbb{R}^3$ into two half-spaces, namely *positive region* f^+ and *negative region* f^-. The face f is invisible from f^- (assuming $f \in f^+$, $f^+ \cap f^- = \emptyset$). We now classify each face

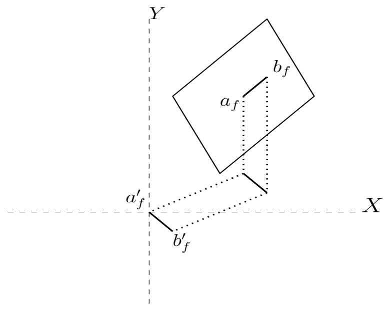


Figure 5: Method of classification of the faces of $\mathcal{T}$

$f \in F_{\mathcal{T}}$ into one of the following eight classes as follows. Let $[a_f, b_f]$ be a line segment of unit length, with $a_f \in f$ and $b_f \in f^+$, that is normal to the face f. Let, $[a'_f, b'_f]$ be the vertical projection of $[a_f, b_f]$ on x-y plane. Translate the line segment $[a'_f, b'_f]$ such that a'_f coincides with the origin of the coordinate system (see Figure 5). Now,

- if b'_f lies in the proper inside of the i-th quadrant in the projection, then $f \in F_i$, $i = 1, 2, 3, 4$,
- if b'_f lies at a point on the positive (resp. negative) part of the x-axis then $f \in F_{X+}$ (resp. $f \in F_{X-}$), and
- if b'_f lies at a point on the positive (resp. negative) part of the y-axis then $f \in F_{Y+}$ (resp. $f \in F_{Y-}$).

Thus, we have eight sets of faces, namely $F_1, F_2, F_3, F_4, F_{X+}, F_{X-}, F_{Y+}, F_{Y-}$. From now onwards, we use F_5, F_6, F_7, and F_8 to denote F_{X+}, F_{X-}, F_{Y+}, and F_{Y-}, respectively. Observe that, every pair of sets (F_i, F_j), $i, j \in \{1, \ldots, 8\}$, $i \neq j$, are disjoint, and $\bigcup_{i=1}^{8} F_i = F_{\mathcal{T}}$. Also we consider four sets as $Q_0 = F_1 \cup F_7$, $Q_1 = F_2 \cup F_6$, $Q_2 = F_3 \cup F_8$ and $Q_3 = F_4 \cup F_5$. We use $F_i^+ = \bigcap_{f \in F_i} f^+$; any point $\pi \in F_i^+$ can see all the faces $f \in F_i$. Finally, we define $\mathcal{S} = \bigcap_{i=1}^{8} F_i^+ = \bigcap_{f \in F} f^+$, the region bounded by the upper envelopes of all the planes $f \in F$. It is a convex polytope that is unbounded above (See Figure 6(a)). We refer the point $r \in \mathcal{S}$ having the minimum z-coordinate (say z_0) as the *critical point.* A horizontal plane G passing through the point r is referred to as the *threshold plane* (see Figure 6(a)).

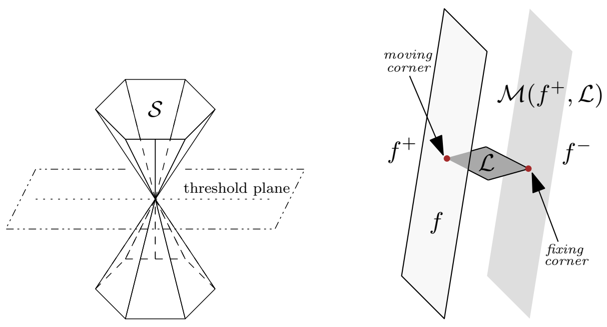


Figure 6: Demonstration of (a) $\mathcal{S}$ region, and (b) Minkowski sum of a plane and the platform $\mathcal{L}$

Result 6 *[7] From any point $\pi \in \mathcal{S}$ the entire terrain is visible.*

Definition 5 *Let R be a convex region in $\mathbb{R}^3$. We use B_R to denote the boundary surface (a polyhedron in $\mathbb{R}^3$) of the object R. Let $\mathcal{M}(B_R, O)$ denote the boundary surface of the Minkowski sum $R \oplus O$ of R with another convex object O.*

For a non-vertical face $f \in F_{\mathcal{T}}$, and the object $O = \mathcal{L}$, the Minkowski sum $\mathcal{M}(f^+, \mathcal{L})$ is obtained by moving one corner (called the *moving corner*) of $\mathcal{L}$ on the boundary surface of f^+ such that the entire $\mathcal{L}$ lies in the f^- region, (see Figure 6(b)).

Definition 6 *The corner of $\mathcal{L}$ opposite to the moving corner with respect to a face $f \in F_{\mathcal{T}}$ is referred to be as the* fixing corner.

Note that $\mathcal{M}(f^+,\mathcal{L})$ is the locus of the fixing corner of $\mathcal{L}$, and hence $\mathcal{M}(f^+,\mathcal{L})$ is a plane lying in f^- region. If $\mathcal{L}$ is placed above threshold plane by putting its fixing corner at $\mathcal{M}(f^+,\mathcal{L})$, then $\mathcal{L}$ will touch $\mathcal{S}$, and the entire terrain $\mathcal{T}$ will be visible from $\mathcal{L}$. We define $\mathcal{M}(F_i^+,\mathcal{L}) = \bigcap_{f\in F_i} \mathcal{M}(f^+,\mathcal{L})$, the boundary of the envelope of the planes $\mathcal{M}(f^+,\mathcal{L})$ for all $f \in F_i$. Finally, we compute $\mathcal{M} = \bigcap_{i=1}^{8} \mathcal{M}(F_i^+,\mathcal{L})$, which is the envelope of all the planes $\mathcal{M}(f^+,\mathcal{L})$ for $f \in F_{\mathcal{T}}$. $\mathcal{M}$ is a convex polytope, which is unbounded above. We will use $\mathcal{M}^+$ (resp. $\mathcal{M}^-$) to denote the portion of $\bigcap_{i=1}^{8} \mathcal{M}(F_i^+,\mathcal{L})$ above (resp. below) the threshold plane G.

Let us first consider $\mathcal{M}^+$. Observe that $\mathcal{M}^+ = \mathcal{M}(\mathcal{S},\mathcal{L})$. It properly contains $\mathcal{S}$. The boundaries of $\mathcal{M}^+$ and $\mathcal{S}$ are mutually parallel, and the platform $\mathcal{L}$ exactly (horizontally) fits in the annulus $(\mathcal{M}^+ \setminus \mathcal{S})$. Observe that, if $\mathcal{L}$ lies completely inside $\mathcal{M}^+$, then at least one point $\mathcal{L}$ must lie inside (or may touch) $\mathcal{S}$. Thus, from Result 6, we have

Result 7 *If $\mathcal{L}$ lies completely inside $\mathcal{M}^+$ then $\mathcal{T}$ is completely visible from $\mathcal{L}$.*

As $\mathcal{S} \neq \emptyset$ above the threshold plane, we have positions of placing $\mathcal{L}$ such that it lies entirely inside $\mathcal{M}^+$. As in Section 2, here also we show that though $\mathcal{S} = \emptyset$ below the threshold plane, there are regions to place $\mathcal{L}$ below the threshold plane such that the entire $\mathcal{T}$ is weakly visible from $\mathcal{L}$. Observe that, $\mathcal{M}^-$ is also a convex region (consult Figure 4a). Here, if we can place $\mathcal{L}$ such that it intersects the f^+ regions of all the faces $f \in F_{\mathcal{T}}$, then every point on the surface of $\mathcal{T}$ is weakly visible from $\mathcal{L}$. This implies $\mathcal{T}$ is weakly visible from $\mathcal{L}$ if $\mathcal{L}$ is placed completely inside the boundary of $\mathcal{M}^-$. So, our next objective is to identify the feasible region of $\mathcal{M}^-$ such that if an appropriate corner of $\mathcal{L}$ is placed inside $\mathcal{M}^-$ then $\mathcal{T}$ becomes weakly visible from $\mathcal{L}$. From now onwards, we will use $H(z)$ to denote a planar convex region obtained by the intersection of $\mathcal{M}^-$ and the horizontal plane at height z. From the convexity of $\mathcal{M}^-$, we have the following result.

Result 8 *The bottom side of the feasible region of placing $\mathcal{L}$ in $\mathcal{M}^-$ is bounded by a horizontal plane $H(z^*)$ at $z = z^*$ such that the region $H(z)$ for any height $z \in [z^*, z_0]$ completely contains $\mathcal{L}$, and there exists no $z < z^*$ such that $\mathcal{L}$ entirely lies inside $H(z)$ (see Figure 7).*

To obtain $H(z^*)$ we need the following results:

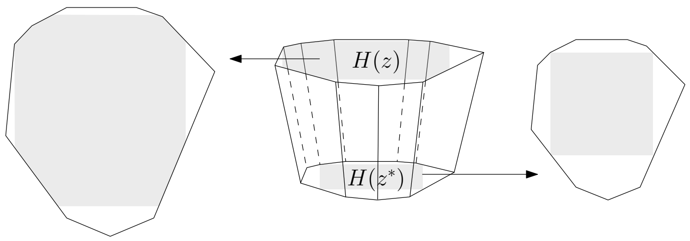


Figure 7: A portion of $\mathcal{M}^-$ bounded by two horizontal planes. Top view of $H(z)$ and $H(z^*)$ are shown on the left and right, respectively. Shaded region denotes the feasible region for the placement of $\mathcal{L}$.

Result 9 *One can test whether $\mathcal{L}$ entirely fits in a convex polygonal region $\mathcal{C}$ by formulating the problem as a four variate linear programming problem, and it can be solved in $O(n)$ time.*

Proof. Let the coordinates of the bottom-left and the top-right corners of the diagonal of $\mathcal{L}$ be (x_1, y_1) and (x_2, y_2), respectively. We need to test whether all the four points (x_1, y_1), (x_1, y_2), (x_2, y_1) and (x_2, y_2) lie inside $\mathcal{C}$. In other words, each point must satisfy the linear inequality with respect to every edge of $\mathcal{C}$. In addition, $x_2 - x_1 = y_2 - y_1$. Assuming the coordinate system such that the entire $\mathcal{M}^-$ lies in the first quadrant, we also have the non-negativity constraints for the variables x_1, x_2, y_1, y_2. This is a 4-variate linear programming for the constraint satisfaction, which can be solved in $O(n)$ time, where n is the number of sides of $\mathcal{C}$. They are bounded by the number of faces of $\mathcal{T}$. □

The above result suggests the following procedure for computing $H(z^*)$:

- Perform a binary search among the vertices of $\mathcal{M}^-$ with respect to their z-coordinate.
- At each chosen vertex v, compute the intersection of a horizontal plane $h(v)$ passing through v and the polytope $\mathcal{M}^-$, which is a convex polygonal region $\mathcal{C}(v)$.
- One can test whether $\mathcal{L}$ fits in $\mathcal{C}(v)$ in $O(n)$ time by formulating the problem as a four variate linear programming problem (see Result 9)
- Thus, we can identify two vertices v' and v'' of $\mathcal{M}^-$ such that $\mathcal{L}$ can be placed inside $\mathcal{C}(v')$ but $\mathcal{L}$ cannot be placed completely inside $\mathcal{C}(v'')$, and there exists no vertex v of $\mathcal{M}^-$ satisfying $z(v') < z(v) < z(v'')$.
- Now, we can obtain the exact height $z^* \in [z(v''), z(v')]$ such that the polygon $\mathcal{C}(z^*)$ obtained

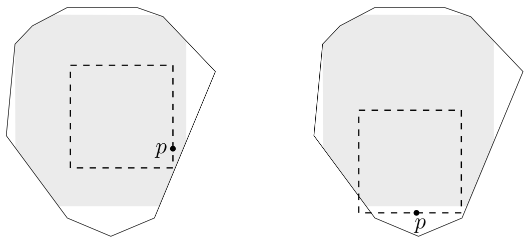


Figure 8: Illustration of feasible region (shaded) of $H(z)$: (a) p is feasible, (b) p is infeasible.

by the horizontal plane at height z^* contains $\mathcal{L}$, and for any height $z' < z^*$, the polygon $\mathcal{C}(z')$ does not contain $\mathcal{L}$ by the following result.

Result 10 *Let v' and v'' be two consecutive vertices of $\mathcal{M}^-$ as above. Consider the glass shaped convex polyhedron μ obtained by slicing $\mathcal{M}^-$ at the heights $z(v')$ and $z(v'')$. The top and bottom faces of μ are bounded by two horizontal planes at height z' and z'', respectively, such that the top face contains the unit square platform $\mathcal{L}$, but the bottom face does not contain the platform $\mathcal{L}$. We can formulate the problem of identifying a height z^* ($z'' < z^* \leq z'$) such that the polygon $\mathcal{C}(z^*)$ obtained by the horizontal plane at height z^* contains $\mathcal{L}$, and for any height $z < z^*$ the polygon $\mathcal{C}(z)$ does not contain $\mathcal{L}$ as a linear programming problem and can be solved in $O(n)$ time.*

Proof. We formulate this problem also as an optimization version of the linear programming problem. Let the coordinate of the two end-points of the diagonal of $\mathcal{L}$ fitted in μ at height z be (x_1, y_1, z) and (x_2, y_2, z). The LP formulation of the problem is to minimize z subject to the constraints that the four vertices (x_1, y_1, z), (x_1, y_2, z), (x_2, y_1, z) and (x_2, y_2, z) lies inside μ. If μ has k faces, we have $4k$ linear constraints; in addition, we have $x_2 - x_1 = y_2 - y_1$, $z(v'') \leq z \leq z(v')$, and all the five variables are positive. This also can be solved in $O(n)$ time to obtain the optimum value of z^*. □

By Result 8, the portion of $\mathcal{M}^-$ below $z = z^*$ is infeasible. With little abuse of notation, from now onwards, we will use $\mathcal{M}^-$ to denote the portion of $\mathcal{M}^-$ inside the slab bounded by the horizontal plane at $z = z_0$ and $z = z^*$. We know that the intersection region $H(z)$ of a horizontal plane at any height $z \in (z^*, z_0]$ with $\mathcal{M}^-$ completely contains $\mathcal{L}$. However, the entire region of $H(z)$ is not feasible (see Definition 7) for placing $\mathcal{L}$. We now discuss the method of identifying the feasible region (of placing $\mathcal{L}$) on $H(z)$.

Definition 7 *A point $p \in H(z)$ is said to be feasible if there exists a point $q \in \mathcal{L}$ such that if $\mathcal{L}$ is positioned coinciding the point q with the point p then entire $\mathcal{L}$ lies inside $H(z)$ (see Figure 8).*

We now discuss the problem of computing the feasible region of $H(z)$. Let $E_{H(z)}$ denotes the set of edges on the boundary of $H(z)$. Consider a pair of edges e_1 and e_2 of $E_{H(z)}$. Let ℓ_1 and ℓ_2 be the half-lines containing e_1 and e_2 respectively such that the cone $\chi(e_1, e_2)$ formed by ℓ_1 and ℓ_2 contains $H(z)$. In Lemmas 4 and 5, we first characterize the feasible region in $\chi(e_1, e_2)$. The intersection of the feasible regions for pairs of edges in $E_{H(z)}$ will lead us to the feasible region of $H(z)$.

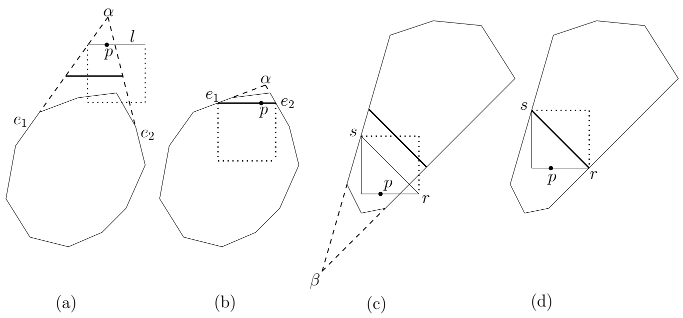


Figure 9: (a) and (b): Illustration of proof of Lemma 4 - (a) p is in the infeasible region of $\chi(e_1, e_2)$, and (b) p is in the feasible region of $\chi(e_1, e_2)$; (c) and (d): Illustration of proof of Lemma 5

Lemma 4 *For a pair of edges e_1 and e_2 of two sets where $e_1 \in Q_i$ and $e_2 \in Q_{\overline{i+1} \bmod 4}$, a point $p \in \chi(e_1, e_2)$ is feasible for the placement of $\mathcal{L}$ (see Definition 7) inside $\chi(e_1, e_2)$ if and only if the segment ℓ obtained by the intersection of the cone $\chi(e_1, e_2)$ and a horizontal (resp. vertical) line (depending on i is even (resp. odd)) passing through p has length greater than or equal to 1.*

Proof. If part follows from the fact that the slope of e_1 and e_2 have different signs. If an edge of $\mathcal{L}$ is matched with the segment ℓ, the entire $\mathcal{L}$ lies inside the truncated region of the cone $\chi(e_1, e_2)$ by ℓ.

The only if part follows from (i) if $p \notin \chi(e_1, e_2)$, p is infeasible, and (ii) if p lies inside the triangular region formed by the cone $\chi(e_1, e_2)$ and the line segment ℓ then the horizontal (resp. vertical) line incident on i, if i is even (resp. odd), through p with the cone $\chi(e_1, e_2)$ is of length less than one, and if a point $q \in \mathcal{L}$ is made coincident with p then the unit length horizontal line segment through q inside $\mathcal{L}$ goes outside $\chi(e_1, e_2)$ (see Figure 9a and 9b)). □

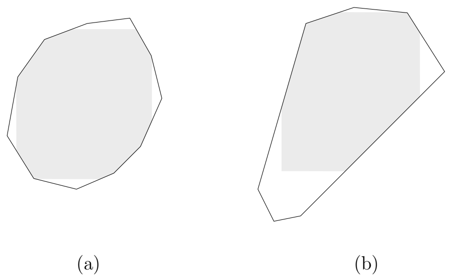


Figure 10: Feasible region $\widehat{H}(z)$ of $H(z)$: two instances are demonstrated, where $\widehat{H}(z)$ is the shaded region.

Lemma 5 *For a pair of edges e_1 and e_2 of two sets where $e_1 \in Q_i$ and $e_2 \in Q_{\overline{i+2} \bmod 4}$, a point $p \in \chi(e_1, e_2)$ is feasible for the placement of $\mathcal{L}$ inside $\chi(e_1, e_2)$ if and only if the length of both the line segments making an angle $\frac{\pi}{4}$ from the projections q_1 and q_2 of p on the lines ℓ_1 and ℓ_2, respectively, are of length greater than or equal to $\sqrt{2}$.*

Proof. Since the slopes of e_1 and e_2 are of the same sign, the platform $\mathcal{L}$ may not fit inside the cone $\chi(e_1, e_2)$ by aligning a side of $\mathcal{L}$ through the point p (see Figure 9c). In order to fit $\mathcal{L}$ inside the cone $\chi(e_1, e_2)$ keeping $p \in \chi(e_1, e_2)$, one may need to fit the diagonal of $\mathcal{L}$ (of length $\sqrt{2}$) at the horizontal projection of p (a point r) on either e_1 or e_2 depending on whichever is closer. In other words, a segment of length $\sqrt{2}$ making an angle $\frac{\pi}{4}$ can be placed inside $\chi(e_1, e_2)$ aligning one of its end-points at the point r (see Figure 9d). Thus, the if part follows.

To prove the only if part assume that p is feasible inside $\chi(e_1, e_2)$ and a segment of length $\sqrt{2}$ making an angle $\frac{\pi}{4}$ cannot fit inside $\chi(e_1, e_2)$ by aligning one of its end-points at the point r (on the line, say ℓ_1). Thus, the unit-length horizontal line segment through p with one of its end-points coinciding at r intersects ℓ_2. Thus, p cannot be on the boundary of $\mathcal{L}$. If p lies properly inside $\mathcal{L}$ then the said boundary moves closer to the point of intersection of ℓ_1 and ℓ_2 than the previous case, and hence it would intersect either ℓ_1 or ℓ_2 or both. Thus, we have the contradiction that p is feasible inside $\chi(e_1, e_2)$. □

Lemmas 4 and 5 suggest the shape of the feasible region $\widehat{H}(z)$ of $H(z)$, where $z \in [z_0, z^*]$, in $\mathcal{M}^-$ as follows:

Observation 1 *$\widehat{H}(z)$ has two horizontal edges e_N, e_S at its North and South sides, respectively, and two vertical edges e_E, e_W at its East and West sides, respectively. The length of each of the edges in $\{e_N, e_E, e_S, e_W\}$ is greater than or equal to 1. The edges e_S and e_W may or may not be adjacent on the boundary of $\widehat{H}(z)$. In the latter case e_S and e_W can have length ≥ 1, and they are connected by a polygonal chain, which is a part of the boundary of $H(z)$ in the third quadrant (see Figure 10a). In the former case the length of e_N and e_E are exactly equal to 1, and their point of contact lies inside the interior of $H(z)$ (see Figure 10b). The same holds for each of the pairs (e_W, e_N), (e_N, e_E) and (e_E, e_S).*

During the presentation of the algorithm, the edges e_N, e_E, e_S and e_W may sometimes be referred to as the extreme edges in North, East, South and West sides, respectively.

Extraction of feasible region in $\mathbf{H(z)}$: Let P_1, P_2, P_3, and P_4 be the polychains of $H(z)$ that correspond to the sets Q_0, Q_1, Q_2 and Q_3, respectively. We can compute $\widehat{H}(z) \subseteq H(z)$ by executing the following three steps:

Step 1: We consider each pair of adjacent quadrants P_i and $P_{\overline{i+1} \bmod 4}$. Consider every pair of edges (e_a, e_b) of $H(z)$, $e_a \in P_i$ and $e_b \in P_{\overline{i+1} \bmod 4}$; if they are not adjacent (i.e., do not have a common vertex) in $H(z)$, we extend e_a and e_b to meet at a point α (see Figures 9a and 9b). Let e'_a and e'_b be those two half-lines (originating at α). Let ℓ_{ab} be a horizontal line segment of unit length that touches both e'_a and e'_b. Thus, the cone $\chi(e_1, e_2)$ splits into two parts: (i) a triangle and (ii) an open polygon Π_{ab} bounded by e'_a, e'_b, ℓ_{ab}. Part (i) is infeasible and part (ii) is feasible for placing $\mathcal{L}$. For each pair of adjacent quadrants, consider all possible pairs of edges taking one from each of these quadrants, and generate such polygons. Let $E_{\mathcal{N}}$ be the set of the edges ℓ_{ab} of all the generated polygons Π_{ab}. These open polygons will be referred to as *type-1 polygons*.

Step 2: Now, consider each pair of diagonally opposite quadrants P_i and $P_{\overline{i+2} \bmod 4}$. Consider every pair of edges $e_a \in P_i$ and $e_b \in P_{\overline{i+2} \bmod 4}$, and extend them to meet at a point β (see Figure 9c). As earlier, name these two half-lines as e'_a and e'_b, respectively. Now, draw a line segment of length $\sqrt{2}$ and making an angle $\frac{\pi}{4}$ with the positive (resp. negative) direction of the x-axis depending on whether $i = 1$ or 2 touching e'_a and e'_b at r and s respectively (see Figure 9d). Draw a horizontal line segment ℓ^1_{ab} at r and a vertical line segment ℓ^2_{ab} at s that touch at a point π inside $H(z)$. Now, we have an open polygon with four edges $\ell^1_{ab}, \ell^2_{ab}, e'_a$, and e'_b, called *type-2 polygon*. Generate all possible type-2 polygons. Let $E_{\mathcal{O}}$ be the set of edges ℓ_{ab} of all the generated polygons.

Step 3: Finally, we obtain $\widehat{H}(z)$ as the intersection of all possible type-1 polygons, all possible type-2 polygons and $H(z)$. The significance of the sets $E_{\mathcal{N}}$ and $E_{\mathcal{O}}$ will be discussed in Lemma 6.

We denote the set of edges on the boundary of $\widehat{H}(z)$ by $E_{\widehat{H}(z)}$. So any edge $e \in E_{\widehat{H}(z)}$ is either part of an edge $e' \in E_{H(z)}$ or a newly added edge from $\{E_{\mathcal{N}} \cup E_{\mathcal{O}}\}$.

Lemma 6 *Any point $p \in H(z)$ is feasible if and only if $p \in \widehat{H}(z)$.*

Proof. First we show that if $p \in \widehat{H}(z)$ then p is a feasible point of $H(z)$. We need to show that for every point $p \in \widehat{H}(z)$ there exists a point q on the platform $\mathcal{L}$ such that if q and p coincides in a placement of $\mathcal{L}$ then every point in $\mathcal{L}$ lies in $\widehat{H}(z)$.

Let us draw a horizontal half-line ℓ from p to its right side, that may hit (i) the polychain P_1, or (ii) the edge e_E, or (iii) the polychain P_4. In Case (i) (resp. (iii)), let θ be the point on the edge $e \in P_1$ (resp. P_4) where ℓ hits. Due to the presence of e_N, e_S, e_W and e_E, it can be ensured that if the top-right (resp. bottom-right) corner is placed on θ then all the other corners of $\mathcal{L}$ lie inside $\widehat{H}(z)$. In Case (ii), let ℓ hits e_E at the point θ. We move upward until the top end-point of e_E is reached. We place the top-right corner of ℓ at that point, and can ensure that $\mathcal{L}$ lies inside $\widehat{H}(z)$ arguing as above.

Now we show that if $p \in H(z)$ is feasible then $p \in \widehat{H}(z)$. For contradiction, assume that $p \notin \widehat{H}(z)$. Since, $p \in H(z)$ but $p \notin \widehat{H}(z) \implies p \in (H(z) \setminus \widehat{H}(z))$. Then, surely there exist an edge $e \in E_{\widehat{H}(z)}$ such that, either $e \in E_{H(z)}$ or $e \in \{E_{\mathcal{N}} \cup E_{\mathcal{O}}\}$, and $p \in e^-$. If $e \in E_{H(z)}$, then p must be outside of $H(z)$ which is a contradiction. If $e \in \{E_{\mathcal{N}} \cup E_{\mathcal{O}}\}$, e must bound an infeasible region of some pair of edges say $e_1, e_2 \in E_{H(z)}$. Since $p \in e^-$, then p is in the infeasible region of $\chi(e_1, e_2)$, where $\mathcal{L}$ cannot be placed, which is a contradiction that p is feasible for $H(z)$. □

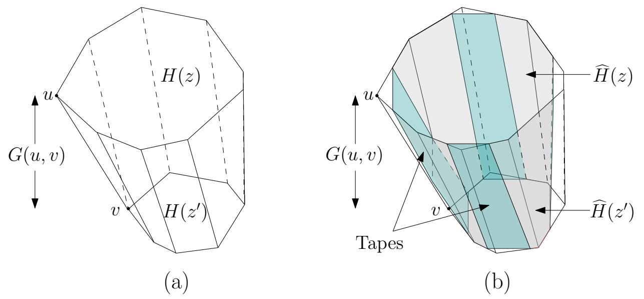


Figure 11: (a) A glass $G(u,v)$, and (b) Feasible region of $G(u,v)$: $\widehat{H}(z)$, $\widehat{H}(z')$ and tapes

The characterization of the feasible region in $H(z)$ of Lemma 6 will be used in the computation of the feasible region of $\mathcal{M}^-$. As we will see, the intersection of the feasible region of the cones corresponding to every pair of boundary planes of $\mathcal{M}^-$ will determine its feasible region. In order to compute the feasible region $\widehat{\mathcal{M}^-}$ of $\mathcal{M}^-$, we process every pair of consecutive vertices u and v of $\mathcal{M}^-$ with respect to their z-coordinate values. We define a *glass* $G(u,v)$, which is the intersection of $\mathcal{M}^-$ and a slab U defined by two horizontal planes at two consecutive vertices u and v (see Figure 11a).

For each glass $G(u,v)$ of $\mathcal{M}^-$, we compute the feasible region $\widehat{G}(u,v)$ as follows: Let $H(z)$ (resp. $H(z')$) be the top (resp. bottom) face of the glass $G(u,v)$, where z (resp. z') are the z-coordinate of the vertex u (resp. v). Here also two types of open polyhedron will be generated. Type 1 polyhedrons are obtained by considering each pair of adjacent quadrants $(Q_i, Q_{\overline{i+1} \bmod 4})$ and then consider every pair of faces (f, f'), $f \in Q_i$ and $f' \in Q_{\overline{i+1} \bmod 4}$. Draw a plane passing through two line segments parallel to the x (resp. y) axis depending on i is odd (resp. even) that creates unit length edges of $\widehat{H}(z)$ (resp. $\widehat{H}(z')$). The type 2 open polyhedrons are bounded by four planes where two planes are mutually orthogonal with respect to the x-y plane, and are obtained by considering the diagonal of $\mathcal{L}$ as was done in $\mathbb{R}^2$. Finally, we compute the intersection of all the type-1 and type-2 polyhedrons and $G(u,v)$ to compute the feasible region $\widehat{G}(u,v)$ inside the glass $G(u,v)$ (see Figure 11b). Since the feasible region $\widehat{H}(z')$ is same for each pair of consecutive glasses $G(u,v)$ and $G(v,w)$ of $\mathcal{M}^-$, one can obtain the feasible region $\widehat{\mathcal{M}^-}$ by merging the feasible regions of all the glasses of $\mathcal{M}^-$.

Lemma 7 *The feasible region of all the glasses in $\mathcal{M}^-$ can be computed in $O(n^2)$ time.*

Proof. We process the glasses in $\mathcal{M}^-$ from top to bottom with respect to the z-coordinate of its vertices.

For the first glass $G(u,v)$[2], let us consider the computation of the right-most face(s)[3] that is parallel to the y-axis. We have $O(n^2)$ such faces obtained from $O(n^2)$ type-1 and type-2 polyhedrons. We compute the envelope $\mathcal{E}(u,v)$ of these faces closer to the vertical line through the critical point r in time linear in the num-

[2] u corresponds to the top face of $\mathcal{M}^-$ and v is the next vertex with respect to the z-coordinate values of the vertices

[3] We may have multiple faces whose projection on the x-y plane is parallel to the y axis, and bound the right side of the feasible region of $\mathcal{M}^-$; these faces make different angles with the x-y plane.

ber of faces. The shape of this envelope is a *tape* like structure. The computation of the tapes in the left, top and bottom sides of $\widehat{G}(u, v)$ is analogous. Merging the generated faces with the faces of the glass can also be done in time linear in the number of faces of the glass.

From the next glass $G(v, w)$ onwards, observe that we may have exactly one new face f' that appears/disappears in this glass. Thus, we may have at most n many y-axis parallel new faces generated. The computation of the tape in the right side of the feasible region of this glass needs the computation of the envelope of the extension of $\mathcal{E}(u, v)$ and the $O(n)$ new y-axis parallel faces that are generated due to the appearance of the new face f' in the glass $G(v, w)$. Thus $O(n)$ time is needed to compute the envelope $\mathcal{E}'(v, w)$ of these new y-axis parallel faces, and finally another $O(n)$ time is needed to compute the envelope $\mathcal{E}(v, w)$ by merging $\mathcal{E}(u, v)$ and $\mathcal{E}'(v, w)$. Since the number of vertices of $\mathcal{M}^-$ is $O(n)$, the overall time complexity bound for computing $\widehat{\mathcal{M}^-}$ follows. □

Data structure: We can store $\mathcal{M}^+$ and $\widehat{\mathcal{M}^-}$ as a planar map on the x-y plane; its each vertex is attached with the z-coordinate of the corresponding vertex in $\mathcal{M}^+$ and $\widehat{\mathcal{M}^-}$, respectively, and each face is associated with the moving corner (top-left/top-right/bottom-left/bottom-right) information of $\mathcal{L}$. This information will be used to decide whether to place $\mathcal{L}$ at a point q on the boundary of $\mathcal{M}^+$ and $\widehat{\mathcal{M}^-}$.

Computation of minimum height watchtower for $\mathcal{T}$: We have computed the feasible region $\widehat{\mathcal{M}}$ of $\mathcal{M}$ by merging $\mathcal{M}^+$ and the feasible region $\widehat{\mathcal{M}^-}$ of $\mathcal{M}^-$. As in Section 2, here also, we first vertically project all the vertices and edges of $\widehat{\mathcal{M}}$ to get a set $V_{\widehat{\mathcal{M}}}$ of points on the surface of $\mathcal{T}$. It needs to mention that $V_{\widehat{\mathcal{M}}}$ contains the projection of vertices of $\widehat{\mathcal{M}}$ as well as contains the intersections of the projection of edges of $\widehat{\mathcal{M}}$ with the edges of $\mathcal{T}$. Now, the event points are $\widehat{V} = V_{\widehat{\mathcal{M}}} \cup V_{\mathcal{T}}$. We project each of these event points on the surface of $\widehat{\mathcal{M}}$. Thus, for each event point γ, we have a tower $\tau(\gamma)$. We choose the tower whose length is minimum among these towers for the placement of the platform $\mathcal{L}$.

Lemma 8 *The number of vertices in the feasible region of $\mathcal{M}$ is $O(n)$.*

Proof. Size of the set $F_{\mathcal{T}}$ is $O(n)$. $\mathcal{M}$ is the intersection region of Minkowski planes corresponding to each face $f \in F_{\mathcal{T}}$. Since, $\mathcal{M}$ is the intersection of $O(n)$ number of half-planes, the size of $\mathcal{M}$ is $O(n)$ [7]. While computing the feasible region $\widehat{\mathcal{M}^-}$ of $\mathcal{M}^-$, we have merged the feasible region of each pair of consecutive glasses. The number of faces whose projection on the x-y plane are not axis-aligned is $O(n)$ since these are (part of) the original faces of $\mathcal{M}$. We now need to count the number of new faces (whose projection on the x-y plane are axis-aligned) that are generated in $\widehat{\mathcal{M}^-}$.

Let us consider the faces whose projection of the x-y plane is parallel to the y-axis and keeps $\mathcal{M}^-$ to its right side. These form a continuous *tape* such that at every height (z-coordinate), the width of the tape is greater than or equal to 1. Observe that the intersection of this tape with each face is a single line segment. Thus, the number of edges (and hence the number of vertices) of this tape is $O(n)$. The same argument holds for the other three tapes bounding $\mathcal{M}^-$ to its South, West and North sides, respectively. Thus, the result follows. □

Theorem 9 *Given a terrain $\mathcal{T}$ consisting of n vertices, we can compute in $O(n^2)$ time a watchtower τ of minimum height with its base on $\mathcal{T}$ and the location of an axis-parallel unit-square plate $\mathcal{L}$ placed on the top of τ such that $\mathcal{T}$ is weakly visible from $\mathcal{L}$.*

Proof. Correctness and the optimality of the height of the tower follow from the above discussion. Now we analyze the overall time complexity by analyzing each of the tasks.

Task 1: Computing the Minkowski sum of each face $f \in \cup_{i=1}^{8} F_i^+$ and $\mathcal{L}$ needs $O(1)$ time. $\mathcal{M}$ is the intersection of a set of half-spaces which needs $O(n \log n)$ time to compute, and its combinatorial complexity is $O(n)$ [7]. By Lemma 7, the time complexity of computing the feasible region $\widehat{\mathcal{M}^-}$ of $\mathcal{M}^-$ is $O(n^2)$. Thus computation of $\widehat{\mathcal{M}}$ takes $O(n^2)$ time.

Task 2: By Lemma 8, the number of vertices and faces of $\widehat{\mathcal{M}}$ is $O(n)$. We triangulate the surface of $\widehat{\mathcal{M}}$. Project $\widehat{\mathcal{M}}$ on the terrain $\mathcal{L}$ obtaining an arrangement consisting of $O(n^2)$ planar cells, where each cell has an $O(1)$ complexity. Now, the minimum height watchtower for each cell can be computed in $O(1)$ time.

Task 3: Finally, the placement of the plate $\mathcal{L}$ above the tower at a point q on the surface of $\widehat{\mathcal{M}}$ so that it completely lies in $\widehat{\mathcal{M}}$ can be done in $O(1)$ time as follows:

- If q is on a face ϕ of $\widehat{\mathcal{M}}$, then we place the corner of $\mathcal{L}$ that is the fixing corner (see Definition 6) attached with the face ϕ.
- If q is on a tape of $\widehat{\mathcal{M}^-}$, we align the corresponding

boundary of $\mathcal{L}$ with q. The point on the boundary of $\mathcal{L}$ that is to be aligned with q can be computed in $O(1)$ time by considering the two boundary edges of the tape on which q lies.

Thus, the overall time complexity of the algorithm is dominated by that of Task 1. □

4 Conclusions

In this paper, we study a new variant of a watchtower problem where the terrain is guarded by a unit-size square platform that resides on the top of the tower. We considered the problem of minimizing the height of the tower. One can consider several variants of this problem, including having more than one watchtower and/or having various platform shapes. We believe that our algorithm for the terrain in $\mathbb{R}^3$ is not optimal. It may be possible to find an $o(n^2)$ time algorithm by further exploiting the structure of the feasible region below the threshold plane for the case of a square platform.

References

[1] Pankaj K. Agarwal, Sergey Bereg, Ovidiu Daescu, Heim Kaplan, Simeon Ntafos, Micha Sharir, and Binhai Zhu. Guarding a terrain by two watchtowers. *Algorithmica*, 58:352–390, 2010.

[2] Prosenjit Bose, Thomas Shermer, Godfried Toussaint, and Binhai Zhu. Guarding polyhedral terrains. *Computational Geometry*, 7:173–185, 1997.

[3] Richard Cole and Micha Sharir. Visibility problems for polyhedral terrians. *Journal of Symbolic Computation*, 7:11–30, 1989.

[4] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30:852–865, 1983.

[5] Micha Sharir. The shortest watchtower and related problems for polyhedral terrains. *Information Processing Letters*, 29(5):265–270, 1988.

[6] Nitesh Tripathi, Manjish Pal, Minati De, Gautam Das, and Subhas C Nandy. Guarding polyhedral terrain by k-watchtowers. In *International Workshop on Frontiers in Algorithmics*, pages 112–125. Springer, 2018.

[7] Binhai Zhu. Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time. *Computational Geometry*, 8(4):181–193, 1997.

Constrained Obnoxious Facility Location on a Line Segment

Vishwanath R. Singireddy* Manjanna Basappa*

Abstract

In this paper we study two restricted variations of the obnoxious facility location problem in the plane, given as follows. Given a line segment $\overline{pq}$ and a set $P = \{p_1, p_2, p_3, \ldots, p_n\}$ of n demand points in the plane, and a positive integer k, pack k maximum-radius congruent disks centered on $\overline{pq}$ such that no point of P lies inside any of these disks. We first prove that the decision version of this problem can be solved in linear time and then propose an $(1-\epsilon)$-factor approximation algorithm for the problem, that runs in $O((n+k)\log(\frac{||pq||}{2(k-1)\epsilon}))$ time if the points are given in order, where $||pq||$ is the length of the line segment $\overline{pq}$ and $\epsilon > 0$.

Another restricted problem is the *minsum* obnoxious facility location problem, in which we are given a line segment $\overline{pq}$ and a set $P = \{p_1, p_2, p_3, \ldots, p_n\}$ of n weighted demand points (having weights $w_1, w_2, \ldots, w_n$, respectively) in the plane, an integer k, and a distance λ, the goal is to pack k disks of radius λ centered on the segment $\overline{pq}$ such that the sum of weights of the points covered by the union of these disks is minimal. For $k = 1$ we show that this problem can be solved in $O(n \log n)$ time, and for any $k > 0$ we give a dynamic programming solution that runs in $O(n^3 k)$ time.

1 Introduction

The obnoxious facilities (such as nuclear plants, garbage dump yards, airports, industries from which toxic substances are released, etc.) need to be placed as much as possible away from the other facilities such as residential areas, hospitals, fire stations, post offices, etc. There is a requirement for heavy transportation for these obnoxious facilities. Hence many of these facilities are located on the sides of highway roads. Locating places for such obnoxious facilities on highways such that they are away from other non-obnoxious facilities is an essential problem to solve.

In most spatial location problems, the facilities are located as close as possible to clients such that the clients will get service by traveling less distance. In the case of obnoxious facilities, these facilities have to be placed as far as possible from the other communities to have the minimum nuisance generated by these obnoxious facilities.

The obnoxious facility location problems are modeled in different ways. The most common way is maximizing the cumulative minimum distance between obnoxious facilities and other non-obnoxious facilities in a given location. Church and Garfinkel [6] introduced the obnoxious p-median problem of locating p facilities such that the cumulative minimum distance from non-obnoxious facilities to p obnoxious facilities is maximized. The obnoxious p-median problem is modeled as p-max-sum problem and is proved `NP-hard` [1]. Drezner and Wesolowsky [9] formulated a variant of the obnoxious facility location problem: locating an obnoxious facility that is as far as possible from arcs and nodes of a network. They gave an $(1-\epsilon)$-approximation algorithm that runs in $O(a^3 \log(1/\epsilon)$ time for the weighted version of the problem, where a is the number of arcs in the network. Later, its running time was improved by Michael [7] with a slight modification of the problem by considering rectilinear (grid city) network and reducing its running time to $O(a^2 \log n \log(1/\epsilon))$ time, where n is the number of nodes in the network. Colmenar et al. [3] gave an approximation algorithm for obnoxious p-median problem on a general network using a heuristic method known as greedy randomized adaptive search procedure. Gokalp [5] gave an iterative greedy algorithm that produces a high-quality solution within a short time.

Another variation of the obnoxious facility location problem studied in the literature is the minimum-sum obnoxious facility location problem, in which for a given set of weighted points, we are required to place undesirable facilities that cover a subset of these points, sum of weights of which is minimized. This kind of problem is motivated by the situation where some obnoxious facilities are required to be placed in a crowded area. These facilities affect their close neighborhood area. This neighborhood area may be approximated to be a disk or a rectangle and a crowded region may be approximated to a point with its weight equal to the number of people in it. Now, we would like to place the facilities so that the total number of people who will be affected by these facilities is minimized. Drezner and Wesolowsky [8] first studied this problem that positions only a single facility by modeling this facility as a rectangle or a circle. Their algorithm runs in $O(n^2)$ time in

*Computer Science & Information Systems Dept., Birla Institute of Technology & Science Pilani, Hyderabad Campus, India `{p20190420,manjanna}@hyderabad.bits-pilani.ac.in` both the rectangular and circular cases. Katz et al., [4] later on improved the running time to $O(n \log n)$ for the rectangular case. In this paper, we study a variation of this problem, which may be considered a generalization of this problem in one aspect, and a restricted version of it in other aspects.

Our Contribution

In this paper, we first consider the constrained obnoxious facility location problem on a line segment, both decision and optimization versions. We propose a linear time solution for the decision version and a fully polynomial time approximation scheme for the optimization version of the problem, that runs in $O((n+k)\log(\frac{||pq||}{2(k-1)\epsilon}))$ time if the given demand points are ordered from left to right. Later, we consider the weighted version of a variant of the problem, namely, the minsum obnoxious facility location problem. For $k = 1$ we show that this problem can be solved in $O(n \log n)$ time, and for any $k > 0$ we give a dynamic programming algorithm that runs in $O(n^3 k)$ time.

2 Problem Statement

We now define formally the two variants of the obnoxious facility location problem restricted to a line segment $\overline{pq}$. Instead of maximizing the cumulative distance from demand points to obnoxious facilities, we compute the locations for obnoxious facilities near the highway so that none of the demand points or points with a minimal sum of their weights lie within a specified distance from the facilities.

Problem 1 *The constrained obnoxious facility location (COFL) problem is defined as follows: Given a set $P = \{p_1, p_2, p_3, \ldots, p_n\}$ of n demand points in the plane, a line segment $\overline{pq}$ and a positive integer k, pack k maximum-radius (non-overlapping) congruent disks $d_1, d_2, d_3, \ldots, d_k$ centered on $\overline{pq}$ such that no point of P lies inside any of these disks, where $p = (x(p), y(p))$, $q = (x(q), y(q))$, and $y(p) = y(q)$.*

Figure 1 shows the optimal solution for the problem instance: given a set P of n points, line segment $\overline{pq}$, and $k = 2$.

Problem 2 *The minsum obnoxious facility location (MOFL) problem is defined as follows: Given a horizontal line segment $\overline{pq}$ and a set P of n weighted points $\{p_1, p_2, \ldots, p_n\}$ whose weights are given by $w_1, w_2, \ldots, w_n$ respectively, in the plane, a positive integer k and a distance $\lambda > 0$, pack k (non-overlapping) disks $d_1, d_2, d_3, \ldots, d_k$ of radius λ centered on $\overline{pq}$ such*

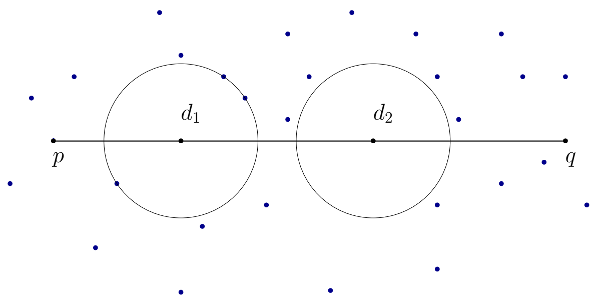


Figure 1: Line segment $\overline{pq}$, n points and $k = 2$

that $\sum_{j=1}^{k} \sum_{\{i|p_i \in d_j\}} w_i$ is minimized, i.e., the sum of weights of the points covered by these non-overlapping disks is minimized.

3 Constrained Obnoxious Facility Location (COFL) Problem

First, we consider the following obvious observation, then discuss how to solve the decision version of the *COFL* problem.

Observation 1 *If k is the number of disks that need to be packed on the segment $\overline{pq}$ and r_{max} is the radius of the optimal packing, then $r_{max} \leq (\frac{||pq||}{2(k-1)})$.*

3.1 Decision version of Constrained Obnoxious Facility Location problem (DCOFL)

Given a set P of n points in the plane, a line segment $\overline{pq}$, an integer k and a real number L, the decision version of the obnoxious facility location problem (*DCOFL*) is to answer the question: can we pack k disks of radius L, centered on $\overline{pq}$ such that none of the n points of P lie inside any of these k disks?

Consider a set P of n points and $\overline{pq}$ as shown below in Figure 2. Then for a given k, observe that if $L \leq r_{max}$, there is always a positive answer (i.e., yes-answer) to the above question. Next, we give an algorithm that answers this decision question and returns a corresponding packing of k disks centered on $\overline{pq}$ if the answer is *yes*.

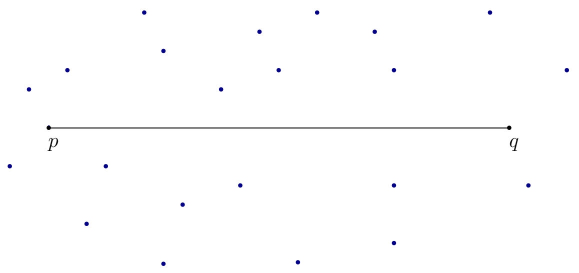


Figure 2: Line segment $\overline{pq}$ and n points

The solution to the *DCOFL* problem is as follows. First, consider a region $\mathcal{R}$ whose boundary is at distance L from the line segment $\overline{pq}$ (i.e., the Minkowski sum of $\overline{pq}$ and a disk of radius L) (see Figure 3).

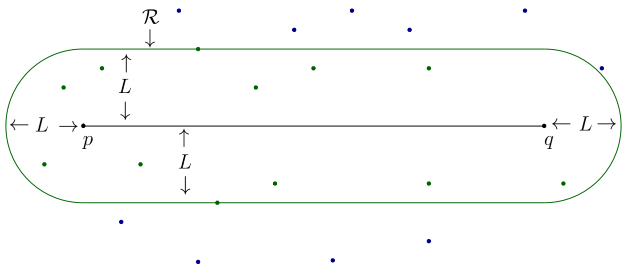


Figure 3: Region $\mathcal{R}$ of distance L from $\overline{pq}$

Observation 2 *None of the points of P lying outside of the region $\mathcal{R}$ will influence the choice of locating centers for disks in the optimal solution to DCOFL.*

Now, from every point inside or on the boundary of the region $\mathcal{R}$, we can locate *center-points* on $\overline{pq}$ which are at distance L from this point, where a *center-point* is a candidate center point for the disks in a packing.

Lemma 1 *Each point of P lying in the region $\mathcal{R}$ will have at least one center-point and at most two center-points on $\overline{pq}$ at distance L.*

Proof. Each of the points lying on the boundary of $\mathcal{R}$ has exactly one center-point on $\overline{pq}$ at distance L, whereas the points lying strictly in the interior of $\mathcal{R}$ will have at most two center-points on $\overline{pq}$ at distance L, (note that the coordinates of these center-points can be computed in $O(1)$ time using formulas from elementary geometry). Hence, there will be $O(1)$ center-points on the line segment $\overline{pq}$ for every point in $P \cap \mathcal{R}$ (see Figure 4). □

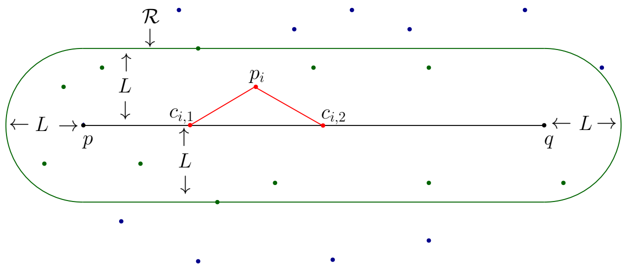


Figure 4: Point p_i in the region $\mathcal{R}$ has two points $c_{i,1}$ and $c_{i,2}$ on $\overline{pq}$ at distance L

Observation 3 *Let $p_i \in P$ be a point inside the region $\mathcal{R}$, and $c_{i,1}$ and $c_{i,2}$ be the center-points corresponding to p_i, then none of the k disks in an optimal solution to DCOFL will have their center points lying on the open interval $(c_{i,1}, c_{i,2})$ of the segment $\overline{pq}$.*

Now, consider another point $p_j \in P$ $(i \neq j)$ inside the region $\mathcal{R}$, which has two center-points $c_{j,1}$ and $c_{j,2}$ on the segment $\overline{pq}$ (see Figure 5). In figure 5, we can also observe that the intervals $[c_{i,1}, c_{i,2}]$ and $[c_{j,1}, c_{j,2}]$ formed by $\overline{c_{i,1}c_{i,2}}$ and $\overline{c_{j,1}c_{j,2}}$ are overlapping. Hence, from observation 3 none of the k disks in the optimal solution will have their centers lying on the interval $([c_{i,1}, c_{i,2}] \cup [c_{j,1}, c_{j,2}]) \setminus \{c_{i,1}, c_{j,2}\}$, excluding the end points of the union of the two intervals.

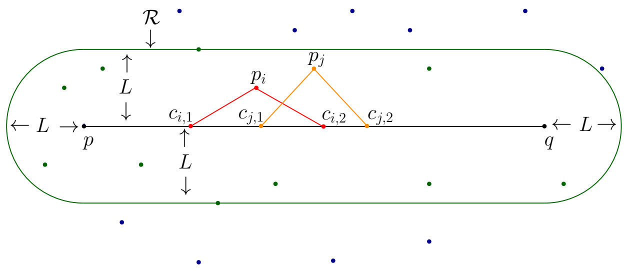


Figure 5: Another point p_j in the region $\mathcal{R}$ has two points $c_{j,1}$ and $c_{j,2}$ on $\overline{pq}$ at distance L

Without loss of generality, let $\{p_1, p_2, \ldots, p_m\}$ be the points of P lying strictly inside $\mathcal{R}$, above the line $y = y(p)$, and ordered from left to right, where $m \leq n$. We have seen that for every point p_i inside the region $\mathcal{R}$ there will be two center-points on the line segment $\overline{pq}$ which are at distance L, i.e., there is an interval $[l_i, r_i]$ for every point p_i inside the region $\mathcal{R}$, where $l_i = c_{i,1}$ and $r_i = c_{i,2}$. Merge all the overlapping intervals and then update the end points of the new intervals on $\overline{pq}$. Let $I = \{[l_1, r_1], [l_2, r_2], \ldots, [l_{m'}, r_{m'}]\}$ be the set of resulting pairwise disjoint intervals ordered by their left-end points in left-right order, where $m' \leq m$.

Consider the complement of I with respect to $\overline{pq}$, denoted as $I^c = \{[p, l_1], [r_1, l_2], \ldots, [r_{m'}, q]\}$, (here we assumed that none of the intervals in I contains the end-points of $\overline{pq}$). We can now greedily pack disks centered from point p to q on the segment $\overline{pq}$ by considering the complemented intervals from I^c. For the convenience assume $r_0 = p$, and $l_{m'+1} = q$. The k disks of radius L are packed on $\overline{pq}$ by Algorithm 1.

3.2 Correctness Proof

Theorem 2 *Given the set I^c of complemented intervals ordered from left to right, and an integer $k > 0$, Algorithm 1 (Greedy_LPacking) solves the DCOFL problem in linear time.*

Proof. The proof is as follows:

We can prove the correctness of Algorithm 1 by induction. If I^c is non-empty and $k = 1$, then we can place

Algorithm 1: Greedy_LPacking(I^c, k)

```
Let $j = 0$.
for $i = 1$ to $m' + 1$ do
    $index = \lfloor \frac{||r_{i-1}l_i||}{2L} \rfloor$, where $[r_{i-1}, l_i] \in I^c$;
    if $(j + index + 1) \leq k$ then
        Pack $index + 1$ disks of radius $L$ on the segment $\overline{r_{i-1}l_i}$.
        $j \leftarrow (j + index + 1)$
    end
    else
        Pack $(k - j)$ disks of radius $L$ on the segment $\overline{r_{i-1}l_i}$
        $j \leftarrow k$
    end
end
if $j = k$ then
    Let $D = \{d_1, d_2, \ldots, d_k\}$ be the set of disks computed in the above for-loop.
    return (yes, D)
end
else
    return (no, $\emptyset$)
end
```

a disk d_1 of radius L, centered on any interval from I^c. The disk d_1 does not contain any point of P in its interior because of the way the set I^c was constructed.

Now, consider an integer k' such that $1 < k' < k$. By induction hypothesis, Algorithm 1 packs k' disks $d_1, d_2, \ldots, d_{k'}$ of radius L as tightly as possible starting with centering the first disk d_1 at the left-most left-end point of an interval in I^c. If $L \leq r_{max}$, it is always possible to place the disk $d_{k'+1}$ of radius L, centered at the right-most right-end point of an interval in I^c (as $k' < k$). Hence, the algorithm is correct.

The analysis of the running time of Algorithm 1 is given below:

- The region $\mathcal{R}$ can be chosen in constant time.
- The points inside the region $\mathcal{R}$ can be found in time linear in the number of points in P.
- Given the points in P ordered from left to right, their corresponding intervals on $\overline{pq}$ can be found in $O(n)$ time, and so are the sets I and I^c.
- Algorithm 1 packs the disks on the complemented intervals of I^c in $O(k)$ time.

Hence, the overall time taken by our solution of *DCOFL* problem is $O(n + k)$. □

3.3 FPTAS for the COFL Problem

Now, we propose a fully polynomial time approximation scheme *(FPTAS)* for the *constrained obnoxious facility location* (*COFL*) problem, i.e., $(1 - \epsilon)$-approximation algorithm for any $1 > \epsilon > 0$. To solve the *COFL* problem, we consider many instances of *DCOFL* problem repeatedly.

In the *COFL* problem, our goal is to pack maximum-radius k (non-overlapping) congruent disks on the line segment $\overline{pq}$ such that none of the points of the set P lie inside of any of these disks. As we discussed above, the answer to the *DCOFL* problem is *yes* if we are able to pack k congruent disks of radius L on $\overline{pq}$ such that none of the points of P lie inside any of the disks, otherwise the answer is *no*. Hence, to find the maximum radius of k congruent disks, we solve the *DCOFL* problem repeatedly for $L = 2^i$, where $i = 0, 1, 2, \ldots,$ as long as *DCOFL* problem returns *yes*. From Observation 1 the optimal radius r_{max} is at most $\frac{||pq||}{2(k-1)}$. Let that small value of L be 2^j when the answer to the *DCOFL* problem is *no*. However, when $L = 2^{j-1}$ the answer to *DCOFL* problem is *yes*. Hence, if $2^j \leq \frac{||pq||}{2(k-1)}$ then r_{max} lies in the interval $[2^{j-1}, 2^j]$, else r_{max} lies in the interval $[2^{j-1}, \frac{||pq||}{2(k-1)}]$ for a given set P of n points, a line segment $\overline{pq}$ and an integer k. Given a real number ϵ, we bisect the interval $[2^{j-1}, 2^j] \log \frac{1}{\epsilon}$ time. Initially, $|2^j - 2^{j-1}| \leq r_{max}$, since we are able to pack k disks of radius 2^{j-1}. Let $[\alpha, \beta]$ be the interval after bisecting the interval $[2^{j-1}, 2^j]$ by $\log \frac{1}{\epsilon}$ times. Then $|\alpha - \beta| \leq \frac{|2^j - 2^{j-1}|}{2^{\log \frac{1}{\epsilon}}} \leq \frac{r_{max}}{2^{\log \frac{1}{\epsilon}}} \leq \epsilon r_{max}$. Hence, r_{max} lies in the interval $[\alpha, \beta]$, which implies that $\alpha \geq \beta - \epsilon r_{max} \geq (1 - \epsilon) r_{max}$. Therefore, the radius of the k congruent disks returned along with a positive answer by the last invocation of Algorithm 1 with $r = \alpha$ is at least $(1 - \epsilon) r_{max}$, where ϵ is an input parameter. Hence, we have the following theorem.

Theorem 3 *For a given line segment $\overline{pq}$ and an ordered set P of n points in the plane, we can get an $(1-\epsilon)$-factor approximation algorithm with $\epsilon > 0$ for the COFL problem, that runs in $O((n + k) \log(\frac{||pq||}{2(k-1)\epsilon}))$ time, by employing doubling search and bisection methods.*

Proof. From Theorem 2, we know that each call to solve the *DCOFL* problem will take $O(n+k)$ time. Doubling search guarantees that the optimal radius r_{max} of k congruent disks lies either in the interval $[2^{j-1}, 2^j]$ or $[2^{j-1}, \frac{||pq||}{2(k-1)}]$. We then divide this interval by $\log \frac{1}{\epsilon}$ times. In the worst case the number of invocations of the *DCOFL* problem is $O(\log \frac{||pq||}{2(k-1)} + \log \frac{1}{\epsilon})$ time, for an input parameter ϵ. Hence, the total running time is $O((n + k) \log(\frac{||pq||}{2(k-1)\epsilon}))$. □

4 Minsum Obnoxious Facility Location (MOFL) Problem

Recall that in the *MOFL* problem, we have an horizontal segment $\overline{pq}$ in the plane, without loss of generality we can assume that all the points in P are lying the above the horizontal line through $\overline{pq}$. As in the previous section, we again compute the intervals $[l_i, r_i]$ on $\overline{pq}$ for every point $p_i \in P$, but now defined by the *center-points* l_i and r_i on $\overline{pq}$, each at the distance λ from p_i. We call these intervals *mega-intervals* denoted by $I^{mega} = \{[l_1, r_1], [l_2, r_2], \ldots, [l_n, r_n]\}$. Observe that I^{mega} can be computed in $O(n)$ time as we are given the ordered set P of n points. Also, observe that a disk of radius λ centered anywhere on $\overline{pq}$ but only within $[l_i, r_i]$, covers the point p_i.

To solve the *MOFL* problem for $k = 1$ we use the approach of Katz et al., [4], but here the mega-intervals are defined differently as we are placing a disk instead of a rectangle in [4]. Then, similar to their approach, we define the elementary intervals on $\overline{pq}$, defined by every consecutive pair of end points of the intervals of I^{mega} starting with the left-end point p of the segment $\overline{pq}$. We call these elementary intervals *mini-intervals*, denoted by $I^{mini} = \{[\mu_1, \tau_1], [\mu_2, \tau_2], \ldots, [\mu_{2n+1}, \tau_{2n+1}]\}$. Observe that there can be at most $2n+1$ mini-intervals defined by n points in P and $\tau_j = \mu_{j+1}$ for $j = 1, 2, \ldots, 2n$. We define the weight of each mini-interval $[\mu_j, \tau_j]$ to be $\kappa_j = \sum_{\{i \mid [\mu_j, \tau_j] \subseteq [l_i, r_i] \in I^{mega}\}} w_i$, where w_i is the weight of the point $p_i \in P$, i.e., the sum of weights of all the points whose corresponding mega-intervals contain $[\mu_j, \tau_j]$ entirely within them (see Fig. 6 for an illustration of mega-interval, mini-interval, and their weights). Observe that an optimal disk d, i.e., the disk d of radius λ such that $\sum_{\{i \mid p_i \in d\}} w_i$ is minimized, can be centered anywhere in the mini-interval $[\mu_j, \tau_j]$ whose κ_j is minimal. As in [4], to efficiently find such a $[\mu_j, \tau_j]$ we construct a segment tree data structure on the mini-intervals $[\mu_j, \tau_j] \in I^{mini}$.

The construction of the segment tree and finding of minimum κ_j is briefly described as follows. We first construct a balanced binary tree T whose leaves correspond to the mini-intervals that are ordered from left to right. Since T is balanced and has n leaves, its depth must be $O(\log n)$. Further, we associate each node v of T with two attributes: (1) an interval which is the union of the mini-intervals of all the leaves of the subtree rooted at v, and (2) the weight which is equal to the sum of the minimum of the weights of its two child nodes and the weights of the mega-intervals that are stored in v. Initially, the weights of all the nodes of T including the leaves are zero. The intervals of the leaves are their mini-intervals. A mega-interval will be stored at a node v if the interval of v is completely contained in it, and if so, then not at any of its descendant nodes. We now insert all the mega-intervals of I into T one by one, and during each insertion, we update the weight attributes of the nodes. The key feature of the segment tree is that each insertion of a mega-interval and the corresponding updates in T takes $O(\log n)$ time. At each insertion, the weight of the root of T is the smallest κ_j. The corresponding mini-interval $[\mu_j, \tau_j]$ can be found by traversing down the tree in the direction of the child with smaller attribute weight, where ties can be broken arbitrarily. This traversal clearly takes $O(\log n)$ time. The mini-interval (not necessarily unique) corresponding to the weight of the root of the segment tree is the location for placing the disk d of radius λ such that $\sum_{\{i \mid p_i \in d\}} w_i$ is minimized. The construction of the segment tree takes $O(n \log n)$ time. Hence, we have the following theorem.

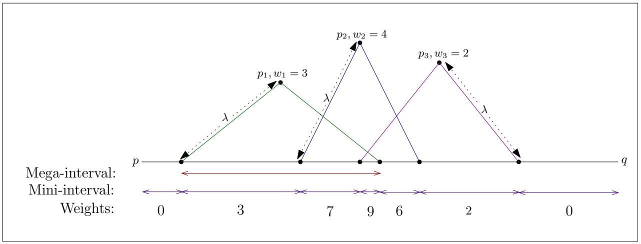


Figure 6: An illustration of splitting of mega-intervals into mini-intervals and their weight calculations

Theorem 4 *The MOFL problem for $k = 1$ can be solved in $O(n \log n)$ time.*

Remark 1 *A lower bound of $\Omega(n \log n)$ for the MOFL problem with $k = 1$ can be obtained using the same reduction as Katz et al., [4] did it for their version of the obnoxious facility location problem.*

4.1 Dynamic programming solution for any $k > 0$

Here, we first discuss a dynamic programming recurrence for computing the minimum weight of the points covered by k non-overlapping disks of radius λ centered on $\overline{pq}$. Then, we discuss how to actually reconstruct the solution, i.e., the mini-intervals on which the k disks of radius λ can be packed such that the sum of weights of the points covered by the union of these disks is minimal.

We now define the subproblem as follows:
Let $C(i, j, h)$ denote the minimum weight of the points covered by j non-overlapping disks of radius λ centered on the sub-segment $\overline{p\tau_i} = \bigcup_{s=1}^{i} [\mu_s, \tau_s]$, where the $j+1$th disk was centered on the mini-interval $[\mu_t, \tau_t]$ such

that $||\mu_i\tau_{t-1}|| = h$. Clearly, $i \in \{1, 2, \ldots, 2n+1\}$, $j \in \{1, 2, \ldots, k\}$, and $h \in \{||\mu_\alpha\tau_\beta|| \mid \alpha < \beta\}$.

To define the recurrence which relates between the subproblems (WLOG), we can assume that all the disks in an optimal solution are centered close to the left-end points of their respective mini-intervals ($\kappa_l \neq 0$ for every mini-interval $[\mu_l, \tau_l]$) and that $C(i, j, h) = \infty$ denotes the impossibility. Now, for computing $C(i, j, h)$, observe that the minimum-weighted packing of $j (\leq k)$ disks of radius λ either centers a disk on the mini-interval $\overline{\mu_i\tau_i}$ or not, but never centers a disk on $\overline{\mu_i\tau_i}$ if $h < 2\lambda$. Therefore, we have the recurrence below.

$$C(i, j, h) = \\ min \begin{cases} \kappa_i + C(i-1, j-1, ||\mu_{i-1}\tau_{i-1}||) & \text{if } h \geq 2\lambda \\ C(i-1, j, h + ||\mu_{i-1}\tau_{i-1}||) & \text{always} \end{cases}$$

The base cases are the following:

- If $i = 1$, $j = 1$, and $h \geq 2\lambda$, then $C(i, j, h) = \kappa_i$.
- If $i = 1$, $j = 1$, and $h < 2\lambda$, then $C(i, j, h) = \infty$.
- If $i > 1$, $C(i, 1, h)$ can be computed in $O(n \log n)$ time using Theorem 4.

The minimum weight of the points covered by the optimal packing can be obtained from $C(2n+1, k, 2\lambda+1)$. The number of possible values for h is $\binom{2n+1}{2}$. We can reconstruct the minimum-weighted packing (and their corresponding mini-intervals) by storing parent pointers between the entries in the three-dimensional array $C[2n+1, k, \binom{2n+1}{2}]$. From the above recurrence, it is clear that a subproblem depends on subproblems with strictly smaller i, and hence there is no cyclic dependency between the subproblems. We can fill the three-dimensional array by start filling its lower indexed entries first. Since computing the value of an entry requires the values of two lower indexed entries of the array, the work per subproblem is $O(1)$. Therefore, we have the following theorem.

Theorem 5 *The MOFL problem for any $k \geq 1$ can be solved in $O(n^3k)$ time using dynamic programming approach.*

5 Conclusion

In this paper, we have considered two *obnoxious facility location* problems restricted to a line segment, namely, the *COFL* and *MOFL* problem. We gave a linear time solution for the decision version of the *COFL* problem, and using this linear time solution as a subroutine, we have proposed a *FPTAS* for the *COFL* problem. In future, we want to investigate whether the *COFL* problem can be solved optimally in polynomial time or is `NP-hard`. We have solved the *MOFL* problem exactly in polynomial time using the dynamic programming approach.

References

[1] A. Tamir. Obnoxious facility location on graphs. *Transportation Science*, 4(4):550–567, 1991.

[2] G. Frederickson and D. Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal of Computing*, 13:14–30, 1984.

[3] J. M. Colmenar, P. Greistorfer, R. Mart and A. Duarte. Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem *European Journal of Operational Research*, 252(2):432–442, 2016.

[4] M. J. Katz, K. Kedem, and M. Segal. Improved algorithms for placing undesirable facilities. *Computers & Operations Research*, 29:1859-1872, 2002.

[5] O. Gokalp. An iterated greedy algorithm for the obnoxious p-median problem. *Engineering Applications of Artificial Intelligence*, 92:103674, 2020.

[6] R. L. Church and R. S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12(2):107–118, 1978.

[7] S. Michael. Placing an obnoxious facility in geometric networks. *Nord. J. Comput*, 10(3):224–237, 2003.

[8] Z. Drezner and G. O. Wesolowsky. Finding the circle or rectangle containing the minimum weight of points. *Location Science*, 2(2):83-90, 1994.

[9] Z. Drezner and G. O. Wesolowsky. Obnoxious facility location in the interior of a planar network. *Journal of Regional Science*, 35:675–688, 1995.

Succinct Euler-Tour Trees

Travis Gagie[*] Sebastian Wild[†]

Abstract

We show how a collection of Euler-tour trees for a forest on n vertices can be stored in $2n + o(n)$ bits such that simple queries take constant time, more complex queries take logarithmic time and updates take polylogarithmic amortized time.

1 Introduction

Tarjan and Vishkin [7] showed how we can efficiently solve several problems in algorithmic graph theory by representing planar embeddings of trees as Euler tours of corresponding directed graphs. To obtain the graph corresponding to a tree, we replace each undirected edge (u, v) in the tree by the directed edges (u, v) and (v, u), so the resulting Eulerian tour of the graph is like a depth-first traversal of the tree without the need for of a distinguished root. Henzinger and King [2] then showed how to implement such Euler tours for trees as dynamic balanced binary search trees, such that we can quickly support navigation queries and updates such as inserting an edge joining two trees and deleting an edge from a tree. These implementations are called Euler-tour trees (ETTs).

As far as we know, all current ETTs of a tree on n vertices take $\Omega(n)$ words of space. In contrast, we can store a *rooted* planar embedding of a tree (that is, an ordinal tree) on n vertices in only $2n + o(n)$ bits of space while still quickly supporting navigation queries and some updates, and such representations of ordinal trees are central to the field of compact data structures; see [4] and references therein. For example, Ferres et al. [1] showed how to represent an embedding of a connected planar graph G with m edges in $4m + o(m)$ bits by rooting and storing compactly a spanning tree of G and an interdigitating spanning tree of the dual of G. (The observation that any connected planar graph has such a pair of spanning trees appeared in Von Staudt's book *Geometrie de Lage* [6] in 1847.) Figure 1 shows the example from Ferres et al.'s paper, with the primal spanning tree shown in red and the dual spanning tree shown in blue.

Ferres et al.'s data structure can be made to support quickly insertions of vertices and edges, but it does not adequately support deletions. To see why, consider a graph on n vertices with three arms of equal length, as shown on the left in Figure 2 (with the dotted edge (c, e) not present initially). The primal spanning tree must be the whole graph and, wherever we root it, two whole arms are branches. Without loss of generality, suppose we root the spanning tree at r, so the paths from b to c and from d to e are branches in the spanning tree, as shown on the right. If we delete the edge (a, b) from the graph and insert the dotted edge (c, e), then we have no choice but to reverse the directions in the spanning tree of all the edges either in the path from b to c or in the path from r to e. With the balanced-parentheses representation Ferres et al. used for their spanning trees, this takes $\Omega(n)$ time. If we then delete (c, e) and reinsert (a, b), undoing the updates again takes $\Omega(n)$ time.

If the spanning tree in Figure 2 were represented by an ETT rather than with balanced paretheses, on the other hand, then deleting (a, b) and inserting (c, e) would be easy. Indeed, we conjecture that it is possible to implement Ferres et al.'s data structure with ETTs such that all queries and updates take polylogarithmic time – but doing so would not currently be interesting because the entire data structure would no longer be compact. Therefore, in this paper we start to investigate whether ETTs can be made compact while still quickly supporting a reasonable selection of queries and updates.

In Section 2 we describe a representation of ETTs that we call *macro-trees*, which slightly extend previous representations to allow weighted corners, where a *corner* is the "gap between consecutive edges incident to some vertex" [3]. For a forest G on n vertices, this representation takes $O(n)$ words of space and supports simple queries in constant time and more complex queries and updates in logarithmic time.

In Section 3 we describe how if G consists of a single tree with maximum degree $d \geq 2$ and n is sufficiently large then, given a positive constant ε, we can cluster the vertices such that each cluster contains between $\lg^{1+\varepsilon}(n)$ and $d \lg^{1+\varepsilon}(n) + 1$ vertices. We represent each cluster with a succinct *micro-tree*, such that all the clusters take a total of $2n + o(n)$ bits, and then represent the $O(n/\log^{1+\varepsilon} n)$ inter-cluster edges with an $o(n)$-bit macro-tree. The weight of each corner between two inter-cluster edges is the number of steps (all taken within the cluster) between those edges in the Euler tour of G.

[*]Faculty of Computer Science, Dalhousie University, travis.gagie@dal.ca

[†]Department of Computer Science, University of Liverpool, sebastian.wild@liv.ac.uk

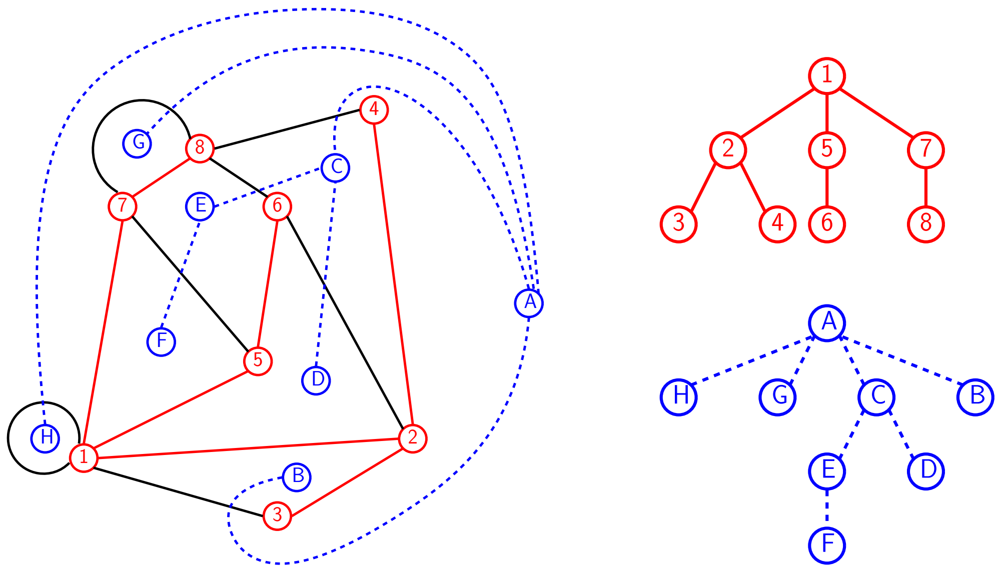


Figure 1: Ferres et al.'s example of a planar graph **(left)**, with the primal spanning tree **(top right)** shown in red and the dual spanning tree **(bottom right)** shown in blue.

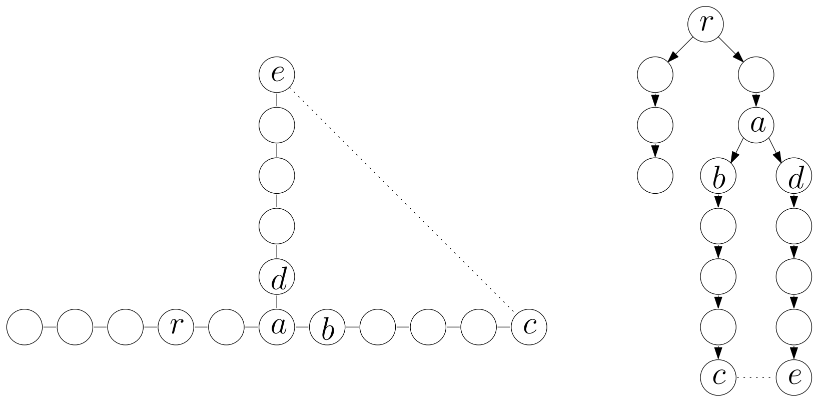


Figure 2: A hard case for Ferres et al.'s data structure: however we choose to root the spanning tree of the graph **(left)**, deleting one edge and inserting another – for example, if the root r is in the left arm, then deleting (a, b) and inserting (c, e) – forces us to reverse at least about $n/3$ edges in the spanning tree **(right)**.

If G consists of multiple trees, we cluster their vertices independently – with the bounds on the cluster sizes still depending on the maximum degree d of the whole graph and the number n of vertices it contains. This gives us our first result: we can store a collection of Euler-tour trees for a forest on n vertices with maximum degree d in $2n + o(n)$ bits and support simple queries in constant time, more complex queries in $O(\log n)$ time and updates in $O(d \log^{1+\varepsilon} n)$ amortized time. Since submitting this paper, we have realized how to remove the dependence on d from the update time, as we describe in Section 4.

Unless specified otherwise, all trees in this paper are taken to be planar embeddings of an unrooted tree (a. k. a. unrooted plane trees). We assume throughout that n is the total number of vertices in the maintained forest and we are working in the word-RAM model with $\Theta(\log n)$-bit words.

2 Macro-Trees

Suppose we are given a planar embedding of a forest G with weighted corners, where each weight fits in a constant number of machine words; Figure 3 shows an example at the top left, consisting of a single tree. For each tree T in G, we store T's edges in a circular, doubly-linked list, in the order they are crossed in the Euler tour of T. We also store a bidirectional pointer between each directed edge (u, v) in T and its reverse, (v, u).

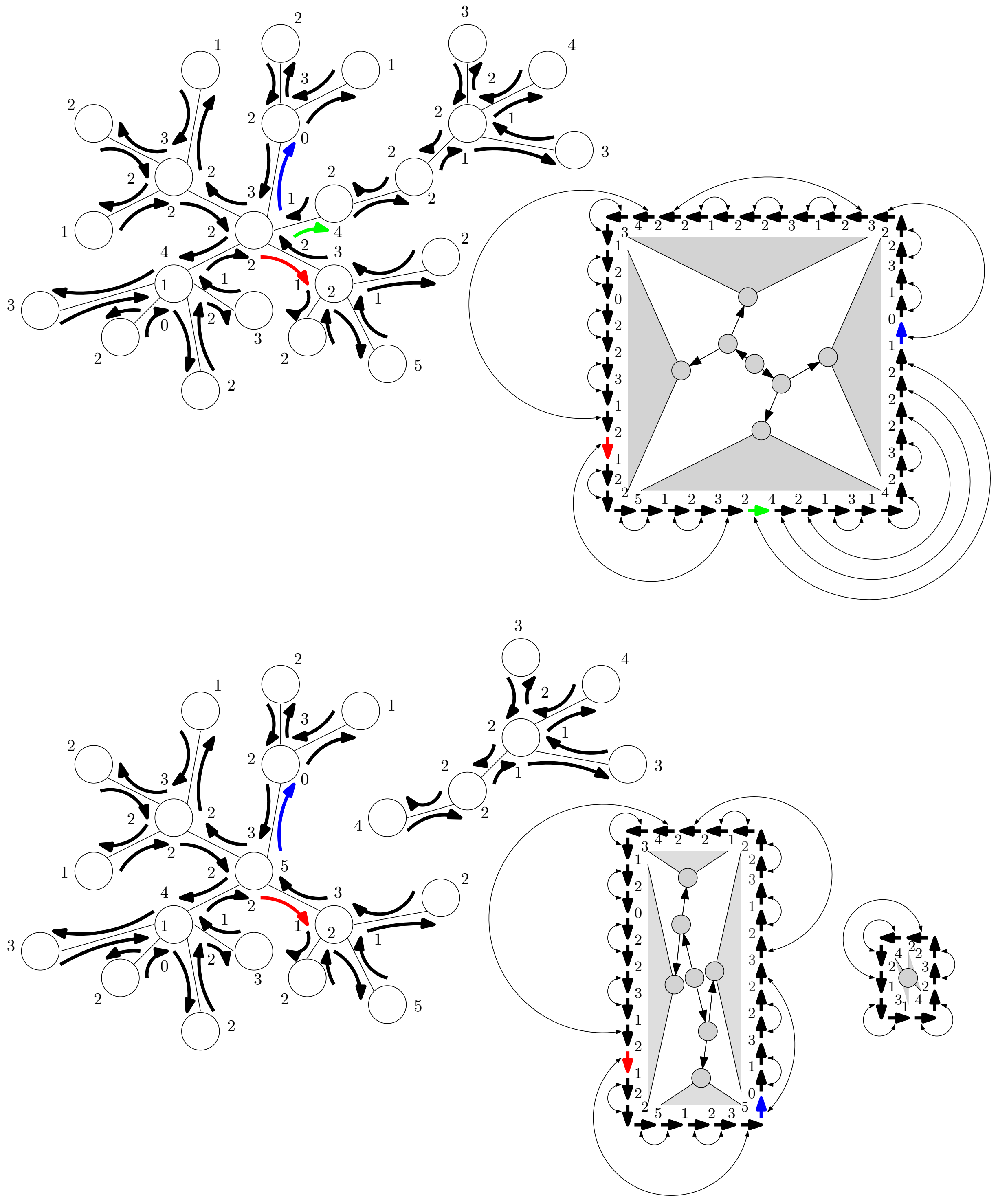


Figure 3: An example of a macro-tree **(top left)** and its representation **(top right)**, and then the forest of two macro-trees obtained by deleting an edge **(bottom left)**, and their representations **(bottom right)**.

This allows us to move forward and backward in the Euler tour for T one edge at a time, and to enumerate the edges incident to a given vertex u in constant time per edge. To see why, notice that if (u, v) precedes (u, w) in the counter-clockwise order of the edges incident to u, then (v, u) precedes (u, w) in the Euler tour.

Finally, we store an AVL tree whose leaves are the nodes in the list for T, and augment it such that given a directed edge e and an integer t, in logarithmic time we can return the edge e' such that the distance from e to e' in the list is t, or the total weight of the corners from e and e' is as close to t as possible without being greater (or optionally less), or the sum of the distance and the total weight of the corners from e to e' is as close to t as possible without being greater (or optionally less).

With the same AVL tree, in logarithmic time we can find the distance, or the total weight of the corners, in the list between two given edges of T. This means, e. g., that we can quickly determine the size and total weight of the subtrees on either side of a given edge.

Figure 3 includes an illustration of our representation at the top right, with the circular, doubly-linked list of directed edges shown as a square of arrows (with the pointer from each edge to its predecessor and successor in the list omitted for the sake of legibility); bidirectional pointers between directed edges (u, v) and (v, u) shown as arcs outside the square; and the AVL tree shown in grey inside the square, with the topmost nodes shown as circles with arrows from parents to children and then lower subtrees shown as triangles.

To advance from the red edge to the green edge, we follow the pointer to the reverse of the red edge, then move one position forward in the list; to advance from the green edge to the blue edge, we do the same thing. To find the size of the subtree we traverse between the red edge and its reverse, we use the AVL tree to count the 6 directed edges between them in the list in logarithmic time, divide by 2 to get the number 3 of undirected edges in the tree, and add 1 to get the number 4 of vertices. Similarly, we can sum the weights of the corners between the red edge and its reverse in the list.

We can change the weight of a corner in logarithmic time or, by splitting and joining doubly-linked lists and AVL trees, delete an edge in a tree represented by a macro-tree or insert an edge between vertices in two trees represented by macro-trees. In our example, if we delete the undirected edge corresponding to the green edge and its reverse and assign the new corners weights 5 and 4, then we obtain two macro-trees shown at the bottom left of Figure 3, whose representations are shown at the bottom right. Notice that now the blue edge follows the reverse of the red edge in the list.

Lemma 1 *Given a planar embedding of a forest G on n vertices with weighted corners, we can store macro-trees for the trees in G in a total of $O(n)$ words of space such that operations (i) and (ii) take constant time and operations (iii) – (xii) take $O(\log n)$ time:*

(i) *given a directed edge e, return its predecessor and successor in the Euler tour of the tree containing e;*

(ii) *given a directed edge (u, v), return its predecessor and successor in the counter-clockwise enumeration of edges incident to u;*

(iii) *given a directed edge e and an integer t, return the directed edge e' such that the distance from e to e' is t in the Euler tour of the tree containing e;*

(iv) *given a directed edge e and an integer t, return the edge e' such that the total weight of the corners from e to e' in the Euler tour of the tree containing e is as close to t as possible without being greater (or optionally less);*

(v) *given a directed edge e and an integer t, return the edge e' such that the sum of the distance and the total weight of the corners from e to e' in the Euler tour of the tree containing e is as close to t as possible without being greater (or optionally less);*

(vi) *given two directed edges e and e' in the same tree, return the distance and the total weight of the corners from e to e' in the Euler tour of that tree;*

(vii) *given an edge e, return the number of vertices and total weight of the corners in the subtrees on either side of e;*

(viii) *given a directed edge e and a weight w, set to w the weight of the corner after e in the Euler tour of the tree containing e;*

(ix) *given an edge e and weights w and w', delete e from the tree containing it and set the new corners' weights to w and w';*

(x) *given corners in two different trees T and T' and four weights, insert an edge between T and T' bisecting those corners and assign the four new corners the given weights;*

(xi) *given an edge e, contract e, thus fusing its endpoints and removing e and its reverse edge from the Euler tour, adding up fused corner weights;*

(xii) *given two corners of the same vertex v, split v into two nodes v_1 and v_2, connected by a new edge e, so that the neighborhoods of v_1 and v_2 result from the neighborhood of v by splitting it at the given corners and inserting the new edge e there.*

3 Micro-Trees

For now, suppose G consists of a single tree with maximum degree $d \geq 2$ and n is sufficiently large. Given a planar embedding of G and a positive constant ε, we can use essentially a centroid decomposition to partition G recursively into clusters each containing between $\lg^{1+\varepsilon}(n)$ and $B = d\lg^{1+\varepsilon}(n) + 1$ vertices.

Suppose at some step of the recursion we are considering a subtree S of G on n_S vertices such that $n_S > \lg^{1+\varepsilon}(n)$. If $n_S \leq B = d\lg^{1+\varepsilon}(n) + 1$, then we can stop recursing. Otherwise, we find a vertex or edge whose removal from S leaves a forest in which each tree has size at most $n_S/2$.

If we find such an edge e, then the two trees in the forest left by e's removal from S each have size

$$\frac{n_S}{2} \geq \frac{B}{2} > \lg^{1+\varepsilon}(n)$$

and so are large enough (and maybe too large) to be clusters; we recurse on them. If instead we find such a vertex v then, since v has degree at most d, at least one of the trees S' in the forest left by v's removal from S has size

$$\frac{n_S - 1}{d} > \lg^{1+\varepsilon}(n)$$

and so is large enough (and maybe too large) to be a cluster. Since S' contains at most $n_S/2$ vertices, the rest of S (including v) is a tree at least as big as S', so it too is large enough (and maybe too large) to be a cluster. We recurse on S' and the rest of S.

Once we have partitioned G into clusters, we consider that partition as a tree P on $O(n/\log^{1+\varepsilon} n)$ vertices, with clusters in G as vertices in P and edges between clusters in G as edges in P. We store a macro-tree for P, which takes $O(n/\log^{1+\varepsilon} n)$ words or $O(n/\log^{\varepsilon} n) \subset o(n)$ bits, with the weight at a corner between e and e' in P the number of steps between e and e' in the Euler tour of G. We note that P can have maximum degree more than d, but this does not affect Lemma 1.

Figure 4 shows an example of clusters embedded in vertices of the macro-tree from Figure 3, before and after an edge is deleted, with triangles representing subtrees. The expanded view of one of the clusters shows why the corners incident to that cluster have weights 3, 2, 2 and 5: if we enter the cluster along the reverse of the blue edge, then we take 3 steps of the Euler tour inside the cluster before leaving the cluster again; when we re-enter for the first time, we take 2 steps inside before leaving again; when we re-enter for the second time, we also take 2 steps inside before leaving again; and finally, when we re-enter for the last time, we take 5 steps inside before leaving again, along the blue edge. We note that the steps in the Euler tour to enter and leave the cluster do not count toward the weights of the corners.

We represent each cluster C with a *micro-tree*: we temporarily ignore inter-cluster edges, root the remaining tree arbitrarily, and represent it succinctly as an ordinal tree using $2n_C + o(n_C)$ bits, where n_C is the number of vertices in C. This takes a total of $2n + o(n)$ bits over all the clusters. Therefore, including the $o(n)$ bits for the macro-tree for P, thus far we are still representing G using $2n + o(n)$ bits. However, we need to provide an interface between the inter-cluster edges and the micro-trees.

For a micro-tree on n_C vertices, we store a "ports bitvector" with $2n_C$ copies of 0 corresponding to the steps in a depth-first traversal of the micro-tree, with copies of 1 marking where inter-cluster edges are incident to vertices in the cluster (the cluster's "ports" to other clusters). The total length of these bitvectors is $2n + O(n/\log^{1+\varepsilon} n)$ but only $O(n/\log^{1+\varepsilon} n)$ of the bits are 1s, so compressed representations takes a total of $o(n)$ bits including support for rank and select [5]. We store a mapping from the inter-cluster edges ending in a cluster to the ranks of the 1s in that cluster's ports bitvector, marking when in the depth-first traversal those inter-cluster edges touch vertices. We also store a mapping back from these 1s to the inter-cluster edges in the macro-tree. This takes $O(\log n \cdot n/\log^{1+\varepsilon} n) \subset o(n)$ bits, so we are still using $2n + o(n)$ bits overall.

For example, if we root the cluster shown in the expanded view in Figure 4 at the vertex reached by the green directed edge, on the right by the 5, then the bitvector is `0010001001001000`. This means that if we start at our chosen root and walk counter-clockwise around the cluster, we take 2 steps before passing the first inter-cluster edge (pointing up and to the right), then 3 steps before passing our second (pointing up and left), then 2 before passing our third (pointing down and left), then 2 before passing our fourth (pointing down and right), and finally three more before reaching the root again. Of course, if we view the bitvector as cyclic – corresponding to an Euler tour of the cluster rather than a depth-first traversal of the micro-tree – then the lengths of the runs of 0s are the weights of the corners around that cluster in P.

If G consists of multiple trees, we cluster them independently – with the bounds on the cluster sizes still depending on the maximum degree d of the whole graph and the number n of vertices in G. If G contains trees with less than $\lg(n)$ nodes, we store these all as a single dynamic string of balanced parentheses. To be able to update the representations of the individual small trees, we keep the string divided into blocks of size roughly $\lg^{1+\varepsilon}(n)$ and completely replace any block we want to edit – much like the clusters. The representation still takes $2n + o(n)$ bits overall; since queries on those tiny trees trivially take $O(\log n)$ time, we obtain the same overall efficiency.

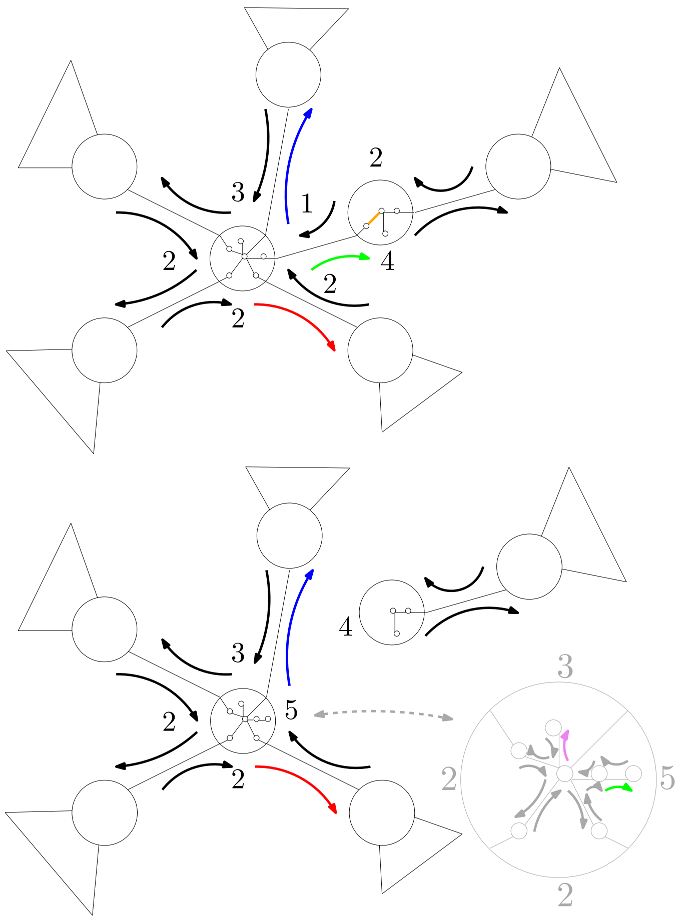


Figure 4: An example of clusters, before the orange edge is deleted **(above)** and after **(below)**. Deleting the orange undirected edge (adjacent to the green edge in the "before" example) means the green directed edge and its reverse retract into the other cluster (in the "after" example), as shown in the expanded view of that cluster **(bottom right)**.

3.1 Queries

Suppose we know that the ith `0` in a cluster u's bitvector indicates the step across a directed edge e in the Euler tour of u, and the jth `0` in a cluster v's bitvector indicates the step across a directed edge e' in the Euler tour of v, and we want to compute the number of steps between e and e' in the Euler tour of G. To do this, we first use rank and select queries on u's and v's bitvector to find the first `1` after the ith `0` in u's bitvector and the last `1` before the jth `0` in v's bitvector, the ranks of those `1`s and their distances from the ith and jth 0s. This tells us how many steps in the Euler tour we take after crossing e before we leave u for the first time on an inter-cluster edge, and how many steps we take before crossing e' after we enter v for the last time on an inter-cluster edge. We then map those `1`s to directed inter-cluster edges (u, w) and (x, v) and use Lemma 1 to compute the distance and total weight of the corners between them in the Euler tour of the tree containing those edges. The distance tells us the number of steps we take across inter-cluster edges between crossing e and e' in the Euler tour, and the total weight of the corners tells us the number of steps we take across intra-cluster edges between crossing (u, w) and (x, v). Since we already know how many edges we cross between e and (u, w) and between (x, v) and e', we can compute the distance from e to e' in the Euler tour of the tree containing them. This all takes $O(\log n)$ time, dominated by the time to query the AVL tree in the macro-tree representation to find out the distance and total weight of the corners between (u, w) and (x, v).

For example, suppose e is the violet directed edge shown in the expanded view of the cluster in Figure 4, and e' is the green edge in the same cluster (so $u = v$ in this case). We cross 2 intra-cluster edges after e before we leave the cluster across the inter-cluster edge pointing up and left, and cross 2 intra-cluster edges after re-entering the cluster across the inter-cluster edge arriving from down and right (the reverse of the red directed edge) before reaching e'. The AVL tree at the bottom of Figure 3 tells us there are 26 steps in the Euler tour of P between when we leave the cluster heading up and left and when we re-enter it from down and right (including crossing those two inter-cluster edges), and the total weight of the corners between those directed inter-cluster edges is

$$\begin{aligned} & 2+1+3+2+2+1+2+2+4+ \\ & 3+1+2+0+2+2+3+1+2+ \\ & 1+2+2+5+1+2+3 \\ = \; & 51\,, \end{aligned}$$

so the number of steps between e and e' in the Euler tour is

$$2+2+26+51=81\,,$$

not including crossing e and e' themselves.

Similarly, if we know the `0` in the bitvector of a cluster indicating a step across a directed edge e and we are given an integer t, we can find the cluster containing the directed edge e' that is t steps after e in the Euler tour of the tree containing e, and find the `0` in that cluster's bitvector indicting the step across e', all in $O(\log n)$ time. Using queries in the micro-trees and macro-tree we can also move forward or backward one step in any Euler tour in constant time, and enumerate the edges incident to a given vertex in constant time per edge. Of course we cannot store identifiers of all the vertices in only $2n + o(n)$ bits, so in general we need intra-cluster directed edges to be specified by which clusters they are in and the ranks of the `0`s indicating them, and vertices to be specified by specifying a directed edge leaving them. We can afford to store mobile fingers to $O(n/\log^{1+\varepsilon} n)$ edges and vertices, however, without affecting our space bound.

Lemma 2 *Given a planar embedding of a forest G on n vertices, we can partition G into clusters, store the partition as macro-trees and store each cluster as a micro-tree in a total of $2n + o(n)$ bits, such that operations (i) and (ii) take constant time and (iii) – (v) take $O(\log n)$ time:*

(i) *given a directed edge e, return its predecessor and successor in the Euler tour of the tree containing e;*

(ii) *given a directed edge (u, v), return its predecessor and successor in the counter-clockwise enumeration of edges incident to u;*

(iii) *given a directed edge e and an integer t, return the directed edge e' such that the distance from e to e' is t in the Euler tour of the tree containing e;*

(iv) *given two directed edges e and e', return the distance from e to e' in the Euler tour of the tree containing them;*

(v) *given an edge e, return the number of vertices in the subtrees on either side of e.*

Notice that we do not mention G's maximum degree d in Lemma 2. This is because d appears only in the upper bound B on clusters' sizes, which does not affect query times but only updates, which we discuss next.

3.2 Updates

Recall that $B = d\lg^{1+\varepsilon}(n)+1$ here. To insert an edge between two trees or delete an edge in a tree, we completely rebuild the micro-trees, bitvectors and mappings for the affected clusters, in $O(B)$ time – possibly choosing new roots for the new micro-trees – and update the macro-tree or macro-trees in $O(\log n)$ time. We may need to split or join a constant number of clusters to maintain the invariant that they all have between $\lg^{1+\varepsilon}(n)$ and B vertices, but this can still be handled in $O(B)$ time using standard techniques. A more drastic problem occurs when so many vertices are added or deleted that the bounds for our cluster sizes change and we must rebuild many clusters, but this cost can be amortized over the insertions and deletions.

To delete the orange edge in Figure 4, we move the shared endpoint of the orange and green edges into the same cluster as the other endpoint of the green edge, then completely rebuild the micro-trees, bitvectors and mappings for the clusters shown inside the vertices of the macro-tree. We update the macro-tree by deleting the green edge and its reverse from the macro-tree (since they are now intra-cluster edges), and weighting the new corners by the number of steps in the Euler tours of the clusters between the preceding and succeeding edges: the cluster that contained the orange edge now has 3 vertices, 2 (undirected) intra-cluster edges and 1 (undirected) inter-cluster edge, so the number of steps in the Euler tour of T between entering it and leaving it is 4, so that is the weight of its new single corner; the cluster that now contains the green edge now has 7 vertices, 6 (undirected) intra-cluster undirected edges and 4 (undirected) inter-cluster edges, and the number of steps in the Euler tour between entering it across the last inter-cluster edge before the green edge and leaving it across the first inter-cluster edge after the green edge is 5, so that is the weight of its new corner.

Lemma 3 *After deleting a given edge from a tree in a forest on a total of n vertices with maximum degree d, we can update our representation of that tree in $O(B)$ amortized time and obtain the representations of the two resulting trees. Similarly, after inserting an edge bisecting given corners in two trees of a forest on a total of n vertices with maximum degree d, we can update our representations of those trees in $O(B)$ amortized time and obtain a representation of the single resulting tree. We can add or delete an isolated vertex in $O(B)$ amortized time.*

Combining Lemmas 2 and 3, we obtain our first theorem:

Theorem 4 *Given a planar embedding of a forest G on n vertices with maximum degree d, we can store G in $2n + o(n)$ bits such that operations (i) and (ii) take constant time, operations (iii) – (v) take $O(\log n)$ time and (vi) and (vii) take $O(d \log^{1+\varepsilon}(n))$ time:*

(i) *given a directed edge e, return its predecessor and successor in the Euler tour of the tree containing e;*

(ii) *given a directed edge (u, v), return its predecessor and successor in the counter-clockwise enumeration of edges incident to u;*

(iii) *given a directed edge e and an integer t, return the directed edge e' such that the distance from e to e' is t in the Euler tour of the tree containing e;*

(iv) *given two directed edges e and e' in the same tree, return the distance from e to e' in the Euler tour of that tree;*

(v) *given an edge e, return the number of vertices in the subtrees on either side of e;*

(vi) *given an edge e, delete e from the tree containing it and return the representations of the two resulting trees;*

(vii) *given corners in two different trees T and T', insert an edge between T and T' bisecting those corners and return the representation of the resulting tree.*

4 Trees of Arbitrary Degree

The above scheme for bounded-degree forests can be generalized to arbitrary forests by splitting high-degree vertices. In the following, we describe the necessary changes to the data structure.

4.1 Macro-Trees

We modify the macro-tree by allowing (inter-cluster) edges to be either “true” or “false”. A true edge is as before, whereas a false edge does not actually correspond to any edge in G, but rather connects two clones of the same graph vertex in different clusters. In the implementation, we can identify each edge and its preceding corner into a single entity with a weight, a “macro-edge”; a false edge adds weight 0 and a true edge adds weight 1 to the macro-edge. Conceptually, edges in the macro-tree are (potentially empty) sequences of (consecutive) true edges in the Euler tour, plus optionally one false edge at the end of such a sequence.

As before, macro-edges are kept in a linked list, with pointers to their reverse traversals. We also add pointers to the immediate true successors and predecessors of each macro-edge. Any balanced BST that supports splitting and merging, augmented with subtree weights, can be used to implement efficient access to macro-edges. Operations stay the same, except that false edges have to be counted with weight 0.

4.2 Micro-Trees

Our decomposition for unbounded degrees follows a similar approach as above, but caters for large degrees by splitting vertices. Given a total size n and a constant $\varepsilon > 0$, we now set $B = 3 \lg^{1+\varepsilon}(n)$. We decompose a tree T on n_T vertices as follows: If $n_T \leq B$, it forms a cluster of its own. Else, we find a centroid, i.e., a node or edge, so that after its removal, all remaining subtrees have size at most $\frac{1}{2} n_T$. If we find a centroid edge in T, we recurse as before. Otherwise we find a node v that splits T into subtrees $S_1, \ldots, S_d$, for $d = \deg(v)$, of sizes $n_{S_1}, \ldots, n_{S_d} \leq \frac{1}{2} n_T$. If any of these subtrees has size $n_{S_i} \geq \frac{1}{3} B$, we recursively decompose S_i and the rest of the tree. Otherwise, if $n_{S_1}, \ldots, n_{S_d} < \frac{1}{3} B$, let j be the index that minimizes

$$\left|(n_{S_1} + \cdots + n_{S_j}) - (n_{S_{j+1}} + \cdots + n_{S_d})\right| < \tfrac{1}{3} B.$$

We split v into two “clones”, v_1 and v_2, and give v_1 the neighbors $S_1, \ldots, S_j, v_2$ and v_2 the neighbors $v_1, S_{j+1}, \ldots, S_d$. The edge $\{v_1, v_2\}$ is marked as a “false” edge and separates T into two components, which are recursively decomposed.

If we start with a tree T with $n_T \geq B$, then all clusters have between $\frac{1}{3} B = \lg^{1+\varepsilon}(n)$ and B nodes, and the number of clusters is $\Theta(n/B)$. Note that intra-cluster edges are always true edges, hence the representation of clusters using micro-trees remains unaffected.

We thus obtain the same result as in Theorem 4 for general forests:

Theorem 5 *Given a planar embedding of a forest G on n vertices, we can store G in $2n + o(n)$ bits such that operations (i) and (ii) from Theorem 4 take constant time, operations (iii) – (v) take $O(\log n)$ time and (vi) and (vii) take $O(\log^{1+\varepsilon}(n))$ time.*

5 Acknowledgments

The first author was funded by NSERC Discovery Grant RGPIN-07185-2020.

References

[1] L. Ferres, J. Fuentes-Sepúlveda, T. Gagie, M. He, and G. Navarro. Fast and compact planar embeddings. *CGTA*, 89:101630, 2020.

[2] M. Rauch Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[3] J. Holm and E. Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory Comp. Sys.*, 61(4):1054–1083, 2017.

[4] G. Navarro. *Compact Data Structures: A Practical Approach.* Cambridge University Press, 2016.

[5] R. Raman, V. Raman, and S. Rao Satti. *Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets.* *ACM Trans. Alg.*, 3(4):43–es, 2007.

[6] K. G. C. von Staudt. *Geometrie de Lage.* Bauer und Raspe, Nürnberg, 1847.

[7] R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. Comp.*, 14(4):862–874, 1985.

Turning Around and Around: Motion Planning through Thick and Thin Turnstiles

Aster Greenblatt* Oscar Hernandez† Robert A. Hearn‡ Yichao Hou§ Hiro Ito¶ Minwoo Kang‖ Aaron Williams** Andrew Winslow††

Abstract

We examine the computational complexity of turnstile puzzles, which are grid-based tour puzzles with walls and turnstiles. A turnstile is a wall that can be rotated in 90° increments, either clockwise or counter-clockwise, around a central pivot when pushed by the player's token. In a 'thick' turnstile, the pivot and arms occupy cells of the grid, whereas in a 'thin' turnstile the pivot and arms occupy grid points and lines, respectively. We prove that reaching an exit is PSPACE-hard, even when restricted to just one of the following turnstiles types: ┬ , ─ , ┌ , ┬ , ─ , or ┌ . This establishes PSPACE-hardness for a dozen video games spanning several decades, including Kwirk (1989), Pokemon Ruby (2002), and Super Mario Odyssey (2017). Our hardness results are obtained by applying the motion planning framework in Jayson Lynch's PhD thesis *A framework for proving the computational intractability of motion planning problems* [MIT, 2020]. We also show that the decision problem can be solved in polynomial-time when restricted to ┼ or ┼ . We also formulate new open problems, and provide a survey of puzzles and games using turnstiles, which have also been called rotating doors, revolving gates, and spinning blocks.

1 Introduction

This article explores the familiar territory of grid-based motion planning, but with a few twists. In Section 1.1 we discuss the mechanism that we investigate, and in Section 1.2 we specify the tool that we will use.

*Division of Science, Mathematics, and Computing, Bard College at Simon's Rock, `asterj.greenblatt@gmail.com`

†Department of Mathematics & Statistics, University of Alaska Fairbanks, `oihernandez@alaska.edu`

‡`bob@hearn.to`

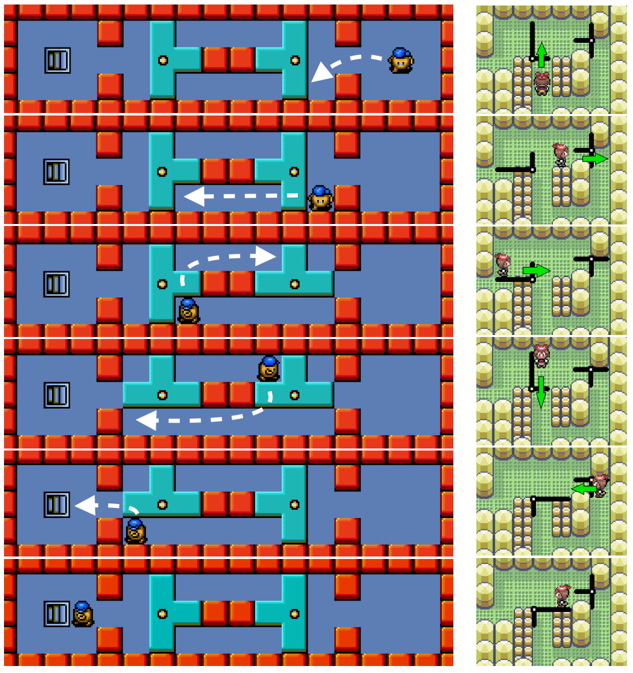
§`yhou17@simons-rock.edu`

¶School of Informatics and Engineering, University of Electro-Communications, `itohiro@uec.ac.jp`

‖Electrical Engineering and Computer Science, University of California, Berkeley, `minwoo_kang@berkeley.edu`

**Department of Computer Science, Williams College, `aaron.williams@williams.edu`

††`andrewwinslow@gmail.com`



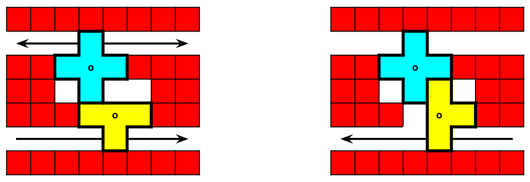
(a) Level 1. (b) Fortree City.

Figure 1: Solutions from (a) *Kwirk*, and (b) *Pokemon Ruby*.

1.1 Pushing, Pulling, Sliding, . . . and Rotating

There have been numerous studies on the computational complexity of grid-based puzzles and games that use pushing, pulling, or sliding as a core mechanism, with early contributions involving pushing by Fryers and Greene [11], Dor and Zwick [10], and Culberson [5]. More recently, in the *Games in Particular* section of Hearn and Demaine's *Games, Puzzles, and Computation*, 7 of the 10 puzzles use some form of pushing or sliding, including generalizations of Dad's Puzzle and Sokoban [15]. Research in this area is still active, with Barr et al. using a side-view and normal gravity [4], and Ani et al. [1] on block pulling being recent examples.

We consider a push-based rotation mechanism. Turnstiles have appeared in physical puzzles and video games for at least four decades, with two examples in Figure 1. Our main result is that single-player turnstile puzzles are PSPACE-hard, even when restricted to any of the following shapes: ┬ , ─ , ┌ , ┬ , ─ , or ┌ .



(a) State 1 with traversals. (b) State 2 with traversals.

Figure 2: One of the first gadgets that we designed has two 'tunnels' (black arrows) and two states. (a) The top tunnel is traversable in both directions since the ✚ turnstile can spin freely, and the bottom tunnel is traversable only from left-to-right since the ┳ turnstile can only spin counter-clockwise. (b) The top tunnel is not traversable in either direction since the ✚ turnstile is blocked, and the bottom tunnel is traversable only from right-to-left since the ┳ turnstile can only spin clockwise. It is not possible to move between the two tunnels, and traversing the bottom tunnel always causes the state to change.

1.2 Motion Planning Framework

Jayson Lynch's PhD thesis [16] facilitates "proof by picture" hardness results. For example, Figure 2 shows one of the first constructions developed by the authors (see Section 5.2). Although it appears to do "something" non-trivial, we were unable to integrate it into a standard hardness reduction. Miraculously, the framework views this as a "non-crossing toggle lock (NTL)" gadget, and it suffices as the key component of a PSPACE-hardness proof. The framework[1] was developed and applied across the following publications.

- A simplified framework was introduced in *Computational Complexity of Motion Planning of a Robot through Simple Gadgets* at FUN 2018 by Demaine, Grosof, Lynch, and Rudoy [7]. Its focus is on two state gadgets and a single agent.

- A generalization appears in *Toward a General Complexity Theory of Motion Planning: Characterizing Which Gadgets Make Games Hard* by Demaine, Hendrickson, and Lynch at ITCS 2020 [9].

- The framework has been applied in *Trains, Games, and Complexity: 0/1/2-Player Motion Planning through Input/Output Gadgets* by Ani, Demaine, Hendrickson, and Lynch [3], and also in [2].

As indicated by the third item, the general framework is quite broad. In many ways, it promises to be an agent-based analog to the well-known constraint logic framework developed by Hearn and Demaine [14, 15]. In this paper, we are only concerned with 1-player puzzles, so the simplified setting of [7] still holds particular value.

[1]Appendix A includes an auxiliary series of images that shows how Figure 1a can be modeled using the framework.

1.3 Outline

In Section 2, we discuss rotation mechanisms, with a focus on turnstile puzzles. In Sections 3–4, we introduce concepts from the motion planning framework, and show how they can be applied to turnstile puzzles. Section 5 concludes with a summary, and open problems. Throughout the paper, we include additional information with an eye towards facilitating new research.

2 Rotation Mechanisms

Readers are likely familiar with several pushing mechanisms found in the literature, many of which can be categorized using the `Push[Push]-1/k/*-[X]` classification, as discussed in Demaine, Demaine, Hoffmann, and O'Rourke [6] (also see [8]). Some of these variations have been inspired by video games, with *Sokoban* (1981) and *Pengo* (1982) being two prominent examples. In Sokoban, the player controls a warehouse worker who can push a single box one grid cell at a time, and the worker moves with the box. In Pengo, the player controls a penguin who can push a single ice block, but in this case, the block slides along the ice as far as possible, and the penguin remains stationary.

Similarly, there are many rotation mechanisms found in video games, but fewer of them have been studied in the literature. We discuss five such mechanisms in Section 2.1. Then in Section 2.2 we define our decision problem, and prove that two special cases can be solved efficiently. Section 2.3 provides a survey of games and puzzles that have used the turnstile mechanism.

2.1 Specific Mechanisms

Here we discuss five different rotation mechanisms found in video games. The first mechanism was previously studied using the motion planning framework [7]. To the best of our knowledge, the remaining mechanisms have not previously been considered.

2.1.1 k-Spinners

The Legend of Zelda: Oracle of Seasons and *Oracle of Ages* were released for Nintendo's Game Boy Color in 2001. Both games include a mechanism that is referred to as a 4-*spinner* in [7]. The mechanism can be embedded in a 3-by-3 grid of cells, with a pivot in the center, walls in each corner, and open chambers in the remaining four cells. The player interacts with the mechanism by entering one the chambers. Once inside, the 4-spinner turns 90°, and then stops, so that the player must exit the chamber. The direction of the turn depends on its state. In its blue state, the spinner turns counterclockwise, while in its red state, it turns clockwise. After the spinner has rotated, it changes state.

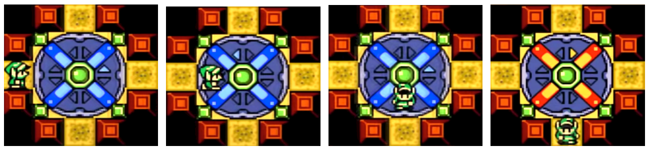
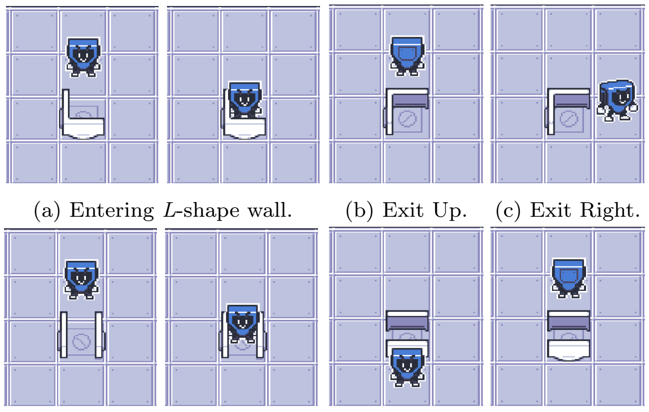


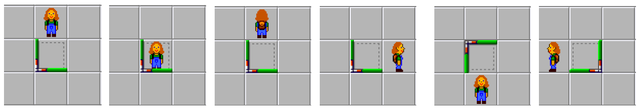
Figure 3: 4-Spinner. Link enters the chamber on the left. The spinner is blue, so it rotates 90°counterclockwise, and Link must exit downward. Once Link exits, the spinner turns red, indicating that its next rotation is 90°clockwise.



(a) Entering L-shape wall. (b) Exit Up. (c) Exit Right.

(d) Entering =-shape wall. (e) Exit Down. (f) Exit Up.

Figure 4: Rotating walls. (a) Circuit Dude enters an L-shape wall from an open side, then (b)–(c) can exit through either open side, and upon exiting, the walls rotate 90°clockwise in-place. (d)–(f) The =-shape wall behaves similarly. Note: Rotating walls appeared earlier in Bobby Carrot.



(a) Entering swivel (b) Exit ↑ (c) Exit → (d) Exit ↓ (e) Exit ←

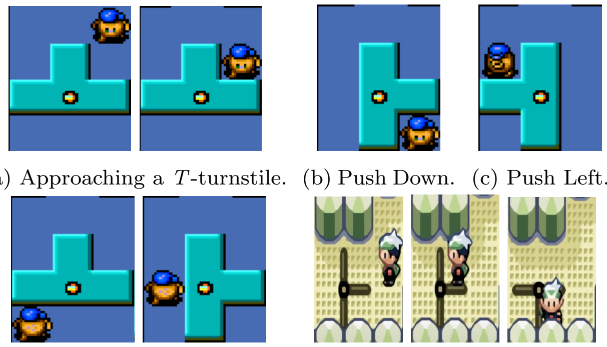
Figure 5: Swivel Door. (a) Melinda enters the swivel through an open side, then (b)–(c) exiting through an open side leaves the swivel unchanged, or (d)–(e) exiting through a wall causes the swivel to rotate 90°so that side opens.

Figure 3 illustrates a 4-spinner from the Moonlit Grotto in the Oracle of Ages. A *k-spinner* generalizes a 4-spinner by having k chambers.

Theorem 1 ([7]) *Motion planning is PSPACE-complete when restricted to k-spinners, for all $k \geq 4$.*

2.1.2 Rotating Walls

Bobby Carrots is a series of early mobile puzzle games. The first game was implemented in Java (J2ME) and was available for various handsets in 2004, including Nokia phones running Symbian. The most recent game,



(a) Approaching a T-turnstile. (b) Push Down. (c) Push Left.

(d) Rotate with no extra move. (e) Turning a thin T-turnstile.

Figure 6: Turnstiles. (a) Kwirk approaches an internal corner of a thick turnstile, then (b)–(c) turns it clockwise or counterclockwise with a single push, with the trailing arm moving Kwirk one extra cell. (d) Pushing the turnstile with a trailing arm does not cause the extra move. (f) Thin turnstiles behave similarly, but without the extra move.

Bobby Carrot 5: Forever, was released on the Nintendo Wii and iPhone[2]) and elsewhere in 2011. The series includes a both L-shaped and =-shaped *rotating walls*. These mechanisms occupy one grid cell, and a pair of *thin walls* occupy two of the four gridlines inside its perimeter. The player can enter or exit the cell through either open side, and the cell rotates 90°once the player exits it. The same mechanism has been used in other games, including *Circuit Dude*, which launched on the Arduboy in 2016, followed by iOS / Android / Steam / Switch. Figure 4 illustrates the mechanism.

2.1.3 Swivel Doors

Chip's Challenge was released for the Atari Lynx in 1989, and was popularized in *Microsoft Entertainment Pack 4* for Windows 3.1 in 1992. Its sequel, *Chip's Challenge 2* (CC2), was developed in 1999, but was not available to the public until its 2015 release on Steam. The sequel added new mechanisms, including the *swivel door*. The swivel door is visually identical to L-shaped rotating walls, as they both involve two consecutive thin walls around a single grid cell. Furthermore, the player enters the mechanisms in the same way (i.e. through one of the two openings). However, Chip and Melinda can exit a swivel door in any of the four cardinal directions. If they exit through either of the two openings, then the swivel door does not change. Otherwise, if they exit through one of the walls, then the cell rotates 90°. More specifically, the rotation is chosen so that the exited side of the mechanism becomes open. In other words, the mechanism behaves as if the walls are slid out of the way upon exit. Figure 5 provides an illustration.

[2]Due to Apple's iconoclastic update policies, this game, and so many other classics, are no longer available on their platforms.

2.1.4 Thick Turnstiles

Puzzle Boy (🇯🇵 1989) was developed for the Game Boy by Atlus, and localized as *Kwirk* (🇺🇸 1990). The game uses *thick turnstiles*, which have a central pivot that occupies a single cell, and arms that occupy at most one cell radiating outward in each direction, producing the ✚ , ┳ , ━ , ┏ , and ╺ (and their rotations). Each mechanism can rotate 90°around its pivot, either clockwise or counterclockwise, when the player pushes one of its arms. If a turnstile can rotate, then the player will also move in the direction of the push. Furthermore, if an arm is following behind the player, and would occupy their cell after the rotation, then the player is pushed one extra cell by the trailing arm. A turnstile cannot be rotated if a wall, or another turnstile, occupies a cell that the arm would need to rotate through. Figure 6 shows a thick T-shaped turnstile.

2.1.5 Thin Turnstiles

Lady Bug is a maze chase game released in arcades in 1981 by Universal, and later ported to home consoles. The game features *thin turnstiles*, which have a central pivot that occupies a grid point, and arms that radiate out along single grid lines, in the shape ┌ (or its rotation │). In *Pokemon Ruby and Sapphire*, the mechanism was generalized to include additional shapes. The mechanism behaves similarly to thick turnstiles. However, the player is never pushed an extra cell by a trailing arm, since the arms occupy grid lines instead of cells. Figure 6 shows a thin T-shaped turnstile.

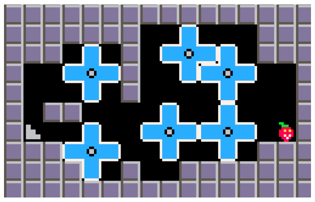
When considering multiple thin turnstiles, it is natural to wonder if *overlapping arms* are allowed. In other words, can the arms of two turnstiles occupy the same grid line?[3] This question is not answered by the Pokemon games, since the turnstiles are placed in ways that prevent this possibility. We take the position that overlapping arms are never allowed. Thus, a thin turnstile cannot be given a particular rotation if that rotation would result in an overlap. For the same reason, we do not mix thin and thick turnstiles in the same puzzles.

2.2 Turnstile Decision Problem

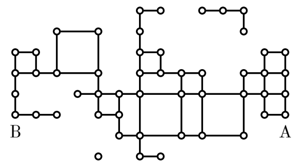
The Turnstile decision problem takes as input a *level* L, which is a grid in which the cells, points, and lines form walls and well-formed turnstiles (either thick or thin), with an *initial position* and *exit position*. If the player can move from the initial position to the exit by a sequence of moves, then the output is yes. The proof of Proposition 2 appears in Appendix B.

Proposition 2 Turnstile *is in PSPACE.*

[3]This is not a concern with rotating walls or swivel doors, since their thin obstacles are internal to a specific grid cell.



(a) A level with ✚ turnstiles.



(b) Graph corresponding to (a).

Figure 7: When restricted to ✚ , the Turnstile decision problem can be solved using undirected s-t connectivity.

We will prove that Turnstile is PSPACE-hard, and hence PSPACE-complete. In fact, we'll see that Figure 2 guarantees this. Therefore, we'll focus on determining whether Turnstile remains PSPACE-hard (and PSPACE-complete) when the turnstile types are restricted. Let Turnstile$_S$ denote the restriction of Turnstile to levels that only use turnstiles from the set S. For example, we now prove that the decision problem can be solved efficiently when restricted to ✚ or ┼ . Figure 7 illustrates the proof of Proposition 3.

Proposition 3 Turnstile$_S$ *for* $S = \{$✚$\}$ *is decidable in polynomial-time.*

Proof. We'll show that the decision problem can be transformed into an undirected s-t connectivity problem in polynomial-time.

First observe that the ✚ turnstile is unique amongst the thick turnstiles in that it has only one orientation. In other words, rotating a ✚ turnstile does not change which cells are occupied. Therefore, when $S = \{$✚$\}$, we can partition the turnstiles into two classes: those that can always rotate, and those that can never rotate. Furthermore, it is easy to identify those in the latter category, since they are precisely those in which one of the four cells that are diagonally-adjacent to its pivot is occupied by a wall or the arm of another ✚ turnstile.

We create a graph associated with the level as follows. Begin with a grid graph with points associated with cells of the grid that are either empty, along with the initial the exit positions. Now we consider each turnstile that can rotate. Given such a turnstile whose pivot occupies position (i, j) of the grid, we add edges of the form (a, b) where $a \in \{(i+1, j+1), (i-1, j-1)\}$ and $b \in \{(i-1, j+1), (i+1, j-1)\}$. In other words, we add a square that connects the empty cells that are diagonally-adjacent to the pivot. The creation of this graph can be done in polynomial-time.

Finally, we check if the start and exit positions are connected, which can be done in deterministic log-space. Hence, the overall algorithm takes polynomial-time. □

The following proposition can be proven similarly.

Proposition 4 Turnstile$_S$ *for* $S = \{$┼$\}$ *is decidable in deterministic log-space.*

2.3 History

Tables 1–2 list puzzles and games that use thick or thin turnstiles, either as a primary or secondary mechanism, respectively. Our results apply to all of the entries, except for *Lady Bug* and *Drelbs*, which are action games rather than puzzle games, and the physical puzzle *Turnstiles* which is a multi-player game.

3 Gadgets in General

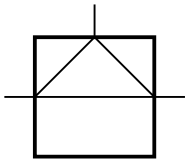
We begin our presentation of the motion planning framework by considering gadgets and their properties. Our presentation focuses on sets and functions, and we use the graphical styles from both the initial framework [7] and the generalized framework [9]. We begin by introducing the simplest gadgets: hallways and 1-toggles.

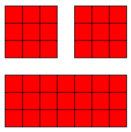
3.1 Branching Hallways

A *branching hallway* simply connects three locations. Since there is no gravity in TURNSTILE, it is easy to implement these gadgets.

Proposition 5 *Branching hallways can be implemented in* TURNSTILE *without turnstiles.*

Proof. The branching hallway gadget from [7] is shown below on the left, with an implementation on the right.


Branching hallway.


Implementation in TURNSTILE.

□

3.2 1-Toggle

A *1-toggle* connects two locations, A and B, via a dynamic path known as a *tunnel*. In state 1, it possible to travel from A to B, while in state 2, it is possible to travel from B to A. Furthermore, the gadget toggles its state whenever one of these *traversals* in completed. The 1-toggle is illustrated in several ways in Figure 8.

- Figure 8a is the graphical style from [7], in which a single diagram represents two states. In this type of diagram, a directed arrow represents a *toggle*, and traversing it causes its direction to switch.
- Figure 8b is the graphical style from [9], in which both states are shown. One can think of the rounded rectangles as being drawn on top of each other, since the left side of both rectangles refer to the same location, as do the right sides of both

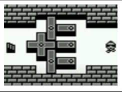
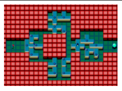
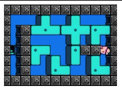
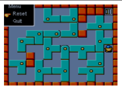
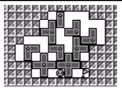
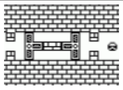
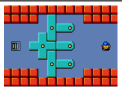
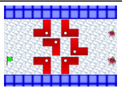
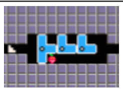
Release	Title	Developer	Platform(s)	Screenshot
1989 🇯🇵 1990 🇺🇸	Puzzle Boy Kwirk	Atlus	Game Boy	
1989 🇯🇵	Maze-kun (Mr. Maze)	Telenet Japan	NEC PC-8800, MSX 2	
1990 🇯🇵	Puzzle Boys	Atlus	Famicom Disk System	
1991 🇯🇵	Puzzle Boy	Telenet Japan	NEC PC Engine	
1991 🇯🇵 1992 🇺🇸	Puzzle Boy 2 Amazing Tater	Atlus	Game Boy	
1998	Kwirk[†]	Ludvig Strigeus	TI-89 / TI-90 Calculator	
2003	Shin Megami Tensei: Nocture	Atlus	PlayStation 2	
2003	Devil Children: Puzzle de Call!	Atlus	Game Boy Advance	
2008	Puzzle Boy Flash[†]	Blawars	Web browser	
2013	Kwirk Free[†] Kwirk 2[†]	Galiksoft	Android / Amazon App Store	
2020	Turner[†]	Pico Beast	Pico-8	

Table 1: Video games using thick turnstiles. [†]clone

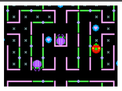
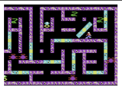
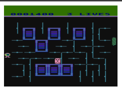
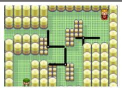
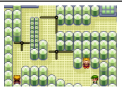
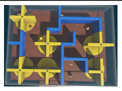
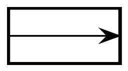
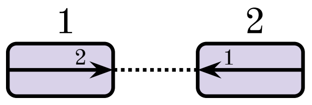
Release	Title	Developer	Platform(s)	Screenshot
1981	Lady Bug	Universal	Arcade Intellivision Colecovision	
1983	Lady Tut	Programe	Apple][Commodore 64	
1983	Drelbs	Synapse Software	Atari 8-bit Apple][Commodore 64	
2002 🇯🇵 2003 🇺🇸	Pokemon Ruby and Sapphire	Game Freak	Game Boy Advance	
2004 🇯🇵 2005 🇺🇸	Pokemon Emerald	Game Freak	Game Boy Advance	
2012	Turnstile	ThinkFun	Physical puzzle	
2014	Pokemon Omega Ruby and Alpha Sapphire	Game Freak	Nintendo 3DS	
2017	Super Mario Odyssey	Nintendo	Nintendo Switch	

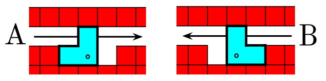
Table 2: Puzzles and games using thin turnstiles.



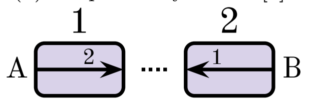
(a) Graphical style from [7].



(b) Graphical style from [9].



(c) Implementation in Kwirk. Left: State 1. Right: State 2.



(d) Graphical style from [9] with location labels.

Figure 8: A 1-Toggle gadget in several graphical styles, and implemented with one . In state 1, the gadget's only traversal is $A \to B$ (left to right). In state 2, the gadget's only traversal is $B \to A$ (right to left). The state changes if and only if a traversal occurs.

rectangles. In this style, the arrows are labeled with the state number that the gadget changes to after that particular traversal. The dotted line illustrates that the tunnel is reversible (i.e. once a traversal is complete, it can be done in reverse, and the gadget will return to its previous state).

- Figure 8c illustrates how a 1-toggle can be implemented with a single turnstile. The two states can again be interpreted as being drawn on top of each other, and in this case we have explicitly labeled the left location as A, and the right location as B.

- Figure 8d provides our slight modification of the graphical style from [9], which is the result of student feedback from the second-last author's course on the theory of computation. Specifically, we add explicit location labels, and we shorten the dotted line between the states. The first modification makes it easier to discuss the gadget, while the second is avoid a common misunderstanding. When an A to B traversal is conducted in state 1, the gadget toggles to state 2, and the agent ends in location B. The agent does not magically transport back to location A, as some have interpreted the dotted line to indicate.

3.3 Definition of a Gadget

Now we formally define a gadget. A *gadget* is a triple $g = (n, L, T)$ whose components are defined below, where $[x]$ denotes $\{1, 2, \ldots, x\}$.

- n is the finite number of *states*.

- L is a finite ground set of m *locations*.

- $T \subseteq [n] \times L \times [n] \times L$ is a set of *traversals*. An individual *traversal* is a 4-tuple $(s_1, \ell_1, s_2, \ell_2) \in T$, where s_1 is the *current state*, ℓ_1 is the *entry location*, s_2 is *next state*, and ℓ_2 is the *exit location*.

The reader may wonder why the definition includes a set of locations, rather than the number of locations. The reason is that sets allow for gadgets to share, or not share, locations. For example, we could have two instances of a 1-toggle, the first with states $L_1 = \{A, B\}$, and the second with states $L_2 = \{B, C\}$.

In later sections, we will be considering systems of gadgets, and how they are connected to each other. In this context, planarity is a consideration, and we need to know the cyclic order of the locations around a gadget. This leads to the following definition.

A *planar gadget* is a pair $p = (g, \pi)$ where

- $g = (n, L, T)$ is a gadget.

- π is a cyclic order of the gadget's locations. (In other words, π is a necklace over L.)

4 Gadget Types

At the end of Section 3, we formally defined gadgets. Now we define several specific types of (planar) gadgets, and show how to implement them with turnstiles.

As a simple example, we can formally define a *1-toggle* as $g_{1T} = (n, L, T)$, with $n = 2$ states, two locations $L = \{A, B\}$, and two transitions

$$T = \{(1, A, 2, B), (2, B, 1, A)\}.$$

There are no planar variations of this gadget, since at least four locations are required to have multiple cyclic orders of the location set.

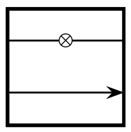
When implementing a gadget type, it is important to think of the type as a template, rather than a single gadget. More specifically, an *implementation* of a gadget must have the property that separate copies are *independent* in the sense that they do not share any state. For example, if a video game has a button that acts globally (i.e. opens or toggles all doors), then it cannot be used in an implementation. This is will not be an issue in TURNSTILE since each turnstile has its own orientation. For example, if we make two copies of the 1-toggle implementation in Figure 8c, then they will act independently of each other.

4.1 Noncrossing Toggle Lock (NTL)

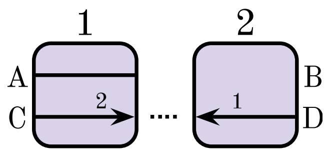
A *toggle-lock (TL)* is a gadget $g_{TL} = (n, L, T)$ with $n = 2$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions T:

$$\{(1, A, 1, B), (1, B, 1, A), (1, C, 2, D), (2, D, 1, C\}. \quad (1)$$

In other words, a toggle lock has two tunnels, A-B and C-D. The first tunnel is traversable in both directions in state 1, and is not traversable in state 2, while the second tunnel is always traversable in one-direction.

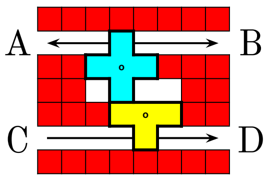


(a) NTL drawing from [7].

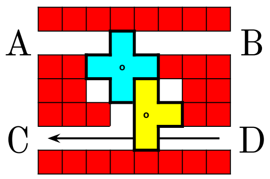


(b) NTL drawing from [9].

Figure 9: Noncrossing toggle lock (NTL) gadget.



(a) State 1 of NTL.



(b) State 2 of NTL.

Figure 10: Implementing a noncrossing toggle-lock.

Traversing the second tunnel toggles the gadget's state, which changes the traversability of both tunnels. The first tunnel will be referred to as a *lock* in [7].

We are particularly interested in the one of the planar toggle-locks.

- A *noncrossing toggle-lock (NTL)* is a pair $p = (g_{TL}, \pi)$ with $\pi_n = ABDC$.

The π_n order implies that the two tunnels do not cross each other. The NTL gadget is illustrated in Figure 9.

The first non-trivial gadget that the authors constructed happened to be a noncrossing toggle-lock, and Figure 2 is reproduced in Figure 10. Fortuitously, the simplified motion planning framework proves that the ability to implement NTL gadgets (and branching hallways) is sufficient for establishing PSPACE-hardness. One detail that we mention

Theorem 6 Tᴜʀɴsᴛɪʟᴇ *is PSPACE-complete.*

Proof. The decision problem is in PSPACE by Proposition 2, and branching hallway can be implemented in Tᴜʀɴsᴛɪʟᴇ by Proposition 5. Figure 2 implements a noncrossing toggle-lock. Therefore, the result follows from Corollary 5.2 of [7]. ☐

4.2 Crossing 2-Toggle (C2T)

A *2-toggle (2T)* is a gadget $g_{2T} = (n, L, T)$ with $n = 2$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions:

$$T = \{(1, A, 2, B), (1, C, 2, D), (2, B, 1, A), (2, D, 1, C\}.$$

In other words, a 2-toggle has two tunnels, A-B and C-D, which are always one-directional, and traversing either either tunnel toggles the direction of both tunnels.

We are particularly interested in the one of the planar 2-toggles.

- A *crossing 2-toggle (C2T)* is a pair $p = (g_{2T}, \pi_c)$ with $\pi_c = ADBC$.

The order π_c implies that the two cross each other. The C2T gadget is illustrated in Figure 11.

Although we did not have a name for it at the time, the second non-trivial gadget that the authors constructed was a crossing 2-toggle using a pair of ┳ turnstiles. Later we added thin turnstiles to our investigation, and it was possible to mimic the construction with thick turnstiles using a pair of ┬ turnstiles. In both cases, the turnstiles are positioned so that they each have two possible orientations, and a single enclosed cell between them. During a traversal, the player rotates one of the turnstiles to enter the enclosed cell, then must either retreat, or turn the other turnstile to continue along the direction that the entered from. Figures 12 and 13 illustrate the two constructions. Amazingly, the simplified motion planning framework again proves that these images are central to establishing PSPACE-hardness.

Theorem 7 Tᴜʀɴsᴛɪʟᴇ *is PSPACE-complete when restricted to thick T-shaped or thin T-shaped turnstiles. That is,* Tᴜʀɴsᴛɪʟᴇ$_S$ *is PSPACE-complete when* $S = \{$┳$\}$ *or* $S = \{$┬$\}$.

Proof. The decision problem is in PSPACE by Proposition 2, and branching hallway can be implemented in Tᴜʀɴsᴛɪʟᴇ by Proposition 5. Figures 12 and 13 implement a crossing 2-toggle $S = \{$┳$\}$ and $S = \{$┬$\}$, respectively. Therefore, the result follows from Corollary 5.2 of [7]. ☐

4.3 Locking 2-Toggles (L2T)

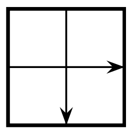
A *locking 2-toggle (L2T)* is a gadget $g_{L2T} = (n, L, T)$ with $n = 3$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions:

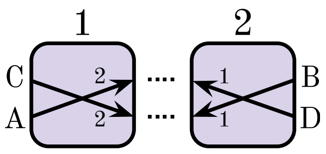
$$T = \{(1, B, 3, A), (2, D, 3, C), (3, A, 1, B), (3, C, 2, D\}.$$

In other words, a locking 2-toggle has two tunnels, A-B and C-D, which are always traversable in at most one direction. In state 3, both tunnels are traversable, and traversing the first tunnel changes the gadget to state 1, while traversing the second tunnel changes the gadget to state 2. In state 1, only the first tunnel is traversable (in the opposite direction), while only the second tunnel is traversable (in the opposite direction) in state 2.

We consider two planar locking 2-toggles.

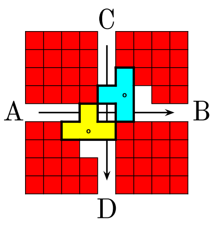
- A *parallel locking 2-toggle (PL2T)* is a pair $p = (g_{L2T}, \pi_p)$ with $\pi_p = ACDB$.


(a) C2T drawing from [7].

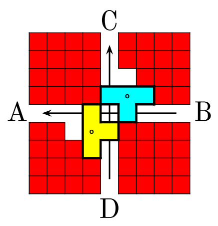


(b) C2T drawing from [9].

Figure 11: Crossing 2-toggle (C2T) gadget drawing..

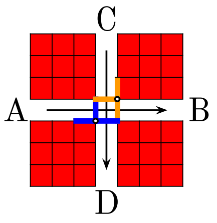


(a) State 1 of C2T

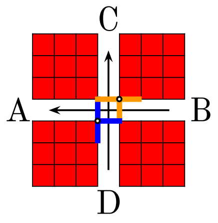


(b) State 2 of C2T

Figure 12: Crossing 2-toggle using .



(a) State 1 of C2T



(b) State 2 of C2T

Figure 13: Crossing 2-toggle using .

- An *antiparallel locking 2-toggle (AL2T)* is a pair $p = (g_{L2T}, \pi_a)$ with $\pi_a = ADCB$.

Both orders implies that the two tunnels do cross each. In the first case, the traversal directions in state 3 are the same, while in the second case, the traversal directions in state 3 are opposite. The PL2T and AL2T gadgets are illustrated in Figure 14 and 16, respectively. Since the L2T gadget has three states, the simplified graphical style in [7] cannot be used.

We were able to implement the PL2T gadget in one way, and the AL2T gadget in three ways. In each case, the general idea is to place two 1-toggles together in such a way that they interfere with the other 1-toggle in one or the two states. Figures 15 illustrates our construction of PL2T, and Figures 17, 18, and 19 illustrate our constructions of AL2T. The generalized motion planning framework again proves that these images are central to establishing PSPACE-hardness.

Theorem 8 Turnstile *is PSPACE-complete when restricted to thick or thin L-shaped or 1-shaped turnstiles. That is,* Turnstile$_S$ *is PSPACE-complete when* $S = \{\ \}$ *or* $S = \{\ \}$ *or* $S = \{\ \}$ *or* $S = \{\ \}$.

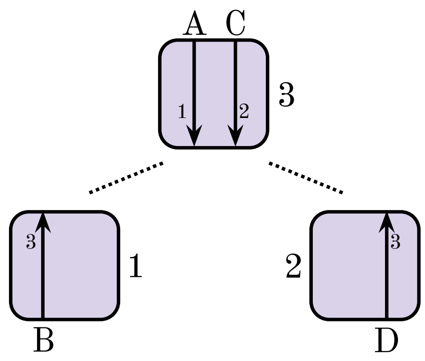
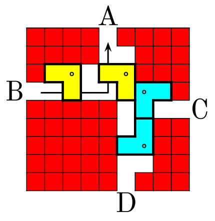
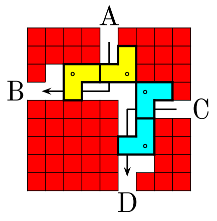


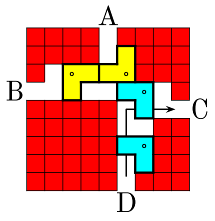
Figure 14: Parallel locking 2-toggle (PL2T) gadget drawing.



(a) State 1 of PL2T



(b) State 3 of PL2T



(c) State 2 of PL2T

Figure 15: Parallel locking 2-toggle using .

Proof. The decision problem is in PSPACE by Proposition 2. Figures 15, 18, 17, and 19, implement locking 2-toggles with $S = \{\ \}$ or $S = \{\ \}$ or $S = \{\ \}$ or $S = \{\ \}$, respectively. Therefore, the result follows from Theorem 10 of [9]. □

5 Summary and Future Work

We have shown that motion planning through turnstiles is PSPACE-complete for three shapes of turnstiles, regardless of whether they are thick or thin. However, the decision problem is solvable in polynomial-time when restricted to +-shaped turnstiles that are thick or thin. The unresolved singleton case is *i*-shaped turnstiles (i.e. and); the pairings of +-shaped and *i*-shaped is also open. These complexity results are summarized in Figure 20 for thick turnstiles, and the same results hold for thin turnstiles.

5.1 Future Work

One can classify the turnstiles that we have considered in this article as *short turnstiles*, in the sense that the arms have length one. Considering long turnstiles is a natural next step, and these mechanisms would better model the Pokemon series of games (see Figure 1b).

Rotating walls and swivel doors from Section 2 also provide open problems. Another variation involves *ratchet rotation*, in which turnstiles can only turn one direction (i.e. clockwise or counterclockwise).

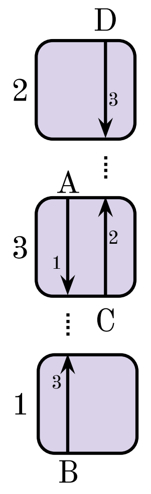
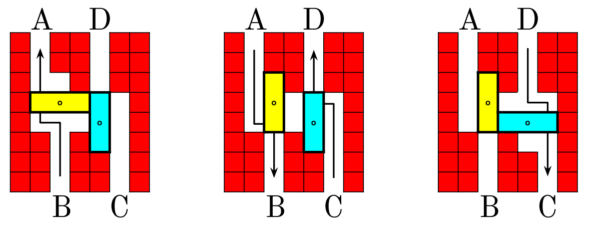
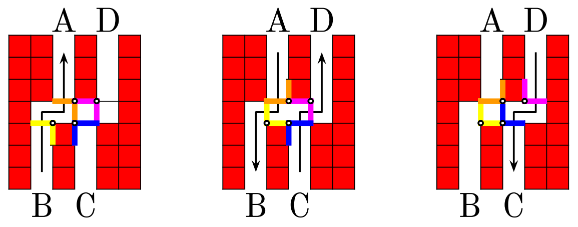


Figure 16: Antiparallel locking 2-toggle (AL2T) gadget.



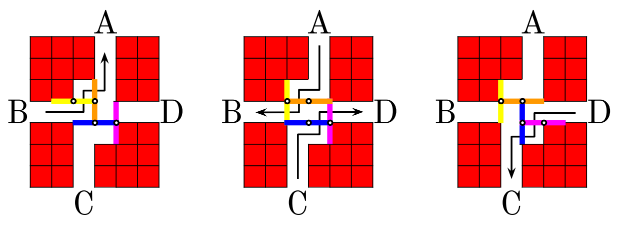
(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 17: Antiparallel locking 2-toggle using ▬ .



(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 18: Antiparallel locking 2-toggle using ┏ .



(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 19: Antiparallel locking 2-toggle using ━ .

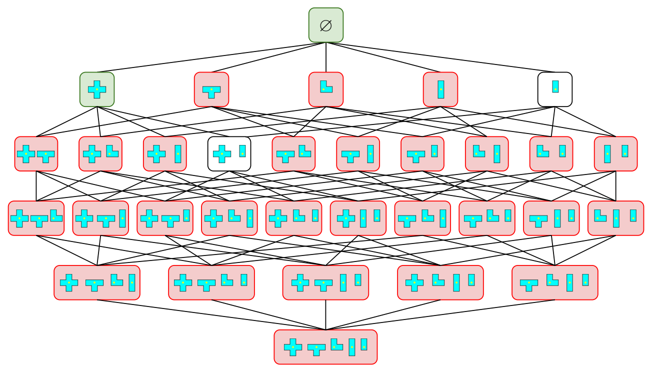


Figure 20: A summary of computational complextiy results for the TURNSTILE$_S$ decision problem for all $S \subseteq \{$✚, ┳, ━, ┏, ▬$\}$. The subsets in red nodes are PSPACE-complete, and the subsets in green nodes are in P. The two subsets in white nodes, $\{$▬$\}$ and $\{$✚, ▬$\}$, remain open.

5.2 Acknowledgements

The initial work on this project was conducted at the *32nd Bellairs Winter Workshop on Computational Geometry* in early 2017. We thank participant Mikhail Rudoy for pointing us towards the motion planning framework, which was still in its early stages of development at the time. We also thank Erik Demaine and Jayson Lynch for several helpful discussions.

References

[1] J. Ani, S. Asif, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, J. Lynch, S. Scheffler, and A. Suhl. Pspace-completeness of pulling blocks to reach a goal. *J. Inf. Process.*, 28:929–941, 2020.

[2] J. Ani, J. Bosboom, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, and J. Lynch. Walking through doors is hard, even without staircases: Proving pspace-hardness via planar assemblies of door gadgets. In M. Farach-Colton, G. Prencipe, and R. Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 3:1–3:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[3] J. Ani, E. D. Demaine, D. H. Hendrickson, and J. Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. *CoRR*, abs/2005.03192, 2020.

[4] A. Barr, C. Chang, and A. Williams. Push-2-f is PSPACE-complete. In *Proceedings of the 33rd Canadian Conference on Computational Geometry, Dalhousie University, Halifax, Canada, August 10-12, 2021*, 2021.

[5] J. Culberson. Sokoban is PSPACE-complete. In *In Proceedings of the 1st International Conference on Fun with Algorithm*, pages 65–76, 1998.

[6] E. Demaine, M. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *Computational Geometry*, 26:21–36, 2003.

[7] E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In H. Ito, S. Leonardi, L. Pagli, and G. Prencipe, editors, *9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy*, volume 100 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[8] E. D. Demaine, R. A. Hearn, and M. Hoffmann. Push-2-f is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry, University of Lethbridge, Alberta, Canada, August 12-14, 2002*, pages 31–35, 2002.

[9] E. D. Demaine, D. H. Hendrickson, and J. Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 62:1–62:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[10] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215 – 228, 1999.

[11] M. Fryers and M. T. Greene. Sokoban, 1995.

[12] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. MazezaM levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, pages 109–110, 2017.

[13] A. Greenblatt and A. Williams. MazezaM – puzzle game. `https://community.arduboy.com/t/mazezam-puzzle-game/3723/25`.

[14] R. A. Hearn and E. D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. J. Eidenbenz, and R. Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2002.

[15] R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation.* A K Peters/CRC Press, 1st edition, 2009.

[16] J. R. Lynch. *A framework for proving the computational intractability of motion planning problems.* PhD thesis, MIT, 9 2020. https://dspace.mit.edu/handle/1721.1/129205.

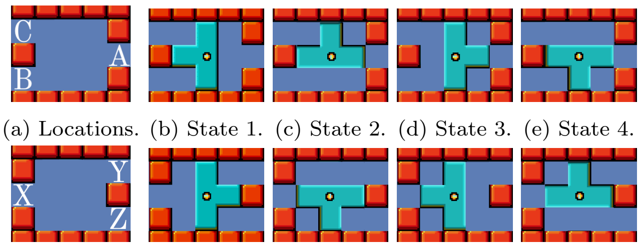
[17] B. North. Simpler exponential MazezaM level family. `https://bennorth.github.io/simpler-exponential-mazezam/index.html`.

Appendix

We conclude with several notes, which may be helpful for some readers.

A Modeling Level 1

Figures 21–23 shows how Level 1 in Kwirk can be modeled using gadgets and the motion planning framework. Note that the two gadgets are simply rotations of each other.



(a) Locations. (b) State 1. (c) State 2. (d) State 3. (e) State 4.

(f) Locations. (g) State 1. (h) State 2. (i) State 3. (j) State 4.

Figure 21: (a)–(e) Gadget $g_1 = (n_1, L_1, T_1)$ with $n_1 = 4$ states, location set $L_1 = \{A, B, C\}$, and traversal set $T_1 = \{(1, A, 2, B), (1, A, 3, B), (1, A, 3, C), (1, A, 4, C), \ldots\}$. (f)–(j) Gadget $g_2 = (4, L_2, T_2)$ with $L_2 = \{X, Y, Z\}$ and $T_2 = \{(1, X, 2, Y), (1, X, 3, Y), (1, X, 3, Z), (1, X, 4, Z), \ldots\}$.

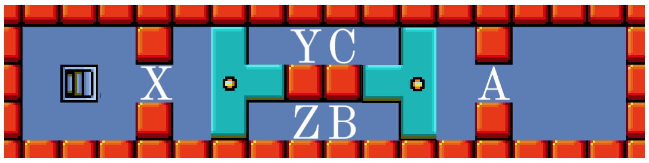
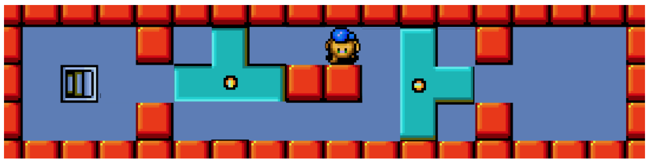
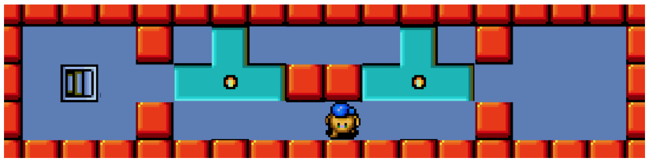


Figure 22: Modeling Level 1 from Figure 1a using the system of gadgets $S = (G, C)$, with gadget set $G = \{g_1, g_2\}$, and connections $C = \{\{B, Z\}, \{C, Y\}\}$. Note that gadget g_1 is on the right.



(a) System state (a, s_1, s_2) with agent location $a = C$, and gadget states $s_1 = 3$ and $s_2 = 4$.



(b) System state (a, s_1, s_2) with agent location $a = B$, and gadget states $s_1 = 2$ and $s_2 = 4$.

Figure 23: (a)–(b) A traversal move in gadget g_1 in system S. The traversal is $(3, C, 2, B) \in T_1$, and the agent can move right, down, down, right, left, left, left to make this traversal.

B Membership in PSPACE

The proof of Proposition 2 uses standard techniques and appears below.

Proof. We will prove that TURNSTILE is in NPSPACE, which establishes the result by Savitch's theorem. For simplicity, our argument only considers levels with thick turnstiles; thin turnstiles can be handled by considering grid lines as well as grid cells.

Consider an instance of the problem TURNSTILE(L) in which L has a total of n grid cells. The size of the input is then $O(n)$ since each cell can be occupied by a small number of different game elements, hence, each cell contributes a small number of bits to the level's encoding.

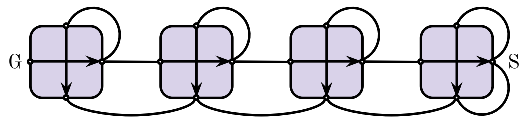
Our algorithm starts in the initial state of the level, and then non-deterministically makes moves (i.e. up, down, left, right) until the player reaches the exit position and the algorithm returns yes, or we exceed a pre-determined maximum number of moves. The maximum number of moves that we allow is chosen to be at least the number of different states that the level can have, since if there is a solution, then there is a solution that does not repeat any states. The state of a level consists of the position of the player's token, and the orientation of each turnstile. Each turnstile can have at most 4 orientations, and the number of turnstiles is at most n. Thus, the number of states is at most $n \cdot 4^n$. To implement our algorithm, we need to store the current state of the level, and the move counter. The state of the level requires $O(n)$-bits, and the move counter requires $\log n \cdot 4^n = 2n \log n$ bits. Therefore, our algorithm uses $O(n \log n)$ nondeterministic space. $\square$

C Explicit Exponential Level Construction

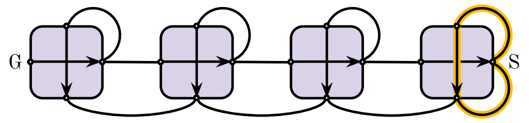
There are two distinct benefits to constructing simple levels that require an exponential number of moves to solve.

1. It illustrates that the simplest certificate (i.e. the sequence of moves) is not sufficient for establishing membership in NP. For example, see [12, 17] for the puzzle game *MazezaM*.
2. Game developers may wish to include such a level in their game. For example, this is true for MazezaM on the Arduboy [13].

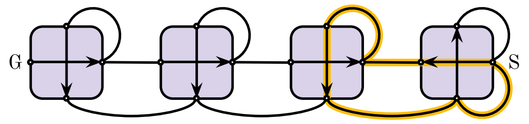
For these reasons, a lexicographic 4-bit binary counter using crossing 2-toggles (C2T) is given in Figure 25, along with an implementation in Kwirk using turnstiles.



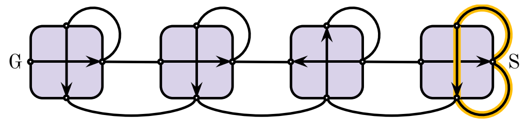
(a) A 4-bit counter in its initial state $b_4b_3b_2b_1 = 0000$.



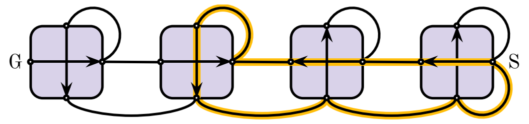
(b) State 0000. The highlighted loop complements b_1.



(c) State 0001. The highlighted loop complements b_2b_1.

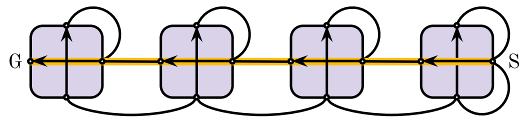


(d) State 0010. The highlighted loop complements b_1.



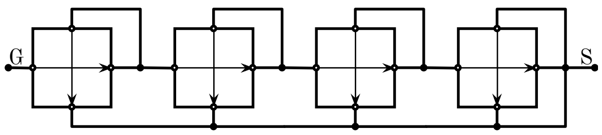
(e) State 0011. The highlighted loop complements $b_3b_2b_1$.

⋮



(f) State 1111. Reaching the goal via the highlighted path.

Figure 24: Binary counting with motion planning gadgets. (a) A 4-bit counter using crossing 2-toggles. (b)–(f) Each image begins with the player at the start location S, and the counter state is a binary string $b_4b_3b_2b_1$ based on the state of each gadget. The player must traverse states $0000, 0001, \ldots, 1111$ to reach the goal location G.



(a) The 4-bit counter drawn with branching hallways as per [7].

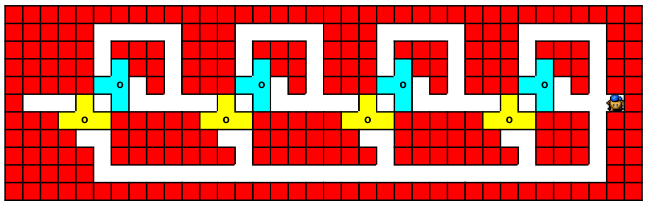

(b) A 4-bit counter in Kwirk.

Figure 25: (a) The 4-bit counter from Figure 24 implemented in Kwirk using the crossing 2-toggle with $S = \{\ \}$. It follows the same design, but with (a) branching hallways, as in the style of [7].

Integer Cow-path Problem and Simple Robot Street Search

Azadeh Tabatabaei[*] Farehe Soheil[†] Mohammad Aletaha[‡] Mohammad Ghodsi[§]

Abstract

In this paper, we revisit the well-known cow-path problem and introduce a new variation called Integer Cow-path Problem (ICP). In the general cow-path problem, w rays with one common end-point and a robot standing on the end-point are given. A target point is put along one of the rays and can be detected only when it is reached by the robot. The robot has to find the target by traversing the rays starting from the end-point. In the ICP, the robot is restricted to take an integer number of steps. The goal is to design a strategy for the robot to find the target such that the length of the traveled path is as small as possible. We present a randomized strategy that gives an upper bound on the competitive ratio for the ICP. Furthermore, as an application of this variation, we study the Simple Robot Street Search problem and give a randomized strategy that is inspired from the strategy for the ICP.

1 Introduction

The problem of searching in unknown geometric environments is a fundamental problem in the fields of computational geometry, robotics, and online algorithms [13]. The cow-path problem is a well-known problem that has been studied by mathematicians and computer scientists [3, 6]. Many geometric search problems have applied the cow-path search algorithms.

In the general cow-path problem, we are given $w \geq 2$ rays with a common end-point s and a target point t that is put along one of the rays, see Figure 1(a). A searcher (robot), which is not aware of the location of the target t, must find it. The searcher can only move along the rays and cannot detect t before reaching it. The name «cow-path» comes from the scenario in which, metaphorically a cow is searching for her calf in w concurrent paths [15]. Note that the cow-path problem is also called star-search or ray-search. Furthermore, when $w = 2$, it is called searching on a line or linear search, see Figure 1(b).

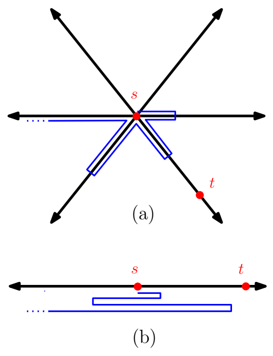


Figure 1: (a) An example of cow-path problem with $w = 6$. The robot's search path is shown in blue (b) Searching on a line ($w = 2$).

Searching in an unknown environment can be seen as an online problem since the robot does not have access to the information about the position of the target and must decide in an online manner. The notion of the competitive analysis is proposed for measuring the performance of the online algorithms [17]. In the cow-path problem, the competitive ratio is the length of the path traveled by the robot from the start point s to the target point t over the shortest path from s to t. A strategy is α-competitive if its competitive ratio is at most α.

The distances in this problem are expressed in steps. A *step* is the unit of measurement in the literature [6]. We revisit the general cow-path problem in a way that the robot is limited to take an integer number of steps along the rays. While in the general problem such a limitation does not exist and the robot can take any real number of steps, see Figure 2. Notice that like the general cow-path problem, the searcher stops whenever it achieves the target and does not go further even if it is in a middle of a step. We name this new variation *Integer Cow-path Problem* (ICP) and will study it in section 2.

The motivation behind considering such a variation (ICP) is to use simpler and cheaper searching agents (robots). The robots which can take any real number of steps need powerful movement capabilities, while the robots whose number of steps are limited to integers have simpler and less powerful movement capabilities. More powerful and complicated dynamic abilities will always lead to more expenses and costs. A stepper-

[*]Department of Computer Engineering, University of Science and Culture, Iran. a.tabatabaei@usc.ac.ir

[†]Department of Computer Engineering, Sharif University of Technology, Iran. soheil@ce.sharif.edu

[‡]Department of Computer Engineering, Sharif University of Technology, Iran. mohammadaletaha@ce.sharif.edu

[§]Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM), Iran. ghodsi@sharif.edu

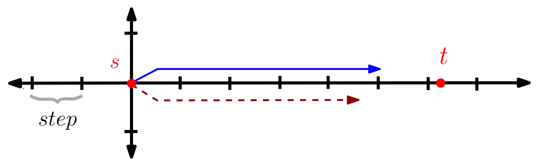


Figure 2: Example of the cow-path problem with $w = 4$. Traversing an integer number of steps (shown in blue) in the ICP, and traversing a real number of steps (shown in dashed line) in the general problem. The target t is placed $n \in \mathbb{R}$ $(n \geq 1)$ steps away from s.

motor (or stepping-motor) is an electric motor that instead of rotating continuously, rotates in a number of discrete equal steps. Then, one can command the motor to move or hold at each step without any position sensor for feedback (see [2] for more details about stepper-motors). A stepper-motor makes the robot take only an integer number of steps when attached to one. Note that we configure the step-size of the robot's stepper-motor to be equal to the step-size of the problem. In the general cow-path problem the searcher might take any real number of steps along the rays. Hence, we have introduced the ICP to find the same target in the same environment with a simpler robot equipped with a stepper-motor.

As an application of the ICP, we study the *Simple Robot Street Search* problem. This problem refers to searching for a specific target in a particular polygonal environment (a street polygon) using a minimal sensing robot. We will discuss it properly in section 3.

1.1 Our contribution

In this paper, we introduce and study the Integer Cow-path Problem (ICP) and present the first randomized search strategy for it. In fact, this strategy gives an upper bound on the competitive ratio of the ICP. Our randomized strategy for the ICP is β-competitive where

$$\beta = \min_{r \in \mathbb{Z}, r>1} \left\{ \frac{2\,(r^w - 1)}{w(r-1)\ln r} + (w+1)\cdot(0.56) + 1 \right\}$$

and w is the number of rays, given as the input of the problem. We have summarized the competitive ratio of our strategy for some values of w in Table 1.

Furthermore, we study the Simple Robot Street Search problem and show that in the worst case, this problem can be seen as the ICP with $w = 2$. So, we give a 6.29-competitive randomized strategy that uses the strategy for the ICP as a subroutine.

2 Integer Cow-path Problem (ICP)

In this section, first, we express the Integer variation of the Cow-path Problem (ICP) in detail and mention some related works. Then, we present a randomized strategy for the ICP and finally discuss the performance of the proposed strategy in the remaining of this section.

w	β
2	6.29
3	9.98
4	14.63
5	22.25
6	35.22

Table 1: The competitive ratio β for some values of w for the *integer* cow-path problem.

2.1 Problem Definition

Given $w \geq 2$ concurrent rays sharing a common endpoint s, and a robot $\mathcal{R}$ standing on s. A target point t is put $n \in \mathbb{R}$ $(n \geq 1)$ steps away from s along an arbitrary ray. The robot $\mathcal{R}$ moves in a round-to-round manner and can only move back and forth along the rays. At each round $i \in \mathbb{Z}^{0+}$, $\mathcal{R}$ takes $d_i \in \mathbb{N}$ steps on ray $l_i \in \{0, 1, ..., w-1\}$ and if it does not find t, returns back to s and advances on round $i+1$. So, we use a function $S(i) = (d_i, l_i)$ to formulate the robot's moving strategy at round i.

Note that before t is reached, $\mathcal{R}$ always takes an integer number of steps. Since the target's distance can be non-integer $(n \in \mathbb{R})$, $\mathcal{R}$ is allowed to take only one non-complete step during its travel, which is when it encounters t. The robot $\mathcal{R}$ immediately stops as soon as it reaches t even in a middle of a step. Let D denote the length of the path (in steps) traveled by $\mathcal{R}$ from s to find t. The problem refers to designing an online strategy minimizing $C = D/n$, where C is the competitive ratio of the strategy.

Note that a lower bound on n is necessary for having a bounded competitive ratio. Otherwise, the competitive ratio will be unbounded in the worst case. Consider t is placed at distance $\epsilon > 0$ steps from s on ray j. Now, if the robot takes λ steps in the first round, on any other ray $k \neq j$, then the competitive ratio is at least $\frac{\lambda}{\epsilon}$ and it might be unbounded. Hence, like many previous works, we assume $n \geq 1$.

2.2 Related works

The general cow-path problem was first proposed by Bellman [8] and then, studied by mathematicians like Beck and Newman [7] and Gal [12] who gave solutions to this problem. After that, the problem was rediscovered by computer scientists. Baeza-Yates et al. [6] and Kao et al. [15] gave optimal deterministic and randomized strategies for the general problem, respectively. The deterministic strategy in [6] is λ-competitive where

$$\lambda = 2 \cdot \frac{w^w}{(w-1)^{w-1}} + 1$$

and the randomized strategy in [15] is γ-competitive where

$$\gamma = \min_{r>1} \left\{ \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r} + 1 \right\}.$$

Here, we mention the values of λ and γ for some w in Table 2. So far, several variations of the general problem

w	λ	γ
2	9	4.6
3	14.5	7.74
4	19.97	10.85
5	25.42	13.95
6	30.86	17.04

Table 2: The competitive ratios λ and γ for some values of w for the *general* cow-path problem.

have been introduced and studied by researchers, including moving target, having turn cost, the existence of lower and upper bounds on the target's location, and maximum clearance [5, 9, 10, 11].

2.3 Algorithm for the ICP strategy

Here, we present a randomized strategy for the ICP. Our strategy is a modified version of the SmartCow strategy by Kao et al. [15] such that it always generates integer values for d_i at each round i. In fact, we apply the idea of rounding using the ceiling function. We denote this strategy by *IntegerCow* and express it as an algorithm in Algorithm 1. The analysis of the algorithm can be

```
Algorithm 1: IntegerCow
Compute a random permutation $\sigma$ of
  $\{0, 1, ..., w-1\}$
Compute the best value for $r$
Choose $\varepsilon$ u.a.r.[1] from $[0, 1)$
$d \leftarrow r^\varepsilon$
$i \leftarrow 0$
while the target $t$ is not achieved do
    Move along ray $\sigma(i \bmod w)$ up to $\lceil d \rceil$ step(s)
    Move back to the start point $s$
    $d \leftarrow r \cdot d$
    $i \leftarrow i + 1$
end
```

represented in terms of $r > 1$ which is a constant real value. Also, r approximately determines by what factor the number of steps in the next round should be more than the current round. We will show how to find the best value for r in the analysis section. Compared to the SmartCow algorithm, the modification is slight but it makes the analysis more complicated and non-trivial.

2.4 Analysis

Now, we obtain the competitive ratio of the IntegerCow algorithm. Since this algorithm is randomized, its competitive ratio is defined as the *expected* distance traveled by the robot over the actual distance between s and t. The algorithm is designed to move the robot $\lceil r^{i+\varepsilon} \rceil$ steps from s on round $i \geq 0$, i.e. $S(i) = (\lceil r^{i+\varepsilon} \rceil, l_{\sigma(i \bmod w)})$. As we noted earlier, n denotes the distance between s and t in the worst case. Let $k \in \mathbb{Z}^{0+}$ and $0 \leq \delta < 1$ such that $n = r^{k+\delta}$. Suppose t lies on ray l_j and let m be the *first round* where $\mathcal{R}$ travels a distance of *at least* $\lceil r^k \rceil$ steps from s on l_j.

Depending on the random order of rays in σ, the value of m in the *worst case* satisfies $k \leq m \leq k + w - 1$ (It is possible for m to take values less than k because of the ceiling function, but this only decreases the competitive ratio). Let D be the random variable denoting the overall distance traveled by $\mathcal{R}$ to find t. We denote the competitive ratio of the algorithm by C that is computed as $C = E[D]/n$. To obtain C, we need to calculate $E[D]$ for which there are two cases:

Case 1: When $m = c \geq k + 1$, $\mathcal{R}$ certainly finds t at round c and the expected value of D is calculated as

$$\begin{aligned} E[D|m = c] &= E\left[2\sum_{i=0}^{c-1} \lceil r^{i+\varepsilon} \rceil + n\right] \\ &= E\left[2\sum_{i=0}^{c-1} (r^{i+\varepsilon} + \lceil r^{i+\varepsilon} \rceil - r^{i+\varepsilon}) + n\right] \\ &= 2\sum_{i=0}^{c-1} \left(E\left[r^{i+\varepsilon}\right] + E\left[\lceil r^{i+\varepsilon} \rceil - r^{i+\varepsilon}\right]\right) + n. \end{aligned}$$

Now, we define functions $f(r, i)$ and $f_M(r)$ as follows:

$$f(r, i) = E\left[\lceil r^{i+\varepsilon} \rceil - r^{i+\varepsilon}\right] = \int_0^1 (\lceil r^{i+\varepsilon} \rceil - r^{i+\varepsilon}) d\varepsilon,$$

$$f_M(r) = \max_{i \in \mathbb{Z}^{0+}} f(r, i).$$

So, the expression for $E[D|m = c]$ can be written as

$$\begin{aligned} E[D|m = c] &= 2\sum_{i=0}^{c-1} \left(E\left[r^{i+\varepsilon}\right] + f(r, i)\right) + n \\ &\leq 2\sum_{i=0}^{c-1} \left(E\left[r^{i+\varepsilon}\right] + f_M(r)\right) + n \end{aligned}$$

[1]uniformly at random

$$
\begin{aligned}
&= \frac{2(r^c-1)}{r-1}\cdot E[r^\varepsilon] + 2c\cdot f_M(r) + n\\
&= \frac{2(r^c-1)}{r-1}\cdot \int_0^1 r^\varepsilon d\varepsilon + 2c\cdot f_M(r) + n\\
&= \frac{2(r^c-1)}{r-1}\cdot (\frac{r-1}{\ln r}) + 2c\cdot f_M(r) + n\\
&= \frac{2(r^c-1)}{\ln r} + 2c\cdot f_M(r) + n.
\end{aligned}
$$

Case 2: When $m = k$, depending on the values of ε and δ, $\mathcal{R}$ may find t on round k or may fail. In case of failure, it continues to search and finally finds t on round $k + w$. We classify this case into three events as follows:

F_1**:** When $\varepsilon > \delta$, then trivially $\lceil r^{k+\varepsilon}\rceil > r^{k+\delta}$ and $\mathcal{R}$ will find t on round k.

F_2**:** When $\varepsilon \le \delta$ and $\lceil r^{k+\varepsilon}\rceil \ge r^{k+\delta}$, again $\mathcal{R}$ will find t on round k.

F_3**:** When $\varepsilon < \delta$ and $\lceil r^{k+\varepsilon}\rceil < r^{k+\delta}$, $\mathcal{R}$ will not find t on round k. So, it continues searching and will find t on round $k + w$.

So, the expected traveled distance in case 2 is

$$
\begin{aligned}
E[D|m=k] &= Pr(F_1)\cdot E\left[2\sum_{i=0}^{k-1}\lceil r^{i+\varepsilon}\rceil + n \mid F_1\right]\\
&+ Pr(F_2)\cdot E\left[2\sum_{i=0}^{k-1}\lceil r^{i+\varepsilon}\rceil + n \mid F_2\right]\\
&+ Pr(F_3)\cdot E\left[2\sum_{i=0}^{k+w-1}\lceil r^{i+\varepsilon}\rceil + n \mid F_3\right].
\end{aligned}
$$

Similar to calculations of case 1, we have

$$
\begin{aligned}
E[D|m=k] &\le Pr(F_1)\cdot\left[\frac{2(r^k-1)}{r-1}\right]\cdot E[r^\varepsilon|F_1]\\
&+ Pr(F_2)\cdot\left[\frac{2(r^k-1)}{r-1}\right]\cdot E[r^\varepsilon|F_2]\\
&+ Pr(F_3)\cdot\left[\frac{2(r^{k+w}-1)}{r-1}\right]\cdot E[r^\varepsilon|F_3]\\
&+ Pr(F_1)\cdot 2k\cdot f_M(r) + Pr(F_2)\cdot 2k\cdot f_M(r)\\
&+ Pr(F_3)\cdot 2(k+w)\cdot f_M(r) + n.
\end{aligned}
$$

To calculate the expectations on r^ε, we define $\alpha = \min x \in [0,\delta]$ such that $\lceil r^{k+x}\rceil \ge r^{k+\delta}$. Notice that $r > 1$ and $0 \le \alpha \le \delta$. By the definition of α, for event F_2 we have $\alpha \le \varepsilon \le \delta$ and for event F_3 we have $0 \le \varepsilon < \alpha$. So, in events F_2 and F_3, we have $r^\alpha \ge 1$ and $r^\alpha \le r^\delta$, respectively. Now, we can compute the conditional expectations as follows:

$$
\begin{aligned}
E[r^\varepsilon|F_1] &= \int_\delta^1 \frac{r^\varepsilon d\varepsilon}{Pr(F_1)} = \frac{r-r^\delta}{Pr(F_1)\cdot\ln r},\\
E[r^\varepsilon|F_2] &= \int_\alpha^\delta \frac{r^\varepsilon d\varepsilon}{Pr(F_2)} = \frac{r^\delta-r^\alpha}{Pr(F_2)\cdot\ln r} \le \frac{r^\delta-1}{Pr(F_2)\cdot\ln r},\\
E[r^\varepsilon|F_3] &= \int_0^\alpha \frac{r^\varepsilon d\varepsilon}{Pr(F_3)} = \frac{r^\alpha-1}{Pr(F_3)\cdot\ln r} \le \frac{r^\delta-1}{Pr(F_3)\cdot\ln r}.
\end{aligned}
$$

Putting all these together, we have

$$
\begin{aligned}
E[D|m=k] &\le 2(k+w)\cdot f_M(r) + n\\
&+ \frac{2\cdot\left[(r-r^\delta)(r^k-1)+(r^\delta-1)(r^{k+w}+r^k-2)\right]}{(r-1)\ln r}.
\end{aligned}
$$

Thus, considering both cases and knowing that $Pr(m=j) = \frac{1}{w}$ for any $k \le j \le k+w-1$, the total expected traveled distance equals to

$$
\begin{aligned}
E[D] &= \sum_{j=k}^{k+w-1} Pr(m=j)\cdot E[D|m=j]\\
&= \frac{1}{w}\cdot E[D|m=k] + \sum_{j=k+1}^{k+w-1}\frac{1}{w}\cdot E[D|m=j].
\end{aligned}
$$

By the results of case 1 and case 2, we get

$$
\begin{aligned}
E[D] \le& \frac{2\cdot\left[(r-r^\delta)(r^k-1)+(r^\delta-1)(r^{k+w}+r^k-2)\right]}{w(r-1)\ln r}\\
&+ \frac{2}{w}(k+w)\cdot f_M(r) + \frac{n}{w}\\
&+ \frac{1}{w}\sum_{c=k+1}^{k+w-1}\left(\frac{2(r^c-1)}{\ln r} + 2c\cdot f_M(r) + n\right).
\end{aligned}
$$

After simplification, we have

$$
\begin{aligned}
E[D] \le& \frac{2\cdot\left[r^{k+\delta+w}-r^k-r^\delta-rw+w+1\right]}{w(r-1)\ln r}\\
&+(2k+w+1)\cdot f_M(r) + n.
\end{aligned}
$$

Since $k,\delta \ge 0$ and $r > 1$, we have $-r^k < -1$, $-r^\delta \le -1$, and $-rw < -w$. So,

$$
E[D] \le \frac{2\left(r^{k+\delta+w}-1\right)}{w(r-1)\ln r} + (2k+w+1)\cdot f_M(r) + n.
$$

Since $k,\delta \ge 0$ and $n = r^{k+\delta}$, we have

$$
C \le \frac{2\left(r^{k+\delta+w}-1\right)}{r^{k+\delta}w(r-1)\ln r} + \frac{(2k+w+1)\cdot f_M(r)}{r^{k+\delta}} + 1. \tag{1}
$$

Using (1), we demonstrate the following lemma.

Lemma 1 *The competitive ratio C is upper bounded by*

$$
\min_{r>1}\left\{\frac{2\left(r^w-1\right)}{w(r-1)\ln r} + (w+1)\cdot f_M(r) + 1\right\}
$$

Proof. We are interested to obtain an upper bound on C and we want this upper bound to be as small as possible. The number of rays, w, is the input of the problem. The value of r is a parameter of the algorithm for which we determine its value. After deciding about the value of r, the adversary determines the value of k and δ to maximize the competitive ratio C. As a result, we must choose the proper value for r to assure that for a fixed value of δ and k, given by the adversary, still we obtain a minimum upper bound for C. So, we consider the right hand side of (1) as a function of r. This function has a minimum in $r \in [1, \infty)$ and it can be verified by taking the first derivative of the $r.h.s$ function with respect to r. Now, inspiring from (1), we define function $F_{rhs}(k, \delta)$ as below

$$\min_{r>1} \left\{ \frac{2\left(r^{k+\delta+w} - 1\right)}{r^{k+\delta} w(r-1)\ln r} + \frac{(2k + w + 1) \cdot f_M(r)}{r^{k+\delta}} + 1 \right\}.$$

It is obvious that $C \leq F_{rhs}(k, \delta)$. The adversary can determine the value of k and δ to maximize $F_{rhs}(k, \delta)$. It can be proven that $F_{rhs}(k, \delta)$ is maximized when $k, \delta = 0$ (by taking the derivatives, one can show this function is non-increasing with respect to $k, \delta \geq 0$). So, we can conclude that $C \leq F_{rhs}(0, 0)$ and this completes the proof of lemma. □

From now on, we focus on proving an upper bound for $f_M(r)$. Since we use Ramanujan's bounds for $\ln(n!)$ further, we claim it as the following theorem.

Theorem 2 [4] *For* $\ln(n!)$*, the following lower and upper bounds hold:*

$$\ln(n!) < n\ln(n) - n + \tfrac{1}{6}\ln\left(8n^3 + 4n^2 + n + \tfrac{1}{30}\right) + \tfrac{1}{2}\ln(\pi),$$
$$\ln(n!) > n\ln(n) - n + \tfrac{1}{6}\ln\left(8n^3 + 4n^2 + n + \tfrac{1}{100}\right) + \tfrac{1}{2}\ln(\pi).$$

Now, we state the following lemma which gives an upper bound for $f_M(r)$.

Lemma 3 *For all integers* $r > 1$*, we get* $f_M(r) < 0.56$.

Proof. According to definition of $f(r, i)$, we have

$$f(r,i) = \int_0^1 (\lceil r^{i+\varepsilon} \rceil - r^{i+\varepsilon}) d\varepsilon$$
$$= \int_0^1 \lceil r^{i+\varepsilon} \rceil \, d\varepsilon - \int_0^1 r^{i+\varepsilon} d\varepsilon. \tag{2}$$

The second part of (2) is easily calculated as

$$\int_0^1 r^{i+\varepsilon} d\varepsilon = \frac{r^{i+1} - r^i}{\ln r} \tag{3}$$

In order to compute the first part of (2), let $z = i + \varepsilon$, then $dz = d\varepsilon$ which gives

$$\int_0^1 \lceil r^{i+\varepsilon} \rceil \, d\varepsilon = \int_i^{i+1} \lceil r^z \rceil \, dz$$
$$= \int_0^{i+1} \lceil r^z \rceil \, dz - \int_0^i \lceil r^z \rceil \, dz.$$

Let

$$g(i) = \int_0^i \lceil r^z \rceil \, dz,$$

then

$$\int_0^1 \lceil r^{i+\varepsilon} \rceil \, d\varepsilon = g(i+1) - g(i). \tag{4}$$

Considering the graph of $f(x) = \lceil r^x \rceil$ in Figure 3, we can compute $g(i)$ by summing up the areas of the blue rectangles.

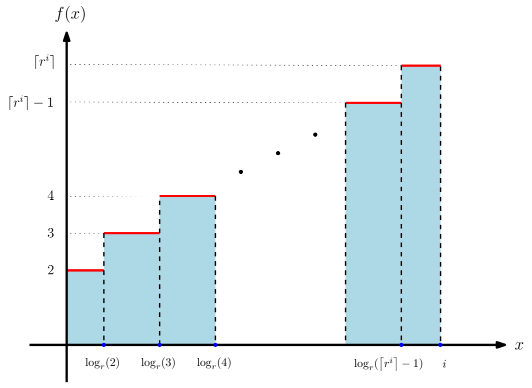


Figure 3: The function $f(x) = \lceil r^x \rceil$ is shown in red. The blue region is $g(i)$.

According to Figure 3, $g(i)$ can be calculated as:

$$g(i) = \int_0^i \lceil r^z \rceil \, dz = \sum_{j=2}^{\lceil r^i \rceil - 1} j \cdot (\log_r(j) - \log_r(j-1))$$
$$+ \lceil r^i \rceil \cdot \left(i - \log_r\left(\lceil r^i \rceil - 1\right)\right).$$

The above expression is hard to simplify for all real values of $r > 1$ because of the ceiling functions. So, by considering only integer values for r, one can reduce the difficulties induced by the ceiling functions. Since i is an integer, by letting r be an integer we will have $\lceil r^i \rceil = r^i$ and $i = \log_r \lceil r^i \rceil$. Note that r is a parameter of the algorithm not the input of the problem, so we are allowed to decide its value. Henceforth, we study r only for integer values, i.e. $r > 1$ and $r \in \mathbb{Z}$. So the above summation can be simplified as

$$g(i) = \sum_{j=2}^{r^i} j \cdot (\log_r(j) - \log_r(j-1))$$
$$= r^i \cdot \log_r(r^i) - \sum_{j=2}^{r^i - 1} \log_r(j)$$
$$= i \cdot r^i - \log_r\left((r^i - 1)!\right). \tag{5}$$

By applying (5) in (4) and using (3), we can calculate (2) as follows:

$$\begin{aligned} f(r,i) &= (i+1)r^{i+1} - \log_r\left((r^{i+1}-1)!\right) \\ &\quad - (i)r^i + \log_r\left((r^i-1)!\right) - \frac{r^{i+1}-r^i}{\ln r} \\ &= r^i\left((r-1)\,i + r - \frac{r-1}{\ln r}\right) - \log_r\left(\frac{(r^{i+1}-1)!}{(r^i-1)!}\right) \\ &= r^i\left((r-1)\,i + r - \frac{r-1}{\ln r}\right) - \log_r\left(\frac{r^{i+1}!}{r\cdot r^i!}\right) \\ &= r^i\left((r-1)\,i + r - \frac{r-1}{\ln r}\right) \\ &\quad - \frac{1}{\ln r}\left(\ln\left(r^{i+1}!\right) - \ln\left(r^i!\right)\right) + 1. \end{aligned}$$

Using inequalities in Theorem 2, we get

$$\begin{aligned} f(r,i) &< 1 + r^i\left((r-1)\,i + r - \frac{r-1}{\ln r}\right) \\ &+ \frac{1}{\ln r}\left(r^i\ln\left(r^i\right) - r^i - r^{i+1}\ln\left(r^{i+1}\right) + r^{i+1}\right) \\ &- \frac{1}{\ln r}\left(\frac{1}{6}\ln\left(8\left(r^{i+1}\right)^3 + 4\left(r^{i+1}\right)^2 + r^{i+1} + \frac{1}{100}\right)\right) \\ &+ \frac{1}{\ln r}\left(\frac{1}{6}\ln\left(8\left(r^i\right)^3 + 4\left(r^i\right)^2 + r^i + \frac{1}{30}\right)\right). \end{aligned} \tag{6}$$

We denote the right hand side of (6) by $f_{ub}(r,i)$. According to the definition of $f_M(r)$, any upper bound on $f(r,i)$ for all $i \in \mathbb{Z}^{0+}$ yields an upper bound on $f_M(r)$. If we fix r (for all integers $r > 1$), then $f_{ub}(r,i)$ is non-increasing with respect to $i \in [0,\infty)$ (Section A in the Appendix covers the proof of this statement). So, the maximum value of $f_{ub}(r,i)$ happens at $i = 0$.

The function $f_{ub}(r,0)$ is non-increasing with respect to r (proof is similar). Therefore, for all integers $r > 1$ the maximum value of $f_{ub}(r,0)$ happens at $r = 2$ and it approximately equals to 5.557. So, we conclude that for all integers $r > 1$, we have

$$f_M(r) < f_{ub}(2,0) < 0.56.$$

This completes the proof of lemma. □

Now, by Lemma 1 and Lemma 3, we conclude the following theorem which is the main contribution of this paper.

Theorem 4 *The IntegerCow algorithm for the ICP is β-competitive where*

$$\beta = \min_{r\in\mathbb{Z}, r>1}\left\{\frac{2\left(r^w-1\right)}{w(r-1)\ln r} + (w+1)\cdot(0.56) + 1\right\}.$$

By Theorem 4, for any $w \geq 2$, Algorithm 1 computes the best value for r (denoted by r^*) which minimizes the above function. For some values of w, the values for r^* and their corresponding β are given in Table 3.

w	r^*	β
2	4	6.29
3	2	9.98
4	2	14.63
5	2	22.25
6	2	35.22

Table 3: The values of r^* and β for some w

3 Simple Robot Street Search

In this section, we introduce an application of the ICP called Simple Robot Street Search and give a randomized strategy for it, inspired by the ICP strategy.

3.1 Preliminaries

A simple polygon P with two distinct vertices s and t is called a *street* if the clockwise chain (L_{chain}) and counter-clockwise chain (R_{chain}) which are constructed on the polygon from s to t, are mutually weakly visible. In other words, each point on L_{chain} must see at least one point on R_{chain} and vice versa, see Figure 4.

The set of points that are visible from a point q in P is called the visibility polygon of q, denoted by $vis(q)$. An edge of $vis(q)$ which does not belong to the boundary of P is called a *gap* (sometimes called window), see Figure 4. Each gap g is induced by a reflex vertex called $ref(g)$ and a region of P is hidden on one of its sides called pocket of g. We assume a gap g is oriented such that its start point is $ref(g)$. The gap g is called a l-gap (r-gap), denoted by L (R), if the pocket of g lies on its left (right) side, see Figure 4.

A *simple robot* $\mathcal{B}$ is a point robot which can only detect gaps in a cyclical order, see Figure 4. As a technical point of view, it is equipped with a sensor which detects the order of discontinuities in depth information (gaps) in its visibility polygon. A simple robot can only

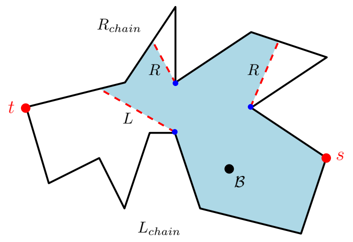


Figure 4: A street polygon with respect to s and t, R_{chain} and L_{chain}, two r-gaps and one l-gap. The visibility polygon of the robot $\mathcal{B}$ is shown in blue.

move toward gaps and the target t when it becomes visible. The unit of simple robot's movement is called *step* which is a constant distance that has been specified by the robot's manufacturer. Technically speaking, a stepper-motor is put on the robot which makes it move in discrete equal distances (steps). Furthermore, a simple robot has a *stopping function* that, whenever called, forces the robot to stop immediately even if in a middle of a step. For example, at the point that the target t becomes visible, one can call its stopping function to stop the robot and prevent it from going further in the wrong direction.

3.2 Problem Definition

We are given a street polygon P with point s and t and a simple robot $\mathcal{B}$. The robot $\mathcal{B}$ is placed at s and moves through P in an arbitrary number of steps. We assume a step is small enough compared to the scale of P. The robot is presumed to take an integer number of steps except when the strategy forces the robot to stop. The goal is to reach the target t starting from s such that the traversed path by $\mathcal{B}$ is as short as possible.

3.3 Related Works

In 1992, Klein [16] introduced street polygons and proposed a 5.73-competitive strategy for the street search problem using a 360°-vision robot. The 360°-vision robots can detect any vertex and edge of the polygon, measure any angle and distance between objects, and move freely in any direction. After several improvements, finally, Icking et al. [14] presented the optimal $\sqrt{2}$-competitive strategy for this problem.

The limited sensing model (gap sensor) that a simple robot is equipped with, was introduced by Tovar et al [22]. Note that unlike 360°-vision robots, a simple robot's vision and movement are strictly limited. For the first time, Tabatabaei et al. [20] used the simple robots for the street search problem and gave an 11-competitive strategy using auxiliary tools called pebbles. After that, Tabatabaei et al. [19] and Wei and Tan [23] independently presented the optimal 9-competitive deterministic strategies for the Simple Robot Street Search problem. In this paper, we study the latter problem and give a randomized 6.29-competitive strategy. The cooperation of two robots [1] and minimizing the number of turns in street searching [18], and searching in generalized streets [21] are other works that used simple robots.

3.4 Motion primitives

Gaps are maintained in a data structure called S-GNT [20]. While $\mathcal{B}$ is moving, combinatorial changes called critical events, occur in its visibility polygon which update S-GNT. There are four critical events:

- *Appearance* and *Disappearance* of gaps occur when the robot crosses the inflection rays.
- *Merge* and *Split* of gaps occur when the robot crosses the bitangent complements.

as illustrated in Figure 5. An inflection ray is the ex-

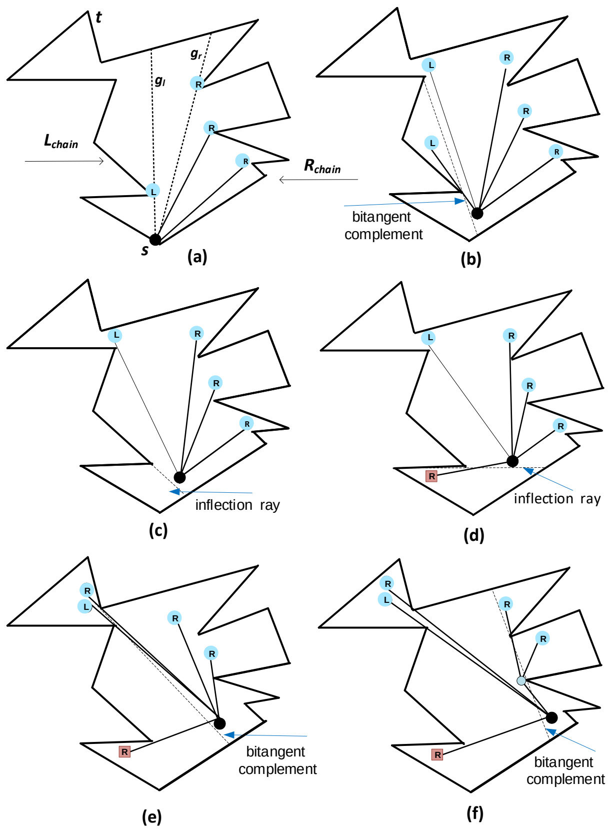


Figure 5: A street polygon and the dynamical changes of the gaps as the robot moves. The black circle is the location of $\mathcal{B}$. Pink squares and blue circles denote primitive and non-primitive gaps, respectively. (a) Existing gaps at s. (b) A split event. (c) A disappearance event. (d) An appearance event. (e) Another split event. (f) A merge event.

tension of the hidden edge of a gap into the interior of the polygon, see Figure 5(c). A bitangent complement is a line that is tangential to two reflex vertices of the polygon, see Figure 5(b). A new gap whose pocket was formerly visible is called *primitive*. Otherwise, it is called *non-primitive*, see Figure 5. We are only interested in probing non-primitive gaps since the region behind a non-primitive gap had never been visible to the robot before. We define g_l to be the rightmost non-primitive l-gap, and g_r to be the leftmost non-primitive r-gap, see Figure 5(a). As the robot moves, the critical events may dynamically change g_l and g_r in a way that one the followings happens:

- *Uni-gap:* if there exists only one of g_r or g_l or they are collinear, a uni-gap occurs, see Figure 6(a).
- *Funnel:* When both g_r and g_l exist, we say a *funnel* is created, see Figure 6(b). Also, by the robot's

movement, a funnel may end or a new one may start. We refer to the point, in which a funnel ends, a *critical point* of that funnel, see point 2 in Figure 6(b).

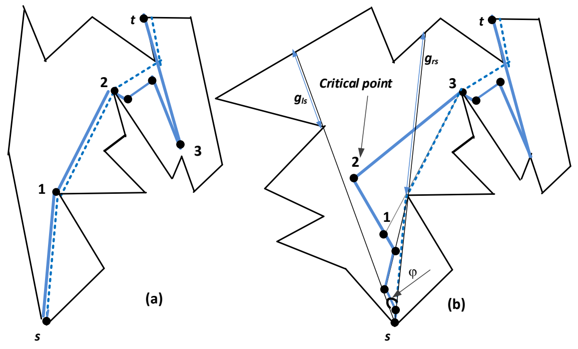


Figure 6: The bold path is the robot's search path, the dotted path is the shortest path. (a) There is only g_r. (b) Both g_r and g_l at the start point s and a funnel case. The angle φ between the gaps, is the opening angle.

In a funnel case, as the robot moves toward g_l or g_r, the following events update the location of g_l and g_r as well as the location of the funnel.

1. When g_r/g_l splits into itself and a new r-gap/l-gap. Then the g_r/g_l will be replaced by this new r-gap/l-gap (point 1 in Figure 6(b)).

2. When g_r/g_l splits into itself and a new l-gap/r-gap, then this new l-gap/r-gap will be set as g_l/g_r. This point is a critical point in which the funnel ends (point 2 in Figure 6(b)).

3. When g_l or g_r disappears, a critical point has been achieved and the funnel ends (point 3 in Figure 6(a)).

Note that the split and disappearance events may occur concurrently (point 3 in Figure 6(b)).

3.5 Strategy

When the target t is not visible, it is hidden behind one of the gaps. Otherwise, the robot could see it. The following theorem states that when t is not visible, it is behind one of g_l or g_r. So, we can ignore other ones and focus only on g_l and g_r.

Theorem 5 [20] *While t is not visible, it is behind g_l or g_r.*

In a uni-gap case, where there exists only one of g_r or g_l or they are collinear, according to Theorem 5, t is behind that gap, see Figure 6(a). But at each funnel, where both g_r and g_l exist, the robot is not aware behind which of g_r or g_l the target t is hidden. So, usually, a detour from the shortest path (an extra undesirable longer path) is unavoidable. Notice that potentially, there might be more than one critical point for a funnel. But, as soon as the robot achieves a critical point, according to its strategy, the funnel ends. So, $\mathcal{B}$ achieves only one critical point for each funnel.

Now, we demonstrate our strategy. Until $\mathcal{B}$ does not see the target t, it continues to search. If t becomes visible, $\mathcal{B}$'s stopping function will be called. So, $\mathcal{B}$ immediately stops and moves directly toward t. Before t becomes visible, at each point of the search path, two cases might happen:

1. If there exists only one of g_r or g_l (uni-gap case), then t is behind the existing gap, see Figure 6(a). In this case, $\mathcal{B}$ moves toward this gap until one of these happens: either t becomes visible, a funnel case occurs or $\mathcal{B}$ reaches the reflex vertex of this gap. In any of these three conditions, $\mathcal{B}$'s stopping function will be called.

2. If a funnel case occurs, to bound the detour, the robot moves toward g_r and g_l alternatively, see Figure 6(b). In fact, a funnel can be seen as two concurrent rays created by g_r and g_l. So, we can apply the ICP strategy when $w = 2$ with a minor difference. In the ICP when $w = 2$, the robot goes d steps along one ray, takes another d steps to turn back to the starting point, and then goes d' steps along the other ray. However in this strategy, after traversing d steps along a gap, the robot stops but does not turn back to the starting point of the funnel. Rather, it changes its direction toward the other gap and takes $d + d'$ steps in the new direction. As the robot moves, whenever t becomes visible or $\mathcal{B}$ reaches the critical point of the funnel, the strategy will call $\mathcal{B}$'s stopping function.

Each time the strategy calls $\mathcal{B}$'s stopping function, the robot stops and checks which of the above two cases is happening. We have demonstrated the above strategy as an algorithm in Algorithm 2.

3.6 Analysis

Throughout the search, in uni-gap cases or when the target t is visible, the search path coincides with the shortest path. But as noted earlier, in funnel cases detours are unavoidable. So, to prove the competitive ratio of the strategy, we compare the length of the traversed path and the shortest path in funnel cases. In this case, the angle between g_r and g_l, which is always smaller than π, is called the opening angle, denoted by φ, see Figure 6(b). In Lemma 6, we show that the detour from the shortest path depends on the size of φ.

Algorithm 2: Randomized Street Search

```
while the target $t$ is not visible do
    if a uni-gap case occurs then
        repeat
            Move toward the existing gap
        until a funnel case happens or reflex vertex of the gap is achieved or $t$ becomes visible;
    else if a funnel case occurs then
        $\{d_0, d_1\} \leftarrow \{r, l\}$
        Choose $i$ u.a.r. from $\{0, 1\}$
        Choose $\varepsilon$ u.a.r. from $[0, 1)$
        $j \leftarrow 4^{\varepsilon}$
        Move toward $g_{d_i}$ up to $\lceil j \rceil$ step(s)
        $i \leftarrow (i+1) mod(2)$
        repeat
            $d \leftarrow \lceil j \rceil + \lceil 4 \cdot j \rceil$
            Move toward $g_{d_i}$ up to $d$ steps
            $i \leftarrow (i+1) mod(2)$
            $j \leftarrow 4 \cdot j$
        until a critical point of the funnel is achieved or $t$ becomes visible;
end
Move directly toward $t$
```

Lemma 6 *In a funnel case, the detour from the shortest path for a small opening angle is shorter than the detour for a large opening angle.*

Proof. According to our strategy, in a funnel case, $\mathcal{B}$ moves toward g_r and g_l alternatively. In this alternative movement, moving toward one of the directions is correct and moving toward the other is a deviation. Assume that at point p_i, when a funnel case occurs, $\mathcal{B}$ moves toward g_r and reaches point p_{i+1} while the target t is behind g_l, see Figure 7(a). Now, to achieve t, $\mathcal{B}$ should traverse at least a distance

$$\delta = \sqrt{|p_ip_{i+1}|^2 + |p_iv_l|^2 - 2 \cdot |p_ip_{i+1}| \cdot |p_iv_l| \cdot \cos\varphi}$$

by the law of cosines, see Figure 7(a). It can be verified that δ is strictly increasing as a function of φ by taking the derivative with respect to φ where $0 < \varphi < \pi$. So, greater opening angle φ results in greater detour taken by the robot. □

The following theorem demonstrates the competitive ratio of the strategy.

Theorem 7 *Given a street P and a simple robot $\mathcal{B}$, Our randomized strategy for the street search problem is 6.29-competitive.*

Proof. As we noted before, except in funnel cases, the search path coincides with the shortest path. So, the detours from the shortest path happen only in the presence of funnels. Therefore, to compute the competitive ratio of our strategy, only funnel cases matter. In a funnel case, by Theorem 6, for larger opening angles the robot deviates more from the shortest path. Since φ never exceeds π, the worst case happens when the opening angle φ is adequately close to π and the robot can only move toward left and right. In this situation, searching for a critical point in a funnel case reduces to the ICP with $w = 2$, see Figure 7(b). Hence, to obtain the worst-case analysis, we consider the case when φ is very close to π. According to Algorithm 2, our strategy in a funnel case with opening angel φ close to π, operates the same as the strategy for the ICP presented in Algorithm 1 with $w = 2$ and $r = 4$. Note that for the ICP, we have obtained $r^* = 4$ for the case of $w = 2$, see Table 1. By Theorem 4, the strategy for the ICP is 6.29-competitive for $w = 2$ and this completes the proof of the theorem. □

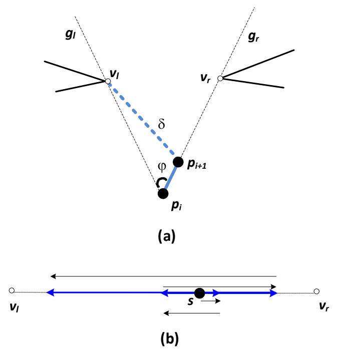


Figure 7: (a) p_ip_{i+1} is a detour from the shortest path. (b) The worst case.

4 Conclusion

We introduced and studied a new variation of the cow-path problem called the Integer Cow-path Problem (ICP). In this variation, the robot is restricted to take only an integer number of steps. We present a randomized strategy for this problem which gives an upper bound on the competitive ratio of all randomized strategies. Moreover, we use this strategy for the problem of searching in a street using a simple robot. Improving the upper bound and proving a lower bound on the competitive ratio can be one of the future works. Also, it is interesting to design deterministic strategies for the ICP. Albeit the doubling strategy in [6] is a deterministic solution to the ICP for $w = 2$, the ICP remains an interesting open problem for $w > 2$.

References

[1] M. Abouei Mehrizi, M. Ghodsi, and A. Tabatabaei. Robots' cooperation for finding a target in streets. In *International Conference on Topics in Theoretical Computer Science*, pages 30–43. Springer, 2015.

[2] P. P. Acarnley. *Stepping motors: a guide to theory and practice*. Number 63. Iet, 2002.

[3] S. Alpern and S. Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.

[4] G. E. Andrews and B. C. Berndt. *Ramanujan's lost notebook*, volume 1. Springer, 2005.

[5] S. Angelopoulos and M. Voss. Online search with maximum clearance. *arXiv preprint arXiv:2011.14144*, 2020.

[6] R. A. Baezayates, J. C. Culberson, and G. J. Rawlins. Searching in the plane. *Information and computation*, 106(2):234–252, 1993.

[7] A. Beck and D. Newman. Yet more on the linear search problem. *Israel journal of mathematics*, 8(4):419–429, 1970.

[8] R. Bellman. Problem 63-9, an optimal search. *SIAM review*, pages 274–274, 1963.

[9] P. Bose and J.-L. De Carufel. A general framework for searching on a line. *Theoretical Computer Science*, 703:1–17, 2017.

[10] P. Bose, J.-L. De Carufel, and S. Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.

[11] E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2-3):342–355, 2006.

[12] S. Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27(1):17–30, 1974.

[13] S. K. Ghosh and R. Klein. Online algorithms for searching and exploration in the plane. *Computer Science Review*, 4(4):189–201, 2010.

[14] C. Icking, R. Klein, E. Langetepe, S. Schuierer, and I. Semrau. An optimal competitive strategy for walking in streets. *SIAM Journal on Computing*, 33(2):462–486, 2004.

[15] M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.

[16] R. Klein. Walking an unknown street with bounded detour. *Computational Geometry*, 1(6):325–351, 1992.

[17] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[18] A. Tabatabaei and M. Ghodsi. Optimal strategy for walking in streets with minimum number of turns for a simple robot. In *International Conference on Combinatorial Optimization and Applications*, pages 101–112. Springer, 2014.

[19] A. Tabatabaei and M. Ghodsi. Randomized strategy for walking in streets for a simple robot. *arXiv preprint arXiv:1512.01784*, 2015.

[20] A. Tabatabaei and M. Ghodsi. Walking in streets with minimal sensing. *Journal of Combinatorial Optimization*, 30(2):387–401, 2015.

[21] A. Tabatabaei, F. Shapouri, and M. Ghodsi. A competitive strategy for walking in generalized streets for a simple robot. In *CCCG*, pages 75–79, 2016.

[22] B. Tovar, R. Murrieta-Cid, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, 2007.

[23] Q. Wei, X. Tan, and Y. Ren. Walking an unknown street with limited sensing. *International Journal of Pattern Recognition and Artificial Intelligence*, 33(13):1959042, 2019.

.

Appendix

A

To prove that $f_{ub}(r,i)$ is non-increasing with respect to $i \in [0,\infty)$, all we need is to show that the derivative of $f_{ub}(r,i)$ is least equal to zero. It can be calculated as:

$$\frac{d}{di}f_{ub}(r,i) = r^i(i(r-1)+r)\ln(r)$$
$$+ (\frac{1}{3})r^i\left(-3r\ln\left(r^{(i+1)}\right) + 3\ln\left(r^i\right)\right)$$

$$-(\frac{1}{3})r^i\left(\frac{50r\left(8r^{(i+1)}+24r^{(2i+2)}+1\right)}{100r^{(i+1)}+400r^{(2i+2)}+800r^{(3i+3)}+1}\right)$$

$$+(\frac{1}{3})r^i\left(\frac{15\left(24r^{2i}+8r^i+1\right)}{240r^{3i}+120r^{2i}+30r^i+1}\right)$$

After multiplying all the above terms, we get:

$$= r^i(ir-i+r)\ln r - r^i(ir-i+r)\ln r$$

$$+\frac{9600r^{4i+3}(1-r)+4800^{3i+1}(1-r^2)}{(800r^{3i+3}+400r^{2i+2}+100r^{i+1}+1)(240r^{3i}+120r^{2i}+30r^i+1)}$$

$$+\frac{1200r^{2i+1}(r-1)+24r^i(1-\frac{10}{3}r^2+3r^i-10r^{i+3})}{(800r^{3i+3}+400r^{2i+2}+100r^{i+1}+1)(240r^{3i}+120r^{2i}+30r^i+1)}$$

Since the first expression yields 0 and the denominator of the remaining fraction above is positive, we only need to consider the sign of the numerator, so:

$$= (1-r)(9600r^{4i+3}+4800^{3i+1}(1+r)+1200r^{2i+1})$$

$$+ 24r^i(1-\frac{10}{3}r^2+3r^i-10r^{i+3}). \quad (7)$$

According to the fact that $r > 1$, with a precise look at (7) we get

$$(9600r^{4i+3}+4800^{3i+1}(1+r)+1200r^{2i+1}) > 0$$

and we know $(1-r) < 0$. So,

$$(1-r)(9600r^{4i+3}+4800^{3i+1}(1+r)+1200r^{2i+1}) < 0.$$

Also, we have $-\frac{10}{3}r^2 < -\frac{10}{3}$, and $3r^i < 10r^{i+3}$. Then,

$$24r^i(1-\frac{10}{3}r^2+3r^i-10r^{i+3}) < 0.$$

Hence, (7) is always least equal to zero and this completes the proof.

Author Index