Proceedings of the 28th Canadian Conference on Computational Geometry

August 3-5, 2016
Simon Fraser University
Vancouver, British Columbia
Canada

Sponsored by

FIELDS

ELSEVIER

SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD
Computing Science

Contents

Square Formation by Asynchronous Oblivious Robots

Marcello Mamino* Giovanni Viglietta†

Abstract

A fundamental problem in Distributed Computing is the Pattern Formation problem, where some independent mobile entities, called robots, have to rearrange themselves in such a way as to form a given figure from every possible (non-degenerate) initial configuration.

In the present paper, we consider robots that operate in the Euclidean plane and are dimensionless, anonymous, oblivious, silent, asynchronous, disoriented, non-chiral, and non-rigid. For this very elementary type of robots, the feasibility of the Pattern Formation problem has been settled, either in the positive or in the negative, for every possible pattern, except for one case: the Square Formation problem by a team of four robots.

Here we solve this last case by giving a Square Formation algorithm and proving its correctness. Our contribution represents the concluding chapter in a long thread of research. Our results imply that in the context of the Pattern Formation problem for mobile robots, features such as synchronicity, chirality, and rigidity are computationally irrelevant.

1 Introduction and Background

Consider a finite set of independent computational entities, called *robots*, that live and operate in the Euclidean plane and are capable of observing each other's positions and moving to other locations, through so-called *Look-Compute-Move cycles*. A fundamental motion planning question in Distributed Computing is which *patterns* can be formed by such robots, regardless of their initial positions. This is known as the *Pattern Formation problem*, and has been extensively studied under different robot models (see the monography [5]).

In this paper we focus on a very weak type of robots, which are modeled as geometric points (*dimensionless*), are all indistinguishable from each other (*anonymous*), and execute the same deterministic algorithm. Moreover, they retain no memory of past events and observations (*oblivious*), they cannot communicate explicitly (*silent*), they have no common notion of time (*asynchronous*), of a North direction (*disoriented*), of a clockwise direction (*non-chiral*), and they may be unpredictably stopped during each cycle before reaching their intended destination (*non-rigid movements*). This robot model is often called *ASYNCH*.

Note that if n such robots initially form a regular n-gon and their local coordinate systems are oriented symmetrically, then they all have the same "view" of the world. Now, if they are all activated synchronously, they are bound to make symmetric moves forever, implying that they will always form a regular n-gon, or perhaps collide in the center. As the pattern has to be formed from every possible initial configuration, the Pattern Formation problem is unsolvable if the pattern is not a regular polygon or a point. Clearly, the existence of a general algorithm that will indeed make the robots form a regular polygon or a point from any initial configuration is not obvious and has been the object of intensive research by several authors.

In the case of a point, the Pattern Formation problem has been eventually settled in [1], where an algorithm is presented that always makes $n \neq 2$ ASYNCH robots gather in a point (if $n = 2$, the problem is unsolvable).

In the case of a regular polygon, there is a long history of algorithms that solve the Pattern Formation problem under increasingly weaker robot models. We start from the semi-synchronous model, *SSYNCH*, in which we assume the existence of a global "clock" that discretizes time. In each time unit, some robots (chosen by an external "adversary") perform a complete Look-Compute-Move cycle synchronously, while the other robots remain inactive. In [10] it is shown that SSYNCH robots can always form a regular polygon, provided that they have the ability to remember their past observations (hence they are not oblivious). In [2] the obliviousness of the robots is restored, but the algorithm proposed only makes the robots *converge* to a regular polygon, perhaps without ever forming one. These results were improved in [3], where it is shown how $n \neq 4$ SSYNCH robots, without additional requirements, can form a regular polygon. The case of a square, $n = 4$, was solved separately in [4] with an ad-hoc algorithm.

For ASYNCH robots, a simple solution was given in [8], under the assumption that the local coordinate systems of all robots have the same orientation. This result was improved in [9], where it is only assumed that the local coordinate systems are all right-handed (*chirality*), but may be rotated arbitrarily. In [6], an algorithm is given for $n \neq 4$ ASYNCH robots with no assumptions on their local coordinate systems, but allowing them to move along circular arcs, as well as straight line seg-

*Institut für Algebra, TU Dresden, 01062 Dresden, Germany, `marcello.mamino@tu-dresden.de`

†University of Ottawa, Canada, `gvigliet@uottawa.ca`

ments. A solution for $n \neq 4$ ASYNCH robots with no extra assumptions was finally given in [7]. The general algorithm lets only a few robots move at a time, so that the others will provide a stable "reference frame" for them. The case $n = 4$ is left unsolved in [7], essentially because four robots are too few to implement this strategy, yet enough to make ad-hoc solutions elusive.

In the following sections, we formalize the *Square Formation* problem for $n = 4$ ASYNCH robots, we give an algorithm for it, and we prove its correctness, thus completing the characterization of the patterns that are formable by ASYNCH robots from every initial configuration. Since the proof of non-formability of asymmetric patterns that we outlined above holds even for fully synchronous robots with chirality and rigid movements (i.e., movements that cannot be unpredictably stopped by an adversary), all these features turn out to be computationally irrelevant with respect to the Pattern Formation problem.

2 Model Specification

Let $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$ be a set of *robots*, each of which is thought of as a computational entity occupying a point in the plane $\mathbb{R}^2$ and having its own local Cartesian coordinate system. Each robot's coordinate system is always centered at the robot's location, and different robots' coordinate systems may have different orientation, handedness, and unit of length.

Each robot cyclically goes through three phases: *Look*, *Compute*, and *Move*. In a Look phase, the robot takes an instantaneous "snapshot" of all the robots' locations and it expresses them as points in $\mathbb{R}^2$ within its own local coordinate system. In the next Compute phase, these four points are fed, in any order, to an *algorithm* $\mathcal{A}$, which outputs a *destination* point p, again expressed in the robot's coordinate system. The algorithm $\mathcal{A}$ is the same for all robots, and it can only compute algebraic functions of the input points (for our purposes, we will only need arithmetic functions and square roots). In the next Move phase, the robot moves toward p along a straight line. Note that, even though the robots are indistinguishable, each robot can identify itself in the snapshots it takes, because it is always located at $(0, 0)$.

When a Move phase ends, the Look phase of the next cycle starts. We may assume that the Look and Compute phases of a robot are executed together and instantaneously at each cycle, but the Move phase's duration may vary, although it must be finite. The duration of each Move phase of each cycle of each robot is decided arbitrarily by an external "adversary", called *scheduler*. As a consequence, a robot may perform a Look while some other robots are in the middle of a movement, and there is no way to tell it from the snapshot. The scheduler also arbitrarily sets the speed of each robot at each moment of each Move phase; the velocity vector must always be directed toward the current destination point, or be the null vector. In particular, a robot may actually start moving a long time after the last Look-Compute phase, when the snapshot it has taken and the destination it has computed are already "obsolete".

The scheduler can also decide to end a robot's Move phase before it reaches its intended destination. The only constraint is that it cannot do so before the robot has moved by at least δ during that phase, where δ is a fixed positive distance (in absolute units), not known to the robots. This is to guarantee that if a robot keeps computing the same destination point (in absolute coordinates), it reaches it in finitely many cycles.

An initial configuration of the robots is *non-degenerate* if no two robots are located in the same point, and no robot is moving (formally, each robot's initial destination point coincides with the robot's initial location, and all robots are in a Move phase initially).

The Square Formation problem asks for a specific algorithm $\mathcal{A}$, whose input is a quadruplet of points and the output is a single destination point, such that, if the four robots of $\mathcal{R}$ execute $\mathcal{A}$ in all their Compute phases, starting from any non-degenerate initial configuration, and regardless of the scheduler's choices and of the value of δ, they always end up forming a square in finite time. Once a square is formed, the robots have to maintain their positions forever. (Recall that the input quadruplet always contains $(0, 0)$ as the executing robot's location, and the value of δ cannot be accessed by $\mathcal{A}$.)

3 Preliminary Constructions and Definitions

The following geometric construction will be useful in our Square Formation algorithm. Let $r_1r_2r_3r_4$ be a strictly convex quadrilateral whose diagonals r_1r_3 and r_2r_4 are not orthogonal. Let q be the unique point such that $r_1q = r_2r_4$, the lines r_1q and r_2r_4 are orthogonal, and the ray emanating from r_1 and passing through q intersects the line r_2r_4. Let ℓ_3 be the line through r_3 and q, and let ℓ_1 be the line through r_1 parallel to ℓ_3.

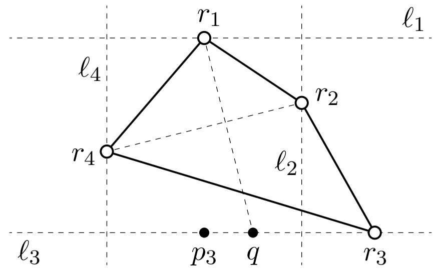


Figure 1: Constructing the guidelines and the targets

(Since r_1r_3 and r_2r_4 are not orthogonal, ℓ_3 is well defined and is distinct from ℓ_1.) Let ℓ_2 be the line through r_2 orthogonal to ℓ_1, and let ℓ_4 be the line through r_4 parallel to ℓ_2. By construction, these four lines intersect at four points that are vertices of a square Q. In turn, the midpoints of the edges of Q form a second square Q', called the *target square.*

Ideally, if the robots of $\mathcal{R}$ are located at r_1, r_2, r_3, r_4 (with abuse of notation, we identify each robot with its location), our Square Formation algorithm will attempt to make the team move to the vertices of Q'. More precisely, the *target* of r_i is the vertex p_i of Q' that lies on ℓ_i. The line ℓ_i is called the *guideline* of r_i, and the segment r_ip_i is the *pathway* of r_i. If r_i lies in the interior of an edge of Q, then r_i is said to be *internal*; otherwise, r_i is *external.* If two robots have parallel guidelines, they are said to be *opposite* to each other.

In Section 5 we will prove that if we permute the labels of the vertices of the above quadrilateral arbitrarily, as long as the indices follow a clockwise or a counterclockwise order, and we repeat the same construction, the resulting target square will be the same. Hence, if several robots compute the above construction at the same time within their local coordinate systems, they necessarily obtain the same target square. Note also that the target square remains unaltered as the robots move along their pathways, as long as the quadrilateral stays convex, and its diagonals stay non-orthogonal.

Let c be the center of the target square. The "signed distance" between r_i and its target can be computed as $(r_i - c) \times (r'_i - c)$, where $\times$ denotes the cross product in $\mathbb{R}^2$, defined as $(x_1, y_1) \times (x_2, y_2) = x_1y_2 - x_2y_1$. If this number is 0, then r_i is said to be *finished.* If the product of the signed distances of two robots is not negative (respectively, not positive), the two robots are said to be *concordant* (respectively, *discordant*). Intuitively, they are concordant if they move around c in the same "direction" (i.e., clockwise or counterclockwise) as they go toward their targets.

Let the guidelines of two discordant robots r_i and r_j intersect in a point g. If p_i lies on the segment r_ig and p_j lies on the segment r_jg, then r_i and r_j are said to be *convergent*; otherwise, they are *divergent* (if both r_i and r_j are finished, they are both convergent and divergent).

If the pathway of r_i intersects the segment r_jr_k in v, then r_i is said to be *blocked* at v. If the pathway of r_i intersects an extension of the segment r_jr_k in v, then r_i is said to be *hindered* at v (see Figure 4).

Let us give one last definition. A *thin hexagon* is a hexagon $H = h_1h_2h_3h_4h_5h_6$ such that $h_1h_2 = h_3h_4 = h_4h_5 = h_6h_1 = h_1h_4/4$, the angles at h_1 and h_4 are $50°$, and all other angles are equal. h_1 and h_4 are the *extremes* of H, and the segment h_1h_4 is the *main diagonal.* The other four vertices are called *beacons*, and the midpoint of two adjacent beacons is a *haven.*

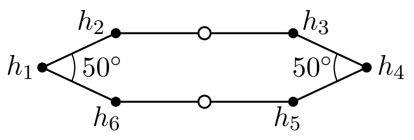


Figure 2: Thin hexagon (empty dots denote havens)

4 The Square Formation Algorithm

The Square Formation algorithm will identify the current configuration's *class* among the ones listed below, and it will execute a different procedure based on the class. If the configuration belongs to several classes, the relevant one is the one that appears first in the list.

So, when a new class is defined, it is assumed that none of the definitions of the previous classes are satisfied. In particular, after quadrilaterals with orthogonal diagonals and non-convex quadrilaterals have been ruled out, the target square described in Section 3 will be well defined, as well as each robot's guideline, etc.

Again, we identify each robot r_i with its position, and a class' definition is fulfilled when there is a permutation of the indices that satisfies the corresponding condition.

We will use expressions of the form, "robot r_i moves to point d_i; robot r_j moves to point d_j" without specifying which robot is actually running the algorithm, as if there was a global coordinator overseeing the execution. However, since the robots are anonymous and independent, we will always move symmetric robots in symmetric ways, so as to comply with the specification of the robot model given in Section 2.

Borrowing from [6, 7], we will make use of *cautious moves*, whose purpose is, roughly speaking, to prevent situations in which a robot is still in the middle of a movement when a configuration class change occurs. This is done by identifying a finite number of *critical points* and making each moving robot stop at the first critical point it encounters. In this section, we will only generically say, "robot r_i cautiously moves toward d_i", leaving out the tricky details of how critical points are chosen and maintained. A complete analysis of every case will be carried out in Section 5, along with the proof of correctness of the Square Formation algorithm.

Configuration 1: Orthogonal
Definition. The segments r_1r_3 and r_2r_4 are orthogonal and intersect in a point c (possibly an endpoint).
Execution. Each of r_1, r_2, r_3, r_4 moves away from c, to a point at distance $\max\{r_1c, r_2c, r_3c, r_4c\}$ from it.

Configuration 2: Thin Hexagon
Definition. The thin hexagon H with main diagonal r_1r_2 contains also r_3 and r_4, but not on two adjacent beacons.
Execution. If both r_3 and r_4 are on the main diagonal,

they move orthogonally to it, remaining within H. If r_3 is on the main diagonal and r_4 is not, r_3 moves to the opposite side of the main diagonal, orthogonally to it, remaining within H. If r_3 and r_4 are on different sides of the main diagonal, they move to the haven on their respective side; the closest moves first while the other one waits (in case of a tie, they both move). If r_3 and r_4 are on the same side of the main diagonal, they move to the two adjacent beacons on that side, minimizing the total distance traveled.

Configuration 3: Non-Convex
Definition. The triangle $r_1r_2r_3$ contains r_4.
Execution. r_4 moves to the foot of an altitude of $r_1r_2r_3$ that lies in the interior of an edge of $r_1r_2r_3$.

Configuration 4: Pinwheel
Definition. r_1, r_2, r_3, r_4 are all concordant.
Execution. Let r_1 be opposite to r_3 and assume without loss of generality that $r_1r_3 < r_2r_4$ (if $r_1r_3 = r_2r_4$, the configuration is Orthogonal). Suppose that r_1 and r_3 are both finished. If the thin hexagon H with main diagonal r_2r_4 contains r_1 (and not r_3), assume without loss of generality that $r_2r_1 < r_2r_3$, and let r_2 cautiously move toward its target. If neither r_1 nor r_3 is in H, both r_2 and r_4 cautiously move toward their targets. Suppose now that r_1 and r_3 are not both finished. if r_1 is blocked at v, it cautiously moves toward v. If neither r_1 nor r_3 is blocked and r_1 is hindered at v, r_1 cautiously moves toward v. If neither r_1 nor r_3 is blocked or hindered, r_1 and r_3 cautiously move toward their targets. (These rules are exhaustive due to Lemma 4 below.)

Configuration 5: Scissors
Definition. r_1, r_2 are divergent; r_3, r_4 are divergent.
Execution. If exactly one robot is external, it cautiously moves toward its target. If all the internal robots are finished, all the external robots cautiously move toward their targets. In all other cases, all the internal robots cautiously move toward their targets.

Configuration 6: Flowing
Definition. r_1, r_2 are divergent; r_3, r_4 are convergent.
Execution. If two opposite robots are finished, the non-finished one that is closest to its target cautiously moves toward it (there cannot be a tie, or the configuration would be Orthogonal). Otherwise, if exactly one of r_1 and r_2 is finished, the robot opposite to it cautiously moves toward its target. Otherwise, both r_3 and r_4 cautiously move toward their targets.

Configuration 7: One Discordant
Definition. r_1, r_2, r_3 are concordant; r_4 is discordant.
Execution. r_4 cautiously moves toward its target.

5 Correctness of the Algorithm

As noted in Section 3, the target square is well defined, no matter how each robot computes it.

Lemma 1 *Given a strictly convex quadrilateral with non-orthogonal diagonals, regardless of how labels r_1, r_2, r_3, r_4 are assigned to its vertices following a clockwise or a counterclockwise order, the construction in Section 3 yields the same guidelines and target square.*

Proof. The construction does not change if we invert the labels of r_2 and r_4, hence we may assume that the indices are arranged in clockwise order. Now it is enough to prove that the construction does not change if we shift the labels clockwise by one position. So, let q' be the unique point such that $r_2q' = r_1r_3$, the lines r_2q' and r_1r_3 are orthogonal, and the ray emanating from r_2 and passing through q' intersects the line r_1r_3. By construction, the triangle r_2r_4q' is a copy of r_1r_3q rotated by 90°, which means that the line r_4q' is orthogonal to ℓ_3, and hence coincident with ℓ_4. It follows that all the new guidelines are the same as in the original construction, and so is the target square. □

Lemma 2 *Given a strictly convex quadrilateral with non-orthogonal diagonals, if its vertices are labeled r_1, r_2, r_3, r_4 in clockwise order, then their targets p_1, p_2, p_3, p_4 also appear in clockwise order, and vice versa.*

Proof. It suffices to prove that if p_1, p_2, p_3 are in clockwise order, then so are r_1, r_2, r_3. Referring to Figure 3, if r_1, r_2, r_3 are in counterclockwise order, then r_2 is located to the right of the line r_1r_3. Therefore, q must be to the left of ℓ_2, contradicting the fact that q must lie on ℓ_4, which in turn is located to the right of ℓ_2. □

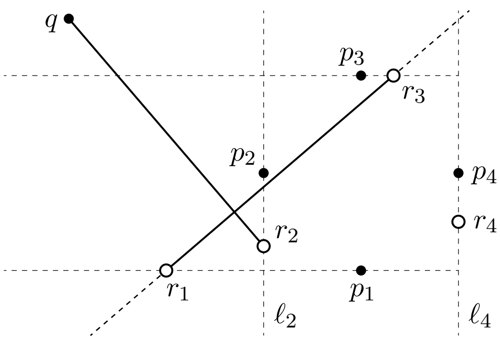


Figure 3: r_1, r_2, r_3 cannot be in counterclockwise order

The following is an easy consequence of Lemma 2.

Corollary 3 *No two robots have intersecting pathways. If two robots are external, they are either concordant or convergent. If three robots are external, they are concordant.* □

Lemma 4 *In a Pinwheel configuration, two opposite robots cannot be both blocked. Moreover, if $r_1r_3 < r_2r_4$, robots r_1 and r_3 cannot be both hindered.*

Proof. Referring to Figure 4, if r_2 is blocked, it means that the segment r_1r_3 crosses ℓ_2 above p_2. Hence r_1r_3 must cross ℓ_4 further above, and therefore r_4 cannot be blocked, because its pathway lies below p_4.

Suppose now that r_1 and r_3 are both hindered, as in Figure 4. Because the line r_1r_4 intersects the pathway of r_3, it is immediate to see that r_1 must be external and r_4 must be internal. Symmetrically, r_3 must be external and r_2 must be internal. This means that $r_1r_3 > r_2r_4$, contradicting our assumption. □

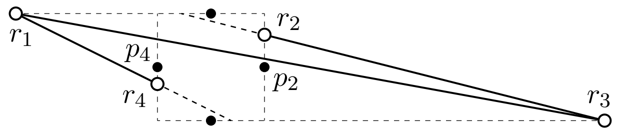


Figure 4: r_2 is blocked; r_1 and r_3 are hindered

Lemma 5 *In a Pinwheel configuration where r_1 and r_3 are opposite, both are finished, and neither of them is in the thin hexagon H with main diagonal r_2r_4, there is a choice of critical points that lets r_2 and r_4 reach their targets in finite time as they perform a cautious move.*

Proof. Our critical points will prevent r_2 and r_4 from forming an Orthogonal, Thin Hexagon, or Non-Convex configuration before reaching their targets.

An Orthogonal configuration cannot be formed before r_2 and r_4 reach their targets, because r_1r_3 is parallel to their guidelines.

A Thin Hexagon configuration cannot be formed, either. Note that r_1 and r_3 must be on opposite sides of H, due to Lemma 2. As Figure 5 suggests, it is straightforward to verify that if r_1 and r_3 are both outside H, the sum of the distances of r_1 and r_3 from the main diagonal is greater than the height of H (this is true because each short edge of a thin hexagon is a quarter of the main diagonal). As r_2 and r_4 move toward their targets, the main diagonal becomes shorter and the sum of the distances of r_1 and r_3 from the main diagonal grows. In particular, this sum is greater than the new height of H. It follows that, no matter how r_2 and r_4 move, a Thin Hexagon configuration will never be formed.

Observe that both r_2 and r_4 could be hindered, say at v and v', respectively (as r_1 and r_3 in Figure 4). It suffices to set their critical points halfway to v and v' to guarantee that a Non-Convex configuration will never be formed. We call the segments r_2v and r_4v' *safe zones.* It is easy ot see that, as r_2 and r_4 move (and recompute a new v and v' pair), their safe zones get longer, giving them even more leeway, until they are not hindered any more, and can safely reach their targets. □

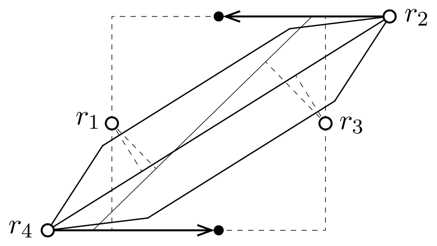


Figure 5: As r_2 and r_4 move, no Thin Hexagon is formed

Lemma 6 *In a Thin Hexagon configuration that is also a Scissors one, both extremes must be external.*

Proof. By Corollary 3, at most two robots can be external. Clearly, each external robot must necessarily be an extreme of the thin hexagon. Suppose for a contradiction that at most one robot is external. Let r_1 be either an external robot or an extreme of the thin hexagon (in case there are no external robots). The other extreme must be the farthest robot, hence either r_3 or r_4 (assuming that r_1 and r_2 are divergent). Now it is straightforward to verify that both angles $\angle r_1r_3r_4$ and $\angle r_1r_4r_3$ are greater than $\arctan(1/2) > 25°$, and hence the thin hexagon cannot contain both r_3 and r_4. □

Lemma 7 *If a configuration is not Thin Hexagon, r_3 and r_4 are convergent, and r_1 and r_2 do not move while r_3 and r_4 move toward their targets, the configuration never becomes Thin Hexagon.*

Proof. By Lemma 2, r_1 and r_2 are on the same side of the line r_3r_4, as in Figure 6. Let H be the thin hexagon with main diagonal r_3r_4. By assumption, either r_1 and r_2 occupy two adjacent beacons of H or one of them, say r_1, is not in H. In both cases, as soon as H starts moving together with r_3 and r_4, r_1 is guaranteed to remain strictly outside of H. Indeed, the position of H at each time can be obtained from its initial position by a composition of two types of transformations: shrinkage about an extreme and rotation about an extreme by less than 90° in the direction opposite to r_1. Both these operations cause r_1 to stay out of H if it is already out and to get out of H if it is initially on a beacon. □

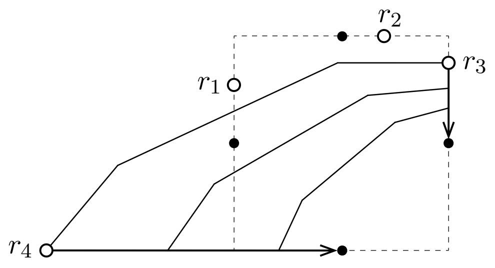


Figure 6: As r_3 and r_4 move, no Thin Hexagon is formed

Theorem 8 *The algorithm of Section 4 solves the Square Formation problem in the ASYNCH model.*

Proof. We prove that the rules given in Section 4 are well defined, exhaustive, and make the four robots form a square in a finite amount of time. The general idea is that each configuration can only remain in the same class, or transition to a class with lower index, with one exception: a Thin Hexagon can become a Scissors configuration, and this case will be examined separately. We also have to verify that no robot is moving when the team's behavior changes, in order to prevent inconsistencies arising from robots believing to be in different classes, due to asynchronicity. The cautious move technique serves this purpose, but we have to show that suitable critical points exist. When only one robot is tasked with moving, it is sufficient to identify the first location on its path that may cause other robots to start moving. So, no discussion will be needed in this case.

Verifying the above for Configurations 1–3 is trivial, so let us assume that the robots' initial configuration is none of those, and in particular that a target square is well defined (cf. Lemma 1). Now, as the robots move toward their targets, the target square remains unaltered, and collisions are impossible due to Corollary 3.

Let the configuration be in the Pinwheel class. Lemma 4 implies that the rules are unambiguous and a robot always moves, eventually forming a lower-index configuration or reaching a target. If $r_1r_3 < r_2r_4$, the main diagonal of a possible thin hexagon H must be r_2r_4, so it is easy for r_1 and r_3 to stop as soon as they reach the boundary of H, since H remains still as they move. When r_1 and r_3 are finished, the cautious move of r_2 and r_4 succeeds due to Lemma 5.

Let the configuration be in the Scissors class. By Lemma 6, if there are fewer than two external robots, no Thin Hexagon can ever be formed, and if there are two external robots (not more, by Corollary 3), there is only one candidate thin hexagon H having these two robots as extremes. If the internal robots move, they can set their critical points on the boundary of H. If the external robots move, they must be convergent (or they would be concordant by Corollary 3, and the configuration would be Pinwheel), hence no Thin Hexagon can be formed, due to Lemma 7. Also, no moving robot can be blocked or hindered at any point. When two opposite robots are finished, the configuration may transition to Pinwheel while other robots are still moving, but this is fine because it does not cause a change in behavior.

Let the configuration be in the Flowing class. If r_3 and r_4 move together, no critical points originating from thin hexagons have to be set, due to Lemma 7, and the other critical points are easy to spot. If two opposite robots are finished, moving only the non-finished robot closest to its target prevents the formation of an Orthogonal configuration. A transition to Pinwheel or Scissors may occur, but only when no robots are moving.

If the configuration class is none of the above, there must be a unique discordant robot, which turns the configuration into a Pinwheel one upon reaching its target.

As we mentioned, the only anomaly in this process is the transition from a Thin Hexagon to a Scissors configuration, which occurs when two robots reach adjacent beacons. According to the rules for the Scissors case, the two robots on the beacons (which are internal) move to their targets. Since they move away from the main diagonal, they can form no Thin Hexagon. Afterwards, the two external robots (which are convergent) move to their targets without forming a Thin Hexagon, by Lemma 7. No Orthogonal or Non-Convex configurations can be formed during these procedures, either. □

References

[1] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

[2] X. Défago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1–3):97–112, 2008.

[3] Y. Dieudonné and F. Petit. Swing words to make circle formation quiescent. *14th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 166–179, 2007.

[4] Y. Dieudonné and F. Petit. Squaring the circle with weak mobile robots. *19th International Symposium on Algorithms and Computation (ISAAC)*, 354–365, 2008.

[5] P. Flocchini, G. Prencipe, and N. Santoro. *Distributed computing by oblivious mobile robots.* Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool, 2012.

[6] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: solving the uniform circle formation problem. *18th International Conference on Principles of Distributed Systems (OPODIS)*, 217–232, 2014.

[7] P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: solving the uniform circle formation problem. arXiv:1407.5917 [cs.DC], 2015.

[8] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous oblivious robots. *Theoretical Computer Science*, 407(1–3):412–447, 2008.

[9] N. Fujinaga, Y. Yamauchi, S. Kijima, and M. Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. *26th International Symposium on Distributed Computing (DISC)*, 312–325, 2012.

[10] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

Finding Points in General Position

Vincent Froese[*] Iyad Kanj[†] André Nichterlein[†] Rolf Niedermeier[†]

Abstract

We study the General Position Subset Selection problem: Given a set of points in the plane, find a maximum-cardinality subset of points in general position. We prove that General Position Subset Selection is NP-hard, APX-hard, and present several fixed-parameter tractability results for the problem.

1 Introduction

For a set $P = \{p_1, \ldots, p_n\}$ of n points in the plane, a subset $S \subseteq P$ is in *general position* if no three points in S are *collinear* (that is, lie on the same line). A frequent assumption for point set problems in computational geometry is that the given point set is in general position. Nevertheless, the problem of computing a maximum-cardinality subset of points in general position from a given set of points has received little attention from the computational complexity perspective, although not from the combinatorial geometry perspective. In particular, to the best of our knowledge, the classical complexity of the aforementioned problem until now was unresolved. Formally, the decision version of the problem is as follows:

General Position Subset Selection
Input: A set P of points in the plane and $k \in \mathbb{N}$.
Question: Is there a subset $S \subseteq P$ in general position of cardinality at least k?

A well-known special case of General Position Subset Selection, referred to as the No-Three-In-Line problem, asks to place a maximum number of points in general position on an $n \times n$-grid. Since at most two points can be placed on any grid-line, the maximum number of points in general position that can be placed on an $n \times n$-grid is at most $2n$. Indeed, only for small n it is known that $2n$ points can always be placed on the $n \times n$-grid. Erdős [19] observed that, for sufficiently large n, one can place $(1-\epsilon)n$ points in general position on the $n \times n$-grid, for any $\epsilon > 0$. This lower bound was improved by Hall et al. [13] to $(\frac{3}{2}-\epsilon)n$. It was conjectured by Guy and Kelly [12] that, for sufficiently large n, one can place more than $\frac{\pi}{\sqrt{3}}n$ many points in general position on the $n \times n$-grid. This conjecture remains unresolved, hinting at the challenging combinatorial nature of No-Three-In-Line, and hence of General Position Subset Selection as well.

A problem closely related to General Position Subset Selection is Point Line Cover: Given a point set in the plane, find a minimum-cardinality set of lines, the size of which is called the *line cover number*, that cover all points. Interestingly, the size of a maximum subset in general position is related to the line cover number (see Observation 1). While Point Line Cover has been intensively studied, we aim to fill the existing gap for General Position Subset Selection by providing both computational hardness and fixed-parameter tractability results for the problem. In doing so, we particularly consider the parameters solution size k (size of the sought subset in general position) and its dual $h := n - k$, and investigate their impact on the computational complexity of General Position Subset Selection.

Related Work Payne and Wood [18] provide lower bounds on the size of a point set in general position, a question originally studied by Erdős [6]. In his Master's thesis, Cao [3] gives a problem kernel of $O(k^4)$ points for General Position Subset Selection (there called Non-Collinear Packing problem) and a simple greedy $O(\sqrt{\text{opt}})$-factor approximation algorithm for the maximization version. He also presents an Integer Linear Program formulation and shows that it is in fact the dual of an Integer Linear Program formulation for Point Line Cover. As to results for the much more studied Point Line Cover, we refer to the work of Kratsch et al. [15] and the work cited therein.

Our Contributions We show that General Position Subset Selection is NP-hard and APX-hard. Our main algorithmic results, however, concern the power of polynomial-time data reduction for General Position Subset Selection: We give an $O(k^3)$-point problem kernel and an $O(h^2)$-point problem kernel, and show that the latter is asymptotically optimal under a reasonable complexity-theoretic assumption. Table 1 summarizes our results. Due to the lack of space, some

[*]Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany, {vincent.froese, andre.nichterlein, rolf.niedermeier}@tu-berlin.de

[†]School of Computing, DePaul University, Chicago, USA, ikanj@cs.depaul.edu. Supported by the DFG project DAPA (NI 369/12) during a Mercator fellowship when staying at TU Berlin.

Table 1: Overview of the results we obtain for General Position Subset Selection, where n is the number of input points, k is the parameter size of the sought subset in general position, $h = n - k$ is the dual parameter, and ℓ is the line cover number.

Hardness	Tractability
NP-hard (Theorem 2)	$(15k^3)$-point kernel (running time $O(n^2 \log n)$) (Theorem 4)
APX-hard (Theorem 2)	$O(n^2 \log n + 41^k \cdot k^{2k})$-time solvable (Corollary 5)
no $2^{o(n)} \cdot n^{O(1)}$-time algorithm[a] (Theorem 2)	$O(2.08^h + n^3)$-time solvable (Proposition 8)
no $O(h^{2-\epsilon})$-point kernel[b] (Theorem 13)	$(2h^2 + h)$-point kernel (running time $O(n^2)$) (Theorem 10)
	$(120\ell^3)$-point kernel (running time $O(n^2 \log n)$) (Corollary 7)
	$O(n^2 \log n + 41^{2\ell} \cdot (2\ell)^{4\ell})$-time solvable (Corollary 7)

[a]Unless the Exponential Time Hypothesis fails.
[b]Unless coNP $\subseteq$ NP/poly.

details are omitted and can be found in a full version[1].

2 Preliminaries

Geometry All coordinates of points are assumed to be represented by rational numbers. The *collinearity* of a set of points P is the maximum number of points in P that lie on the same line. A *blocker* for two points p, q is a point on the open line segment pq.

Graphs Let $G = (V(G), E(G))$ be an undirected graph. We write $|G|$ for $|V(G)| + |E(G)|$. A vertex $u \in V(G)$ is a *neighbor* of (or is *adjacent* to) a vertex $v \in V(G)$ if $\{u, v\} \in E(G)$. The *degree* of a vertex v is the number of its neighbors.

An *independent set* of a graph G is a set of vertices such that no two vertices in this set are adjacent. The NP-complete Independent Set problem is: Given a graph G and $k \in \mathbb{N}$, decide whether G has an independent set of cardinality k.

Parameterized Complexity A *parameterized problem* is a set of instances of the form $(\mathcal{I}, k)$, where $\mathcal{I} \in \Sigma^*$ for a finite alphabet set Σ, and $k \in \mathbb{N}$ is the *parameter*. A parameterized problem Q is *fixed-parameter tractable*, shortly FPT, if there exists an algorithm that on input $(\mathcal{I}, k)$ decides whether $(\mathcal{I}, k)$ is a yes-instance of Q in $f(k)|\mathcal{I}|^{O(1)}$ time, where f is a computable function independent of $|\mathcal{I}|$. A parameterized problem Q is *kernelizable* if there exists a polynomial-time algorithm that maps an instance $(\mathcal{I}, k)$ of Q to another instance $(\mathcal{I}', k')$ of Q such that:
(1) $|\mathcal{I}'| \le \lambda(k)$ for some computable function λ,
(2) $k' \le \lambda(k)$, and
(3) $(\mathcal{I}, k)$ is a yes-instance of Q if and only if $(\mathcal{I}', k')$ is a yes-instance of Q.

[1]Available at arxiv.org/abs/1508.01097.

The instance $(\mathcal{I}', k')$ is called a *problem kernel* of $(\mathcal{I}, k)$. A parameterized problem is FPT if and only if it is kernelizable [2]. A general account on applying methods from parameterized complexity analysis to problems from computational geometry is due to Giannopoulos et al. [9].

Exponential Time Hypothesis The Exponential Time Hypothesis (ETH) [14] states that 3-SAT cannot be solved in $2^{o(n)} \cdot n^{O(1)}$ time, where n is the number of variables in the input formula.

3 Hardness Results

In this section, we prove that General Position Subset Selection is NP-hard, APX-hard, and presumably not solvable in subexponential time. Our hardness results follow from a transformation (mapping arbitrary graphs to point sets) that is based on a construction due to Ghosh and Roy [8, Section 5], which they used to prove the NP-hardness of the Independent Set problem on so-called *point visibility graphs*. This transformation, henceforth called Φ, allows us to obtain the above-mentioned hardness results (using reductions from NP-hard restrictions of Independent Set to General Position Subset Selection). Moreover, in Section 4.2, we will use Φ to give a reduction from Vertex Cover to General Position Subset Selection in order to obtain problem kernel size lower bounds with respect to the dual parameter (see Theorem 11 and Theorem 13). We start by formally defining some properties that are required for the output point set of the transformation. As a next step, we prove that such a point set can be realized in polynomial time.

Let G be a graph with vertex set $V(G) = \{v_1, \ldots, v_n\}$. Let $C = \{p_1, \ldots, p_n\}$ be a set of points that are in strictly convex position (that is, the points in C are vertices of a convex polygon), where $p_i \in C$ corresponds to v_i, $i = 1, \ldots, n$. For each edge $e = \{v_i, v_j\} \in E(G)$, we place a *blocker* b_e on the line segment p_ip_j such that the following three conditions are satisfied:

(I) For any edge $e \in E(G)$ and for any two points $p_i, p_j \in C$, if b_e, p_i, p_j are collinear, then p_i, p_j are the points in C corresponding to the endpoints of edge e.

(II) Any two distinct blockers $b_e, b_{e'}$ are not collinear with any point $p_i \in C$.

(III) The set $B := \{b_e \mid e \in E(G)\}$ of blockers is in general position.

Proposition 1 *There is a polynomial-time transformation Φ mapping arbitrary graphs to point sets that satisfy Conditions (I) to (III). Moreover, no four points in the point set $C \cup B$ produced by Φ are collinear.*

Proof. Given a graph G, let $n = |V(G)|$ and let $C = \{p_1, \dots, p_n\}$ be a set of rational points that are in a strictly convex position; for instance, let $p_j := (\frac{2j}{1+j^2}, \frac{1-j^2}{1+j^2})$ for $j \in \{1, \dots, n\}$ be n rational points on the unit circle centered at the origin [20]. To choose the set B of blockers, suppose (inductively) that we have chosen a subset B' of blockers such that all blockers in B' are rational points and satisfy Conditions (I) to (III). Let $b_e \notin B'$ be a blocker corresponding to an edge $e = \{v_i, v_j\}$ in G. To determine the coordinates of b_e, we first mark the intersection points (if any) between the line segment p_ip_j and the lines formed by every pair of distinct blockers in B', every pair of distinct points in $C \setminus \{p_i, p_j\}$, and every pair consisting of a blocker in B' and a point in $C \setminus \{p_i, p_j\}$. We then choose b_e to be an interior point of p_ip_j with rational coordinates that is distinct from all marked points. To this end, let q be the first marked point on the segment p_ip_j (starting from p_i), and let b_e be the midpoint of p_iq. This point is rational since it is the midpoint of rational points. It is easy to see that $C \cup B$ can be constructed in polynomial time and that all points in $C \cup B$ are rational and satisfy Conditions (I) to (III). Moreover, it easily follows from the construction of $C \cup B$ that it satisfies Conditions (I) to (III) and that no four points in $C \cup B$ are collinear. □

Using transformation Φ, we can prove (proof omitted) the following hardness results via reductions from (variants of) Independent Set.

Theorem 2 *The following are true:*

(a) General Position Subset Selection *is* NP-*complete.*

(b) Maximum General Position Subset Selection *is* APX-*hard.*

(c) Unless ETH fails, General Position Subset Selection *is not solvable in $2^{o(n)} \cdot n^{O(1)}$ time.*

We note that parts (a)–(c) above even hold for the restriction of General Position Subset Selection *to instances in which no four points are collinear.*

Currently, the best approximation result for Maximum General Position Subset Selection is due to Cao [3], who provided a simple greedy $\sqrt{\text{opt}}$-factor approximation algorithm. Therefore, a large gap remains between the proven upper and the lower bound on the approximation factor.

4 Fixed-Parameter Tractability

In this section, we prove several fixed-parameter tractability results for General Position Subset Selection. In Section 4.1 we develop cubic-size problem kernels with respect to the parameter size k of the sought subset in general position, and with respect to the line cover number ℓ. In Section 4.2, we show a quadratic-size problem kernel with respect to the *dual parameter* $h := n - k$, that is, the number of points whose deletion leaves a set of points in general position. Moreover, we prove that this problem kernel is essentially optimal, unless an unlikely collapse in the polynomial hierarchy occurs.

4.1 FPT Results for the Parameter Solution Size k

Let (P, k) be an instance of General Position Subset Selection, and let $n = |P|$. Cao [3] gave a problem kernel for General Position Subset Selection of size $O(k^4)$ based on the following idea. Suppose that there is a line L containing at least $\binom{k-2}{2} + 2$ points from P. For any subset $S' \subset P$ in general position with $|S'| = k - 2$, there can be at most $\binom{k-2}{2}$ points on L such that each is collinear with two points in S'. Hence, we can always find at least two points on L that together with the points in S' form a subset S in general position of cardinality k. Based on this idea, Cao [3] introduced the following data reduction rule:

Rule 1 ([3]) *Let (P, k) be an instance of* General Position Subset Selection*. If there is a line L that contains at least $\binom{k-2}{2} + 2$ points from P, then remove all the points on L and set $k := k - 2$.*

Cao showed that Rule 1 can be exhaustively applied in $O(n^3)$ time ([3, Lemma B.1.]), and he showed its correctness, that is, an instance (P', k') that is reduced with respect to Rule 1 is a yes-instance of General Position Subset Selection if and only if (P, k) is ([3, Theorem B.2.]). Using Rule 1, he gave a kernel for General Position Subset Selection of size $O(k^4)$ that is computable in $O(n^3)$ time ([3, Theorem B.3.]). We shall improve on Cao's result, both in terms of the kernel size and the running time of the kernelization algorithm. We start by showing how, using a result by Guibas et al. [11, Theorem 3.2], Rule 1 can be applied exhaustively in $O(n^2 \log n)$ time. Notably, the idea of reducing lines with many points (based on Guibas et al. [11]) also yields kernelization results for Point Line Cover [16].

Lemma 3 *Given an instance (P,k) of* General Position Subset Selection *where $|P|=n$, in $O(n^2\log n)$ time we can compute an equivalent instance (P',k') such that either (P',k') is a trivial yes-instance, or the collinearity of P' is at most $\binom{k'-2}{2}+1$.*

Proof. Let $\lambda=\binom{k-2}{2}+2$. We start by computing the set $\mathcal{L}$ of all lines that contain at least λ points from P. By a result of Guibas et al. [11, Theorem 3.2], this can be performed in $O(n^2\log(n/\lambda)/\lambda)$ time. We then iterate over each line $L\in\mathcal{L}$, checking whether L, at the current iteration, still contains at least λ points; if it does, we remove all points on L from P and decrement k by 2. For each line L, the running time of the preceding step is $O(\lambda)$, which is the time to check whether L contains at least λ points. Additionally, we might need to remove all points on L. If k reaches zero, we can return a trivial yes-instance (P',k') of General Position Subset Selection in constant time. Otherwise, after iterating over all lines in $\mathcal{L}$, by Rule 1, the resulting instance (P',k') is an equivalent instance to (P,k) satisfying that no line in P' contains λ points, and hence the collinearity of P' is at most $\binom{k'-2}{2}+1$. Overall, the above can be implemented in time $O((n^2\log(n/\lambda)/\lambda)\cdot\lambda)=O(n^2\log n)$. □

We move on to improving the size of the problem kernel. Payne and Wood [18, Theorem 2.3] proved a lower bound on the maximum cardinality of a subset in general position when an upper bound on the collinearity of the point set is known. We show next how to obtain a kernel for General Position Subset Selection of cubic size based on this result of Payne and Wood [18].

Theorem 4 General Position Subset Selection *admits a problem kernel containing at most $15k^3$ points that is computable in $O(n^2\log n)$ time.*

Proof. By Lemma 3, after $O(n^2\log n)$ preprocessing time, we can either return an equivalent yes-instance of (P,k) of constant size, or obtain an equivalent instance for which the collinearity of the point set is at most $\binom{k-2}{2}+1$. Therefore, without loss of generality, we can assume in what follows that the collinearity of P is at most $\lambda=\binom{k-2}{2}+1$.

Payne and Wood [18, Theorem 2.3] showed that any set of n points whose collinearity is at most λ contains a subset of points in general position of size at least $\alpha n/\sqrt{n\ln\lambda+\lambda^2}$, for some constant $\alpha\in\mathbb{R}$. A lower bound of $\alpha\geq\sqrt{6}/72$ can be computed based on Payne [17, Lemmas 4.1, 4.2, and Theorems 2.2, 2.3, 4.3]. Since $\lambda\leq\binom{k-2}{2}+1$, we can compute a value of n, as a function of k, above which we are guaranteed to have a subset in general position of cardinality at least k. We do this by solving for n in the inequality $\alpha n/\sqrt{n\ln\lambda+\lambda^2}\geq k$ after substituting λ with $\binom{k-2}{2}+1$ and α with $\sqrt{6}/72$. We obtain that if $n\geq 15k^3$, then the aforementioned inequality is satisfied for all $k\geq 29337$. The kernelization algorithm distinguishes the following three cases: First, if $k<29337$, then the algorithm decides the instance in $O(1)$ time, and returns an equivalent instance of $O(1)$ size. Second, if $k\geq 29337$ and $n\geq 15k^3$, then the algorithm returns a trivial yes-instance of constant size. Third, if none of the two above cases applies, then it returns the (preprocessed) instance (P,k) which satisfies $|P|\leq 15k^3$. □

We can derive the following result by a brute-force algorithm on the above problem kernel:

Corollary 5 General Position Subset Selection *can be solved in $O(n^2\log n+41^k\cdot k^{2k})$ time.*

Proof. Let (P,k) be an instance of General Position Subset Selection. By Theorem 4, after $O(n^2\log n)$ preprocessing time, we can assume that $|P|\leq 15k^3$. We enumerate every subset of size k in P, and for each such subset, we use the result of Guibas et al. [11, Theorem 3.2] to check in $O(k^2\log k)$ time whether the subset is in general position. If we find such a subset, then we answer positively; otherwise (no such subset exists), we answer negatively. The number of enumerated subsets is

$$\binom{|P|}{k}\leq\binom{15k^3}{k}\leq\frac{(15k^3)^k}{k!}$$
$$\leq\frac{(15k^3)^k}{(k/e)^k}=(15ek^3/k)^k\leq(40.78)^k k^{2k},$$

where e is the base of the natural logarithm and $k!\geq(k/e)^k$ follows from Stirling's formula. Putting everything together, we obtain an algorithm for General Position Subset Selection that runs in $O(n^2\log n+(40.78)^k\cdot k^{2k}\cdot k^2\log k)=O(n^2\log n+41^k\cdot k^{2k})$ time. □

Let 3-General Position Subset Selection denote the restriction of General Position Subset Selection to instances in which the point set contains no four collinear points. By Theorem 2, 3-General Position Subset Selection is NP-complete. Füredi [7, Theorem 1] showed that every set P of n points in which no four points are collinear contains a subset in general position of size $\Omega(\sqrt{n\log n})$. Based on Füredi's result and the idea in the proof of Theorem 4, we get:

Corollary 6 *3-*General Position Subset Selection *admits a problem kernel containing $O(k^2/\log k)$ points that is computable in $O(n)$ time.*

Cao [3] made the following observation on the relation between the cardinality of a maximum-cardinality point subset in general position and the *line cover number*, that is, the minimum number of lines that cover all points in the point set. For the sake of self-containment, we also give a short proof.

Observation 1 ([3]) *For a set P of points let $S \subseteq P$ be a maximum subset in general position and let ℓ be the line cover number of P. Then, $\sqrt{\ell} \le |S| \le 2\ell$.*

Proof. For the first inequality, note that $|S|$ points in general position define $\binom{|S|}{2} \le |S|^2$ lines. Since all other points in P have to lie on a line defined by two points in S, it follows that $\ell \le |S|^2$. The second inequality clearly holds since any maximum subset in general position can contain at most two points that lie on the same line. □

As a consequence of Observation 1, we can assume that $k \le 2\ell$ and, thus, we can transfer our results for the parameter k to the parameter ℓ.

Corollary 7 General Position Subset Selection *can be solved in $O(n^2 \log n + 41^{2\ell} \cdot (2\ell)^{4\ell})$ time, and there is a kernelization algorithm that, given an instance (P, k) of* General Position Subset Selection, *computes an equivalent instance containing at most $120\ell^3$ points in $O(n^2 \log n)$ time.*

4.2 FPT Results for the Dual Parameter

In this section we consider the dual parameter number $h := n - k$ of points that have to be *deleted* (*i.e.*, excluded from the sought point set in general position) so that the remaining points are in general position. We show a problem kernel containing $O(h^2)$ points for General Position Subset Selection. Moreover, we show that most likely this problem kernel is essentially *tight*, that is, there is presumably no problem kernel with $O(h^{2-\epsilon})$ points for any $\epsilon > 0$.

We start with the problem kernel that relies essentially on a problem kernel for the 3-Hitting Set problem:

3-Hitting Set
Input: A universe U, a collection $\mathcal{C}$ of size-3 subsets of U, and $h \in \mathbb{N}$.
Question: Is there a subset $H \subseteq U$ of size at most h containing at least one element from each subset $S \in \mathcal{C}$?

There is a close connection between General Position Subset Selection and 3-Hitting Set: For any collinear triple $p, q, r \in P$ of distinct points, one of the three points has to be deleted in order to obtain a subset in general position. Hence, the set of deleted points has to be a hitting set for the family of all collinear triples in P. Since 3-Hitting Set can be solved in $O(2.08^h + |\mathcal{C}| + |U|)$ time [21], we get:

Proposition 8 General Position Subset Selection *can be solved in $O(2.08^h + n^3)$ time.*

3-Hitting Set is known to admit a problem kernel with a universe of size $O(h^2)$ computable in $O(|U| + |\mathcal{C}| + h^{1.5})$ time [1]. Based on this, one can obtain a problem kernel of size $O(h^2)$ computable in $O(n^3)$ time. The bottleneck in this running time is listing all collinear triples. We can improve the running time of this kernelization algorithm by giving a direct kernel exploiting the simple geometric fact that two non-parallel lines intersect in one point. We first need two reduction rules.

Rule 2 *Let (P, k) be an instance of* General Position Subset Selection. *If there is a point $p \in P$ that is not collinear with any two other points in P, then delete p and decrease k by one.*

Clearly, Rule 2 is correct since we can always add a point which is not lying on any line defined by two other points to a general position subset. The next rule deals with points that are in too many conflicts. The basic idea here is that if a point lies on more than h distinct lines defined by two other points of P, then it has to be deleted. This is generalized in the next rule.

Rule 3 *Let (P, k) be an instance of* General Position Subset Selection. *For a point $p \in P$, let $\mathcal{L}(p)$ be the set of lines containing p and at least two points of $P \setminus \{p\}$, and for $L \in \mathcal{L}(p)$ let $|L|$ denote the number of points of P on L. Then, delete each point $p \in P$ satisfying $\sum_{L \in \mathcal{L}(p)} (|L| - 2) > h$.*

Lemma 9 *Rule 3 is correct.*

Proof. Let (P, k) be an instance of General Position Subset Selection and let $(P' := P \setminus D, k)$ be the reduced instance, where $D \subseteq P$ denotes the set of removed points. We show that (P, k) is a yes-instance if and only if (P', k) is a yes-instance.

Clearly, if (P', k) is a yes-instance, then also (P, k) is one. For the converse, we show that any size-k subset of P in general position does not contain any point $p \in D$: For each line $L \in \mathcal{L}(p)$, all but two points need to be deleted. If a subset $S \subseteq P$ in general position contains p, then the points that have to be deleted on the lines in $\mathcal{L}(p)$ are all different since any two of these lines only intersect in p. This means that $\sum_{L \in \mathcal{L}(p)} (|L| - 2)$ points need to be deleted. However, since this value is by assumption larger than h, the solution S is of size less than $k = |P| - h$. □

Theorem 10 General Position Subset Selection *admits a problem kernel containing at most $2h^2 + h$ points that is computable in $O(n^2)$ time.*

Proof. Let (P, k) be a General Position Subset Selection instance. We first show that applying Rule 2 exhaustively and then applying Rule 3 once indeed gives a small instance (P', k'). Note that each point $p \in P'$ is "in conflict" with at least two other points, that is, p is on at least one line containing two other points in P', since the instance is reduced with respect to Rule 2. Moreover, since the instance is reduced with respect to Rule 3, it follows that each point is in conflict with at most $2h$ other points. Thus, deleting h points can give at most $h \cdot 2h$ points in general position. Hence, if P' contains more than $2h^2 + h$ points, then the input instance is a no-instance.

We next show how to apply Rules 2 and 3 in $O(n^2)$ time. To this end, we follow an approach described by Edelsbrunner et al. [5] and Gómez et al. [10] which uses the dual representation and line arrangements. The dual representation maps points to lines as follows: $(a, b) \mapsto y = ax+b$. A line in the primal representation containing some points of P corresponds in the dual representation to the intersection of the lines corresponding to these points. Thus, a set of at least three collinear points in the primal corresponds to the intersection of the corresponding lines in the dual. An *arrangement* of lines in the plane is, roughly speaking, the partition of the plane formed by these lines. A representation of an arrangement of n lines can be computed in $O(n^2)$ time [5]. Using the algorithm of Edelsbrunner et al. [5], we compute in $O(n^2)$ time the arrangement $A(P^*)$ of the lines P^* in the dual representation of P.

Rule 2 is now easily computable in $O(n^2)$ time: Initially, mark all points in P as "not in conflict". Then, iterate over the vertices of $A(P^*)$ and whenever the vertex has degree six or more (each line on an intersection contributes two to the degree of the corresponding vertex) mark the points corresponding to the intersecting lines as "in conflict". In a last step, remove all points that are marked as "not in conflict".

Rule 3 can be applied in a similar fashion in $O(n^2)$ time: Assign a counter to each point $p \in P$ and initialize it to zero. We want this counter to store the number $\sum_{L\in\mathcal{L}(p)}(|L| - 2)$ on which Rule 3 is conditioned. To this end, we iterate over the vertices in $A(P^*)$ and for each vertex of degree six or more we increase the counter of each point corresponding to a line in the intersection by $d/2 - 2$ where d is the degree of the vertex. After one pass over all vertices in $A(P^*)$ in $O(n^2)$ time, the counters of the points store the correct values and we can delete all points whose counter is more than h. □

We remark that the results in Proposition 8 and Theorem 10 also hold when replacing the parameter h by the "number γ of *inner* points", where we call a point an inner point if it is not a corner point of the convex hull of P. The reason is that in all non-trivial instances we have $h \leq \gamma$ since removing all inner points yields a set of points in general position.

We can prove a matching (conditional) lower bound on the problem kernel size for General Position Subset Selection via a reduction from Vertex Cover. Given an undirected graph G and $k \in \mathbb{N}$, Vertex Cover asks whether there is a subset C of at most k vertices such that every edge is incident to at least one vertex in C. Using a lower bound result by Dell and van Melkebeek [4] for Vertex Cover (which is based on the common assumption in complexity theory that coNP is not in NP/poly since otherwise the polynomial hierarchy collapses to its third level), we obtain the following:

Theorem 11 *Unless coNP $\subseteq$ NP/poly, for any $\epsilon > 0$,* General Position Subset Selection *admits no problem kernel of size $O(h^{2-\epsilon})$.*

Proof. We give a polynomial-time reduction from Vertex Cover, where the resulting dual parameter h equals the size of the sought vertex cover. The claimed lower bound then follows because, unless coNP $\subseteq$ NP/poly, for any $\epsilon > 0$, Vertex Cover admits no problem kernel of size $O(k^{2-\epsilon})$, where k is the size of the vertex cover [4].

Given a Vertex Cover instance (G, k), we first reduce it to the equivalent Independent Set instance $(G, |V(G)| - k)$. We then apply transformation Φ (see Section 3) to G to obtain a set of points P, where $|P| = |V(G)| + |E(G)|$; we set $k' := |V(G)| + |E(G)| - k$, and consider the instance (P, k') of General Position Subset Selection. Clearly, G has a vertex cover of cardinality k if and only if G has an independent set of cardinality $|V(G)| - k$, which is true if and only if P has a subset in general position of cardinality $|E(G)| + |V(G)| - k$. Hence, the dual parameter $h = |P| - k'$ equals the sought vertex cover size. □

Note that Theorem 11 gives a lower bound only on the *total size* (*i.e.*, instance size) of a problem kernel for General Position Subset Selection. We can show a stronger lower bound on the number of points contained in any problem kernel using ideas from Kratsch et al. [15], which are based on a lower bound framework by Dell and van Melkebeek [4]. Kratsch et al. [15] showed that there is no polynomial-time algorithm that reduces a Point Line Cover instance (P, k) to an equivalent instance with $O(k^{2-\epsilon})$ points for any $\epsilon > 0$ unless coNP $\subseteq$ NP/poly. The proof is based on a result by Dell and van Melkebeek [4] who showed that Vertex Cover does not admit a so-called *oracle communication protocol* of cost $O(k^{2-\epsilon})$ for $\epsilon > 0$ unless coNP $\subseteq$ NP/poly. An oracle communication protocol is a two-player protocol, in which one player is holding the input and is allowed polynomial (computational) time in the length of the input, and the second player is computationally unbounded. The *cost* of the communication protocol is the number of bits communicated from the first player to the second player in order to solve the input instance.

Kratsch et al. [15] devise an oracle communication protocol of cost $O(n \log n)$ for deciding instances of Point

Line Cover with n points. Thus, a problem kernel for Point Line Cover with $O(k^{2-\epsilon})$ points implies an oracle communication protocol of cost $O(k^{2-\epsilon'})$ for some $\epsilon' > 0$ since the first player could simply compute the kernelized instance in polynomial time and subsequently apply the protocol yielding a cost of $O(k^{2-\epsilon} \cdot \log(k^{2-\epsilon}))$, which is in $O(k^{2-\epsilon'})$ for some $\epsilon' > 0$. This again implies an $O(k^{2-\epsilon''})$-cost oracle communication protocol for Vertex Cover for some $\epsilon'' > 0$ (via a polynomial-time reduction with a linear parameter increase [15, Lemma 6]). We show that there exists a similar oracle communication protocol of cost $O(n \log n)$ for General Position Subset Selection.

The protocol is based on *order types* of point sets. Let $P = \langle p_1, \dots, p_n \rangle$ be an ordered set of points and denote by $\binom{[n]}{3}$ the set of ordered triples $\langle i, j, k \rangle$ where $i < j < k$, $i, j, k \in [n] := \{1, \dots, n\}$. The *order type* of P is a function $\sigma : \binom{[n]}{3} \to \{-1, 0, 1\}$, where $\sigma(\langle i, j, k \rangle)$ equals 1 if p_i, p_j, p_k are in counter-clockwise order, equals -1 if they are in clockwise order, and equals 0 if they are collinear. Two point sets P and Q of the same cardinality are *combinatorially equivalent* if there exist orderings P' and Q' of P and Q such that the order types of P' and Q' are identical.

A key step in the development of an oracle communication protocol is to show that two instances of Point Line Cover with combinatorially equivalent point sets are actually equivalent [15, Lemma 2]. We can prove an analogous result for General Position Subset Selection:

Observation 2 *Let (P, k) and (Q, k) be two instances of* General Position Subset Selection. *If the point sets P and Q are combinatorially equivalent, then (P, k) and (Q, k) are equivalent instances of* General Position Subset Selection.

Proof. Let P and Q be combinatorially equivalent point sets with $|P| = |Q| = n$ and let $P' = \langle p_1, \dots, p_n \rangle$ and $Q' = \langle q_1, \dots, q_n \rangle$ be orderings of P and Q, respectively, having the same order type σ.

Now, a subset $S \subseteq P'$ is in general position if and only if no three points in S are collinear, that is, $\sigma(\langle p_i, p_j, p_k \rangle) \neq 0$ holds for all $p_i, p_j, p_k \in S$. Consequently, it holds that $\sigma(\langle q_i, q_j, q_k \rangle) \neq 0$, and thus the subset $\{q_i \mid p_i \in S\} \subseteq Q'$ is in general position. Hence, (P, k) is a yes-instance if and only if (Q, k) is a yes-instance. □

Based on Observation 2, we obtain an oracle communication protocol for General Position Subset Selection. The proof of the following lemma is completely analogous to the proof of Lemma 4.1 in [15]:

Lemma 12 *There is an oracle communication protocol of cost $O(n \log n)$ for deciding instances of* General Position Subset Selection *with n points.*

The basic idea is that the first player only sends the order type of the input point set so that the computationally unbounded second player can solve the instance (according to Observation 2 the order type contains enough information to solve a General Position Subset Selection instance). We conclude with the following lower bound result:

Theorem 13 *Let $\epsilon > 0$. Unless coNP $\subseteq$ NP/poly, there is no polynomial-time algorithm that reduces an instance (P, k) of* General Position Subset Selection *to an equivalent instance with $O(h^{2-\epsilon})$ points.*

Proof. Assuming that such an algorithm exists, the oracle communication protocol of Lemma 12 has cost $O(h^{2-\epsilon'})$ for some $\epsilon' > 0$. Since the reduction from Vertex Cover in Theorem 11 outputs a General Position Subset Selection instance where the dual parameter h equals the size k of the vertex cover sought, we obtain a communication protocol for Vertex Cover of cost $O(k^{2-\epsilon'})$, which implies that coNP $\subseteq$ NP/poly [4, Theorem 2]. □

Remark on the Kernel Lower Bound Framework of Kratsch, Philip and Ray As a final observation, we mention that the framework of Kratsch et al. [15] indeed is more generally applicable than stated there. It only relies on the equivalence of instances with respect to order types of point sets. Hence, we observe that for every decision problem on point sets for which

1. two instances with combinatorially equivalent point sets are equivalent (cf. Observation 2), and
2. there is no oracle communication protocol of cost $O(k^{2-\epsilon})$ for some parameter k and any $\epsilon > 0$ unless coNP $\subseteq$ NP/poly,

there is no problem kernel with $O(k^{2-\epsilon'})$ points for any $\epsilon' > 0$ unless coNP $\subseteq$ NP/poly.

5 Conclusion and Outlook

The intent of our work is to stimulate further research on the computational complexity of General Position Subset Selection. The kernelization results we presented rely mostly on combinatorial arguments; the main geometric property we used is that two distinct lines intersect in at most one point. Therefore, a natural question to ask is whether there are further geometric properties that can be exploited in order to obtain improved algorithmic results for General Position Subset Selection. We conclude with the following concrete open questions:

1. Can the $(15k^3)$-point kernel (Theorem 4) for General Position Subset Selection be asymptotically improved? Or can we derive a cubic, or even a quadratic, lower bound on the (point) kernel size of General Position Subset Selection?

2. Can the FPT algorithm (see Corollary 5) for General Position Subset Selection be (significantly) improved?

3. With respect to polynomial-time approximation, we could only show the APX-hardness of Maximum General Position Subset Selection. It remains open whether Cao's $O(\sqrt{\mathrm{opt}})$-factor approximation can be improved.

References

[1] R. van Bevern. Towards optimal and expressive kernelization for d-Hitting Set. *Algorithmica*, 70(1):129–147, 2014.

[2] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997.

[3] C. Cao. Study on Two Optimization Problems: Line Cover and Maximum Genus Embedding. Master's thesis, Texas A&M University, May 2012.

[4] H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014.

[5] 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.

[6] P. Erdős. On some metric and combinatorial geometric problems. *Discrete Mathematics*, 60:147–153, 1986.

[7] Z. Füredi. Maximal independent subsets in Steiner systems and in planar sets. *SIAM Journal on Discrete Mathematics*, 4(2):196–199, 1991.

[8] S. K. Ghosh and B. Roy. Some results on point visibility graphs. *Theoretical Computer Science*, 575:17–32, 2015.

[9] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *The Computer Journal*, 51(3):372–384, 2008.

[10] F. Gómez, S. Ramaswami, and G. T. Toussaint. On removing non-degeneracy assumptions in computational geometry. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity (CIAC '97)*, volume 1203 of *LNCS*, pages 86–99. Springer, 1997.

[11] L. J. Guibas, M. H. Overmars, and J. Robert. The exact fitting problem in higher dimensions. *Computational Geometry: Theory and Applications*, 6(4):215–230, 1996.

[12] R. K. Guy and P. A. Kelly. The no-three-in-line problem. *Canadian Mathematical Bulletin*, 11:527–531, 1968.

[13] R. Hall, T. Jackson, A. Sudbery, and K. Wild. Some advances in the no-three-in-line problem. *Journal of Combinatorial Theory, Series A*, 18(3):336–341, 1975.

[14] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[15] S. Kratsch, G. Philip, and S. Ray. Point line cover: The easy kernel is essentially tight. *ACM Transactions on Algorithms*, 12(3):40:1–40:16, 2016.

[16] S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[17] M. S. Payne. *Combinatorial geometry of point sets with collinearities.* PhD thesis, University of Melbourne, February 2014.

[18] M. S. Payne and D. R. Wood. On the general position subset selection problem. *SIAM Journal on Discrete Mathematics*, 27(4):1727–1733, 2013.

[19] K. F. Roth. On a problem of Heilbronn. *Journal of the London Mathematical Society*, 1(3):198–204, 1951.

[20] L. Tan. The group of rational points on the unit circle. *Mathematics Magazine*, 96(3):163–171, 1996.

[21] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems.* PhD thesis, Linköping University, March 2007.

Minimizing the Total Movement for Movement to Independence Problem on a Line

Mehrdad Ghadiri* Sina Yazdanbod †

Abstract

Given a positive real value δ, a set P of points along a line and a distance function d, in the movement to independence problem, we wish to move the points to new positions on the line such that for every two points $p_i, p_j \in P$, we have $d(p_i, p_j) \geq \delta$ while minimizing the sum of movements of all points. This measure of the cost for moving the points was previously unsolved in this setting. However for different cost measures there are algorithms of $O(n \log(n))$ or of $O(n)$. We present an $O(n \log(n))$ algorithm for the points on a line and thus conclude the setting in one dimension.

1 Introduction

The problem of minimizing the movement of points to reach a property was introduced first by Demaine et al. [4], which was for the most part in graphical settings. Many applications appear for the minimizing movement problem is in the contexts of reliable radio networks [1, 2], robotics [8] and map labeling [9][6]. In simple terms, the problem of movement to independence on graphs is defined as given a graph G and a set of pebbles P, move the pebbles such that no two pebbles occupy the same vertex. They considered the Total Sum measure on different problems. Although, they proved different NP-completeness results for other problems, the problem of whether the movement to independence problem with Total Sum measure is NP-complete, remains open to this day. Time complexity of the algorithms given in [4] were polynomial in the number of vertices. However, the number of pebbles can be much smaller than the number of vertices of the graph. That is why in [5], they turned to fixed-parameter tractability. Dumitrescu et al. [7] were the first to consider the settings of a real line. They gave LP-based algorithms for movement to independence on a line and on a closed curve with the measure of minimizing the maximum movement of points. In closed-curve version of the problem, authors defined distance as the length of the smallest subcurve between two points. Dumitrescu et al. [7]'s algorithms for both real line settings and closed curve settings were recently improved by Li et al. [10] with a linear time algorithm. Our contribution in this paper is considering the problem of Total Sum on the same settings of [7].

The rest of this paper is structured as follows. In Section 2, we explain preliminaries and the definitions of our problem. In Section 3, the formal settings of the problem is presented. The algorithm and its proof are written in Section 4 and the $O(nlogn)$ implementation and complexity analysis of it are presented in Section 5. In the end, we conclude the article in the last section and give an open problem for further research.

2 Preliminaries

In movement to independence problem, we are given a positive real value δ, a set P of points and a distance function d and we wish to move the points to new positions such that any two points are at least δ apart. The goal is to minimize this movement. There are several different measures of movement. We consider the *TotalSum measure* which is the sum of movements of all points. We examine this problem in the setting of real line. In this section, we define our the terminology and introduce the problems considered in this paper.

Definition 1 (Configuration) *For a set of points P, we define a* configuration *H of P to be a placement of points in the domain. For a point $p \in P$, we use $H(p)$ to denote the location of p in configuration H.*

In this paper, we will investigate movement to independence problem in the setting defined bellow.

Definition 2 (Independence) *Given a set of points P, a positive value $\delta \in \mathbb{R}^+$ and a distance function d, a configuration H is called independent, whenever for every two points $p_i, p_j \in P$, we have $d(H(p_i), H(p_j)) \geq \delta$.*

The formal definition of the movement to independence problem with total sum measure is as follows.

Definition 3 *Let $P = \{p_1, \ldots, p_n\}$ be a set of points and I be its initial configuration. Given $\delta \in \mathbb{R}^+$ and a distance function d, find an independent configuration F of P, so as to minimizes $\sum_{i=1}^{n} d(I(p_i), F(p_i))$.*

*Computer Engineering department, Sharif University of Technology, ghadiri@ce.sharif.edu

†Computer Engineering department, Sharif University of Technology, syazdanbod@ce.sharif.edu

The point set P can be from different domains. In the following, we define the distance function used for these domains.

Definition 4 (On a Line) *For two points $p_i, p_j \in P$ and configurations H and H' (not necessarily different) of points P on the real line, we define $d(H(p_i), H'(p_j)) = |H(p_i) - H'(p_j)|$.*

In our algorithm, we make use of chains of points. In a linear domain, a set of points in a configuration form a chain, whenever they are tightly put together in distances of δ.

Definition 5 *In a configuration H of points P, we call a subset $C = \{q_1, \dots, q_j\}$ of P, where $H(q_1) < \cdots < H(q_j)$, a* chain *in H, , if we have $d(H(q_i), H(q_{i+1})) = \delta$ for all $i = 1, \dots, j-1 (see Figure\ 1)$.*

A chain is maximal if it is not a proper subset of another chain. Unless noted explicitly, we consider chains to be maximal. Chain partitioning is the act of partitioning independent configuration into maximal chains. Figure 1 shows an example of this partitioning. In this figure the rectangles show the chains.

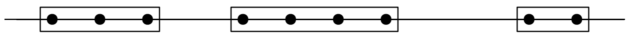


Figure 1: Partitioning H into chains

3 Setting of a Real Line

In this section we study the problem of movement to independence in real line domain with total sum measure. Let a point set $P = \{p_1, \dots, p_n\}$ be in $\mathbb{R}$ with initial configuration I. For the sake of simplicity, we assume that the input is given in sorted order and that points are initially in distinct positions. That is $I(p_i) < I(p_j)$ for every $i < j$. The following lemma shows that in fact we can make such assumptions.

Lemma 1 *The initial order of points P is preserved in the optimal configuration OPT. In other words, an optimal configuration OPT exists in which for any two points $p_i, p_j \in P$ with $I(p_i) < I(p_j)$, we have $OPT(p_i) < OPT(p_j)$.*

Proof. Let OPT be an optimal configuration. Assume that there exist $p_i, p_j \in P$ violating this. Then, we can swap the location of p_i and p_j in OPT, as in Figure 2, without increasing the total movement of points. Based on symmetry, one should only consider the three situations shown in Figure 2. □

In a given configuration H, We define the sets $L_H(S)$, $R_H(S)$ and $O_H(S)$ as follows.

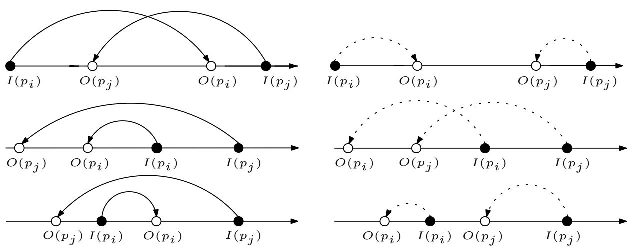

$I(p_i)$ $O(p_j)$ $O(p_i)$ $I(p_j)$
$I(p_i)$ $O(p_i)$ $O(p_j)$ $I(p_j)$
$O(p_j)$ $O(p_i)$ $I(p_i)$ $I(p_j)$
$O(p_i)$ $O(p_j)$ $I(p_i)$ $I(p_j)$
$O(p_j)$ $I(p_i)$ $O(p_i)$ $I(p_j)$
$O(p_i)$ $I(p_i)$ $O(p_j)$ $I(p_j)$

Figure 2: Three different situations of unordered points that can be put in order, without increasing total sum. Figures on the left shows the problem and figures on the right show the reordered version without moving the points

Definition 6 *For a given configuration H of points P with initial configuration I. For a subset $S \subseteq P$.*

- $L_H(S) = \{p \in S \mid H(p) < I(p)\}$
- $R_H(S) = \{p \in S \mid H(p) > I(p)\}$
- $O_H(S) = \{p \in S \mid H(p) = I(p)\}$

We may use the notations $L(S)$, $R(S)$ and $O(S)$ instead, when there is no ambiguity.

Our algorithm for real line domain is iterative and adds one point at a time until all the points are inserted. Let I be the initial configuration of points P and H_i be the configuration generated by our algorithm at the end of i-th iteration, which is an independent configuration (Note that H_i is defined over the set $\{p_1, \dots, p_i\}$). Let $C = \{c_1, \dots, c_k\}$ be the sorted set of chain partitioning of H_i, where c_k is the rightmost chain.

The main idea in this algorithm is that at the end of each iteration of the algorithm, the following properties are preserved for every chain $c_j \in C$:

Property 1 *For every chain $c_j \in C$, we have $|L_{H_i}(c_j)| + |O_{H_i}(c_j)| > |R_{H_i}(c_j)|$*

Property 2 *For every chain $c_j \in C$, we have $|L_{H_i}(c_j)| \leq |O_{H_i}(c_j)| + |R_{H_i}(c_j)|$*

Any configuration with these two properties is ,in a sense, locally optimal. We show that our algorithm creates the optimal solution with these properties.

4 The Algorithm

Our algorithm starts from p_1 and at each iteration adds one new point to the configuration. Each iteration of the algorithm consists of two phases. In the first phase, we insert a new point into the current configuration and if that violates Property 1, in the second phase, we restore that property. In the following, we explain the procedure for each phase.

4.1 Phase 1

Assume that we have the configuration H_i after processing the first i points and let p_{i+1} be the first remaining point and $\{c_1, \ldots, c_k\}$ be the chain partitioning of H_i. In this phase, we construct configuration G_{i+1} over the set $\{p_1, \ldots, p_{i+1}\}$, let $G_{i+1}(p_j) = H_i(p_j)$, for all $j = 1, \ldots, i$. So, we just need to determine the location of p_{i+1} in G_{i+1}. We consider the two following cases based on the distance of the new point from the last inserted point in our created configuration:

Case 1. If $d(H_i(p_i), I(p_{i+1})) \geq \delta$ and $I(p_{i+1})$ is to the right of $H_i(p_i)$, then we set $G_{i+1}(p_{i+1}) = I(p_{i+1})$. If $d(G_{i+1}(p_i), G_{i+1}(p_{i+1})) > \delta$, then the chain partitioning of G_{i+1} is $\{c_1, \ldots, c_k, c_{k+1}\}$ where c_{k+1} is a new chain which its only member is p_{i+1} (Figure 3) . If $d(G_{i+1}(p_i), G_{i+1}(p_{i+1})) = \delta$, the chain partitioning of G_{i+1} is $\{c_1, \ldots, c_{k-1}, c'_k\}$ where c'_k is a new chain which is $c_k \cup p_{i+1}$ (Figure 4). Clearly, the new point is in $O_{c'_k}$(the set of points from c'_k that are on their initial location) or $O_{c_{k+1}}$. That is, the resulting configuration preserves Property 1 and 2. Therefore, there is no need to run Phase 2 and we proceed to the next iteration. Note that in this case H_{i+1} will be G_{i+1}.

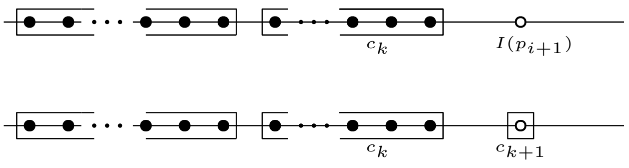


Figure 3: This is the case where $d(H_i(p_i), I(p_{i+1})) > \delta$. The new point will create a chain consisting of itself

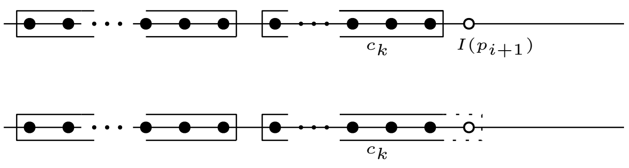


Figure 4: This is the case where $d(H_i(p_i), I(p_{i+1})) = \delta$. The new point will merge with the previous chain

Case 2. If $d(H_i(p_i), I(p_{i+1})) < \delta$ (Figure 5) or $d(H_i(p_i), I(p_{i+1})) \geq \delta$ and $I(p_{i+1})$ is to the left of $H_i(p_i)$ (Figure 6), we set $G_{i+1}(p_{i+1})$ to $H_i(p_i) + \delta$. Therefore, the chain partitioning of G_{i+1} is $\{c_1, \ldots, c_{k-1}, c'_k\}$ where c'_k is a new chain which is $c_k \cup p_{i+1}$. The only complication is that Property 1 might get violated in G_{i+1} because $p_{i+1} \in R_{G_{i+1}}(c'_k)$. In this case, we proceed to Phase 2, to move the chains so that Property 1 is restored again. Otherwise, there is no need to run Phase 2 and we proceed to the next iteration. Note that in this case H_{i+1} will be G_{i+1}.

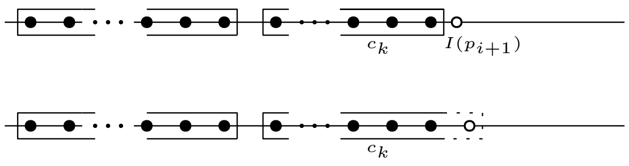


Figure 5: This is the case where $d(H_i(p_i), I(p_{i+1})) < \delta$ but the new point is still located after $H_i(p_i)$.

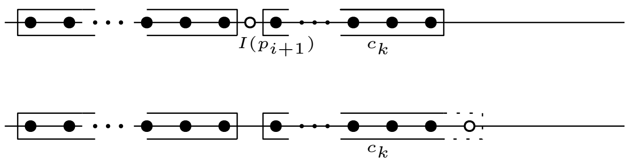


Figure 6: This is the case where $d(H_i(p_i), I(p_{i+1})) \geq \delta$ but the new point is still located before $H_i(p_i)$.

4.2 Phase 2

In this phase, we construct the configuration H_{i+1} given the configuration G_{i+1} from the previous phase which its chain partitioning is $\{c_1, \ldots, c_{k-1}, c'_k\}$. For configuration H_{i+1}, we have $H_{i+1}(p_j) = G_{i+1}(p_j)$, if $p_j \in c_1 \cup \cdots \cup c_{k-1}$ and we just need to determine location of points in c'_k in configuration H_{i+1}. The reason for running this phase is that Property 1 is violated for the chain c'_k in Phase 1, which means $|R_{G_{i+1}}(c'_k)| \geq |L_{G_{i+1}}(c'_k)| + |O_{G_{i+1}}(c'_k)|$. Since $|R_{G_{i+1}}(c'_k)| = |R_{H_i}(c_k)| + 1$ and $|R_{H_i}(c_k)| < |L_{H_i}(c_k)| + |O_{H_i}(c_k)|$, we can infer that $|R_{G_{i+1}}(c'_k)| = |L_{G_{i+1}}(c'_k)| + |O_{G_{i+1}}(c'_k)|$. It is clear that Property 1 and 2 still hold for the other chains and also Property 2 holds for c'_k. To restore Property 1 for c'_k, we do as follows.

Let $\alpha = \min_{p_j \in R_{c'_k}} |d(G_{i+1}(p_j), I(p_j))|$ be the value of the minimum distance between a point in the new configuration and their initial configuration. and $\beta = d(G_{i+1}(p_r), G_{i+1}(p_l)) - \delta$, where p_r is the rightmost point of c_{k-1} in configuration G_{i+1} and p_l is the leftmost point of c'_k in that configuration. In other words, $\beta + \delta$ is the distance of the last point of the chain c_{k-1} and first point of the chain c'_k. We consider the two following cases:

Case 1. If $\alpha < \beta$, we set $H_{i+1}(p_j) = G_{i+1}(p_j) - \alpha$, for all $p_j \in c'_k$. It is clear that we have $|R_{G_{i+1}}(c'_k)| > |R_{H_{i+1}}(c'_k)|$. Therefore $|L_{H_{i+1}}(c'_k)| + |O_{H_{i+1}}(c'_k)| > |R_{H_{i+1}}(c'_k)|$. We also know that $L_{G_{i+1}}(c'_k) \cup O_{G_{i+1}}(c'_k) = L_{H_{i+1}}(c'_k)$ and $R_{G_{i+1}}(c'_k) = O_{H_{i+1}}(c'_k) \cup R_{H_{i+1}}(c'_k)$. Since $|R_{G_{i+1}}(c'_k)| = |L_{G_{i+1}}(c'_k)| + |O_{G_{i+1}}(c'_k)|$, we conclude that $|L_{H_{i+1}}(c'_k)| \leq |O_{H_{i+1}}(c'_k)| + |R_{H_{i+1}}(c'_k)|$. Therefore, Property 1 is restored and Property 2 is preserved.

Case 2. If $\alpha \geq \beta$, we set $H_{i+1}(p_j) = G_{i+1}(p_j) -$

β, for all $p_j \in c'_k$. It is clear that c_{k-1} and c'_k are not maximal chains in H_{i+1}. Therefore, the chain partitioning of H_{i+1} is $\{c_1 \dots, c_{k-2}, c'_{k-1}\}$, where $c'_{k-1} = c_{k-1} \cup c'_k$. We can say two chains c_{k-1} and c'_k are merged (Figure 7). We have $|L_{H_{i+1}}(c'_k)| + |O_{H_{i+1}}(c'_k)| \geq |R_{H_{i+1}}(c'_k)|$, because $|L_{G_{i+1}}(c'_k)| + |O_{G_{i+1}}(c'_k)| = |R_{G_{i+1}}(c'_k)|$. We also have $|L_{H_{i+1}}(c_{k-1})| + |O_{H_{i+1}}(c_{k-1})| > |R_{H_{i+1}}(c_{k-1})|$. Hence, $|L_{H_{i+1}}(c'_{k-1})|+|O_{H_{i+1}}(c'_{k-1})| > |R_{H_{i+1}}(c'_{k-1})|$ and Property 1 is restored. By a reasoning similar to previous case, we can conclude that Property 2 holds for c'_{k-1} in H_{i+1}.

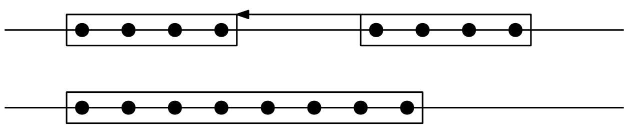


Figure 7: When the left chain reaches δ radius of the right chain, they merge.

4.3 Correctness

We claim that this algorithm returns an optimal configuration. But before we go on to prove that claim, we state a lemma.

Lemma 2 *Let $c = \{p_i, \dots, p_j\}$ be a maximal chain in H_n. For all l, where $i \leq l \leq j$, for the non-maximal chain $c' = \{p_i, \dots, p_l\}$, we have*

$$|L_{H_n}(c')| + |O_{H_n}(c')| > |R_{H_n}(c')|.$$

Proof. In the l-th iteration of the algorithm, p_l was inserted into the configuration. Let $\{c_1, \dots, c_k\}$ be the chain partitioning of H_l and $c' = c_m \cup \dots \cup c_k$. We have $|L_{H_l}(c')| + |O_{H_l}(c')| > |R_{H_l}(c')|$, because Property 1 holds for all chains in $\{c_1, \dots, c_k\}$. After iteration l, points in the c' only move leftwards or does not move in each iteration. Thus, the left side of the inequality is non-decreasing and the right side is non-increasing. Therefore, at the end of every further iteration, the inequality still holds. In particular, the inequality holds at the end of the algorithm. □

Now we have the sufficient tools to prove optimality of the output of this algorithm. We make the argument in two cases, once we take the rightmost difference from the optimal and second we use the leftmost difference. In the end, our solution is optimal or simply a shift of the optimal to the right that does not increase the sum.

Theorem 3 *Configuration H_n is optimal.*

Proof. Let OPT be an optimal configuration of points P which preserves the order of initial configuration. According to Lemma 1, this configuration exists. Take the rightmost point p_l in H_n such that $OPT(p_l) < H_n(p_l)$ (Figure 8). Assume that this point is in the chain $c = \{p_i, \dots, p_j\}$. Figure 8 depicts this situation.

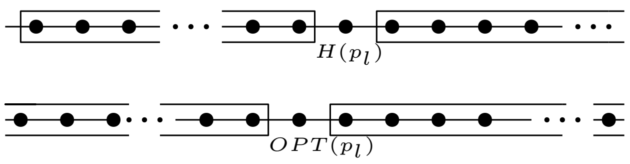


Figure 8: The above figure is the chain containing p_l in our solution while below figure show the same points of the above chain in the optimal solution. This chain is not necessarily intact, and may have been divided into several different chains in the optimal solution.

We know from Lemma 2 that for the $c' = \{p_i, \dots, p_l\}$ in H_n, we have:

$$|L_{H_n}(c')| + |O_{H_n}(c')| > |R_{H_n}(c')|$$

For each $k = i, \dots, l$ we have $H_n(p_k) < OPT(p_k)$. Since, the order of points in OPT is like H_n and also OPT is an independent configuration. Therefore, $O_{H_n}(c') \cup L_{H_n}(c') \subset L_{OPT}(c')$ and $|L_{OPT}(c')| > |O_{OPT}(c')|+|R_{OPT}(c')|$. Hence, if we shift the points of c' in OPT to right by $d(H_n(p_l), OPT(p_l))$, the number of points getting further away from their initial location will be smaller than the number of points getting closer and also the points will remain independent from each other. Hence, total movement of points will decrease, which contradicts the optimality of OPT. Therefore, there are no points in the optimal configuration to the left of their corresponding point in H_n.

On the other hand, let $H_n(p_l)$ be the leftmost point such that $OPT(p_l) > H_n(p_l)$(Figure 8). Assume that this point is in the chain $c = \{p_i, \dots, p_j\}$ in H_n. This case is shown in Figure 9.

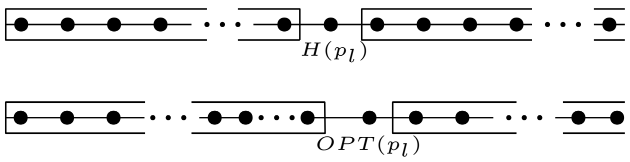


Figure 9: Same as before, rectangles are chains and p_l is the first difference.

This time, we use Property 2. For the chain c we have:

$$|L_{H_n}(c)| \leq |O_{H_n}(c)| + |R_{H_n}(c)|.$$

It is fairly easy to see that:

$$|L_{OPT}(c)| \leq |L_{H_n}(c)| \leq |O_{H_n}(c)| + |R_{H_n}(c)|$$

giving

$$|O_{H_n}(c)| + |R_{H_n}(c)| \leq |O_{OPT}(c)| + |R_{OPT}(c)|$$

because the order of points in OPT is like H_n and also OPT is an independent configuration. If we shift the points $p_l, \ldots, p_j$ in the configuration OPT to left, total movement of points will not increase until after p_l coincides with $H_n(p_l)$. That is to say, the total movement of the solution returned by our algorithm is less than or equal to that of the optimal configuration. After placing p_l on $H_n(p_l)$ by moving all the points $\{p_l, \ldots, p_j\}$ in the configuration OPT to left, we have a new configuration OPT' with the same (if not less) total movement. Now, we find the next point from OPT' with this property (leftmost point such that $OPT'(p_l) > H_n(p_l)$) and we continue until all the points with this property are converted to their corresponding point in H_n, therefore, proving that the cost of our solution is at most that of the optimal solution. □

A naive implementation of this algorithm runs in $O(n^2)$ time. However, in section 5 we give a more efficient implementation that runs in $O(n \log(n))$ time.

5 Implementaion and complexity analysis

In each iteration of the algorithm, there are two phases. In the first one, a point is placed on its initial location or on the end of the last chain. Obviously, the complexity of this phase is $O(1)$. In the second phase, we move all of a chain and possibly merge it with another chain. If we update location of all the points in this phase then in each iteration, the complexity of this phase is as the size of the moving chain. Therefore, in the worst case, the complexity of the algorithm will be $O(n^2)$. We use a little trick to reduce the complexity of the algorithm. Let $c_1, \ldots, c_k$ be the chains in a configuration like H. Let $r(c_j)$ be a real number that shows the total movement of c_j to the left since it was created. In other words, $r(c_j)$ is the total movement of the left most point of c_j to the left since it has been added to the c_j.

Let p be a newly added point to the c_k and its location be ℓ. The trick is that instead of storing the actual location of p, we store $\ell - r_j$. When we need the actual location of p, we can easily recompute that. Also, when we move the chain to the left, it is sufficient to update just r_j and we do not need to update a number for each point. With this trick, we reduce the time complexity of moving the chains to $O(1)$. There are two other things that affect the complexity of the algorithm: finding the amount of movement of a chain in the second phase of each iteration and merging two chains when we deal with Case 2 of the second phase.

For finding the amount of movement of a chain, we can store all the right points of a chain in a min-heap according to their distances to their initial locations. When we add a point to the chain, we can easily add it to the heap in $O(\log n)$ and when we move the chain to the left, we need to remove the point that is locates on its initial location from the heap which can be done in $O(\log n)$.

In case of merging the chains, let c_j and c_{j+1} be the chain that merged and the new chain be c'. We need to merge c_j and c_{j+1}'s heaps which can be done in $O(\log n)$ if we use a binomial heap[3, p. 462]. The other thing that we need to do is to set a value for r(c'). To do this we choose one of c_j or c_{j+1} that have more points and set $r(c')$ as its r value and update the location value of the points of the other chain using $r(c')$. Note that, the new value of $r(c')$ does not show necessarily the amount of movement of the new chain but it can be treated as before. The amortized cost of this action is $O(\log n)$ like the disjoint-set data structure [3, p. 504].

Due to the above analysis, we can conclude that the cost of each iteration of the algorithm is $O(\log n)$ and the complexity of the algorithm is $O(n \log n)$.

Theorem 4 *Running time of the algorithm is* $O(n \log(n))$

6 Conclusion

In this paper we considered the problem of minimizing total sum of movement of points to reach independence and presented an $O(nlogn)$ algorithm. While the problem for minimizing the movement of point on a circle(or a closed curve) still remains unsolved. It is easy to see that our properties determining a local optimal can be considered in the circle case as well. However, this problem shows to be a little more trickier to solve and these properties might not be enough.

7 Acknowledgment

In the end, we would like to thank our dear friend, Sahand Mozaffari, for his thoughtful comments and suggestions.

References

[1] J. L. Bredin, E. D. Demaine, M. Hajiaghayi, and D. Rus. Deploying sensor networks with guaranteed capacity and fault tolerance. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 309–319. ACM, 2005.

[2] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3602–3608. IEEE, 2004.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

[4] E. D. Demaine, M. T. Hajiaghayi, H. Mahini, A. S. Sayedi-Roshkhar, S. O. Gharan, and M. Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3), 2009.

[5] E. D. Demaine, M. T. Hajiaghayi, and D. Marx. Minimizing movement: Fixed-parameter tractability. *ACM Transactions on Algorithms*, 11(2):14:1–14:29, 2014.

[6] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. Map labeling and its generalizations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana.*, pages 148–157, 1997.

[7] A. Dumitrescu and M. Jiang. Constrained k-center and movement to independence. *Discrete Applied Mathematics*, 159(8):859–865, 2011.

[8] T. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V, Selected Contributions of the Fifth International Workshop on the Algorithmic Foundations of Robotics, WAFR 2002, Nice, France, December 15-17, 2002*, pages 77–94, 2002.

[9] M. Jiang, J. Qian, Z. Qin, B. Zhu, and R. J. Cimikowski. A simple factor-3 approximation for labeling points with circles. *Inf. Process. Lett.*, 87(2):101–105, 2003.

[10] S. Li and H. Wang. Algorithms for minimizing the movements of spreading points in linear domains. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, 2015.

Searching with Advice: Robot Fence-Jumping (Extended Abstract)

Kostantinos Georgiou*[†] Evangelos Kranakis*[‡] Alexandra Steau[§]

Abstract

We study a new search problem on the plane involving a robot and an immobile treasure, initially placed at distance 1 from each other. The length β of an arc (a fence) within the perimeter of the corresponding circle, as well as the promise that the treasure is outside the fence, is given as part of the input. The goal is to device movement trajectories so that the robot locates the treasure in minimum time. Notably, although the presence of the fence limits searching uncertainty, the location of the fence is unknown, and in the worst case analysis is determined adversarially. Nevertheless, the robot has the ability to move in the interior of the circle. In particular, the robot can attempt a number of chord-jump moves if it happens to be within the fence or if an endpoint of the fence is discovered.

The optimal solution to our question can be obtained as an evaluation to a complicated optimization problem, which involves trigonometric functions, and trigonometric equations that do not admit closed form solutions. For the 1-Jump Algorithm, we fully describe the optimal trajectory, and provide an analysis of the associated cost as a function of β. Our analysis indicates that the optimal k-Jump Algorithm requires that the robot has enough memory and computation power to compute the optimal chord-jumps. Motivated by this, we give an abstract performance analysis for every k-Jump Algorithm. Subsequently, we present a highly efficient Halving Heuristic k-Jump Algorithm that can effectively approximate the optimal k-Jump Algorithm, with very limited memory and computation requirements.

Key words and phrases. Disk, Fence, Optimization, Robot, Search, Speed, Treasure.

1 Introduction

Geometric search is concerned with finding a target placed in a geometric region and has been investigated in many areas of mathematics, theoretical computer science, and robotics. In each instance, one aims to provide search algorithms that optimize a certain cost, which may take into account a variety of important characteristics and features of the domain, computational abilities of the searcher, assumptions about the target, etc. In this paper, we introduce and study *fence-jumping search*, a new search problem involving a robot, an unknown stationary fence (barrier), and an unknown stationary target (or treasure) in the plane.

The location of the treasure is unknown to the robot. However, the robot has knowledge that at the start it is located at a distance of 1 (unit) away from the treasure. Equivalently, the treasure is stationed on the perimeter of a disk (within the known environment), which is centered at the start point of the robot. A *fence*, a given circular arc of length β, is placed on the perimeter of the disk, whose location is also unknown to the robot. Further, the robot has the knowledge that the treasure is located on the perimeter but not on the fence. Depending on its trajectory, the robot may move along the perimeter of the circle and occasionally, say when within the fence, it may want to move along a chord, or as we say to *fence-jump*, so as to reduce the time necessary to perform the search. We will analyze several fence-jumping algorithms that will allow us to reach the treasure in minimal time.

We study the fence-jumping search problem for one robot starting at the center of the disk and moving at a constant speed 1. We assume the treasure is stationary and placed by an adversary at the beginning of each round depending on the fence location. The adversary positions the treasure on the perimeter, but outside the fence. The robot may move anywhere on the disk in an attempt to find this treasure; it is also able to use geometric knowledge so as to decrease the amount of time spent during the search. That is to say, since the robot knows that the treasure is not located on the fence, it could try to bypass it by “jumping over the fence”. The goal of this paper is to determine a trajectory so that the robot finds the treasure in optimal time.

1.1 Related Work

The type of search problem investigated in our work was first seen sixty years ago when Beck [4] and Bellman [5]

*Research supported in part by NSERC.

[†]Department of Mathematics, Ryerson University, konstantinos@ryerson.ca

[‡]School of Computer Science, Carleton University, kranakis@scs.carleton.ca

[§]School of Computer Science, Carleton University, AlexandraSteau@cmail.carleton.ca

asked an important, yet simplistic question tied to the minimization of distance. Motivated from this, several different natural search problems have been studied including the use of a fixed [8, 16] or mobile target [18], the tools searchers have access to, the number of searchers, the communication restrictions and many more. Often, the essential part of the robot activity is the recognition and/or mapping of the terrain. In the case of a known structure, the main objective of the search is to minimize the time to find the treasure. Searching for a motionless target has been studied in the cow-path problem [4], lost in a forest problem [12, 15] and plane searching problem [2, 3].

Baeza-Yates *et al* in their well known paper [3] study the worst-case time for search involving one robot and a treasure at an unknown location in the plane, such as a simple line. Useful surveys on search theory can also be found in [6] and [10].

Search by multiple robots with communication capabilities has been considered in [11, 14], while [8, 9] study the evacuation of k robots searching for an exit located on the perimeter of a disk. The problem of finding trajectories for obstacle avoidance in both known and unknown terrains has been considered in several papers including [1, 7, 17].

1.2 Outline and Results of the Paper

As a main objective, our approach will have to design algorithms for finding the treasure in optimal time, while adapting to the fence structure located on the perimeter. Thus, leading us to propose algorithms that attempt to deliver the optimal shortcuts necessary to exit and/or avoid the fence structure.

An outline of our results is as follows. In Section 2, we introduce the basic concepts and analyze a simple search algorithm for finding the treasure without involving any jumps. In Section 3, we introduce and analyze the optimal 1-Jump search algorithm. Meanwhile, in Section 4, we propose a generic description of k-Jump algorithms. In Section 5, we study a k-Jump algorithm based on a *halving* heuristic, which approximates the optimal jump without relying on solutions of trigonometric optimization problems. In Section 6, we contrast the choices and performance of the Halving k-Jump Algorithm with the choices and performance of the Optimal k-Jump Algorithm that was obtained using optimization software packages, for $k \leq 3$. We conclude with Section 7. Due to space limitations, all omitted lemma proofs can be found in the extended paper [13].

2 Preliminary Observations

First, we introduce the basic concepts and assumptions of our model. Initially, we make the assumption that the robot is located in the center of the disk with a radius of 1, and a treasure is located at distance of 1 from the robot, on the perimeter of the disk. We define this treasure to be a point on the disk and, thus, does not take any space on the perimeter. The treasure location is always unknown to the robot until it moves directly over its point location. That is to say, the robot has no vision capabilities, in that it becomes aware of what each point on the circle is, i.e. a fence point, treasure or nothing special, only if the point is visited. The robot moves at the same speed throughout its search on the disk and the movement of the robot from the center always takes 1 unit of time. The robot has the computational power to numerically solve trigonometric equations through the use of deterministic processors which possess the required memory for these processes.

Recall that robot's goal is to optimize the length of its trajectory using various types of movements, i.e. the robot may walk on the fence or even jump over the fence moving along a chord (within the interior of the circle).

To begin we provide a naive solution to our treasure finding problem, which we will then improve with a number of algorithms. In what follows, we denote the length of the fence by β, given as part of the input. Independently of the algorithm considered, any deterministic algorithm will first have the robot move to an arbitrary point on the perimeter of the circle, thereafter referred to as the *basic landing point*, with the intention that the robot will start moving/searching the circle counterclockwise, which is further examined in Algorithm 1.

Algorithm 1 0-Jump Algorithm

```
1: Walk to the perimeter of the disk
2: Continue walking on the perimeter counterclockwise
3: if you reach fence endpoint then
4:     Hop along the corresponding chord of length
5:     $2\sin(\beta/2)$
6: else
7:     Walk on perimeter until you find treasure
```

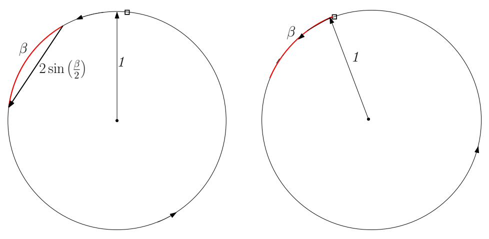
Our work provides a focus on algorithms that perform well under worst case analysis. As such, the performance of any algorithm will be determined after an adversary decides on the location of both the basic landing point, the fence itself, and the treasure. For the sake of exposition, we now present the worst case termination time depending on the location of the basic landing point.

Lemma 1 *The worst case termination time of Algorithm 1 is*[1]

$$\begin{cases} c_0^0 := 1 + 2\pi - \beta + 2\sin(\beta/2) & \textit{Figure 1a} \\ c_1^0 := 1 + 2\pi & \textit{Figure 1b} \end{cases}$$

[1]The usefulness of notation c_j^i for the cost of the algorithm will be transparent in the next sections

where the reference in the right column indicates the Figure which applies to the case.



(a) Landing outside fence. (b) Landing within fence.

Figure 1: Basic landing point.

Proof. Suppose that the basic landing point is outside the fence, as seen in Figure 1a, and say that the clockwise distance between the landing point and the fence is $x \in (0, 2\pi - \beta)$. It is straightforward that the adversary would place the treasure clockwise inbetween the landing point and the fence, at clockwise distance $y \in (0, x)$ from the landing point. Then, for all $x \in (0, 2\pi - \beta)$ the cost of the algorithm would be

$$\sup_{y\in(0,x)} \{1+2\pi-\beta+2\sin(\beta/2)-y\} = 1+2\pi-\beta+2\sin(\beta/2)\,.$$

In the other case, the landing point is within the fence, as illustrated in Figure 1b. Suppose that the clockwise distance between the landing point and the endpoint of the fence is $x \in (0, \beta)$. Also suppose that the clockwise distance between the same fence endpoint and the treasure is $y \in (0, 2\pi - \beta)$. Then, the robot will locate the treasure in time

$$\sup_{x,y}\{1 + 2\pi - x - y\} = 1 + 2\pi.$$

This proves Lemma 1. □

It is intuitive that having the basic landing point outside the fence is a "favorable event" in that for all β, $c^0_0 \le c^0_1$. This follows formally from the fact that the non-negative expression $\beta - 2\sin(\beta/2)$ is increasing in $\beta > 0$. Hence, the performance of Algorithm 1 is $1+2\pi$.

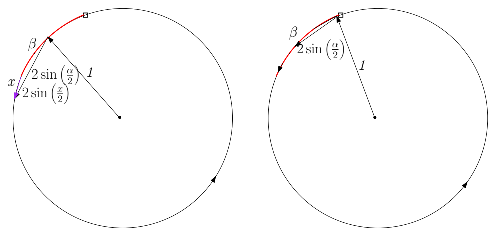
Next, we focus on algorithms that can address the choice of the adversary placing (basic) landing points within the fence. In such algorithms the robot will try to jump in an attempt to land outside the fence.

3 The Optimal 1-Jump Algorithm

In this section we analyze the optimal 1-Jump Algorithm, which also serves as a warm-up for the analysis of the generic k-Jump Algorithm. 1-Jump Algorithms are fully determined by the (unique) chord jump of corresponding arc-length α they make in case the basic landing point (of the robot) is within the fence.

It is worthwhile discussing the required specifications for the algorithm to be correct. First, we require the jump to be in "counter-clockwise" direction, i.e. that $\alpha \le \pi$ (this also breaks the symmetry for the adversarial placements of the fence and the treasure). Second, we further require that the chord jump does not pass over the area that could hold the treasure, landing back to the fence. For that, it is of importance that $\alpha \le 2\pi - \beta$. To summarize, the 1-Jump Algorithm is fully determined by choosing α satisfying $0 < \alpha \le \min\{\pi, 2\pi - \beta\}$.

In continuation, with parameter α, Algorithm 2 runs similarly to Algorithm 1, except from the last landing case of Algorithm 1 falling within the fence (which happens to be the basic one). If that happens, Algorithm 2 makes a counterclockwise jump corresponding to arc length α. If the 1st-jump landing point is in the fence, then it runs Algorithm 1. Otherwise, the 1st-jump landing point is outside the fence and the robot applies the following *remedy phase*: move clockwise along the periphery of the circle till the endpoint of the fence is found, say at arc distance x, and then return to the 1st-jump landing point along the corresponding chord of length $2\sin(x/2)$, and continue executing Algorithm 1.



β
$2\sin\left(\frac{\alpha}{2}\right)$
1
x
$2\sin\left(\frac{x}{2}\right)$
β
$2\sin\left(\frac{\alpha}{2}\right)$
1

(a) 1st jump landing is outside fence. (b) 1st jump landing is inside fence.

Figure 2: 1-Jump Algorithm basic landing point.

Algorithm 2 1-Jump

```
1: Walk to the perimeter of the disk
2: if your landing point is inside the fence then
3:     make a ccw chord jump of arc length $\alpha$
4: Perform Algorithm 1
```

Lemma 2 *Depending on the landing points, the cost of Algorithm 2 with parameter α is*

$$\left\{\begin{array}{lcll} c^1_0 & := & 1+2\pi-\beta+2\sin(\beta/2) & \textit{Figure 1a} \\ c^1_1 & := & c^1_0 + 4\sin(\alpha/2) - 2\sin(\beta/2) & \textit{Figure 2a} \\ c^1_2 & := & 1+2\pi-(\alpha-2\sin(\alpha/2)) & \textit{Figure 2b} \end{array}\right. \tag{1}$$

with the understanding that c^1_0, c^1_1, c^1_2 are functions on β and α, and the reference in the right column indicates the Figure which applies to the case.

Critical to our analysis toward specifying the optimal choice of α is the solution to a specific equation that does not admit a closed form. Consider expression $\alpha + 2\sin(\alpha/2)$ which is monotonically increasing. As such, for every $\beta \in \mathbb{R}$, the equation $\alpha + 2\sin(\alpha/2) = \beta$ admits a unique solution in α. Motivated by this observation we write that "α_β is the unique real number satisfying equation $\alpha_\beta + 2\sin(\alpha_\beta/2) = \beta$".

Moreover, since $\alpha + 2\sin(\alpha/2)$ is increasing in the variable α, so is α_β in the variable β. We are now ready to define and analyze the optimal 1-Jump Algorithm.

Theorem 3 *Let γ be the unique solution to equation $\pi = \gamma - \sin(\gamma/2)$ ($\gamma \approx 4.04196$). The optimal 1-Jump Algorithm chooses jump step corresponding to arc length $\alpha = \alpha_\beta$ if $\beta \le \gamma$, $\alpha = 2\pi - \beta$ if $\beta > \gamma$ and terminates in time*

$$1 + \begin{cases} 2\pi - \alpha_\beta + 2\sin(\alpha_\beta/2) & \text{if } \beta \le \gamma \\ \beta + 2\sin(\beta/2) & \text{if } \beta > \gamma. \end{cases} \tag{2}$$

Proof. By Lemma 2, the optimal 1-Jump Algorithm is determined by choosing α that minimizes

$$\sup_{0<\alpha<\min\{\pi,\pi-\beta\}} \{c_0^1, c_1^1(\alpha), c_2^1(\alpha)\},$$

where in the expression above, we make the dependence on α explicit. Again, it should be clear that having the basic landing point outside the fence is a "favorable event". Intuitively, this is the only case that the robot makes full use of the fact that the treasure does not lie within the fence, jumping over it and using the corresponding chord. Effectively, this implies that for all β, α we have $c_0^1 \le \min\{c_1^1(\alpha), c_2^1(\alpha)\}$.

Next, for any $\beta \in (0, 2\pi)$ we need to choose α so as to minimize $\max\{c_1^1(\alpha), c_2^1(\alpha)\}$. To that end, note that c_1^1, c_2^1 exhibit different monotonicities with respect to α so that, if possible, the minimum will be attained when the two costs are equal. Equating the two costs gives that $\alpha + 2\sin(\alpha/2) = \beta$. Recall that we have denoted the unique solution to the equation by α_β which is increasing in β. Since the jump step needs to stay no more than $\min\{\pi, 2\pi - \beta\}$, the choice $\alpha = \alpha_\beta$ (which is the best possible) is valid only when $\alpha_\beta \le \min\{\pi, 2\pi - \beta\}$. Numerically we can compute $\alpha_\pi \approx 1.66$, which due to the monotonicity of α_β implies that the dominant constraint is that $\alpha_\beta \le 2\pi - \beta$. Hence, any restrictions will be imposed for $\beta > \pi$. Indeed, in setting $\alpha_\beta = 2\pi - \beta$, and substituting in $\alpha_\beta + 2\sin(\alpha_\beta/2) = \beta$, we obtain $2\pi - \beta + 2\sin(\pi - \beta/2) = \beta$. The value of β that satisfies this equation is $\gamma \approx 4.04196$.

To resume, as long as $\beta \le \gamma$, the best choice for the jump is the solution to the equation $\alpha + 2\sin(\alpha/2) = \beta$. When $\beta > \gamma$, the best jump step is equal to $2\pi - \beta$.

From the discussion above, the induced cost when $\beta \le \gamma$ would be equal to $c_1^1(\alpha_\beta)$, as it reads in Lemma 2. Finally, when $\beta > \gamma$ the induced cost would be

$$\begin{aligned} &\max\{c_1^1(2\pi - \beta), c_2^1(2\pi - \beta)\} \\ &= 1 + 2\pi + \max\{4\sin(\pi - \beta/2) - \beta, 2\sin(\pi - \beta/2) - \\ &\quad 2\pi + \beta\} \\ &= 1 + 2\pi + 2\sin(\beta/2) + \max\{2\sin(\beta/2) - \beta, -2\pi + \beta\} \\ &= 1 + \beta + 2\sin(\beta/2) \end{aligned}$$

where the last equality is because $\beta \ge \gamma$, the definition of γ and the fact that $-2\pi + \beta$ is increasing in β. This proves Theorem 3. □

Notably, the proof of Theorem 3 suggests that for the best strategy α as a function of β, we have that $c_0^1 \le c_2^1(\alpha) \le c_1^1(\alpha)$. This was expected, since having the basic landing outside the fence is intuitively more favourable than having it inside the fence and without needing the remedy phase, which is more favourable than needing the remedy phase. It is also interesting to note that for $\beta \le \gamma$, the best jump choice α_β attains values close to $\beta/2$. This suggests an alternative approach to the problem that does not require the ability to solve technical trigonometric equations, and that will be explored later. Finally, there is a nice suggested recursive relation between costs c_0^1, c_1^1 that is soon to be generalized for k-Jump Algorithms.

4 Generic Description of k-Jump Algorithms

Analogously to the previous sections, the k-Jump Algorithm has parameters $\alpha_1, \ldots, \alpha_k$ and runs similarly to the (k-1)-Jump Algorithm, except from the case that the last landing point of the (k-1)-Jump Algorithm (if this is realized, it will be the (k-1)st-jump landing point) is within the fence. If that happens, the k-Jump Algorithm makes an additional counterclockwise jump corresponding to arc length α_k. If the kth-jump landing point is in the fence, then it runs Algorithm 1. Otherwise, the kth-jump landing point is outside the fence, and the robot applies the remedy phase from Algorithm 2 in Section 3.

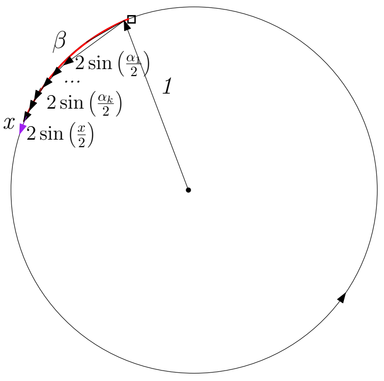
Algorithm 3 k-Jump Algorithm

```
1: Walk to the perimeter of the disk
2:   $i \leftarrow 0$
3: while landing point is inside the fence & $i < k$ do
4:     $i \leftarrow i + 1$
5:     make a ccw chord jump of arc length $\alpha_i$
6: Perform Algorithm 1
```

It is clear from the discussion above that any k-Jump Algorithm is specified by the jump steps $\alpha_1, \alpha_2, \ldots, \alpha_k$, where the ith jump is realized only if the basic landing point, along with the landing points of the previous $i-1$ jumps fall within the fence. In order to preclude the possibility that a jump passes over the area that holds the treasure and bring the robot back to the fence we require that $\alpha_i \leq \beta$. Moreover, for the jumps to be in counterclockwise direction (and to break the symmetry) we also require that $\alpha_i \leq \pi$.

Similarly, for the 1-Jump Algorithm we required that $\alpha_1 \leq \min\{\pi, 2\pi-\beta\}$. However, according to Theorem 3, the optimal jump step is less than β (for all β), meaning that the correctness condition for choosing the jump step could have been replaced by $\alpha_1 \leq \min\{\beta, 2\pi-\beta\}$. Indeed, our intuition tells us that an algorithm, which after the basic landing point within the fence makes a jump more than the length of the fence, will land outside the fence and subsequently will, unavoidably, need to apply the (suboptimal) remedy phase. Motivated by this observation, we require the following condition regarding the step sizes of k-Jump Algorithm's: $\alpha_i \leq \min\{\pi, 2\pi-\beta\}, \quad i = 1, \dots, k$.

The next lemma generalizes Lemma 2 and provides a handy recurrence description of the cost of the k-Jump Algorithm with jump steps $\alpha_1, \dots, \alpha_k$ depending on the first landing point outside the fence. In this direction, we denote by c_t^k to be the worst case cost of the k-Jump Algorithm when the basic landing point along with the landing points of the first $t-1$ jumps fall all inside the fence and the robot lands outside the fence in the the tth jump, which is shown in Figure 3. Let us observe that, c_0^k is the cost of the case when the basic landing point is outside the fence, while c_{k+1}^k corresponds to the case that the landing points of all k jumps, as well as the basic landing point, fall inside the fence.



β
$2\sin\left(\frac{\alpha_1}{2}\right)$
$\dots$
$2\sin\left(\frac{\alpha_k}{2}\right)$
x
$2\sin\left(\frac{x}{2}\right)$
1

Figure 3: k-Jump Algorithm

Lemma 4 *For any β, let $\alpha_0 = \beta$. Depending on the landing points, the cost of the k-Jump Algorithm with jump steps $\alpha_1, \dots, \alpha_k$ is*

$$\begin{cases} c_0^k & := 1 + 2\pi - \alpha_0 + 2\sin(\alpha_0/2) \\ c_t^k & := c_{t-1}^k + 4\sin(\alpha_t/2) - 2\sin(\alpha_{t-1}/2) \\ c_{k+1}^k & := 1 + 2\pi - \sum_{i=1}^k (\alpha_i - 2\sin(\alpha_i/2)) \end{cases} \quad (3)$$

with the understanding that c_t^k are functions on β and $\alpha_1, \dots, \alpha_{t-1}$, for $t = 1, \dots k+1$.

5 The Halving Heuristic k-Jump Algorithm

In this section, we present a simple heuristic that is meant to approximate the optimal jump steps without relying on solutions of trigonometric optimization problems. Most importantly, our algorithm requires very limited memory and does not need to perform numerical operations other than simple algebraic manipulations. In fact, there are only constant many operations needed to determine every possible jump size. Moreover, parameter k, i.e. the number of jumps, may not necessarily be determined in advance, and is allowed to be even infinite. First, we present the algorithm and analyze it. Then, in Section 6, we contrast it to the Optimal k-Jump Algorithm (for certain values of k).

Closely examining the optimal solution for the 1-Jump Algorithm in Section 3, we are tempted to choose an alternative first jump step equal to $\beta/2$, which is a good approximation to α_β. This choice is valid, as long as the jump does not exceed $2\pi-\beta$, and indeed for large enough values of β, i.e. for $\beta \geq 4.041$, as per Theorem 3, the best choice for just one jump is $2\pi-\beta$. Note that changing the first jump from α_β to $\beta/2$ results to a new threshold value $\frac{4}{3}\pi \approx 4.188$ after which the first jump should become $2\pi-\beta$. Interestingly, the pattern repeats also in the optimal k-Jump Algorithms (see Section 6).

The previous observation suggests a natural heuristic for k-Jump Algorithms. First, go to an arbitrary point on the circle. While in some unknown position on the fence, make a valid jump (i.e. no more than $\pi, 2\pi-\beta$) equal to half of the unexplored fence, unless this value exceeds $2\pi-\beta$ in which case the jump should be $2\pi-\beta$. Formally, the description of the heuristic follows if we can determine the length of the chord-jump α_i in every i-th jump, and then invoke Algorithm 3.

Algorithm 4 Halving Heuristic jumps

```
1: $explored \leftarrow 0$
2: $temp \leftarrow \frac{\beta - explored}{2}$
3: **if** $temp \leq 2\pi - \beta$ **then**
4:     $\alpha_i \leftarrow temp$
5: **else**
6:     $jump \leftarrow 2\pi - \beta$
7: $explored \leftarrow explored + jump$
```

Note that the calculations of jumps α_i can be incorporated within Algorithm 3 and do not need to be computed in advance. As the maximum number of jumps can be part of the input, Algorithm 4 can be performed only for k many landings within the fence (see Algorithm 3), or as long as the the jump step does not drop below a given threshold. Interestingly, the definition of step sizes on the fly by Algorithm 4 even allows for $k = \infty$. That would correspond to the theoretical case that the robot makes an infinite number of jumps for which all landings happen within the fence. Still, the time for the robot to reach the endpoint of the fence would be finite (Zeno's paradox).

The process above fully determines the jump step of the t-th jump as a function of β, for every $t = 1, \dots, k$, and for every k. In what follows we provide an analytic description of these values so that we can analyze the performance of the algorithm. The lemma below will allow us to derive later a nicer closed formula for the jump steps of the halving Algorithm.

Lemma 5 *Let $h_t = \frac{2\pi(t+1)}{t+2}$ for $t \geq 1$ and $h_0 = 0$. For any $\beta \in (0, 2\pi)$, the value of the i-th jump in the Halving Algorithm equals*

$$\alpha_i = \begin{cases} \frac{j\beta-(j-1)2\pi}{2^{i-j+1}} & \text{if } \beta \leq h_i \text{ and } \beta \in (h_{j-1}, h_j] \\ 2\pi - \beta & \text{if } \beta > h_i \end{cases}$$

We are now ready to present a closed formula for the jump steps of the Halving Algorithm along with its performance, as a function on the number of jumps.

Theorem 6 *For any $\beta \in (0, 2\pi)$, let $\rho_\beta := \max\left\{\frac{2\beta-2\pi}{2\pi-\beta}, 1\right\}$. The value of the i-th jump ($i \geq 1$) in the Halving Algorithm equals*

$$\alpha_i = \begin{cases} 2\pi - \beta & \text{if } i < \rho_\beta \\ \frac{2\pi-\lceil\rho_\beta\rceil(2\pi-\beta)}{2^{i-\lceil\rho_\beta\rceil+1}} & \text{otherwise} \end{cases}$$

Proof. According to Lemma 5, $\beta > h_i$ is satisfied as long as $i < \frac{2\beta-2\pi}{2\pi-\beta}$. Hence, if $i < \rho_\beta$ we have $\alpha_i = 2\pi - \beta$. For the same reason $\beta \leq h_i$ if and only if $i \geq \rho_\beta$, so ρ_β is the smallest integer for which $\beta \leq h_{\rho_\beta}$, meaning that $\beta \in (h_{\rho_\beta - 1}, h_{\rho_\beta}]$. Therefore, again by Lemma 5, we set $j = \lceil\rho_\beta\rceil$ (and rearrange the terms) to derive the promised formula. This proves Theorem 6. □

In Figure 4, we depict the behaviour of the decreasing sequence α_i (in i) as a function of β for a $i = 1, 2, 3, 4, 5$ and 6. Notably, for every k there is some threshold value of β, after which $\alpha_i = 2\pi - \beta$ for all $i \leq k$.

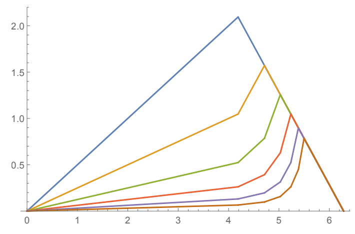


Figure 4: The plot jumps $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_6$ of the Halving Algorithm as a function of β.

Finally, we use the closed formula for the jump steps to derive a closed formula for the cost of the Halving k-Jump Algorithm. The main idea for the proof is to show that the worst case for the algorithm is when all k jumps fall within the fence, and that the performance is strictly decreasing in k. This is what the next lemma establishes.

Lemma 7 *The Halving $k-$JumpAlgorithm incurs the maximum possible cost when all jump landings (including the basic one) fall within the fence.*

We are ready to conclude with the cost of the Halving k-Jump Algorithm.

Theorem 8 *The cost of the Halving k-Jump Algorithm is strictly decreasing with k and it equals*

$$1 + 2\pi + \frac{\lceil\rho_\beta\rceil(2\pi - \beta) - 2\pi}{2^{k-\lceil\rho_\beta\rceil+1}} + 2(\lceil\rho_\beta\rceil - 1)\sin(\beta/2) + 2\sum_{i=0}^{k-\lceil\rho_\beta\rceil} \sin\left(\frac{\alpha_{\lceil\rho_\beta\rceil}}{2^{i+1}}\right),$$

where $\rho_\beta = \max\left\{\frac{2\beta-2\pi}{2\pi-\beta}, 1\right\}$ and $\alpha_{\lceil\rho_\beta\rceil} = \pi - \frac{\lceil\rho_\beta\rceil}{2}(2\pi - \beta)$.

Proof. Using the terminology of Lemma 4, and by Lemma 7, the cost of the Halving k-Jump Algorithm equals

$$c^k_{k+1} = 1 + 2\pi - \beta - \sum_{i=1}^{k}(\alpha_i - 2\sin(\alpha_i/2))$$

where the jump steps are as determined in Theorem 6. From the expression above, it is immediate that the cost is strictly increasing in k, as long as all jump steps are positive. Next, we compute the summation in parts. We have,

$$\begin{aligned} &\beta + \sum_{i=1}^{k}\alpha_i \\ &= \beta + \sum_{i=1}^{\lceil\rho_\beta\rceil-1}\alpha_i + \sum_{i=\lceil\rho_\beta\rceil}^{k}\alpha_i \\ &= \beta + (\lceil\rho_\beta\rceil - 1)(2\pi - \beta) + \sum_{i=0}^{k-\lceil\rho_\beta\rceil}\frac{\alpha_{\lceil\rho_\beta\rceil}}{2^i} \\ &= \beta + (\lceil\rho_\beta\rceil - 1)(2\pi - \beta) + \alpha_{\lceil\rho_\beta\rceil}\left(2 - \frac{1}{2^{k-\lceil\rho_\beta\rceil}}\right) \\ &= \beta + (\lceil\rho_\beta\rceil - 1)(2\pi - \beta) + \left(\pi - \frac{\lceil\rho_\beta\rceil}{2}(2\pi - \beta)\right) \cdot \\ &\quad \left(2 - \frac{1}{2^{k-\lceil\rho_\beta\rceil}}\right) \\ &= \frac{\lceil\rho_\beta\rceil(2\pi - \beta) - 2\pi}{2^{k-\lceil\rho_\beta\rceil+1}}. \end{aligned}$$

Finally,

$$\sum_{i=1}^{k} \sin(\alpha_i/2) = \sum_{i=1}^{\lceil \rho_\beta \rceil - 1} \sin(\alpha_i/2) + \sum_{i=\lceil \rho_\beta \rceil}^{k} \sin(\alpha_i/2)$$
$$= (\lceil \rho_\beta \rceil - 1)\sin(\beta/2) + \sum_{i=0}^{k-\lceil \rho_\beta \rceil} \sin\left(\frac{\alpha_{\lceil \rho_\beta \rceil}}{2^{i+1}}\right)$$

Putting the two expression together gives the promised cost. This proves Theorem 8. □

Figure 5 summarizes the cost of the Halving k-Jump Algorithm for $k = 1, 2, 3$ and 4.

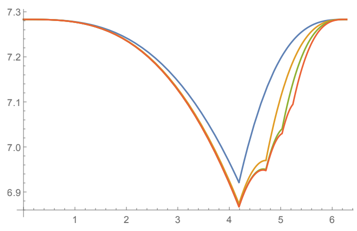


Figure 5: The performance of the Halving Algorithm for 1,2,3 and 4 jumps (decreasing in the number of jumps, respectively) as a function of β.

6 Some Optimal k-Jump Algorithms & Comparison

It is apparent from Lemma 4 that choosing the optimal jump steps $\alpha_1, \dots, \alpha_k$ amounts to solving the involved optimization problem $\min_{\alpha_1,\dots,\alpha_k} \max_{t=1,\dots,k+1}\{c_t^k\}$, where $\alpha_i \le \min\{\pi, 2\pi - \beta\}$. In this section, we contrast the choices and performance of the Halving k-Jump Algorithm with the choices and performance of the Optimal k-Jump Algorithm that was obtained using optimization software packages, for $k \le 3$ (except from $k = 1$ whose formal analysis appears in Section 3). Our findings are summarized in the figures below.

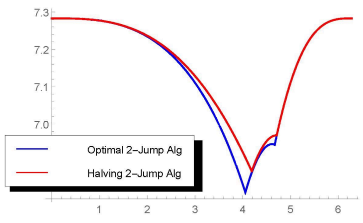


Figure 6: Performance comparison between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β.

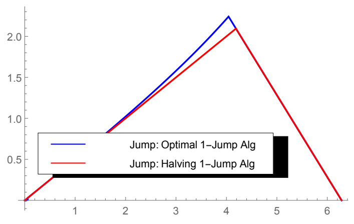


Figure 7: Comparison of jump choices between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β.

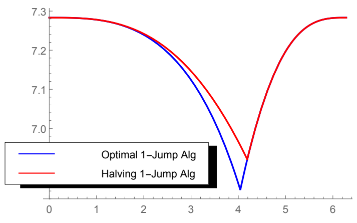


Figure 8: Performance comparison between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β.

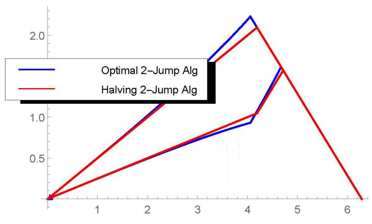


Figure 9: Comparison of the two jump choices between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β. The first jump of each algorithm is always no smaller than the second one, and eventually they all attain the value $2\pi - \beta$.

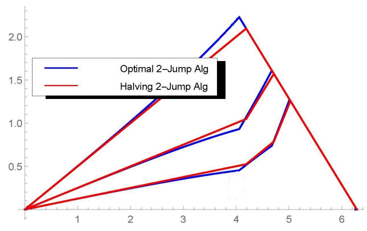


Figure 10: Comparison of the three jump choices between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β. For both algorithms, the first jump of each algorithm is always no smaller than the second one, which is no smaller than the third one. Eventually they all attain the value $2\pi - \beta$.

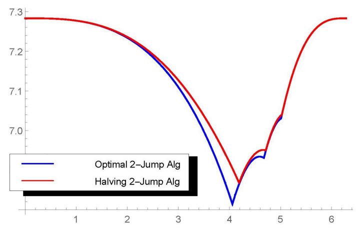


Figure 11: Performance comparison between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β. Notably, performance is nearly the same for all values of $\beta > 5$. The bigger discrepancy is observed for values of β close to 4, for which also the jump steps between the two algorithms exhibit the larger gaps (see Figure 10).

For $k = 1, 2, 3$ we numerically compute the optimal k-Jump Algorithm (note that for $k = 1$, the rigorous analysis appears in Section 3). Then, we contrast the performance of the optimal and of the Halving algorithm (for the same number of jumps), as well as contrasting their corresponding jump steps. The numerical calculations indicate that the choices of the Halving algorithm are nearly optimal for a wide spectrum of β (with the largest discrepancy for $\beta \approx \gamma$). Interestingly, for larger values of β, the choices of the Halving algorithm are nearly optimal that also reflects on the cost of the two algorithms which becomes nearly identical. More importantly, experiments indicate that for large values of β, the optimal choices for k jump steps is to make all equal to $2\pi - \beta$, which is also the choice of the Halving algorithm.

7 Conclusion

In this paper we investigated a new search problem for a mobile robot to find a stationary target placed at an unknown location at distance 1, in the presence of a fence placed on the perimeter of a unit disc. First we determined the optimal 1-Jump algorithm for the robot to find the target. Then we provided a generic description of k-Jump algorithms and analyzed their cost depending on the jump landings. Subsequently, we analyzed the Halving k-Jump algorithms, where k is the max number of jumps the robot makes so as to overcome the fence and find the target. Several interesting questions remain open, when e.g., there are multiple fences on the perimeter of the disc, and the robot's speed changes when traversing a fence.

References

[1] S. Badal, S. Ravela, B. Draper, and A. Hanson. A practical obstacle detection and avoidance system. In *Appl. of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 97–104. IEEE, 1994.

[2] R. Baeza-Yates and R. Schott. Parallel searching in the plane. *Computational Geometry*, 5(3):143–154, 1995.

[3] R. A. Baezayates, J. C. Culberson, and G. J. Rawlins. Searching in the plane. *Information and computation*, 106(2):234–252, 1993.

[4] A. Beck. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964.

[5] R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.

[6] S. Benkoski, M. Monticino, and J. Weisinger. A survey of the search theory literature. *Naval Research Logistics (NRL)*, 38(4):469–494, 1991.

[7] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *STOC 1991*, pages 494–504. ACM, 1991.

[8] J. Czyzowicz, L. Gasieniec, T. Gorry, E. Kranakis, R. Martin, and D. Pajak. Evacuating robots via unknown exit in a disk. In *DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 122–136, 2014.

[9] J. Czyzowicz, K. Georgiou, E. Kranakis, L. Narayanan, J. Opatrny, and B. Vogtenhuber. Evacuating robots from a disk using face-to-face communication (extended abstract). In *CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, pages 140–152, 2015.

[10] J. Dobbie. A survey of search theory. *Operations Research*, 16(3):525–537, 1968.

[11] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Mobile search for a black hole in an anonymous ring. In *DISC 2001, Lisbon, Portugal, October 3-5, 2001, Proceedings*, pages 166–179, 2001.

[12] S. R. Finch and J. E. Wetzel. Lost in a forest. *The American Math. Monthly*, 111(8):645–654, 2004.

[13] K. Georgiou, E. Kranakis, and A. Steau. Searching with advice: Robot fence-jumping (http://arxiv.org/abs/1606.08023), 2016.

[14] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001.

[15] J. Isbell. An optimal search pattern. *Naval Research Logistics Quarterly*, 4(4):357–359, 1957.

[16] B. O. Koopman. The theory of search: III. the optimum distribution of searching effort. *Operations research*, 5(5):613–626, 1957.

[17] C. Pozna, F. Troester, R.-E. Precup, J. K. Tar, and S. Preitl. On the design of an obstacle avoiding trajectory: Method and simulation. *Mathematics and Computers in Simulation*, 79(7):2211–2226, 2009.

[18] L. D. Stone and H. R. Richardson. Search for targets with conditionally deterministic motion. *SIAM Journal on Applied Mathematics*, 27(2):239–255, 1974.

On the Stability of Medial Axis of a Union of Disks in the Plane

David Letscher[*] Kyle Sykes[†]

Abstract

We show that the medial axis of union of disks in the plane is stable provided that the topology is preserved and every disk meets the boundary in a single arc. If the second condition is removed, the medial axis is no longer stable, but if pruned using any of four significance measures (circumradius, erosion thickness, object angle or potential residue) it remains stable.

1 Introduction

The medial axis of a shape has proven to be one of the most useful tools in computational geometry. One of its most important properties is that it represents the full topology of the shape [5]. For most shape representations there are no known algorithms to calculate the medial axis exactly. The only context where it can be calculated efficiently for sizable datasets is when the shape is a union of balls [2]. Because of this, most practical algorithms that utilize the medial axis do so for shapes represented as a finite union of balls.

For general shapes, it is well known that the medial axis is unstable under small perturbations. One approach to stability has been to deal with subsets of the medial axis, such as the λ-medial axis, which can be shown to represent the topology of the shape and be stable under small perturbations of the shape [4].

In this paper, we consider a different approach; we restrict our family of shapes to be a union of disks in the plane and study how the medial axis can change when disks move and change size. The one restriction that we will use thoughout is that the topology of the shape will not be allowed to change.

First we will show that if we add the additional assumption that each disk meets the boundary of the shape in at most one arc then the medial axis is stable, i.e. the medial axis changes continuously in the Hausdorff metric.

The second result applies to several significance measures on the medial axis. If we allow disks to change their intersection patterns with the boundary but not the topology of the shape, the medial axis can change substantially but the significance measures are still stable. This implies that is we were to truncate the medial axis using these significance measures, similar to the λ-medial axis, then the truncated medial axis change continuously in the Hausdorff metric.

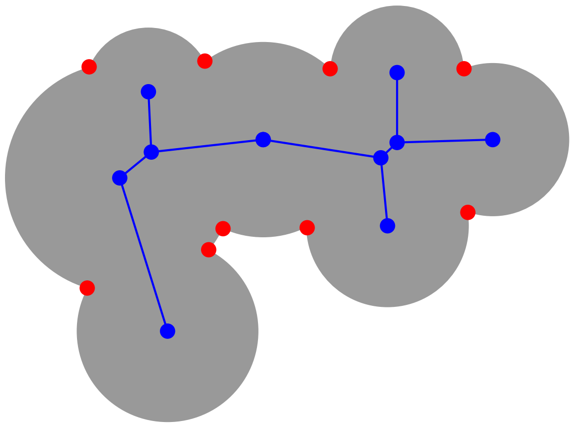


Figure 1: The medial axis of a union of disks. The singular points and vertices of the medial axis are marked.

2 Medial Axis of a Union of Disks

Both Attali and Montanvert [3] and Amenta and Kolluri [2] have characterized the medial axis of a union of balls. A simple modification of either yields the following characterization of the medial axis of a union of disks in the plane. It utilizes the Voronoi diagram of the *singular points*, the points of intersection of the boundary of two or more disks. These points are precisely where the boundary of the union of disks fails to be differentiable.

Lemma 1 *Suppose that $X \subset \mathbb{R}^2$ is a union of disks that is also a surface with boundary then the medial axis of X is a graph where*

- *Vertices are either vertices of the Voronoi diagram of the singular points of X or centers of disks with two or more singular points on their boundaries.*
- *Edges between vertices connect two vertices and are equidistant from two singular points with no closer singular point.*

[*]Department of Mathematics and Computer Science, Saint Louis University, letscher@slu.edu. Research partially supported by NFS grant IIS-1319944.

[†]Department of Mathematics and Computer Science, Saint Louis University, ksykes2@slu.edu. Research partially supported by NFS grant CCF-1054779.

Definition 2 *A union of disks, $X \subset \mathbb{R}^2$, is* generic *if*

- *The disks have positive radii and distinct centers.*
- *No three centers are co-linear.*
- *X is a manifold with boundary.*
- *No singular point lies on the boundary of three or more disks.*
- *Any disks contained in X that has four or more singular points on its boundary is one of the disks comprising the union.*

Otherwise, the union of disks is called degenerate.

For generic unions of disks, the vertices of its medial axis are either centers of disks or Voronoi vertices with exactly three adjacent edges. See Figure 1 for an example.

3 Configurations of Unions of Disks

We will distinguish between the set of disks and the shape their union forms. A single disk in $\mathbb{R}^2$ can be represented as a tuple (x, y, r) storing the disk's center and radius (which is required to be non-negative). The set of configurations of n disks in $\mathbb{R}^2$ will be denoted $\mathcal{C}_n$ which is a subset of $\mathbb{R}^{3n}$. For a configuration $\alpha \in \mathcal{C}_n$, we will denote the corresponding shape $X_\alpha \subset \mathbb{R}^2$ as the union of the disks specified by α.

Consider a path γ in $\mathcal{C}_n$ and the corresponding unions of disks. As the disks move around and change sizes there are two ways the medial axis can change. The first is that singular points could appear or disappear. The second is the Voronoi diagram of the singular points changes. In the study of dynamic Voronoi diagrams, it has been shown that if points move continuously in the plane then the Voronoi diagram changes continuously in the Hausdorff metric and it combinatorics change when more than three points lie on a circle simultaneously [1].

Each possible change to the medial axis corresponds to a particular degenerate configuration of the disks. We will show that any path in the configuration space can be infinitesimally perturbed to hit these degeneracies in isolated points and only one at a time. By studying what happens at each of these degeneracies we will be able to understand the stability of the medial axis.

The degenerate configurations correspond to each of the conditions in definition 2. Each can be described by a set of polynomial constraints, shown below. Assume that $\{(x_i, y_i)\}$ are the centers of the disks and $\{r_i\}$ are their radii. We will use (s_j, t_j) to represent the coordinates of a singular point.

Zero radius disk Some fixed disk has zero radius.

$$r_i = 0 \text{ for some } i$$

Centers coinciding Two fixed disks have identical centers.

$$(x_i, y_i) = (x_j, y_j) \text{ for some } i \neq j$$

Co-linear centers Three of more centers of circles are co-linear.

$$\begin{vmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{vmatrix} = 1 \text{ for some distinct } i, j, k$$

Co-tangent disks The boundaries of two specified disks are tangent to each other.

$$(x_i - x_j)^2 + (y_i - y_j)^2 = (r_i \pm r_j)^2 \text{ for some } i, j$$

Triple point A singular point is the intersection of the boundary of three of the disks.

$$(x_i - x)^2 + (y_i - x)^2 = r_i^2, \forall i \in I$$

where $|I| = 3$ and (x, y) are the coordinates of the triple point

Co-circular singular points There are four specified singular points that lie on a circle that is not the boundary of one of the disks in the configuration.

$$\begin{aligned} (x_i - s_j)^2 + (y_i - t_j)^2 &= r_i^2 \\ (x - s_j)^2 + (y - t_j)^2 &= r^2 \end{aligned}$$

Where each pair (i, j) determines which circles the singular points lie on and (x, y) is the center of the circle of radius r the singular points lie on

Since solutions to polynomial systems have zero measure you can immediately show the following.

Proposition 3 *For any configuration $\alpha \in \mathcal{C}_n$ and $\epsilon > 0$ there exists a generic configuration $\beta \in \mathcal{C}_n$ such that $d(\alpha, \beta) < \epsilon$.*

The results in this paper utilize two main techniques. The first, allows us to perform infinitesimal perturbations of paths in the configuration space to ones that are generic almost everywhere along their length. The medial axis changes smoothly between these few degenerate configurations. By analyzing what happens to the medial axis at each of these degeneracies we will be able to see exactly how the medial axis can change and what happens to several significance measures.

Proposition 4 *For any path $\gamma : [0, 1] \to \mathcal{C}_n$ with end-points being generic configurations and $\epsilon > 0$ there exists a path $\gamma_\epsilon[0, 1] \to \mathcal{C}_n$ such that*

- $\gamma_\epsilon(0) = \gamma(0), \gamma_\epsilon(1) = \gamma(1)$
- $\text{length}(\gamma_\epsilon) < \text{length}(\gamma) + \epsilon$
- $d_H(\gamma_\epsilon, \gamma) < \epsilon$
- *For all but finitely many values of t, $\gamma(t)$ is a generic configuration and at those values the configuration has exactly one degeneracy.*

If we define $\mathcal{G}_n \subset \mathcal{C}_n$ to be all configurations that have at most one degeneracy, then the proposition implies that

$$d_{\mathcal{G}_n}(\alpha, \beta) = d_{\mathcal{C}_n}(\alpha, \beta)$$

So instead of having to worry about configurations with multiple degeneracy, we can assume that we only have configurations that are either generic or have exactly one degeneracy. The major steps in the proof are:

1. Perturb the path so that no disk has zero radius.
2. The set of configurations where two centers coincide is a co-dimension 2 linear subspace of the configuration space. Any path can be perturbed off of all of these subspaces.
3. The configurations that do not satisfy the other genericity conditions form an algebraic variety in a possibly extended set of coordinates. It can be shown that the singular points of these varieties and the intersections of these varieties are all co-dimension 2 or higher and can be avoided.
4. Finally, the path can be perturbed to intersect each of the manifold regions of these varieties transversely.

4 Changes to the Medial Axis at Degenerate Configurations

Figure 2 shows the changes to the medial axis at each type of degenerate configuration. The figures are slightly before, at and slightly after the degenerate configuration. Only configurations where the topology of the union of disks is unchanged and where the medial axis combinatorics are changed. At all other degeneracies the medial axis moves continuously.

The first of the three cases deals with co-tangent disks. Notice that this tangency occurs with one disk contained in another. If they were adjacent then either the point would be in the interior of another disk or a change in topology would occur. In the first case there is not change in the combinatorics of the medial axis and the second violates our assumptions. When one disk is contained in another, immediately after the disk emerges a new edge is created. This edge goes between the centers of the two disks.

In the second case in the figure, a disk emerges from the union of disks passing through a singular point in the boundary. Similar to the previous case a new edge appears instantaneously. The edge goes between the center of an emerging disk and a newly formed Voronoi vertex. In the degenerate configuration this Voronoi vertex lies on a pre-existing edge that is adjacent to the singular point the disk is passing through.

In the final case, the singular points change positions but do not appear or disappear. This case is covered in the literature on dynamic Voronoi diagrams [1].

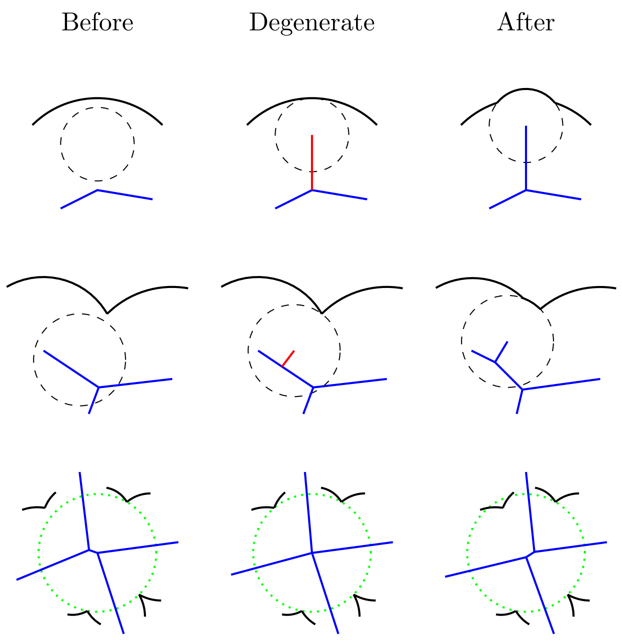


Figure 2: The changes in the medial axis for (a) cotangent disks, (b) triples points and (c) four circular singular points. The relevant portion of the medial axis is shown in blue. The red edges represent edges of the medial axis that appear instantaneously after the moving circle meets the boundary.

One edge of the medial axis collapses to a point and a different edge emerges. Throughout this combinatorial change, the medial axis changes continuously in the Hausdorff metric.

If we are willing to change the shape slightly, then we can assume each hits the boundary in a single arc. If this property is maintained through a perturbation singular points never appear or disappear and the only changes in the combinatorics of the medial axis are edges shrinking to a point and the reverse. Away from these changes in combinatorics, the disks, singular points and vertices of the medial axis all move continuously. This shows that, in the Hausdorff metric, the union of disks change continuously.

Theorem 5 *If $B \subset \mathcal{G}_n$ consists of configurations of disks where every disk meets the boundary of the shape in at most a single arc, then the medial axis of X_α varies continuously in the Hausdorff metric as α varies continuously over configurations in B.*

It is worth noting that the analogous result is not true in 3-dimensions.

5 Stability of Significance Measures

We will examine several significance measures on the medial axis. These are all real valued functions, with larger values corresponding to more significant regions. These measures we study are circumradius, object angle [9], potential residue [7] and erosion thickness [6, 8]. Note that the λ-medial axis uses circumradius as its significance measure to prune the medial axis.

Consider a simply connected shape $X \subset \mathbb{R}^2$ and a point x on the medial axis M. The significance measures can be defined as follows:

Circumradius $R(x)$ is the minimum radius of a disk in X containing the two closest points on the boundary to x.

Object angle $OA(x)$ is the angle between the vectors from x to the two closest points on ∂X to x.

Potential residue $PR(x)$ is the distance on ∂X between the two closest points on ∂X to x.

Erosion thickness We choose a definition equivalent to the one in [6], $ET(x)$ is defined as

$$\max\{\min\{d_M(x,l)+D(l), d_M(x,l')+D(l')\}-D(x)\}$$

where $d_M(x,y)$ is distance measured on the medial axis, $D(x) = d(x, \partial X)$ and the maximum is taken over all leaves l, l' such that x is on the path between the two leaves.

As a shape is perturbed, we will show that each of these significance measures changes continuously as the vertices move. In particular, when new spurs are created in the medial axis, see Figure 2, all of these significance measures are zero on the newly created points. We will adapt the Hausdorff metric to measure the differences between functions with different domains.

Definition 6 *Consider two sets $X_1, X_2 \subset \mathbb{R}^2$ and functions $f_i : X_i \to \mathbb{R}$, the extended Hausdorff distance $d_{EH}((X_1, f_1), (X_2, f_2))$ is defined as $d_H(Y_1, Y_2)$ where $Y_i = \{(x, f_i(x)) \mid x \in X_i\} \cup \big(\mathbb{R}^2 \times \{0\}\big)$.*

Notice that two spaces are considered close in this extended Hausdorff distance if every point has a small function value or it is close to a point on the other set that has a similar function value.

We can examine what happens to each of these significant measure as the union of disks changes under the assumption that there is no change in the topology of the union of disks. There are two ways an edge could disappear: shrinking continuously to a point and disappearing when two disks become cotangent. An edge appears in the reverse of either of these processes. See Figure 2 for examples. Consider a point x on one of these edges that appear instantaneously. Immediately after the edge appears, there are two singular points that are closest to the edge. As you get closer to the time where the edge appears, these singular points merge. In the limit cirucumradius, object angle and potential residue are all identically zero on the edge. Erosion thickness is also zero on the edge at the time the edge appears since the shortest path from any point on the edge to the boundary goes through a leaf of the medial axis.

Away from these degenerate configurations the singular points and the medial axis all move continuously. It is easily shown that all four significance measures also change continuously with the changes in the shape. This yields the following theorem.

Theorem 7 *If f is either circumradius, object angle, potential residue or erosion thickness then $\mathcal{MA}(X_\alpha)$ varies continuously as α moves continuously in $\mathcal{G}_n$.*

Corollary 8 *For every $t \geq 0$,*

$$M_{\alpha,t} = \{x \in \mathcal{MA}(X_\alpha) \mid f(x) \geq t\}$$

varies continuously in the Hausdorff metric as α moves continuously in $\mathcal{G}_n$.

This corollary implies that if you truncate the medial axis using any of these significance measures then it changes continuously as the union of disks are perturbed. An example of this is shown in Figure 3 using erosion thickness. It is worth noting that for arbitrary truncation values, only erosion thickness is known to preserve topology. This corollary is already known for sufficiently small thresholds for circumradius since in this case $M_{\alpha,t}$ is the λ-medial axis.

Future Work

There are several natural extensions that are worth considering. These include considering more general shapes and extending to three dimensions. Another nice project would be building kinetic data structure to efficiently track the changes in the medial axis as the balls move around.

Acknowledgments

We would like to thank Erin Chambers, Tao Ju and the anonymous reviewers for helpful suggestions on this work.

References

[1] G. Albers, L. J. Guibas, J. S. Mitchell, and T. Roos. Voronoi diagrams of moving points. *International Journal of Computational Geometry & Applications*, 8(03):365–379, 1998.

[2] N. Amenta and R. K. Kolluri. The medial axis of a union of balls. *Computational Geometry*, 20(1):25–37, 2001.

[3] D. Attali and A. Montanvert. Computing and simplifying 2d and 3d continuous skeletons. *Computer Vision and Image Understanding*, 67(3):261–273, 1997.

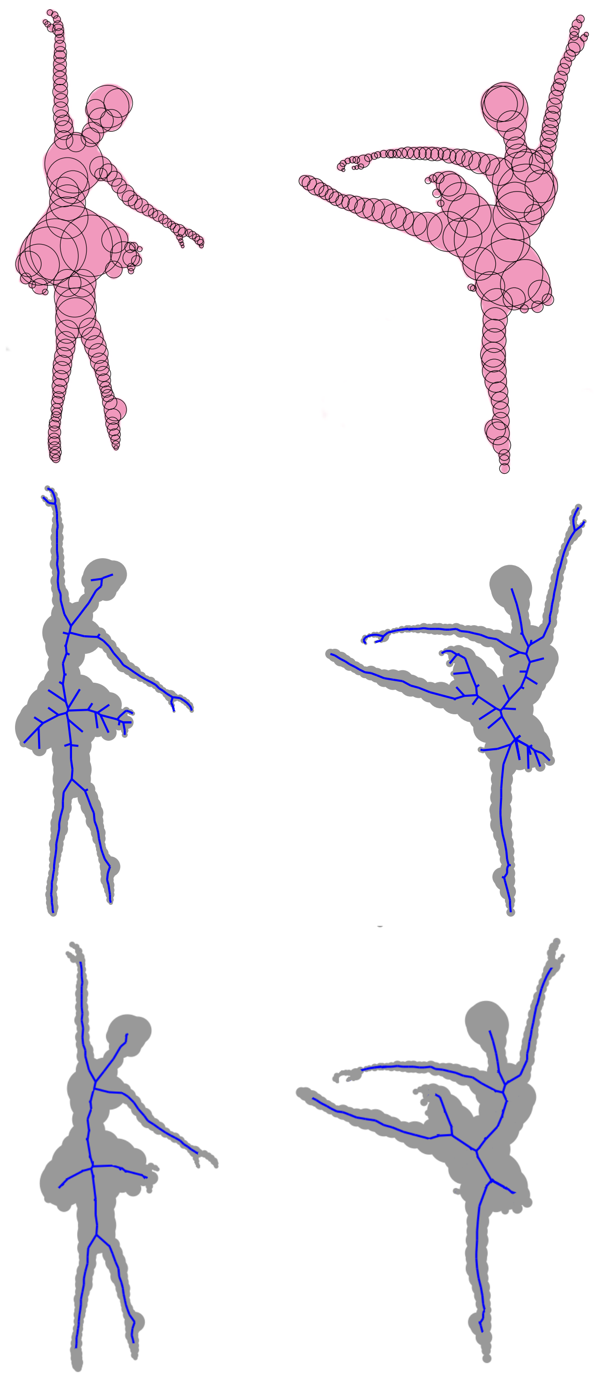


Figure 3: Two poses of a ballerina. Top row: the union of disks representing the shapes. Middle row: the medial axes. Bottom row: the medial axes pruned using erosion thickness. As the dancer moves these pruned medial axes transform continuously (in the Hausdorff metric) from one to the other.

[4] F. Chazal and A. Lieutier. Stability and homotopy of a subset of the medial axis. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 243–248. Eurographics Association, 2004.

[5] A. Lieutier. Any open bounded subset of R^n has the same homotopy type as its medial axis. *Computer-Aided Design*, 36(11):1029–1046, 2004.

[6] L. Liu, E. W. Chambers, D. Letscher, and T. Ju. Extended grassfire transform on medial axes of 2d shapes. *Computer-Aided Design*, 43(11):1496–1505, 2011.

[7] R. Ogniewicz and M. Ilg. Voronoi skeletons: Theory and applications. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 63–69. IEEE, 1992.

[8] D. Shaked and A. M. Bruckstein. Pruning medial axes. *Computer vision and image understanding*, 69(2):156–169, 1998.

[9] A. Sud, M. Foskey, and D. Manocha. Homotopy-preserving medial axis simplification. *International Journal of Computational Geometry & Applications*, 17(05):423–451, 2007.

kth Nearest Neighbor Sampling in the Plane[*]

Kirk P. Gardner
University of Connecticut
kirk.gardner@uconn.edu

Donald R. Sheehy
University of Connecticut
don.r.sheehy@gmail.com

Abstract

Let B be a square region in the plane. We give an efficient algorithm that takes a set P of n points from B, and produces a set $M \subset B$ with the property that the distance to the second nearest point in M approximates the distance to the kth nearest point of P. That is, there are constants $\alpha, \beta \in \mathbb{R}$ such that for all $x \in B$, we have $\alpha \mathbf{d}_{P,k}(x) \le \mathbf{d}_{M,2}(x) \le \beta \mathbf{d}_{P,k}(x)$, where $\mathbf{d}_{M,2}$ and $\mathbf{d}_{P,k}$ denote the second nearest and kth nearest neighbor distance functions to M and P respectively. The algorithm is based on Delaunay refinement. The output set M also has the property that its Delaunay triangulation has a lower bound on the size of the smallest angle. The primary application is in statistical density estimation and robust geometric inference.

1 Robust Sizing Functions

Since the pioneering work of Chew [3] and Ruppert [9], Delaunay refinement has remained an important approach to mesh generation (see for example the book [2]). The algorithm: Starting from the Delaunay triangulation of the input points P (restricted to a bounding box B), repeatedly add the circumcenter of any triangle whose circumradius is more than a constant times larger than the length if its shortest edge.[1] Such a triangulation is said to have *bounded radius-edge ratio* and will be referred to as a *quality mesh* and will necessarily also have a lower bound on the size of the smallest angle. Ruppert showed that not only does this remarkably simple algorithm terminate, it produces a point set that is asymptotically optimal in size [9].

The key to Ruppert's analysis is the so-called *feature size function*, which for a point set P is the distance to the second nearest point of P, denoted $\mathbf{d}_{P,2} : \mathbb{R}^2 \to \mathbb{R}$. There is a constant γ such that output set M has the property that

$$\gamma \mathbf{d}_{P,2} \le \mathbf{d}_{M,2} \le \mathbf{d}_{P,2}.$$

The optimality of the approach comes from proof that if M is the vertex set of a quality mesh containing P, then

$$|M| = \Theta\left(\int_{x\in B} \frac{dx}{\mathbf{d}_{P,2}(x)^2}\right).$$

The preceding integral defines a measure, whose density is related to the *second nearest neighbor density estimator* used in statistics. It is a useful feature of Delaunay refinement that it reveals this function without explicitly computing or estimating it. However, for the convergence of such estimators, one must generally choose k as a function of $n = |P|$ so that $k(n)/n \to 0$ and $k(n)/log(n) \to \infty$ as $n \to \infty$ [5]. So, for example, taking $k(n) = \sqrt{n}$ is reasonable and sufficient. Motivated by this relationship between mesh generation and density estimation, we address the following problem.

Problem 1 *Given a set P in a bounding box $B \subset \mathbb{R}^2$, find a quality mesh with vertex set $M \subset B$ such that*

$$\alpha \mathbf{d}_{P,k} \le \mathbf{d}_{M,2} \le \beta \mathbf{d}_{P,k}$$

for some constants α and β.

We show how to solve this problem using a variation of Delaunay refinement.

2 Background

We will denote the Euclidean distance between points $a, b \in \mathbb{R}^2$ as $\|a-b\|$. For any set $S \subset \mathbb{R}^2$ and integer $k \ge 1$, define $\mathbf{d}_{S,k} : \mathbb{R}^2 \to \mathbb{R}$ so that $\mathbf{d}_{S,k}(x)$ is the distance to the kth nearest point of S to $x \in \mathbb{R}^2$. Formally, letting $\binom{S}{k}$ denote the set of k element subsets of S,

$$\mathbf{d}_{S,k}(x) := \min_{U\in\binom{S}{k}} \max_{y\in U} \|x-y\|.$$

The distance from a point $x \in \mathbb{R}^2$ to a set S is $\mathbf{d}_{S,1}(x)$ and will be denoted $\mathbf{d}_S(x)$.

We define the ball centered at a point $x \in \mathbb{R}^2$ with radius r as

$$\texttt{ball}(x, r) := \{y \in \mathbb{R}^2 \mid \|x-y\| \le r\}.$$

The smallest ball centered at a point $x \in \mathbb{R}^2$ containing k points in a set S will be denoted

$$\texttt{ball}_{x,S_k} := \texttt{ball}(x, \mathbf{d}_{S,k}(x)).$$

[*]This work was partially supported under grants CCF-1464379 and CCF-1525978.

[1]Ruppert's analysis also works for more general inputs including line segments as well.

For a set $S \subset \mathbb{R}^2$ we will define a triangle with vertices $u, v, w \in S$ as the convex closure of the points u, v, w:

$$t_{u,v,w} := \{au + bv + cw \mid a + b + c = 1,\ a, b, c \in \mathbb{R}_{\geq 0}\}$$

The **circumcenter** of a triangle $t = t_{u,v,w}$ is the unique point that is equidistant to each vertex u, v, w, and will be denoted $\mathtt{cc}(t)$. This distance to each vertex u, v, w is the **circumradius** of t, and will be denoted $\mathtt{rad}(t)$ so that

$$\mathtt{rad}(t) := \|\mathtt{cc}(t) - u\| = \|\mathtt{cc}(t) - v\| = \|\mathtt{cc}(t) - w\|$$

The **circumcircle** of t is the smallest ball containing the points u, v, w, and will be denoted $\mathtt{cball}(t)$. Formally,

$$\mathtt{cball}(t) := \mathtt{ball}(\mathtt{cc}(t), \mathtt{rad}(t)).$$

2.1 Delaunay Triangulations and Voronoi Diagrams

Definition 1 *For a set $S \subset \mathbb{R}^2$ the* ***Delaunay triangulation*** *of S is the set of triangles $t = t_{u,v,w}$ such that no point $p \in S \backslash \{u, v, w\}$ is contained in the circumcircle of t and is denoted*

$$\mathtt{Del}_S := \{t_{u,v,w} \mid \mathtt{cball}(t_{u,v,w}) \cap S = \{u, v, w\}\}$$

The set of Delaunay vertices of $\mathtt{Del}_S$ is the set S itself.

The **Voronoi cell** of a point $u \in S$ is the set of points $x \in \mathbb{R}^2$ that are closer to u than any other point in S.

$$\mathtt{Vor}_S(u) := \{x \in \mathbb{R}^2 \mid \mathbf{d}_S(x) = \|u - x\|\}.$$

Definition 2 *For a set $S \subset \mathbb{R}^2$ the* ***Voronoi diagram*** *of S is the set of all Voronoi cells of the points in S and is denoted*

$$\mathtt{Vor}_S := \{\mathtt{Vor}_S(u) \mid u \in S\}.$$

For a set $S \subset \mathbb{R}^2$ the Voronoi diagram $\mathtt{Vor}_S$ is dual to the Delaunay triangulation $\mathtt{Del}_S$. That is, Delaunay vertices $u \in S$ correspond to Voronoi cells (faces) $\mathtt{Vor}_S(u)$ and the Delaunay triangles $t \in \mathtt{Del}_S$ correspond to **Voronoi vertices**, defined to be the circumcenters $\mathtt{cc}(t)$. The set of Voronoi vertices corresponding to a point $u \in S$ will be denoted

$$\mathtt{Vor}^0_S(u) := \{\mathtt{cc}(t_{u,v,w}) \in \mathtt{Vor}_S(u) \mid t_{u,v,w} \in \mathtt{Del}_S\}.$$

The **Voronoi edge** corresponding to points $u, v \in S$ is the intersection of the Voronoi cells of u and v and will be denoted

$$\mathtt{Vor}_S(u, v) := \mathtt{Vor}_S(u) \cap \mathtt{Vor}_S(v).$$

For any $u, v \in M$ such that $\mathtt{Vor}_S(u, v) \neq \emptyset$ the Voronoi edge $\mathtt{Vor}_S(u, v)$ corresponds to an edge of the Delaunay triangulation, defined to be the convex closure of the points u and v.

The **outradius** of a Voronoi cell $\mathtt{Vor}_S(u)$ is the radius of the smallest ball centered at u containing $\mathtt{Vor}^0_S(u)$, and will be denoted

$$\mathtt{R}(u) := \max_{c \in \mathtt{Vor}^0_S(u)} \|c - u\|$$

The outradius of u is the distance to the farthest Voronoi vertex of $\mathtt{Vor}_S(u)$, denoted

$$\mathtt{farCorner}(\mathtt{Vor}_S(u)) := \operatorname*{argmax}_{c \in \mathtt{Vor}^0_S(u)} \|c - u\|.$$

Definition 3 *The* ***aspect ratio*** *of a Voronoi cell $\mathtt{Vor}_S(u)$ is the ratio of the distance to its farthest corner to the distance to its nearest edge and is denoted*

$$\mathtt{aspect}(\mathtt{Vor}_S(u)) := \frac{2\mathtt{R}(u)}{\mathbf{d}_{S,2}(u)}.$$

A set $S \subset \mathbb{R}^2$ is said to be **τ-well spaced** if

$$\mathtt{aspect}(\mathtt{Vor}_S(u)) \leq \tau$$

for all $u \in S$.

2.2 Periodic Point Sets

To avoid additional boundary conditions we will work in a covering space of the **flat torus**, which can be defined as $\mathbb{T}^2 = \mathbb{R}^2/\mathbb{Z}^2$ (see [1]). That is, we will restrict ourselves to a **bounding box** $B = [0, 1)^2$ and use copies of a set $S \subset B$ to simulate periodicity.

For any finite point set $S \subset B$ we will define the corresponding **periodic point set** as the set $S + \mathbb{Z}^2$ imbued with an equivalence relation so that $x \sim y$ if there exists some $a \in \mathbb{Z}^2$ such that $y = x + a$. Noting that $s \in P$ for all $s \in S$ we may therefore refer to the set of periodic copies of s by the equivalence class

$$[s] := \left\{x \in S + \mathbb{Z}^2 \mid \exists a \in \mathbb{Z}^2 : x = s + a\right\}.$$

3 Algorithm

For a bounding box $B = [0, 1)^2$ the following algorithm will use periodic points $x \in B + \mathbb{Z}^2$ to denote equivalence classes $[x]$ of which $x \in B$ are representative. The use of periodic points is a technical requirement of the algorithm in order to avoid additional points along the edge of the bounding box. As our analysis does not depend on the use of periodic points we will return to the original sets in the following sections.

Algorithm 1 takes as input a finite point set $P \subset B$, a set of initial mesh vertices $M_0 \subset B$ which we will take as a set of points arranged along a square in B, and constants $\tau \geq 2$, $k \geq 1$. The algorithm constructs a periodic set of mesh vertices $M \subset B + \mathbb{Z}^2$ satisfying

$$\mathtt{aspect}(\mathtt{Vor}_M(v)) \leq \tau$$

for all $v \in M$ and

$$|\mathtt{Vor}_M(v) \cap P| < k,\ \mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$$

for all $v \in M$, $t \in \mathtt{Del}_M$.

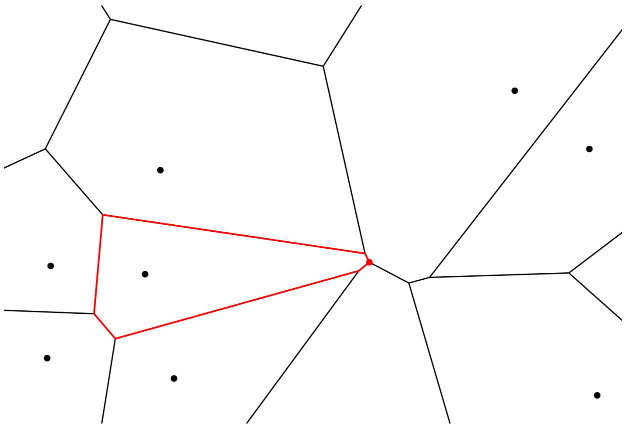
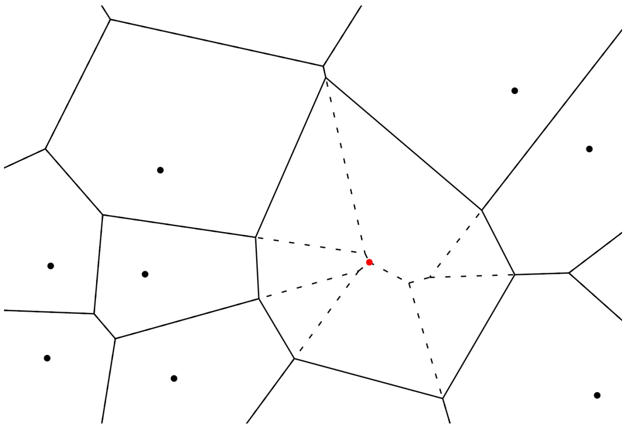



Figure 1: An application of the CLEAN procedure which chooses a point $v \in M$ with $\mathtt{aspect}(\mathtt{Vor}_M(v)) > \tau$ (red cell, top) and adds $\mathtt{farCorner}(\mathtt{Vor}_M(v))$ (red point, bottom).

Algorithm 1 KNNREFINE(P, M_0, τ, k)

```
 1: $M_0 \leftarrow M_0 + \mathbb{Z}^2$, $P \leftarrow P + \mathbb{Z}^2$, $M \leftarrow M_0$
 2: while there is a $v \in M$ or $t \in \mathtt{Del}_M$ such that
    $\mathbf{d}_{P,k}(\mathtt{cc}(t)) \leq \mathtt{rad}(t)$ or $|\mathtt{Vor}_M(v) \cap P| \geq k$ do
 3:     $M \leftarrow$ BREAK$(M, P)$
 4:     while $\exists v \in M$ with $\mathtt{aspect}(v) > \tau$ do
 5:         $M \leftarrow$ CLEAN$(M, v)$
 6: procedure BREAK$(M, P)$
 7:     if $\exists t \in \mathtt{Del}_M$ with $\mathbf{d}_{P,k}(\mathtt{cc}(t)) \leq \mathtt{rad}(t)$ then
 8:         $M \leftarrow M \cup \{\mathtt{cc}(t)\}$
 9:     else if $\exists v \in M$ with $|\mathtt{Vor}_M(v) \cap P| \geq k$ then
10:         $M \leftarrow M \cup \{\mathtt{farCorner}(\mathtt{Vor}_M(v))\}$
11: procedure CLEAN$(M, v)$
12:     $M \leftarrow M \cup \{\mathtt{farCorner}(\mathtt{Vor}_M(v))\}$
```

Figure 1 depicts an application of the CLEAN procedure to a point $v \in M$ such that $\mathtt{aspect}(\mathtt{Vor}_M(v)) > \tau$, resulting in the insertion of $\mathtt{farCorner}(\mathtt{Vor}_M(v))$.

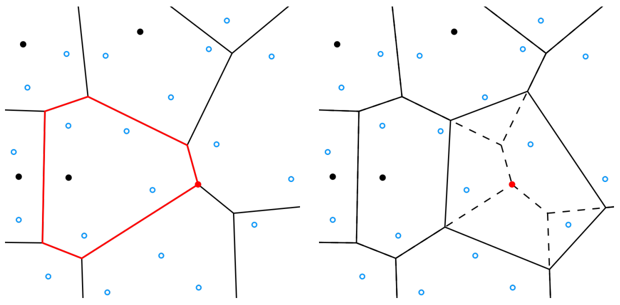


Figure 2: An example of the BREAK procedure applied to an instance with $v \in M$ such that $\mathtt{Vor}_M(v)$ (red cell, left) containts at least $k = 4$ points in P (blue rings) initiating the insertion of $\mathtt{farCorner}(\mathtt{Vor}_M(v))$ (red point, right).

Figure 2 illustrates an application of the BREAK procedure to a Voronoi cell containing at least $k = 4$ points in P, initiating the insertion its farthest corner. Similarly, Figure 3 depicts an application of the BREAK procedure to a configuration in which a Delaunay circumcircle contains at least $k = 4$ points in P, initiating the insertion its circumcenter.

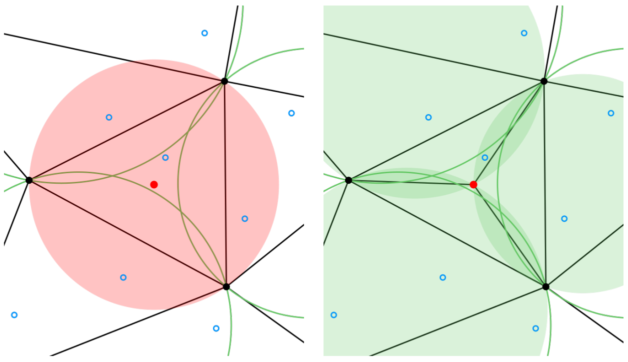


Figure 3: An example of the BREAK procedure applied to an instance with $t \in \mathtt{Del}_M$ such that $\mathtt{cball}(t)$ (red disk, left) contains at least $k = 4$ points in P (blue rings) initiating the insertion of $\mathtt{cc}(t)$ (red point, right).

Restricting ourselves to the bounding box $B = [0, 1)^2$ the remainder of this section will provide upper and lower bounds on the second nearest neighbor function $\mathbf{d}_{M,2}$ in terms of the kth nearest neighbor function $\mathbf{d}_{P,k}$ in order to prove Theorem 6 stated below.

Theorem 6 (Main Theorem) *Let $P \subset B$ be a finite point set and $\tau \geq 2$, $k \geq 1$ be constants. Let $M_0 \subset B$ be a set of initial mesh vertices.*

If KNNREFINE(P, M_0, τ, k) *terminates the resulting* *set of mesh vertices M is τ-well spaced,*

$$|\mathtt{Vor}_M(v) \cap P| < k, \ \mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$$

for all $v \in M$, $t \in \mathtt{Del}_M$, and

$$\alpha \mathbf{d}_{P,k} \le \mathbf{d}_{M,2} \le \beta \mathbf{d}_{P,k}$$

where $\alpha = \frac{\tau-2}{5\tau-2}$, $\beta = \frac{3-\vartheta_\tau}{1-\vartheta_\tau}$ and $\vartheta_\tau = \sqrt{1-\frac{1}{\tau^2}}$.

3.1 Lower Bound

We will first show that the second nearest neighbor function $\mathbf{d}_{M,2}$ to the output set $M \subset B$ is not too small compared to the kth-nearest neighbor function $\mathbf{d}_{P,k}$ to the input set $P \subset B$.

Let $M_0 \subseteq M$ be a set of input vertices and v_i be the ith circumcenter added to M. We define an order relation $\prec$ on M such that for all $i > 0$

$$u_0 \prec v_{i-1} \prec v_i$$

where $u_0 \in M_0$.

Definition 4 *For a set $M \subset B$ imbued with the order relation $\prec$ the* **insertion radius** *of a $v \in M$ is the distance from v to its nearest predecessor and is denoted*

$$\lambda_v := \min_{u \prec v} \|u - v\|.$$

The insertion radius λ_{v_i} of the ith vertex is the distance to the closest point in $M_i \subset M$ where $M_i = \{v_j \mid 0 < j < i\}$ is the ordered set of mesh vertices in M before the point v_i is added. Note that the insertion radius of a point v_i will be at most $\|v_i - v_j\|$ for all $v_j \in M_i$.

Lemma 1 shows that it suffices to bound the insertion radius for each $v \in M$ in order to bound $\mathbf{d}_{M,2}$ by $\mathbf{d}_{P,k}$ over M.

Lemma 1 *Let $K > 0$ be a constant.*
If $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ then

$$\mathbf{d}_{P,k}(v) \le K\mathbf{d}_{M,2}(v)$$

for all $v \in M$.

Proof. Let $v \in M$ and $u \in M \setminus \{v\}$ be such that $\|u - v\| = \mathbf{d}_{M,2}(v)$. If $u \prec v$ then by assumption

$$\begin{aligned} \mathbf{d}_{P,k}(v) &\le (K-1)\lambda_v < K\lambda_v \\ &\le K\|u-v\| = K\mathbf{d}_{M,2}(v). \end{aligned}$$

So, we may assume $v \prec u$, which implies $\lambda_u \le \|u - v\| = \mathbf{d}_{M,2}(v)$. Thus,

$$\begin{aligned} \mathbf{d}_{P,k}(v) &\le \mathbf{d}_{P,k}(u) + \|u-v\| && [\mathbf{d}_{P,k} \text{ is Lipschitz}] \\ &\le (K-1)\lambda_u + \|u-v\| && [\mathbf{d}_{P,k}(u) \le (K-1)\lambda_u] \\ &\le K\mathbf{d}_{M,2}(v). && [\lambda_u \le d_{M,2}(v)] \end{aligned}$$

□

We now apply Lemma 1 to the kth-nearest neighbor function to provide a lower bound for $\mathbf{d}_{M,2}$ by induction on the set of mesh vertices M produced by Algorithm 1. We will set $\lambda_{u_0} \ge \mathbf{d}_{P,k}(u_0)$ for all $u_0 \in M_0$.

Lemma 2 *Let $M_0 \subset B$ be a set of initial mesh vertices such that $\mathbf{d}_{P,k}(u_0) \le \lambda_{u_0}$ for all $u_0 \in M_0$. For constants $\tau \ge 2$, $k \ge 1$ let $M \subset B$ be a set of mesh vertices imbued with the order relation $\prec$ resulting from* kNNRefine(P, M_0, τ, k).
If M is τ-well spaced then for all $v \in M$

$$\mathbf{d}_{P,k}(v) \le K\mathbf{d}_{M,2}(v)$$

where $K = \frac{2\tau}{\tau-2}$.

Proof. Lemma 1 implies that it suffices to show that $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ for all $v \in M$. We will show this by induction on the number of circumcenters added.

Let M_i denote the set of mesh vertices in M before the ith circumcenter is added. In the base case we require $\mathbf{d}_{P,k}(u_0) \le \lambda_{u_0}$ for all $u_0 \in M_0$. It follows $\mathbf{d}_{P,k}(u_0) \le (K-1)\lambda_{u_0}$ as $K \ge 2$.

Assume inductively that $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ for all $v \in M_i$, and note that Lemma 1 implies

$$\mathbf{d}_{P,k}(v) \le K\mathbf{d}_{M_i,2}(v)$$

for all $v \in M_i$.

Let v_i be the ith circumcenter added and let $u \in M_i$ be the vertex whose Voronoi cell had poor quality, initiating the insertion of v_i. Letting $w \in M_i$ be the second nearest neighbor of u so that $w \ne u$ and $\|u - w\| = \mathbf{d}_{M_i,2}$ we have

$$\begin{aligned} \mathbf{d}_{P,k}(v_i) &\le \mathbf{d}_{P,k}(u) + \|u - v_i\| && [\mathbf{d}_{P,k} \text{ is 1-Lipschitz}] \\ &\le K\mathbf{d}_{M_i,2}(u) + \|u - v_i\| && [\text{Lemma 1 and hypothesis}] \\ &\le K\|u-w\| + \|u - v_i\| && [\text{Definition of } w] \\ &\le \|u - v_i\| \left(\frac{2K}{\tau} + 1\right) && [\mathtt{aspect}(\mathtt{Vor}_{M_i}(u)) > \tau] \\ &\le \|u - v_i\|(K-1) && \left[K = \frac{2\tau}{\tau - 2}\right] \\ &\le \lambda_{v_i}(K-1). && [\text{Definition of } u] \end{aligned}$$

As $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ for all $v \in M_i$ by our inductive hypothesis, $M_{i+1} = M_i \cup \{v_i\}$, and $\mathbf{d}_{P,k}(v_i) \le \lambda_{v_i}(K-1)$ it follows that $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ for all $v \in M_{i+1}$. It follows by induction that $\mathbf{d}_{P,k}(v) \le (K-1)\lambda_v$ for all $v \in M = \bigcup M_i$ by induction, and we may therefore conclude

$$\mathbf{d}_{P,k}(p) \le K\mathbf{d}_{M,2}(p)$$

for all $p \in M$ by Lemma 1. □

Figure 4 illustrates the proof of Lemma 2, depicting the Voronoi cell $\mathtt{Vor}_{M_i}(u)$ with bad aspect ratio in the top figure, and the resulting Voronoi diagram $\mathtt{Vor}_{M_{i+1}}$ after the insertion of v_i on the bottom. Note, the insertion radius $\lambda_{v_i} = \|v_i - u\|$ of v_i satisfies $\min_{v \prec v_i} \|v - v_i\|$

as the closest point to v_i is the point u initiating its insertion. This fact allows for the inductive proof of Lemma 2 as for all i such that $\texttt{aspect}(\texttt{Vor}_{M_i}(u)) > \tau$ for some $u \in M_i$ the ith mesh vertex added to M is $v_i = \texttt{farCorner}(\texttt{Vor}_{M_i}(u))$ and $\lambda_{v_i} = \|v_i - u\|$.

Theorem 3 extends the bound on $\mathbf{d}_{M,2}$ over M provided by Lemma 2 to all points in B for τ-well spaced sets M.

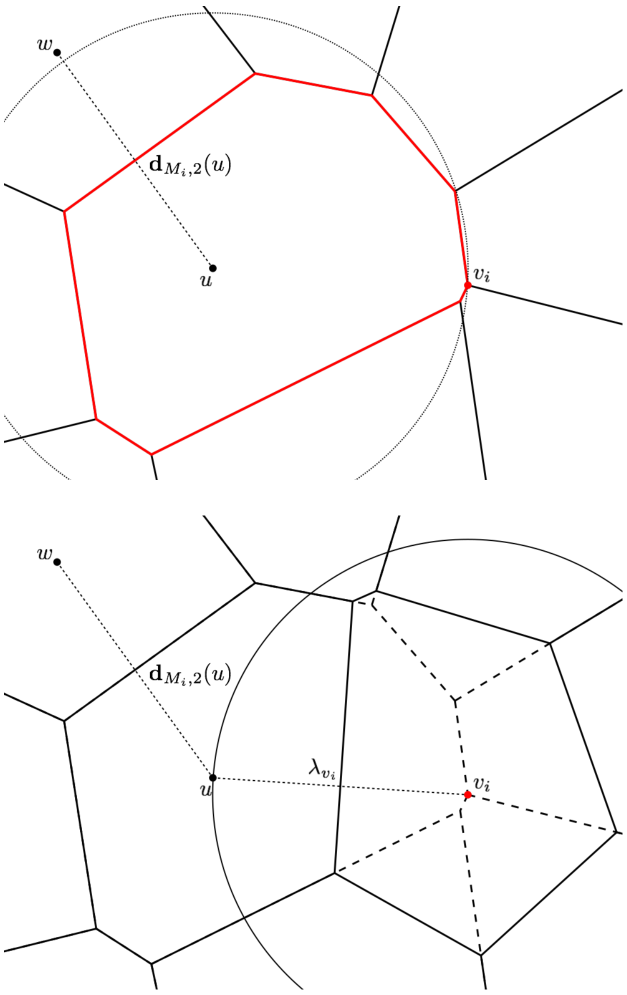


Figure 4: An illustration of Lemma 2 in which $\texttt{aspect}(\texttt{Vor}_{M_i}(u)) > \tau$ in $\texttt{Vor}_{M_i}$ (red cell, top) and the insertion radius λ_{v_i} of $v_i = \texttt{farCorner}(\texttt{Vor}_{M_i}(u))$ is the distance from u to v_i (red point, bottom).

Theorem 3 *Let $P \subset B$ be a finite point set and let $M_0 \subset B$ be a set of initial mesh vertices such that $\mathbf{d}_{P,k}(u_0) \le \lambda_{u_0}$ for all $u_0 \in M_0$. For constants $\tau \ge 2$, $k \ge 1$ let $M \subset B$ be a set of mesh vertices imbued with the order relation $\prec$ resulting from* kNNRefine(P, M_0, τ, k).

If M is τ-well spaced then for all $x \in B$

$$\alpha \mathbf{d}_{P,k}(x) \le \mathbf{d}_{M,2}(x).$$

where $\alpha = \frac{\tau-2}{5\tau-2}$.

Proof. First note that because M is a ordered and $\mathbf{d}_{P,k}(u_0) \le \lambda_{u_0}$ for all $u_0 \in M_0$ Lemma 2 implies

$$\mathbf{d}_{P,k}(v) \le K\mathbf{d}_{M,2}(v)$$

for all $v \in M$.

Let $x \in B$ and $v \in M$ be such that $x \in \texttt{Vor}_M(v)$. Because $\mathbf{d}_{M,2}$ and $\mathbf{d}_{P,k}$ are 1-Lipschitz, it follows

$$\begin{aligned}\mathbf{d}_{P,k}(x) - \|x - v\| &\le \mathbf{d}_{P,k}(v)\\ &\le K(\mathbf{d}_{M,2}(x) + \|x - v\|),\end{aligned}$$

therefore, because $\|x - v\| \le \mathbf{d}_{M,2}(x)$ we have

$$\begin{aligned}\mathbf{d}_{P,k}(x) &\le K\mathbf{d}_{M,2}(x) + \|x - v\|(K + 1)\\ &\le \mathbf{d}_{M,2}(x)(2K + 1)\end{aligned}$$

where $K = \frac{2\tau}{\tau-2}$, which implies $\alpha \mathbf{d}_{P,k}(x) \le \mathbf{d}_{M,2}(x)$ for all $x \in B$. $\square$

3.2 Upper Bound

We now must show that the second nearest neighbor function to the mesh vertices $M \subset B$ is not too large compared with the kth-nearest neighbor function to the input set P. We will first show that the distance from any point $x \in \texttt{Vor}_M(p)$ to two points in M is within a constant factor of the distance from p to the circumcenter $\texttt{cc}(t)$ of some Delaunay triangle $t \in \texttt{Del}_M$.

Lemma 4 *Let M be a τ-well spaced such that $|\texttt{Vor}_M(v) \cap P| < k$ for all $v \in M$ and let $\beta > 1$ be a constant. Let $p \in M$ be such that $x \in \texttt{Vor}_M(p)$ for any $x \in B$.*

If $\mathbf{d}_{P,k}(x) < \frac{1}{\beta}\mathbf{d}_{M,2}(x)$ then there exists a Delaunay triangle $t \in \texttt{Del}_M$ with $\texttt{cc}(t) \in \texttt{Vor}^0_M(p)$ such that

$$\mathbf{d}_{M,2}(x) \le \frac{\beta}{\beta - 1}\|p - \texttt{cc}(t)\|.$$

Proof. Note that because $|\texttt{Vor}_M(v) \cap P| < k$ for all $v \in M$ we have that $\texttt{ball}_{x,P_k} \not\subset \texttt{Vor}_M(v)$, and there therefore must exist some $q \in M$ such that $\texttt{ball}_{x,P_k} \cap \texttt{Vor}_M(q) \neq \emptyset$ and $\texttt{Vor}_M(p, q) \neq \emptyset$. It follows that there exists some $x' \in \texttt{Vor}_M(p, q) \cap \texttt{ball}_{x,P_k}$ with $\mathbf{d}_{M,2}(x') = \|p - x'\| = \|q - x'\|$ such that

$$\begin{aligned}\mathbf{d}_{M,2}(x) &\le \mathbf{d}_{M,2}(x') + \|x - x'\| && [\mathbf{d}_{M,2} \text{ is 1-Lipschitz}]\\ &= \|p - x'\| + \|x - x'\| && [\|p - x'\| = \mathbf{d}_{M,2}(x')]\\ &\le \|p - x'\| + \mathbf{d}_{P,k}(x) && [\|x - x'\| \le \mathbf{d}_{P,k}(x)]\\ &\le \|p - x'\| + \frac{1}{\beta}\mathbf{d}_{M,2}(x) && \left[\mathbf{d}_{P,k}(x) < \tfrac{1}{\beta}\mathbf{d}_{M,2}(x)\right]\end{aligned}$$

Thus, $\mathbf{d}_{M,2}(x) \le \frac{\beta}{\beta-1}\|p - x'\|$.

Because the point x' lies on the Voronoi boundary $\texttt{Vor}_M(p, q)$ there must exist some $t \in \texttt{Del}_M$ with $\texttt{cc}(t) \in$

$\texttt{Vor}^0_M(p) \cap \texttt{Vor}^0_M(q)$ such that $\|p - x'\| \le \|p - \texttt{cc}(t)\|$. It follows that

$$\begin{aligned}\mathbf{d}_{M,2}(x) &\le \frac{\beta}{\beta-1}\|p - x'\| \\ &\le \frac{\beta}{\beta-1}\|p - \texttt{cc}(t)\|\end{aligned}$$

□

Theorem 5 states that when Algorithm 1 terminates $\mathbf{d}_{M,2} \le \beta\mathbf{d}_{P,k}$. We will draw a contradiction, depicted in Figure 5, in which there must be some Voronoi cell or Delaunay circumcircle containing at least k points in P whenever there exists some $x \in B$ such that $\mathbf{d}_{P,k}(x)$ is within a constant factor *less* than $\mathbf{d}_{M,2}(x)$, as in Lemma 4.

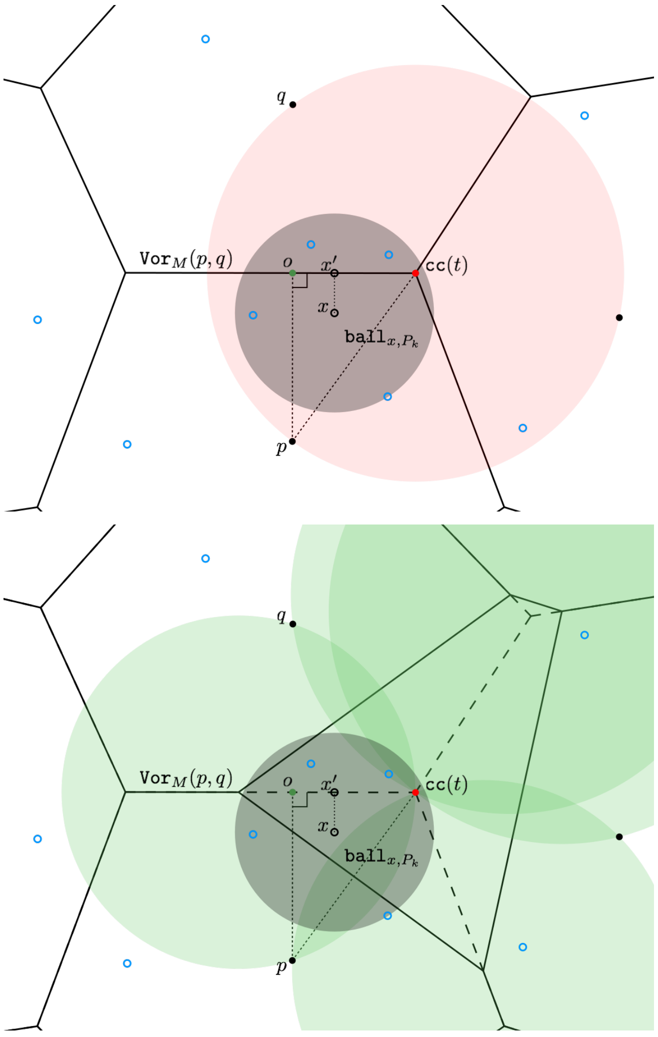


Figure 5: An illustration of the contradiction drawn in Theorem 5 (top) in which $\texttt{ball}_{x,P_k}$ (gray disk) is contained in the circumcircle of t (red disk). In this case, Algorithm 1 would not have terminated, as another Break move could be performed (bottom).

Theorem 5 *Let $P \subset B$ be a finite point set and $\tau \ge 2$, $k \ge 1$ be constants. Let $M \subset B$ be a τ-well spaced set such that $|\texttt{Vor}_M(v) \cap P| < k$ for all $v \in M$.*

If $\mathbf{d}_{P,k}(\texttt{cc}(t)) > \texttt{rad}(t)$ for all $t \in \texttt{Del}_M$ then

$$\mathbf{d}_{M,2} \le \beta\mathbf{d}_{P,k}$$

for a constant

$$\beta = \frac{3-\vartheta_\tau}{1-\vartheta_\tau} \text{ where } \vartheta_\tau = \sqrt{1 - \frac{1}{\tau^2}}.$$

Proof. Suppose, for the sake of contradiction, there exists a point $x \in B$ such that $\mathbf{d}_{P,k}(x) < \frac{1}{\beta}\mathbf{d}_{M,2}(x)$. Letting $p, q \in M$, $t \in \texttt{Del}_M$, and $x' \in \texttt{ball}_{x,P_k} \cap \texttt{Vor}_M(p,q)$ be such that $x \in \texttt{Vor}_M(p)$ and $\texttt{cc}(t) \in \texttt{Vor}^0_M(p) \cap \texttt{Vor}^0_M(q)$ as in Lemma 4 we have

$$\mathbf{d}_{M,2}(x) \le \frac{\beta}{\beta-1}\|p - \texttt{cc}(t)\|.$$

To simplify notation we will set the point $o = \frac{p-q}{2}$ at the origin. Because the Delaunay edge of t containing the points p, q and the Voronoi edge $\texttt{Vor}_M(p,q)$ are orthogonal we can bound the distance from x' to the circumcenter of t by the Pythagorean Theorem as follows.

$$\begin{aligned}\|x' - \texttt{cc}(t)\|^2 &\le \|\texttt{cc}(t) - o\|^2 \\ &\le \|p - \texttt{cc}(t)\|^2 - \|p - o\|^2 \\ &\le \left(1 - \frac{1}{\tau^2}\right)\|p - \texttt{cc}(t)\|^2. \quad \left[\frac{\|p - \texttt{cc}(t)\|}{\|p\|} \le \tau\right]\end{aligned}$$

Noting that $\texttt{rad}(t) = \|p - \texttt{cc}(t)\|$ we can now prove that there are at least k points in P is the circumcircle of t.

$$\begin{aligned}\mathbf{d}_{P,k}(\texttt{cc}(t)) &\le \mathbf{d}_{P,k}(x') + \|x' - \texttt{cc}(t)\| && [\mathbf{d}_{P,k} \text{ is 1-Lipschitz}] \\ &\le \mathbf{d}_{P,k}(x') + \|p - \texttt{cc}(t)\|\vartheta_\tau && [\|x' - \texttt{cc}(t)\| \le \vartheta_\tau] \\ &\le 2\mathbf{d}_{P,k}(x) + \|p - \texttt{cc}(t)\|\vartheta_\tau && [\mathbf{d}_{P,k}(x') \le 2\mathbf{d}_{P,k}(x)] \\ &< \frac{2}{\beta}\mathbf{d}_{M,2}(x) + \|p - \texttt{cc}(t)\|\vartheta_\tau && \left[\mathbf{d}_{P,k}(x) < \frac{1}{\beta}\mathbf{d}_{M,2}(x)\right] \\ &\le \left(\frac{2}{\beta-1} + \vartheta_\tau\right)\|p - \texttt{cc}(t)\| && [\text{Lemma 4}] \\ &\le \|p - \texttt{cc}(t)\|. && \left[\beta \ge \frac{3-\vartheta_\tau}{1-\vartheta_\tau}\right]\end{aligned}$$

It follows that $\mathbf{d}_{P,k}(\texttt{cc}(t)) \le \|p - \texttt{cc}(t)\| = \texttt{rad}(t)$, a contradiction, as we assumed $\mathbf{d}_{P,k}(\texttt{cc}(t)) > \texttt{rad}(t)$ for all $t \in \texttt{Del}_M$. We may therefore conclude that if $M \subset B$ is τ-well spaced, $|\texttt{Vor}_M(v) \cap P| < k$ for all $v \in M$, and $\mathbf{d}_{P,k}(\texttt{cc}(t)) > \texttt{rad}(t)$ for all $t \in \texttt{Del}_M$, then $\mathbf{d}_{M,2} \le \beta\, \mathbf{d}_{P,k}$. □

3.3 Main Theorem

Our final Theorem 6 states that when Algorithm 1 terminates the output set M is τ-well spaced with strictly less than k points from P in each Voronoi cell and Delaunay circumcircle. It will then follow from Theorems 3 and 5 that the second nearest neighbor distance function $\mathbf{d}_{M,2}$ defined on the bounding box $B = [0,1)^2$ is bounded above and below by the kth-nearest neighbor function of the input set $P \subset B$.

Theorem 6 (Main Theorem) *Let $P \subset B$ be a finite point set and $\tau \geq 2$, $k \geq 1$ be constants. Let M_0 be a set of initial mesh vertices with $\mathbf{d}_{P,k}(u_0) \leq \lambda_{u_0}$ for all $u_0 \in M_0$.*

When kNNRefine(P, M_0, τ, k) *terminates the resulting ordered set of mesh vertices $M \subset B$ is τ-well spaced,*

$$|\mathtt{Vor}_M(v) \cap P| < k, \ \mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$$

for all $v \in M$, $t \in \mathtt{Del}_M$, and

$$\alpha \mathbf{d}_{P,k} \leq \mathbf{d}_{M,2} \leq \beta \mathbf{d}_{P,k}$$

where $\alpha = \frac{\tau-2}{5\tau-2}$, $\beta = \frac{3-\vartheta_\tau}{1-\vartheta_\tau}$ and $\vartheta_\tau = \sqrt{1-\frac{1}{\tau^2}}$.

Proof. First note that each $v \in M$, $t \in \mathtt{Del}_M$ must satisfy $\mathtt{aspect}(\mathtt{Vor}_M(v)) \leq \tau$ in order for each internal Clean loop to complete. That is, if $\mathtt{aspect}(\mathtt{Vor}_M(v)) > \tau$ kNNRefine(P, M_0, τ, k) will not have terminated, as another Clean procedure can be performed.

In order for each outer Break loop to complete we must have that $|\mathtt{Vor}_M(v) \cap P| < k$ and $\mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$ for each $v \in M$, $t \in \mathtt{Del}_M$. Otherwise, kNNRefine(P, M_0, τ, k) will not have terminated, as an additional Break procedure can be performed.

So, we may assume $\mathtt{aspect}(\mathtt{Vor}_M(v)) \leq \tau$, $|\mathtt{Vor}_M(v) \cap P| < k$, and $\mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$ for all $v \in M$, $t \in \mathtt{Del}_M$. As M is an indexed set, $\mathbf{d}_{P,k}(u_0) \leq \lambda_{u_0}$ for all $u_0 \in M_0$, and $\mathtt{aspect}(\mathtt{Vor}_M(v)) \leq \tau$ for all $v \in M$ it follows from Theorem 3 that

$$\alpha \mathbf{d}_{P,k} \leq \mathbf{d}_{M,2}.$$

Moreover, because $|\mathtt{Vor}_M(v) \cap P| < k$, and $\mathbf{d}_{P,k}(\mathtt{cc}(t)) > \mathtt{rad}(t)$ for all $v \in M, t \in \mathtt{Del}_M$ it follows from Theorem 5 that

$$\mathbf{d}_{M,2} \leq \beta \mathbf{d}_{P,k}.$$

We may therefore conclude that in order for Algorithm 1 to terminate the set M of mesh vertices constructed by kNNRefine(P, M_0, τ, k) must satisfy

$$\alpha \mathbf{d}_{P,k} \leq \mathbf{d}_{M,2} \leq \beta \mathbf{d}_{P,k}$$

for constants $\alpha = \frac{\tau-2}{5\tau-2}$ and $\beta = \frac{3-\vartheta_\tau}{1-\vartheta_\tau}$. □

4 Point Location

At the heart of any Delaunay refinement algorithm is an incremental Delaunay triangulation algorithm, constructing the Delaunay triangulation one vertex at a time. The standard approach in computational geometry for bounding the running time is to use randomization, yielding the randomized incremental algorithm. However, Delaunay refinement does not permit such arbitrary reordering of the points, because the points to be added are discovered in the course of running the algorithm. Thus, the original Chew and Ruppert algorithms could have $O(n^2)$ running times. This was improved by Miller [6] and later by Hudson et al. [4] who developed the so-called *Sparse Refinement* approach with their Sparse Voronoi Refinement (SVR) algorithm.

As the algorithm needs to know the number of input points contained in every Voronoi cell as well as the number of points contained in every Delaunay circumball, we will maintain two different point location data structures. The first is a 2-way mapping between points of P and the Voronoi cells (points of M). The second is a 2-way association between the points of P and the Delaunay circumballs. For each local update to the Delaunay triangulation induced by a single insertion, some Voronoi cells are affected as well as some circumballs. The sparse refinement approach always maintains some guarantee on the quality of the underlying triangulation.

We adopt the vocabulary used by Hudson et al. [4] when describing the algorithm in terms of Break and Clean moves. In fact, we will do a strict subset of the operations that would usually be performed by SVR. The difference is that in SVR, if any input point from P has not yet been added, then the algorithm will continue, whereas our algorithm will halt early if every Voronoi cell and every Delaunay circumball contains fewer than k points. Thus, it follows immediately that our algorithm will also achieve a running time (and output size) of $O(n \log \Delta)$. Here, Δ denotes the *spread* of the input defined as the ratio of the largest to smallest pairwise distances. We only need to observe that counting the points in a Voronoi cell or Delaunay cicumball is not more expensive than iterating through the list of these points which happens anyway each time a Voronoi cell changes or a Delaunay circumball is created or destroyed.

5 Conclusions and future work

We have shown how a simple modification of Delaunay refinement solves the kth nearest neighbor sampling problem. Several interesting open problems remain.

1. Does the algorithm work in $\mathbb{R}^d$ for $d > 2$? We believe the answer is yes.

2. Is it possible to eliminate the dependence on $\log(\Delta)$ as was done for Voronoi refinement of points [7, 8]?
3. Is the size of the sample we produce asymptotically optimal? Specifically, we would like to extend the optimality theory of Ruppert [9].

References

[1] M. Bogdanov, M. Teillaud, and G. Vegter. Covering spaces and delaunay triangulations of the 2d flat torus. In *28th European Workshop on Computational Geometry*, 2012.

[2] S.-W. Cheng, T. K. Dey, and J. R. Shewchuk. *Delaunay Mesh Generation.* CRC Press, 2012.

[3] L. P. Chew. Guaranteed-quality triangular meshes. Technical Report TR-89-983, Department of Computer Science, Cornell University, 1989.

[4] B. Hudson, G. Miller, and T. Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, Birmingham, Alabama, 2006. Long version available as Carnegie Mellon University Technical Report CMU-CS-06-132.

[5] D. O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *Ann. Math. Statist.*, 36(3):1049–1051, 1965.

[6] G. L. Miller. A time efficient Delaunay refinement algorithm. In J. I. Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 400–409. SIAM, 2004.

[7] G. L. Miller, T. Phillips, and D. R. Sheehy. Beating the spread: Time-optimal point meshing. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 321–330, 2011.

[8] G. L. Miller, D. R. Sheehy, and A. Velingker. A fast algorithm for well-spaced points and approximate delaunay graphs. In *Proceedings of the 29th annual Symposium on Computational Geometry*, pages 289–298, 2013.

[9] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

NearptD: A Parallel Implementation of Exact Nearest Neighbor Search using a Uniform Grid

David Hedin* W. Randolph Franklin†

Abstract

We present **NearptD**, a very fast parallel nearest neighbor algorithm and implementation, which has processed 10^7 points in E^6 and $184 \cdot 10^6$ points in E^3. It uses 1/5 the space and as little as 1/100 the preprocessing time FLANN (a well-known approximate nearest neighbor program). Up to E^4, its query time is also faster, by up to a factor of 100. NearptD uses Nvidia Thrust and CUDA in C++ to perform parallel preprocessing and querying of large point cloud data. Nearest neighbor searching is needed by many applications, such as collision detection, computer vision or machine learning. This implementation is an extension of the Nearpt3 algorithm performed in parallel on the GPU for a variable number of dimensions. NearptD shows that a uniform grid can outperform a kd-tree for preprocessing and searching large datasets.

1 Introduction

Nearest neighbor searching is an operation performed in many applications, in fields such as computer graphics, computer vision, statistics and machine learning. Nearest neighbor searching typically consists of preprocessing the data into a search structure to reduce the time needed for future queries on that dataset. A popular data structure for preprocessing spatial data is the kd-tree[4]. Kd-trees cost $\Theta(NlogN)$ time to preprocess N fixed points and an average $\Theta(logN)$ time per query.

NearptD uses a uniform grid[1], which stores the fixed points in a flat search structure. This reduces the time and space complexity of kd-trees. Because the uniform grid used by NearptD is not a hierarchical data structure, the data can be preprocessed, with the appropriate choice of grid size, into a grid with a cost of $\Theta(N)$. Because querying against the uniform grid does not involve traversing a tree structure, NearptD can obtain expected query times of $\Theta(1)$. It is possible for adversarial input to increase query times to $\Theta(N)$, but these inputs are not often found in real world datasets. Some of our test datasets have very unevenly spaced data, but NearptD still processed them quickly. These types of input would still induce many levels in a hierarchical data structure, which would slow them as well.

Exact nearest neighbor searching can be an expensive operation. It often requires continuing to search after one near neighbor has been found, to ensure that it is, in fact, the nearest neighbor. One method to increase the performance of many data structures is to perform approximate nearest neighbor search instead of exact[6]. This reduces the time required to ensure the validity of the output, only ensuring that it is "good enough" for the application. However, NearptD shows that it can perform exact nearest neighbor searching and still outperform approximate searching.

2 Related Work

This work is mainly based on Nearpt3[2], a nearest neighbor search algorithm which uses a uniform grid in 3 dimensions. NearptD extends Nearpt3 to allow the flexibility of any number of dimensions to be used, though practicality limits this to 6 dimensions.

A widely used nearest neighbor search library is the Fast Library for Approximate Nearest Neighbors (FLANN)[3], a part of the OpenCV library. FLANN is an approximate nearest neighbor library that utilizes kd-trees and k-means trees[5] to preprocess data into a search tree. The Computational Geometry Algorithms Library (CGAL)[7] also offers both approximate and exact nearest neighbor searching using kd-trees.

3 Parallel Programming in Geometry

NearptD executes in parallel on Nvidia GPUs. Perhaps 1/3 of all PCs have them, intended to accelerate graphics. However, they can also be used for general parallel programming. The low-level access is via CUDA, a small set of extensions to C++ together with a library. Higher level APIs like Thrust add more powerful tools like a functional language paradigm, at the cost of less low-level control. GPUs provide so much computing power that geometric algorithms that are not parallelizable are quite possibly obsolete. The challenge is that parallelizable algorithms require simple regular data structures and algorithms.

For parallel programming, multicore CPUs present an attractive alternative to GPUs. All modern pow-

*Rensselaer Polytechnic Institute, `david.hedin13@gmail.com`
†Rensselaer Polytechnic Institute, `mail@wrfranklin.org`

erful CPUs are multicore, even those in smart phones. The two types of parallel hardware have different capabilities. Thrust can also be compiled to use multicore CPUs, so that many algorithms can use either.

4 Algorithm

4.1 Antepreprocess

As described later, the query step will spiral out from the cell containing the query point. A table of cells, called the *spiral order table*, containing the order in which to spiral out, cell is computed before preprocessing the data. The table is computed as follows.

1. Generate coordinates $(x_1, x_2, \ldots, x_d)$ for all grid cells such that $0 \le x_1 \le x_2 \le \ldots \le x_d \le R$ for some R, calculated to ensure the total number of cells will be less than 2^{20}.
2. Sort coordinates by $\sqrt{{x_1}^2 + {x_2}^2 + \ldots + {x_d}^2}$
3. For each cell, c, find its *stop cell*, whose closest point to the origin is at least as close as the farthest point in c.

This table does not depend on the data. It also contains the *stop cell*, which says many more cells to query after the first cell containing a fixed point is found. This is to ensure the nearest neighbor is, in fact, found, because a later cell in the spiral order might contain a closer point than the first point found. The spiral order table's size in the GPU memory depends only on the dimensionality of the data, shown in Table 1. The spiral order table can be computed by the programmer while developing NearptD, and distributed in a file to be read at run time.

Dimensions	Memory
2	8MB
3	10MB
4	12MB
d	$(2d+4)$MB

Table 1: GPU memory usage of cells array.

4.2 Preprocess

The uniform grid will contain G cells in each dimension, for a total of G^d cells, where d is the dimensionality of the data. G is called the grid's *resolution.* Before preprocessing the data, G must be determined. We use $G = G_r \sqrt[d]{N_f}$, where N_f is the number of fixed points, and G_r is a scaling factor, usually $0.5 \le G_r \le 3$, though the ideal factor is another possible area of research. As G_r varies, parts of NearptD run faster, and others more slowly, so that the total time varies relatively little even as G_r varies a factor of two away from its optimum. However, larger values of G_r do increase the memory footprint.

Next, three arrays are allocated, as follows.

1. *cells*, an array of size N_f, to contain pointers to the points in each cell,
2. *base*, an array of size $G^d + 1$ for the indices of the start of each cell within *cells*, so that the *j-th* point of the *i-th* cell is point # `cells[base[i]+j]`, and
3. *index*, a temporary array of size N_f allocated to preprocess the points, and discarded afterwards.

Preprocessing the data into a uniform grid is done on the GPU in parallel using these three arrays, as follows.

1. Store a sequence from 0 to N_f in *index*, to maintain the initial order of the array.
2. Calculate the ID of the cell that each fixed point belongs to, and store it in *cells*.
3. Sort *cells* and *index* based on the calculated IDs, to group points together by cell.
4. Calculate the index of the start of each cell, and store it in *base*.
5. Scan across each cell to calculate the number of points in each cell, stored in *cells*.
6. Resort *cells* by the original *index* array, to restore the original order of the points.
7. Transform the *index* array to contain the offset of each point within its cell, calculated from the *cells* and *base* arrays.
8. Fill *cells* with a sequence from 0 to N_f to keep track of the order of the array.
9. Sort *index* and *cells* by the offset of each point.
10. *cells* now contains the index of each point, sorted by their position in the grid.

If only the points' coordinates are relevant, and not, say, their location in some other data structure, then *cells* could contain the points' coordinates themselves instead of pointers. In machine-level programming, this is called *immediate mode.* The benefits are decreased memory use and increased locality of memory reference. On either CPUs or GPUs, that can reduce memory access times by reducing cache misses.

4.3 Query

There are three possible types of query that can be performed for a given query point, q, denoted as the *fast case*, the *slow case*, and the *exhaustive case.* A *fast case* query is performed if c, the cell containing q, contains at least one fixed point. If c is empty, a *slow case* query is performed, spiraling out from c, checking if any nearby cells contain fixed points for querying against. If the *slow case* query fails, *exhaustive* querying is performed to query against every fixed point.

This implementation supports querying many points at once, which is performed in parallel on the GPU, returning a list of the index of the closest fixed point for each query point, as well as single point queries. In more detail:

1. Calculate the number of fixed points in the cell containing each query point.
2. For all queries that contain at least one fixed point in their cell, perform a *fast case* query.
3. For all queries that don't contain a fixed point in their cell, perform a *slow case* query.
4. If the slow case query failed to find a fixed point, perform an *exhaustive* query.

4.3.1 Fast Case Query

This query is only performed if there are fixed points in c. Each point in the query cell is tested and the closest fixed point, f, is found. It is possible, however, that a neighboring cell could contain a point closer to q than f, if, for instance, q was near a wall of c. So, calculate the nearby cells that could contain a point closer than f, and search them for the closest fixed point.

4.3.2 Slow Case Query

If c does not contain any points, the next step is to begin searching around c for cells that may contain points. This is done by using the spiral order table computed in the antepreprocessing step to spiral out from c. For each cell in the table, we derive other reflected and rotated cells. For d dimensional data, there are up to $2^d d!$ possible reflections and permutations, although if some indices are zeros or repeated values, this number can be smaller. These could be pre-computed in the antepreprocessing stage, but this would require much more fixed memory. If a fixed point is found in one of these cells, we continue spiraling out until the *stop cell* is reached, ensuring that the closest point is found.

4.3.3 Exhaustive Query

Exhaustive queries are the worst case query performed if neither c nor any cells near c contain a fixed point. This is very rare, not happening once in any of the real 3D datasets that we tested. However, an adversarial query could generate such a case. We exhaustively query by linearly searching all the fixed points, in parallel on the GPU.

5 Performance

All tests were run on an Intel i7-5820k with 32 GB of DDR4 memory and a Nvidia GTX 980Ti with 6GB of GDDR5 memory.

NearptD was implemented using Thrust 1.8.2 in C++ with CUDA 7.5, compiled using nvcc and clang++ 3.5 with level 3 optimization. (Thrust is an efficient API on top of C++ and CUDA that adds a functional programming paradigm.) For uniform datasets, a $G_r = 0.5$ was used, and for all other datasets $G_r = 1.0$.

Nearpt3 was compiled using clang++ 3.5 with level 3 optimization, using the same scheme to choose G_r as NearptD.

FLANN was compiled using clang++ 3.5 with level 3 optimization, preprocessing the data into a kd-tree with default parameters and performing a KNN search with default search parameters and $k = 1$.

The following datasets were used as real world comparisons of NearptD, Nearpt3 and FLANN in 3 dimensions. We are grateful to these projects for kindly making this data available.

- *uniXXX*: A uniformly and independently distributed set of 10^4 to 10^8 random points.
- *bunny* ($N_f = 35,947$): Stanford University Computer Graphics Laboratory[9].
- *hand* ($N_f = 327,323$): Clemson's Stereolithography Archive, via Georgia Tech[10].
- *dragon* ($N_f = 437,645$): Brian Curless, via Stanford and Georgia Tech.
- *bone6* ($N_f = 569,636$): The Visible Human Project, and William E. Lorensen, via Georgia Tech.
- *blade* ($N_f = 882,954$): Visualization Toolkit (VTK), via Georgia Tech.
- *powerplant* ($N_f = 5,423,053$): The complete powerplant from the University of North Carolina's UNC Chapel Hill Walkthru Project[11].
- *david* ($N_f = 28,168,109$), and
- *stmatthew* ($N_f = 184,098,599$): The Stanford Digital Michelangelo Project Archive[8].

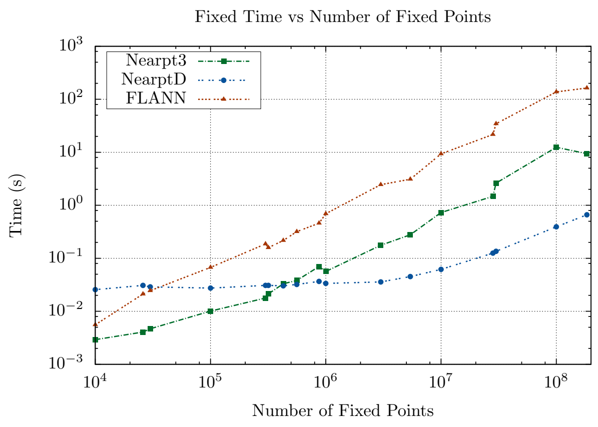


Figure 1: Time to preprocess fixed points into a search structure, not including I/O.

NEARPTD has some fixed costs independent of data size, mainly the antepreprocessing step to create the cell search order necessary for slow case queries, as well as overhead to run tasks on the GPU. For this reason, smaller datasets can take longer than existing programs, but on larger datasets, this cost is negligible. NEARPTD exhibits an order of magnitude speedup over Nearpt3 on larger datasets, and two orders of magnitude speedup vs FLANN. Figure 1 shows that NEARPTD becomes faster than both FLANN and Nearpt3 for preprocessing datasets of at least 10^6 points, with an order of magnitude speedup as the number of points increases.

Arguably the antepreprocessing costs should not be included any more than the compilation costs, since both are incurred only once, not once per dataset.

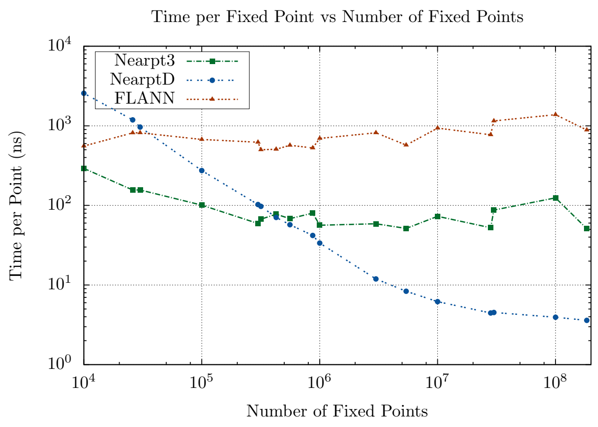


Figure 2: Per point time to preprocess fixed points into a search structure, not including I/O.

Figure 2 compares the time per point for each program to preprocess the fixed points into their respective search structures, and we see that NEARPTD benefits from its parallelism more on larger datasets. FLANN averages roughly 753ns per fixed point, and Nearpt3 averages around 95ns per fixed point. NEARPTD can take up to 2.5μs per fixed point for smaller datasets, but preprocesses *stmatthew* in just 3.5ns per fixed point.

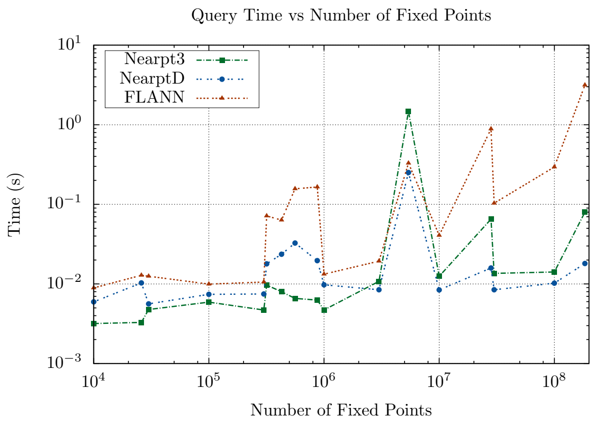


Figure 3: Time to complete 10^4 queries sampled from the fixed distribution.

To compare query times, each dataset had 10^4 points sampled from it to use as query points, with the rest preprocessed into a search structure. Figure 3 shows that as the number of fixed points increases, NEARPTD becomes faster than either existing program. For small datasets, it does not demonstrate any speedup, likely due to the GPU overheads necessary to compute these queries. Even for extremely adversarial data, such as the *powerplant* dataset (the large spike in Figure 3 at 5 million points), where 98% of fixed points are contained within one cell, NEARPTD still performs just well as FLANN. These results used an unoptimized grid resolution, however, and doubling the grid resolution reduces query time by 3 times, although it increases the memory usage.

The *powerplant* dataset is important because it disproves the notion that an adaptive dataset like the kd-tree will process uneven data better than the uniform grid. All three programs become significantly slower when querying such an adversarial dataset, and the end result is that NEARPTD and FLANN have about the same query time here, with NEARPTD being a little faster.

There are two reasons. On such uneven data, the kd-tree has many levels and more of its cells are empty. These cells are allocated on the memory heap, whose time cost is superlinear (the more objects on the heap, the more time that allocating and freeing each object costs). Each query has to walk down the deep tree. In contrast, with the uniform grid, empty cells are almost free to allocate, since only the complete grid is allocated, and that in one step. Querying a grid with mostly empty cells is cheaper, the only unknown is how far we need to spiral out.

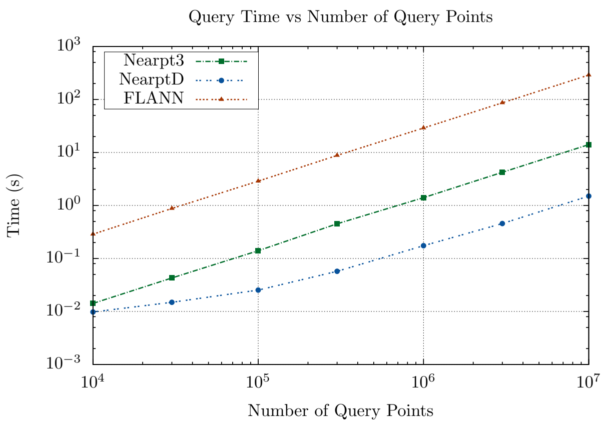


Figure 4: Time to complete queries on 10^8 fixed points vs number of queries.

Figure 4 shows the speedup NEARPTD has as the number of query points increases, with 10^8 fixed points. FLANN and Nearpt3 take $28.9\mu s$ and $1.4\mu s$ per query point, respectively, while NEARPTD takes $1.0\mu s$ per query on small numbers of queries and $0.15\mu s$ per query on larger numbers of queries.

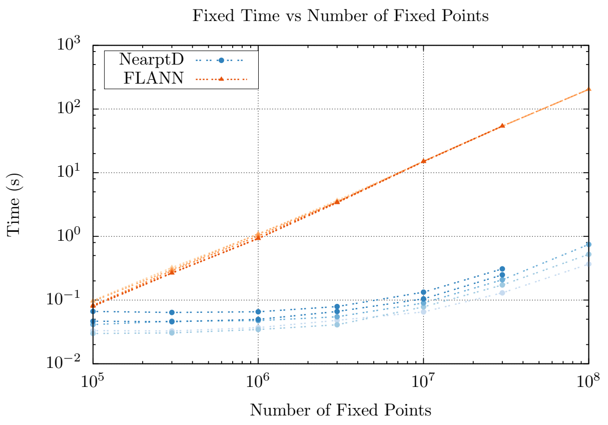


Figure 5: Time to preprocess fixed points into a search structure for varying number of dimensions. The darkest line is 6 dimensional data, with lighter colors indicating lower dimensions, down to 2.

For preprocessing fixed points, NEARPTD exhibits over two orders of magnitude speedup vs FLANN, even for higher dimensional data, as shown in Figure 5. At 5 and 6 dimensions, the 10^8 dataset did not fit into the GPU memory, so it is not shown. Preprocessing points into a uniform grid is largely independent of the dimensionality of the data, for both FLANN and NEARPTD.

Concerning the limited size of the GPU's memory: There will always be datasets too big to fit into the available memory. However, that occurs less often than is generally realized. Later in 2016, Nvidia GPUs with 32GB of memory are expected to become available. In addition, the bandwidth between the GPU and the CPU is increasing, so that the cost of the GPU accessing the CPU memory is shrinking. Indeed, one problem with our research into designing parallel algorithms on GPUs today, to process the large datasets expected in the future, is finding large real test datasets today.

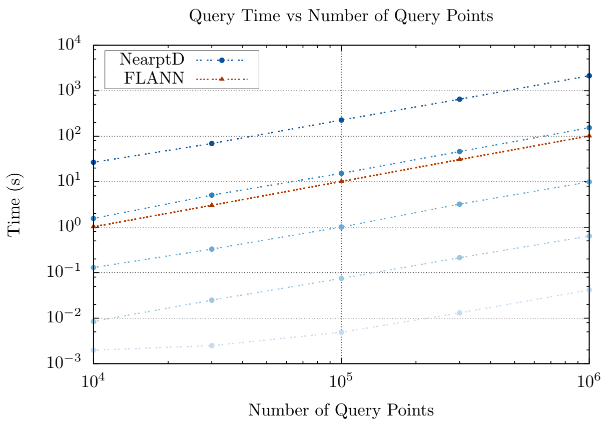


Figure 6: Time to perform queries on 10^7 fixed points for varying number of dimensions. The darkest line is 6 dimensional data, with lighter colors indicating lower dimensions, down to 2.

Figure 6 shows the time to perform queries against 10^7 fixed points for dimensions 2 to 6. NEARPTD exhibits an order of magnitude slowdown in query time for each extra dimension. While NEARPTD is significantly faster than FLANN in 2 to 4 dimensions, it performs worse as dimensionality increases. Query times for FLANN are completely independent of the dimensionality of the data.

6 Space Complexity

N_f	Fixed	NEARPTD	Nearpt3	FLANN
1M	5.7	115.9/137.7	36.9	263.8
10M	57.2	173.3/225.6	156.8	2594.1
100M	572.2	682.4/2223.6	1358.4	25897.6

Table 2: Comparison of total memory footprint of different programs, in MB, as well as the size of the fixed points. NEARPTD values are given as CPU/GPU memory usage.

Besides speed, another benefit of a uniform grid is the relatively small memory footprint to maintain the search structure. As the number of fixed points grows, almost the entirety of host memory used by NearptD is dedicated to simply holding the fixed points. The uniform grid is constructed entirely on the GPU, as well as a copy of the fixed points. FLANN requires an order of magnitude more memory to hold the kd-tree in memory, almost exceeding the amount of host memory available in the largest test case.

7 Future Work

The main limiting factor for the data size NearptD can handle is the memory of the GPU. While the largest real world dataset, *stmatthew*, fits into the 6GB of memory on the test machine, higher dimensions can reduce the effective maximum size of the data that can be processed on the GPU. Although GPU memory is constantly increasing, modifying the NearptD algorithm to preprocess the data in chunks that could fit into GPU memory would allow for arbitrarily large datasets to be processed, especially in higher dimensions. Another possible solution would be to utilize Unified Memory in CUDA along with Thrust to process datasets too big to fit into GPU memory, but this could lead to lower performance with the high cost of moving large amounts of data between the CPU and GPU. Extending NearptD to utilize multiple GPUs could also help in processing large datasets.

If the Thrust library implements C++11 variadic templates for tuples, NearptD could be refactored to use variadic templates, which would remove the need for template specialization. This could reduce the compilation time and make the code more straightforward.

NearptD could also be extended to a k nearest neighbor search by simply extending the query scheme to continue searching until the k nearest neighbors are found. For fast case queries where the cell contains at least k fixed points, this does not increase the running time in any significant manner. If a cell is empty or contains less than k points, a slow case queries could be performed until k fixed points are found, falling back on exhaustive querying only when necessary.

8 Conclusion

This paper presented NearptD as an improvement on more common nearest neighbor libraries that utilize kd-trees to preprocess data. The uniform grid used by NearptD has lower time and space complexity compared to traditional kd-trees and by utilizing the GPU, NearptD exhibits an order of magnitude speedup for larger datasets over existing libraries for both preprocessing and querying. On a dataset with over 184 million points, each point can be preprocessed into a search structure in just $3.5ns$. When performing 10 million queries on 100 million points, queries completed in an average of $0.15\mu s$. NearptD shows that a non hierarchical search structure can enable exact nearest neighbor searching to outperform even approximate searching using kd-trees.

The broader lesson from NearptD is that, counter-intuively, simple data structures work better to process large datasets in parallel. An implementation of this code is freely available for nonprofit research and education at github.com/Lucky1313/NearptD.

9 Acknowledgments

This research was supported by NSF grant CCR-0306502. We are grateful to be able to use datasets from the Stanford University Computer Graphics Laboratory, including the Stanford Digital Michelangelo Project Archive, Georgia Institute of Technology's Large Geometric Models Archive, and the University of North Carolina's UNC Chapel Hill Walkthru Project.

References

[1] Varol Akman, W. Randolph Franklin, Mohan Kankanhalli and Chandrasekhar Narayanaswami Geometric Computing and the Uniform Grid Data Technique *Computer Aided Design* 21(7):410-420, 1989.

[2] W. Randolph Franklin Nearest Point Query on 184M Points in E^3 with a Uniform Grid *Canadian Conference on Computational Geometry* 17:239-242, 2005.

[3] Marius Muja and David G. Lowe Scalable Nearest Neighbor Algorithms for High Dimensional Data *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 36, 2014.

[4] Jon Louis Bentley Multidimensional binary search tress used for associative searching *Communications of the ACM* 18(9):509-517, 1975.

[5] Fukunaga, Keinosuke, and Patrenahalli M. Narendra A branch and bound algorithm for computing k-nearest neighbors *Computers, IEEE Transactions on* 100(7):750-753, 1975.

[6] Ting Liu, Andrew W. Moore, Alexander Gray and Ke Yang An Investigation of Practical Approximate Nearest Neighbor Algorithms *Advances in neural information processing systems* 825-832, 2004.

[7] CGAL The CGAL home page *http://www.cgal.org/* 2016.

[8] Marc Levoy The Digital Michelangelo Project *http://graphics.stanford.edu/projects/mich/* 2003.

[9] Marc Levoy The Stanford 3D Scanning Repository *http://graphics.stanford.edu/data/3Dscanrep/* 2005.

[10] Greg Turk and Brendan Mullins Large Geometric Models Archive *http://www.cc.gatech.edu/projects/large_models/* 2003.

[11] UNC Chapel Hill Walkthru Project Complete Power Plant Model *http://www.cs.unc.edu/~walk/* 1997.

Realizing Farthest-Point Voronoi Diagrams

Therese Biedl [*] Carsten Grimm [†‡] Leonidas Palios [§] Jonathan Shewchuk [¶] Sander Verdonschot [‖]

Abstract

The farthest-point Voronoi diagram of a set of n sites is a tree with n leaves. We investigate whether arbitrary trees can be realized as farthest-point Voronoi diagrams. Given an abstract ordered tree T with n leaves and prescribed edge lengths, we produce a set of n sites S in $O(n)$ time such that the farthest-point Voronoi diagram of S represents T. We generalize this algorithm to smooth, strictly convex, symmetric distance functions. Lastly, given a subdivision Z of $\mathbb{R}^k$ with k a small constant, we check in linear time whether Z realizes a k-dimensional farthest-point Voronoi diagram.

1 Background

In 1999, Liotta and Meijer posed the following question: Given a tree T, can one draw T in the plane so that the resulting embedding is the Voronoi diagram of some set of sites in the plane? They consider the *ordered model*: The tree T is given as an abstract ordered tree, i.e., as a set of vertices, a set of edges, and a cyclic order of the of the edges incident to each vertex. We are searching for a set of sites S such that the vertices and edges of the Voronoi diagram of S form an embedding of T that respects the cyclic order of the edges around each vertex in T. Liotta and Meijer showed that every ordered tree can be realized as a Voronoi diagram [7, 8].

Quite related to this is the *Inverse Voronoi Problem*, which asks the question in the *geometric model*. Here we are given a tree (or more generally a graph) and also a drawing of it, i.e., coordinates for all interior nodes and rays to infinity for all edges to leaves. We are searching for a set of sites S such that the Voronoi diagram of S is exactly this tree with this drawing. The problem was introduced by Ash and Bolker [4] and the question can be answered in linear time [6], even if the tree has vertices of degree exceeding three [5].

A number of variants have been studied. Aloupis et al. [3] posed an extension-version of the Inverse Voronoi Problem. Other papers study the straight skeleton, rather than the Voronoi diagram. Aichholzer et al. resolved this for the ordered model [2], and (with different coauthors) for the ordered model where edge directions are given [1]. The Inverse Straight Skeleton Problem was resolved by Biedl et al. [5].

Our results. We ask whether trees can be realized by yet another computational geometry construct, namely, the *farthest-point Voronoi diagram* (defined below). We consider both models and obtain the following results.

Ordered Model: Similarly as in [3, 8], for the ordered model the answer is always "yes". Thus for any given ordered tree T, we can find a set of sites S in convex position such that the farthest-point Voronoi diagram of S is T, with the edges in the specified order. In contrast to related results, we can also realize edge lengths, i.e., if each interior edge e is assigned a positive weight $w(e)$, then we can find sites so that e has length $w(e)$.

We give the construction first for the "normal" (Euclidean) farthest-point Voronoi diagram, and then generalize it to any convex distance function for which the unit circle is smooth and strictly convex.

Geometric Model: Similarly as in [5, 6], for the geometric model not every geometric tree can be realized. Nonetheless, one can test in polynomial time whether for a given geometric tree T there exists a set of points whose farthest-point Voronoi diagram is T. If so, then the set of sites is not always unique, but it can be described as the solution space of a linear program.

We describe this result for arbitrary fixed dimension. For a given convex subdivision Z of $\mathbb{R}^k$ with n cells, we formulate a linear program with k variables that tests whether there exists a set of n sites whose farthest-point Voronoi diagram realizes Z. This linear program can be solved in linear time if k is a small constant [10].

2 Preliminaries

Let S be a set of sites and let p be a point in the plane. Let $\mathrm{F}_S(p)$ be the smallest disc centered at p that con-

[*] David R. Cheriton School of Computer Science, University of Waterloo, Canada, biedl@uwaterloo.ca. Supported by NSERC.

[†] Computational Geometry Lab, School of Computer Science, Carleton University, Ottawa, Ontario, Canada.

[‡] Institut für Simulation und Graphik, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany carsten.grimm@ovgu.de.

[§] Dept. of Computer Science and Engineering, University of Ioannina, Greece, palios@cse.uoi.gr.

[¶] Computer Science Division, University of California, Berkeley, CA, USA, jrs@cs.berkeley.edu. Supported by the National Science Foundation under Award CCF-1423560.

[‖] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada, sander@cg.scs.carleton.ca. Supported by NSERC and the Ontario Ministry of Research and Innovation.

tains all sites in S; we call this the *full disc* of p with respect to S. For a set S of sites, the *farthest-point Voronoi diagram* of S, denoted by fVor(S), is defined as follows: A point p is a vertex of fVor(S) if and only if $\mathrm{F}_S(p)$ passes through three or more sites in S. A point p is located in the relative interior of an edge of fVor(S) if and only if $\mathrm{F}_S(p)$ passes through exactly two sites in S. fVor(S) divides the plane into convex cells, and one easily verifies that each cell consists of all points that are farthest from one site s. We say that site s is *relevant* if there is a point in the plane for which s is a farthest point, and *proper* if there is a point for which s is the *unique* farthest point. (For strictly convex distance functions "relevant" and "proper" are the same thing; see Section 4.2 for more details.)

The structure of the farthest-point Voronoi diagram is closely related to the convex hull CH(S) of S: (i) A site $s \in S$ is proper if and only if s is an extreme point of S. (ii) Two sites s and s' are adjacent along CH(S) if and only if the farthest-point Voronoi cells of s and s' share an unbounded edge (ray or line). (iii) The circular order of the sites along CH(S) is the circular order of the farthest-point Voronoi cells in fVor(S).

3 Ordered Trees

Consider the farthest-point Voronoi diagram fVor(S) of a set S of sites in the plane. We introduce symbolic vertices as endpoints for the unbounded edges of fVor(S). We say that fVor(S) is a *realization* of an ordered tree T if T is isomorphic to the abstract ordered tree formed by the Voronoi vertices, the symbolic vertices and the Voronoi edges of fVor(S). In the following, we consider only ordered trees without degree two vertices, since there are no degree two vertices in a farthest-point Voronoi diagram.

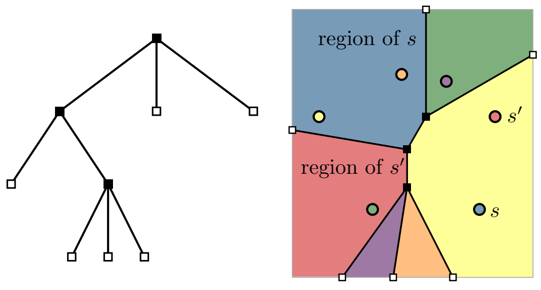


Figure 1: (a) An ordered tree T; (b) a realization of T as a farthest-point Voronoi diagram. Empty squares are leaves; also symbolic endpoints of unbounded edges.

Given an ordered tree T, we seek to determine a set S of sites in the plane such that fVor(S) realizes T. We proceed in an incremental fashion where we place sites to create the internal vertices of T one by one.

Realizing a star. We begin with an ordered tree T_1 with one internal node v of degree ℓ. We realize T_1 by placing ℓ sites $s_1, s_2, \dots, s_\ell$ on a unit circle C centered at the origin. The origin becomes the Voronoi vertex that we identify with v.

Any subsequent site s has to be placed at a location that is *safe* for the current sites S in the following sense: Every vertex in the diagram for S remains a vertex in the diagram for $S \cup \{s\}$ and every bounded edge in the diagram for S remains a bounded edge in the diagram for $S \cup \{s\}$. It is acceptable for a safe site to increase the degree of a vertex of the diagram. After the initial step, every point s strictly inside C is safe.[1]

On the other hand, any subsequent site s must be proper. Any site outside the convex hull CH(S) is proper.[1] Thus, all additional sites will be placed in the *lunes* that remain when we remove CH($\{s_1, \dots, s_\ell\}$) from the disc bounded by C.

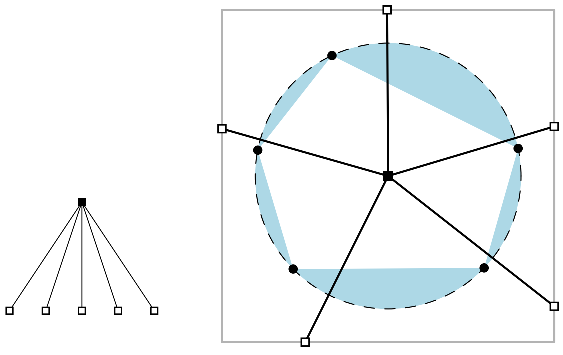


Figure 2: (a) An ordered tree with one internal vertex; (b) a realization of that ordered tree as a farthest-point Voronoi diagram. All subsequent sites will be placed in the lunes (shaded blue).

Realizing larger trees. Suppose we can realize every ordered tree with $k \geq 1$ internal vertices as farthest-point Voronoi diagram, for some $k \in \mathbb{N}$. Consider an ordered tree T_{k+1} with $k+1$ internal vertices. There is an internal vertex v in T_{k+1} that becomes a leaf when all leaves adjacent to v are deleted. Let T_k be the tree that results from deleting the leaves adjacent to v. Since T_k is an ordered tree with k internal vertices, we can find a set S of sites such that fVor(S) is a realization of T_k. We seek to place additional sites such that the resulting farthest-point Voronoi diagram realizes T_{k+1}.

Vertex v is a leaf in T_k, hence corresponds to a symbolic endpoint in fVor(S) that lies on a ray r. Let u be the internal vertex at which r ends (hence u is the neighbor of v in T_k). Ray r separates the regions of two

[1]In the appendix, we provide full proofs for the claim for smooth, strictly convex, symmetric distance functions.

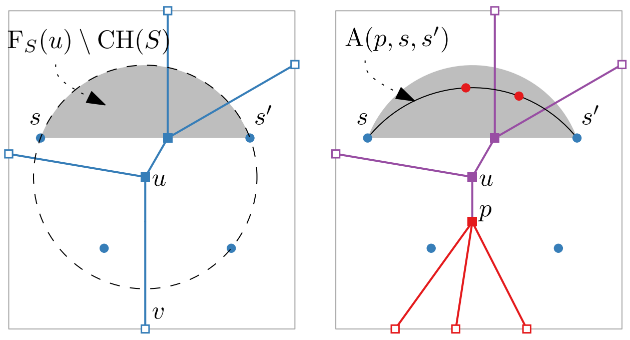


Figure 3: Extending the realization of an ordered tree.

sites s and s', so by the definition of $\mathrm{fVor}(S)$, for every point $p \in r$ the full disc $\mathrm{F}_S(p)$ goes through s and s' and contains all other sites in its interior.

We want to place sites such that we create a Voronoi vertex at some point p on ray r (and then assign this point to v). To create a Voronoi vertex at p, we have to place a new site s'' on the boundary of $\mathrm{F}_S(p)$. To make its region appear between the ones of s and s', we should place s'' on the (shorter) circular arc $\mathrm{A}(p, s, s')$ from s to s' along $\mathrm{F}_S(p)$. If v is adjacent to ℓ leaves in T_{k+1} ($\ell > 1$ since we have no vertices of degree 2), then we should place $\ell - 1$ new sites along $\mathrm{A}(p, s, s')$.

Observe that the choice of p is arbitrary, as long as it is on the ray. We can therefore choose the distance between u and p (the future location of v) and realize any specified edge length of (u, v). To summarize, we can realize every ordered tree T as a farthest-point Voronoi diagram by placing the sites for some vertex of T on a circle and then repeatedly expanding the resulting farthest-point Voronoi diagram by placing the next vertex on the appropriate ray and sites for it on the corresponding arc. We place n sites for an ordered tree with n leaves. The entire construction takes $O(n)$ time, since computing the coordinates of each site takes constant time in the real RAM model of computation.

Theorem 1 *For every ordered tree T with $n \geq 2$ leaves, without vertices of degree two, and with edge lengths for edges connecting non-leaves, we can find a set S of n sites in $O(n)$ time such that the farthest-point Voronoi diagram of S is a realization of T where every bounded edge in* $\mathrm{fVor}(S)$ *has a prescribed length.*

4 Other Distance Functions

Voronoi diagrams and farthest-point Voronoi diagrams can naturally be generalized to a wider class of distance functions defined as follows: a distance function d is specified by giving its *unit circle* C_d, i.e., all those points considered to have distance one from the origin. We assume throughout that d is convex and symmetric, i.e., C_d is a closed curve that bounds a convex shape that has 2-fold rotational symmetry about the origin.

To measure distances, we use *homothets* of C_d, i.e., scaled and translated copies. We call such a homothet a *d-disc* and say that it is *centered at p* if the origin was translated to p. Given a set S of sites, let the *full d-disc* $\mathrm{F}^d_S(p)$ be the smallest d-disc centered at p that encloses all sites of S. The *d-farthest-point Voronoi diagram* of a set S of sites, denoted by $\mathrm{fVor}_d(S)$, is defined as before by letting p be a vertex (resp. interior point of an edge) if and only if $\mathrm{F}^d_S(p)$ contains three (resp. two) sites.[2]

We briefly argue that this indeed expresses "farthest" correctly. For two points p and q, the distance $d(p, q)$ (with respect to the distance function defined by C_d) is defined to be the smallest scaling factor at which a d-disc centered at p touches q. Since d is symmetric, we have $d(p, q) = d(q, p)$. In particular, a site $s \in S$ is farthest from the point p if s is on the boundary of $\mathrm{F}^d_S(p)$. If p is a point on an edge of $\mathrm{fVor}_d(S)$, then by definition there are two sites s, s' on $\mathrm{F}^d_S(p)$. Thus p is equidistant from s, s' and all other sites are no farther. Hence any edge of $\mathrm{fVor}_d(S)$ bounds a region where all points have the same farthest point. See Figure 4.

4.1 Smooth Strictly Convex Symmetric Distances

We call a distance function d *strictly convex* if the boundary of C_d contains no line segments, and *smooth* if every point on the boundary of C_d has a unique tangent. We now show that we can realize arbitrary ordered trees as d-farthest-point Voronoi diagram for any smooth and strictly convex symmetric distance function d.

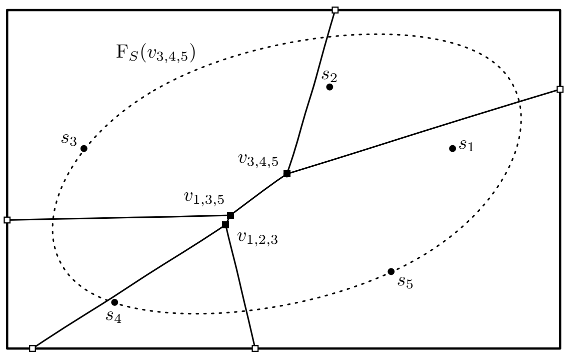


Figure 4: A d-farthest-point Voronoi diagram.

The approach is the same as for the Euclidean case, with the only change that we use C_d, rather than geometric circles, to define arcs to place sites on. Thus, for a tree T_1 with a single interior node v with ℓ incident leaves, place ℓ sites on the unit circle C_d. The origin becomes the Voronoi vertex that we identify with v.

[2]For non-symmetric convex distances, the full d-disc is a mirrored homothet of C_d and the correspondence to vertices and edges of the diagram no longer holds [9].

To create sites for a tree T_{k+1} with $k+1$ interior nodes, find one node v that is adjacent to only one other interior node u, and remove all incident leaves of v. Recursively find sites for the resulting tree T_k. Find the unbounded edge r from u on which the symbolic endpoint for v resides, and pick an arbitrary point p on it. Find the full d-disc $\mathrm{F}^d_S(p)$; this contains the two sites s, s' whose farthest regions meet at edge r on their boundaries. Turn p into a vertex of the d-farthest-point Voronoi diagram by placing sites at the shorter arc of $\mathrm{F}^d_S(p)$, placing $\ell - 1$ sites if v was incident to ℓ leaves.

It remains to argue that this is correct, i.e., that all newly placed sites are safe and proper. In a nutshell, this holds because they are strictly inside $\mathrm{F}^d_S(u)$ and strictly outside $\mathrm{CH}(S)$. We give a proof in the appendix.

Theorem 2 *Let d be a smooth and strictly convex symmetric distance function. For every ordered tree T with $n \geq 2$ leaves, without vertices of degree two, and with edge lengths for edges connecting non-leaves, we can find a set S of n sites in $O(n)$ time such that the d-farthest-point Voronoi diagram of S is a realization of T where every bounded edge has its prescribed length.*

4.2 Polygonal Convex Symmetric Distances

We now illustrate some of the challenges that arise when our distance function is not smooth or not strictly convex. Unlike for strictly convex distances, the *d-bisector* of two sites s and s' (i.e., the set of all points that are equidistant from s and s' with respect to d) is not necessarily homeomorphic to a line, and indeed, may be a 2-dimensional region. Ma [9] shows that this occurs precisely when the line segment ss' is parallel to a line segment on the boundary of the unit circle C_d that defined d. This limits our ability to realize ordered trees as d-farthest-point Voronoi diagrams when d is *polygonal*, i.e., C_d is a k-sided convex polygon.

Theorem 3 *Let d be a convex distance function defined by a polygon with k edges and let T be a tree with more than k leaves. There is no set of sites S such that the d-farthest-point Voronoi diagram of S realizes T.*

Proof. For every edge e of the unit circle C_d, at most one site can be extreme in the direction normal to e. More precisely, for any half-plane $h \supset S$ whose bounding line ℓ is parallel to e, there is at most one site on ℓ—otherwise $\mathrm{fVor}_d(S)$ is not a tree. So if $\mathrm{fVor}_d(S)$ is a tree, then at most k sites in S have nonempty cells, hence the tree has at most k leaves. Therefore, we cannot realize trees with more than k leaves. □

For example, for the L_1-distance and the L_∞-distance, the unit circle C_d is a 4-sided polygon, so no tree with more than four leaves can be realized as farthest-point Voronoi diagrams under these distances.

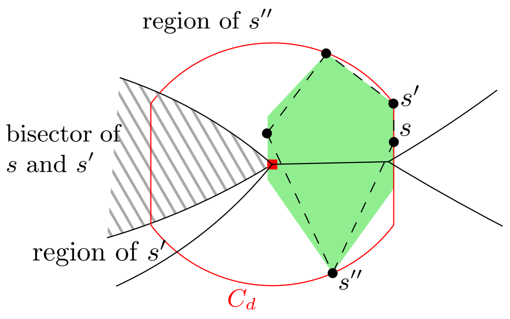


Figure 5: If some portion of C_d (red) is a line segment, two sites on that line segment (e.g., s, s') can have a two-dimensional bisector (grey region). The generalized convex hull $\mathcal{H}(S)$ (green) may strictly include the convex hull (dashed). Here, the site s is a vertex of the (ordinary) convex hull but s is not proper: removing s leaves the generalized convex hull unchanged.

A second problem with distance functions that are not strictly convex is that not all extreme points of the convex hull are proper; for example point s in Figure 5 is an extreme point of $\mathrm{CH}(S)$ but any point p for which s is farthest also has s' as farthest point.

However, we can prove a similar relationship. Let $\mathcal{H}(S)$ be the intersection of all d-discs that contain S. We refer to $\mathcal{H}(S)$ as the *generalized convex hull* of S. We call a site s an *extreme point* of $\mathcal{H}(S)$ if $\mathcal{H}(S) \neq \mathcal{H}(S \setminus \{s\})$. We give in the appendix the following characterization:

Lemma 4 *A site s in S is proper if and only if s is an extreme point of the generalized convex hull $\mathcal{H}(S)$.*

We may attempt to follow the steps of the algorithm from the Euclidean setting, in the hope of always finding proper sites. We now show that this can fail. As before define v, u, r, s, s' in the expansion step. Presume we are in a situation where $\mathrm{F}^d_S(u)$ contains s, s' on adjacent straight-line edges. Then the generalized hull $\mathcal{H}(S)$ coincides with $\mathrm{F}^d_S(u)$ on the stretch between s and s'. Thus, the region where we placed sites for strictly convex distances is empty, giving no suitable, safe, proper candidates. Put differently, we cannot longer realize ordered trees in the carefree online fashion we use for the Euclidean distance. Rather, we need to know the ordered tree in advance and we need to decide *a priori* which site will occupy which edge of C_d. We conjecture that with a judicious choice, we can realize every tree with at most k leaves if C_d is a k-sided polygon, but this remains an open problem. Without giving details, we note that all ordered trees *can* be realized by any convex symmetric distance function for which C_d is strictly convex and smooth in at least one region, by placing all initial sites and later additions only within that part of C_d.

5 Geometric Trees

In this section, we study how to test whether a specified geometric tree is a farthest-point Voronoi diagram in the Euclidean metric. We are given a tree with a fixed drawing in the plane, with the leaves at infinity. Reinterpreting this, we are given a subdivision of the plane into cells, and we ask whether there exists a set of sites whose farthest-point Voronoi diagram comprises these cells. An affirmative answer is possible only if every cell of the subdivision is convex and unbounded.

Our approach generalizes to arbitrary dimension k, so assume that we are given a convex subdivision Z of $\mathbb{R}^k$, where each cell in Z is a convex, unbounded polyhedron. We wish to determine whether Z is the farthest-point Voronoi diagram of some set S of sites. Each cell in Z has some number of $(k-1)$-dimensional facets (e.g., edges if $k=2$), and we assume that for each such facet f we know a unit normal vector n_f. Thus, for each facet f, its affine hull has the form $\{p : \langle n_f, p\rangle = \alpha_f\}$, where α_f is a suitable scalar. Let $f_{\sigma\tau}$ denote a facet whose incident cells are σ and τ, where n_f is directed from τ into σ and thus σ is the cell whose interior points have a positive signed distance from $f_{\sigma\tau}$ (i.e., $\langle n_f, p\rangle \geq \alpha_f$ for all points $p \in \sigma$.)

Suppose Z can be realized as farthest-point Voronoi diagram. In this realization each cell σ is assigned a site $\rho(\sigma)$ such that the points in σ are exactly those points for which $\rho(\sigma)$ is the farthest site. We will describe any (putative) realization as such a function $\rho(\sigma)$.

The following result holds for realizations of farthest-point Voronoi diagrams in arbitrary dimension (and also for ordinary Voronoi diagrams [5]).

Lemma 5 (bisector condition) *Let ρ be a realization of Z. For every facet $f_{\sigma\tau}$ in Z, the affine hull of $f_{\sigma\tau}$ must be the bisector of $\rho(\sigma)$ and $\rho(\tau)$.*

Hence, given $\rho(\sigma)$ we can compute $\rho(\tau)$ by reflecting $\rho(\sigma)$ about f, i.e., $\rho(\tau) = \rho(\sigma) - 2(\langle n_f, \rho(\sigma)\rangle - \alpha_f)n_f$. As this is an affine equation in $\rho(\sigma)$, it can be expressed in the matrix form

$$\begin{bmatrix} \rho(\tau) \\ 1 \end{bmatrix} = R_{\sigma\tau} \begin{bmatrix} \rho(\sigma) \\ 1 \end{bmatrix}$$

where $R_{\sigma\tau}$ is a $(k+1)\times(k+1)$ matrix determined solely by the normal vector and scalar of the face $f_{\sigma\tau}$. Thus we have a system of $k+1$ equations for each facet of Z. Let $\bar{\tau}$ denote the vector $[\rho(\tau) \;\; 1]^{\mathrm{T}}$, so the equation becomes $\bar{\tau} = R_{\sigma\tau}\bar{\sigma}$.

We need a second condition. In the ordinary Voronoi diagram, a site must lie inside the cell of points for which it is the nearest site. For the farthest-point Voronoi diagram, we need a condition that is essentially the inverse.

Lemma 6 (outside condition) *Let ρ be a realization of a subdivision Z. For every facet f incident to a cell σ, the affine hull H of f has the cell σ on one side and the site $\rho(\sigma)$ on the other.*

Proof. Say the facet is $f = f_{\sigma\tau}$. According to the bisector condition, $\rho(\sigma)$ and $\rho(\tau)$ are on opposite sides of H, and every point on the same side of H as $\rho(\sigma)$ is closer to $\rho(\sigma)$ than it is to $\rho(\tau)$. No point $p \in \sigma$ can lie on the same side of H as $\rho(\sigma)$, as p's farthest site cannot be $\rho(\sigma)$. □

We express the outside condition as the two inequalities

$$\langle n_{\sigma\tau}, \rho(\sigma)\rangle \leq \alpha_{\sigma\tau} \leq \langle n_{\sigma\tau}, \rho(\tau)\rangle,$$

where, as before, $n_{\sigma\tau}$ is a unit vector normal to $f_{\sigma\tau}$ such that $\langle n_{\sigma\tau}, p\rangle \geq \alpha_{\sigma\tau}$ for all points $p \in \sigma$ and $\langle n_{\sigma\tau}, p\rangle \leq \alpha_{\sigma\tau}$ for all points $p \in \tau$. Crucial to our testing routine is the following.

Theorem 7 *Let Z be a convex subdivision of $\mathbb{R}^k$. Let $S = \rho(\cdot)$ be an assignment of sites to cells in Z. Then Z is the farthest-point Voronoi diagram of S if and only if the bisector condition and the outside condition holds for every facet of Z.*

Proof. Necessity has been shown already. Suppose for the sake of contradiction that the two conditions hold, yet Z is not the farthest-point Voronoi diagram of S. Then there exists some cell σ of Z containing an interior point p for which the farthest site in S is not $\rho(\sigma)$ but instead some other site $\rho(\tau)$ assigned to a cell τ.

Shoot a ray from the interior of τ toward p, and let $f_{\tau\omega}$ be the first facet (breaking ties arbitrarily) of τ that the ray strikes. The ray strikes $f_{\tau\omega}$ before reaching p, as p is in the interior of a cell other than τ; therefore, p is on ω's side of the affine hull of $f_{\tau\omega}$. By the outside condition therefore p is *not* on $\rho(\omega)$'s side of the affine hull of $f_{\tau\omega}$. As $f_{\tau\omega}$ bisects $\rho(\omega)$ and $\rho(\tau)$ by the bisector condition, therefore p is closer to $\rho(\tau)$ than to $\rho(\omega)$, contradicting the fact that $\rho(\tau)$ is the site in S that is farthest from p. The result follows. □

Theorem 7 implies that we can answer the question by finding a set S of sites that satisfy all the bisector conditions and outside conditions—one of the former and two of the latter for each facet of Z—or by showing that no such set of sites exists. As the bisector conditions are linear equations and the outside conditions are linear inequalities, the question reduces to finding a feasible point of a linear program.

For efficiency, we recommend reducing the linear program to k variables prior to solution by performing substitutions of the bisector conditions. We achieve this with a *propagation* procedure that exploits the dual graph of the convex subdivision Z, as Biedl et al. [5] do for the ordinary Voronoi diagram. Form the dual graph G of Z: G's vertices correspond to the k-cells of

Z and G's edges correspond to Z's facets. Choose a distinguished k-cell σ in Z (hence a distinguished node in the graph). The variables in our system are the coordinates of the putative site $\rho(\sigma)$, hence the first k entries of vector $\bar{\sigma}$. Perform a depth-first search of G, during which we express the coordinates of every other site as a linear combination of $\bar{\sigma}$'s coordinates by composing reflections of the form $\bar{\tau} = R_{\omega\tau}\bar{\omega}$. Composing these reflections is simply matrix multiplication; thus we obtain a linear relationship of the form $\bar{\tau} = R'_{\sigma\tau}\bar{\sigma}$ for every cell τ, even those that do not share a facet with σ. ($R'_{\sigma\tau} = R_{\sigma\tau}$ if (σ, τ) is an edge of G.)

Next, consider the edges of G that the depth-first search did not traverse. Each such edge (ω, τ) corresponds to a facet of Z that introduces an additional reflection equation of the form $\bar{\omega} = R_{\tau\omega}\bar{\tau}$, which hence becomes another linear equality constraint imposed on $\bar{\sigma}$: $R'_{\sigma\omega}\bar{\sigma} = R_{\tau\omega}R'_{\sigma\tau}\bar{\sigma}$. However, these constraints are often redundant or trivial (i.e., $\bar{\sigma} = \bar{\sigma}$). We can stack these linear equations ($k+1$ equations per untraversed edge) in the form of a matrix equation $M\bar{\sigma} = b$, where M has $k+1$ columns and $O(mk)$ rows, and m is the number of facets in Z. This linear system hence defines an affine subspace Λ of vectors $\bar{\sigma}$ that are compatible with the bisector condition. Typically Λ is a single point or empty, but it could have dimension as high as k.

The outside condition imposes another system of $O(mk)$ linear inequalities, two per facet. If Λ is a single point, it is now a simple matter to check whether it satisfies all these inequalities. If Λ is a larger subspace, we restrict the inequalities to the subspace Λ and solve the consequent linear program. Any feasible point can be used for $\bar{\sigma}$ (hence gives the site $\rho(v)$), and we can compute the other sites by applying the reflection equations. The solution space may have dimension up to k, as Figure 6 illustrates. If $\Lambda = \emptyset$ or the linear program is infeasible, Z is not a farthest-point Voronoi diagram of any set of sites.

Suppose Z has n cells and m facets in k dimensions. It takes $O(nk^3)$ time to compute the propagation matrices $R'_{\sigma\tau}$ (accounting for fewer than n multiplications of $(k+1) \times (k+1)$ matrices); $O(mk^3)$ time to compute the remaining equations and inequalities due to the bisector and outside conditions; and $O(f(k)(n+m))$ time to solve the linear program where $f(\cdot)$ is a function (typically exponential) [10]. As the size of the input subdivision Z is $\Omega(m+n)$, the total running time is linear in the input size if the dimension k is a small constant.

Theorem 8 *Given a convex subdivision Z of $\mathbb{R}^k$, where k is a small constant, we can test in linear time whether there exists a set of sites whose farthest-point Voronoi diagram is Z.*

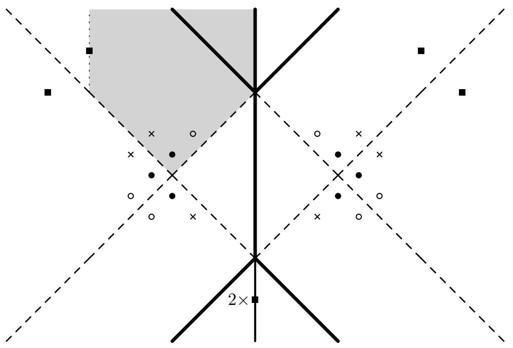


Figure 6: A farthest-point Voronoi diagram (thick edges) and three sets of sites (discs, circles, crosses) that realize it. Once one site is fixed in the open gray cell $\mathcal{G}$, the others follow by reflection at the bisectors (thick or dashed). Sites on the boundary of $\mathcal{G}$ (e.g., the square) yield sites that coincide, and sites outside $\mathcal{G}$ generate sites that violate the outside condition.

References

[1] O. Aichholzer, T. Biedl, T. Hackl, M. Held, S. Huber, P. Palfrader, and B. Vogtenhuber. Representing directed trees as straight skeletons. In *Graph Drawing and Network Visualization (GD '15)*, pages 335–347, 2015.

[2] O. Aichholzer, H. Cheng, S. L. Devadoss, T. Hackl, S. Huber, B. Li, and A. Risteski. What makes a tree a straight skeleton? In *Canadian Conference on Computational Geometry (CCCG '12)*, pages 253–258, 2012.

[3] G. Aloupis, H. Pérez-Rosés, G. Pineda-Villavicencio, P. Taslakian, and D. Trinchet-Almaguer. Fitting Voronoi diagrams to planar tesselations. In *Intl. Workshop on Combinatorial Algorithms (IWOCA 2013)*, pages 349–361, 2013.

[4] P. Ash and E. Bolker. Recognizing Dirichlet tesselations. *Geometriae Dedicata*, 19:175–206, 1985.

[5] T. Biedl, M. Held, and S. Huber. Recognizing straight skeletons and Voronoi diagrams and reconstructing their input. In *10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2013)*, pages 37–46, 2013.

[6] D. Hartvigsen. Recognizing Voronoi diagrams with linear programming. *ORSA J. Comput.*, 4:369–374, 1992.

[7] G. Liotta and H. Meijer. Voronoi drawings of trees. In *Graph Drawing (GD '99)*, pages 369–378, 1999.

[8] G. Liotta and H. Meijer. Voronoi drawings of trees. *Comput. Geom.*, 24(3):147–178, 2003.

[9] L. Ma. *Bisectors and Voronoi Diagrams for Convex Distance Functions.* PhD thesis, FernUniversität Hagen, 2000.

[10] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984.

A Smooth Strictly-Convex Distance Functions

Recall that the distance function d is given by specifying its *unit circle* C_d, a *d-disc* is a homothet of C_d, and the *radius* of a d-disc D is the scaling factor used to obtain D from C_d. In this section, we show in detail that if C_d is strictly convex and smooth, then our algorithm to find sites whose farthest-point Voronoi diagram realizes a given ordered tree T works correctly. There are two things that must be shown: every added site s is *d-proper* (there exists a point p for which s is the unique farthest site) and *d-safe* (all previously placed sites remain d-proper).

A.1 Proper Sites

Recall that an *extreme point* of the convex hull $\mathrm{CH}(S)$ is a site $s \in S$ such that $\mathrm{CH}(S \setminus \{s\})$ is a strict subset of $\mathrm{CH}(S)$. Equivalently, a site $s \in S$ is an extreme point of S if there exists a half-space ℓ that has all points in $S \setminus \{s\}$ in its interior and s in its exterior.

Theorem 9 *Let S be a set of sites in the plane and let d be a smooth strictly convex distance function.*

1. *A site s is d-proper if and only if s is an extreme point of the convex hull of S.*
2. *The regions of two sites s_i and s_j share an unbounded edge if and only if s_i and s_j are consecutive extreme points of the convex hull of S.*
3. *The d-proper sites appear in the same order along the convex hull of S as their corresponding regions in the d-farthest-point Voronoi diagram.*

Proof. To show the first claim, suppose the site s is d-proper. Then there is a point p such that $\mathrm{F}^d_S(p)$ has only the site s on its boundary. Since C_d is convex, the convex hull $\mathrm{CH}(S)$ is contained in $\mathrm{F}^d_S(p)$. Since C_d is strictly convex, $\mathrm{CH}(S)$ intersects $\mathrm{F}^d_S(p)$ only in point s. Hence, $\mathrm{CH}(S \setminus \{s\})$ is strictly inside $\mathrm{F}^d_S(p)$, which proves that $\mathrm{CH}(S \setminus \{s\}) \subset \mathrm{CH}(S)$ and, thus, the site s is an extreme point of the convex hull $\mathrm{CH}(S)$.

Conversely, suppose s is an extreme point of $\mathrm{CH}(S)$, say half-space ℓ separates s from the rest of S. Since C_d is smooth, there exist two points on C_d whose tangent has the same slope as the affine hull of ℓ. By scaling C_d sufficiently much, we can hence find a homothet D of C_d that in the vicinity of one of these points is arbitrarily close to ℓ. Hence D contains $S \setminus \{s\}$ and not s. Scaling D while keeping its center then yields a d-disc with only s on its boundary, proving that the region of s is non-empty.

The proof of (2) and (3) is very similar to part (1) after observing that (s_i, s_j) is an edge of the convex hull if and only if there exists a half-space ℓ that contains all points in $S \setminus \{s_i, s_j\}$ in its interior and s_i, s_j in its exterior. With this we can find an unbounded region of points whose farthest site is either s_i or s_j, and therefore there must be an unbounded edge separating their two regions. □

As we will see below, we always choose the next site(s) to be outside the convex hull of the current sites. As such, all sites that we choose will be d-proper.

A.2 Properties of Homothets

Before proving safety, we need some basic observations about homothets of a strictly convex smooth C_d.

Theorem 10 (Ma [9]) *Let D and D' be two different homothets of a compact convex set C_d. Then the boundaries of D and D' intersect in at most two points, or in a point and a line segment, or in two line segments.*

Corollary 11 *Let D and D' be two different homothets of a strictly convex smooth compact set C_d. Then the boundaries of D and D' intersect at most two points.*

Proof. The claim follows from Theorem 10, since the boundary of a homothet of a strictly convex compact set does not contain any line segments, by definition. □

We say that two curves C, C' *truly intersect* at some point p if they have p in common, and any sufficiently small circle centered at p intersects the curves in four points and in order C, C', C, C'.

Lemma 12 *Let D and D' be two different homothets of a strictly convex smooth compact set C_d. If the boundaries of D and D' intersect in two points a, b, then they truly intersect at both a and b.*

Proof. We consider the situation near a. Since D and D' are smooth, there are unique tangents t_a and t'_a at a for D and for D', respectively. We argue that these tangents have different slopes.

Since C_d is strictly convex, the slope of the tangent determines the point on C_d uniquely, up to reflection through the center-point, and the line from this point to the center-point has the same slope regardless of how we scale or translate C_d. Thus, the line from a to the center-point p of D has the same slope as the line from a to the center-point p' of D', so p, a, p' are all on one line.

Repeating the argument at b, we see that p, b, p' (and therefore also a) are all on one line. But then D and D' must have the same scale-factor (else they could not both contain both a and b), and therefore the same center-point, and so are the same homothet. Contradiction, so t_a and t'_a have different slopes. Since D and D' are smooth, their boundary locally follows the lines along t_a and t'_a, which means that they truly intersect at a. □

Finally we need a rather technical observation, which will be crucial for defining the "lunes" which are used for placing sites safely.

Lemma 13 (Inside-Outside Lemma) *Let a and b two points in the plane and let h and $\bar{h}$ be the half-planes bounded by the line through a and b. Consider two d-discs D and D' such that*

(a) the centers of D and D' both lie in h,

(b) the radius of D' is larger than the radius of D, and

(c) the boundaries of D and D' intersect at a and b.

Then we have the following.

(1) Within the half-plane h, the d-disc D' contains D, i.e., $h \cap D \subset h \cap D'$.

(2) Within the half-plane $\bar{h}$, the d-disc D contains D', i.e., $\bar{h} \cap D' \subset \bar{h} \cap D$.

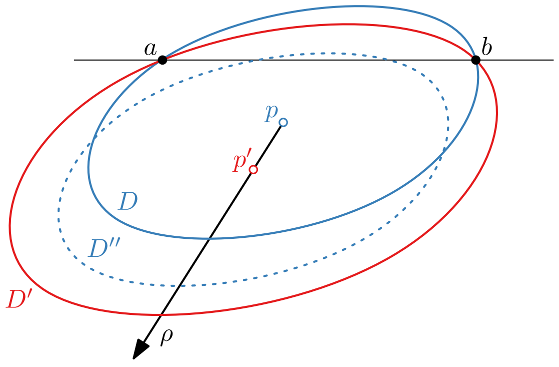


Figure 7: Two d-discs D (blue) and D' (red) that have their centers p and p' on the same side as the line through their two intersection points a and b. The ray ρ from p through p' first hits D, then ρ hits a copy D'' of D centered at p' (dotted, blue), and finally ρ hits D'.

Proof. Let p be the center of D and p' the center of D'. Consider the ray ρ that shoots from p through p'. We argue that ρ hits D strictly before D'.

As illustrated in Figure 7, we place a copy D'' of D centered at p'. The ray ρ hits D before D'', since D'' is a copy of D translated from p to p'. Furthermore, the ray ρ hits D'' strictly before D', since D' is a strictly larger copy of D'' with the same center. This means that the ray ρ hits the boundary of D strictly before the boundary of D'. Since D and D' are strictly convex and homothetic, the boundaries of D and D' cannot have any intersection other than a and b. Therefore, within the half-space h, the boundary of D lies in the interior of D', i.e., $h \cap D \subset h \cap D'$. This proves (1).

To show (2), observe that since the boundaries of D and D' intersect in two points, at both points we have true intersections. Due to (1), we *enter* D as we traverse the boundary of D' from h to $\bar{h}$ through a (or through b). Since the boundaries of D and D' intersect only at a and b, we know that, within $\bar{h}$, the boundary of D' lies in the interior of D, i.e., $\bar{h} \cap D' \subset \bar{h} \cap D$. □

A.3 Lunes and Safe Sites

Let us assume that the sites are numbered $s_1, s_2, \ldots, s_n$ in an arbitrary manner. Let $v_{i,j,k}$ be the point equidistant to sites s_i, s_j, and s_k; and let $e_{i,j}$ be the edge (if any) on the bisector of sites s_i and s_j. Suppose p is a point along an unbounded edge $e_{i,j}$ defined by the sites s_i and s_j, and we want to place a new site s on the d-arc $\mathrm{A}_d(p, s_i, s_j)$ to create a new vertex at some point p. Define the *d-lune* $\mathrm{Lune}_d(s_i, s_j)$ to be the union of all d-arcs $\mathrm{A}_d(p, s_i, s_j)$ such that p is an interior point of ray r. Figure 8 depicts an example of a d-lune.

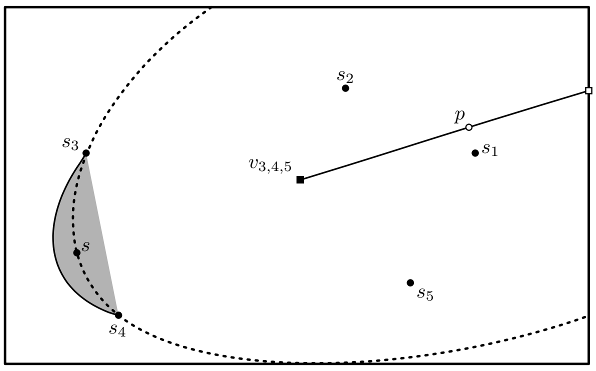


Figure 8: The d-lune $\mathrm{Lune}_d(s_3, s_4)$ for the sites from Figure 4 together with its defining edge $e_{3,4}$. A new site s in this d-lune creates a new vertex at p along $e_{3,4}$, where p is the center of the d-disc through s_3, s_4, and s.

Lemma 14 *For any two consecutive vertices s_i, s_j on* $\mathrm{CH}(S)$*, if $v_{i,j,k}$ is the finite end of edge $e_{i,j}$, then any point in $\mathrm{Lune}_d(s_i, s_j)$ belongs to $\mathrm{F}^d_S(v_{i,j,k}) \setminus \mathrm{CH}(S)$.*

Proof. Consider $\mathrm{F}^d_S(p)$ for some point p on $e_{i,j}$. By definition of a full circle it contains all sites in S, so $\mathrm{CH}(S) \subset \mathrm{F}^d_S(p)$ since C_d is convex. Therefore $A(p, s_i, s_j)$ is outside $\mathrm{CH}(S)$. On the other hand, both p and $v_{i,j,k}$ are within one half-plane h defined by the line through s_i, s_j (since $e_{i,j}$ consists of those points for which these are the farthest sites). By the Inside-Outside lemma therefore $A(p, s_i, s_j)$ (which is outside h) therefore is within $\mathrm{F}^d_S(v_{i,j,k}) \cap \bar{h}$. □

So as promised previously, all newly placed sites are outside the convex hull of preexisting sites, and so are proper. Now we are ready to prove safety.

Lemma 15 (Safety Lemma) *For any two consecutive vertices s_i, s_j on* $\mathrm{CH}(S)$*, every new site in* $\mathrm{Lune}_d(s_i, s_j)$ *is safe.*

Proof. Let s be be a new site for S that is contained in $\mathrm{Lune}_d(s_i, s_j)$. Let $e_{i,j}$ be the unbounded edge where the regions of s_i and s_j meet, and let $v_{i,j,k}$ be the vertex where $e_{i,j}$ ends. By the definition of $\mathrm{Lune}_d(s_i, s_j)$, the new site s is contained in the full d-disc $\mathrm{F}^d_S(v_{i,j,k})$ that passes through s_i and s_j. Thus, s is safe for $v_{i,j,k}$.

Consider a vertex $v_{i,k,l}$ that is connected to $v_{i,j,k}$ by the edge $e_{i,k}$. We argue that $\mathrm{Lune}_d(s_i, s_j)$—and, therefore, the new site s—is contained in $\mathrm{F}^d_S(p)$ for any point $p \in e_{i,k}$, i.e., the new site s is safe for $e_{i,k}$ and $v_{i,k,l}$.

Let h_s be the half-plane containing s that is bounded by the line through s_i and s_k. We apply Lemma 13 in two ways, depending on whether p lies in h_s or not.

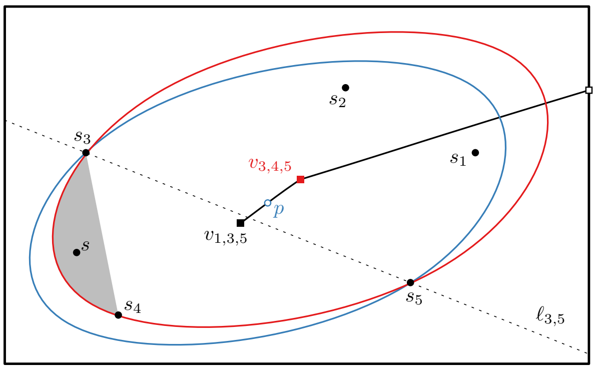


Figure 9: An example for the case $p \notin h_s$ from the proof of Lemma 15 with $i = 3$, $j = 4$, $k = 5$, and $l = 1$.

Suppose $p \notin h_s$, as illustrated in Figure 9. We approach s_i and s_k when we walk from $v_{i,j,k}$ along $e_{i,k}$ towards $v_{i,k,l}$. Therefore, $\mathrm{F}^d_S(v_{i,j,k})$ is larger than $\mathrm{F}^d_S(p)$. Since $p, v_{i,j,k} \notin h_s$, Lemma 13 implies $h_s \cap \mathrm{F}^d_S(v_{i,k,l}) \subset h_s \cap \mathrm{F}^d_S(p)$. We know $s \in \mathrm{Lune}_d(s_i, s_j) = h_s \cap \mathrm{F}^d_S(v_{i,k,l})$. Therefore, $s \in h_s \cap \mathrm{F}^d_S(p)$, and, thus, s is safe for p.

Suppose $p \in h_s$, as illustrated in Figure 10. Then there is a point w along $e_{i,k}$ that intersects $\ell_{i,k}$, since $v_{i,j,k} \notin h_s$. We move away from s_i and s_k when we walk from w along $e_{i,k}$ to $v_{i,k,l}$. Therefore, $\mathrm{F}^d_S(p)$ is larger than $\mathrm{F}^d_S(w)$. Since $p, w \in h_s$, Lemma 13 implies $h_s \cap \mathrm{F}^d_S(w) \subset h_s \cap \mathrm{F}^d_S(p)$. We know from the previous case, when $p \notin h_s$, that $s \in h_s \cap \mathrm{F}^d_S(w)$. Therefore, $s \in h_s \cap \mathrm{F}^d_S(p)$ and, thus, the new site s is safe for p.

In summary, if $s \in \mathrm{Lune}_d(s_i, s_j)$ is safe for $v_{i,j,k}$ then s is safe for all edges incident to $v_{i,j,k}$, except for the unbounded edge $e_{i,j}$. We can repeat the above argument for all neighbors of $v_{i,j,k}$ and their neighbors and so forth. In this fashion, the safety of s propagates to all vertices and all bounded edges of the d-farthest-point Voronoi diagram of S.[3] Therefore s is safe for S. □

[3]In fact, the safety of s extends to all unbounded edges other than $e_{i,j}$ in the diagram for S, as well.

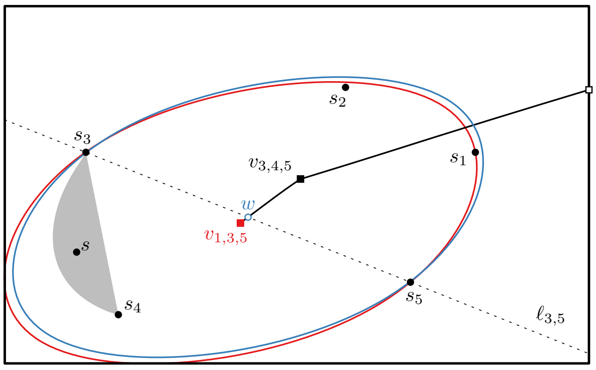


Figure 10: An example for the case $p \in h_s$ from the proof of Lemma 15 with $i = 3$, $j = 4$, $k = 5$, and $l = 1$.

B Polygonal Distance Functions: Proof of Lemma 4

Proof. Suppose s is a proper site in S. Then there is a point p such that $\mathrm{F}^d_S(p)$ has only the site s on its boundary. All other sites of S are in the interior of $\mathrm{F}^d_S(p)$ by definition of full disc. Scaling $\mathrm{F}^d_S(p)$ down while staying centered at p gives another homothet D of C_d; note that $D \subset \mathrm{F}^d_S(p)$ since d is convex. If we shrink little enough then D hence contains all of $S \setminus \{s\}$, but it does not contain s. Therefore, $\mathcal{H}(S \setminus \{s\}) \subseteq D$ does not contain s. By definition, s is an extreme point of $\mathcal{H}(S)$.

Conversely, suppose s is an extreme point of $\mathcal{H}(S)$. That means there is a homothet D of C_d that contains $S \setminus \{s\}$ and that does not contain s. Let p be the center of D. Suppose we grow D until we arrive at a d-disc D' centered at p with s on the boundary. We have $D \subset D'$, since both D and D' are convex and symmetric to p. Hence, D' is a d-disc centered at p that contains S and has only the site s on its boundary. This means s is the only d-farthest point from p, i.e. s is a proper site. □

Recognition of Triangulation Duals of Simple Polygons With and Without Holes

Martin Derka* Alejandro López-Ortiz* Daniela Maftuleac*

Abstract

We investigate the problem of determining if a given graph corresponds to the dual of a triangulation of a simple polygon. This is a graph recognition problem, where in our particular case we wish to recognize a graph which corresponds to the dual of a triangulation of a simple polygon with or without holes and interior points. We show that the difficulty of this problem depends critically on the amount of information given and we give a sharp boundary between the various tractable and intractable versions of the problem.

1 Introduction

Triangulating a polygon is a common preprocessing step for polygon exploration algorithms [10] among many other applications (see [7]). The exploration of the polygon is thus reduced to a traversal of the triangulation, which is equivalent to a vertex tour of the dual graph of the triangulation. In the study of lower bounds for such a setting, the question often arises if a given constructed graph is or is not the dual of a triangulation of an actual polygonal region (with or without holes) [10]. Thus, the recognition of a graph class is a well established problem of theoretical interest and given the importance of triangulations likely to be of use in the future. More formally, given a graph, does it represent a triangulation dual of a simple polygon? There are three aspects of this problem: the geometric problem, the topological problem and the combinatorial problem[1]. In the geometric problem, we are given a precise embedding of the graph. In the topological problem, we are given a topological embedding (also called "face embedding"). In the combinatorial problem, we are given the adjacency matrix only. Furthermore, the problem can be stated in both the decision version when the task is to recognize the graph of a triangulation, and the constructive version when the task is to realize the corresponding triangulation. For some graph classes, recognition may be easier than realization.

		Aspect		
		Geometric	**Topological**	**Combinatorial**
Polygon	**w/o holes**	a necessary condition that can be checked in $\Theta(n)$ time [Theorem 2]	$\Theta(n)$ [Theorem 6]	$\Theta(n)$ [Theorem 8]
	w holes	a necessary condition that can be checked in $\Theta(n)$ time [Theorem 12]	$\Theta(n)$ if holes are assigned to faces [Theorem 11] / NP-complete w/o hole assignment [Theorem 13]	NP-complete [Theorem 14]

Table 1: Summary of results.

Some specialized versions of this problem were studied in the past. Sugihara, and Hiroshima [14] as well as Snoeyink and van Kreveld [13] consider the problem of realization of a Delaunay triangulation for the combinatorial version of the problem. In [12], the authors define three aspects of the recognition problem of a Voronoi/Delaunay diagram, where the first two of them are what we call the geometric and topological aspects. The most relevant part of their work is the following question in the geometrical setting [12, Problem V10, p. 108]: *Given a triangulation graph, decide whether it is a (non-degenerate) Delaunay triangulation realizable graph.* For this case, the authors give necessary and sufficient conditions for a graph to be Delaunay triangulation realizable graph in the geometric setting.

In this paper, we study the problem of recognizing the dual of a triangulation of a simple polygon with or without holes and interior points in the geometric, topological and combinatorial setting. To the best of our knowledge, this paper is the first work which considers the problem for general triangulations of polygons. We draw a clear line between tractability and NP-completeness of the problem as the degrees of freedom increase from the geometric to the topological to the combinatorial problem and as we consider holes. Our results are summarized in Table 1. The recognition algorithms presented in this paper are constructive and allow realization of the polygon.

2 Preliminaries

Let P be a simple polygon with or without holes with n vertices, S a set of m interior points located inside P and $\mathcal{T}$ a *triangulation* of the $n+m$ given points inside P (for an example of a triangulation, see solid lines in

*University of Waterloo. The first author was supported by Vanier CGS. The authors would like to thank T. Biedl for valuable suggestions and discussion.

[1]In [14], Sugihara and Hiroshima call "the topological embedding problem" what we call "the combinatorial problem" here.

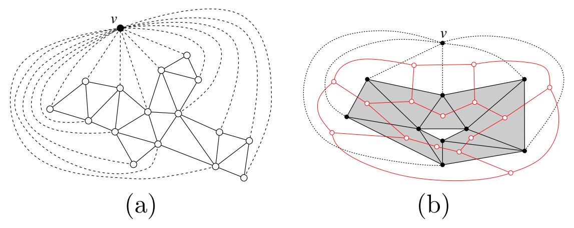


Figure 1: (a) An example of a triangulation of a polygon (solid lines) and its graph (solid and dashed lines), (b) a polygonal region P with one (white) hole (shown in solid black lines and gray interior); its triangulation $\mathcal{T}$ (in solid black lines); the graph G of the triangulation $\mathcal{T}$ (in black, solid and dashed lines); the dual graph G^* of the triangulation (in solid red lines).

Fig. 1(a)). Let G be the *graph of the triangulation* $\mathcal{T}$ as the graph on vertices $P \cup S$ plus an additional vertex v "at infinity" located outside P and the edges of G are the edges of $\mathcal{T}$ plus the edges connecting every vertex on the boundary of P to v (see Fig. 1).

This paper reconstructs triangulations of polygons from duals via reconstructing their graphs (which include the point at infinity). As we show, the point at infinity provides one with tools which are fully sufficient for such a reconstruction. If graphs of triangulations were defined without points at infinity, one would discover that there are many triangulations of a polygon with the same dual (see [3, Fig. 1]). Furthermore, we suggest that adding the point at infinity to representations of triangulations is easy to accomplish: Given a triangulation $\mathcal{T}$ of a polygon, one can construct its graph G by adding the point at infinity. In the other direction, if the vertex at infinity is known, one can easily construct triangulation $\mathcal{T}$ from its graph G. The information about which is the point at infinity can be given as a part of the input, or in some cases, this may be even implicitly determined by formulation of the problem (see Definition 1; TDR-without-holes).

Given a plane graph Γ, the *dual graph* of Γ, denoted by Γ^*, is a planar graph whose vertex set is formed by the faces of Γ (including the outer face), and two vertices in Γ^* are adjacent if and only if the corresponding faces in Γ share an edge.

Let G be a graph of a triangulation of a polygon P and G^* its dual graph. For brevity, we say that G^* is the *dual graph of the triangulation* $\mathcal{T}$ and from now on we will use this notion instead of "the dual graph of the graph of a triangulation $\mathcal{T}$."

Definition 1 (The TDR Problems) *Given a planar graph G^*, decide if G^* is a dual graph of a triangulation of a polygon P with a set of interior points S. We distinguish between: (1)* TDR-without-holes *if P is not allowed to have holes and $S = \emptyset$; (2)* TDRS-without-holes *if P is not allowed to have holes and S may be non-empty; (3)* TDR-with-known-holes *if P is allowed to have holes, $S = \emptyset$, and the positions of holes are part of the input; and (4)* TDR-with-unknown-holes *if P is allowed to have holes, $S = \emptyset$, and the positions of holes are unknown.*

The following proposition summarizes some well-known facts about planar graphs and their duals (see [3] for the proof).

Proposition 1 *1. The dual of a planar graph G is a planar graph. 2. The embedding of a 3-connected graph is unique up to the choice of the outer face. 3. The dual graph of a 3-connected planar graph is a 3-connected planar graph.*

3 Triangulation Dual Recognition (TDR)

We present a sequence of increasingly complex dual recognition problems. We draw a clear line between the tractability of the problem and the NP-completeness depending on the degrees of freedom in the particular setting being considered. We first establish some properties of the triangulation dual of a polygon that will allow us to decide if the input graph is a dual of a triangulation or not. We consider separately the cases where the triangulated polygon has holes or not, and contains interior points or not.

We consider three aspects of this problem depending on the amount of information given. In the most restricted case, we are given a *geometric* embedding of the dual of a triangulation. Each triangle of G is represented in the dual G^* by a distinguished point in its interior. In particular, following Hartvigsen [6] we consider the circumcenter of the triangle (which does not necessarily lie inside the triangle) and we are given the edge adjacencies between the triangles. In the second case we are given the faces of the dual of the triangulation but not their precise geometric embedding. This forms the *topological* recognition problem. Lastly, in the least restrictive case we are simply given a dual graph without any knowledge of which vertices form a face in the triangulation dual. This is the *combinatorial* recognition problem.

Geometric TDR- and TDRS-without-holes. For the geometric recognition problem, we do not consider the point at infinity, since it does not have a natural geometric representation. Thus, in this problem the input is a geometric embedding of the dual of the triangulation $\mathcal{T}$. In the dual, each triangle is represented by a distinguished point. The natural choices for such a point are (a) the circumcenter, (b) the incenter, (c) the orthocenter, (d) the centroid or (e) an arbitrary point in the interior of the triangle.

For the case of (a), the circumcenter, which is the choice of Hartvigsen for the recognition of Delaunay triangulations [6], we use a similar technique and create a two dimensional linear program. This is based on the observation that the edges in the triangulation are perpendicular to the dual edges in the geometric embedding. The intersections of such edges are the vertices of the polygon. Observe as well that the center of the triangulation edges lies on the corresponding dual edge. We can then set up a linear program with the coordinates of the vertices of the polygon as unknowns, and the orthogonality and bisection equations as linear constraints. We then solve the two dimensional LP program in linear time using Megiddo's fixed dimension LP algorithm [11]. If there is no feasible solution then we know that necessarily the given input graph is not the dual of a triangulation since otherwise, the actual triangulation graph satisfies the given linear constraints.

A similar approach works for the case of (c) the centroid. See [3] for details.

Theorem 2 *The linear program described above gives a necessary condition for the realization of the geometric TDR- and TDRS-without-holes problems in linear time with input G^* given the triangulation graph with the circumcenters/centroids of the triangles of G^* as vertices.*

However, it is important to observe that the feasible solution by the LP only obeys orthogonality/median constraints and has no knowledge of planarity constraints of the resulting triangulation. Thus, the proposed solution might not be a realizable triangulation. One way of resolving this problem is testing (in linear time) if the proposed solution is planar. If it is, we now have a realization of the triangulation. If on the other hand the solution is not planar, we cannot decide if there is not another realization that would have been planar. This is illustrated in [3, Fig. 3] and [3, Fig. 4], where we give different solutions to the LP constraints over the same dual triangulation graph, one leading to a feasible triangulation and the other does not.

It remains an open problem if recognition is possible under either of this models, as well as any bounds for necessary and/or sufficient conditions under other choices for triangle representatives.

To the best of our knowledge, planarity constraints between two triangles are a disjunction of three linear constraints which leads to a third degree equation which cannot be resolved using the LP program. Thus full recognition of geometric graphs remains open.

Topological TDR- and TDRS-without-holes. There are two cases of the problem in this setting: (1) the output triangulation possibly contains interior points (TDRS-without-holes) and (2) the triangulation does not contain any interior points (TDR-without-holes).

TDRS-without-holes. See [3, Lemma 1] for the proof of the following lemma:

Lemma 3 *Let P be a polygon without holes, S be a set of points in the interior of P, and $\mathcal{T}$ a triangulation of $P \cup S$. The graph G of $\mathcal{T}$ is a 3-connected maximal planar graph.*

We establish necessary (Lemma 4) and sufficient (Lemma 5) conditions for a graph to be a dual graph of a triangulation of a polygon with no holes.

Lemma 4 *If G^* is a dual graph of a triangulation of a polygon P and set S of interior points inside P with no holes, then G^* is a planar 3-regular 3-connected graph.*

Proof. The fact that G^* is planar and 3-connected follows from Lemma 3 and Proposition 1. As the graph G of the triangulation is a maximal planar graph, every face of G is a triangle. Hence, every vertex in G^* has precisely three incident edges. □

Lemma 5 *If G^* is planar, 3-regular and 3-connected, then G^* is a dual graph of a triangulation of a polygon without holes P and a set S of interior points.*

Proof. By Proposition 1(3), G^* has a dual graph G which is 3-connected, and thus G can be uniquely embedded in the plane up to the selection of the outer face (Proposition 1(2)). Such an embedding can be achieved using straight lines only (see e.g. [2]). Now, remove the vertex v of G which represents the outer face of G^*. As G is 3-connected, by removing v, we obtain a 2-connected plane graph G'. Hence, every face of G' is a simple cycle, and it is a triangulation of a polygon formed by its outer face. Moreover, since the graph G^* is 2-connected and every face is a triangle, it is the triangulation of a polygon. □

Theorem 6 *The answer to the topological TDRS-without-holes problem is "yes" if and only if the input G^* is a 3-connected 3-regular planar graph. Furthermore, such a polygon can be constructed in linear time.*

Proof. The first part of the claim follows directly from Lemmas 4 and 5. The linear running time follows from linearity of verifying 3-connectivity of a graph [8]. The reconstruction is linear as the number of faces in any planar graph is linear due to Euler's formula, so the dual graph can be constructed in linear time. The straight line embedding can be found in linear time as well [2], and deleting a vertex from an embedded graph takes at most $\mathcal{O}(n)$ steps too. □

TDR-without-holes. Previously, we showed that topological TDRS-without-holes problem can be solved in linear time. This can be done even if the set of interior points S is required to be empty (i.e., the graph of the triangulation should consist only of vertices at the boundary of the polygon P, and one vertex outside P). The following is proved in [3, Prop. 2]:

Proposition 7 *Let G^* be a 3-regular planar graph and $\widetilde{G^*}$ the subgraph of G^* obtained by removing the vertices of the outer face. $\widetilde{G^*}$ is a tree if and only if it corresponds to a polygon with no holes or interior points.*

Combinatorial TDR- and TDRS-without-holes. It is easy to see that the topological and combinatorial input are equivalent in this case. Since G^* must be 3-connected and 3-regular, the algorithm can first verify this necessary condition. If it is satisfied, it can construct the embedding (e.g., applying the linear straight line embedding algorithm of [2]) and proceed with the topological input.

Theorem 8 *The answer to combinatorial TDR- and TDRS-without-holes problems is affirmative if and only if the input G^* is a 3-connected 3-regular planar graph. Furthermore, such a polygon can be constructed in linear time.*

Topological TDR- and TDRS-with-known-holes. Let us start with the following observation:

Proposition 9 *If a polygon has a hole, the dual graph G^* of its triangulation contains vertices of degree 2 or less.*

From the proof of Proposition 9 (see [3, Prop. 3]), we can see that vertices of degree 2 in G^* are adjacent to holes in the initial polygon. Observe that if P is a polygon with or without holes, the triangles of the graph G of a triangulation created by the point at infinity and the outer face of the polygon form a 3-connected graph. Thus, the dual graph G^* cannot contain a 2-cut on the outer face corresponding to these triangles.

We can associate each degree 2 vertex to its adjacent hole. Formally, let G^* be a planar graph that contains at least one vertex of degree 2. We define an *assignment* of a vertex u of degree 2 to a face of G^*, as a mapping $\mathcal{H}$ from the set containing u to the set of faces incident to u of G^*, such that if u is incident to faces F and F' in G^*, then $\mathcal{H}(u) \in \{F, F'\}$. The same way we can define: an *empty assignment*, which does not assign any vertex of degree 2 to a face of G^*; a *partial assignment*, which assigns a subset of vertices of degree 2 to their incident faces in G^* and a *total assignment* which assigns all the vertices of degree 2 to faces of G^* (see [3, Fig. 11] for a total assignment example).

Lemma 10 *Let $\{G^*, \mathcal{H}\}$ be such that G^* is a dual graph of a triangulation of a polygon with holes. A face that is assigned vertices of degree 2 contains a hole in the initial polygon. Moreover, $\mathcal{H}$ assigns to each face of G^* zero or at least three vertices of degree 2.*

Proof. The claim follows from the proof of Proposition 9; the proof is provided in [3, Lemma 4]. □

We know that the presence of a vertex of degree 2 in the graph G^* means there is a hole in the output polygon in one of the faces incident to this vertex in G^*. The reason to define an assignment of vertices of degree 2 to faces of the graph is to establish in which of the two incident faces the hole is contained (see [3, Fig. 5]). We call $\mathcal{H}$ a *valid assignment* if we can realize G^* as a triangulation dual of a polygon with holes. A polygon P with holes is a realization of $\{G^*, \mathcal{H}\}$ if the polygon is a realization of G^* and $\mathcal{H}$ is a valid assignment with respect to P.

Let us now focus on the one-edge cuts in G^*, such as the one shown in Fig. 2(a). These one-edge cuts in the dual represent edges in the original polygon where a straight line cut applied to that edge would separate the polygon into two disjoint subpolygons. The two possible cases of how this separation looks like are illustrated in Fig. 2(b) and (c).

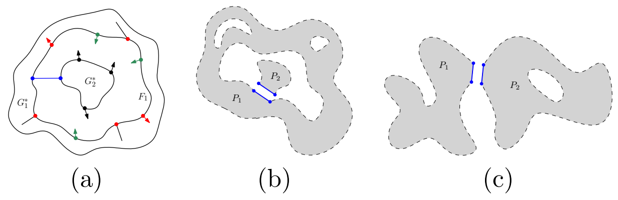


Figure 2: (a) Dual graph with an one-edge cut (shown in blue) and its two connected components G_1^* and G_2^*, (b) The realization of G_1^* and G_2^* from (a) as P_1 and P_2, (c) Another possible realization of two components of an one-edge cut.

Now we observe that when the first such cut is applied, the case shown in Fig. 2(c) is not possible since the outer face is 3-connected due to the point at infinity and the upper and lower chain of the original polygon. So in what follows, we need only consider case (b) in the figure. Let G_1^* and G_2^* denote the subgraph duals of P_1 and P_2, respectively in G^*. The algorithm now recursively creates a topological embedding for P_1 and P_2 and merges the two embeddings. We show in [3] that this process is deterministic and results in a unique topological graph which can be embedded using straight line edges. This resulting polygonal graph is a simple polygon with point at infinity if and only if G^* is a triangulation dual of a simple polygon. Hence, we have the following theorem (proof in [3, Thm. 4]).

Theorem 11 *Given an input $\{G^*, \mathcal{H}\}$, the topological TDR- and TDRS-with-known-holes problems are decidable in linear time.*

Geometric TDR- and TDRS-with-known-holes. Given a precise geometric embedding of the input graph, we want to decide if the graph is the triangulation dual of a polygon with holes with or without interior points.

Theorem 12 *The linear program described in Section 3 gives a necessary condition for the realization of the geometric TDR- and TDRS-with-known-holes problems with input* $\{G^*, \mathcal{H}\}$ *in linear time given the triangulation graph with the circumcenters/centroids of the triangles as vertices.*

Proof. Note that if a vertex v is of degree 2 in G^*, deciding which face incident to v contains the associated hole can be done by observing the location of the convex angle formed by the two edges of the triangulation perpendicular to the edges incident to v in G^*. (For an illustration, see [3, Fig. 5(a)] in which the hole can only reside in the right face.) We then set up an LP as in Theorem 2 which gives a potential realization of the triangulation. We then test this solution to verify that the polygon and holes obtained are simple. □

Topological TDR- and TDRS-with-unknown-holes. The input for this version of the problem is a planar graph G^* with its face-embedding. However, the total assignment of its vertices of degree 2 to faces of G^* is unknown. Here, we only state that the problem is NP-complete (Theorem 13) and proceed in our analysis. The proof of this claim is provided in Section 4.

Theorem 13 *Determining if an input graph* G^* *is the dual of a triangulation of a polygon with holes and with or without interior points is NP-complete.*

Combinatorial TDR- and TDRS-with-unknown-holes. In this subsection, the input graph G^* is given by its adjacency matrix. We will show that the 3-SAT reduction from the topological TDR- and TDRS-with-unknown-holes problems (see Section 4) holds as well. If the embedding found by the combinatorial TDR solver is the same as in the reduction, we would need to solve the 3-SAT problem. However, it remains to be shown that there does not exist a different embedding with an alternate polygonal realization and the answer being "yes", without this embedding necessarily implying satisfiability of the 3-SAT formula.

Recall that a 3-regular graph has a unique embedding in the plane. We now remove the vertices of degree 2 from the 3-SAT reduction graph and replace them by an edge, thus giving a 3-regular graph with a unique embedding. If the combinatorial TDR- and TDRS-with-unknown-holes problems found a different embedding, we can replace the vertices of degree 2 in this alternate embedding with a single edge, thus obtaining a different embedding for the 3-regular graph, which is a contradiction. Hence, the combinatorial graph obtained from the reduction above has a polygonal realization if and only if the underlying formula is satisfiable and we obtain:

Theorem 14 *The combinatorial TDR- and TDRS-with-unknown-holes problems are NP-complete.*

4 NP-Completeness of topological TDR- and TDRS-with-unknown-holes problems

In this section, we prove Theorem 13. Let $X = (x_1, x_2, \ldots, x_m)$ be a set of boolean variables. Let φ be a 3-SAT boolean formula of the type $\varphi = (a_{11} \vee a_{12} \vee a_{13}) \wedge (a_{21} \vee a_{22} \vee a_{23}) \wedge \ldots \wedge (a_{n1} \vee a_{n2} \vee a_{n3})$, where a_{ij} is either x_k or $\neg x_k$ (called a *literal*). We restrict our attention to planar 3-SAT formulae. A planar 3-SAT formula, by definition, can be represented by a planar graph which has a vertex for every clause and every variable, and has an edge connecting said variable to every clause in which it appears (negated or non-negated).

Planar 3-SAT is known to be NP-complete [9]. We will reduce planar 3-SAT to dual triangulation recognition by constructing a graph G^* that is the dual of a triangulation of a polygon with holes if and only if given formula φ is satisfiable. Our reduction creates G^* which consists of four types of gadgets (Fig. 3(a)–(d) resp.):

1. variable faces which correspond to variable vertices;
2. clause gadgets which correspond to clause vertices;
3. splitter faces which correspond to some edges connecting a variable vertex to a clause; and
4. absorber gadgets which act as dead ends for extra splitter wires which are not needed.

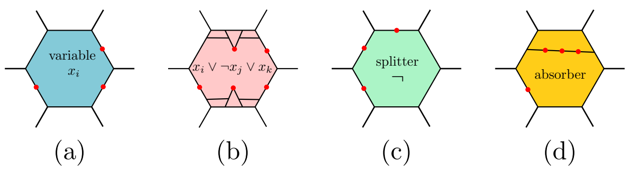


Figure 3: The types of faces and gadgets of G^*: (a) a variable face, (b) a clause gadget, (c) a splitter face and (d) an absorber gadget.

See [3] for the detailed description of the gadgets and their roles.

We construct a graph such that if the variable x_i corresponding to the variable face F_{x_i} is false in a satisfiable assignment of φ, the degree 2 vertices are assigned to F_{x_i} (the red arrows in our figures point inwards), and if the variable is true in the assignment, then all the degree 2 vertices are assigned to the other face. The construction begins by constructing the planar graph G_φ, which represents φ, and embedding it in the plane.

Later, we will replace its vertices by corresponding gadgets. However, for this to be possible, the graph needs to be modified first.

Each edge in G_φ indicates a "transfer" of a degree 2 vertex. We first need to modify the graph so that the vertices representing variables of φ have degree precisely 3. If the degree of such a vertex x_i is less than 3, we increase it by attaching the required number of new vertices (those will be replaced by absorber gadgets). If the degree of x_i is more than 3, we reduce its degree by detaching $deg(x_i) - 2$ edges consecutive in cyclic order around x_i (with respect to the embedding of G_φ), routing them into a new splitter vertex s, and connecting x_i to the splitter. Note that this negates the variable x_i, so some of the edges may need to be routed through another splitter to cancel this negation. This produces a plane graph where x_i has degree 3 and the splitter vertex s has degree $deg(x_i) - 1$. Repeatedly applying this construction, the degree of s can be decreased to 3.

By the construction above, we obtain a plane graph H_φ where all the variable, splitter and clause vertices have degree 3, and absorbers have degree 1. Now we replace every vertex with the respective gadget so that every edge in H_φ is represented by a degree 2 vertex surrounded by edges shared between two gadgets, and so that the topology of the gadgets is equivalent to the embedding of H_φ (this is similar to constructing a dual graph of H_φ). Let us denote the obtained graph by H^*. The embedding of H^* contains some "void" areas between some gadgets. Those areas can be suitably attributed to faces of gadgets by removing edges. We obtain graph G^*, call it the *gadget graph of* φ, formed by vertices of degree 3 and 2. See [3, Fig. 13(b)] for an example of a formula with its gadget graph.

We can now argue that graph G^* is a triangulation dual if and only if the formula φ is satisfiable. See [3, Lemma 5] for the proof.

Lemma 15 *The gadget graph G^* of formula φ is dual of a triangulation of a simple polygon with holes if and only if φ is satisfiable.*

5 Conclusions and Open Questions

We provided an exhaustive analysis of the triangulation dual recognition problem. We showed that some of them can be solved in linear time and some of them are NP-complete. Our work focused on duals of general triangulations of simple polygons. We proposed several models for the geometric setting. We presented a method which in linear time finds a candidate solution, or rejects. The candidate solution needs to be further tested. As our approach is not capable of enumerating all the candidate solutions, it remains an open problem if recognition is possible under either of these models. Any bounds for necessary and/or sufficient conditions under other choices for triangle representatives are open.

References

[1] M. De Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 3rd rev. ed. (2008).

[2] H. De Fraysseix, J. Pach, R. Pollack, *How to draw a planar graph on a grid*, Combinatorica vol. 10(1):41–51 (1990).

[3] M. Derka, A. López-Ortiz and D. Maftuleac, *Recognition of Triangulation Duals of Simple Polygons With and Without Holes*, arXiv:1607.05739.

[4] R. Diestel, *Graph Theory, 4th Edition*, Grad. texts in math. 173, Springer (2012).

[5] B. Grunbaum, *Convex Polytopes, 2nd Edition*, Springer (2004).

[6] D. Hartvigsen, *Recognizing Voronoi diagrams with linear programming*, ORSA Journal on Computing, 4(4), 369-374 (1992).

[7] Ø. Hjelle and M. Dæhlen, *Triangulations and Applications (Mathematics and Visualization)*, Springer-Verlag New York (2006).

[8] J.E. Hopcroft, R.E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput. 2(3):135–158 (1973).

[9] D. Lichtenstein, *Planar Formulae and Their Uses*, SIAM J. Comput. 11(2), 329-343 (1982).

[10] D. Maftuleac, S. Lee, S.P. Fekete, A.K. Akash, A. López-Ortiz, J. McLurkin *Local Policies for Efficiently Patrolling a Triangulated Region by a Robot Swarm*, International Conference on Robotics and Automation (ICRA), 2015.

[11] N. Megiddo, *Linear Programming in Linear Time When the Dimension Is Fixed*, Journal of the Association for Computing Machinery, 31 (1984), No. 1, 114-127.

[12] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*, ISBN: 978-0-471-98635-5, 2000.

[13] J. Snoeyink, M. van Kreveld, *Linear-Time Reconstruction of Delaunay Triangulations with Applications*, European Symposium on Algorithms (1997).

[14] K. Sugihara, T. Hiroshima, *How to Draw a Delaunay Triangulation with a Given Topology*, Abstracts 13th European Workshop Comput. Geom. (1997), 1315.

Sliding k-Transmitters: Hardness and Approximation

Therese Biedl* Saeed Mehrabi* Ziting Yu*

Abstract

A *sliding k-transmitter* in an orthogonal polygon P is a mobile guard that travels back and forth along an orthogonal line segment s inside P. It can see a point $p \in P$ if the perpendicular from p onto s intersects the boundary of P at most k times. We show that guarding an orthogonal polygon P with the minimum number of k-transmitters is NP-hard, for any fixed $k > 0$, even if P is simple and monotone. Moreover, we give an $O(1)$-approximation algorithm for this problem.

1 Introduction

Art gallery problems are one of the standard problems in computational geometry. In the original setting, we are given a polygon (modelling the art gallery) and we want to know a set of points (modelling guards or cameras) that can see any point in the polygon, where "see" in the original setting means that the line segment from the guard to the point is inside the polygon. There have been numerous result, concerning bounds on the number of guards needed, NP-hardness and approximation algorithms. See e.g. [13, 10] and the references therein.

Recently, motivated by covering a region with wireless transmitters, Aichholzer et al. [1] introduced variants where guards can see through a limited number of walls. Hence a *k-transmitter* is a point p in a polygon P that is considered to see all points q in P for which the line segment $\overline{pq}$ intersects the boundary of P at most k times. Only cases of even k are interesting.

We combine in this paper the concept of a k-transmitter with the concept of a mobile guard. A *mobile guard* is a guard that is not stationary, but walks along a line segment s inside the polygon, and can see all points that are visible from some point of s. For orthogonal polygons, a common restriction has been to demand that line segment s is horizontal or vertical, and that it guards only those points p that it can see in an orthogonal fashion, i.e., the perpendicular from p onto s is inside P. This is called a *sliding camera.* We combine the concept of sliding cameras with k-transmitters, and hence define a *sliding k-transmitter* as follows: It is a horizontal or vertical line segment s inside an orthogonal polygon P and it can see all points p such that the perpendicular from s onto p intersects the boundary of P at most k times. We allow sliding k-transmitters to include edges of the polygon.[1] The objective is to guard P with the minimum number of sliding k-transmitters.

Related Work. Sliding cameras were introduced by Katz and Morgenstern [9]. Finding the minimum set of sliding cameras is NP-hard in polygons with holes [8], even if only horizontal sliding cameras are allowed [4]. The optimum set of sliding cameras can be found in polynomial time for monotone polygons [7]. The complexity for simple polygons is open.

Finding the minimum set of k-transmitters is NP-hard in simple polygons [6], regardless whether the transmitters are points or polygon-edges. Numerous bounds are known on the number of k-transmitters that are necessary and sufficient, depending on the type of transmitter (point or edge) and the type of polygon [1, 2, 3, 6]. Regarding sliding k-transmitters, an approximation algorithm for monotone polygons is claimed in [12], but the algorithm needs a minor modification to deal with an example (private communication); it is not clear whether this modification suffices. Other optimization criteria for sliding k-transmitters have also been considered [11].

Our Results. In this paper, we study the complexity of finding the minimum set of sliding k-transmitters to guard an orthogonal polygon. Unsurprisingly, we can show that this is NP-hard, but we prove NP-hardness even in a very restricted version: The polygon is orthogonal and y-monotone, and there is an optimal solution with only horizontal sliding k-transmitters. We are not aware of *any* other variant of the art gallery problem that is NP-hard on orthogonal monotone polygons (the traditional art gallery problem is NP-hard for monotone polygons [10], but slanted edges are crucial for the reduction to work).

As a second result, we show that the $O(1)$-approximation algorithm that we recently developed for sliding cameras [4] works similarly for sliding k-transmitters. Hence we have an $O(1)$-approximation for finding the minimum set of sliding k-transmitters, in any (not necessarily simple) orthogonal polygon. The algorithm works also (and becomes even easier) if only horizontal sliding k-transmitters are allowed.

*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. {biedl,smehrabi, z44yu}@uwaterloo.ca

[1]With some minor modifications, the results in this paper also hold if guards must be strictly inside P except at their end.

2 NP-Hardness

In this section, we show that guarding with sliding k-transmitters is NP-hard, even if the polygon is orthogonal and monotone (hence simple). We first prove this for $k = 2$ and then extend to larger k.

2.1 Sliding 2-Transmitters

We use a reduction from Minimum Vertex Cover in a graph G, which is known to be NP-hard even if G is required to be planar and 2-connected (see e.g. [5]). So the objective is to compute a minimum set C of vertices such that every edge has at least one endpoint in C.

Given a planar 2-connected graph G with n vertices and m edges, we first compute a *bar visibility representation* of G in which each vertex is assigned a horizontal line segment (called *bar*) and for each edge there is a vertical *strip* with positive width that connects the bars of endpoints and does not intersect other vertices. It has been shown multiple times (see e.g. [14]) that this exists and can be computed in linear time. We may move vertex-bars up and down slightly as needed so that all vertex-bars have distinct y-coordinates. Also, since edge-strips have positive width, we can make them thin enough such that no two of them have overlapping x-range. Since the graph is 2-connected, the construction in [14] guarantees that all vertices except the bottom-most one have a neighbour below, and all vertices except the topmost one have a neighbour above.

Gadgets. We start by thickening each vertex-bar into a box, and place three copies of this box above each other with the same x-range. These three boxes are connected to each other by *channels*, which are thin vertical corridors (thin enough so that their x-range is strictly within that of the vertex-box, and does not intersect an edge-strip). We place these two channels at opposite ends of the vertex-boxes, resulting in a Z-shape or an S-shape (the choice between the two is arbitrary for now, but will be determined later). We call the result a *vertex-gadget*; see Fig. 1. By making the height of boxes small enough, we may assume that no two vertex-gadgets have overlapping y-range.

For each edge e, the *edge-gadget* of e is a small axis-aligned box placed strictly within the strip representing e in such a way that its y-range intersects no y-range of another (vertex- or edge-) gadget. See Fig. 1. Notice that from any edge-gadget there are vertical lines-of-sight to the vertex-gadgets of the endpoints of the edge.

The Reduction. Let P' be the polygon obtained by replacing all vertex-bars and edge-strips with these gadgets. P' is y-monotone (i.e., any horizontal line intersects it in one interval), but not connected (for now we allow the polygon to be disconnected, but we will discuss the modifications to make it connected later).

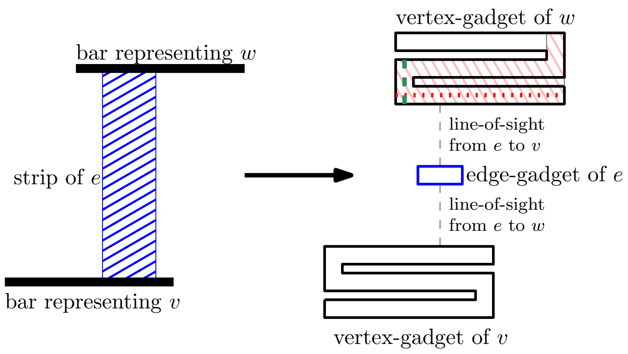


Figure 1: Vertex- and edge-gadgets. The pink (falling pattern) region is guarded by the red (dotted) horizontal 2-transmitter. Note that it includes everything that the green (dashed) vertical 2-transmitter can see.

Since no y-ranges overlap, one can easily verify that vertical 2-transmitters are never required.

Observation 1 *Any vertical sliding 2-transmitter in P' can be replaced by a horizontal sliding 2-transmitter that guards at least as much.*

See also Fig. 1. We call the three boxes of a vertex-gadget the *top, middle* and *bottom* box, and also use *outer boxes* to mean the top and bottom box.

Lemma 1 *For any set S of horizontal sliding 2-transmitters that guard P' entirely, there exists a set S' of horizontal sliding 2-transmitters that guard P' entirely such that $|S'| \le |S|$ and no sliding 2-transmitter of S' is located in an edge-gadget.*

Proof. Let $s \in S$ be a sliding 2-transmitter that lies in an edge-gadget B corresponding to edge $e = (v, w)$. After possible renaming, assume that (the vertex-gadget corresponding to) v is below e and w is above e.

Assume first that one of v, w (say v) has a horizontal sliding 2-transmitter s' in the outer box facing e. After possibly extending s' we may assume that it spans the entire outer box of v. Since the x-range of B is within the x-range of v, s' sees everything that s saw and that was below s. So we can replace s by a sliding 2-transmitter in the outer box of w facing e, and this can only increase the guarded region.

So now assume that neither v nor w has a horizontal sliding 2-transmitter in the outer box facing e. Consider a point p in the top box of v that is just outside the x-range of B, but still within the x-range of w. The only horizontal sliding 2-transmitters that could guard p are in the bottom box of w or in the middle box of v. By assumption we therefore have a sliding 2-transmitter in the middle box of v. Likewise w must have a sliding 2-transmitter in the middle box of w. We can thus move the sliding 2-transmitter in B to the bottom box of w without decreasing the guarded region. □

Lemma 2 *Let S be a set of horizontal sliding 2-transmitters that guard P' entirely and that do not lie in edge-gadgets. Then for any vertex v, there must be at least one sliding 2-transmitter intersecting the vertex-gadget of v. If there is exactly one such sliding 2-transmitter, then it must be in the middle box of v.*

Proof. Pick a point p in the middle box of v that is not in the x-range of the channels. Let s be a horizontal sliding 2-transmitter that guards p. Then s must be in one of the three boxes of v.

Assume now that exactly one sliding 2-transmitter intersects the vertex-gadget of v, and it is not in the middle box. Say the sliding 2-transmitter is in the bottom box. If v has any neighbour w above, then let p be a point in the top box of v and in the same x-range as the edge-gadget of (v, w). To guard p, we need either a sliding 2-transmitter in the edge-gadget (which was excluded) or in the top or middle box of v (which was also excluded). So v cannot have any neighbour above. By construction that means that v is the topmost of all vertices. To guard the top box of v, we then must have a sliding 2-transmitter in the top or middle box of v. Again contradiction. □

Lemma 3 *The following statements are equivalent: (i) G has a vertex cover of size k, (ii) P' can be guarded by $n + k$ sliding 2-transmitters, and (iii) P' can be guarded by $n + k$ horizontal sliding 2-transmitters.*

Proof. Given a vertex cover C of G, we place horizontal transmitters as follows: If $v \in C$, then place a maximal horizontal sliding 2-transmitter in both outer boxes of v, else place a maximal horizontal sliding 2-transmitter in the middle box of v. Clearly we have $n + |C|$ sliding 2-transmitters and every vertex-gadget is guarded. For every edge e, one endpoint v is in C, and hence both bottom and top box of v contain sliding 2-transmitters. The one in the outer box of v that faces e then guards the edge-gadget of e.

Vice versa, assume that set S of sliding 2-transmitters guards P'. By the above results, we may assume that they are all horizontal and none are in an edge-gadget. Define C to be all those vertices whose vertex-gadgets are intersected by at least two sliding 2-transmitters. Since every vertex-gadget intersects at least one sliding 2-transmitter we have $|C| \leq |S| - n$. For every edge (v, w), the edge-gadget must be guarded by a sliding 2-transmitter that is in an outer box of v or w, say v. Then v must contain at least two sliding 2-transmitters by Lemma 2, so $v \in C$. Hence C is a vertex cover. □

Connecting the Polygon. Now we explain how to make the polygon connected while staying monotone. Let $g_1, \ldots, g_{m+n}$ be the gadgets in P', sorted in bottom-to-top order (since y-ranges are disjoint, this is well-defined). The idea is to connect each g_i to g_{i+1} using

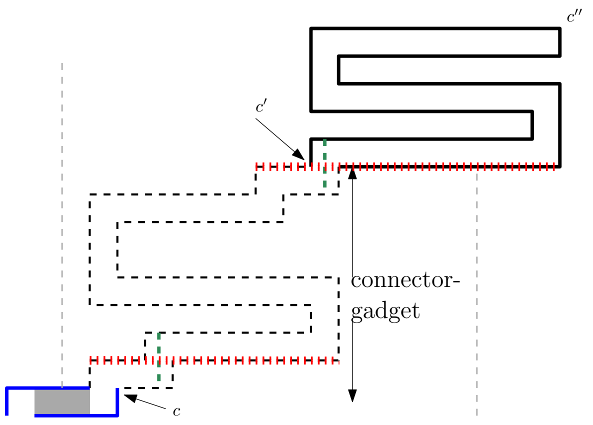


Figure 2: Connecting an edge-gadget to a vertex-gadget if there is no line of sight between them. We again show how some vertical transmitters can be replaced by horizontal transmitters.

a *connector-gadget*. This is an S-shaped or Z-shaped gadget much like a vertex-gadget, except that the top and bottom box both add a zig-zag near the end. Also, one of the channels has flexible height, so that the connector-gadget can have arbitrary height. We attach the ends of the connector-gadget C to corners of g_i and g_{i+1}. Fig. 2 shows how to do this if the x-range of C is disjoint (except at the ends) from the ones of g_i and g_{i+1}, and the inset in Fig. 3 shows how to do this if C shares x-range with them (in case of which we push the zig-zag to the very end to avoid overlap.)

However, we cannot connect consecutive gadgets if the connector-gadget would cross a line-of-sight. To avoid doing this, we will subdivide edges.

Observation 2 (Folklore) *If G^s results from graph G by subdividing one edge twice, then G has a vertex cover of size k if and only if G^s has a vertex cover of size $k+1$.*

We proceed as follows. First "parse" the bottommost gadget g_1: use an S-shape for it and fix as current corner its top right corner. Assume now we have parsed gadget g_i already, and fixed one top corner c of it as current corner. Let g_{i+1} be the next gadget above g_i. Considering its two bottom corners, we choose the corner c' so that $\overline{cc'}$ crosses as few lines-of-sight as possible.

If line segment $\overline{cc'}$ crosses no line-of-sight, then attach a connector-gadget between c and c', using as shape (i.e., S or Z) the one that has c and c' at its ends. Let c'' be the diagonally opposite corner from c' in gadget g_{i+1} and (if g_{i+1} is a vertex-gadget) use as shape (i.e., S or Z) for it the one that has c' and c'' at its ends. This finishes (in this case) the parsing of gadget g_{i+1}, and we continue to connect to the next gadget with current corner c''.

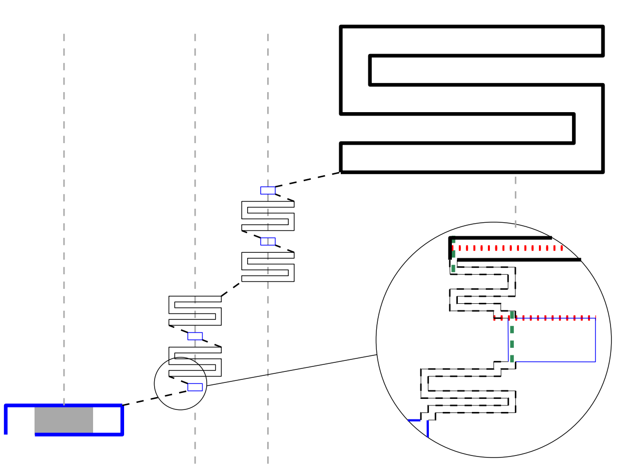


Figure 3: Connecting an edge-gadget to a vertex-gadget if there are lines of sight between them.

Now assume that $\overline{cc'}$ crosses some lines-of-sight, say $l_1, \ldots, l_\ell$ in order from c to c'. For all j, line-of-sight l_j represents an edge e_j; subdivide e_j twice. This adds two new vertex-gadgets and two new edge-gadgets that we place along l_j, in the y-range between g_i and g_{i+1}. We make their height small enough and move them up and down suitably (while staying between g_i and g_{i+1}), so that all their y-ranges are disjoint and the ones of l_j are below the ones of l_{j+1} for all j.

All these gadgets can be connected with line segments that do not cross a line-of-sight. See Fig. 3. We can hence connect all these gadgets as explained above. The only difference is that the next current corner c'' must be chosen to be the end of the line segment connecting to the next gadget. Normally c'' will again be diagonally opposite from the previous corner c', but there is one exception per set of gadgets added for subdivisions. With that we have connected to g_{i+1}, and we repeat from there (after choosing its shape and the current corner as before).

Reduction Revisited. With the addition of connector-gadgets, Observation 1 (vertical transmitters can be replaced by horizontals) is not as obvious anymore, but still holds as long as sliding k-transmitters may run along polygon-edges. See Fig. 2. With this, Lemma 1, Lemma 2, and the equivalent of Lemma 2 for connector-gadgets, also hold. Let N_s be the total number of subdivisions that we did over all connecting of all gadgets (N_s is even), and let G' be the graph that results. We started with $n+m \in O(n)$ vertex-gadgets and edge-gadgets and $2m \in O(n)$ lines-of-sight. Connecting two of these gadgets hence creates $O(n)$ subdivisions, and therefore $N_s \in O(n^2)$ is polynomial. After all subdivisions we have $n + m + 2N_s$ gadgets, and hence use $N_c := n + m + 2N_s - 1$ connector-gadgets to connect all of them into one polygon P. So the construction is

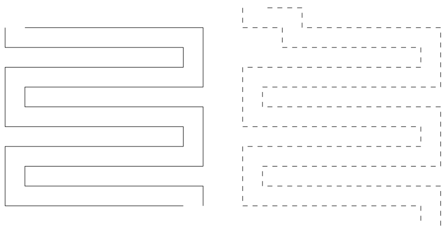


Figure 4: Vertex- and connector-gadget for $k = 4$.

polynomial. G has a vertex cover of size k if and only if G' has a vertex cover of size $k' := k + N_s/2$ if and only if P can be guarded with $k' + n + N_c$ horizontal sliding 2-transmitters.

With that, the reduction is complete for $k = 2$. Note that the constructed polygon is connected and y-monotone (and in particular therefore simple).

2.2 k-Transmitters for $k > 2$

We now generalize to sliding k-transmitters for any fixed $k > 0$. The reduction is exactly the same as before, with the exception of the definition of vertex-gadgets and connector-gadgets.

The vertex-gadget now consists of $k + 1$ copies of the thickened bar in the visibility representation (earlier we had $3 = 2 + 1$ copies). They are connected with k channels at alternate ends, resulting in a zig-zag line. The connector-gadget is a vertex-gadget with additional small zig-zags in the top and bottom box (possibly pushed towards the end.) See Fig. 4.

We can verify that again vertical sliding k-transmitters are never better than horizontal ones. Define for a vertex-gadget the *middle* box to be the $(k/2+1)$st box (recall that k is even), and the *outer boxes* to be the top and bottom box as before. With that, the proofs of Lemmas 1 and 2 carry almost verbatim, and the reduction holds again. We conclude:

Theorem 4 *For any $k > 0$, guarding a polygon with the minimum set of sliding k-transmitters is* NP-*complete, even if (i) the polygon is a simple y-monotone orthogonal polygon, and (ii) only horizontal sliding k-transmitters are allowed.*

Notice that every gadget is a *thickened path* obtained by sliding a unit square along an orthogonal path. With suitable rescaling, in fact the entire polygon can be made into a thickened path, with one exception: Whenever we subdivide edges, we must (at one edge-gadget) attach both connecting gadgets on the same (left or right) side, hence have a “leg” sticking out. (This could perhaps be called a *thickened caterpillar.*) We suspect that the construction could be modified to become a thickened path, but have not been able to work out the details yet.

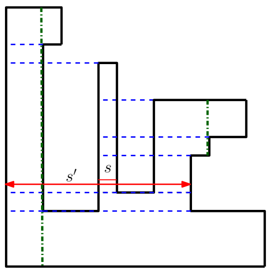


Figure 5: Horizontal partition-segments (blue dashed), one horizontal slice-segment (red solid) and two vertical guard-segments (green dot-dashed) for $k = 2$.

3 An $O(1)$-Approximation Algorithm

In this section, we give an $O(1)$-approximation algorithm for the sliding k-transmitter problem, using as key ingredient an $O(1)$-approximation developed in [4] for a certain hitting problem among segments. The main difference between our approach and the one in [4] is that we need to define the segments differently so that we encapture that guards can see through k walls.

Let P denote an orthogonal polygon with n vertices, and let δP denote the boundary of P. We first compute a subdivision of P and then define sets of orthogonal line segments in P which will be used to define the hitting set problem.

Slices. Define *horizontal partition-segments* as follows: Start with a horizontal edge e. Expand e leftwards until we hit a vertical edge of the polygon, coming from the strict inside of P, for the $(k/2)$th time. Likewise expand e rightwards. If there are not enough such intersections, then stop at the last one. See Fig. 5.

The horizontal partition-segments split the interior of the polygon into rectangles that we call *horizontal slices*. Since any edge gives rise to one partition-segment, and any partition-segment intersects $O(k)$ vertical edges, we have $O(kn)$ horizontal slices.

The following lemma argues that this partitioning is "correct" in the sense that any transmitter either guards all or nothing of the interior of a slice.

Lemma 5 *Let σ be a horizontal slice and let c be a point in its interior. If a maximal vertical sliding k-transmitter g sees c, then it sees all points of σ.*

Proof. Let q be the point on g where the perpendicular from c onto g ends. By assumption the horizontal line segment $\overline{cq}$ intersects the boundary of P at most k times. Expand $\overline{cq}$ until it spans the x-range of σ; this cannot add crossings since $\sigma \subseteq P$. Now sweep the resulting segment s' upward until we hit either the top side of σ or a horizontal edge of P. Say we hit an edge e first. We created a partition-segment from e, which extends in both directions until it hits at most $(k/2)$ vertical edges from the inside, hence at most k vertical edges. This partition segment contains the entire (translated) s' and splits B, so we have reached the top side of B.

So we can sweep the region of B above s' without encountering new edges of P, which shows that g guards all of this. Likewise we can sweep downward until the bottom side of B, and so g guards all of B. $\square$

Slice-segments. We now assign a segment to each horizontal slice that captures "being guarded". For any horizontal slice σ, let s be a horizontal segment strictly inside σ. Extend s (much like we did for partition-segments) to both sides until it hits a vertical edge from the inside for the $(k/2)$th time. We call the resulting segment s' the *slice-segment* of σ. See also Fig. 5.

We define vertical slices of P and vertical slice-segments in an analogous fashion. There are $O(kn)$ slice-segments since there are $O(kn)$ slices.

Guard-segments Γ. Our definition of sliding k-transmitters allowed any horizontal or vertical segment to be used as such. We now describe a finite set of sliding k-transmitters and argue that these suffice. Let s be a horizontal edge of P. Define a sliding k-transmitter s' obtained by extending s until we hit an interior point of a vertical edge of δP. (If some vertices are aligned, then s' may run along multiple horizontal edges of P.) The resulting segments are the *horizontal guard-segments* Γ_H. Define *vertical guard-segments* Γ_V similarly, and set $\Gamma = \Gamma_H \cup \Gamma_V$ to be the guard-segments. We have at most n guard segments (one per edge).

Crosses X: Let a *pixel* be any rectangle that has the form $\sigma_H \cap \sigma_V$ for a horizontal slice σ_H and vertical slice σ_V. Let c be the point where the slice-segments s_H, s_V corresponding to σ_H and σ_V intersect; we call c a *cross*, and say that s_H and s_V *support* c. Note that c is in the interior of the pixel since slice-segments were defined using segments strictly in the interior of the slice. We denote the set of crosses by X.

We say that a cross c is *hit* by a guard-segment g if g intersects the supporting slice-segment of c that is perpendicular to g. We now show that reducing the problem to just crosses and guard-segments is enough.

Lemma 6 *A set S of m sliding k-transmitters guards P if and only if there exists a set $S' \subseteq \Gamma$ of m guard-segments such that every cross c is hit by some guard-segment $\gamma \in S'$.*

Proof. ($\Rightarrow$) Suppose that we have a set S of m sliding k-transmitters that guards P entirely. Fix one sliding k-transmitter s. Translate s in parallel (i.e., move it horizontally if s is vertical, move it vertically if s is horizontal) until we reach δP. Thus s is now intersecting an edge of P. Extend s so that it is maximal while still within P. Both operations can only increase the region seen. The resulting segment s' is a guard-segment. After doing this to all sliding k-transmitters, we now have a set of guard-segments S' that sees all of P. Now consider any cross $c \in X$. Since c is a point in P, it is guarded by some guard-segment $\gamma \in S'$. Thus, there exists a point $g \in \gamma$ such that the line segment gc is normal to γ and intersects δP in at most k points. But, gc is part of the slice-segment that supports c and is perpendicular to γ. So g is the intersection point between that slice-segment and guard-segment γ.

($\Leftarrow$) This is straightforward by Lemma 5 since (i) any point in P belong to at least one pixel, (ii) there is a 1-to-1 correspondence between the pixels and crosses of P, and (iii) crosses are interior points of pixels. $\square$

Our problem has now been discretized as follows: Given the set X of all crosses, each supported by two line segments, find a subset $S \subseteq \Gamma$ such that for every cross one of the two line segments is intersected by at least one guard-segment in S. We call this the *cross-hitting* problem. This problem is *exactly* the same problem as studied in [4] when solving the sliding-cameras problem (the only difference is in the choice of supporting segments of crosses, which are longer for sliding k-transmitters). They give an $O(1)$-approximation algorithm for this problem which uses no information about how the segments were obtained (other than that they are horizontal or vertical). Using this, we hence have:

Theorem 7 *For any $k > 0$, there exists a polynomial-time $O(1)$-approximation algorithm for guarding an orthogonal polygon with sliding k-transmitters.*

As in [9], we also consider the variant when only horizontal sliding k-transmitters are allowed. This also reduces to the cross-hitting problem, with the only change that we use Γ_H in place of Γ. This in fact simplifies the problem, because now only vertical supporting segments are relevant for crosses. So there is also an $O(1)$-approximation algorithm for guarding an orthogonal polygon with horizontal sliding k-transmitters.

4 Conclusion

In this paper, we studied how to guard an orthogonal polygon using the minimum number of sliding k-transmitters. We showed that this is NP-hard, even if the polygon is y-monotone. We also gave an $O(1)$-approximation algorithm.

The main open problem is to find better approximation factors. (The "$O(1)$" in [4] stems from the use of ε-nets, and the constant is unspecified but likely quite large.) Is the problem APX-hard? Also, for what subclass of polygons is guarding with sliding k-transmitters polynomial? This is true for orthogonally convex polygons (one or two guards are always enough), but are there other, less trivial classes?

References

[1] O. Aichholzer, R. F. Monroy, D. Flores-Peñaloza, T. Hackl, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. In *European Workshop on Computational Geometry*, pages 167–170, 2009.

[2] O. Aichholzer, R. F. Monroy, D. Flores-Peñaloza, T. Hackl, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. *CoRR*, abs/1503.05062, 2015.

[3] B. Ballinger, N. Benbernou, P. Bose, M. Damian, E. D. Demaine, V. Dujmovic, R. Y. Flatland, F. Hurtado, J. Iacono, A. Lubiw, P. Morin, V. S. Adinolfi, D. L. Souvaine, and R. Uehara. Coverage with k-transmitters in the presence of obstacles. *J. Comb. Optim.*, 25(2):208–233, 2013.

[4] T. Biedl, T. M. Chan, S. Lee, S. Mehrabi, F. Montecchiani, and H. Vosoughpour. On guarding orthogonal polygons with sliding cameras. *CoRR*, abs/1604.07099, 2016. Submitted.

[5] T. C. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19(4):427–446, 1997.

[6] S. Cannon, T. G. Fai, J. Iwerks, U. Leopold, and C. Schmidt. Combinatorics and complexity of guarding polygons with edge and point 2-transmitters. *CoRR*, abs/1503.05681, 2015.

[7] M. de Berg, S. Durocher, and S. Mehrabi. Guarding monotone art galleries with sliding cameras in linear time. In *Combinatorial Optimization and Applications (COCOA 2014)*, volume 8881 of *LNCS*, pages 113–125, 2014.

[8] S. Durocher and S. Mehrabi. Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results. In *Proceedings of Mathematical Foundations of Computer Science (MFCS 2013)*, volume 8087 of *LNCS*, pages 314–324, 2013.

[9] M. J. Katz and G. Morgenstern. Guarding orthogonal art galleries with sliding cameras. *Inter. J. Comp. Geom. & App.*, 21(2):241–250, 2011.

[10] E. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

[11] S. S. Mahdavi, S. Seddighin, and M. Ghodsi. Covering orthogonal polygons with sliding k-transmitters. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada, 2014*, 2014.

[12] S. Mehrabi and A. Mehrabi. A note on approximating 2-transmitters. *CoRR*, abs/1512.01699, 2015.

[13] J. O'Rourke. *Art Gallery Theorems and Algorithms.* The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[14] R. Tamassia and I. Tollis. A unified approach a visibility representation of planar graphs. *Discrete & Computational Geometry*, 1:321–341, 1986.

The Length of the Beacon Attraction Trajectory

Bahram Kouhestani, David Rappaport, Kai Salomaa [*]

Abstract

We study the attraction trajectory of a point under the beacon model. We show that when a point object p is attracted to a point beacon b inside a simple polygon, its trajectory is at most $\sqrt{2}$ times longer than the geodesic distance between p and b. We show that in polygons with holes, the ratio between the length of the beacon trajectory and the length of geodesic distance can be unbounded.

1 Introduction

Consider a dense sensor network deployed in a known environment. Geographic greedy routing is a common and efficient strategy to send messages between sensors (nodes) of the network. Each node is assumed to know its planar location and the location of the destination that may be obtained through a local service. Each node has a *neighbourhood* of a constant number of near neighbours that are used for all of its communications. A sensor communicates with a destination by forwarding a message to a node in its neighbourhood that is closest to the destination. The message travels through the network until it reaches the destination or gets stuck on a node that is closer to the destination than any of its neighbours. Geographic greedy routing tends to be efficient, computationally inexpensive, and low-state, that is, each node only needs to store information related to nodes in its neighbourhood. However, it has the drawback of messages at times not reaching their destination.

Motivated by geographic greedy routing, Biro *et al.* [3] introduced the "beacon model". A beacon is a point with a force of attraction that pulls objects towards it. The beacon represents the destination of a message in a dense sensor network. The attraction of the beacon causes a point object to move towards it as long as its Euclidean distance towards the beacon is decreasing. The trajectory of the point represents the path of a message in the sensor network.

Let P be a simple polygon with n vertices. A *beacon* is a stationary point inside P that can induce a force of attraction within P. When beacon b is activated, it can attract a point p in P so that it moves towards b always getting closer to b. Note: we use p to denote the initial location as well as the name of a moving point. The trajectory of point p as it moves toward beacon b either pulls in a straight line towards b or it slides on the boundary of P. See Fig. 1 for an illustrative example.

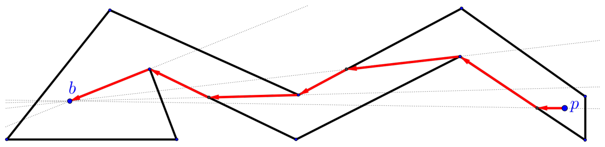


Figure 1: The movement of a point p on its trajectory towards beacon b alternates between being pulled straight towards the beacon and sliding on the polygon boundary.

The beacon model can be used to study and answer various questions that arise in geographic greedy routing, such as the set of destinations a particular node can successfully send a message to, the set of sources a particular node can receive a message from, the set of all nodes that can send a message to any other nodes in the network and the set of all nodes that can receive a message from any node in the network.

A compendium of interesting results pertaining to beacon routing can be found in Biro's PhD thesis [2]. Biro *et al.* [3] studied the combinatorics of guarding a polygon with beacons and showed that $\lceil \frac{n}{2} \rceil$ beacons are sometimes necessary and always sufficient to succesfully route between any pair of points in a simple n-gon. They also proved that it is NP-hard to find a minimum cardinality set of beacons to cover a simple polygon. In 2013, Biro *et al.* [5] presented a polynomial time algorithm for routing between two fixed points using a discrete set of candidate beacons in a simple polygon and gave a 2-approximation algorithm where the beacons are placed with no restrictions. For polygons with holes, Biro *et al.* [4] showed that $\lceil \frac{n}{2} \rceil - h - 1$ beacons are sometimes necessary and $\lceil \frac{n}{2} \rceil + h - 1$ beacons are always sufficient to guard a polygon with h holes. Combinatorial results on the use of beacons in orthogonal polygons have been studied by Bae

*Queen's University, Kingston, ON, Canada, {kouhesta,daver,ksalomaa}@cs.queensu.ca

et al. [1] and by Shermer [9]. Examining another special case Kouhestani*et al.* [7] give an $O(n \log n)$ time algorithm for beacon routing in a polygonal terrain. In [8] Kouhestani *et al* present algorithms to efficiently compute the inverse attraction region of a point for simple, monotone, and terrain polygons with respective time complexities $O(n^3)$, $O(n \log n)$ and $O(n)$.

In this paper, we compare the length of a "successful" beacon trajectory to the length of the shortest path. We show that in a simple polygon the length of a successful beacon trajectory is less than $\sqrt{2}$ times the length of a shortest (geodesic) path. In contrast if the polygon has internal holes then the length of a successful beacon trajectory may be unbounded.

1.1 Definitions

Let e be an edge of P and let L be the supporting line of e. Let h be the orthogonal projection of b on L as shown in Fig 2. Observe that h is the point on L closest to b, so if a point slides on edge e on its trajectory to b the direction of the slide must be towards h. If h is located in the interior of the edge e then a point p sliding on e will stop when it reaches h, otherwise, p slides all the way to an endpoint, v, of e. Let e' denote the neighbour of e sharing endpoint v. From v, the point p is pulled straight towards b if possible, or p slides on e' or p remains stuck on v depending on the location of the orthogonal projection of b on e' (See Fig. 3).

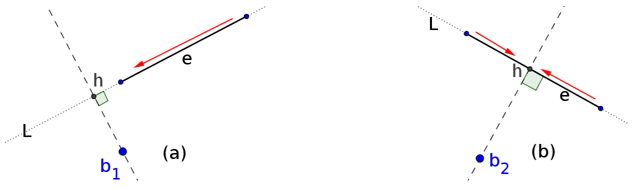


Figure 2: A point object following the beacon trajectory slides on the edge e towards the orthogonal projection of the beacon onto the supporting line of e.

A point p in P is *attracted* by beacon b if its trajectory makes it all the way to b without getting stuck. The *attraction region* of a beacon b is the set of all points in P that b can attract. In [2, 5] Biro *et al.* show that the attraction region of a beacon can be computed in linear time. Whenever p is attracted to b we define its *attraction trajectory*, denoted by $AT(p, b)$, as the path from p to b. We use $|AT(p, b)|$ to denote the length of $AT(p, b)$. Recall, a traversal $AT(p, b)$ from p to b alternates between pulling straight towards

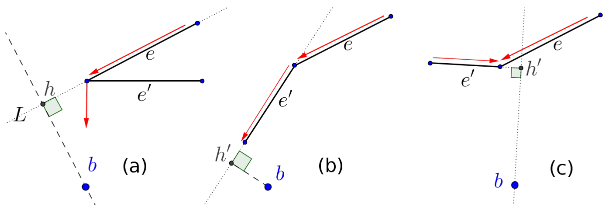


Figure 3: Three outcomes after a point object slides to an endpoint of e. (a) It pulls straight towards b. (b) It slides on the adjacent edge, e'. (c) It get stuck on the endpoint. Here h' is the orthogonal projection of b on the supporting line of the edge e'.

b and sliding on the edges of the polygon. An edge of $AT(p, b)$ is called a *pull edge* if it pulls straight towards b, otherwise it is called a *slide edge*. A polygon P is called a *routable* polygon if for any two points x and y in P, x attracts y. A routable polygon is shown in Fig. 4.

Let b and p be two points inside a simple polygon P. The Euclidean shortest path (geodesic path) between p and b, denoted by $SP(p, b)$, is a path inside P that connects p and b and among all such paths has the smallest path length. Note that $SP(p, b)$ only turns at reflex vertices of P and has the so called *outward convex* property, i.e. the angle facing the exterior of P at every turn on $SP(p, b)$ is convex [6]. We use $|SP(p, b)|$ to denote the length of $SP(p, b)$.

1.2 Related work

Tan and Kermarrec [10] have shown that for any pair of points b and p in a routable polygon, $|AT(p, b)| \leq 3\,|SP(p, b)|$. Their proof is based on the claim that in a routable polygon, the attraction path between b and p passes through all reflex vertices on $SP(p, b)$. We provide an example in which this claim does not hold. We also give conditions when a reflex vertex is shared between the attraction trajectory and the shortest path. The main result of this paper is that in a simple polygon, when b attracts p,

$$\frac{|AT(p, b)|}{|SP(p, b)|} < \sqrt{2}.$$

We also show that in polygons with holes

$$\frac{|AT(p, b)|}{|SP(p, b)|}$$

may be unbounded.

2 The length of attraction trajectory

Let P be a simple polygon. We begin by studying conditions under which reflex vertices of P are shared by $AT(p, b)$ and $SP(p, b)$.

Observation 1 *$AT(p,b)$ does not necessarily pass through all reflex vertices on $SP(p,b)$ (see Fig. 4).*

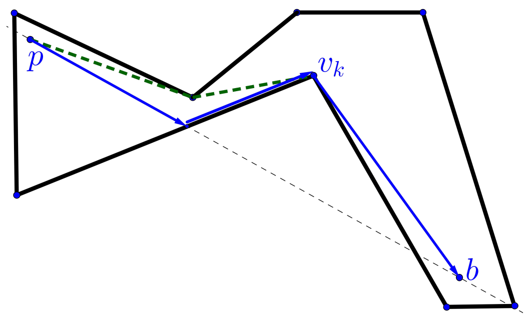


Figure 4: The attraction trajectory $AT(p,b)$ does not necessarily pass through all reflex vertices of $SP(p,b)$ even if the polygon is routable.

The following two lemmas indicate which reflex vertices are always common to $SP(p,b)$ and $AT(p,b)$.

Lemma 1 *Assuming that p does not see b, let v_k be the last reflex vertex on the shortest path from p to b. Then $AT(p,b)$ passes through v_k.*

Proof. Consider the visibility polygon of b in P, i.e. all points in P visible to b. A *window* of b is the boundary between points that are visible and points that are not visible to b. This window can be computed by extending $\overline{bv_k}$ from v_k to the first intersection with the boundary of P. Observe that v_k is the base of a window (shown in dashed red in Fig. 5).

Note that any path from p to b must cross this window. In the attraction of b, a point on the window will move towards b and therefore it moves along the window until it reaches v_k. Thus, we conclude that $AT(p,b)$ passes through v_k. □

Let $SP(p,b)$ be the polygonal chain $p, v_1, v_2, ..., v_k, b$ and assume that b attracts p. We call the edge $\overline{v_iv_{i+1}} \in SP(p,b)$ a *separating diagonal* if cutting through $\overline{v_iv_{i+1}}$ partitions P into two sub-polygons such that b and p are not in the same sub-polygon.

Lemma 2 *Let $\overline{v_iv_{i+1}} \in SP(p,b)$ be a separating diagonal. Then at least one of v_i or v_{i+1} is on $AT(p,b)$.*

Proof. We extend $\overline{v_iv_{i+1}}$ from each endpoint until it intersects the boundary of P. Let w_i (w_{i+1}) be the first intersection of the extension from v_i (v_{i+1}) with the boundary of P (Fig. 5). The line segment $\overline{v_iw_i}$ cuts P into two sub-polygons. Due to outward convexity of the shortest path, p belongs to the sub-polygon below $\overline{v_iw_i}$. Similarly, $\overline{v_{i+1}w_{i+1}}$ cuts P into two sub-polygons and b belongs to the sub-polygon above $\overline{v_{i+1}w_{i+1}}$. There are two cases to consider.

Case 1: b is below the supporting line of $\overline{v_iv_{i+1}}$. We show that $AT(p,b)$ passes through v_i. Consider a point on the open line segment $\overline{v_iw_i}$. In the attraction of b, this point cannot move above $\overline{v_iw_i}$ (directly) without a slide movement. As v_i and w_i are mutually visible, the only slide movement that can move a point above $\overline{v_iw_i}$ occurs at v_i. Therefore, the attraction trajectory must pass through v_i.

Case 2: b is on or above the supporting line of $\overline{v_iv_{i+1}}$. We show that $AT(p,b)$ passes through v_{i+1}. Consider a point on the open line segment w_iv_i. This point cannot move below $\overline{v_iv_{i+1}}$ (directly) without a slide movement. Similarly, as v_i and v_{i+1} are mutually visible, the only slide movement is through v_{i+1}. Therefore, the attraction trajectory of p passes through v_{i+1}.

Thus we have shown that at least one of v_i or v_{i+1} is on $AT(p,b)$. □

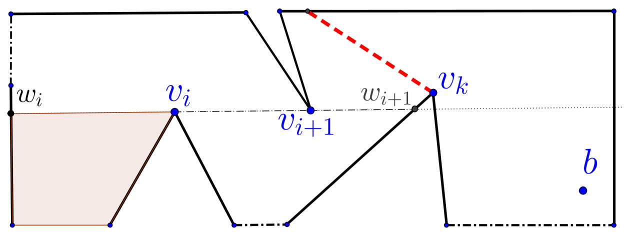


Figure 5: Proof of Lemma 2. The coloured region indicates the possible positions of p.

We compare the lengths of $AT(p,b)$ and $SP(p,b)$ to the Euclidean distance between p and b.

Lemma 3 *The attraction trajectory can rotate an arbitrary number of times around b and the length of $AT(p,b)$ and $SP(p,b)$ can be arbitrarily longer than the Euclidean distance between b and p (See Fig. 6).*

Proof. Let b be a beacon which we assume is at the origin of a plane coordinate system. Let m be an arbitrary positive integer. We partition the plane into m equal angles around b so that each angle has a degree of $\theta = 2\pi/m$ (in Fig. 6, $m = 16$ and $\theta = \pi/8$). Next we construct a spiral shape as follows. We begin at a point p on the negative side of the x-axis and going counter-clockwise we draw a line segment from p orthogonal to the next line. We continue this process to obtain

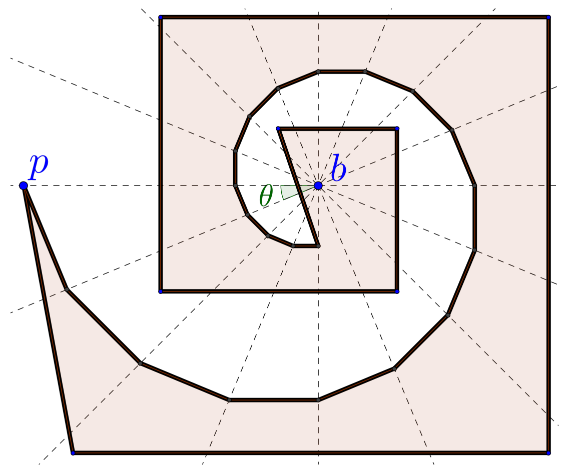


Figure 6: The attraction trajectory $AT(p,b)$ and $SP(p,b)$ follows the polygon boundary and may rotate an arbitrary number of times around b.

a polygonal spiral. This polygonal chain approximates the Archimedean spiral [11]. It is straightforward to construct a simple polygon by adding a convex chain around the spiral as in Fig. 6. Now consider the movement of p in the attraction of b. It slides along the spiral and eventually it is attracted by the beacon. □

In order to determine the quality of the attraction trajectory we compare its length to the length of $SP(p,b)$. We partition $AT(p,b)$ into maximal sub-paths that alternately coincide with and diverge from $SP(p,b)$. We then show that each maximal divergent sub-path of $AT(p,b)$ is at most $\sqrt{2}$ longer than the corresponding part of $SP(p,b)$.

Let u and v be points common to both $SP(p,b)$ and $AT(p,b)$ such that, $AT(p,b)$ diverges from $SP(p,b)$ at u and only returns to a common point at v. We call the subpath of $AT(p,b)$ between the points u and v the *uv fragment* of $AT(p,b)$. We use the notation $F(u,v)$ to denote the uv fragment of $AT(p,b)$.

Without loss of generality we assume that the input is oriented so that the line from the point u to the beacon b, denoted by $L_{u,b}$ is horizontal, with u to the left of b and v above $L_{u,b}$. We adopt a polar coordinate system with polar reference point b, with the components r (the distance of the point to b) and ϕ (the counter clockwise angle between the axis and a line joining b to the point). Note that $SP(u,v)$ does not go below the line $L_{u,b}$, because otherwise $F(u,v)$ and $SP(u,v)$ will intersect before they reach v.

Lemma 4 *Let $F(u,v)$ be the uv fragment of $AT(p,b)$ oriented as described above. The traversal of $F(u,v)$ from u to v monotonically decreases in its r-coordinate and is non-increasing in the ϕ-coordinate.*

Proof. Note that the r-coordinate represents the Euclidean distance between b and (the current position of) p. Therefore by the definition of beacon model the r-coordinate monotonically decreases.

Recall that every attraction trajectory is made up of two types edges, pull and slide edges. Observe that the ϕ-coordinate stays constant as a moving point p is pulled along a pull edge. We complete the proof by showing that the ϕ-coordinate decreases along all slide edges. Observe that a slide edge can begin at a vertex of polygon P or begin from a point interior to a polygon edge. We consider each of these two cases separately.

If a slide begins on a polygon vertex, then it does so because there is no direct pull to b. This implies that the ϕ-coordinate must decrease along the slide.

For the case where a slide begins at an interior point we consider the closed curve obtained by concatenating $F(u,v)$ and $SP(u,v)$ (See Fig. 7). Observe that this closed curve bounds a simply connected subset of the simple polygon P. (Note: This is the crucial point where the absence of holes in P is necessary.) Based on the orientation assumptions, the interior of this simply connected region lies to the left in a counter clockwise traversal of $F(u,v)$. Therefore, if a slide begins from a point interior to a polygon edge again the ϕ-coordinate must decrease along the slide.

□

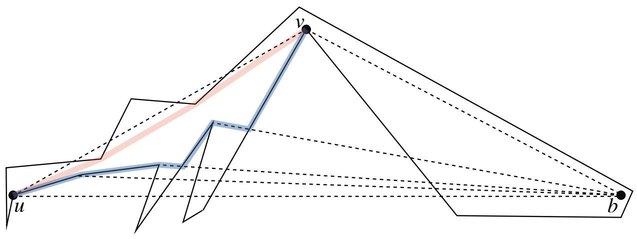


Figure 7: A simple closed subset of polygon P is bounded by concatenating $F(u,v)$ and $SP(u,v)$.

We have established the characteristics of a maximal fragment of $AT(p,b)$ that diverges from $SP(p,b)$. Using these characteristics we obtain an upper bound on the length of this divergent fragment.

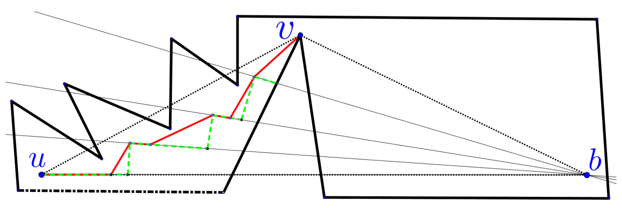


Figure 8: An attraction trajectory is shown in red. Note that we have not drawn portions of the polygon where the sliding occurs. The green path shows a longer attraction trajectory than the original red one.

Theorem 5 *In a simple polygon P, for any two points $b, p \in P$, $|AT(p,b)| < \sqrt{2}\,|SP(p,b)|$.*

Proof. We partition $AT(p,b)$ alternating between fragments that are congruent to $SP(p,b)$ and fragments that are divergent from $SP(p,b)$. We only need to consider the divergent fragments. Consider one such divergent fragment beginning at u and ending at v as given above, and described as the uv fragment of $AT(p,b)$. Let $F(u,v)$ be used to denote the uv fragment of $AT(p,b)$. Let $|\overline{uv}|$ denote the length of the line segment $\overline{uv}$. As $|\overline{uv}| \le |SP(u,v)|$, it is sufficient to show that

$$\frac{|F(u,v)|}{|\overline{uv}|} < \sqrt{2}.$$

In Fig. 8 the red path represents a fragment $F(u,v)$ of the attraction trajectory $AT(p,b)$. We convert this path to a longer one by considering each slide to end on a right angle with the line going through b and the end of slide (shown in green). Next we convert to an even longer attraction trajectory by forcing each slide to be parallel to the green ones but with an end point located on the edge $\overline{uv}$ (shown in blue in Fig. 9). Note that parts of the blue and the green chains that do not collide form a trapezoid where the angle between the two green segments is greater than $\pi/2$. This guarantees that the blue chain is longer than the green chain.

Now consider a triangle bounded by two adjacent blue line segments and the portion of $\overline{uv}$ between these two line segments. As the angle between the blue line segments is at least $\pi/2$, the total length of the blue path is at most $\sqrt{2}$ times the length of $\overline{uv}$. □

3 The length of attraction trajectory in a polygon with holes

In this section we show that in polygons with holes, the length of the attraction trajectory may be arbitrarily longer that the length of the shortest path between b and p. Fig. 10 illustrates an example with this property.

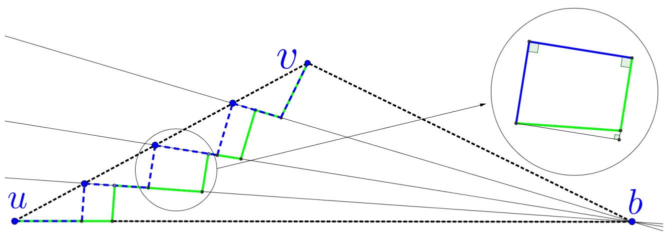


Figure 9: Converting to a longer attraction trajectory.

In this figure, holes are represented as thin spiral shapes which are approximated by a polygonal chain similar to the case shown in Fig. 6. The length of the shortest path between b and p is roughly equal to the length of the line segment $\overline{bp}$, while the attraction trajectory slides on each spiral portion, and therefore it can be arbitrarily long.

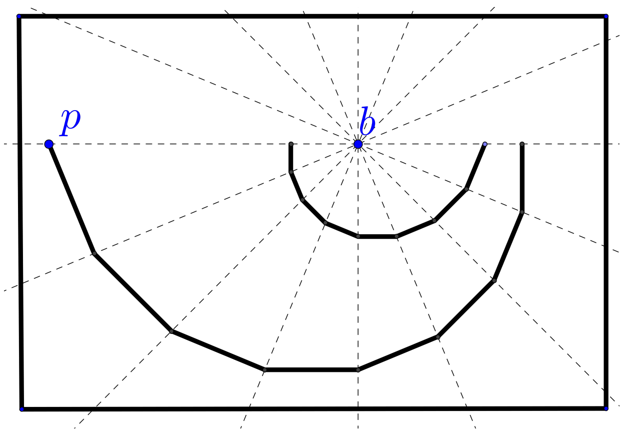


Figure 10: The attraction trajectory can be arbitrarily longer than the shortest path in a polygon with holes.

4 Conclusion

We show that the length of the attraction trajectory is at most $\sqrt{2}$ of the length of the shortest path, indicating that beacon attraction is not only inexpensive to compute but also efficiently short. It remains an open question whether this bound is tight. We conjecture that it is tight, i.e., for any small positive number ϵ, there exists a simple polygon P and points $b, p \in P$ such that $\sqrt{2} - \epsilon < |AT(p,b)| \,/\, |SP(p,b)| < \sqrt{2}$.

In [10] Tan and Kermarrec present a routing protocol which is at most 7 times longer than the shortest path. In their paper they consider the attraction trajectory to be at most 3 times longer than the shortest path. Our result immediately improves their approximation ratio to $4+\sqrt{2}$. We hope that this paper will further motivate the use of greedy routing in various applications.

References

[1] S. Bae, C. Shin and A. Vigneron. Tight Bounds for Beacon-Based Coverage in Simple Rectilinear Polygons. Latin American Theoretical Informatics Symposium, 2016.

[2] M. Biro. Beacon-based Routing and Guarding. PhD Dissertation, Stony Brook University, 2013.

[3] M. Biro and J. Gao, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Beacon-based routing and coverage. Proceedings of the 21st Fall Workshop on Computational Geometry, 2011.

[4] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Combinatorics of beacon routing and coverage. Proceedings of the 25th Canadian Conference on Computational Geometry, 2013.

[5] M. Biro, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Beacon-Based Algorithms for Geometric Routing. Proceedings of the 13th International Symposium on Algorithms and Data Structures, 2013.

[6] S. Ghosh. Visibility Algorithms in the Plane. Cambridge University Press, 2007.

[7] B. Kouhestani, D. Rappaport, K. Salomaa: Routing in a Polygonal Terrain with the Shortest Beacon Watchtower.Proceedings of the 26th Canadian Conference on Computational Geometry, 2014.

[8] B. Kouhestani, D. Rappaport, K. Salomaa. On the Inverse Beacon Attraction Region of a Point. Proceedings of the 27th Canadian Conference on Computational Geometry, 2015.

[9] T. Shermer. A Combinatorial Bound for Beacon-based Routing in Orthogonal Polygons. Proceedings of the 27th Canadian Conference on Computational Geometry, 2015.

[10] G. Tan, A. Kermarrec. Greedy geographic routing in large-scale sensor networks: A minimum network decomposition approach. IEEE/ACM Transactions on Networking, 20(3), 864-877, 2012.

[11] Wikipedia contributors. Archimedean spiral. Retrieved March 14, 2016, from `https://en.wikipedia.org/wiki/Archimedean_spiral`.

A Competitive Strategy for Walking in Generalized Streets for a Simple Robot

Azadeh Tabatabaei[*] Mohammad Ghodsi[†] Fardin Shapouri[‡]

Abstract

In this study, we consider the problem of walking in an unknown generalized street or G-street, for a simple robot. The basic robot is equipped with a sensor that only detect the discontinuities in depth information (gaps). In the former recent researches some competitive strategies for walking the robot in street polygons have been presented. In this research we have empowered the robot by adding a compass to patrol a more general class of polygons. We present an online strategy that generates a search path for the empowered robot in G-streets; a more general class of polygons that contains all streets properly. The empowered robot, using the local information gathered through its sensors and using some pebbles as marker, locates target t, starting from a vertex s in a G-street. Length of the traveled path by the robot is at most 9.06 times longer than the shortest path. The competitive ratio is optimal.

1 Introduction

Exploring an unknown environment is a fundamental problem characterized by researcher in robotics, computational geometry, game theory and online algorithm [9, 12]. An autonomous mobile robot without access to the geometry of the scene depending the information collected through its sensor moves to reach a goal. Variants of robot models, and problems have been studied in this context [1, 4, 7]. We are interested in using a minimalist robot model system for walking in unknown scene.

Our basic robot is a simple point robot with the sensing model of gap sensor. At each point the robot locates the depth discontinuities (gaps) of its visibility region in a circularly ordered, (Figure 1). All times the robot can track the gaps and detects each topological changes of the gaps. These changes are the appearance, disappearance, merging, or splitting of gaps which are called *critical events*. While the robot traverses an environment, it can rotate as often as each of the critical events arises, or a target point enters in its visibility region.

In order to measure the performance of an online search strategy, the notation of competitive analysis is used. The competitive ratio is the worst case ratio of the path traveled by the robot in the unknown environment to shortest path. Tabatabaei and Ghodsi designed an online strategy for the simple robot to walk in streets. By the strategy the robot explores a street from a vertex s to a vertex t such that the traveled distance by the robot is at most 9 times longer than the shortest path [15]. A street polygon is characterized by the feature that the two boundary chains from s to t are mutually weakly visible.

In this study, our goal is equipping the simple robot with a smallest set of additional capabilities to empower the robot for searching more general classes of polygons by a competitive search strategy. So, we consider the following extension of the simple robot. The robot carries a compass that denotes to it the north, west, south and east directions. It can moves toward the directions, in additional to the gap tracking. Also, it can put a pebble for marking anywhere. We present an online search strategy for exploring a generalized street environment, from a vertex s to a vertex t, for the empowered robot with the competitive ratio of 9.06. A generalized street is a polygon for which every point on its boundary is visible from a point on a horizontal line segment that connects the two boundary chains from s to t, see Figure 1.

Our robot sensing model is strongly weaker than model of the robot in the previous research. Datta and Icking presented an optimal online strategy with a competitive ratio of 9.06 for searching a generalized street [2]. Datta and Icking robot equipped with a 360 degrees vision also, it memorized the map of the scene has seen so far while our robot using the local information gathered through its sensor walks in the street. The ratio of 9.06 is optimal, Lopez-Ortiz and Schuierer have shown the lower bound of 9.06 in [9].

Related Works: Klein presented the first competitive strategy for walking in streets problem for a robot that was equipped with a 360 degrees vision system [8]. Many online algorithms for patrolling unknown environments such as street, generalized street, and star polygons are proposed in [6, 10].

[*]Department of Computer Engineering, Sharif University of Technology, atabatabaei@ce.sharif.edu

[†]Sharif University of Technology and School of Computer Science, Institute for Research in Fundamental Sciences (IPM), ghodsi@sharif.edu

[‡]Department of Computer Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran, shapouri@qiau.ac.ir

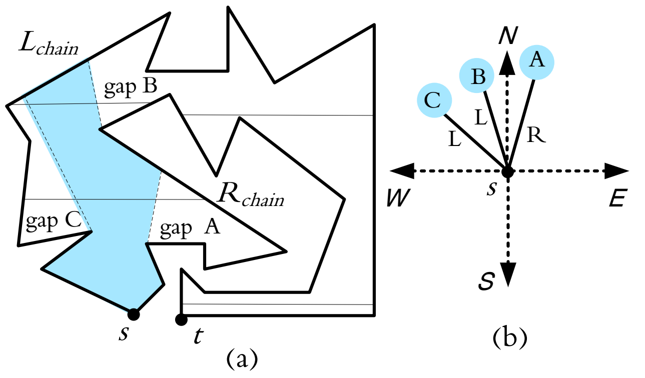


Figure 1: (a) A G-street polygon. The colored region is the visibility polygon of the point robot at the start point s. (b) The position of discontinuities in the depth information (gaps) reported by the sensor and directions of the compass.

The basic sensing model (gap sensor) that our robot is equipped with, in this study, was first introduced by Tovar, Murrieta-Cid, and LaValle [17]. They offered Gap Navigation Tree (GNT) as a means to record and update the gaps seen along an exploring path. Other researcher presented some strategies, using GNT, for searching unknown environments [5, 11, 13]. An optimal search strategy with minimum number of turns, for the basic simple robot equipped with the gap sensor, presented in [16].

Another minimal sensing model offered by Suri, Vicari, and Widmayer [14]. They assumed that the simple robot can only sense the combinatorial (non-metric) properties of the environment. The robot can locate the vertices of the polygon in its visibility region, and can report if there is a polygonal edge between them. Despite of the minimal capability, they showed that the robot performs many non-trivial tasks. Then, Disser, Ghosh, Mihalak, and Widmayer empowered the robot with a compass to solve the mapping problem in polygons with holes [3].

2 preliminaries

2.1 Workspace

Generalized street polygons are considered as the robot workspace. So, we briefly repeat its definitions, and some of its properties. In a simple polygon P with two vertices s and t the counter-clockwise polygonal chain from s to t is called the *right chain* or R_{chain}, and the clockwise one from s to t is called the left chain or L_{chain}.

Defnition 1 *[8] A polygon is a street polygon if each point on the left chain is visible from at least one point on the right chain and vice versa.*

Defnition 2 *[2] A chord is a horizontal line segment inside a polygon P such that its both end points are on the boundary of P. The chord is called an LR-chord when it touches both the L_{chain} and R_{chain}.*

Defnition 3 *[2] A simple polygon in the plane is called a generalized street or G-street if for every boundary point $p \in L \cup R$, there exists an LR-chord c such that p is visible from a point on c.*

Figure 1 displays an example of a G-street, and Some of its LR-chords, the horizontal lines. Class of the generalized street polygons is strictly larger than class of the street polygons that our empowered robot explores it.

2.2 Sensing Model and Motion

From the vertex s in an unknown G-street, a simple robot starts walking to achieve the target t. The robot based on the local information collected by its sensor explores the scene. The basic robot is a point robot equipped with a sensor that detects each discontinuity in depth information that referred as gaps. The sensor reports a cyclically ordered location of the gaps in its visibility region. Also, the robot assigns a label of L or R (left or right) to each gap based on the direction of the hidden region that is behind the gap [17], (Figure 1). The robot can only track the gaps and detects their topological changes. These changes are: appearance, disappearance, merging, and splitting of gaps. The appearance and disappearance events arise when the robot crosses the inflection rays, (at point 3 in Figure 2 gap $R - EG_1$ disappears). The merge and split events occur when the robot crosses a bitangent complement, (at point 2 in Figure 2 gap $L - EG_1$ splits).

The robot can move along a straight line towards gaps to cover the region hidden behind it. The robot may rotate as a critical event occurs, or as soon as the target enters in the robot's visibility region. By equipping the robot with a compass, the robot is empowered; such that it can detects and tracks the north, west, south and east directions, see Figure 1. Also, some pebbles are available to the robot as marker.

2.3 Main Strategy for Walking in Rectilinear G-streets

Now, we present an online search strategy for walking in a rectilinear G-street in which all of the edges are either horizontal or vertical, (Figure 2). From the start vertex s, the robot starts walking in the G-street to reach the target t, using the information collected by the gap sensor and the compass. The goal is to minimize length of the search path.

In contrast with the previous research, our robot has no access to the map of its visibility region has seen so far, and especially angles of the region boundary. Only

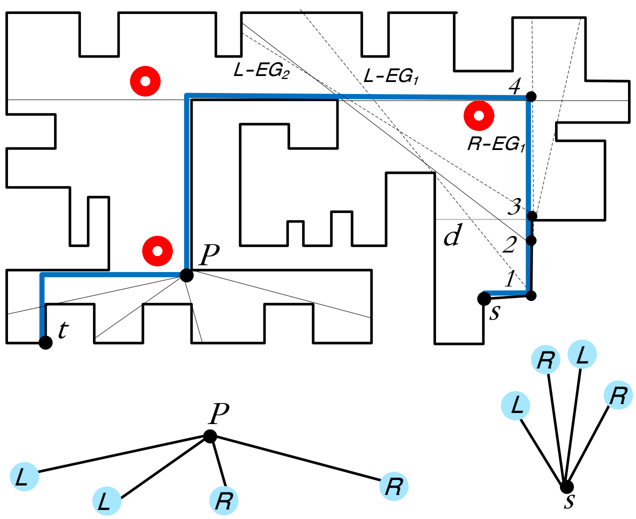


Figure 2: A rectilinear G-street, and located gaps (in unexplored region) at points s and p. $R-EG_i$ and $L-EG_i$ are essential gaps at point i. Colored bold path is the robot search path (traversed path for performing the doubling search strategy to reach the turn points is omitted). Colored discs are the pebbles for marking the regions that the robot comes from.

local information about the gaps location and the compass directions, north, south, east and west, are available to the simple robot.

There is a reflex vertex correspond to each gap. We refer to the horizontal chord that crosses the vertex as a *gap chord.* Each gap chord divides the polygon into three parts, or two parts as shown in Figure 2. Also, the edge of the polygon that is collinear with the gap chord is called as *gap edge.*

Defnition 4 *A gap is an essential gap when the target is hidden beyond its gap chord. If the gap is a right/left gap it is denoted by $R-EG/L-EG$.*

From the definition of G-street following Lemma is straightly obtained.

Lemma 1 *The target vertex t or an essential gap is visible from some point on the horizontal line containing the start point.*

Examples of gap chord, gap edge, and essential gap are shown in Figure 3. The important key is that the simple robot how can distinguish if a gap is essential.

Theorem 2 *While the robot searching on the horizontal line, it reports an essential gap as soon as it locates a vertical gap that is collinear with north or south directions of the compass.*

Proof. Assume, d is the gap chord of the detected vertical gap. It divides the polygon into three parts, or two parts. s and t must be in different parts. If they are in the same part which contains the horizontal line, then there is at least one point on edge of gap chord which is not visible from any LR-chord and the polygon is not a G-street, a contradiction. So, d is an LR-chord that each path from s to t intersects it. Then, the vertical gap is an essential gap. See the vertical gap detected at point 1 in Figure 2. □

We refer to the point in theorem 2 as a *turn point.* Note that at the turn point the robot detects a vertical essential gap.

Theorem 3 *If the detected gap, at the turn point is $L-EG$, the first right gap which lies clockwise after the gap is $R-EG$. Analogously, if the gap is $R-EG$, the first left gap which lies counterclockwise after the gap is $L-EG$.*

Proof. Assume that at the turn point, $R-EG$ is detected, also there is a left gap clockwise after the essential gap. If the robot moves towards left direction, the left gap will coincide with the vertical direction. So, by definition of the essential gap, the left gap is $L-EG$, see the turn point 1 in Figure 2. The other case is similar. □

Now, by the above discussion, we can describe our algorithm. At the start point the simple robot searches on the horizontal line containing the start point. Three cases may arise.

- If all of the detected gaps are in the left side of the vertical line containing s, the robot moves toward left to reach the turn point, using the compass.
- If all of the detected gaps are in the right side of vertical line containing the start point, the robot moves toward right to reach the turn point. For example consider point p as start point in Figure 3.
- If there are some gaps in both sides of the vertical line, we performs the doubling strategy; the robot walks back and forth on the horizontal line, at each step doubling the distance to the start point, until the turn point is reached (start point in Figure 2).

At the turn point, from Theorem 3, the robot can locate the other existing essential gap. Assume, the essential vertical gap is $R-EG$; the other case is similar. The robot moves towards the gap along the vertical direction while maintaining location of $L-EG$. During the walking, $L-EG$ may disappears, also it updates as the robot crosses over bitangent compliment of $L-EG$ and another left gap, see point 2 in Figure 2. The robot continues walking along the vertical direction until the vertical $R-EG$ disappears.

At the event point, the first right gap (if exits) which lies clockwise after the movement direction is current $R-EG$; by a similar argument to the proof of Theorem 3. Now, at the event point, there are three cases:

1. $L-EG$ and $R-EG$ exist (point 3 in Figure 2). The robot continues walking along the current vertical direction while maintaining and updating the essential gaps until the two gaps disappear (case 3 arises).

2. One of the essential gaps exists (point 2 in Figure 3). The robot continues walking along the current vertical direction until the gap merges with another gap, or the gap disappears. In the former case, the robot turns and moves along the horizontal direction in the region containing the gap until achieves a turn point in which a vertical essential gap is detected, (point 3 in Figure 3). In the later case when the essential gap disappears, case 3 arises.

3. No essential gap exists (point 4 in Figure 2). The robot puts a pebble in the region where it comes from. Then, same as the start point, it performs the doubling search strategy on the horizontal direction in region containing its current location for achieving a turn point. The detected essential gap must be in a region other than the marked region by the pebble.

   Note that an especial case arises when the robot, after finding the first turn point and moving along the vertical direction, has to perform the doubling strategy again for achieving the next turn point, without any movement along a horizontal direction. In this case, the robot resumes the previous doubling strategy (point 4 in Figure 2).

The robot repeats the process until the target t is visible. At the point, the robot, using the compass direction, can move a long a rectilinear path to achieve the target t.

2.4 Analysis of the Algorithm

We show that our simple robot, using the local information about location of gaps and the compass achieves the target t starting from s in the G-street. Although our robot is strongly weaker than Datta and Icking robot [2], and its search path differs from the robot path, we demonstrate the competitive ratio of our strategy is 9.06 like their strategy.

Theorem 4 *Our search strategy terminates while a search path to t is generated, starting from s in the G-street.*

Proof. At the start point, if the target is not visible to the robot, it is hidden behind the existing gaps. The robot searches on the horizontal line to reach an essential gap, then tracks the gap which is a correct vertical direction. The robot continues moving a long a direction unless both of $R-EG$ and $L-EG$ disappear. Then, the robot on the horizontal line searches for another essential gaps, and selects again a correct vertical direction which results in being one step closer to the target. So, the strategy terminates. □

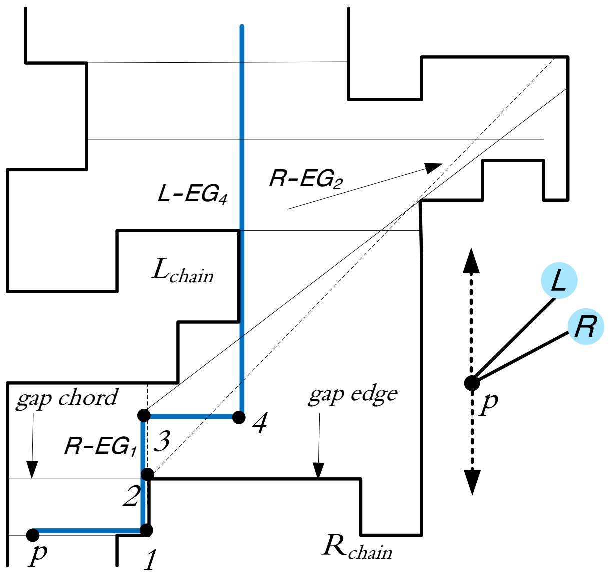


Figure 3: Traversed path between consecutive gap chords, the L_1-shortest path.

Lemma 5 *[1] The doubling strategy for searching a point on a line has a competitive factor of 9 which is optimal.*

Lemma 6 *[9] Lower bound of the competitive factor for each online search strategy in G-street is 9.06.*

Theorem 7 *Competitive ratio of our strategy is 9.06, and this is optimal.*

Proof. In order to compute the ratio, we compare length of the generated path with L_1-shortest path. As explained in the strategy our robot always chooses a correct vertical direction (as shown in Figure 3). For detecting an essential gap, the robot either moves along a correct direction or performs the doubling strategy. So, the horizontal traversed path is most 9 times longer than the shortest path. It means that looking for a target in a G-street is at least as hard as searching a point on a line. So, from Lemma 5, the competitive factor of our strategy is 9 in L_1- metric. Now by an argument, similar to the proof of the competitive factor of Datta and Icking strategy [2], we show the competitive ratio of our strategy is 9.06 in L_2- metric. Note that the two paths are different. The length of the L_2-shortest path between two consecutive turn points in which the essential gaps are reported is $\sqrt{x_1{}^2 + y_1{}^2}$). By our strategy, length of the simple robot's path is at most $9x_1 + y_1$. The maximum value of $\frac{9x_1+y_1}{x_1{}^2+y_1{}^2}$ is 9.06. Then, from Lemma 6 our strategy is optimal.

□

3 Conclusions

In this paper, we studied the problem of walking in generalized streets for a simple point robot. The basic robot has a minimal sensing model that can only detect the gaps and the target in the street. We have empowered the robot by adding a compass. The robot using local information reported by its sensor explores the scene while in the former research a robot that memorizes the region has seen so far, by its complete vision system, searches the scene. We demonstrated that, despite the weakness in our robot system model, performance of our strategy equals with the optimal strategy for the stronger robot; the competitive ratio of 9.06. Proposing a competitive search strategy for more general classes of polygons and offering other minimal sensing model are attractive problems for future research.

References

[1] Baeza-yates, Ricardo A., Joseph C. Culberson, and Gregory JE Rawlins. Searching in the plane. *Information and Computation 106(2): 234-252, 1993.*

[2] Datta, Amitava, and Christian Icking. Competitive searching in a generalized street. *In Proceedings of the tenth annual symposium on Computational geometry, pp. 175-182. ACM, 1994.*

[3] Disser, Yann, Subir Kumar Ghosh, Matus Mihalak, and Peter Widmayer. Mapping a polygon with holes using a compass. *Theoretical Computer Science, 553: 106-113, 2014.*

[4] Fekete, Sandor P., Joseph SB Mitchell, and Christiane Schmidt. Minimum Covering with Travel Cost. *Journal of Combinatorial Optimization, 24: 32-51, 2003.*

[5] Guilamo, Luis, Benjamin Tovar, and Steven M. LaValle. Pursuit-evasion in an unknown environment using gap navigation trees. *Intelligent Robot's and Systems Proceedings Vol. 4, (pp 3456-3462), 2004.*

[6] Ghosh, Subir Kumar, and Rolf Klein. Online algorithms for searching and exploration in the plane. *Computer Science Review 4(4): 189-201, 2010.*

[7] Hammar, Mikael, Bengt J. Nilsson, and Mia Persson. Competitive exploration of rectilinear polygons. *Theoretical computer science 354(3): 367-378, 2006.*

[8] Klein, Rolf. Walking an unknown street with bounded detour. *Computational Geometry, 1(6): 325-351, 1992.*

[9] Lopez-Ortiz, Alejandro, and Sven Schuierer. Generalized streets revisited. *Algorithms ESA'96. Springer Berlin Heidelberg, 546-558, 1996.*

[10] Lopez-Ortiz, Alejandro, and Sven Schuierer. Lower bounds for streets and generalized streets. *International Journal of Computational Geometry and Applications (11)04: 401-421, 2011.*

[11] Lopez-Padilla, Rigoberto, Rafael Murrieta-Cid, and Steven M. LaValle. Optimal Gap Navigation for a Disc Robot. *In Algorithmic Foundations of Robotics, Springer Berlin Heidelberg, (pp 123-138), 2012.*

[12] Mitchell, Joseph SB. Geometric shortest paths and network optimization, Handbook of Computational Geometry. *Elsevier Science Publishers, 1998.*

[13] Sachs, Shai, Steven M. LaValle, and Stjepan Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *The International Journal of Robotics Research 23(1): 3-26, 2004.*

[14] Suri, Subhash, Elias Vicari, and Peter Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *The International Journal of Robotics Research, 27(9): 1055-1067, 2008.*

[15] Tabatabaei, Azadeh, and Mohammad Ghodsi. Walking in Streets with Minimal Sensing. *Journal of Combinatorial Optimization, 30(2): 387-401, 2015.*

[16] Tabatabaei, Azadeh, and Mohammad Ghodsi. Optimal Strategy for Walking in Streets with Minimum Number of Turns for a Simple Robot. *Combinatorial Optimization and Applications. Springer International Publishing, (pp. 101-112), 2014.*

[17] Tovar, Benjamn, Rafael Murrieta-Cid, and Steven M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *Robotics IEEE Transactions 23(3): 506-518, 2007.*

Polynomial volume point set embedding of graphs in 3D*

Farshad Barahimi† Stephen Wismath‡

Abstract

Two algorithms are presented for computing a point-set embedding of a graph in 3D on a given point set with a volume that is polynomial in the size of the graph and the size of the point set, and with at most a logarithmic number of bends per edge. This resolves the previously open *general 3D point set embedding problem* [12].

1 Introduction

A *drawing* of a graph, is a mapping of each vertex to a point in 2D or 3D Euclidean space and each edge to a simple curve between the mapped points of its endpoints. Although 2D graph drawing has been studied extensively, there has also been some significant progress on drawing graphs in 3D. One such model is a *3D Fary grid drawing*, in which each vertex is mapped to an integer grid point in 3D Cartesian coordinate system and each edge is mapped to a straight line segment, such that there is no crossing between edges or vertices.

Cohen, et al. [5] showed that it is possible to have a 3D Fary grid drawing of any graph with n vertices such that the volume does not exceed $n \times 2n \times 2n$. Although they proved that their $O(n^3)$ result is asymptotically optimal for complete graphs, other classes of graphs can be drawn in a lower volume. Calamoneri and Sterbini [4] showed that it is possible to draw every 4-colorable graph on integer coordinates and with no crossing in an $O(n^2)$ volume. Pach, et al. [13] showed that for any constant r, every r-colorable graph can be drawn crossing-free on integer coordinates in $O(n^2)$ volume. They also showed that their result is asymptotically tight by showing that a balanced complete 2-partite graph with n vertices requires $\Omega(n^2)$ volume.

Bose, et al. [2] showed that the maximum number of non-crossing edges that can be contained in an $X \times Y \times Z$ volume is exactly $(2X-1)(2Y-1)(2Z-1) - XYZ$ and as a result, $\frac{m+n}{8}$ is a lower bound for the volume of a 3D Fary grid drawing of a graph with n vertices and m edges.

Felsner, et al. [9] showed that it is possible to have a 3D Fary grid drawing of any outerplanar graph with n vertices in $O(n)$ volume, using a 3D prism. It remains an open problem to determine if all planar graphs can be drawn in linear volume.

Although in the above results each edge is a straight line segment, another model of drawing graphs in 3D introduces bends to subdivide an edge into straight line segments. Unless otherwise specified, we assume here that all such bend points occur at points with integer coordinates.

Dujmović and Wood [8] showed that it is possible to obtain a 3D crossing-free grid drawing of every graph with n vertices and m edges in a $O(n + m \ \log q)$ volume and with $O(\log q)$ bends per edge, where q is the queue number of the graph. The problem of computing the queue number of a graph is NP-Complete [10]. Di Battista, et al. [6] showed that the queue number of every planar graph is $O(\log^2 n)$ and based on these two results, every planar graph can be drawn crossing-free on integer coordinates in an $O(n \log\log n)$ volume and with $O(\log\log n)$ bends per edge; they also showed that any planar graph can thus be drawn in $O(n \log^8 n)$ volume. Dujmović [7] has recently shown that the queue number of planar graphs is $O(\log n)$, thus improving the volume bound to $O(n\ logn)$.

After acceptance of this paper, we were made aware of a result by D. Wood [15] that uses a technique similar to ours. Both these results were obtained independently.

1.1 Point set embedding

The class of point set embedding problems studies the layout of graphs when a set of fixed points are given for the location of vertices. If the mapping between the vertices and points is specified then it is called *with mapping* otherwise it is called *without mapping*. In the with mapping variant of the problem the layout is determined only by establishing the position of the bends, whereas in the without mapping variant of the problem, identifying the mapping between the vertices and the given point set, is also required.

One formulation of the two dimensional point set embedding problem (2DPSE) was suggested by Meijer and Wismath [12]:

> Given a planar graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with inte-

*Supported in part by the Natural Sciences and Engineering Research Council of Canada.

†Department of Mathematics and Computer Science, University of Lethbridge, farshad.barahimi@uleth.ca

‡Department of Mathematics and Computer Science, University of Lethbridge, wismath@uleth.ca

ger coordinates in the plane, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in an area polynomial in n and the dimension of P?

Cabello [3] considered a version of the problem where bends are not allowed and proved that it is NP-Hard to determine whether a planar graph has a straight-line crossing-free drawing on a predefined set of points when the mapping between the vertices and the points is not specified. Pach and Wenger [14] proved that it is possible to draw any planar graph crossing-free on a predefined set of points with $O(n^2)$ bends per edge where the mapping between vertices and points is fixed (but bend points are not constrained to occur at integral coordinates). Kaufmann and Wiese [11] proved that it is possible to have a crossing-free drawing of every planar graph, with at most two bends per edge where each vertex can be positioned at any point of a set of predefined positions, but the area of the drawing may be exponential.

In 3D similar issues can be considered. Meijer and Wismath [12] formulated the three dimensional point set embedding problem (3DPSE) as follows:

Given a graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and a set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in a volume polynomial in n and the dimension of P?

In this paper without loss of generality, the bounding box of P is assumed to range from $(1, 1, 1)$ to (w, l, h).

In [12], this general problem is stated as an open problem and solutions to modified versions of the problem are given. Barahimi [1], in his master's dissertation provided two algorithms for the general problem, one of which is presented in section 2, and the second one is replaced by another algorithm discussed in section 3.

The first modification to 3DPSE that is considered in [12] is to remove the polynomial volume constraint from the problem definition. They prove that K_n can be drawn crossing-free on any predefined set of integer points in 3D with at most 3 bends per edge, but the volume is unbounded. The proof incrementally adds edges to the graph. For each endpoint of each edge, a visible bend point outside the bounding box of the current drawing is found and the endpoint is connected to that bend. The bends found for each edge can be connected by finding a third visible bend point and connecting both to it. The idea of finding visible bend points is used in the proposed algorithms in sections 2 and 3 but the visible bend points are found in a bounded volume.

The second modification to 3DPSE that is considered in [12] is to restrict P to the XY plane and the problem is called $3DPSE_p$. They proved that a graph with n vertices and m edges can be drawn crossing-free in 3D with vertices on a predefined set of integer points in a $W \times H$ rectangular area of the XY plane using $O(\log m)$ bends per edge and within a bounding box of $\max(W, m) \times (H + 3) \times (2 + \log m)$. To create such a drawing, they first introduce a method to draw a perfect matching of two sets of m points in 2D on $O(\log m)$ tracks with $O(\log m)$ bends per edge and no X-Crossings. An X-Crossing occurs if there are two edges (u, v) and (w, z) such that u and w are on the same track and v and z are both on a different track, and u appears before w in their track but v appears after z in their track. This track layout can be converted to 3D without any edge crossings in a box of volume of $m \times 3 \times (1 + \log m)$ and with $O(\log m)$ bends per edge. This technique is also used in the first proposed algorithm of this paper described in section 2. To draw an arbitrary graph in 3D, two lines are considered and for each edge two bend points are added, one on the first line and one on the second line. In the first line the order of the bends representing edges is lexicographic meaning that edges of the vertex v_i appear before the edges of the vertex v_{i+1}. For the second line the order of the bends representing edges is opposite of the first line meaning that edges of the vertex v_i appear after the edges of the vertex v_{i+1}. The two corresponding bends of each edge on these two lines are connected on $O(\log n)$ tracks with $O(\log n)$ bends using the perfect matching technique. Next another line is added for vertices of the graph and vertices are connected to the corresponding bend of their incident edges without creating any crossings. For the $3DPSE_p$ problem, without loss of generality it is assumed that the vertices are ordered by X coordinate and then by Y coordinate in case of a tie. The vertices are placed in the $Z = 0$ plane. The first line for the matching is placed at the $Z = -1$ plane and the second line of the matching is put on the $Z = 1 + \log m$ plane.

Barahimi [1], in his master's dissertation proposed two algorithms for the general $3DPSE$ problem.The first algorithm which is also presented here creates a drawing of volume $O(m + n + w) \times O(m + n + l) \times O(\log n + h)$, with at most $O(\log n)$ bends per edge. A second algorithm is proposed in this paper which fits the drawing in a $O(m + n + w) \times O(m + n + l) \times O(h)$ volume and uses only one bend per edge.

2 The algorithm with a logarithmic number of bends per edge

In this section an algorithm is given which will produce a drawing of size
$O(m + n + w) \times O(m + n + l) \times O(\log n + h)$, with at most $O(\log n)$ bends per edge.

2.1 General idea

The algorithm has three phases and the general ideas are outlined below while details follow later:

- Phase one: Consider two rectangles R_A and R_B that lie in planes parallel to the XY plane. R_A is one unit in front of the bounding box of the points in the direction of the Z axis and R_B is one unit from the back of the bounding box of the points in the direction of the Z axis. For each edge find two visible integer bend points, one in R_A and one in R_B. Connect the first vertex of the edge to the bend point in R_A and connect the second vertex of the edge to the bend point in R_B.

- Phase two: Consider two lines L_A and L_B parallel to the Y axis. L_A is at least one unit in front of R_A in the direction of the Z axis and two units to the left of R_A in the direction of the X axis. L_B is one unit from the back of R_B in the direction of the Z axis and two units to the left of R_A in the direction of the X axis. Connect each bend point in R_A to a corresponding integer bend point in L_A and connect each bend point in R_B to a corresponding integer bend point in L_A.

- Phase three: Each edge has two corresponding bend points in L_A and L_B. If the corresponding bend points of each edge in L_A and L_B are connected then they form a matching. This matching can be drawn crossing free using the perfect matching technique of [12] in a bounding box of $3 \times m \times (1 + \log m)$.

Figure 1 shows a conceptual picture of R_A, R_B, L_A, L_B and the bounding box of the points.

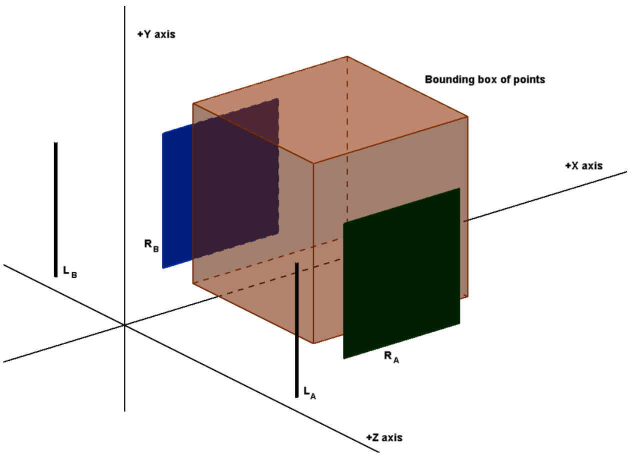


Figure 1: The conceptual picture of R_A, R_B, L_A, L_B and the bounding box of the points.

2.2 Phase one

Let $k = \max(n, m)$. Let P_A denote the plane $z = h + 1$ and R_A denote the rectangle going from $(1, 1, h + 1)$ to $(2k, 2k, h+1)$ in the plane P_A. Let P_B denote the plane $z = 0$ and R_B denote the rectangle going from $(1, 1, 0)$ to $(2k, 2k, 0)$ in the plane P_B. A point s is visible from point t if the line segment connecting s to t does not intersect any vertex of G or any line segment that is previously drawn. The edges are considered one by one in m steps. At the i^{th} step $(1 \leq i \leq m)$, the i^{th} edge e_i, connecting vertices u_i and w_i is considered. Now a visible integer bend point a_i from u_i is found in R_A and a line segment α_i is drawn between u_i and a_i. Next a visible integer bend point b_i from w_i is found in R_B and a line segment β_i is drawn between w_i and b_i. At the end of this phase each edge has one corresponding bend point in R_A and one corresponding bend point in R_B.

To prove that there is always a visible integer bend point from u_i in R_A, or from w_i in R_B, at the i^{th} step of this phase of the algorithm, consider that there are only two ways that an integer bend point in R_A or R_B becomes invisible from u_i or w_i:

1. A previously drawn line segment is between R_A and u_i, or R_B and w_i. The previously drawn line segment can be any of α_j or β_j for $1 \leq j < i$, or α_i for w_i. There are at most $2k - 1$ such line segments and each line segment can make at most $2k$ integer points of R_A or R_B invisible. So this case will make at most $(2k - 1)2k$ integer points of R_A or R_B invisible. To prove that each such line segment connecting vertices or bend points q and t, will make at most $2k$ integer points in R_A or R_B invisible from a vertex v which can be u_i or w_i, consider the plane P_{vqt} containing v, q and t. If the plane P_{vqt} intersects with the plane P_A or P_B the intersection will be a line. This line can contain at most $2k$ integer points of R_A or R_B. If v, q, and t are collinear, at most one integer point of R_A or R_B is made invisible.

2. A vertex is between R_A and u_i, or R_B and w_i: This can be any vertex other than u_i or w_i. Each such vertex can make at most one integer point of R_A or R_B invisible. There are at most $k-1$ such vertices. So this case can make at most $k - 1$ integer points of R_A or R_B invisible.

Subtracting the maximum number of invisible points of both cases from the number of integer points of R_A or R_B, leaves at least $k + 1$ visible points as shown in equation 1.

$$4k^2 - (2k - 1)2k - (k - 1) = k + 1 \tag{1}$$

2.3 Phase two

Let $\lambda = \max(h+2, \log m)$. Let L_A denote the line segment going from $(-1,1,\lambda)$ to $(-1,m,\lambda)$ and let L_B denote the line segment going from $(-1,1,-1)$ to $(-1,m,-1)$.

For each bend point a_i in R_A, find a corresponding integer bend point in L_A called $\bar{a}_i$ and draw a line segment between a_i and $\bar{a}_i$. To find such corresponding bend points, consider the integer bend points of L_A in the order of increasing Y coordinate and consider a_i integer bend points in the order of X coordinate and in case of a tie in the order of Y coordinate, and match them one by one. This ordering will avoid any crossings.

Similarly, for each bend point b_i in R_B, find a corresponding integer bend point in L_B called $\bar{b}_i$ and draw a line segment between b_i and $\bar{b}_i$. To find such corresponding bend points, consider the integer bend points of L_B in the order of increasing Y coordinate and consider b_i integer bend points in the order of X coordinate and in case of a tie in the order of Y coordinate, and match them one by one. This ordering will avoid any crossings. At the end of this phase each edge has four corresponding bend points, one in R_A, one in L_A, one in L_R and one in L_B.

2.4 Phase three

Each edge e_i has a corresponding bend point $\bar{a}_i$ in L_A and a corresponding bend point $\bar{b}_i$ in L_B. If each $\bar{a}_i$ is connected directly to each $\bar{b}_i$ they form a perfect matching but it may introduce crossings. To avoid crossings the perfect matching technique of [12] is used to draw this perfect matching in 3D. Such a 3D perfect matching drawing can be drawn in a bounding box of size $3 \times m \times (1 + \log m)$ using the [-2,0] range of X coordinates and at most $O(\log n)$ bends per edge. Also it is notable that this phase does not use any bend point on the two lines $X = 0, Z = \lambda$ and $X = 0, Z = -1$, otherwise it may introduce crossings with the line segments of the previous phase. This phase will add at most $O(\log n)$ bends per edge, and at most three units to the dimension of drawing in X direction.

2.5 Summary

Each phase of the algorithm does not create any crossings. Each of the three phases uses different partitions of space which will avoid crossings between the three phases.

To find the visible points, for each vertex v, the algorithm maintains a set of integer points in R_A or R_B that are visible from v. The set is implemented using a balanced binary search tree. After adding each line segment at each step of the algorithm, for each vertex v, the algorithm removes the integer points blocked by that line segment from the set of visible points of v. The algorithm has $O(m \cdot n \cdot k \cdot \log n)$ time complexity and $O(nk^2)$ memory complexity. The algorithm is summarized in Theorem 1 and Algorithm 2.1. Also figure 2 shows the drawing of K_5 on a given point set produced by the proposed algorithm using software We3Graph [1].

Theorem 1 *Given a graph G with m edges, and n vertices, $V = \{v_1, v_2, \dots, v_n\}$, and a given set of n distinct points $P = \{p_1, p_2, \dots, p_n\}$ each with integer coordinates in three dimensions, G can be drawn crossing-free on P with v_i at p_i and with at most $O(\log n)$ bends per edge and in a $O(m+n+w) \times O(m+n+l) \times O(\log n + h)$ volume such that each bend has three dimensional integer coordinates. The drawing can be produced in $O(m \cdot n \cdot k \cdot \log n)$ time and $O(nk^2)$ memory.*

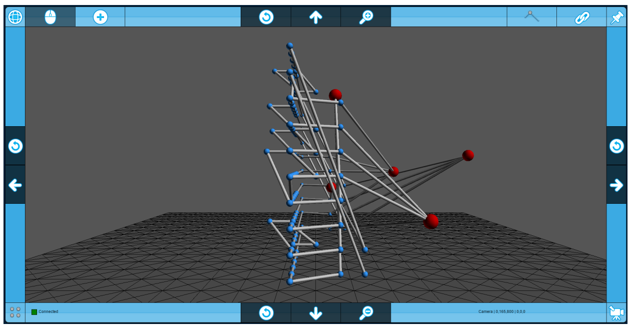


Figure 2: 3D drawing of K_5 on a given point set using the first proposed algorithm. Y axis upward and camera looking toward the negative side of Z axis direction.

3 The algorithm with one bend per edge

In this section an algorithm is given which will produce a drawing of size $O(m+n+w) \times O(m+n+l) \times O(h)$, with exacly one bend per edge. The algorithm considers a rectangle parallel to the XY plane in front of the bounding box of the points in the direction of the Z axis and for each edge, finds an integer bend point in that rectangle that is visible from both endpoints of the edge and connects the endpoints of the edge directly to the bend point. Here is a detailed explanation of the algorithm.

Let $k = \max(n, m)$. Let P_C denote the plane $z = h+1$ and R_C denote the rectangle going from $(1, 1, h+1)$ to $(4k, 4k, h+1)$ in the plane P_C. A point s is visible from point t if the line segment connecting s to t does not intersect any vertex of G or any line segment that is previously drawn. At the i^{th} step $(1 \le i \le m)$, the i^{th} edge e_i, connecting vertices u_i and w_i is considered. Now an integer bend point a_i is found in R_C that is visible both from u_i and w_i. A line segment α_i is drawn between u_i and a_i and a line segment β_i is drawn between w_i and a_i. Figure 3 shows a conceptual picture of R_C and the bounding box of the points.

Algorithm 2.1 The algorithm with logarithmic number of bends per edge

```
    $R_A$ denotes the rectangle going from $(1, 1, h+1)$ to $(2k, 2k, h+1)$ in the plane $z = h+1$
    $R_B$ denotes the rectangle going from $(1, 1, 0)$ to $(2k, 2k, 0)$ in the plane $z = 0$
 1: $\lambda = \max(h+2, \log m)$
 2: Let $S_v$ be the set of all visible integer points from the vertex $v$, in $R_A$.
 3: Let $\hat{S}_v$ be the set of all visible integer points from the vertex $v$, in $R_B$.
 4: for all vertex $v$ in $V$ do
 5:   for all vertex $v_2$ in $V$ - $v$ do
 6:     Remove the point in $S_v$ or the point in $\hat{S}_v$ that is blocked by $v_2$ from $v$ (if it exists).
 7: for all edge $e_i = (u_i, w_i)$ in $E$ do
 8:   Let $a_i$ be a point in $S_{u_i}$
 9:   Draw a line segment $\alpha_i$ from $u_i$ to $a_i$.
10:   for all vertex $v$ in $V$ do
11:     Remove every point in $S_v$ and $\hat{S}_v$ that is blocked by $\alpha_i$ from $v$.
12:   Let $b_i$ be a point in $\hat{S}_{w_i}$
13:   Draw a line segment $\beta_i$ from $w_i$ to $b_i$.
14:   for all vertex $v$ in $V$ do
15:     Remove every point in $S_v$ and $\hat{S}_v$ that is blocked by $\beta_i$ from $v$.
16: counter=1
17: for all $\alpha_i$ ordered by x coordinate and in case of a tie by y coordinate do
18:   Draw a line segment between $\alpha_i$ and the point $\bar{a}_i = (-1, counter, \lambda)$.
19:   counter++
20: counter=1
21: for all $\beta_i$ ordered by x coordinate and in case of a tie by y coordinate do
22:   Draw a line segment between $\beta_i$ and the bend point $\bar{b}_i = (-1, counter, -1)$.
23:   counter++
24: Use the technique of [12] for drawing a perfect matching in 3D to connect each $\bar{a}_i$ to $\bar{b}_i$.
```

To prove that there is always an integer bend point visible from both of u_i and w_i in R_C at the i^{th} step of the algorithm, consider that there are only two ways that an integer bend point in R_C becomes invisible from u_i or w_i:

1. A previously drawn line segment is between R_C and, u_i or w_i. The previously drawn line segment can be any of α_j or β_j for $1 \leq j < i$. There are at most $2k-2$ such line segments and each line segment can make at most $4k$ integer points of R_C invisible from u_i and at most $4k$ integer points of R_C invisible from w_i. So this case will make at most $2(2k-2)4k$ integer points of R_C invisible from either of u_i or w_i. To prove that each such line segment connecting vertices or bend points q and t, will make at most $4k$ integer points in R_C invisible from a vertex v which can be u_i or w_i, consider the plane P_{vqt} containing v, q and t. If the plane P_{vqt} intersects with the plane P_C the intersection will be a line. This line can contain at most $4k$ integer points of R_C. If v, q, and t are collinear, at most one integer point of R_C is made invisible.

2. A vertex is between R_C and, u_i or w_i: This can be any vertex other than u_i or w_i. Each such vertex can make at most one integer point of R_C invisible from u_i and at most one integer point of R_C invisible from w_i. There are at most $k-1$ such vertices. So this case can make at most $2(k-1)$ integer points of R_C invisible from either of u_i or w_i.

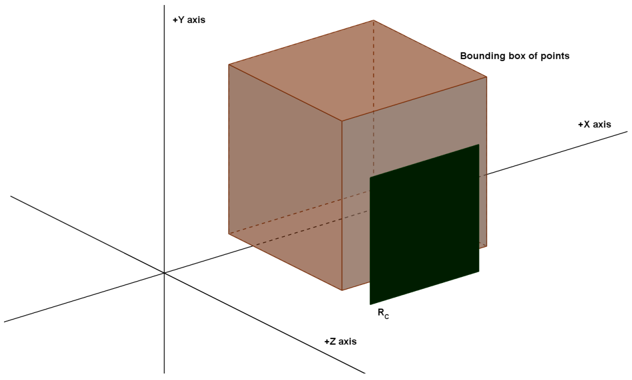


Figure 3: The conceptual picture of R_C and the bounding box of the points.

Subtracting the maximum number of integer points invisible from both u_i and w_i of both cases from the total number of integer points of R_C, leaves at least $14k+2$ visible points as shown in equation 2.

$$16k^2 - 2(2k-2)4k - 2(k-1) = 14k + 2 \qquad (2)$$

To find the visible points, for each vertex v, the algorithm maintains a set of integer points in R_C that are visible from v. The set is implemented using a 2D array. After adding each line segment at each step of the algorithm, for each vertex v, the algorithm removes the integer points blocked by that line segment from the set of visible points of v. The algorithm has $O(mk^2)$ time complexity and $O(nk^2)$ memory complexity. The algorithm is summarized in Theorem 2 and Algorithm 3.1. Also figure 4 shows the drawing of K_5 on a given point set using the proposed algorithm.

Theorem 2 *Given a graph G with m edges, and n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and a given set of n distinct* *points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, G can be drawn crossing-free on P with v_i at p_i and with exactly one bend per edge and in a $O(m+n+w) \times O(m+n+l) \times O(h)$ volume such that each bend has three dimensional integer coordinates. The drawing can be produced in $O(mk^2)$ time and $O(nk^2)$ memory.*

Algorithm 3.1 The algorithm with one bend per edge

```
    $R_C$ denotes the rectangle going from $(1,1,h+1)$ to $(4k,4k,h+1)$ in the plane $z = h+1$
 1: Let $S_v$ be the set of all visible integer points from the vertex $v$, in $R_C$.
 2: for all vertex $v$ in $V$ do
 3:   for all vertex $v_2$ in $V$ - $v$ do
 4:     Remove the point in $S_v$ that is blocked by $v_2$ from $v$ (if it exists).
 5: for all edge $e_i = (u_i, w_i)$ in $E$ do
 6:   Let $a_i$ be a point in both $S_{u_i}$ and $S_{w_i}$
 7:   Draw a line segment $\alpha_i$ from $u_i$ to $a_i$.
 8:   Draw a line segment $\beta_i$ from $w_i$ to $a_i$.
 9:   for all vertex $v$ in $V$ do
10:     Remove every point in $S_v$ that is blocked by $\alpha_i$ or $\beta_i$ from $v$.
```

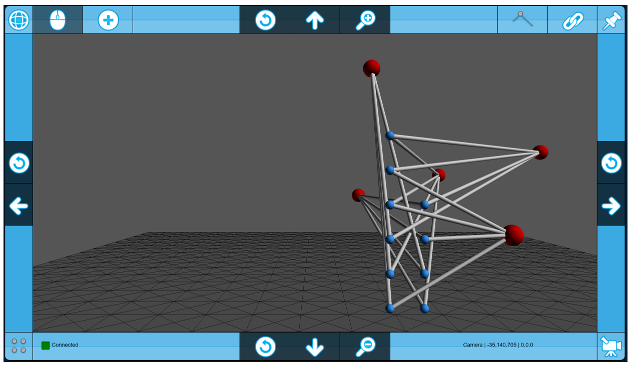


Figure 4: 3D drawing of K_5 on a given point set using the second algorithm. Y axis upward and camera looking toward the negative side of Z axis direction.

4 Comparison of the two algorithms

While the second algorithm, with lower number of bends per edge provides an equal or better asymptotic volume, the first algorithm with a better asymptotic running time for dense graphs, might result in lower exact volume since it uses two $2k \times 2k$ rectangles instead of one $4k \times 4k$ rectangle to find visible integer bend points.

5 Conclusions and open problems

Two algorithms were presented to answer a previously raised question in the 3D graph drawing literature. Although the algorithms run in polynomial time, improving the practical and asymptotic runtime performance should be considered. Also, although the algorithms produce drawings in 3D without crossings, edges can be very close, thus finding an algorithm which can guarantee a particular minimum distance between edges or vertices is another area which can be investigated.

References

[1] F. Barahimi. Web-based drawing software for graphs in 3d and two layout algorithms. Master's thesis, U. of Lethbridge, Dept. of Math. and Comp. Sci., 2015.

[2] P. Bose, J. Czyzowicz, P. Morin, and D. R. Wood. The maximum number of edges in a three-dimensional grid-drawing. *J. Graph Alg. & App.*, 8(1):21–26, 2004.

[3] S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is np-hard. *J. Graph Alg. & App.*, 10(2):353–363, 2006.

[4] T. Calamoneri and A. Sterbini. 3d straight-line grid drawing of 4-colorable graphs. *Information Processing Letters*, 63(2):97–102, 1997.

[5] R. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1997.

[6] G. Di Battista, F. Frati, and J. Pach. On the queue number of planar graphs. *SIAM J. Computing*, 42(6):2243–2285, 2013.

[7] V. Dujmović. Graph layouts via layered separators. *J. of Combinatorial Theory, Series B*, 110:79–89, 2015.

[8] V. Dujmović and D. Wood. Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Math. & Theoretical Computer Science*, 7(1), 2006.

[9] S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *J. Graph Alg. & App.*, 7(4):363–398, 2003.

[10] L. Heath and A. Rosenberg. Laying out graphs using queues. *SIAM J. Computing*, 21(5):927–958, 1992.

[11] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Alg. & App.*, 6(1):115–129, 2002.

[12] H. Meijer and S. Wismath. Point set embedding in 3d. *J. Graph Alg. & App.*, 19(1):243–257, 2015.

[13] J. Pach, T. Thiele, and G. Toth. Three-dimensional grid drawings of graphs. In G. Di Battista, editor, *Graph Drawing*, volume 1353 of *LNCS*, pages 47–51. Springer Berlin Heidelberg, 1997.

[14] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

[15] D. Wood. Three dimensional graph drawing with fixed vertices and one bend per edge. *arXiv: http://arxiv.org/abs/1606.09188*, 2016.

Boundary Labeling with Obstacles

Martin Fink[*] Subhash Suri[*]

Abstract

Boundary labeling is a map labeling technique in which annotations (text labels) are placed along the map boundary, rather than inside the map next to the target sites; each label is connected to its target site via a line, called the *leader*. When many sites in a small area require annotation, boundary labeling can significantly reduce clutter. In this paper, we explore the boundary labeling problem under the natural constraint that the leaders of different labels do not cross each other and they must also avoid a prescribed set of *obstacles* in the map—the obstacles may be predefined labels of high priority or important map features that should not be obscured by the leaders. We present dynamic-programming-based polynomial-time algorithms for optimizing labelings for different leader styles. We also show how to extend our algorithms to the case where areas rather than points are labeled; this is possible as long as the labeled areas are obstacles for all other leaders.

1 Introduction

We are given a rectangular map showing, e.g., part of a city including labels of streets and landmarks. As an overlay to this map we want to present points of interest (POIs) such as restaurants, shops, or museums to a user. As such sites are often close to each other it is not possible to simply place labels on the map due to the limited space. Hence, we place the labels on the boundary (but outside) of the map. We connect each label by a line (called *leader*) to its site and try to avoid crossings of leaders as well as intersections of leaders and labels on the map.

Problem Definition. Our input map is an axis-aligned rectangle R. We are further given a set P of points inside R, for each point $p \in P$ a label $l(p)$, and a set O of obstacles as rectangular polygons. We want to find a drawing in which each label is placed on the boundary outside rectangle R and connected by a leader to its point such that the leaders do not cross and no leader intersects an obstacle.

Different styles for the leaders can be used; see Fig. 1. Apart from straight-line segments, rectangular polylines

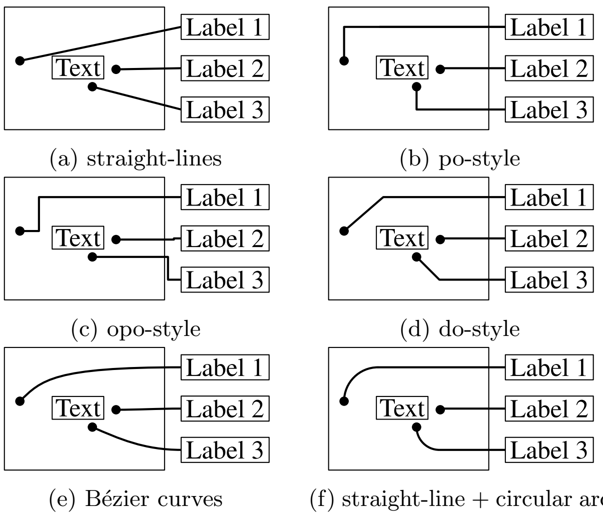


Figure 1: Different leader styles for boundary labeling.

with one (po-style) or two (opo-style) bends are common. Also leaders with a horizontal segment connected to the label and a diagonal segment connected to the point have been used (do-style). Furthermore, curved leaders are possible. A further distinction of problem variants is based on the placement of labels. While theoretically all four sides of rectangle R can be used for placing labels, the horizontal label text and the smaller height of the labels lead to a preference for using only the left and the right boundary for label placement.

Often, the positions of labels are further restricted by predefining a set of n (e.g., equally spaced) candidate positions for placing the labels on the boundary. Since in the presence of obstacles, this can easily lead to instances without a crossing-free solution, we do not follow this restriction. Instead, we allow labels to be freely distributed (without overlapping) along only the right (1-sided version) or both the left and the right side (2-sided version) of the boundary. We assume that each leader is connected to the center of its labels boundary.

Previous Work. Boundary Labeling has been introduced by Bekos et al. [4]. They considered straight-line and rectilinear leaders (po- and opo-styles) and tried to minimize total leader length or total number of bends. Later, also cases such as do-style leaders [2] or leaders connected to multiple points [1] were considered.

On the other hand, there is little work on treating

[*]Department of Computer Science, University of California, Santa Barbara, {fink|suri}@cs.ucsb.edu

Table 1: Runtimes for leader length minimization.

style	opo	po	do	straight
1-sided	$O(n^{11})$	$O(n^7)$	$O(n^8)$	$O(n^{11})$
2-sided	$O(n^{27})$	$O(n^{15})$	$O(n^{21})$	$O(n^{27})$

Table 2: Runtimes with fixed label positions.

style	opo	po	do	straight
1-sided	$O(n^8)$	$O(n^4)$	$O(n^5)$	$O(n^5)$
2-sided	$O(n^{21})$	$O(n^9)$	$O(n^{15})$	$O(n^{15})$

obstacles, which are normally just ignored. Bekos et al. [3] tried combining labels on the map with boundary labeling. They did, however, not distinguish different types of label that have to be placed internally or on the boundary; that is, intersection of leaders and labels can be avoided by moving an intersected internal label to the boundary. Benkert et al. [5] presented a polynomial-time algorithm for boundary labeling in the po- or do-style (see Fig. 1b and 1d, respectively) that allows to optimize quality measures for the leaders, e.g. the total leader length or the number of bends. Their approach is flexible enough to work with general quality measures for a single leader which could also take the underlying map into account, as the authors mention, although this has not been tried so far. As a starting point, this could also be used to forbid intersections with obstacles by making such leaders expensive; however, note that their work uses the restricted model in which the n candidate label positions are part of the input.

For a strongly restricted version of boundary labeling (po-leaders and congruent, axis-aligned rectangular obstacles), Löffler and Nöllenburg [6] developed an algorithm that labels the input points such that the minimum number of obstacles is intersected.

Our Results. While the presence of obstacles could potentially make boundary labeling hard, we show that boundary labeling with obstacles can be solved in polynomial time if all points' labels have the same height. This holds both in the case where all labels must be placed on the same side of the boundary (one-sided) and in the case where both the left and the right boundary can be used for placing labels (two-sided). For straight-line leaders, po-leaders, opo-leaders, and do-leaders, we can decide whether a feasible crossing-free solution that avoids all obstacles exists, both for the one-sided and the two-sided case. Similarly, we can integrate minimum distance constraints between leaders and between leaders and obstacles, and also minimize the total leader length.

Although the different leader styles can be handled similarly, the runtimes vary since the leaders have different complexity. The resulting runtimes for different leader styles are as shown in Table 1; the input complexity n is the total number of points and obstacle vertices. Opo-leaders naturally require a higher runtime than po- or do-leaders due to their higher complexity. Note that in contrast to previous works that use opo-leaders, we actually make use of the flexibility for placing the middle segment within the drawing region, which helps us route leaders around obstacles. Straight-line leaders require the same runtimes as opo-leaders. On the first look, this seems surprising since straight-line leaders are the least complex shapes. However, at closer analysis, it turns out that straight-line leaders make it necessary to consider a higher number of candidate y-coordinates for label placement, which increases the complexity. Our algorithms can also be applied for the case that n (or $O(n)$) candidate label positions are part of the input. In this case, the runtimes decrease; see Table 2.

We also considered the case where one wants to label (polygonal) regions instead of points. As long as each region also acts as an obstacle for all other leaders, we can still solve the different problem variants efficiently, yet with slightly increased runtimes.

On the other hand, boundary labeling with obstacles is NP-hard if labels can have different sizes, even if all labels must be placed on the same side of the boundary and even if there is only a single rectangular obstacle. Note that a similar hardness result holds for classic boundary labeling without obstacles, but only for the two-sided case, where two opposite sides of the boundary rectangle are used for placing labels.

2 One-Sided Boundary Labeling with Obstacles

We now show that one-sided boundary labeling with obstacles can solved in polynomial time using dynamic programming. We present our result in detail for the opo-leader style and then explain how to generalize it to other leader styles. We use the total leader length as the optimization criterion.

We first discretize the search space by showing that considering a polynomial set of x- and y-coordinates for label positions and leader segments suffices. We first assume that labels can come infinitesimally close without overlapping. Clearly, we have to take all x-coordinates of points as well as vertices of rectangular obstacles into account for the middle segments of leaders. Similarly, all y-coordinates of points and obstacle vertices are candidates for label positions. Additionally, in order to minimize the leader length, labels can be "stacked" above or below a leader at such a coordinate; see Fig. 2. This gives rise to up to $2n$ additional positions with distance h (the label height) between neighboring positions. We now show that this set of coordinates is sufficient.

Lemma 1 *Let X and Y' be the sets of x- and y-*

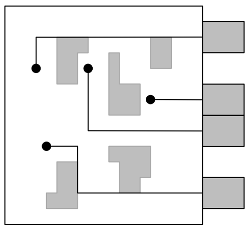


Figure 2: 1-sided labeling.

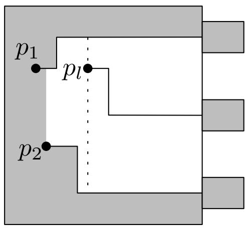


Figure 3: Split subinstance.

coordinates, respectively, of points and obstacle vertices; let $Y = \{y + k \cdot h \mid y \in Y', -n \leq k \leq n\}$. *If there is a crossing-free solution, then there is an optimal solution in which all* x*-coordinates of middle leader segments are in* X *and all* y*-coordinates of labels are in* Y.

Proof. We apply transformations to a given optimal solution that leave it crossing-free and do not increase the total leader length. First, we simultaneously move all middle leader segments to the left (which does not change the total leader length). We stop moving a segment if it reaches an x-coordinate in X or if it touches another middle segment; it is easy to see that the latter implies that we have reached a coordinate in X.

Now, we try to move the label positions while maintaining a crossing-free solution, so that the leader length is minimized. That means that a leader that is above its point is moved downwards and a leader that is below its point is moved upwards. We stop if we reach a y-coordinate of a point or obstacle vertex, or if we touch another label. If in the end there are coordinates not in Y, there must be some stacks of labels touching each other and none of the labels in each stack has the y-coordinate of a point or an obstacle vertex. If the stack has more labels that should be moved upwards or more that should be moved downwards, we move the whole stack in this direction, thus minimizing the leader length. On the other hand, if the number of labels that should be moved upwards and downwards are the same, we arbitrarily choose a direction. If during this process two stacks touch each other, we merge them and decide again in which direction to move the whole stack. Eventually, we reach a situation in which each stack has a label whose y-coordinate is the y-coordinate of a point or an obstacle vertex. Since the labels in the stacks touch, this means that all labels of the stack have a y-coordinate in Y. $\square$

From now on, we can focus on the restricted problem of finding an optimal solution with x- and y-coordinates in X and Y, respectively. Since $|X| = O(n)$ and $|Y| = O(n^2)$ and each leader can be described by its point, the x-coordinate of the middle segment, and the y-coordinate of the label, there are $O(n^4)$ potential leaders. In $O(n^5)$ total time, we can precompute for each of them the length and a possible intersection with an obstacle; in the latter case we set the length to $+\infty$. (Note that this precomputation could be sped-up to $O(n^4)$ total time).

The dynamic program is based on subinstances bounded by an upper leader and lower leader; see Fig. 3. All points in the area enclosed by upper and lower leader and lying right of both leaders' points must be connected to a label within the subinstance in any crossing-free solution. Hence, the optimum cost of labeling points within the subinstance is independent of anything outside. We try all possible leaders connecting the leftmost point p_l of the subinstance to its label; each such leader splits the subinstance into two new subinstances; see Fig. 3. Let the upper leader be the one connecting p_1 to a label at $y = y_1$ with x_1 as the x-coordinate of the middle segment; similarly, let the lower leader be described by p_2, y_2, and x_2. Let $\text{cost}[p_1, x_1, y_1, p_2, x_2, y_2]$ be the total leader length of an optimal solution for the subinstance, or $+\infty$ if no crossing-free solution exists. Let $\text{left}[p_1, x_1, y_1, p_2, x_2, y_2] = p_l$ be the leftmost point in a subinstance; if there is no point in the subinstance, we set $\text{left}[p_1, x_1, y_1, p_2, x_2, y_2] = \emptyset$ and $\text{cost}[p_1, x_1, y_1, p_2, x_2, y_2] = 0$. Then,

$$\begin{aligned}\text{cost}[p_1, x_1, y_1, p_2, x_2, y_2] = \min\big\{ &\text{cost}[p_1, x_1, y_1, p_l, x, y] \\ &+ \text{cost}[p_l, x, y, p_2, x_2, y_2] + c[p_1, x_1, y_1, p_2, x_2, y_2, x, y] \\ &\mid p_l = \text{left}[p_1, x_1, y_1, p_2, x_2, y_2], x \geq x(p_l), x \in X, \\ &y_1 \leq y \leq y_2, y \in Y\big\},\end{aligned}$$

where $c[p_1, x_1, y_1, p_2, x_2, y_2, x, y]$ is the cost of the leader from p_l to the label position at y-coordinate y with the vertical segment at x-coordinate x in the subinstance. This cost is $+\infty$ if the leader intersects with an obstacle or with the leaders of p_1 or p_2.

By placing dummy points and leaders above and below the instance, we create an entry in table cost that in the end contains the cost of an optimal solution. Instances with no unlabeled point get cost 0. For any other instance, there are $O(|X| \cdot |Y|) = O(n^3)$ possible leaders for p_l to be taken into account. Since leader length and intersection with obstacles were precomputed, we only have to check for intersections with p_1 and p_2's leaders, which takes only constant time. Hence, each entry is computed in $O(n^3)$ time. Since there are $O(n^8)$ entries, the total runtime is $O(n^{11})$.

Theorem 2 *We can compute a length-minimal one-sided opo-labeling with unit-height labels in* $O(n^{11})$ *time.*

Other Leader Styles. For other leader styles, we can use essentially the same dynamic program; we have to adjust the representation of possible leaders—with impact on the runtime—by developing new versions of Lemma 1; see Appendix A.

- po-leaders are opo-leaders with the middle segment fixed to the point's x-coordinate. For Lemma 1, we get the same set of $|Y| = O(n^2)$ possible label coordinates. This reduces both the complexity of the subinstances' (by a factor of $\Theta(n^2)$) and the number of possible leaders for p_l (by $\Theta(n)$). We can further decrease the complexity of a subinstance due to the following observation: Since the vertical segments are at the point's x-coordinate, the new leader of the leftmost point cannot interfere with previous vertical segments. It suffices to encode the y-coordinates of the labels and one of the leaders' points (the one more to the right) so that the leftmost point in the subinstance is specified. Hence, the total runtime is $O(n^7)$.

- do-leaders are very similar to po-leaders (being a generalization of them). The new version of Lemma 1 requires a slightly refined proof, but the result stays the same. We can also optimize do-leaders in $O(n^8)$ time. The above improvement to $O(n^7)$ does not generalize to do-leaders.
- Somewhat surprisingly, straight-line leaders require the same complexity as opo-leaders. The reason is that the number of possible y-coordinates of labels increases to $O(n^3)$: When minimizing the length of a single leader, we end up in a situation where the leader leaves the point and touches another point or an obstacle—or the leader is a horizontal segment. This gives rise to $O(n^2)$ possible label positions; stacks of touching labels increase this number to $O(n^3)$. Since this affects both the complexity of subinstances and the number of possible leaders for p_l, we get a total time complexity of $O(n^{11})$.

3 Two-Sided Boundary Labeling with Obstacles

The two-sided case is more complex than the one-sided case, but still can be solved using dynamic programming. Again, we focus on opo-leaders and then show how to extend the result to other leader styles.

First, we show that the sets X and Y of possible coordinates for middle segments of leaders are still sufficient.

Lemma 3 *Let X and Y' be the set of all x- and y-coordinates, respectively, of points and obstacle vertices (and 0); let $Y = \{y + k \cdot h \mid y \in Y', -n \leq k \leq n\}$. If there is a crossing-free solution, then there is an optimal solution in which all x-coordinates of middle segments are in X and all y-coordinates of labels are in Y.*

Proof. Again, we slightly modify a given an optimal solution—without increasing the total leader length—so that we get a new optimal solution using only the desired coordinates. Again, we start by moving all middle segments of leaders to the left; this time we stop moving also if we reach $x = 0$, i.e., the left boundary.

When moving labels in the second step, it can now happen that two segments adjacent to labels of different

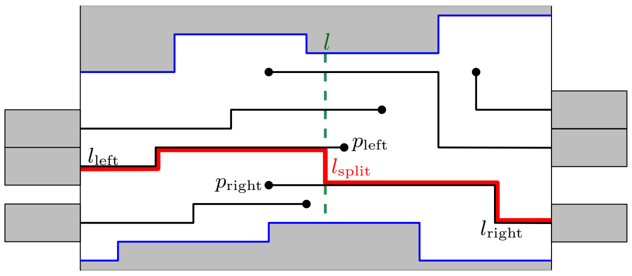


Figure 4: Splitting a 2-sided subinstance.

sides touch and can only be moved in the same direction. We extend stacks of labels that must be moved together to include this case. As before, we always move all labels of a stack in a direction that does not increase the total leader length, merging stacks if they touch. Eventually, we end up with a situation in which at least one label per stack has the y-coordinate of a point or an obstacle vertex; all other labels of the stack then have a y-coordinate in Y. □

We now must extend the dynamic program to take care of the two-sided case. We use the main idea of Benkert et al. [5] for the subdivision; however, notice that in the work of Benkert et al. the number and location of label positions was part of the input.

Consider a crossing-free solution for a two-sided instance. If we can find a vertical line l through the drawing region such that no leader of the instance crosses l, then we know that all points left of l must be connected to the left boundary and all points right of l must be connected to the right boundary. Hence, l separates the instance into two one-sided subinstances. On the other hand, if there is no such crossing-free l, there must be at least one vertical line l that crosses at least one leader connected to either boundary. Especially, there are two leaders whose crossings with l are next to each other and that are connected to labels on different boundaries; see Fig. 4. Assume that these are a leader l_{left} connecting point p_{left} to a label on the left boundary and leader l_{right} connecting p_{right} to a label on the right boundary. We split the instance by a polyline l_{split} as follows: First, l_{split} follows l_{left} until it intersects with l; then, l_{split} follows l down (or up) to the intersection with l_{right}; finally, l_{split} follows l_{right} to the right boundary.

Line l_{split} splits the subinstance horizontally into two (smaller) subinstances. Note that labels and leaders can overlap with l_{split}; however, this is only possible from one of the subinstances for each boundary, and the shape of l_{split} makes it clear for which side.

There are $O(n^4)$ possibilities for each of the leaders l_{left} and l_{right}. Furthermore, the line l is at an x-coordinate in X or between two consecutive elements of X. Hence, the complexity for describing l_{split} is $O(n^9)$. For each of the $O(n^{18})$ subinstances described by an upper line l^1_{split}

and a lower line l^2_{split}, we have to try $O(n^9)$ possibilities for l_{split} to split the subinstance into two subinstances. For any subinstance containing only a single point, we try to connect that point optimally (in $O(n^3)$ time since we precomputed the length and obstacle intersection of each possible leader).

Theorem 4 *We can compute a length-minimal two-sided opo-labeling with unit-height labels in $O(n^{27})$ time.*

Other Leader Styles. po-leaders are a special case of opo-leaders with lower complexity. The upper and lower line separating an instance have a complexity that is lower in the order of $O(n^4)$; there are fewer possibilities for subdiving by a splitting line, by the same factor. Hence, the total complexity for leader length minimization with po-leaders is $O(n^{15})$.

The diagonal segment of do-leaders can intersect with the vertical part of upper and lower bounding line. Hence, both have a complexity higher by a factor of $O(n^2)$ compared to po-leaders. The same holds for the splitting line. Therefore, the runtime for two-sided do-leaders is $O(n^{21})$.

Straight-line leaders, again need the same complexity as opo-leaders: There are $O(n^4)$ possible straight-line leaders, same as the number of possible opo-leaders. This carries over to bounding and splitting lines, whose complexity is $O(n^9)$ in both cases.

4 Area Labeling

Connected to labeling points in the presence of obstacles—usually representing areas of the map—is the generalization of labeling polygonal areas. If the areas that are to be labeled act as obstacles for all leaders except their own, then our dynamic program for the one-sided case can be adjusted. The main ideas are the same, only proving that a discrete set of coordinates suffices for representing optimal solutions becomes a bit different. Since a region acts as an obstacle for all leaders except its own the region is never split between two subinstances. However, if we allow leaders to cross regions, this splitting could happens, with the consequence that the subinstances would not be independent, thus making the dynamic program fail.

For the same reason, even if regions act as obstacles, a generalization of the two-sided algorithm fails at the first glance: The upper and lower bounding lines are not just leaders, but also contain a vertical segment connecting two leaders. Such a segment may very well intersect an area completely, without yielding any information on which part of the area will be connected to a label; the two parts lie in different subinstances, thus making it impossible to solve the subinstances independently. Fortunately, we can solve that problem: When splitting

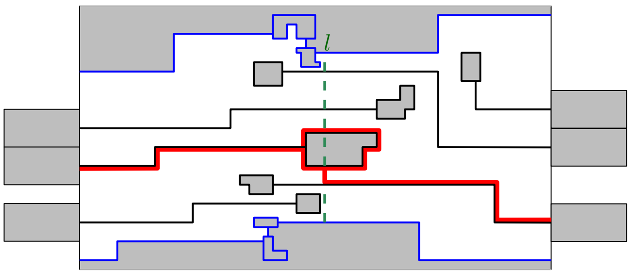


Figure 5: Splitting a 2-sided subinstance of labeled areas.

a solution with labels on both sides, we consider the regions as extensions of their leaders; see Fig. 5. Doing so, we find a vertical line l with two intersections with leaders—or their areas—that are connected to different sides. A different intersected region would break the adjacency between the defining leaders (or their areas); this means that the vertical segment cannot intersect with any area, and every area lies in a unique subinstance. The obvious—possible—exception are regions defining the separator; however, the separators' shape makes it clear which subinstance they belong to, making the subinstances independent.

Complexity. The runtime increases slightly for area labeling. The labeled area no longer fixes where the leader ends; that may be any point in the area. Hence, for describing leaders, we must specify the y-coordinate of the leader segment adjacent to the area for opo-leaders. This y-coordinate fixes also the area; it is the first area that is intersected by extending the segment. Therefore, it is no longer necessary to specify the area. For po-leaders, we now specify the x-coordinate of the vertical segment instead of the area; additionally, we indicate whether the segment connects to an area above or below the first leader segment.

By extending the analysis of Lemma 1 and Lemma 3—with similar movement operations for the respective segments—it is easy to show that it suffices to consider coordinates of area or obstacle vertices. In both cases, the complexity for describing a single leader—bounding a subinstance—stays the same; however, the number of possible leaders for connecting the leftmost area in the subinstance is increased by a factor of n. The new runtime is $O(n^{12})$ for opo-leaders and $O(n^8)$ for po-leaders in the one-sided cases. In the two-sided cases, the runtimes are still $O(n^{27})$ and $O(n^{15})$, respectively.

The case of do-leaders is essentially the same as po-leaders, just that instead of the vertical segment, the x-coordinate of the leader's bend and the direction of the diagonal (up or down) are used for describing the leader. Analogously to labeling points, both versions are a bit slower than for po-leaders; hence, we can solve area labeling with do-leaders in $O(n^9)$ time in the one-sided

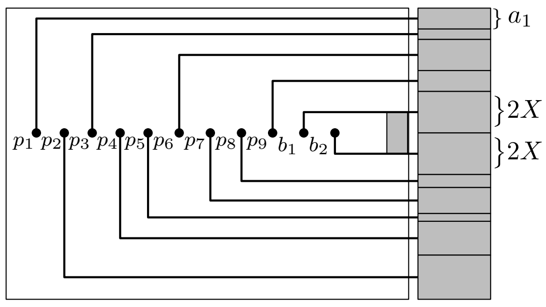


Figure 6: Reduction from PARTITION.

case and in $O(n^{21})$ time in the two-sided case.

For straight-line leaders, there are $O(n^3)$ possible y-coordinates for labels, determined by pairs of vertices of obstacles and areas. A leader is described by the label position and its slope—given by a vertex of the area or an obstacle; hence, there are $O(n^4)$ possible leaders. The runtime requirement is the same as for opo-leaders.

5 NP-Hardness for Non-Uniform Label Height

We now consider the case of *non-uniform* label heights. Even without obstacles, the two-sided version is known to be NP-hard for basically all leader styles by a simple reduction from the PARTITION problem [4]. We show that in the presence of obstacles—in fact, just a single obstacle—boundary labeling with non-uniform label heights is NP-hard even in the one-sided case. Again, this can be proved by a pretty straightforward reduction from PARTITION; we present the result for po-leaders, but other leader styles are hard by only slight modifications of our reduction.

Assume that we have an instance of PARTITION in the form of a set $S = \{a_1, a_2, \dots, a_n\} \subseteq \mathbb{N}$. Let $X = 1/2 \cdot \sum_{i=1}^{n} a_n$; the task is to decide whether there is a subset $A \subseteq S$ of the numbers such that $\sum_{a \in A} a = \sum_{b \in S \setminus A} b = X$. For each number a_i, we create a point p_i whose associated label height is a_i. The points are placed in the middle of drawing region, with ordered from left to right. The idea is to place an obstacle in the middle of the right boundary so that it is split into two gaps of height X. However, this does not take into account that of the labels placed right next to the obstacle, the upper or lower half can be behind the obstacle, with the effect that the gaps no longer have equal height. We work around this problem as follows; see Fig. 6 for an example. We add two points b_1 and b_2 with associated label height $2X$ each, and place an obstacle of height $2X$ in the middle of the right boundary, where the map has height $6X$. We claim that we can label all points if and only if there is a partition of the numbers.

First, assume that there is a crossing-free labeling of the points (with any leader style). Neither of the two gaps is large enough to accommodate the labels for both b_1 and b_2, these two labels must lie in different gaps, as shown in Fig. 6. As a result, there remains total height at most X in each gap. Since the total height of labels sums up to $2X$, all available vertical space must be used for labels, and the assignment of labels to the two gaps above and below the obstacle immediately gives rise to a feasible partition of P.

On the other hand, assume we have a set $A \subseteq P$ with $\sum_{a \in A} = \sum_{b \in P \setminus A} b = X$. We assign the label of each point corresponding to an element of A to the upper gap, and the label of each point corresponding to $P \setminus A$ to the lower gap; b_1 is assigned to the upper and b_2 to the lower gap. The labels of the upper gap are ordered from top to bottom according to the left-to-right order of their points, which already fixes their coordinates; similarly, the labels of the lower gap are ordered from top to bottom according to the order from right to left of their points; see Fig. 6. It is easy to see that this leads to a crossing-free labeling of all points for all leader styles (for opo-leaders, we simply us the restricted po-style).

Theorem 5 *One-sided boundary labeling with obstacles and non-uniform label height is NP-hard.*

References

[1] Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis. Many-to-one boundary labeling with backbones. *J. Graph Alg. Appl.*, 19(3):779–816, 2015.

[2] Michael A. Bekos, Michael Kaufmann, Martin Nöllenburg, and Antonios Symvonis. Boundary labeling with octilinear leaders. *Algorithmica*, 57(3):436–461, 2010.

[3] Michael A. Bekos, Michael Kaufmann, Dimitrios Papadopoulos, and Antonios Symvonis. Combining traditional map labeling with boundary labeling. In *SOFSEM'11*, volume 6543 of *LNCS*, pages 111–122. Springer, 2011.

[4] Michael A. Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Comput. Geom.*, 36(3):215–236, 2007.

[5] Marc Benkert, Herman J. Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria boundary labeling. *J. Graph Algorithms Appl.*, 13(3):289–317, 2009.

[6] Maarten Löffler and Martin Nöllenburg. Shooting bricks with orthogonal laser beams: A first step towards internal/external map labeling. In *CCCG'10*, pages 203–206, 2010.

Appendix

A One-Sided Boundary Labeling with Obstacles – Variants

Lemma 6 *Let Y' be the set of y-coordinates of points and obstacle vertices and let $Y = \{y + k \cdot h \mid y \in Y', -n \leq k \leq n\}$. If the instance has a crossing-free solution with po-leaders, then there also exists an optimal solution in which all y-coordinates of labels are elements of Y.*

Proof. We interpret the po-leaders as opo-leaders and apply the same transformation as in the proof of Lemma 1. Since the vertical segments are already on x-coordinates of points, they are not moved at all. The labels, however, end up at y-coordinates of Y. □

Lemma 7 *Let Y' be the set of y-coordinates of points and obstacle vertices and let $Y = \{y + k \cdot h \mid y \in Y', -n \leq k \leq n\}$. If the instance has a crossing-free solution with do-leaders, then there also exists an optimal solution in which all y-coordinates of labels are elements of Y.*

Proof. We assume we are given an optimal solution. We move the labels (and the horizontal leader segments) as we did in the proof of Lemma 1 so as to not increase the total leader length. Note that this movement implies shortening/lengthening the horizontal and diagonal segments accordingly.

We must stop moving a leader if its horizontal segment touches a point or a vertex of an obstacle, or if the label touches another label. In the former cases, we are done. In the latter case, we proceed as we did for opo- and po-leaders by building stacks of labels and moving them together. In the end, each stack will have a label at the y-coordinate of a point or an obstacle vertex, meaning that all labels in the stack are in Y. □

Lemma 8 *Let Y' be the set of all y-coordinates at which a straight-line defined by either two input points or by an input point and an obstacle vertex intersects the right boundary. Let $Y = \{y + k \cdot h \mid y \in Y' \text{ and } -n \leq k \leq n\}$. If the instance has a crossing-free solution with straight-line leaders, then there also exists an optimal solution in which all y-coordinates of labels are elements of Y.*

Proof. Again, we start with a given crossing-free solution and transform it by moving labels up and down so that the total leader length does not increase. If we cannot move a label because the leader would then intersect another leader or an obstacle, the leader's label must have a y-coordinate in $Y' \subseteq Y$. On the other hand, it can also happen that we cannot move a leader because its label touches another label. We then proceed by forming stacks of labels that are moved together. Eventually, each stack has at least one label with y-coordinate in Y', which means that all the labels' y-coordinates are in Y. □

On the Biplanar Crossing Number of K_n

Stephane Durocher[*] Ellen Gethner[†] Debajyoti Mondal[‡]

Abstract

The crossing number $cr(G)$ of a graph G is the minimum number of edge crossings over all drawings of G in the Euclidean plane. The k-planar crossing number $cr_k(G)$ of G is $\min\{cr(G_1) + cr(G_2) + \ldots + cr(G_k)\}$, where the minimum is taken over all possible decompositions of G into k subgraphs $G_1, G_2, \ldots, G_k$. The problem of computing the crossing number of complete graphs, $cr(K_n)$, exactly for small n and bounding its value for large n has been the subject of extensive recent research. In this paper we examine the biplanar crossing number of complete graphs, $cr_2(K_n)$. Since 1971, Owens' construction [IEEE Transactions on Circuit Theory, 18(2):277–280, 1971] has been the best known construction for biplanar drawings of K_n for large values of n. We propose an improved technique for constructing biplanar drawings of K_n, which reduces the lower order terms of Owens' upper bound. For small fixed n, we show that $cr_2(K_{10}) = 2$, $cr_2(K_{11}) \in \{4, 5, 6\}$, and for $n \geq 12$, we improve previous upper and lower bounds on $cr_2(K_n)$.

1 Introduction

A *drawing* of a graph G on $\mathbb{R}^2$ is a mapping of each vertex of G to a distinct point in $\mathbb{R}^2$ and each edge of G to a simple continuous curve between its corresponding endpoints. Throughout the paper we assume that the drawings are nice, i.e., the interiors of edges do not pass through vertices, edges may create crossings but do not touch otherwise, and finally, no three edges cross in a point. The *crossing number of* G is the smallest integer, denoted by $cr(G)$, such that G admits a drawing with $cr(G)$ edge crossings.

Determining the crossing numbers of complete graphs is one of the most studied problems in combinatorial geometry (e.g., [2, 4, 8, 15, 18, 19]). The problem of determining $cr(K_n)$, i.e., the crossing number of a complete graph with n vertices, has been studied since the early 1960s [11, 12, 21]. From that time it was known [11] that $cr(K_n)$ is bounded from above by Z_n, where $Z_n = \frac{1}{4}\left\lfloor\frac{n}{2}\right\rfloor\left\lfloor\frac{n-1}{2}\right\rfloor\left\lfloor\frac{n-2}{2}\right\rfloor\left\lfloor\frac{n-3}{2}\right\rfloor$. Given a complete graph of n vertices, there are several construction techniques [9] to produce a drawing of the graph with exactly Z_n crossings. In fact, it is conjectured that the equality $cr(K_n) = Z_n$ holds in general [12, 21]. Pan and Richter [17] showed that the conjecture holds for the case when $n \leq 12$.

The definition of crossing number naturally extends to an arbitrary number of planes. Given a graph $G = (V, E)$, the *k-planar crossing number $cr_k(G)$* of G is equal to $\min\{cr(G_1) + cr(G_2) + \ldots + cr(G_k)\}$, where the minimum is taken over all possible decompositions of G into k subgraphs $G_i = (V_i, E_i), 1 \leq i \leq k$, such that $V = \{V_1 \cup \ldots \cup V_k\}$ and $E = \{E_1 \cup \ldots \cup E_k\}$. In 1971, Owens [16] showed that $cr_2(K_n)$ is bounded from above by W_n, where $W_n =$

$$\begin{cases} Z_{\lceil n/2\rceil} + Z_{\lfloor n/2\rfloor} + \frac{n^2(n-4)(n-8)}{384}, & \text{if } n = 4m. \\ Z_{\lceil n/2\rceil} + Z_{\lfloor n/2\rfloor} + \frac{(n-1)(n-3)^2(n-5)}{384}, & \text{if } n = 4m+1. \\ Z_{\lceil n/2\rceil} + Z_{\lfloor n/2\rfloor} + \frac{n(n-2)(n-4)(n-6)}{384}, & \text{if } n = 4m+2. \\ Z_{\lceil n/2\rceil} + Z_{\lfloor n/2\rfloor} + \frac{(n+1)(n-3)^2(n-7)}{384}, & \text{if } n = 4m+3. \end{cases}$$

A rich body of research examines the asymptotic behaviour of the k-planar crossing numbers of complete and complete bipartite graphs [3, 20], and there have also been significant efforts to determine tight bounds on biplanar crossing numbers for these classes of graphs [6, 7, 10]. While tight bounds for $cr(K_n)$ are known for $n \leq 12$ [11, 17], the value of $cr_2(K_n)$ is known only when $n \leq 9$, i.e., $cr_2(K_n) = 0$ if $n < 9$, and $cr_2(K_9) = 1$ [14]. In a survey on the biplanar crossing number, Czabarka et al. [6] posed an open question that asks to determine $cr_2(K_n)$ when n is small.

A 1-*page drawing* Γ of G is a drawing of G on the Euclidean plane such that all the vertices lie on a circle C in Γ and the edges that do not belong to the boundary of C lie interior to C. The 1-*page crossing number* $\nu(G)$ of G is the minimum number of crossings over all the 1-page drawings of G. The *k-page crossing number* $\nu_k(G)$ of G is $\min\{\nu(G_1)+\nu(G_2)+\ldots+\nu(G_k)\}$, where the minimum is taken over all possible decompositions of G into k subgraphs $G_1, \ldots, G_k$, and the order of vertices along C is the same for all these subgraphs. Observe that a 2-page drawing of K_n with t crossings determines a drawing of

[*]Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada, durocher@cs.umanitoba.ca. Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[†]Department of Computer Science, University of Colorado Denver, Denver, USA, ellen.gethner@ucdenver.edu. Work of the author is supported in part by a Simons Foundation Collaboration Grant for Mathematicians.

[‡]Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada, jyoti@cs.umanitoba.ca.

K_n into a single plane with exactly t crossings[1]. Recall that the currently best known upper bound on $cr(K_n)$ is Z_n. Several studies proved that $\nu_2(K_n) = Z_n$ for different values of n [5, 8, 9], and very recently Ábrego et al. [1] proved the equality for every $n \in \mathbb{Z}^+$. However, it is still unknown whether $cr(K_n)$ is strictly smaller than $\nu_2(K_n)$, i.e., we only know that $cr(K_n) \le \nu_2(K_n) = Z_n$.

An analogous relationship between the k-planar crossing number and $2k$-page drawings of K_n is $cr_k(K_n) \le \nu_{2k}(K_n)$. Interestingly, we observe that W_n, which is the best known upper bound on $cr_2(K_n)$ for large values of n, is equal to the best known upper bound [9] on $\nu_4(K_n)$, when $n = 4m$ for some $m \in \mathbb{Z}^+$; see Section 2. However, the equality does not hold in general since $cr_2(K_9) = 1 < \nu_4(K_9) = 3$ [9].

In this paper we propose an improved technique for constructing biplanar drawings, which reduces Owens' [16] upper bound on $cr_2(K_n)$. Although the improvement is obtained by a slight modification of the Owens' construction, this is interesting since no such perturbation is known that can improve the conjectured value of $cr(K_n)$. For small fixed n, we show that $cr_2(K_{10}) = 2$, $cr_2(K_{11}) \in \{4, 5, 6\}$, and for $n \ge 12$, we improve previous upper and lower bounds on $cr_2(K_n)$.

2 Technical Details

De Klerk et al. [9] gave a generalized construction for k-page drawings of complete graphs. For some cases, e.g., when $n = 4m$ and $m \in \mathbb{Z}^+$, their upper bound on 4-page crossing number (thus the biplanar crossing number) of K_n, matches exactly the upper bound obtained by Owens [16] for biplanar drawings of complete graphs. We first briefly recall the construction given by Owens [16], and then the construction given by de Klerk et al. [9].

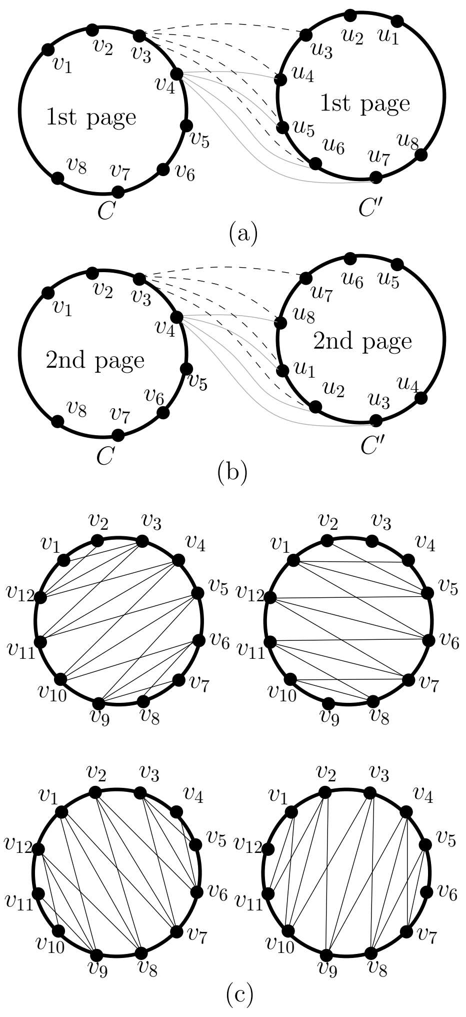


Figure 1: (a)–(b) Owens' [16] Construction. (c) De Klerk et al.'s [9] Construction.

2.1 Owens' [16] Construction

Given a complete graph K_n (assume for convenience that $n = 4m$, where $m \in \mathbb{Z}^+$), in each plane Owens constructed two vertex disjoint cycles $C = (v_1, \ldots, v_{n/2})$ and $C' = (u_1, \ldots, u_{n/2})$, each with $n/2$ vertices. He constructed the complete graph induced by the vertices on C using a 2-page drawing of $K_{n/2}$, i.e., placing the edges of the ith page, $i \in \{1, 2\}$, interior to the cycle C in the ith plane. The complete graph induced by the vertices on C' was constructed exterior to C' in a similar way. The remaining edges that form a complete bipartite graph $K_{n/2,n/2}$ connecting the vertices of C with the vertices of C', were drawn as follows: for each v_j on C, the first plane contains the edges from v_j to $n/4$ consecutive vertices on C' starting at u_j in clockwise order. The remaining edges of $K_{n/2,n/2}$ are drawn in the second plane symmetrically. Figures 1 (a)–(b) illustrate such a construction for K_{16}.

2.2 De Klerk et al.'s [9] Construction

De Klerk et al. [9] showed that for complete graphs K_n, where $n = km$ with $m, k \in \mathbb{Z}^+$, the k-page crossing number of K_n is $\nu_k(K_n) = \frac{1}{12k^2}\left(1 - \frac{1}{2k}\right)n^4 - \frac{1}{4k}n^3 + \left(\frac{7}{24k} + \frac{1}{6}\right)n^2 - \frac{1}{4}n$. We can observe that this is equal to the Owens' [16] upper bound when $k = 4$, as follows. Since $n = 4m$, we may assume $n = 2q$ with $q = 2m$. Then we have

[1]Imagine the drawing on a sphere, where the first page is drawn on the upper hemisphere, and the second page is drawn on the lower hemisphere.

$$\begin{aligned} Z_q &= \frac{1}{4}\left\lfloor\frac{q}{2}\right\rfloor\left\lfloor\frac{q-1}{2}\right\rfloor\left\lfloor\frac{q-2}{2}\right\rfloor\left\lfloor\frac{q-3}{2}\right\rfloor \\ &= \frac{1}{4}\left(\frac{q}{2}\right)\left(\frac{q}{2}-1\right)\left(\frac{q}{2}-1\right)\left(\frac{q}{2}-2\right) \\ &= \frac{1}{1024}q(q-4)^2(q-8). \end{aligned}$$

From Owens' [16] upper bound, we have

$$\begin{aligned} W_n &= Z_{\lceil q\rceil} + Z_{\lfloor q\rfloor} + \frac{n^2(n-4)(n-8)}{384} \\ &= \frac{7}{1536}n^4 - \frac{1}{16}n^3 + \frac{23}{96}n^2 - \frac{1}{4}n \\ &= \nu_4(K_n). \end{aligned}$$

To construct the k-page drawing, let the vertices of K_n be $v_1, \ldots, v_n$, and let M_i be the set of edges $\{(v_a, v_b) : 1 \le a, b \le n, \text{ and } i = (a+b-2) \mod n\}$. Now draw the edges $M_{(j-1)n/k} \cup \ldots \cup M_{jn/k-1}$ in the jth page. Figure 1 (c) illustrates the construction for K_{12} on 4 pages. Pairing the k pages and placing them in each side of a circle yields a $\lceil k/2\rceil$-planar drawing, which implies that $cr_k(K_n) \le \nu_{2k}(K_n)$.

3 Biplanar Crossing Number for Small Values of n

In this section we establish some tight bounds on the biplanar crossing number of K_n when n is small. It has been known for a long time that $cr_2(K_n) = 0$ if $n < 9$, and $cr_2(K_9) = 1$ [16]. We may thus assume that $n > 9$. We first prove that $cr_2(K_{10}) = 2$ and $cr_2(K_{11}) \in \{4, 5, 6\}$, and then provide a technique to compute good upper bounds on $cr_2(K_n)$, when $n > 9$.

Biplanar Crossing Numbers of K_{10} and K_{11}. We construct biplanar drawings of K_{10} and K_{11} with exactly 2 and 6 edge crossings, respectively, as shown in Figure 2. We now show that 2 and 4 edge crossings are necessary for K_{10} and K_{11}, respectively. Suppose for a contradiction that K_{10} admits a biplanar drawing with fewer than two edge crossings, and let Γ be such a biplanar drawing. Since K_{10} contains K_9 as a subgraph, Γ must contain exactly one edge crossing. Let (u, v) be an edge on Γ that is involved in this crossing. Then the deletion of v and its incident edges from Γ would give a biplanar drawing of K_9 without any edge crossing, which contradicts that $cr_2(K_9) = 1$.

For $cr_2(K_{11})$, we prove a lower bound of 4 as follows: Let Γ be an optimal biplanar drawing with at most 3 crossings. Observe that Γ must have at least 3 crossings, otherwise we can delete some vertex which is incident to some crossing in Γ to obtain a biplanar drawing of K_{10} with at most one edge crossing. Observe that no vertex v in Γ can be adjacent to two or more edge crossings, because otherwise deletion of v from Γ would yield a biplanar drawing of K_{10} with at most 1 crossing, which contradicts that $cr_2(K_{10}) = 2$. Since every crossing involves four distinct vertices and every vertex in Γ is incident to at most one crossing, Γ must have at least 12 distinct vertices, which is a contradiction.

Biplanar Crossing Numbers of K_n, where $n \ge 12$. Let Γ be a biplanar drawing of K_n. Observe that one can construct a biplanar drawing of K_{n+1} by executing the following steps:

S_1. Pick a vertex v in Γ and create a copy v' of v in each of the two layers of Γ.

S_2. In each layer of Γ, place v' arbitrarily close to v and add the edge (v, v') so that this edge does not introduce any new crossing.

S_3. Let $W = \{w_1, w_2, \ldots, w_{\lfloor d_v^i/2\rfloor}\}$ be the first half of the neighbors of v in clockwise order in the ith layer of Γ, where d_v^i denotes the degree of v in the ith layer. For each $w \in W$, we add the edge (v', w) closely following the edge (v, w) such that v' appears after v while examining the neighbors of w in clockwise order. The edges from v' to the remaining neighbors $\{w_{\lfloor d_v^i/2\rfloor+1}, \ldots, w_{d_v^i}\}$ of v are added symmetrically.

S_4. Remove the edge (v, v') from the second layer.

Let the resulting drawing be Γ'. It is straightforward to verify that the number of newly created crossings among the edges incident to v and v' is exactly $\sum_{i\in\{1,2\}}\left(\frac{\lfloor d_v^i/2\rfloor(\lfloor d_v^i/2\rfloor-1)+(\lceil d_v^i/2\rceil-1)\lceil d_v^i/2\rceil}{2}\right)$. Moreover, a crossing between two edges (v, w) and (x, y), where $v \notin \{x, y\}$, corresponds to a crossing between (v', w) and (x, y). Therefore, if v is adjacent to c_v^i crossings in the ith layer, then the number of crossings in Γ' is $\sum_{i\in\{1,2\}}\left(\frac{\lfloor d_v^i/2\rfloor(\lfloor d_v^i/2\rfloor-1)+(\lceil d_v^i/2\rceil-1)\lceil d_v^i/2\rceil}{2} + c_v^i\right)$ more than the number of crossings in Γ.

To obtain better drawings, we choose the vertex v that minimizes the number of newly introduced crossings (break ties arbitrarily). Table 1 shows the number of crossings obtained by the above construction technique, when $n \in \{12, 13, \ldots, 30\}$, and the lower bounds using the inequality $cr_2(K_n) \ge \frac{cr_2(K_{n-1})\binom{n}{4}}{\binom{n-1}{4}}$, which is widely used to establish lower bounds on crossing number [7]. Note that the upper bounds of Table 1 are significantly smaller than the values $18, 37, 53, 75, 100, 152$, for $n = 12, \ldots, 17$, obtained by Owens' construction.

4 Upper Bounds on $cr_2(K_n)$

Assume that $n = 8m+4$, where $m \in \mathbb{Z}^+$. We begin with the construction of Owens [16], and later we modify the drawing to improve the number of crossings. We use a

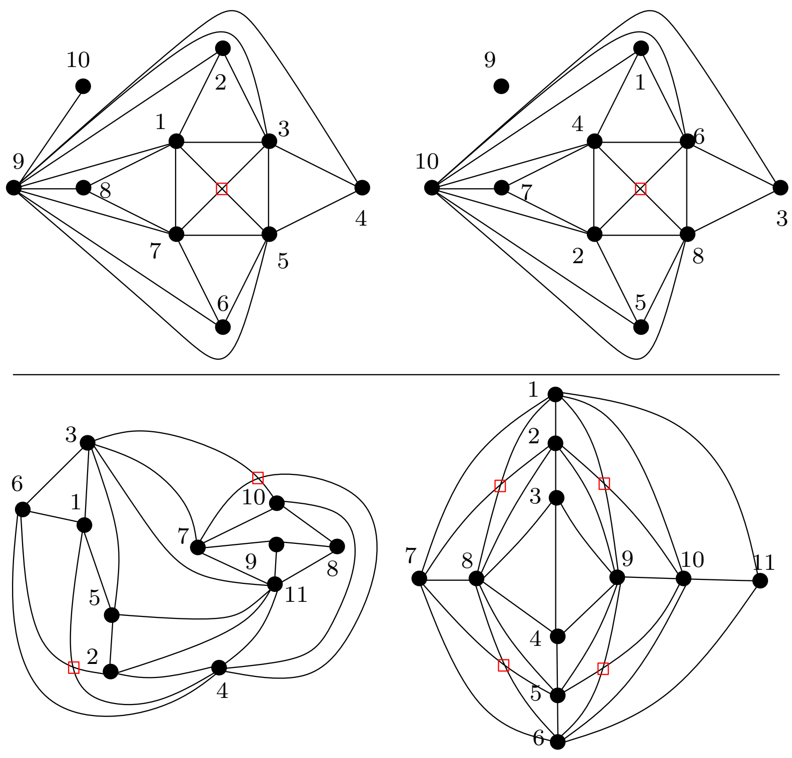


Figure 2: Biplanar drawings of K_{10} and K_{11} with two and six edge crossings, respectively.

Table 1: Upper and lower bounds on $cr_2(K_n)$, where $n \in \{12, 13, \dots, 30\}$.

n	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
$U.B.$	14	26	43	62	81	103	148	176	226	332	469	652	717	958	1261	1399
$L.B.$	6	9	13	19	26	35	46	60	76	95	118	145	176	212	253	299

slightly different presentation for Owens' [16] construction, which will be more convenient for the subsequent description.

4.1 Basic Construction

Let the planar layers of the drawing be $\mathcal{L}_j$, where $j \in \{1, 2\}$. In layer $\mathcal{L}_j$, we arrange the vertices into two circles: C^j_{in} and C^j_{out}, where each of them contains $n/2$ vertices. We then embed the cycle C^j_{in} interior to the cycle C^j_{out} such that the resulting embedding of the cycles remains crossing free, as shown in Figures 3(a)–(b). We now draw the edges that connects the vertices of C^j_{in} and C^j_{out}.

In $\mathcal{L}_1$, let the vertices on C^1_{in} be $v_1, v_2, \dots, v_{4m+2}$ and the vertices on C^1_{out} be $u_1, u_2, \dots, u_{4m+2}$ in clockwise order. For each $j \in \{1, 2, \dots, 4m+2\}$, connect u_j to the vertices $v_{j-m}, \dots, v_j, \dots, v_{j+m}$. Note that the indices wrap around, i.e., for any $v_{j'}$, if $j' < 1$ (respectively, $j' > n/2$), then $v_{j'} = v_{n/2+j'}$ (respectively, $v_{j'} = v_{j'-n/2}$). In the other planar layer $\mathcal{L}_2$, let the vertices on C^2_{in} be $u_1, u_2, \dots, u_{2m+1}$ and the vertices on C^2_{out} be $v_1, v_2, \dots, v_{2m+1}$ in clockwise order. For each $j \in \{1, 2, \dots, 4m+2\}$, connect v_j to those vertices of C^2_{in} that are not incident to v_j in $\mathcal{L}_1$. As illustrated in Figures 3(a)–(b), all these edges lie in the closed region between C^j_{in} and C^j_{out}.

Note that we may now complete the drawing of K_n by adding the edges among $\{u_1, \dots, u_{n/2}\}$ and the edges among $\{v_1, \dots, v_{n/2}\}$. For the set $\{v_1, \dots, v_{n/2}\}$, we construct a 2-page drawing of $K_{n/2}$, where the edges of one page lie inside C^1_{in} and the edges of the other page lie outside of C^2_{out}. Similarly, for the set $\{u_1, \dots, u_{n/2}\}$, we construct a 2-page drawing of $K_{n/2}$, where the edges of one page lie inside C^2_{in} and the edges of the other page lie outside of C^1_{out}. Let the resulting drawing be Γ. Since this construction is equivalent to that of Owens [16], the number of crossings in Γ is W_n.

4.2 Improvement

We now modify the drawing Γ to obtain a biplanar drawing with fewer crossings, as illustrated in Figures 3(c)–(d).

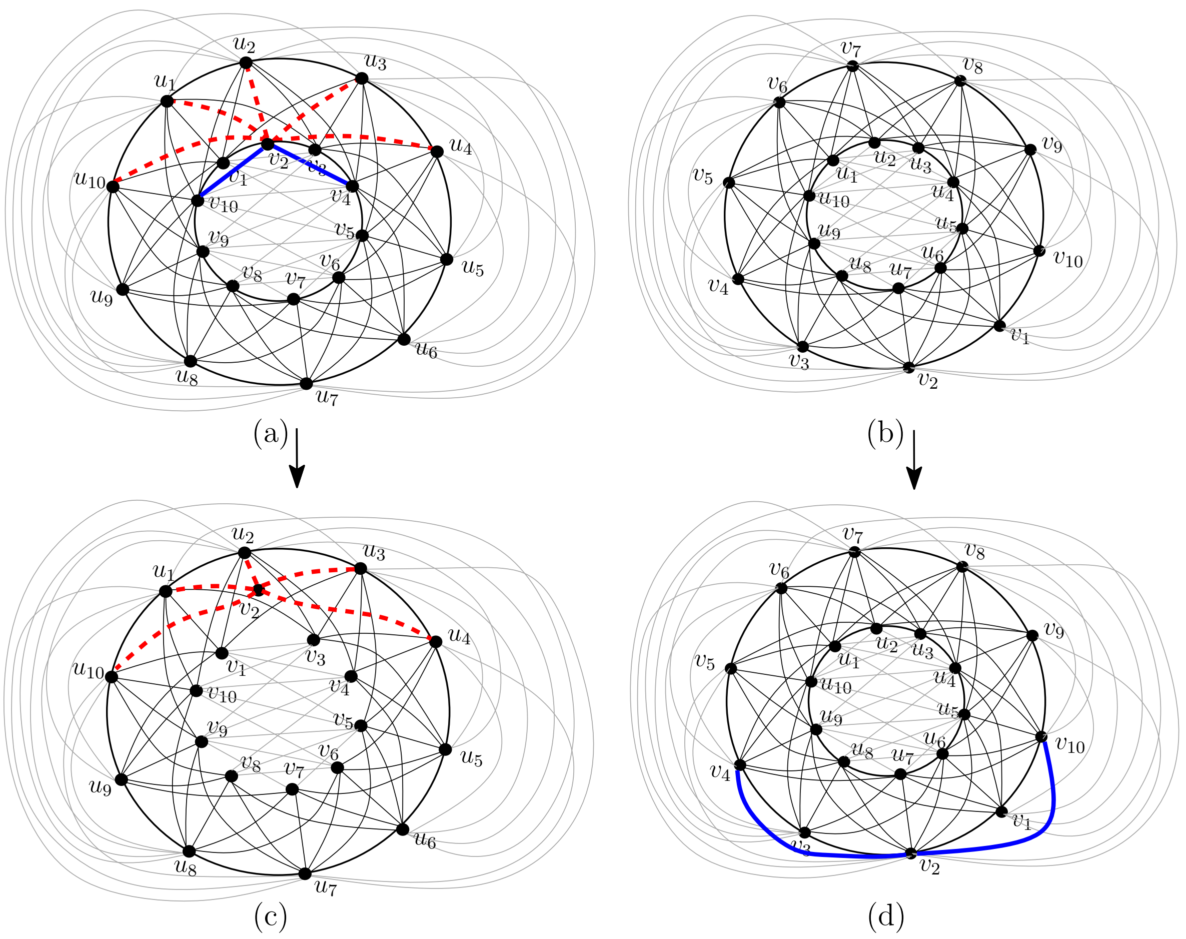

(a)
(b)
(c)
(d)
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 u_{10}
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10}

Figure 3: (a)–(b) Basic construction with 324 crossings. (c)–(d) Modification of Γ with 322 crossings. The blue and red edges are shown in bold and dashed lines, respectively.

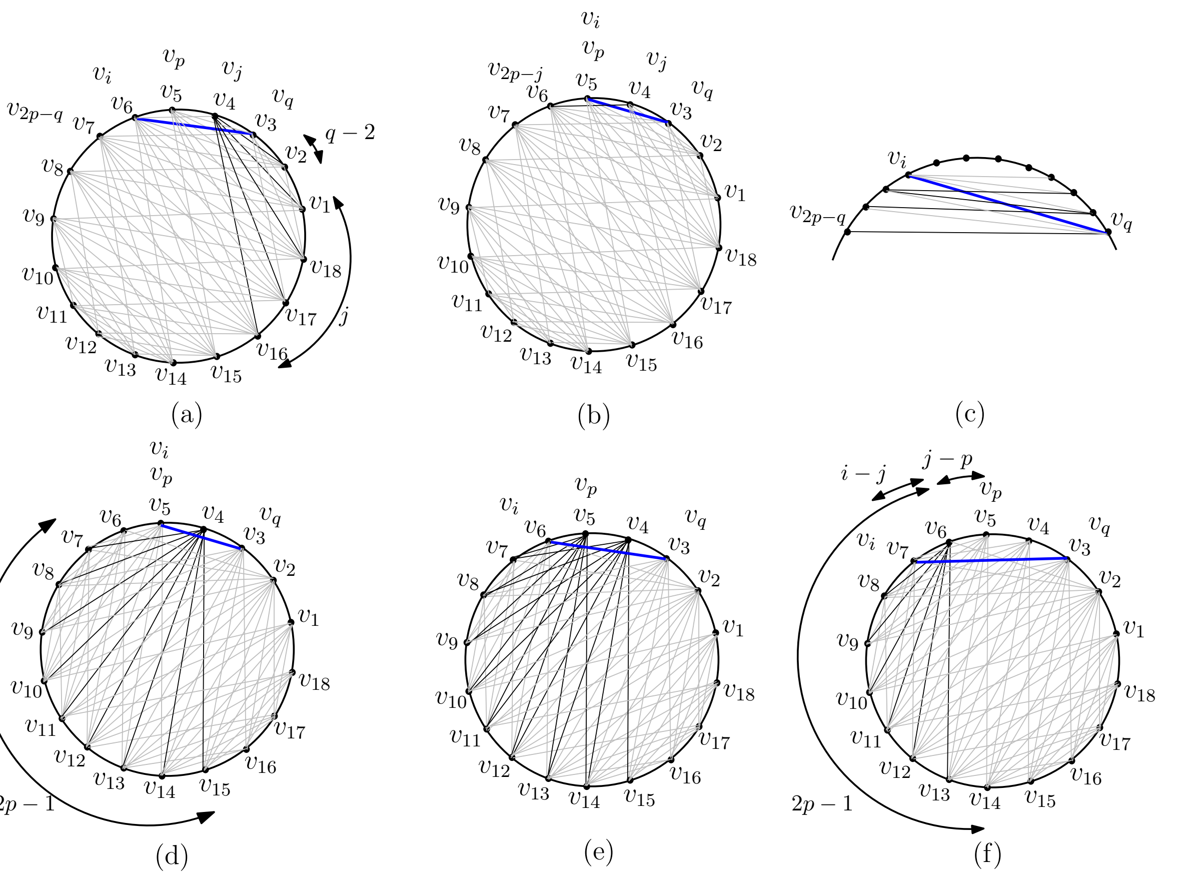
We first delete the incident edges of v_2 that lie inside C^1_{in}, and then add these edges outside of C^2_{out}, as illustrated in thick lines (blue) in Figure 3. We then remove the edges that lie on the boundary of C^1_{in}, and finally, move the vertex v_2 infinitesimally close to u_2 inside the cycle u_2, v_1, v_3, as shown in dashed lines (red) in Figure 3. Let the resulting drawing be Γ', which has smaller number of crossings than Γ. We now show how to modify the drawing for larger values of n.

Let $n = 16m + 4$, $n' = n/2$, $p = \lfloor n'/4 \rfloor + 1$ and $q = \lceil p/2 \rceil$. We now choose v_q to carry out the modifications, note that for $n = 20$, we have $v_q = v_2$. Let the edges incident to v_q that lie inside C^1_{in} in Γ but moved outside of C^2_{out} in Γ', be the blue edges. Denote the incident edges of v_q that lie outside of C^1_{in} in Γ as the red edges. Let the number of edge crossings on the blue edges in Γ and Γ' be α and α', respectively. Similarly, let the number of edge crossings on the red edges in Γ and Γ' be β and β', respectively. Then the number of edge crossings in Γ' is $W_n + (\alpha' + \beta') - (\alpha + \beta)$. We now briefly describe the computation of $\alpha, \alpha', \beta, \beta'$.

Crossings on the Blue Edges in Γ (i.e., α): We partition edge crossings into the following three types.

- A denotes the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{q+2, \ldots, 2p-q\}$, $x \in \{v_{q+1}, \ldots, v_{w-1}\}$, and $y \in \{v_{2p}, \ldots, v_{n'}\}$. Therefore, $A = \sum_{i=q+2}^{2p-q} \sum_{j=q+1}^{i-1} (j + (q-2))$, as shown in Figure 4(a).
- B denotes the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{q+2, \ldots, p\}$, $x \in \{v_{q+1}, \ldots, v_{w-1}\}$, and $y \in \{v_{w+1}, \ldots, v_{2p}\}$. Therefore, $B = \sum_{i=q+2}^{p} \sum_{j=q+1}^{i-1} ((2p-j) - i)$, as shown in Figure 4(b).
- C denotes the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{p+1, \ldots, 2p-q\}$, $x \in \{v_{q+1}, \ldots, v_p\}$, and $y \in \{v_{w+1}, \ldots, v_{2p}\}$. Therefore, $C = \sum_{i=p+1}^{2p-q} \frac{(2p-q-i-1)(2p-q-i)}{2}$, as shown in Figure 4(c).

The drawing is symmetric with respect to the axis through v_q and its diametrically opposite vertex. Thus



v_i v_p v_j v_q v_{2p-q} $q-2$ j
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12} v_{13} v_{14} v_{15} v_{16} v_{17} v_{18}
(a)
v_{2p-j}
(b)
(c)
$2p-1$
(d)
(e)
$i-j$ $j-p$
(f)

Figure 4: Computation with respect to v_q, where $n = 36$. (a)–(c) Computation of α. (d)–(f) Computation of α'.

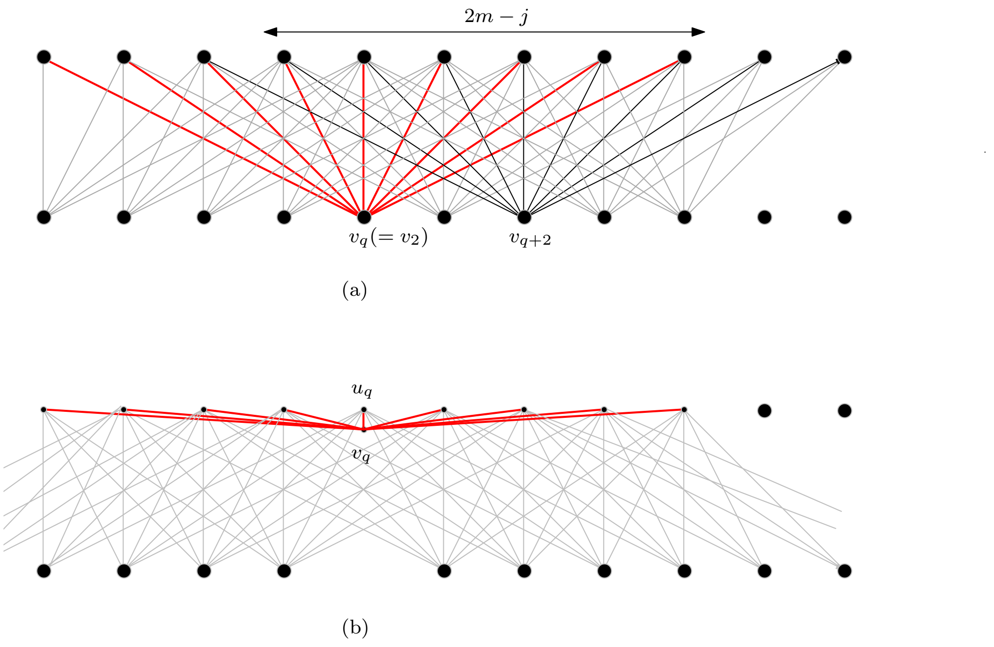


Figure 5: Crossings on the red edges: (a) Γ, and (b) Γ'.

the number of crossings removed from Γ by moving the blue edges from the inner layer is exactly $\alpha = 2(A+B+C)$.

Crossings on the Blue Edges in Γ' ***(i.e.,*** α'***)***: We partition these edge crossings into the following three types.

- A' denotes the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{q+2, \ldots, p\}$, $x \in \{v_{q+1}, \ldots, v_{w-1}\}$, and $y \in \{v_{w+1}, \ldots, v_{n'}\}$. Therefore, $A' = \sum_{i=q+2}^{p} \sum_{j=q+1}^{i-1} 2p-1$, as shown in Figure 4(d).

- B' is an upper bound on the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{p+1, \ldots, 2p-q\}$, $x \in \{v_{q+1}, \ldots, v_p\}$, and $y \in \{v_{w+1}, \ldots, v_{n'}\}$. Therefore, $B' = \sum_{i=p+1}^{2p-q} \left((p-q)(2p-1) - \frac{(i-p)(i-p+1)}{2}\right)$, as shown in Figure 4(e).

- C' denotes the number of crossings between the edges (v_q, v_w) and (x, y), where $w \in \{p+2, \ldots, 2p-q\}$, $x \in \{v_{p+1}, \ldots, v_{w-1}\}$, and $y \in \{v_{w+1}, \ldots, v_{n'}\}$. Therefore, $C' = \sum_{i=p+2}^{2p-q} \sum_{j=p+1}^{i-1} ((2p-1) - 2(j-p) - (i-j))$, as shown in Figure 4(f).

The drawing is symmetric with respect to the axis through v_q and its diametrically opposite vertex. Hence the number of crossings introduced in Γ' by moving the blue edges to the outer layer is at most $\alpha' = 2(A' + B' + C')$.

Crossings on the Red Edges in Γ ***(i.e.,*** β***)***: The number of crossings created by the edges (v_q, u') and (v_{q+j}, u''), where $1 \le j \le 2m-1$ and u', u'' lie on C_{out}^1, is $(2m-j)(2m-j+1)/2$. Figure 5(a) illustrates a scenario where $m = 4$. Symmetrically, the number of crossings created by the edges (v_q, u') and (v_{q-j}, u'') is $(2m-j)(2m-j+1)/2$. Hence the number of crossings in the red edges is $\beta = \sum_{j=1}^{2m-1} (2m-j)(2m-j+1)$.

Crossings on the Red edges in Γ' ***(i.e.,*** β'***)***: It is straightforward to observe that the number of such crossings is $\beta' = 2m + 2\sum_{i=1}^{m-1} 2mi$, as illustrated in Figure 5(b) when $m = 4$.

Now the number of crossings in Γ'' is $W_n + (\alpha' + \beta') - (\alpha + \beta)$, which can be simplified using Maple [13] to get an upper bound of $W_n - \frac{1}{384}n^3 + O(n^2)$. Since the modification we carried out for v_q can also be applied around independently to its diametrically opposite vertex, we can obtain a bound of $W_n - \frac{1}{192}n^3 + O(n^2)$. The following theorem summarizes the result of this section.

Theorem 1 *Every K_n, where $n = 16m + 4$ and $m \in \mathbb{Z}^+$, admits a biplanar drawing with at most $W_n - n^3/192 + O(n^2)$ edge crossings.*

5 Conclusion

In this paper we have given bounds on the biplanar crossing number of K_n. For small values of n, our technique for computing $cr_2(K_n)$ is incremental. Hence it is natural to ask whether every optimal biplanar drawing of K_{n+1} contains an optimal drawing of K_n. We proved that $cr_2(K_{11}) \in \{4, 5, 6\}$. It would be interesting to find an analytical argument to prove a better lower or upper bound on $cr_2(K_{11})$. Finally, given $f(n)$, how efficiently can we find k such that $cr_k(K_n) \in \Theta(f(n))$?

References

[1] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. The 2-page crossing number of K_n. *Discrete & Computational Geometry*, 49(4):747–777, 2013.

[2] B. M. Ábrego, S. Fernández-Merchant, and G. Salazar. The rectilinear crossing number of K_n: closing in (or are we?). In J. Pach, editor, *Thirty essays in Geometric Graph Theory*, pages 5–18. Springer, 2013.

[3] L. Beineke. Biplanar graphs: A survey. *Computers & Mathematics with Applications*, 34(11):1–8, 1997.

[4] P. Brass, W. Moser, and J. Pach. *Research Problems in Discrete Geometry.* Springer, 2006.

[5] C. Buchheim and L. Zheng. Fixed linear crossing minimization by reduction to the maximum cut problem. In *Proceedings of the 12th Annual International Conference on Computing and Combinatorics (COCOON)*, LNCS, pages 507–516. Springer, 2006.

[6] E. Czabarka, O. Sykora, L. A. Szekely, and I. Vrťo. Biplanar crossing numbers I: A survey of results and problems. In *Proceedings of More Sets, Graphs and Numbers: A Salute to Vera Sós and András Hajnal*, pages 57–77. Springer, 2006.

[7] É. Czabarka, O. Sýkora, L. A. Székely, and I. Vrťo. Biplanar crossing numbers. II. comparing crossing numbers and biplanar crossing numbers using the probabilistic method. *Random Structures and Algorithms*, 33(4):480–496, 2008.

[8] E. de Klerk, J. Maharry, D. V. Pasechnik, R. B. Richter, and G. Salazar. Improved bounds for the crossing numbers of $K_{m,n}$ and K_n. *SIAM Journal on Discrete Mathematics*, 20(1):189–202, 2006.

[9] E. de Klerk, D. V. Pasechnik, and G. Salazar. Improved lower bounds on book crossing numbers of complete graphs. *SIAM J. Discrete Math.*, 27(2):619–633, 2013.

[10] E. Gethner, L. Hogben, B. Lidický, F. Pfender, A. Ruiz, and M. Young. On crossing numbers of complete tripartite and balanced complete multipartite graphs. *Journal of Graph Theory*, 2016. (To appear).

[11] R. Guy. A combinatorial problem. *Nabla (Bulletin of the Malayan Mathematical Society)*, 7:68–72, 1960.

[12] R. K. Guy. The decline and fall of Zarankiewicz's theorem. In *In Proof Techniques in Graph Theory, Proceedings of the Second Ann Arbor Graph Theory Conference*, pages 63–69. Academic Press, 1969.

[13] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 10 Programming Guide.* Maplesoft, Waterloo ON, Canada, 2005.

[14] P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: A survey. *Graphs and Combinatorics*, 14(1):59–73, 1998.

[15] N. H. Nahas. On the crossing number of $K_{m,n}$. *Electronic Journal of Combinatorics*, 10(N8), 2003.

[16] A. Owens. On the biplanar crossing number. *IEEE Transactions on Circuit Theory*, 18(2):277 –280, 1971.

[17] S. Pan and R. B. Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56(2):128–134, 2007.

[18] R. B. Richter and G. Salazar. Crossing numbers. In J. L. Gross, J. Yellen, and P. Zhang, editors, *Handbook of Graph Theory*, pages 912–932. Chapman & Hall/CRC, 2nd edition, 2013.

[19] M. Schaefer. The graph crossing number and its variants: A survey. *Electronic Journal of Combinatorics*, DS21, 2014.

[20] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo. On k-planar crossing numbers. *Discrete Applied Mathematics*, 155(9):1106–1115, 2007.

[21] K. Zarankiewicz. On a problem of P. Turán concerning graphs. *Fund. Math.*, 41:137–145, 1954.

Squarability of rectangle arrangements

Matěj Konečný* Stanislav Kučera* Michal Opler* Jakub Sosnovec* Štěpán Šimsa* Martin Töpfer*

Abstract

We study when an arrangement of axis-aligned rectangles can be transformed into an arrangement of axis-aligned squares in $\mathbb{R}^2$ while preserving its structure. We found a counterexample to the conjecture of J. Klawitter, M. Nöllenburg and T. Ueckerdt whether all arrangements without crossing and side-piercing can be squared. Our counterexample also works in a more general case when we only need to preserve the intersection graph and we forbid side-piercing between squares. We also show counterexamples for transforming box arrangements into combinatorially equivalent hypercube arrangements. Finally, we introduce a linear program deciding whether an arrangement of rectangles can be squared in a more restrictive version where the order of all sides is preserved.

1 Introduction

In this paper, we are concerned with the following problem. Given an arrangement of axis-aligned rectangles in $\mathbb{R}^2$, is it possible to find an arrangement of axis-aligned squares with corresponding properties? J. Klawitter, M. Nöllenburg and T. Ueckerdt [2] asked which geometric rectangle arrangements can be transformed into combinatorially equivalent square arrangements. While showing some necessary and sufficient conditions for that, the question whether there exists an unsquarable rectangle arrangement without crossings and side-piercings (see Figure 1) remained open. We show a counterexample for that – an arrangement of rectangles which is not combinatorially equivalent to any square arrangement. Moreover, our counterexample works even in a more general case when we only need to preserve the intersection graph of arrangements and we forbid side-piercing between squares.

In Section 3 we generalize the problem to higher dimensions – considering hypercubes instead of squares and boxes instead of rectangles. We show that allowing crossings or side-piercings in any dimension leads to arrangements of boxes for which no corresponding arrangement of hypercubes exists.

Besides constructing counterexamples we also present an algorithm for deciding whether a given arrangement is squarable when the order of all sides has to be preserved (which implies combinatorial equivalence).

1.1 Preliminaries

Let $\mathcal{R}$ denote a given set of axis-aligned rectangles in $\mathbb{R}^2$ and S be a mapping from $\mathcal{R}$ to axis-aligned squares in $\mathbb{R}^2$ satisfying certain restrictions. If such S exists, we say that $\mathcal{R}$ is *squarable* and S is a *squaring* of $\mathcal{R}$. Thus $\mathrm{S}(\mathcal{R})$ is a set of squares obtained from $\mathcal{R}$ in a way specific to the particular variant and $\mathrm{S}(R)$ is the square representing the rectangle $R \in \mathcal{R}$. In each variant we explain the restrictions placed on the input set of rectangles $\mathcal{R}$ and on the output set of squares $\mathrm{S}(\mathcal{R})$.

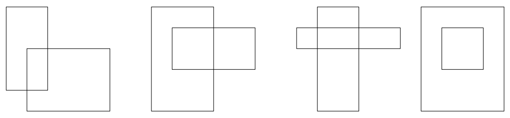


Figure 1: Intersection types. Respectively: corner intersection, side-piercing, cross intersection and containment.

There are four intersection types: corner intersection, side-piercing, cross intersection and containment (see Figure 1). Note that we do not include empty intersection (formed by disjoint rectangles) as an intersection type. Also, we only consider sets of rectangles where no two rectangle sides are collinear.

In all the discussed variants, we assume that the input set $\mathcal{R}$ contains no two rectangles with side-piercing or cross intersection. Allowing these intersection types easily leads to instances of arrangements of rectangles that cannot be squared – any two rectangles with the cross intersection clearly cannot be squared as well as the arrangement of four rectangles in Figure 2 for side-piercing.

Without loss of generality, we assume all the rectangles have positive coordinates. If it is not the case we just translate the whole arrangement. For a rectangle R we denote:

- $t(R)$ to be the y-coordinate of the top side of R,

*Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic. E-mails: `matejkon@gmail.com`, `stanislav.kucera@outlook.com`, `opler@iuuk.mff.cuni.cz`, `j.sosnovec@email.cz`, `simsa.st@gmail.com`, `mtopfer@gmail.com`

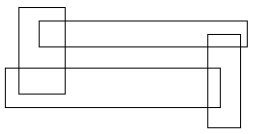


Figure 2: An arrangement that cannot be squared due to side-piercing intersections.

- $b(R)$ to be the y-coordinate of the bottom side of R,
- $r(R)$ to be the x-coordinate of the right side of R,
- $l(R)$ to be the x-coordinate of the left side of R,
- $h(R)$ to be the height of R: $h(R) = t(R) - b(R)$,
- $w(R)$ to be the width of R: $w(R) = r(R) - l(R)$.

1.2 Variants of the squarability problem

Let $\mathcal{R}$ be an arrangement of rectangles and S be a squaring of $\mathcal{R}$. We say that $\mathcal{R}$ and $\mathrm{S}(\mathcal{R})$ are *combinatorially equivalent* if for any $R_1, R_2 \in \mathcal{R}$, the intersection type of $\mathrm{S}(R_1)$ and $\mathrm{S}(R_2)$ is the same as the intersection type of R_1 and R_2 and these intersections happen exactly on the same sides (and corners). For example, if R_1 and R_2 have corner intersection that is in the upper left corner on R_1 and the lower right corner of R_2, the same must hold for $\mathrm{S}(R_1)$ and $\mathrm{S}(R_2)$.

Note that the above definition of combinatorial equivalence is strictly weaker than the one given in [2]. This definition is, however, convenient to us as the basic requirement. Since our counterexample works in this less restrictive case, it is also a counterexample when the referenced definition is used.

The following are variants of the squarability problem. They vary in the strength of the assumptions we put on the mapping S.

Preserve order of all sides. The output $\mathrm{S}(\mathcal{R})$ has to be combinatorially equivalent to $\mathcal{R}$ and, moreover, the respective order of sides on both axes has to be preserved. On a chosen axis, we can construct the sequence of sides of rectangles $\mathcal{R}$ from left to right as they appear, i.e., every rectangle will appear exactly twice. Then the same sequence of sides has to be realized in $\mathrm{S}(\mathcal{R})$.

Combinatorial equivalence. The output $\mathrm{S}(\mathcal{R})$ has to be combinatorially equivalent.

Keep intersections, forbid side-piercing. First, we require that the intersection graphs of $\mathcal{R}$ and $\mathrm{S}(\mathcal{R})$ are isomorphic, i.e., it holds that $R_1 \cap R_2 \neq \emptyset$ if and only if $\mathrm{S}(R_1) \cap \mathrm{S}(R_2) \neq \emptyset$ for all $R_1, R_2 \in \mathcal{R}$. Additionally, the squares in the output set $\mathrm{S}(\mathcal{R})$ must only have corner intersections or containment.

Keep intersection graph. We only require that the intersection graphs of $\mathcal{R}$ and $\mathrm{S}(\mathcal{R})$ are isomorphic.

Note that if S satisfies “Preserve order of all sides”, then it satisfies “Combinatorial equivalence”. In the same sense, “Combinatorial equivalence” implies “Keep intersection, forbid side-piercing” (by the assumption that $\mathcal{R}$ contains no side-piercing), which implies “Keep intersection graph”.

2 Counterexamples

In this section we will discuss examples of arrangements of rectangles, which cannot be squared in terms of the mapping S. In each subsection we consider squarability with respect to of one of the variants. We will start with the most restrictive case and proceed to more general variants.

2.1 Preserving order of all sides

If we want the resulting arrangement of squares to preserve the order of all sides, there is an easy example of four rectangles that cannot be squared.

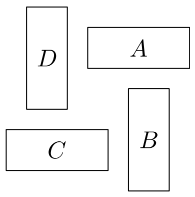


Figure 3: An arrangement not squarable in the most restrictive case.

Theorem 1 *The arrangement of rectangles in Figure 3 cannot be squared while preserving order of all sides.*

Proof. After squaring the arrangement we would get $w(A) > w(B) = h(B) > h(C) = w(C) > w(D) = h(D) > h(A) = w(A)$; thus, the arrangement is unsquarable. □

This is an easy observation but it is important, because this arrangement is exactly the one we will find in latter cases to prove unsquarability of other arrangements.

2.2 Combinatorial equivalence

In the second most restrictive definition of the mapping S we want the resulting arrangement of squares to not only have the same types of intersections but also to have the same position. This means that if there is a rectangle A and a rectangle B intersecting A in the top right corner then $\mathrm{S}(B)$ will intersect $\mathrm{S}(A)$ again in the top right corner.

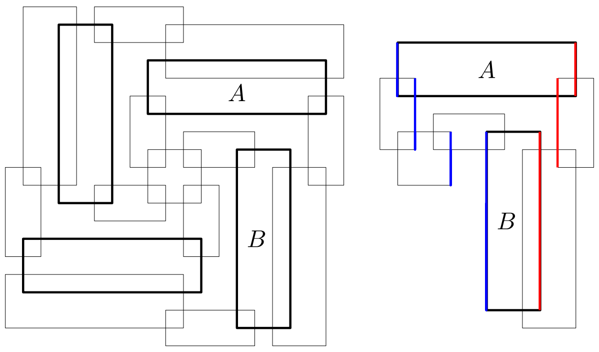


Figure 4: An arrangement not squarable when S keeps the combinatorial equivalence.

Theorem 2 *The arrangement of rectangles in the left picture of Figure 4 cannot be squared.*

Proof. To prove that, we want to show that the four bold rectangles form the pattern from Theorem 1. To do that, we need to prove that there is that cyclic condition on lengths of their sides. It suffices to show the dependency only for one pair of neighbouring rectangles since the arrangement is symmetric.

In the right picture of Figure 4, there is the situation for A and B where only the important rectangles are drawn. Suppose the rectangles are orientated as in the picture (orientation is fixed for the whole arrangement). To prove $w(A) > w(B)$ in all possible mappings S, it is sufficient to show $l(A) < l(B)$ and $r(A) > r(B)$.

We observe that when two rectangles C and D intersect a common rectangle E on its top (or bottom) side, C being the one intersecting it in the left corner, and C, D do not intersect each other it must hold $r(C) < l(D)$. When two rectangles F, G intersect each other then $l(F) < r(G)$. These two observations used on the red sides of the rectangles in Figure 4 together give us $r(A) > r(B)$. To prove $l(A) < l(B)$ we use the observations for the blue sides. □

2.3 Keep intersections, forbid side-piercing

So far we have been mainly building tools and considering easy examples. For S which only keeps intersections without allowing side-piercing in $S(\mathcal{R})$ we still need one more tool.

We refer to the arrangement depicted in the left picture of Figure 5 as a Σ-*gadget*. It is an arrangement of rectangles that can be squared even in the most restrictive case and we use it to force some useful properties.

Lemma 3 *All squarings of the Σ-gadget that keep intersections but forbid side-piercings are combinatorially equivalent, up to rotation and reflection.*

Proof. First look at the rectangles K, L, M, N in the middle. There is only one way to square them upon a rotation and reflection. Then we want to square rectangles A, B, C and D. Notice that A can be contained neither in K nor in L because it intersects P. This, and the fact the side-piercing is forbidden, gives us three possibilities how to place A, relatively to K and L. It can be either in the position as in the Figure 5 or in such position that it contains the intersection of the squares K and L or in the opposite corner than in the first case. In Figure 6 we see all the important cases. The first case is the one we want. In the second case, the position of A forces P (and Q) to intersect the bottom left corner of A because P (Q) needs to intersect D (B) without intersecting K (L). This means P and Q both intersect the bottom left corner of A and so they intersect each other, a contradiction. In the last case, A would intersect M and N, a contradiction. Therefore, there is only one way to square A and by symmetry the same is true for B, C and D. Now the rectangles P, Q, R and S can also be squared in only one way, completing the proof. □

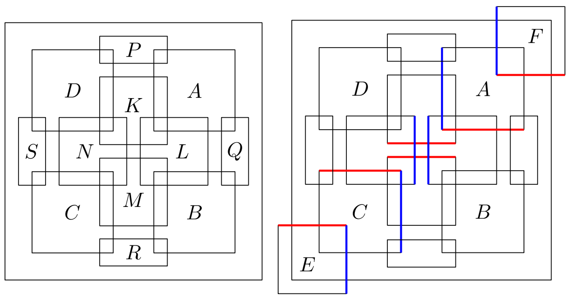


Figure 5: Σ-gadget and its usage.

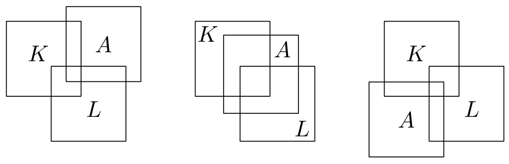


Figure 6: Three possible ways of placing rectangle A.

First we explain how we use the Σ-gadget in an arrangement. If we want another rectangle (or another Σ-gadget) to intersect our Σ-gadget in a corner, it must intersect both the surrounding rectangle and one of A, B, C or D depending on in which corner it intersects Σ-gadget. Besides these two it does not intersect anything else.

Now that we know that the Σ-gadget can be squared in exactly one way and how to use it in an arrangement, let us explore some of its useful properties. As is illustrated in the right picture of Figure 5, the most useful property comes to play when the Σ-gadget is intersected by rectangles in opposite corners, lets call them E and

F. Usually this only gives us one of the following conditions:

- $r(E) < l(F)$ (blue colored sides in the picture).
- $t(E) < b(F)$ (red colored sides in the picture).

The Σ-gadget provides both conditions at the same time, which is very useful when forcing the situation like in Theorem 1. At the same time, if the Σ-gadget is intersected in two corners, we can always say whether the corners are *opposite* or *adjacent*. For the purposes of arrangements in which the Σ-gadget is used, when we talk about the height, width, left side and so on, we always mean the height, width, left side, ... of the outer rectangle.

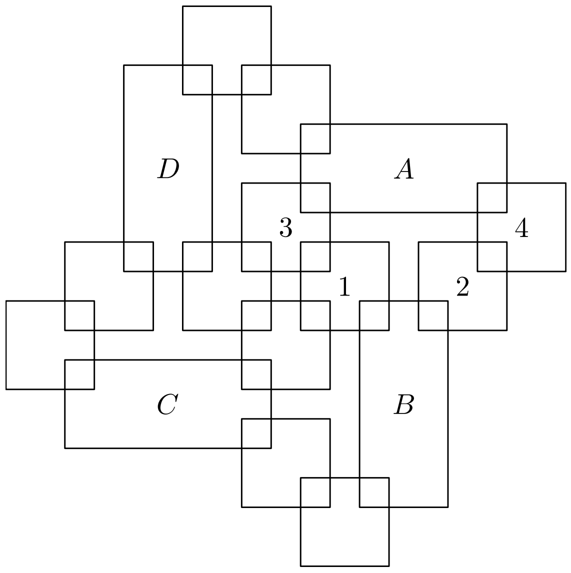


Figure 7: An arrangement using Σ-gadget not squarable even in the least restrictive case without side-piercing.

Having such a strong tool it is now easy to create an arrangement of rectangles that cannot be squared.

Theorem 4 *The arrangement from Figure 7 with the Σ-gadget instead of each rectangle cannot be squared.*

Proof. We show that rectangles A, B, C and D form the same arrangement as we saw in Theorem 1. Rectangles 1 and 2 lie on the same side of B. Rectangles 3 and 4 lie in the opposite corners of 1 and 2 respectively with respect to B. Because rectangles 1 and 2 are Σ-gadgets, this implies $l(B) > r(3)$ and $r(3) > l(A)$ since rectangles A and 3 intersect each other. We showed $l(B) > l(A)$ and similarly using rectangles 2 and 4 we can show $r(B) < r(A)$. Together this gives us $w(B) < w(A)$. Rotating the argument around the arrangement we show that if the arrangement gets squared it holds $w(B) < w(A) = h(A) < h(D) = w(D) < w(C) = h(C) < h(B) = w(B)$, which cannot be true. $\square$

One could think this cannot be all. After all in previous cases we needed to show there is only one way to draw the arrangement of squares and we always ended up with a square which we couldn't add. Note that we did just that by showing the Σ-gadget can be squared in only one way.

3 Higher dimensions

In this section we will make some observations about arrangements of boxes in higher dimensions. We use the same notation as before, that is $\mathcal{R}$ denotes a set of axis-aligned boxes in $\mathbb{R}^d$ and S its mapping to a set of axis-aligned hypercubes in $\mathbb{R}^d$. We will often work with projections of $\mathbb{R}^d$ to a subset of coordinates. For a set $I \subseteq \{1, \ldots, d\}$ let $\mu_I : \mathbb{R}^d \to \mathbb{R}^{|I|}$ be a projection that "forgets" all coordinates not indexed by I. Furthermore for a singleton-indexed projection we shorten its notation $\mu_{\{c\}} = \mu_c$. The result of a projection μ_I applied to $\mathcal{R}$ is an arrangement of axis-aligned boxes or hypercubes in $\mathbb{R}^{|I|}$.

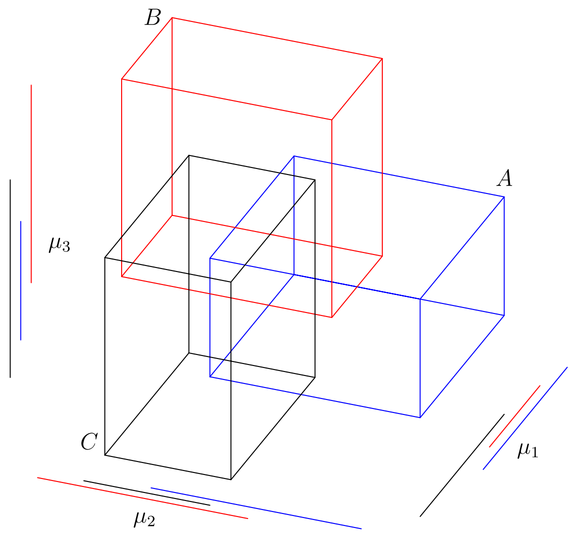


Figure 8: An arrangement of three boxes in $\mathbb{R}^3$ such that there is no combinatorially equivalent arrangement of cubes.

The notion of combinatorial equivalence extends naturally to higher dimensions. We can observe that with each extra dimension we get new intersection types. For example, consider the following arrangement of only three boxes in $\mathbb{R}^3$ from Figure 8. Each pair of these boxes intersects in such a way that one pierces the edge of the other. We claim that there cannot be a combinatorially equivalent arrangement of hypercubes. Assume that we have such an arrangement of hypercubes A', B'

and C'. Then the projection μ_1 forces A' to be bigger than B', similarly μ_2 for B' and C'. Finally, μ_3 forces C' to be bigger than A' and that is a contradiction.

3.1 Boxicity and cubicity

In the beginning we restricted to arrangements without side-piercings and cross intersections. It is fairly easy to see how they lead to counterexamples in the more restricted settings. However it is not so clear whether this restriction is needed in the least restrictive setting, i.e. preserving just the intersection graph. We will construct arrangements with these intersections which cannot be represented by an intersection graph of axis-aligned hypercubes up to a given dimension.

Let G be a simple undirected graph. The *boxicity* of G is the smallest dimension d such that G can be represented as an intersection graph of axis-aligned boxes in $\mathbb{R}^d$. Similar notions are the *cubicity* of G, where we consider a representation as an intersection graph of axis-aligned hypercubes, and the *unit cubicity* of G, where all the hypercubes have to be unit. The notion of boxicity and unit cubicity (usually referred simply as cubicity) was introduced in 1969 by Roberts [3] and has since been actively studied, e.g. in [1]. The results we prove in this section were shown previously for unit cubicity in [3]. But our definition of cubicity is more general.

Furthermore, let $R(k,d)$ denote the smallest integer such that every coloring of the complete graph on $R(k,d)$ vertices with d colors contains a monochromatic clique of size k. This is indeed one of the Ramsey numbers and it is well known that such a value exists.

As before, we want to construct such a graph that if there was an intersection-pattern equivalent arrangement of hypercubes it would force a cyclical inequality of hypercube sizes. However we do not have any tool yet for showing such inequalities in the most general setting.

Lemma 5 *Let G be a graph and v be a vertex which has at least $R(k+2,d)$ neighbours that are pairwise non-adjacent. Suppose G can be represented as an intersection graph of an arrangement $\mathcal{R}$ of axis-aligned hypercubes in $\mathbb{R}^d$ and $f : V(G) \to \mathcal{R}$ is the corresponding mapping. Then there is a neighbour w of v such that the hypercube $f(w)$ is more than k times smaller than the hypercube $f(v)$.*

Proof. Unsurprisingly, we will prove our claim using a coloring of the complete graph on $R(k+2,d)$ vertices. Each vertex gets labelled by one of the $R(k+2,d)$ neighbours of v. Observe that if two axis-aligned hypercubes R_1 and R_2 in $\mathbb{R}^d$ are disjoint then there is an integer c such that $\mu_c(R_1)$ and $\mu_c(R_2)$ are disjoint. We will color an edge with any c such that the corresponding hypercubes are disjoint under μ_c. The number of vertices guarantees us a monochromatic clique of size $k+2$. That means there are $k+2$ neighbours of $f(v)$ that are pairwise disjoint under μ_c for some c. We have $k+2$ pairwise disjoint intervals and all of them need to intersect the interval $\mu_c(f(v))$. From this follows that the smallest interval $\mu_c(f(w))$ is more than k times smaller than the interval $\mu_c(f(v))$. And since we are dealing with axis-aligned hypercubes, the hypercube $f(w)$ is more than k times smaller than the hypercube $f(v)$. □

Theorem 6 *For every d there is a graph G with boxicity 2 and cubicity larger than d.*

Proof. Consider a complete bipartite graph G with each partition of size $R(3,d)$. The boxicity of G is 2 since one partition of G can be represented as a set of vertical rectangles and the other as a set of horizontal rectangles (see Figure 9). Now suppose for a contradiction that the cubicity of G is at most d and fix any intersection representation with hypercubes in $\mathbb{R}^{d'}$, $d' \leq d$. Let v be the vertex of G such that the corresponding hypercube is the smallest one. Since v has exactly $R(3,d)$ pairwise disjoint neighbours, by Lemma 5 there must be a neighbour of v such that its corresponding hypercube is strictly smaller, that is a contradiction. □

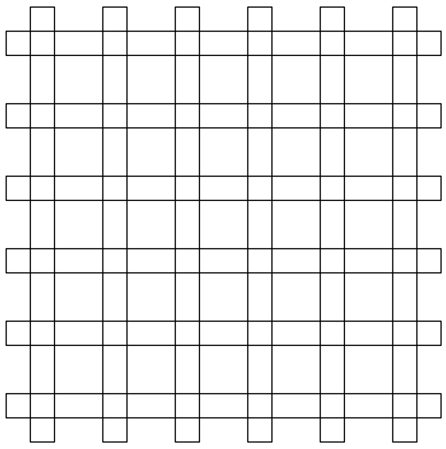


Figure 9: An arrangement of rectangles whose intersection graph is a complete bipartite graph with each partition of size $R(3,2)=6$.

4 Deciding squarability via LP

4.1 The problem

In this section we present a linear program deciding whether a given arrangement of n rectangles $\mathcal{R} = \{R_1,\ldots,R_n\}$ in $\mathbb{R}^2$ can be squared while preserving order of all sides.

Without loss of generality we can assume that all the endpoints of the intervals $[l(R_i), r(R_i)]$ and $[b(R_i), t(R_i)]$ have distinct values for all $i \in \{1, \dots, n\}$. Otherwise we could change the endpoints a little without changing intersections between rectangles.

By ordering the endpoints of the intervals of projected rectangles into an increasing sequence, we obtain the sequence $a'_1 < a'_2 < \dots < a'_{2n}$, where $a'_j = l(R_i)$ or $r(R_i)$ for some $i \in \{1, \dots, n\}$ (see Figure 10). Replacing $l(R_i)$ and $r(R_i)$ by i then yields the sequence $a_1, a_2, \dots, a_{2n}$ of numbers $\{1, \dots, k\}$, we call this sequence the x-sequence of $\mathcal{R}$. Clearly, each $i \in \{1, \dots, n\}$ appears there exactly twice, thus for every $i \in \{1, \dots, n\}$ we can define $a(i) = (j_1, j_2)$ such that $j_1 < j_2$ and $a_{j_1} = a_{j_2} = i$. The x-sequence describes the respective ordering of the rectangles' x-coordinates. A y-sequence and the corresponding function b are defined analogously.

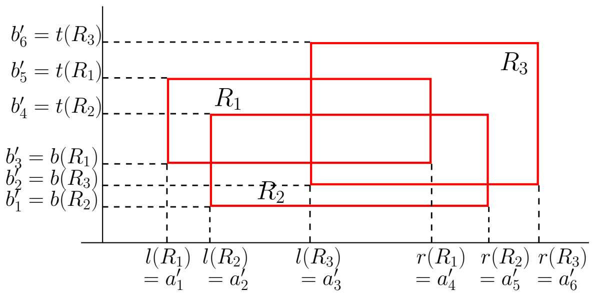


Figure 10: The x-sequence is $1, 2, 3, 1, 2, 3$ and the y-sequence is $2, 3, 1, 2, 1, 3$.

The decision problem can be reformulated in the following way: Given a family of rectangles $\mathcal{R} = \{R_1, \dots, R_n\}$, does there exist a family of squares $\mathcal{S} = \{S_1, \dots, S_n\}$ such that the x-sequence of $\mathcal{S}$ is identical to that of $\mathcal{R}$ and the y-sequence of $\mathcal{S}$ is identical to that of $\mathcal{R}$?

4.2 Linear program

Let us present a linear program solving the problem for an input set $\mathcal{R} = \{R_1, \dots, R_n\}$. Let $a_1, a_1, \dots, a_{2n}$ be the x-sequence and $b_1, b_2, \dots, b_{2n}$ the y-sequence of $\mathcal{R}$. We have variables

$$x_1, \dots, x_{2n-1}, y_1, \dots, y_{2n-1} \geq 1,$$

where the value of x_j represents the distance of the corresponding interval endpoints of rectangles R_{a_j} and $R_{a_{j+1}}$ and the value of y_j represents the distance of the corresponding endpoints of R_{b_j} and $R_{b_{j+1}}$ (see Figure 11). Let $(\mathbf{x}, \mathbf{y}) = (x_1, \dots, x_{2n-1}, y_1, \dots, y_{2n-1})$ be any feasible solution to the following set of equalities. For every $i = 1, \dots, n$ we have an equality

$$\sum_{k=j_1}^{j_2-1} x_k = \sum_{k=j'_1}^{j'_2-1} y_k, \text{ where } a(i) = (j_1, j_2),\ b(i) = (j'_1, j'_2)$$

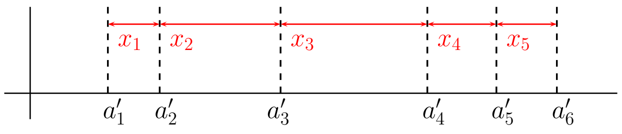


Figure 11: The meaning of the variables $x_1, \dots, x_{2n-1}$.

From the solution $(\mathbf{x}, \mathbf{y})$ we construct the corresponding set of squares $\mathcal{S} = \{S_1, \dots, S_n\}$ as follows. Let $a(i) = (j_1, j_2)$ and $b(i) = (j'_1, j'_2)$, we set the coordinates of S_i such that

$$l(S_i) = \sum_{k=1}^{j_1-1} x_k, \quad r(S_i) = \sum_{k=1}^{j_2-1} x_k,$$

$$b(S_i) = \sum_{k=1}^{j'_1-1} y_k, \quad t(S_i) = \sum_{k=1}^{j'_2-1} y_k$$

As $x_i, y_i \geq 1$ for all $i \in \{1, \dots, 2n-1\}$, it is clear that the x-sequence and y-sequences of $\mathcal{R}$ are preserved in $\mathcal{S}$. The claim that $\mathcal{S}$ consists of squares follows immediately from the constraints of the linear program. Thus we obtain that if the linear program finds a feasible solution, we can construct an appropriate set of squares.

Reversely, let $\mathcal{S}$ be a set of squares that has the same x-sequence and y-sequence as $\mathcal{R}$. We can construct the variables $x_1, \dots, x_{2n-1}$ and $y_1, \dots, y_{2n-1}$ as the corresponding distances. It remains to sufficiently "blow up" this solution so that all of the variables are at least 1. This is easily accomplished by multiplying the variables by the inverse of the minimum of them. We obtain a feasible solution to the linear program, as desired.

5 Acknowledgments

The authors would like to thank Pavel Valtr, Jan Kratochvíl and Stephen Kobourov for supervising the seminar where this paper was created. Our gratitude also goes to the anonymous referees for their helpful comments.

References

[1] L. S. Chandran and K. A. Mathew. An upper bound for cubicity in terms of boxicity. *Discrete Mathematics*, 309(8):2571–2574, 2009.

[2] J. Klawitter, M. Nöllenburg, and T. Ueckerdt. Combinatorial Properties of Triangle-Free Rectangle Arrangements and the Squarability Problem. In *Graph Drawing and Network Visualization: 23rd International Symposium, GD 2015*, pages 231–244. Springer, 2015.

[3] F. S. Roberts. On the boxicity and cubicity of a graph. In *Recent Progress in Combinatorics (Proc. Third Waterloo Conf. on Combinatorics, 1968)*, pages 301–310. Academic Press, New York, 1969.

Transforming Hierarchical Trees on Metric Spaces*

Mahmoodreza Jahanseir† Donald R. Sheehy‡

Abstract

We show how a simple hierarchical tree called a cover tree can be turned into an asymptotically more efficient one known as a net-tree in linear time. We also introduce two linear-time operations to manipulate cover trees called coarsening and refining. These operations make a trade-off between tree height and the node degree.

1 Introduction

There are many very similar data structures for searching and navigating n points in a metric space $\mathcal{M}$. Most such structures support range queries and (approximate) nearest neighbor search among others. For computational geometers, two of the most important such structures for general metric spaces are the cover tree [2] and the net-tree [8]. Cover trees, by virtue of their simplicity, have found wide adoption, especially for machine learning applications. Net-trees on the other hand, provide much stronger theoretical guarantees and can be used to solve a much wider class of problems, but they come at the cost of unrealistic constant factors and complex algorithms. In this paper, we generalize these two data structures and show how to convert a cover tree into a net tree in linear time. In fact, we show that a cover tree with the right parameters, satisfies the stronger conditions of a net tree, thus finding some middle ground between the two. In Section 5, we give efficient algorithms for modifying these parameters for an existing tree.

Related Work For Euclidean points, Quadtrees [5] and k-d trees [1] are perhaps the two famous data structures. Most data structures for general metric spaces are generalizations of these. Uhlmann [11] proposed ball trees to solve the proximity search on metric spaces. Ball trees are generalizations of k-d trees. Yianilos [12] proposed a structure similar to ball trees called a vp-tree that allows $O(\log n)$-time queries in expectation for restricted classes of inputs.

Clarkson [3] proposed two randomized data structures to answer nearest neighbor queries in metric spaces that satisfy a certain sphere packing property. These data structures assume that the input and query point are drawn from the same probability distribution. The nearest neighbor query time of these structures depends on the *spread* Δ of the input, which is the ratio of the diameter to the distance between the closest pair of points.

Karger & Ruhl [9] devised a dynamic data structure for nearest neighbor queries in growth restricted metrics. A *closed metric ball* centered at p with radius r is denoted $\mathrm{B}(p,r) := \{q \in P \mid \mathbf{d}(p,q) \leq r\}$. Karger & Ruhl defined the *expansion constant* as the minimum μ such that for all $p \in \mathcal{M}$ and $r > 0$, $|\mathrm{B}(p,2r)| \leq \mu\,|\mathrm{B}(p,r)|$. A growth restricted metric space has constant μ (independent of n). Karger & Ruhl proved that their data structure has size $\mu^{O(1)} n \log n$, and answers nearest neighbor queries in $\mu^{O(1)} \log n$.

Gupta et al. [7] defined the *doubling constant* ρ of a metric space as the minimum ρ such that every ball in $\mathcal{M}$ can be covered by ρ balls of half the radius. The *doubling dimension* is defined as $\gamma = \lg \rho$. A metric is called doubling when it has a constant doubling dimension.

Krauthgamer & Lee [10] proposed *navigating nets* to answer (approximate) nearest neighbor queries in $2^{O(\gamma)} \log \Delta + (1/\varepsilon)^{O(\gamma)}$-time for doubling metrics. Navigating nets require $2^{O(\gamma)} n$ space.

Gao et al. [6] proposed a $(1+\varepsilon)$-spanner with size $O(n/\varepsilon^d)$ for a set of n points in $\mathbb{R}^d$. Their data structure is similar to navigating nets and can be constructed in $O(n \log \Delta / \varepsilon^d)$ time and answers approximate nearest neighbor queries in $O(\log \Delta)$ time. They also maintained the spanner under dynamic updates and continuous motion of the points.

Har-Peled & Mendel [8] devised a data structure called a *net-tree* to address approximate nearest neighbor search and some other problems in doubling metrics. They proposed a linear-time algorithm to construct a net-tree of size $O(\rho^{O(1)} n)$ starting from a specific ordering of the points called an approximate greedy permutation. Constructing a greedy permutation requires $O(\rho^{O(1)} n \log(\Delta n))$ time. To beat the spread, they proposed a randomized algorithm to generate an approximate greedy permutation in $O(\rho^{O(1)} n \log n)$ time. Net-trees support approximate nearest neighbor search in $O(2^{O(\gamma)} \log n) + (1/\varepsilon)^{O(\gamma)}$ time.

Beygelzimer et al. [2] presented *cover trees* to solve the nearest neighbor problem in growth restricted metrics. Cover trees are a simplificiation of navigating nets and can be constructed incrementally in $O(\mu^6 n \log n)$

*Partially supported by the National Science Foundation under grant numbers CCF-1464379 and CCF-1525978

†University of Connecticut reza@engr.uconn.edu
‡University of Connecticut don.r.sheehy@gmail.com

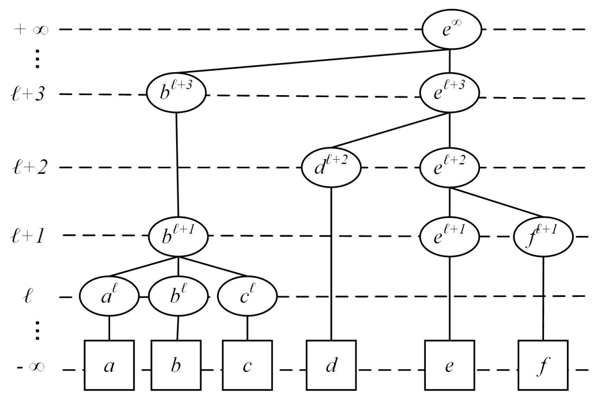


Figure 1: A hierarchical tree on $P = \{a, b, c, d, e, f\}$. Squares and ovals illustrate points and nodes respectively.

time. The space complexity of cover trees is $O(n)$ independent of doubling constant or expansion constant.

Cole & Gottlieb [4] extended the notion of navigating nets to construct a dynamic data structure to support approximate nearest neighbor search in $O(\log n) + (1/\varepsilon)^{O(1)}$ time for doubling metrics. Similar to net-trees, their data structure provides strong packing and covering properties. To insert a new point, they used biased skip lists to make the search process faster. They proved that the data structure requires $O(n)$ space independent of doubling dimension of the metric space.

2 Definitions

Hierarchical trees. Cover trees and net-trees are both examples of hierarchical trees. In these trees, the input points are leaves and each point p can be associated with many internal nodes. Each node is uniquely identified by its associated point and an integer called its *level*. Leaves are in level $-\infty$ and the root is in $+\infty$. The node in level ℓ associated with a point p is denoted p^ℓ. Let $\mathrm{par}(p^\ell)$ be the parent of a node $p^\ell \in T$. Also, let $\mathrm{ch}(p^\ell)$ be the children of p^ℓ. Each non-leaf has a child with the same associated point. Similar to compressed quadtrees, a node skips a level iff it is the only child of its parent and it has only one child. Let L_ℓ be the points associated with nodes in level at least ℓ. Let P_{p^ℓ} denote leaves of the subtree rooted at p^ℓ. The levels of the tree represent the metric space at different scales. The constant $\tau > 1$, called the *scale factor* of the tree determines the change in scale between levels. Fig 1 shows an example of hierarchical trees. Note that in this figure the tree is neither a cover tree nor a net-tree, because there are not any restrictions on the distance between points.

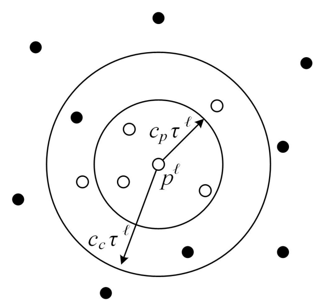


Figure 2: Packing and covering balls for a point p at level ℓ in a net-tree. White points belong to the subtree rooted at node p^ℓ.

Cover Trees. A *cover tree* T is a hierarchical tree with following properties.

- **Packing:** For all distinct $p, q \in L_\ell$, $\mathbf{d}(p, q) > c_p\tau^\ell$.
- **Covering:** For each $r^h \in \mathrm{ch}(p^\ell)$, $\mathbf{d}(p, r) \le c_c\tau^\ell$.

We call c_p and c_c the *packing constant* and the *covering constant*, respectively, and $c_c \ge c_p > 0$. We represent all cover trees with the same scale factor, packing constant, and covering constant with $\mathrm{CT}(\tau, c_p, c_c)$. Note that the cover tree definition by Beygelzimer et al. [2] results a tree in $\mathrm{CT}(2, 1, 1)$.

Net-trees. A *net-tree* is a hierarchical tree. For each node p^ℓ in a net-tree, the following invariants hold.

- **Packing:** $\mathrm{B}(p, c_p\tau^\ell) \bigcap P \subset P_{p^\ell}$.
- **Covering:** $P_{p^\ell} \subset \mathrm{B}(p, c_c\tau^\ell)$. [1]

Here, c_p and c_c are defined similar to cover trees. Fig 2 illustrates both packing and covering balls for a point p at some level ℓ in a net-tree. Let $\mathrm{NT}(\tau, c_p, c_c)$ denote the set of net-trees. The algorithm in [8] constructs a tree in $\mathrm{NT}(11, \frac{\tau-5}{2(\tau-1)}, \frac{2\tau}{\tau-1})$.

The main difference in the definitions is in the packing conditions. The net-tree requires the packing to be consistent with the hierarchical structure of the tree, a property not necessarily satisfied by the cover trees. Also, Har-Peled and Mendel [8] set $\tau = 11$, whereas optimized cover tree code sets $\tau = 1.3$.

A net-tree can be augmented to maintain a list of nearby nodes called relatives defined for each node p^ℓ as follows.

$$\mathrm{Rel}(p^\ell) = \{x^f \in T \text{ with } y^g = \mathrm{par}(x^f) \mid f \le \ell < g, \text{ and } \mathbf{d}(p, x) \le c_r\tau^\ell\}$$

[1]The packing condition we give is slightly different from [8], but it is an easy exercise to prove this (more useful) version is equivalent.

We call c_r the *relative constant*, and Har-Peled and Mendel set $c_r = 13$.

In this paper, we add a new and easy to implement condition on cover trees. We require that children of a node p^ℓ are closer to p than to any other point in L_ℓ.

3 From cover trees to net-trees

In this section, first we show that for every node in a cover tree, the size of children and relatives of that node is constant. Then, we prove that a cover tree with a sufficiently large scale factor satisfies both stronger packing and covering properties of net-trees.

Lemma 1 *For each node p^ℓ in $T \in \mathrm{CT}(\tau, c_p, c_c)$, $|\mathrm{ch}(p^\ell)| = O(\rho^{\lg c_c\tau/c_p})$.*

Proof. When $|\mathrm{ch}(p^\ell)| > 1$, all children of p^ℓ are in level $\ell-1$. From the packing property, the distance between every two nodes in this list is greater than $c_p\tau^{\ell-1}$. We know that all children of p^ℓ are within the distance $c_c\tau^\ell$ of p. By the definition of the doubling constant, the ball centered at p with radius $c_c\tau^\ell$ will be covered by $O(\rho^{\lg c_c\tau/c_p})$ balls of radius $c_p\tau^{\ell-1}$. □

Lemma 2 *Let $p^\ell \in T$ and $T \in \mathrm{CT}(\tau, c_p, c_c)$. For each two nodes $s^e, t^f \in \mathrm{Rel}(p^\ell)$, $\mathbf{d}(s,t) > c_p\tau^\ell$.*

Proof. Let $r^h = \mathrm{par}(s^e)$. By the definition of relatives, $e \le \ell < h$. If $e < \ell$, then $s = r$ and $s \in L_h$. Because $L_h \subset L_\ell$, the distance of s to all points in L_ℓ is greater than $c_p\tau^\ell$. Otherwise, s is in level ℓ. The same argument holds for t^f. Therefore, $s, t \in L_\ell$, and it implies $\mathbf{d}(s,t) > c_p\tau^\ell$. □

Lemma 3 *For each node p^ℓ in $T \in \mathrm{CT}(\tau, c_p, c_c)$, $|\mathrm{Rel}(p^\ell)| = O(\rho^{\lg c_r/c_p})$*

Proof. By the definition of relatives, all nodes in $\mathrm{Rel}(p^\ell)$ are within the distance $c_r\tau^\ell$ of point p. From Lemma 2, the distance between any two points in $\mathrm{Rel}(p^\ell)$ is greater than $c_p\tau^\ell$. Therefore, the total size of $\mathrm{Rel}(p^\ell)$ is $O(\rho^{\lg c_r/c_p})$. □

Lemma 4 *For each descendant x^f of p^ℓ in $T \in \mathrm{CT}(\tau, c_p, c_c)$, $\mathbf{d}(p,x) < \frac{c_c\tau}{\tau-1}\tau^\ell$*

Proof. The covering property and the triangle inequality imply that

$$\mathbf{d}(p,x) \le \sum_{i=f}^{\ell} c_c\tau^i < c_c\sum_{i=0}^{\infty}\tau^{\ell-i} = c_c\frac{\tau^{\ell+1}}{\tau-1}.$$

□

Theorem 5 *For all $\tau > \frac{2c_c}{c_p} + 1$, if $T \in \mathrm{CT}(\tau, c_p, c_c)$, then $T \in \mathrm{NT}(\tau, \frac{c_p(\tau-1)-2c_c}{2(\tau-1)}, \frac{c_c\tau}{\tau-1})$.*

Algorithm 1 Augmenting a given cover tree with relatives

```
1: procedure Augment($T, c_r$)
2:     for all $p^\ell \in T$ in decreasing order of level $\ell$ do
3:         Rel($p^\ell$) ← $p^\ell$
4:         if $p^\ell$ is not the root then
5:             Relatives($p^\ell, c_r, true$)
```

Proof. From Lemma 4, for a node $p^\ell \in T$, $P_{p^\ell} \subset \mathrm{B}(p, \frac{c_c\tau}{\tau-1}\tau^\ell)$. Suppose for contradiction there exists a point $r \in \mathrm{B}(p, \frac{c_p(\tau-1)-2c_c}{2(\tau-1)}\tau^\ell)$ such that $r \notin P_{p^\ell}$. Then, there exists a node $x^f \in T$ which is the lowest node with $f \ge \ell$ and $r \in P_{x^f}$. Let y^g be the child of x^f such that $r \in P_{y^g}$. It is clear that $g < \ell$. First, Let $g < f-1$. So, $x = y$ and $\mathbf{d}(p,x) = \mathbf{d}(p,y) > c_p\tau^\ell$. By the triangle inequality,

$$\begin{aligned}\mathbf{d}(y,r) \ge \mathbf{d}(y,p) - \mathbf{d}(p,r) &> c_p\tau^\ell - \frac{c_p(\tau-1)-2c_c}{2(\tau-1)}\tau^\ell \\ &> \frac{c_p(\tau-1)+2c_c}{2(\tau-1)}\tau^\ell.\end{aligned}$$

Also, $\mathbf{d}(y,r) \le \frac{c_c\tau}{\tau-1}\tau^g < \frac{c_c\tau^\ell}{\tau-1}$. Therefore,

$$\frac{c_p(\tau-1)+2c_c}{2(\tau-1)}\tau^\ell < \frac{c_c\tau^\ell}{\tau-1}.$$

This implies that $c_p(\tau-1) < 0$, which is a contradiction. Now, let $g = f-1$. In this case, we have $f = \ell$ and $g = \ell-1$. By the parent property, $\mathbf{d}(y,p) > \mathbf{d}(y,x)$. So,

$$\mathbf{d}(y,p) \ge \mathbf{d}(p,x) - \mathbf{d}(x,y) > c_p\tau^\ell - \mathbf{d}(y,p) > c_p\tau^\ell/2.$$

Also, by the triangle inequality,

$$\begin{aligned}\mathbf{d}(y,r) \ge \mathbf{d}(y,p) - \mathbf{d}(p,r) &> \frac{c_p}{2}\tau^\ell - \frac{c_p(\tau-1)-2c_c}{2(\tau-1)}\tau^\ell \\ &> \frac{c_c\tau^\ell}{\tau-1}.\end{aligned}$$

We get a contradiction because $\mathbf{d}(y,r) \le \frac{c_c\tau^\ell}{\tau-1}$. Therefore, $r \in P_{p^\ell}$. □

4 Augment cover trees

Theorem 5 shows that for a sufficiently large scale factor packing and covering properties in a cover tree imply packing and covering properties of net-trees. However, net-tree nodes maintain a list of nearby nodes called *relatives*. Algorithm 1 is a procedure that adds a list of relatives to each node of a cover tree. Note that Relatives is similar to the find relative algorithm in [8], but it gives a smaller relative constant.

Algorithm 2 Finding relatives of a node p^ℓ

```
1: procedure Relatives($p^\ell, c_r, update$)
2:     Let $q^m = parent(p^\ell)$
3:     for all $x^f \in \mathrm{Rel}(q^m)$ do
4:         Let $y^g = \mathrm{par}(x^f)$
5:         if $\mathbf{d}(p,x) \le c_r\tau^\ell$ and $f \le \ell < g$ then
6:             Add $x^f$ to $\mathrm{Rel}(p^\ell)$
7:         else if $update = true$ and $\ell \le f < m$ and $\mathbf{d}(p,x) \le c_r\tau^f$ then
8:             Add $p^\ell$ to $\mathrm{Rel}(x^f)$
9:     $candidates \leftarrow \bigcup_{r^e \in \mathrm{Rel}(q^m)} \mathrm{ch}(r^e) \setminus \{p^\ell\}$
10:    for all $x^f \in candidates$ do
11:        Let $y^g = \mathrm{par}(x^f)$
12:        if $\mathbf{d}(p,x) \le c_r\tau^\ell$ and $f \le \ell < g$ then
13:            Add $x^f$ to $\mathrm{Rel}(p^\ell)$
14:        else if $\mathbf{d}(p,x) \le c_r\tau^f$ and $\ell \le f < m$ then
15:            if $update = true$ then
16:                Add $p^\ell$ to $\mathrm{Rel}(x^f)$
17:            $candidates \leftarrow candidates \cup \mathrm{ch}(x^f)$
```

Theorem 6 *For each node p^ℓ in $T \in \mathrm{CT}(\tau, c_p, c_c)$ and $c_r = \frac{c_c\tau^2}{(\tau-1)^2}$, Relatives correctly finds $\mathrm{Rel}(p^\ell)$.*

Proof. Suppose for contradiction there exists x^f with $y^g = \mathrm{par}(x^f)$ such that $x^f \in \mathrm{Rel}(p^\ell)$, and Relatives does not find it. Therefore, either $x^f \notin \mathrm{Rel}(q^m)$ or it has an ancestor s^h with $h \le m-1$ such that $p^\ell \notin \mathrm{Rel}(s^h)$. We consider each case separately.

Case 1: $x^f \notin \mathrm{Rel}(q^m)$. In this case, at least one of the two conditions of relatives does not hold for x^f. If $\mathbf{d}(q,x) > \frac{c_c\tau^2}{(\tau-1)^2}\tau^m$, then by the triangle inequality,

$$\begin{aligned}\mathbf{d}(p,x) \ge \mathbf{d}(q,x) - \mathbf{d}(p,q) &> \frac{c_c\tau^2}{(\tau-1)^2}\tau^m - c_c\tau^m \\ &> c_c\frac{2\tau-1}{(\tau-1)^2}\tau^{\ell+1}.\end{aligned}$$

We assumed that $x^f \in \mathrm{Rel}(p^\ell)$, so $\mathbf{d}(p,x) \le \frac{c_c\tau^2}{(\tau-1)^2}\tau^\ell$. These inequalities imply $\tau < 1$, a contradiction. If $\mathbf{d}(q,x) \le \frac{c_c\tau^2}{(\tau-1)^2}\tau^m$ and $g > f > m$, then $f > \ell$ is also a contradiction. The last case $\mathbf{d}(q,x) \le \frac{c_c\tau^2}{(\tau-1)^2}\tau^m$ and $f < g \le m$ is a special case of $p^\ell \notin \mathrm{Rel}(s^h)$, which is described in the following.

Case 2: $p^\ell \notin \mathrm{Rel}(s^h)$. We know that $\ell < h < m$, so $\mathbf{d}(p,s) > \frac{c_c\tau^2}{(\tau-1)^2}\tau^h$. Also, $\tau^{h-1} \ge \tau^\ell$ because $h \ge \ell+1$. Using the triangle inequality and then applying Lemma 4,

$$\begin{aligned}\mathbf{d}(p,s) \le \mathbf{d}(p,x) + \mathbf{d}(x,s) &< \frac{c_c\tau^2}{(\tau-1)^2}\tau^\ell + \frac{c_c\tau}{\tau-1}\tau^h \\ &< \frac{c_c\tau^2}{(\tau-1)^2}\tau^{h-1} + \frac{c_c\tau}{\tau-1}\tau^h = c_c(\frac{\tau^2}{(\tau-1)^2})\tau^h.\end{aligned}$$

This is a contradiction. □

Theorem 7 *Algorithm 1 has time complexity $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n)$.*

Proof. We use an amortized analysis for the time complexity of Relatives. While finding relatives, if a node is inserted into the relative list of another node, we decrease one credit from the node whose relative list has been grown. Note that in Algorithm 2, for a node x^f in $\mathrm{Rel}(q^m)$ or children of $\mathrm{Rel}(q^m)$, when $x^f \notin \mathrm{Rel}(p^\ell)$ and $p^\ell \notin \mathrm{Rel}(x^f)$, p^ℓ is responsible for checking x^f. Also, a child of a node x^f is required to be checked against relative conditions if $p^\ell \in \mathrm{Rel}(x^f)$. In this case, we charge node x^f one credit. From Lemma 3, the relative list for each node has size $O(\rho^{\lg\frac{c_c\tau^2}{c_p(\tau-1)^2}})$. Also, Lemma 1 implies that each node has at most $O(\rho^{\lg\frac{c_c\tau}{c_p}})$ children. Therefore, the total required credit for each node of the tree is $O(\rho^{\lg\frac{c_c\tau^2}{c_p(\tau-1)^2}}) + 2\cdot O(\rho^{\lg\frac{c_c\tau^2}{c_p(\tau-1)^2}}\rho^{\lg\frac{c_c\tau}{c_p}}) = O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau})$. So, the total total time complexity is $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n)$. □

5 Transform cover trees

For a cover tree, there is a trade-off between the height of the tree and the scale factor. It is not hard to see that the height of a cover tree has upper bound $O(\log_\tau \Delta)$. So by increasing the scale factor, the height of the tree will be decreased. Also, from Lemma 1, increasing the scale factor results in more children for each node of a cover tree.

In this section, we define two operations to change scale factor of a given tree. A *coarsening* operation modifies the tree to increase the scale factor. Similarly, a refining operation results a tree with smaller scale factor. Note that in Theorem 5, we assumed that $\tau > \frac{2c_c}{c_p} + 1$. However, in many cases we may have $\tau \le \frac{2c_c}{c_p} + 1$. For example, Beygelzimer et. al. [2] set $\tau = 2$, and they found $\tau = 1.3$ is even more efficient in practice. In these situations, we can use the coarsening operation to get a cover tree with the stronger packing and covering conditions of net-trees.

5.1 Coarsening

The coarsening operation can be seen as combining every k levels of T into one level in T'. We define a mapping between nodes of T and T'. In this mapping, each

Algorithm 3 Coarsening operation for a given cover tree

```
1: procedure Coarsening($T, k$)
2:     $T' \leftarrow \emptyset$
3:     Augment($T, \frac{3c_c\tau^2}{(\tau-1)^2}$)
4:     for all $p^\ell \in T$ in increasing order of level $\ell$ do
5:         $q^m \leftarrow$ the lowest ancestor of $p^\ell$ with $m' > \ell'$
6:         if $p = q$ then
7:             $p^{\ell'} \leftarrow$ FindNode(high($p$), $\ell'$)
8:             $q^{\ell'+1} \leftarrow$ FindNode(high($q$), $\ell'+1$)
9:         else
10:            $par \leftarrow q^m$
11:            $relatives \leftarrow \mathrm{Rel}(q^m)$
12:            if $m' > \ell'+1$ then
13:                $h \leftarrow k(\lfloor \ell/k \rfloor + 2) - 1$
14:                Relatives($q^h, \frac{3c_c\tau^2}{(\tau-1)^2}, false$) $\cup \{q^h\}$
15:                $relatives \leftarrow \mathrm{Rel}(q^h)$
16:            for all $x^f \in relatives$ do
17:                if $f' = \ell'+1$ then
18:                    $x^f \leftarrow$ RestrictedNN($p, x^f, k$)
19:                if $\mathbf{d}(p,x) < \mathbf{d}(p,par)$ then
20:                    $par \leftarrow x^f$
21:            $par^{\ell'} \leftarrow$ FindNode(high($par$), $\ell'$)
22:            $par^{\ell'+1} \leftarrow$ FindNode(high($par$), $\ell'+1$)
23:            $p^{\ell'} \leftarrow$ FindNode(high($p$), $\ell'$)
24:            Add $p^{\ell'}$ as a child of $par^{\ell'+1}$
```

node p^ℓ in T maps to a node $p^{\ell'} = p^{\lfloor \ell/k \rfloor}$ in T'. Here, we use prime as a function that indicates the level of the node in T' that corresponds to p^ℓ, i.e. $\ell' = \lfloor \ell/k \rfloor$. We also assume that each point p in T' maintains high(p), which is the highest node of T' associated to point p. Algorithm 3 describes the coarsening operation.

Coarsening uses three procedures Relatives, RestrictedNN, and FindNode. The first procedure is described in Algorithm 2. Note that in Algorithm 3, T does not have node $q^{k(\lfloor \ell/k \rfloor+2)-1}$. This node is a dummy and we set q^m as its parent. The only reason to use this dummy node is to bound the running time of the algorithm. The next procedure is RestrictedNN, and it returns the nearest neighbor to point p among those nodes of the subtree rooted at x^f such that their levels in T' are greater that ℓ'. Finally, FindNode receives a node $p^{e'}$ and a level m' in T', and it tries to find node $p^{m'}$. If it finds that node, the node is returned. Otherwise, $p^{m'}$ will be inserted in T' such that it satisfies all properties of a hierarchical tree. More specifically, if $p^{m'}$ has only one child $p^{e'}$ and $p^{e'}$ has only one child, then $p^{e'}$ will be removed from T' and the only child of $p^{e'}$ will be added as a child of $p^{m'}$. Then, this new node will be returned.

Theorem 8 *Algorithm 3 converts a $T \in \mathrm{CT}(\tau, c_p, c_c)$ to $T' \in \mathrm{CT}(\tau^k, c_p, \frac{c_c\tau}{\tau-1})$.*

Proof. The theorem requires showing three invariants holds: the covering property, the packing property, and the parent relation. First we prove that T' satisfies the covering property. If $r^e \in T$ is the descendant at most k levels down from some p^ℓ, then from Lemma 4,

$$\mathbf{d}(p,r) < \frac{c_c\tau}{\tau-1}\tau^\ell < \frac{c_c\tau}{\tau-1}(\tau')^{\ell'}.$$

Because we are combining sets of k consecutive levels, it follows that each node in T' will have a node in the level above whose distance is at most this amount. It follows that T' has a covering constant $\frac{c_c\tau}{\tau-1}$.

Next, we prove that the packing constant is correct. If $\ell = k\ell'$, then the minimum distance between points in level ℓ' of T' is equal to the minimum distance between points in level ℓ of T, which is at least $c_p\tau^\ell = c_p(\tau')^{\ell'}$. Thus, the points in level ℓ' of T' satisfy the packing condition and the packing constant is c_p.

Now, we prove that this algorithm correctly finds the parent of $p^{\ell'}$ in T'. Without loss of generality, let ℓ be divisible by k. Also, let s be the closest point to p among all points in $L_{\ell+k}$, and s has been appeared for the first time in level e such that $e' > \ell'$. So, s is the right parent for p^ℓ. For contradiction, assume that $s^e \notin \mathrm{Rel}(q^m)$ and s^e is not resulted from RestrictedNN over all nodes in $\mathrm{Rel}(q^m)$. Let t^h be parent of s^e. From Lemma 4,

$$\mathbf{d}(p,s) < \mathbf{d}(p,q) \le \frac{c_c\tau}{\tau-1}\tau^m.$$

We have following cases:

Case 1: $\mathbf{d}(s,q) > \frac{3c_c\tau^2}{(\tau-1)^2}\tau^m$. By the triangle inequality,

$$\mathbf{d}(s,q) \le \mathbf{d}(p,s) + \mathbf{d}(p,q) < 2\mathbf{d}(p,q) \le \frac{2c_c\tau}{\tau-1}\tau^m,$$

which is a contradiction, because $\tau > 1$.

Case 2: $\mathbf{d}(s,q) \le \frac{3c_c\tau^2}{(\tau-1)^2}\tau^m$ and $e > m$. In this case, there exists a node s^g with $g \le m$, such that it satisfies both conditions of relatives. So, $s^g \in \mathrm{Rel}(q^m)$ and the algorithm correctly finds s^g. [2]

Case 3: $\mathbf{d}(s,q) \le \frac{3c_c\tau^2}{(\tau-1)^2}\tau^m$, $m \ge h$, and RestrictedNN does not find s^e. Let x^f be the highest ancestor s^e such that $f \le m$. Then, Lemma 4 implies

$$\mathbf{d}(x,s) \le \frac{c_c\tau}{\tau-1}\tau^f < \frac{c_c\tau}{\tau-1}\tau^m.$$

[2] g can be equal to $-\infty$, in this case we have a long edge from a node s in a level greater than m to the point s in level $-\infty$.

Also, by the triangle inequality,

$$\begin{aligned}\mathbf{d}(x,q) &\le \mathbf{d}(x,s)+\mathbf{d}(p,s)+\mathbf{d}(p,q)\\ &< \mathbf{d}(x,s)+2\mathbf{d}(p,q)\\ &< \frac{c_c\tau}{\tau-1}\tau^m+\frac{2c_c\tau}{\tau-1}\tau^m\\ &< \frac{3c_c\tau}{\tau-1}\tau^m < \frac{3c_c\tau^2}{(\tau-1)^2}\tau^m.\end{aligned}$$

Therefore, $x^f \in \mathrm{Rel}(q^m)$. Because $e' > \ell'$, RestrictedNN returns s^e as the nearest neighbor to p^ℓ, which is a contradiction. □

Theorem 9 *The time complexity of Algorithm 3 is* $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n\lg k)$.

Proof. From Theorem 7, T can be augmented with relatives in $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n)$ time. As a preprocessing step, we can maintain the lowest ancestor of all nodes in T in $O(n)$ time, which results a constant time access in the algorithm. By Lemma 3, the size of each list of relatives is $O(\rho^{\lg\frac{c_c\tau^2}{c_p(\tau-1)^2}})$. The time complexity of RestrictedNN is $O(\lg k)$, because the height of the subtree is $O(k)$. When Algorithm 3 is processing all nodes of level ℓ in T, for each point p in T', the level of high(p) is at most $\ell'+1$. So, FindNode requires $O(1)$ to return a node of T' in level ℓ' or $\ell'+1$. Since the number of edges in T is $O(n)$, finding relatives of dummy nodes will be done $O(n)$ times for the entire algorithm. Consequently, because $c_c \ge c_p > 0$, the total time complexity of the algorithm is $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n\lg k)$. □

5.2 Refining

Decreasing the scale factor is another useful operation for cover trees, and we call this operation *refining*. To refine a given cover tree T, each level ℓ in T is split into at most k levels $k\ell,\ldots,(k\ell+k-1)$ in T'. Note that by this division, a node p^ℓ in T may be appeared at most k times in levels $k\ell,\ldots,(k\ell+k-1)$ of T'. Similar to the coarsening operation, suppose that each point p in T' maintains high(p) which is the highest node associated to point p. Algorithm 4 describes the refining operation.

Theorem 10 *Algorithm 4 turns* $T \in \mathrm{CT}(\tau,c_p,c_c)$ *into* $T' \in \mathrm{CT}(\tau^{1/k},c_p,c_c)$.

Proof. First, we prove that the algorithm correctly finds parent of each node in T'. Note that q as the current parent of p^ℓ in T may not be the right parent of it in T' because there may exist a node x^ℓ such that $\mathbf{d}(p,x) < \mathbf{d}(p,q)$ and the level of x in T' be greater than the level of p in T'. In this case, p should be inserted as a child of x. To find the right parent of p^ℓ, we search its nearby nodes and select the closest node that satisfies the covering property with constant c_c.

Algorithm 4 Refining operation for a given cover tree

```
1: procedure Refining($T, k$)
2:     $T' \leftarrow \emptyset$
3:     Augment($T, \frac{3c_c\tau^2}{(\tau-1)^2}$)
4:     for all $p^\ell \in T$ in increasing order of level $\ell$ do
5:         Let $q^m = \mathrm{par}(p^\ell)$
6:         $p^{h'} \leftarrow \mathrm{high}(p)$
7:         if $p = q$ and $h' < k\ell$ then
8:             $p^{k\ell} \leftarrow$ FindNode(high($p$), $k\ell$)
9:             $q^{k(\ell+1)} \leftarrow$ FindNode(high($q$), $k(\ell+1)$)
10:        else if $p \ne q$ then
11:            $par \leftarrow q^m$
12:            $list \leftarrow \bigcup_{s^h \in \mathrm{Rel}(q^m)} \mathrm{ch}(s^h) \setminus \{q^\ell\}$
13:            for all $x^f \in list$ where $f = \ell$ do
14:                $x^{h'} \leftarrow \mathrm{high}(x)$
15:                if $\mathbf{d}(p,x) \le c_c\tau^{h'/k}$ and $\mathbf{d}(p,x) < \mathbf{d}(p,par)$ then
16:                    $par \leftarrow x^f$
17:            Find an $i$ such that $c_p\tau^{\ell+i/k} < \mathbf{d}(p,par) \le c_c\tau^{\ell+(i+1)/k}$
18:            $par^{k\ell+i+1} \leftarrow$ FindNode(high($par$), $k\ell+i+1$)
19:            $par^{k\ell+i} \leftarrow$ FindNode(high($par$), $k\ell+i$)
20:            $p^{k\ell+i} \leftarrow$ FindNode(high($p$), $k\ell+i$)
21:            Add $p^{k\ell+i}$ as a child of $par^{k\ell+i+1}$
```

Now, we show that those nodes of T that have appeared for the first time in level ℓ only required to be checked. Let x have appeared for the first time in level $h > \ell$. By the parent property of T, $\mathbf{d}(p,q) < \mathbf{d}(p,x)$, otherwise p should have x as its parent. Therefore, p cannot be closer to x than q and we can ignore x in the search process.

We also show that the right parent of p in T' is in the set of children of relatives of $q^{\ell+1}$. Let x^ℓ serve as the parent of p in T'. So, $\mathbf{d}(p,x) < \mathbf{d}(p,q) \le c_c\tau^{\ell+1}$. From the previous part, we know that x^ℓ has parent $y^{\ell+1}$ and $x \ne y$. By the triangle inequality,

$$\begin{aligned}\mathbf{d}(q,y) \le \mathbf{d}(q,p)+\mathbf{d}(p,x)+\mathbf{d}(x,y) &< 3c_p\tau^{\ell+1}\\ &< \frac{3c_c\tau^2}{(\tau-1)^2}\tau^{\ell+1}.\end{aligned}$$

It implies that $y^{\ell+1} \in \mathrm{Rel}(q^{\ell+1})$.

Now, we should find the right level $k\ell+i$ such that insertion of p in that level and as a child of x satisfies both packing and covering conditions with constants c_p and c_c, respectively. So, in this way we guarantee that these constants in T' will be the same as T. □

Theorem 11 *Algorithm 4 has time complexity* $O((\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau}+k)n)$.

Proof. By Theorem 7 we can augment T in $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} n)$ time. From Lemma 3 and Lemma 1, number of nearby nodes to p^ℓ is $O(\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau})$. Note that for each node $p^\ell \in T$ in this algorithm, level of high(p) in T' is at most $k(\ell+1)$. Therefore, FindNode requires $O(k)$ to return a node which is in a level between $k(\ell+1)$ to $k\ell$ in T'. Also, finding the right interval i requires $O(\lg k)$. Therefore, the time complexity of the refining algorithm is $O((\rho^{\lg(\frac{c_c\tau}{c_p(\tau-1)})^2\tau} + k)n)$. □

6 Conclusion

In this paper, we add an easy to implement condition to cover trees and we show that a cover tree with a large enough scale factor is a net-tree. We also proposed a linear time algorithm to augment nodes of a cover tree with relatives. Furthermore, we present two linear-time algorithms to transform a cover tree to a coarser or finer cover tree. In fact, these two operations are useful to trade-off between the depth and the degree of nodes in a cover tree.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[2] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, 2006.

[3] L. K. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[4] R. Cole and L.-A. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, pages 574–583, 2006.

[5] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[6] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. *Comput. Geom. Theory Appl.*, 35(1-2):2–19, Aug. 2006.

[7] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 534–, 2003.

[8] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[9] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pages 741–750, 2002.

[10] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798–807, 2004.

[11] J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991.

[12] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

On the Triangulation of non-fat Imprecise Points

Vahideh Keikha* Ali mohades * Mansoor Davoodi†

Abstract

In this paper, we address the problem of computing a triangulation of imprecise points modeled by non-fat regions posed by Van Kreveld *et al.* [SIAM J. Comput(39), 2990-3000 (2010)]. In particular, we study the problem of preprocessing a set of n line segments in the plane so that if one point per region is specified with precise coordinates, a triangulation of the points can be computed in $o(n \log n)$ time. We first model the points with perpendicular line segments and show if imprecise points have uniform distribution in their corresponding bounding box, in $O(n \log n)$ preprocessing time a triangulation of any exact set of points can be computed in expected linear time. Although, we show even computing an arbitrary triangulation in this model can take $\Omega(n \log n)$ time in the worst case. Also some related lower bound proofs are provided at the end.

1 Introduction

In recent years there has been a marked attention on the use of exact data, to the use of uncertain data as the input of some of the geometric algorithms. It is because we widely have to work with data obtained from devices for obtaining certain information in applications of the real world. In this situation, an inherent imprecision seems to be unavoidable. A very common setting in the concept of imprecision of data is region based models: a set of planar regions are given and each of them represents an estimate about one of the input points. In this model there is an equal chance of point existence in everywhere of its corresponding region and also we know the corresponding region of each point.

In this paper, we study the problem of triangulation of a set L of n imprecise points modeled as zero-area regions and each of which can intersect with $O(n)$ other regions. There is a fair amount of studies that assume the input regions have non-zero area. Löffler and Van Kreveld [8] considered the problem of computing the largest and smallest possible convex hull of a set of imprecise points that modeled by line segments and squares and measured by perimeter and by area. The running times of their algorithms varied from $O(n \log n)$ to $O(n^{13})$. Some of their results on the area of the convex hull has been improved by Ju *et al.* [6]. In [3] Ezra and Mulzer studied the problem of computing the convex hull of a set of imprecise points that modeled by lines and presented an algorithm with quadratic preprocessing time and space and $O(n\alpha(n) \log^* n)$ expected query time. They also presented that for the problem of computing the convex hull, the closest pair and the sorting problem on a given query set of exact points, preprocessing of the input regions that include a set of lines is unlikely to decrease the query time to $o(n \log n)$.

With a translation to imprecise context, there are also multiple studies related to the non-fat imprecise points, e.g., Goodrich and Snoeyink [4] studied the problem of finding a point on each of the given parallel line segments such that the resulting point set is in convex position. If there exists a solution, their algorithm can find it in $O(n \log n)$ time, and in $O(n^2)$ time the minimum possible area or perimeter of the solution can be computed. Mukhopadhyay *et al.* [12] studied the problem of computing the smallest possible area convex polygon that intersects a set of parallel line segments in $O(n \log n)$ time, and Rappaport [16] considered the smallest perimeter polygon transversal of a set line segments in a constant number of directions, after spending $O(n \log n)$ time cost.

Also some of the efforts have been made for computing triangulation of a set of imprecise points. Held and Mitchell [5] were the first to study the problem of triangulation of a set of imprecise points modeled by disjoint disks of uniform size. After spending $O(n \log n)$ preprocessing time they computed an arbitrarily triangulation of a set of query points in $O(n)$ time. With some constraints, their results can be extended to convex regions with non-zero area. Van Kreveld *et al.* [7] generalized their results by finding an arbitrary triangulation of a set of imprecise points modeled by polygonal regions. Their algorithm can be extended for a set of line segments, under the constraint that each region has constant number of intersections with the other regions (in this paper, we mainly focus on removing this constraint). Also some studies are concerned with preprocessing a set of imprecise points mainly modeled by disks in $O(n \log n)$ time, so that if one point per region is specified with precise coordinates, Delaunay triangu-

*Laboratory of Algorithms and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of technology, Tehran, Iran, va.keikha@aut.ac.ir, mohades@aut.ac.ir

†Department of Computer Sciences and Information Technology, Institute for Advances Studies in Basic Sciences, Zanjan, Iran, mdmonfared@iasbs.ac.ir

lation of the points can be computed efficiently[1, 9]. In all of the studies computing a triangulation of imprecise points, the imprecision regions are fat and intersection between the regions is a challenging problem. In fact, the extension of previous algorithms work only to a class of fat shapes with constrained number of intersection between the regions. So an interesting question is what can be done for more general regions. As said in previous studies, there is no hope if we have zero-area regions with many intersection between the regions. But there is no lower bound proof for many of the problems in these cases. Also some situations may exist that one can do better. Motivated by these questions we studied the following problem:

Problem definition. Let $L = \{l_1, l_2, \dots, l_n\}$ be a set of n line segments in the plane, each of which represents the possible location of an imprecise point. We wish to preprocess them in such a way that whenever an exact set of points are arrived (one point per region) one can compute a triangulation of them in $o(n \log n)$ time.

Our results. We will give a lower bound proof for the problem stated above, although we show if the regions include perpendicular line segments and have uniform distribution in their corresponding bounding box, after $O(n \log n)$ preprocessing time and using $O(n)$ space, one can compute a triangulation of the query points with high probability in expected linear time. Furthermore, we show that even computing an arbitrary triangulation in this model can take $\Omega(n \log n)$ time in the worst case. In fact, we will show no preprocessing can result in computing a triangulation of the query points more quickly than from scratch. Finally, some related lower bound proofs are provided at the end.

Assumption 1. Our computational model is unit cost Ram model, where every operation on real numbers takes constant time, including $\lfloor . \rfloor$ operation. We also assume exact computations.

Assumption 2. Our uniform distribution assumption is one in which by moving a square [1] in the corresponding bounding box of the regions, the same order of regions have been intersected all the times.

It should be noted that in case of uniform distribution of perpendicular regions, the query points do not necessarily realize the uniform distribution in the bounding box of the regions (think of the problem in $1D$ space), otherwise, one can subdivide the bounding box into a grid whose size is chosen to make most grid cells contain a constant number of points, construct simple triangulation graphs in the grid cells, and the totally triangulation graph can be constructed in a substantially sub-linear (e.g., $O(\sqrt{n} \log n)$) time with high probability. Also, this assumption is not unrealistic as it is possible a set of uniformly distributed points having some

[1]With the sidelength larger than the sidelength of smallest square in intersect with at least one region.

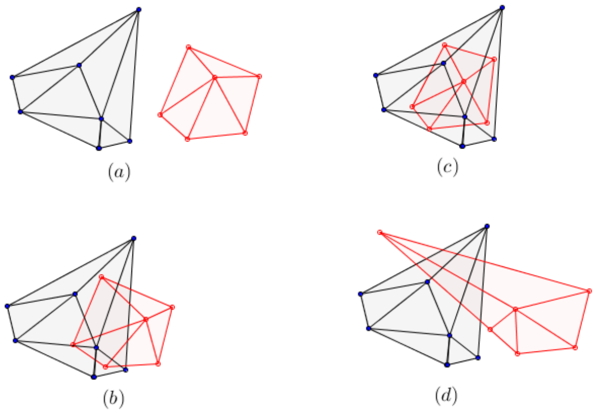


Figure 1: Possible cases for intersection of two triangulation graphs.

imprecisions during their measuring procedure. The Delaunay triangulation also takes $\Omega(n \log n)$ time in the worst case in this model, since Seidel [17] showed that even if the sorted order of a set of points is given, computing the Delaunay triangulation requires $\Omega(n \log n)$ time. So even if the input regions contain only parallel line segments (or lines), because any preprocessing just yields the sorted order of points, computing the Delaunay triangulation of an exact set of points requires $\Omega(n \log n)$ time in the worst situation.

2 Triangulation of Imprecise Points

We first define some notation that we will use in subsequent sections. For a set P of points, $CH(P)$ and $T(P)$ respectively represents the convex hull and the triangulation of P, for a given point Q in $\mathbb{R}^2$, Q^x and Q^y denote the x and y-coordinate of Q. When we use the index h (v) for an object, we mean that object is a horizontal (vertical) line segment or belongs to a horizontal (vertical) line segment. Also we use the word segment instead of line segment.

At first assume that L contains only parallel segments. Without loss of generality, let the segments be vertical. Note that any preprocessing can yield only the sorted order of points. So whenever the exact set of y coordinates are arrived, compute a x-monotone chain of sorted points by x-order in linear time. we can compute the convex hull of these points in linear time by e.g., applying a variant of Melkman's algorithm [11]. The pockets formed between the monotone chain and its convex hull are also y-monotone and can be triangulated in linear time. So a triangulation of the query points in the case of parallel segments can be computed in linear time. We can store this structure in a DCEL in $O(n)$ space.

Lemma 1 *Let $L = \{l_1, l_2, \dots, l_n\}$ be a set of n parallel segments in the plane, each of which corresponds to an*

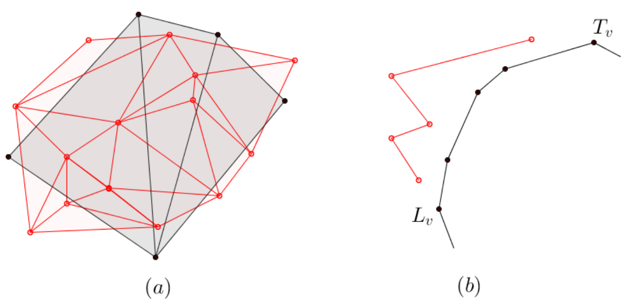


Figure 2: An example. (a) The open points located outside $CH(V)$ show the four possible regions containing the elements of set O. (b) The segments connecting open points show the monotone chain of northwest area points of O and its corresponding monotone chain of $CH(V)$ that starts in L_v and ends with T_v.

imprecise point. A triangulation of an exact set of points (one point per region) can be computed in $O(n)$ time and space after $O(n \log n)$ preprocessing time.

The problem we first discuss in this section is the following:
Let $L = \{l_{v_1}, l_{v_2}, \ldots, l_{v_{n_v}}, l_{h_1}, l_{h_2}, \ldots, l_{h_{n_h}}\}$ be a set of n_v vertical and n_h horizontal segments where $n_v + n_h = n$, each segment represents an imprecise point and $n_v, n_h \in \Omega(n)$. We wish to preprocess them in such a way that one can compute a triangulation of an exact given set of points (one point per region) with high probability in expected linear time.

2.1 Preprocessing

We will show that if the imprecise points modeled by perpendicular segments have uniform distribution in their corresponding bounding box, called $\mathcal{B}$, it is possible to find a triangulation of query points with high probability in expected linear time. As we will show later in this paper, even computing an arbitrary triangulation of query points in this model can take $\Omega(n \log n)$ time in the worst case.
In preprocessing step, we first sort the vertical and horizontal segments separately taking $O(n \log n)$ time, the sorted lists is stored separately in DCEL structures. Also we construct a regular $m \times m$ grid on $\mathcal{B}$. We set m to $\sqrt{n}$. Without loss of generality assume the lowest leftmost corner of the grid has the coordinates $(0, 0)$. For each grid cell $\square$ we compute the covered x and y ranges. These values determine a distinct hash key for each $\square$. So every $\square$ of $\mathcal{B}$ has a unique key; indeed, let Q be any point in $\square$, and consider the pair of integer numbers $key\square = key(Q) = (\lfloor Q^x/ \text{ length of } \square \rfloor, \lfloor Q^y/\text{width of } \square \rfloor)$. Obviously, only points inside $\square$ are going to be mapped to key $\square$. So by constructing a linear size hash structure, a point location operation can be performed in constant time.

2.2 Triangulation Algorithm

In query stage, a set $P = \{p_{v_1}, p_{v_2}, \ldots, p_{v_{n_v}}, p_{h_1}, p_{h_2}, \ldots, p_{h_{n_h}}\} \subset \mathbb{R}^2$ of exact points are given, where $p_{v_i} \in l_{v_i}$, $p_{h_i} \in l_{h_i}$, and put $V = \bigcup p_{v_i}$ and $H = \bigcup p_{h_i}$. Using Lemma 1, the triangulation of each of V and H can be computed in linear time. All the cases of possible intersection of two triangulation graphs are depicted in Figure 1. Also by the intersection detection algorithm in [13] (chapter 7) and the constructed DCEL structures in previous step, we can determine the witnessed case and also the points in intersection area (if any) in linear time. We will show how we handle each of the illustrated cases in the following:

Case a. *There is no intersection between $T(V)$ and $T(H)$.*

If there is no intersection between $T(V)$ and $T(H)$, there is also no intersection between $CH(V)$ and $CH(H)$. So the problem reduces to computing $T(V)$ and $T(H)$ using Lemma 1 and triangulation of the region constructed by merging $CH(V)$ and $CH(H)$. Since we want to find just a triangulation of the points, a simple idea from the algorithm that find the tangents between two disjoint convex hulls [15] can be used to triangulate the area between the hulls.

Case b. *There are some points from one set in the area intersected by the convex hull of other set.*

In this case, we fix the constructed triangulation of one set and handle separately the points of other set that locating both inside and outside of the convex hull of first set. It should be noted that if we first compute the triangulation of both sets, it may happen that all the triangles of one set intersect with the triangles of other set (each triangulation is a planar graph, then we may have two graphs with $O(n)$ pairwise crossing edges). To the best of our knowledge there is no planarization algorithm for handling such situation in linear time (see, e.g., [2]).
Without loss of generality, assume we compute and fix $T(V)$ and add set H to the existing triangulation graph. Clearly the points belong to H can located both inside and outside $CH(V)$. We call I the set of elements of H that located inside $CH(V)$, and O the elements in $H \setminus I$. We first explain the handling of set O. Knowing the elements of I, set O can also be computed in linear time. Firstly find the *topmost*, *rightmost*, *bottommost* and *leftmost* points of $CH(V)$, that are the axis-extreme points of V and denote them with T_v, R_v, B_v and L_v, respectively. These four points divide the $CH(V)$ into four convex (and also monotone in y-direction) chains that are the *northwest* chain, the *southwest* chain, the *southeast* chain and the *northeast* chain. The northwest convex chain, for example, will be the chain that starts

from L_v and ends with T_v (see Figure 2(b)), etc.

Also with a linear scan of O we can classify its elements into four disjoint groups according to their coordinates, as in Figure 2(a):

The *northwest area* points: the points that have x-coordinate smaller than T_v^x and y-coordinate larger than L_v^y and located in the northwest area of the $CH(V)$. These points belong to set H that their supporting segments have been sorted in a DCEL in preprocessing step. Then one can find a y-monotone chain (possibly a point or segment) by connecting these points in bottom-to-top direction.

The *southwest area* points: the points that have x-coordinate smaller than B_v^x and y-coordinate smaller than L_v^y and located in the southwest area of the $CH(V)$.

The *southeast area* points: the points that have x-coordinate larger than B_v^x and y-coordinate smaller that R_v^y and located in the southeast area of the $CH(V)$.

The *northeast area* points: the points that have x-coordinate larger than T_v^x and y-coordinate larger than R_v^y and located in the northwest area of the $CH(V)$.

So we have two monotone chains in each of the four regions. Although all the points of O are located outside the $CH(V)$, but each monotone chain of O may intersect with its corresponding monotone chain of $CH(V)$. We use a bottom-up sweep line that handles the intersection points by constructing some small monotone polygons. For brevity of explanation we consider the triangulation procedure of the northwest region (similarly for other regions) according to Figure 3. We start sweeping up the events from L_v to point t that is the topmost point of O in the northwest region, and checking for and handling the potential intersections between the two northwest monotone chains. An event consisting of both the points of each of chains and the intersection points (if any). With a slight abuse of notation, we use C_o both for the chain belongs to northwest area points of set O and the list of segments in C_o, and C_v both for the convex chain belongs to $CH(V)$ and the list of segments in C_v.

In each step, we check two segments from each of the chains (from two preliminary constructed list of segments on each chain in bottom-up direction) for potentially intersection. Whenever we find an intersection between two segments like $\ell_o \in C_o$ and $\ell_v \in C_v$, we ignore ℓ_o that causes an intersection and insert ℓ'_o and ℓ''_o into C_o instead, for saving the connectivity of C_o. So ℓ_v would be a segment in C_v that causes ℓ_o goes into $CH(V)$. Also let $\tilde{\ell_v} \in C_v$ be the segment that causes ℓ_o goes out of $CH(V)$ (possibly $\tilde{\ell_v} = \ell_v$, if the intersection happens at an endpoint). If we assume a bottom-to-top direction of C_o and C_v, ℓ'_o is a segment that connects the firstpoint of ℓ_o to the endpoint of ℓ_v. Also ℓ''_o is a segment that connects the firstpoint of $\tilde{\ell_v}$ to the endpoint

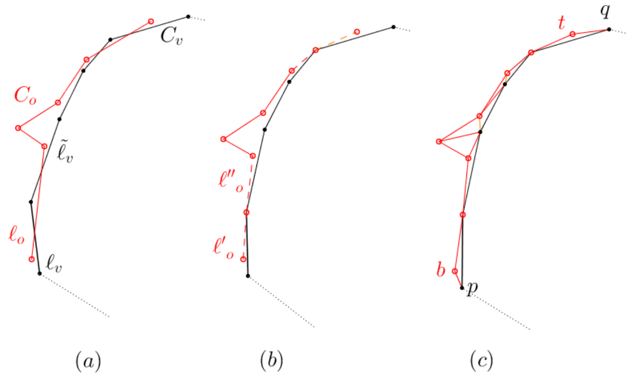


Figure 3: (a) Two monotone chains in northwest region. (b) The construction of small monotone polygons using two intersected monotone chains. (c) A triangulation of the nortwest region.

of ℓ_o. Then we will continue with the next segment of C_o and $\tilde{\ell_v}$, and proceed the lists until we find the next two intersecting segments.

We repeat this procedure until we reach to the topmost point of C_o. Then connect b and t that are the first and last point of C_o to two appropriate vertices of C_v for closing the boundary of the first and last constructed small monotone polygons. Let $p, q \in C_v$ be such vertices, then they are respectively two arbitrary visible vertices from b and t with respectively smaller y and larger x-coordinates than b and t (care about choosing p, q in such a way that don't violating the planarity of the triangulation graph). Now we can triangulate the simple monotone polygons constructed between some parts of C_v and C_o in linear time.

We also do the same procedure for the remaining three other regions. If there was no intersection between the chains, we just need to connect the points b and t to two points like p and q as said above. In this case, just one monotone polygon has been constructed. The procedure of finding the intersection points is similar to the algorithm presented in [14] that works totally in $O(n)$ time.

Now we have a triangulation graph with non-convex outer boundary. We compute a maximal triangulation of this set in linear time using e.g., the idea of the algorithm that finds the convex hull of a simple non-convex polygon [10]. The algorithm ends up when we walk around the polygon and triangulate all the pockets making the polygon non-convex. It takes $O(n)$ time totally. Finally, we proceed with a right triangulation of $O \cup V$.

We should also handle the set I. in this situation we have a set of points located in some of the faces of a triangulation graph. During the past years, a lot of efforts have been made to do efficiently the point location operation in a triangulation geometrical graph. With the best of our knowledge, there is no linear time algorithm for handling such situation. So we try to find the containing triangles of the points of set I in expected linear time, and proceed by splitting each triangle into three new triangles. During the construction of $T(V)$, whenever a new triangle is created, knowing its three vertices we can locate the containing grid cells in constant time. In fact, we use these three grid cells to compute all the cells intersected by this triangle. We should store all the involved cells for each triangle in an adjacency list structure. Likewise, during the construction of above structure, we also store all the triangles involved by a cell. We will demonstrate the first structure can be constructed in expected linear time, likewise the second structure constructed during the construction of the first one and takes expected linear time. So for each point $p \in I$, we can determine the cell containing it and also all the triangles involved by this cell in expected constant time, determine the containing triangle of p and then easily split it into three new triangles. We will show that each triangle is contained within the union of expected constant number of cells and vice versa. It is important because just in this case we will totally spend expected linear time. We discuss later that all the times it is the case with high probability.

Also, the same procedure of *case b* can be applied for other possible cases, e.g., in *case c*; we do triangulation likewise the procedure done in *case b* knowing $I = H$ and $O = \emptyset$ (similarly in *case d*, $O = H$ and $I = \emptyset$).

2.3 Running Time Analysis

In this section, we show the triangulation process with high probability can be down in expected linear time. For this purpose we should demonstrate that all the above procedures and the required updates for the data structures will take linear or expected linear time totally. The linearity of some of the procedures has been explained above. One of the challenges need to be addressed is that if each triangle is contained within the union of a constant number of cells and vice versa? We will show each triangle is contained within the union of a constant number of cells and each cell belongs to constant number of triangles with relatively high probability. We first consider the case of each cell belongs to many ($O(n)$) triangles and give a bound on the number of such cells. Remember the regions have uniform distribution in their corresponding bounding box $\mathcal{B}$ and we construct a regular $\sqrt{n} \times \sqrt{n}$ grid on $\mathcal{B}$. Assume an $n \times n$ matrix M with only 0 and 1 values. Each row represents one cell of the grid and we have n (in fact n' that $n' \in O(n)$) columns representing each of the constructed triangles. For each $\square_i$ belongs to row i ($1 \leq i \leq n$) if the triangle belongs to column j ($1 \leq j \leq n$) intersected by $\square_i$, the M_{ij} value would be 1 and is 0 otherwise. In both problems stated above, the expected number of triangles involved by a cell, and the expected number of cells intersected by a triangle reduces to the problem of finding the expected number of ones in a row or column of M (with independent indicator random variables). So the expected number of ones in a row of M would be

$$X = \sum_{i=1}^{n} x_i, P(x_i = 1) = \frac{r_i}{n}.$$

Each x_i would have a 0 or 1 value and X is the random variable of the number of triangles involved by grid cells. Also, r_i is the number of cells in row i that valued by 1.

$$E(X) = E(x_1 + x_2 + \ldots + x_n) = \frac{r_1 + r_2 + \ldots + r_n}{n}$$

Note that r_i's ($1 \leq i \leq n$) have not the same value due to not uniformly distribution of the area of the triangles. We call T the value of $E(X)$ that shows the ratio between the number of all the involved cells to the total number of triangles.

$T > \ln n$ (using Chernoff's inequality):

$$P(X > 5T) \leq e^{-4(\ln 4)T} \leq e^{\frac{-4(\ln 4)(\ln n)}{2}} = \frac{1}{n^{2\ln 4}} \leq \frac{1}{n^3}$$

Also P(existence of a cell intersected by more than $5T$ triangles) $\leq n\left(\frac{1}{n^3}\right) = \frac{1}{n^2}$.

Now we should show that the required updates for the data structures take linear or expected linear time totally. In each step of the procedure for handling the *case b*, we try one point p from set I, and in constant time we can find the cell contains it. But also the triangles that involved this cell should be found efficiently. Assume the containing triangle of p is $\triangle_p$. By splitting $\triangle_p$ using p, three new triangles should be added to the existing structures and $\triangle_p$ need to be removed. So we only need to check the cells intersected by $\triangle_p$. Also the number of cells may involve by three new created triangles is same as $\triangle_p$. Likewise the adjacency list of the corresponding cells should be updated. In each of the corresponding cell's list, exactly one triangle should be removed and three new triangles should be added and this can be done easily in constant time. Also, the DCEL contained the triangulation graph should be updated by inserting constant number of edges.

From above we know that for each grid cell that intersected by a constant number of triangles (and vice versa), updating the data structures in each step takes constant time and would be linear totally. But the cost of handling a cell that intersected by $O(n)$ triangles would be $O(n)$ time and from above, we know that the number of such cells would be $O(\ln n)$ or more with probability less than $\frac{1}{n^2}$, in which the time of the algorithm would exceed $O(n)$ with probability less than $\frac{1}{n^2}$, therefore we have an expected linear time algorithm with probability higher than $1 - \frac{1}{n^2}$. Regarding these results, the adjacency list of triangles and cells can be constructed and also updated with probability higher than $1 - \frac{1}{n^2}$ in expected linear time. Clearly as all the other procedures can be done in linear time in the worst case, the triangulation can be done in linear time with probability higher than $1 - \frac{1}{n^2}$. Now we can write the following theorem:

Theorem 2 *Let $L = \{l_{v_1}, \ldots, l_{v_{n_v}}, l_{h_1}, \ldots, l_{h_{n_h}}\}$ be a set of n_v vertical and n_h horizontal segments where $n_v + n_h = n$ and $n_v, n_h \in \Omega(n)$. After $O(n \log n)$ preprocessing time, a triangulation of an exact set of query points (one point per region) can be computed in expected linear time.*

3 Lower Bounds

In this section we represent for a set L of n imprecise points modeled as segments or lines, computing an arbitrary triangulation of an exact set of query points, any preprocessing is unlikely to decrease the query time to $o(n \log n)$. Of course sometimes one can obtain better bounds if each point lies on a fat region with some constraints on the intersection between the given input regions [5, 7].

Let $\mathcal{S}$ be a set of exact points in the plan. We first show even if we have the sorted order of a sub-set of $\mathcal{S}$ in one direction in the plan, and for the remaining points of $\mathcal{S}$ we have the sorted order in the other direction, computing a triangulation of $\mathcal{S}$ takes $\Omega(n \log n)$ in the worst situation. In this case, we do reduction from dictionary problem with multiple queries. We know that there is an $\Omega(n \log n)$ lower bound for this problem in $1D$ in the algebraic computational tree model. The multiple query version of this problem is given in the following: Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set in $\mathbb{R}$ with $a_1 < a_2 < \ldots < a_n$ and for an arbitrary query set $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{R}$, the problem of determining whether there exists a j for each b_i, $1 \le i \le n$ such that $a_j \le b_i \le a_{j+1}$, and report it [17].

Theorem 3 *Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of points ordered in terms of x, and $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\} \subset \mathbb{R}^2$ be a set of points ordered in terms of y, even computing an arbitrary triangulation of $\mathcal{P} \cup \mathcal{Q}$ takes $\Omega(n \log n)$ time in the worst case.*

Proof. We call this problem Trinagulation with Partially Ordered Input (TPOI), and we do reduction from the $1D$ dictionary problem with multiple queries. Assume we have some algorithm that solves the problem of computing some triangulation for every set $\mathcal{P}$ and $\mathcal{Q}$ with condition stated in theorem and takes $\mathcal{F}(n)$ in the worst case. We will show this algorithm with an additional time expenditure of $O(n)$ can solve the $1D$ dictionary problem, then $\mathcal{F}(n)$ should be $\Omega(n \log n)$.

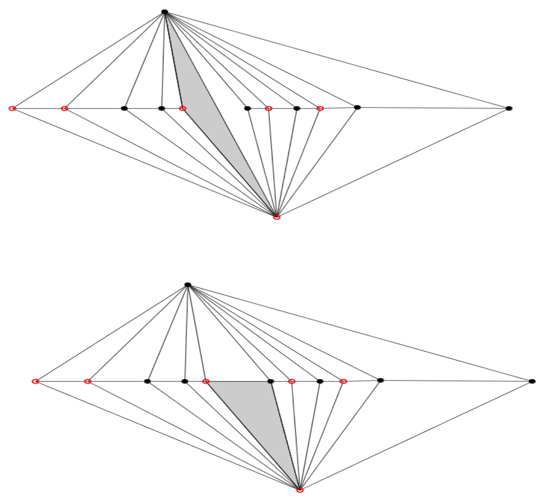


Figure 4: Two possible triangulations for $P \cup Q$

Let we are given a set $A = \{a_1, a_2, \ldots, a_n\}$ sorted in terms of values as the dictionary and the set $B = \{b_1, b_2, \ldots, b_n\}$ as the query set. We can obtain the set $\mathcal{P} = \{(a_1, 0), (a_2, 0), \ldots, (a_p, 1), \ldots, (a_n, 0)\} \subset \mathbb{R}^2$ from A in linear time such that contains $n-1$ collinear points and 1 arbitrary point having a larger y-coordinate. Note that there is only one possible triangulation for set $\mathcal{P}$. Also form the set $\mathcal{Q} = \{(b_1, 0), (b_2, 0), \ldots, (b_q, -1), \ldots, (b_n, 0)\} \subset \mathbb{R}^2$ from B in linear time such that contains $n-1$ collinear points and 1 arbitrary point having a smaller y-coordinate. Note that one can easily compute the sorted order of $\mathcal{Q}$ according to y. There is also one possible triangulation for set $\mathcal{Q}$.
Then we run some algorithm that can solve the problem stated in theorem and compute the triangulation of $\mathcal{P} \cup \mathcal{Q}$ in time $\mathcal{F}(n)$. There exists only two possible triangulations, as in Figure 4. We then find $z = min_x(\mathcal{P}, \mathcal{Q})$ in linear time and run a BFS algorithm from z that takes $O(n)$ totally (due to planarity of triangulation graph), and ignore a_p and b_q. Afterwards, we just need to locate b_q in the dictionary and retain the influence of a_p that ignored previously. So we answer the dictionary problem in time $\mathcal{F}(n) + O(n)$. Clearly, $\mathcal{F}(n)$ must be $\Omega(n \log n)$. It is easy to see that this reduction has a linear running time. □

It can be concluded that this kind of additional input information we consider in Theorem 3 does not reduce the $\Omega(n \log n)$ complexity of the triangulation problem, in contrast to some of the related geometric problems, e.g., computing the planar convex hull. We thus conclude:

Corollary 4 *Let $V = \{p_1, p_2, \ldots, p_{n_v}\} \subset \mathbb{R}^2$ be a set of points belong to vertical segments ordered in terms of x, and $H = \{q_1, q_2, \ldots, q_{n_h}\} \subset \mathbb{R}^2$ be a set of points be-*

long to horizontal segments ordered in terms of y, where $n_v + n_h = n$ and $n_v, n_h \in \Omega(n)$. Even computing an arbitrary triangulation of $V \cup H$ takes $\Omega(n \log n)$ time in the worst case.

Obviously the same lower bound holds for the case of perpendicular lines instead of segments. Now we show for a set L of imprecise points, if a triangulation of a query set P can be computed in $o(n \log n)$ time, after preprocessing L, it would be possible to solve an instance of TPOI problem in the same time, that now we know this can not happen.

Theorem 5 *Let $L = \{l_1, l_2, \dots, l_n\}$ be a set of lines in the plane and each of which is corresponding region of an imprecise point, and let $P = \{p_1, p_2, \dots, p_n\}$ be a set of query points where $p_i \in l_i$, even computing an arbitrary triangulation of P (after preprocessing L) takes $\Omega(n \log n)$ time in the worst case.*

Proof. We do reduction from the TPOI problem. As it was widely observed that if an algorithmic problem with a set of real numbers as the input has a lower bound, some of the special cases of the problem, e.g., the problem with natural numbers as the given input, holds also the same lower bound. Now assume a constraint version of the TPOI problem that is defined as that which each part of the elements of the input that is ordered in terms of x (also for y), can achieve only an integer value between 1 and n_v (for y-coordinate this value would be between 1 and n_h), but the other parts of each of the elements can achieve an arbitrary value, and let $\mathcal{P} = \{p_{v_1}, p_{v_2}, \dots, p_{v_{n_v}}, p_{h_1}, p_{h_2}, \dots, p_{h_{n_h}}\} \subset \mathbb{R}^2$ be such an instance, where $n_v + n_h = n$ and $n_v, n_h \in \Omega(n)$. For $i = 1,\dots,n_v$, let l_i be the line $l_i : p^x_{v_i} = i$, and for $j = i+1,\dots,n$, let l_j be the line $l_j : p^y_{h_i} = i$ for $i = 1,\dots,n_h$ and let $L = \{l_1, l_2, \dots, l_n\}$. Every point of set $\mathcal{P}$ must lie on one of the lines of L. We put $P = \mathcal{P}$, now set $\mathcal{P}$ acts as a query set of L. Then we can find the triangulation of P (and every set of points that lie on the lines of the grid defined by L, as in Figure 5) in $o(n \log n)$ time using the information achieved in the preprocessing of L. It is now easy to see that the output yields the triangulation of $\mathcal{P}$ in $o(n \log n)$ time that it can not happen, and that this reduction has a linear running time. $\square$

Obviously the proof works in the case of the regions including segments instead of lines, with a limitation on the range of the coordinates of points belonging to $\mathcal{P}$.

4 Discussion

In this section, we first consider the case where a constant number of perpendicular segments violating the uniform distribution assumption of the regions (there

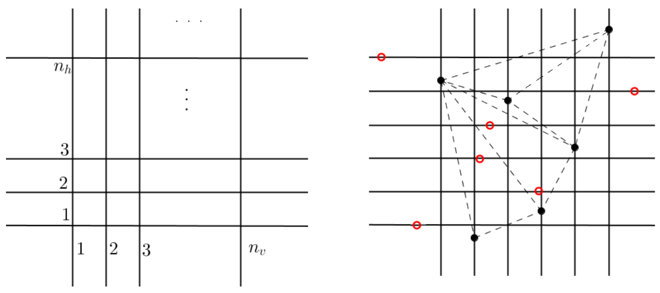


Figure 5: Set L and a triangulation of sub-set of P that is ordered in terms of x.

is a significant difference between the lengths of some of them). In preprocessing phase, we can rebuild the bounding box $\mathcal{B}$ such that the regions satisfy the uniform distribution assumption. The algorithm could be applied for handling of such situations, except that $\mathcal{B}$ is built on the set of supporting lines of the segments that previously constructed the grid cells. Similar to the mentioned procedure in Sect 2.2, we fix the constructed triangulation of set V and handle the points of set H afterwards. But now there are three different types of triangles in $T(V)$ according to the position of the vertices of each of the triangles: all the three vertices located inside or on edges of the grid boundary, the three vertices located both inside and outside of the grid boundary and all the three vertices located outside of the grid boundary. We need only to consider the second case. In this situation there exist some triangles that involved by both of bounded and unbounded grid cells simultaneously. But the portion that belongs to the unbounded grid cells can not contain any point from the set H, and there is no need for doing a point location anyway. For the portion belongs to the bounded grid cells we do the point location operation as said above. So the mentioned time analysis can be considered again.

Finally it should be noted that the lower bound proofs related to perpendicular objects do not work for the cases either n_v or n_h doesn't belong to $\Omega(n)$.

References

[1] O. Devillers. Delaunay triangulation of imprecise points: Preprocess and actually get a fast query time. *J. Comput. Geom.*, 2(1): 30–45, 2011.

[2] D. Eppstein, M. T. Goodrich and D. Strash. Linear-time Algorithms for Geometric Graphs With Sublinearly Many Edge crossings. *SIAM J. Comput*, 39(8): 3814–3829, 2010.

[3] E. Ezra and W. Mulzer. Convex hull of points lying on lines in o(nlogn) time after preprocessing. *Comput. Geom. Theory Appl.*, 46(4): 417–434, 2011.

[4] M.T. Goodrich and J. Snoeyink. Stabbing parallel segments with a convex polygon. *Comput. Vis. Graph. Image Process*, 49:152–170, 1990.

[5] M. Held and J. Mitchell. Triangulating input-constrained planar point sets. *Inf. Process. Lett.*, 109(1): 54–56, 2008.

[6] W. Ju, J. Luo, B. Zhu and O.Daescue Largest area convex hull of imprecise data based on axis-aligned squares. *J. Comb. Opt*, 26(4):832–859, 2013.

[7] M. Kreveld, M. Löffler and J. Mitchell. Preprocessing Imprecise Points and Splitting Triangulations. *SIAM J. Comput*, 39(7):2990–3000, 2010.

[8] M. Löffler and M. Kreveld. Largest and Smallest Convex Hulls for Imprecise Points. *Algorithmica*, 56(2):235–262, 2008.

[9] M. Löffler and J. Snoeyink. Delaunay Triangulations of Imprecise Points in Linear Time after Preprocessing. *Comput. Geom. Theory Appl.*, 43(3): 234–2420, 2010.

[10] D. Mccallum and D. Avis. A linear algorithm for finding the convex hull of a simple polygon. *Inf. Process. Lett.*, 9:201–206, 1997.

[11] A. Melkman. On-line construction of the convex hull of a simple polyline. *Inf. Process. Lett.*, 25:11–12, 1987.

[12] A. Mukhopadhyay, C. Kumar, E. Greene and B. Bhattacharya. On intersecting a set of parallel line segments with a convex polygon of minimum area. *Inf. Process. Lett.*, 105(2):56–64, 2008.

[13] J. O'Rourke. Computational Geometry in C. *Cambridge University Press*, New York, NY, 1998.

[14] S.C. Park and H. Shin. Polygonal chain intersection. *Computers Graphics*, 26:341–350, 2002.

[15] F.P. Preparata and S. J. Hong. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. *Communication of ACM*, 20(2):87–93, 1977.

[16] D. Rappaport. Minimum polygon transversals of line segments. *Int. J. Comput. Geom. Appl.*, 5(3):243–256, 1995.

[17] R. Seidel. A Method for Proving Lower Bounds for Certain Geometric Problems. *Technical Report TR84-592, Cornell University*, Ithaca, NY, USA, 1984.

On the Precision to Sort Line-Quadric Intersections

Michael Deakin Jack Snoeyink*

Abstract

To support exactly tracking a neutron moving along a given line segment through a CAD model with quadric surfaces, this paper considers the arithmetic precision required to compute the order of intersection points of two quadrics along the line segment. When the orders of all but one pair of intersections are known, we show that a resultant can be used to determine the order of the remaining pair using only half the precision that may be required to eliminate radicals by repeated squaring. We compare the time and accuracy of our technique with converting to extended precision to calculate roots.

1 Introduction

In this work, we are concerned with ordering the points of line-quadric intersections in 3 dimensions, where the inputs are representable exactly using w-bit fixed-point numbers. We will actually use floating point in storage and computation, but our guarantees will be for well-scaled inputs, which are easiest to describe as fixed-point numbers. A *representable point* q or *representable vector* v is a 3-tuple of representable numbers (x, y, z). The line segment from point q to $q + v$ is defined parametrically for $t \in [0, 1]$ as $\ell(t) = q + tv$; note that there may be no representable points on line ℓ except its endpoints (and even $q + v$ may not be representable, if the addition carries to $w + 1$ bits.)

A quadric is an implicit surface defined by its 10 representable coefficients,

$$Q(x, y, z) = q_{xx}x^2 + q_{xy}xy + q_{xz}xz + q_x x + \dots \\ + q_{zz}z^2 + q_z z + q_c = 0.$$

For more accuracy, we can allow more precision for the linear and quadratic coefficients, since we will need $3w$ bits to exactly multiply out the quadratic terms, or we can use a representable symmetric 3×3 matrix M, a representable vector v, and a $3w$-bit constant R to give a different set of quadrics $\tilde{Q}(p) = (p-v)^T M(p-v) = R$ that is closed under representable translations of v. Whichever definition of quadrics is chosen, the parameter values for line-quadric intersections are the roots of $Q(\ell(t)) = 0$, which can be expressed as a quadratic $at^2 + 2bt + c = 0$ whose coefficients can have at most $3w + 4$ bits. (Four carry bits suffice to sum the $3w$-bit products; $w = 16$ allows exact coefficient computation as IEEE 754 doubles; $w = 33$ as pairs of doubles.)

These definitions are motivated by a problem from David Griesheimer, of Bettis Labs: rather than tracking a particle through quadric surfaces in a CAD model, would it be more robust to compute the intervals of intersections with a segment? We compare three methods to order line-quadric intersections. Our methods, particularly the third, are developed and tested for the case where only one pair of roots has a difference that is potentially overwhelmed by the rounding errors in the computation. We comment at the end how to handle pairs of quadric surfaces that have more than one pair of ambiguous roots.

2 Methods

This section outlines three methods—Approximate Comparison, Repeated Squaring, and Resultant—to sort the intersections with two quadrics, Q_1 and Q_2, with a given line $\ell(t)$, or equivalently, the roots of two quadratics, $a_1t^2 - 2b_1t + c_1 = 0$ and $a_2t^2 - 2b_2t + c_2 = 0$. For each, we evaluate correctness, precision, and floating-point arithmetic operations (FLOPs) required.

2.1 Approximate Comparison

The approximate comparison method computes, for $i \in \{1, 2\}$, the roots $r_i^{\pm} = (b_i \pm \sqrt{b_i^2 - a_i c_i})/a_i$ approximately by computing each operation in IEEE 754 double precision or in extended precision. Actually, to avoid subtractive cancellation, we calculate one of the two roots as $r_i^{-\operatorname{sign} b_i} = -c_i/(b_i + (\operatorname{sign} b_i)\sqrt{b_i^2 - a_i c_i})$. The order of any two chosen approximate roots can be calculated exactly as $\operatorname{sign}(r_1^{\pm} - r_2^{\pm})$.

The rounding of floating point arithmetic means that even with representable input, the correct order is not guaranteed unless we establish a gap or separation theorem (which are also established using resultants [1, 5]) and compute with sufficient precision. Determining this precision is a longstanding open problem [4]. Without a guarantee, this method requires very little computation. Computing both roots takes 12 FLOPs, with one more to compute the sign of the difference. Moreover, the roots can be reused in a scene of many quadrics.

We also use extended precision, where the multiplications and addition in the discriminants are calculated

*School of Computer Science, University of North Carolina at Chapel Hill, mfdeakin@cs.unc.edu, snoeyink@cs.unc.edu with $6w$ bits, square root and addition at $12w$ bits, and divisions at $24w$ bits. To actually perform the comparison, one final subtraction is required at 24 times the initial precision – 1 FLOP, with an initialization cost of 10 FLOPs per quadric intersecting the line.

2.2 Repeated Squaring

The repeated squaring method computes $\operatorname{sign}(r_1^{\pm} - r_2^{\pm})$ by algebraic manipulations to eliminate division and square root operations, leaving multiplications and additions whose precision requirements can be bounded. It uses, for $x \neq 0$, the property that $\operatorname{sign}(y) = \operatorname{sign}(x)\operatorname{sign}(x \cdot y)$. Divisions can be removed directly, since $\operatorname{sign}(r_1^{\pm} - r_2^{\pm}) = \operatorname{sign}(a_1a_2)\operatorname{sign}(a_1a_2(r_1^{\pm} - r_2^{\pm}))$. One square root can be eliminated by multiplying by $r_1^{\pm} - r_2^{\mp}$, giving $\operatorname{sign}(a_1a_2)\operatorname{sign}(a_1a_2(r_1^{\pm} - r_2^{\mp})) \cdot \operatorname{sign}(a_1^2a_2^2(r_1^{\pm} - r_2^{\pm})(r_1^{\pm} - r_2^{\mp}))$. When simplified, the final sign is computed from $a_2^2b_1^2 - 2a_1a_2^2c_1 + 2a_1^2a_2c_2 - a_1a_2b_1b_2 \pm \sqrt{(a_1a_2b_2 - a_2^2b_1)^2(b_1^2 - 4a_1c_1)}$.

The expression under the radical is correctly computed with 8× the input precision; the remaining expression can be evaluated to a little more than 4× input precision in floating point, or can be evaluated in fixed point in 8× input precision by isolating the radical and squaring one last time.

This method not only requires high precision, but also a large number of FLOPs. Computing the unambiguous sign of the difference of the roots requires 15 FLOPs total, and correctly computing the final sign requires another 24 FLOPs. Unfortunately, many of the computed terms require coefficients from both polynomials; only the discriminants, squares, and products can be precomputed, which reduces the number of FLOPs by 14. This brings us to 25 FLOPs per comparison, with an initialization cost of 14 FLOPs per quadric.

Note that this method uses our assumption that we know $\operatorname{sign}(r_1^{\pm} - r_2^{\mp})$ when computing $\operatorname{sign}(r_1^{\pm} - r_2^{\pm})$, but we can learn this from a lower precision test against $-b_2/a_2$, since $r_2^- \leq -b_2/a_2 \leq r_2^+$.

2.3 Resultant

This method was previously described in [2], but a description is included here for completeness.

The resultant method computes the order of two intersections from the resultant for their polynomials, which can be written as the determinant of their Sylvester Matrix [7, Section 3.5]. The general Sylvester Matrix for polynomials $P(t) = p_mt^m + \cdots + p_0$ and $Q(t) = q_nt^n + \cdots + q_0$ is defined as in Equation 1.

$$res(P,Q) = \begin{pmatrix} p_m & \cdots & & p_0 & 0 & & 0 \\ 0 & p_m & \cdots & & p_0 & & 0 \\ & \ddots & & & & \ddots & \\ 0 & 0 & & p_m & \cdots & & p_0 \\ q_n & \cdots & & q_0 & 0 & & 0 \\ 0 & q_n & \cdots & & q_0 & & 0 \\ & \ddots & & & & \ddots & \\ 0 & & q_n & \cdots & & q_0 & \end{pmatrix} \quad (1)$$

The resultant is also the product of the differences of P's roots, $a_1, \ldots, a_n$, and Q's roots, $b_1, \ldots, b_m$, as in Equation 2. [7, Section 6.4]

$$res(P,Q) = p_m^n q_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(a_i - b_j) \quad (2)$$

The two expressions for the resultant provide us with another method of computing the sign of one of the differences of the two roots. Under our assumption that we know the order of all pairs or roots except, say, a_1 and b_1, we can compute $\operatorname{sign}(a_1 - b_1)$ from the determinant and known signs, as in Equation 3 at the top of the next page. The signs need not be multiplied; we simply count the negatives. With quadratics, $m = n = 2$, so the signs of the leading p_2^2 and q_2^2 will be positive and can be ignored.

The determinant can be computed with half the precision and fewer floating point operations than repeated squaring to correctly compute the sign of the differences of roots of the polynomials.

Computing a general 4×4 determinant takes about 120 multiplications, and computing the determinant of the Sylvester matrix itself would naively take 35 FLOPs for each comparison. We can do better in Equation 5 by writing the determinant in terms of the discriminants and other precomputed 2×2 minors from each polynomial. This brings us to 11 FLOPs per comparison, with an initialization cost of 7 FLOPs per intersection.

3 Experimental Evaluation

We experimentally evaluated the resultant method and the approximate computation method with both machine precision and extended precision. Repeated Squaring is dominated by the other methods so was not tested.

We created two types of test scenes that had touching surfaces so that random lines might have some chance (albeit small) to give incorrect orders under approximation, and count the number of disagreements. We evaluated time per comparison for each method on computers with different processors. Finally, by varying the number of surfaces in the second type of scene, we could use

$$\operatorname{sign}(a_1 - b_1) = \operatorname{sign}(res(P,Q)) \operatorname{sign}(p_m^n) \operatorname{sign}(q_n^m) \prod_{i=2}^{m} \prod_{j=2}^{n} [\operatorname{sign}(a_i - b_j) \operatorname{sign}(a_1 - b_j) \operatorname{sign}(a_i - b_1)] \tag{3}$$

$$\Delta = \begin{vmatrix} a_1 & b_1 & c_1 & 0 \\ 0 & a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 & 0 \\ 0 & a_2 & b_2 & c_2 \end{vmatrix} = a_1^2 c_2^2 + c_1^2 a_2^2 + b_1^2 a_2 c_2 + b_2^2 a_1 c_1 - b_1 c_1 a_2 b_2 - a_1 b_1 b_2 c_2 - 2 a_1 c_1 a_2 c_2$$

$$\alpha_i = a_i^2,\ \gamma_i = c_i^2,\ \delta_i = a_i b_i,\ \epsilon_i = a_i c_i,\ \zeta_i = b_i c_i,\ D_i = b_i^2 - \epsilon_i,\ i \in 1,2 \tag{4}$$

$$\Delta = \alpha_1\gamma_2 + \gamma_1\alpha_2 + D_1\epsilon_2 + \epsilon_1 D_2 - \zeta_1\delta_2 - \delta_1\zeta_2 \tag{5}$$

linear regression to determine the contribution to running time from per quadric and per comparison terms.

3.1 Experimental Setup

All methods were implemented in C++[3], and were tested by computing the line-quadric intersection orders along random lines in scenes of quadric surfaces. The creation of these lines and quadric surfaces is described in the next subsection. Machine precision tests were performed in IEEE 754, with quadratic coefficients and discriminants stored as single precision floats, with all machine precision computations performed as floats. MPFR[6] was used to support arbitrary precision in both the approximate comparison and the resultant methods. The approximate comparison method used 24× the precision of a float. This was done to provide a more analogous comparison to the resultant comparison method, which also used 24× the precision of a float.

The first step of the evaluation for a line ℓ and quadric Q was to determined if there was a real intersection by evaluating the discriminant of the quadratic $p(t) = Q(\ell(t))$. This evaluation was done in machine precision, so there is a small chance that near tangent intersections may have been missed due to numeric error in calculating the discriminant. (In our application, missing near tangent intersections was allowed, but getting orders wrong had been known to trap particles into repeatedly trying to cross the same pair of surfaces, which tends to worry a physicist.)

If the intersections are deemed to exist, the second step is to compute the roots at machine precision. These roots are needed to determine if the order of a pair of intersections is ambiguous or not. Finally, the STL sort algorithm is used to sort the intersections. The full process was timed in nanoseconds with the POSIX clock_gettime function.

The comparison function used for sorting came from the method being evaluated. The machine precision approximate comparison just returns the difference of the previously computed roots. In the increased precision approximation and the resultant method, the difference of the roots is compared against a threshold. If the difference was smaller than a threshold of 2^{-16}, the more accurate method provided is used to determine the order, and an appropriate value is returned. This occurred infrequently for a random line, and is only expected to occur a few times for every 100k lines.

We ran tests on two computers with different speeds and operating systems:
Arch was a Core i3 M370 processor with 2 cores, a 3 MB cache, and 4 GB of DDR3 memory clocked at 1 GHz. It ran an up-to-date installation of Arch Linux, kernel version 4.4, and used GCC 6.0 to compile the code with "-O3" as the optimization level. For the tests, the performance manager was set to keep the CPU clock at 2.4 GHz, and the process was run at nice −20.
Gentoo was a Core 2 Duo E6550 processor with two cores, a 4 MB cache, and 8 GB of DDR2 memory clocked at 667 MHz. It ran an up-to-date installation of Gentoo Linux, kernel version 4.1, and used GCC 4.9 to compile the code with "-O3" as the optimization level. For the tests, the performance manager was set to keep the CPU clock at 2.3 GHz, and the process was run at nice −20.

A Geekbench benchmark was employed to estimate the floating point processor speeds, Arch 1702, and Gentoo 1408. Thus, on average, Arch was capable of about 1.2 times more FLOPS than the Gentoo computer.

3.2 Test Scenes

We created two types of test scenes: a single scene of Packed Spheres and a set of scenes of Nested Spheres. All test scenes consisted of quadric surfaces stored as 4×4 matrices of IEEE754 single precision floating point numbers. We preferred spheres and ellipsoids, since any intersecting line would intersect twice, possibly with a repeated root. Sorting isolated single roots is easier, since, for example, the intersection with a plane requires less precision. The quadric surfaces were constructed from the unit cube that has one corner at the origin and the opposite corner at $(1.0, 1.0, 1.0)$.

The single scene of Packed Spheres consisted of 1331 spheres in a hexagonal close packing lattice shown in Fig. 1. This ensures that the spheres each have 12 intersecting or nearly intersecting neighbors. The spheres each have a radius of about 0.05 units, and are spaced about 0.05 units from each other. The initial sphere is centered at the origin, and one of the axes of the lattice is aligned with the y axis of the coordinate frame. The coefficients of the spheres are scaled so that the coefficients of the squared terms were all 1.0. This caused the exponent range for the non-zero coefficients of the spheres to be between -8 and 1, which is well within the limits required for the resultant method to return correct results.

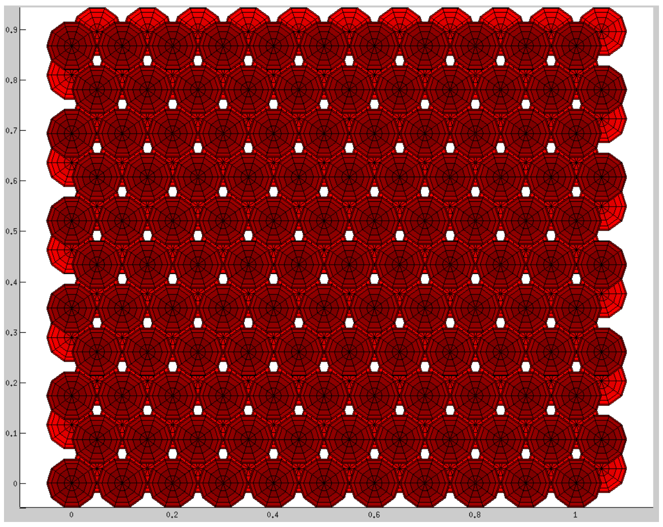
The random lines generated for the scenes of Packed Spheres were generated with an intersect from a uniform distribution over the unit cube. The directions were generated by normalizing a vector chosen from a uniform distribution over the cube with opposite corners at $(-1.0, -1.0, -1.0)$ and $(1.0, 1.0, 1.0)$. To ensure that we are able to compute the order of intersections exactly with the resultant method, the exponents of the non-zero terms were constrained between -20 and 0.

We used eleven scenes of Nested Spheres. One, shown in Fig. 1, had $n = 10$ spheres, the others had $n = 100i$, for $1 \leq i \leq 10$. The first sphere was centered at $x_0 = 0.5, y_0 = 0.5, z_0 = 0.5$ units with a radius of $R_0 = 0.5$ units. The radius of successive spheres decreased linearly so that the final sphere's radius was $R_n = 2^{-16}$ units. Thus, $R_i = R_{i-1} - (R_0 - R_n)/n$. The x position of successive spheres increased linearly to fix the minimum distance at $\epsilon = 2^{-19}$ units. Thus, $x_i = x_{i-1} + (R_0 - R_n)/n - \epsilon$. The exponent range for the non-zero coefficients of the spheres was chosen to be between -1 and 0, which is well within the limits required for the resultant method to return correct results.

Each random line for the scenes of Nested Spheres was generated by chosing a point p_i from a uniform distribution over the unit cube, with a normalized direction vector toward $(1.0, 0.5, 0.5) - p_i$, rounded to single precision, since $(1.0, 0.5, 0.5)$ is close to the points of minimum distance for the sets of spheres. This made it likely that increased precision would be required to correctly compute the order of intersections. To ensure that we are able to compute the order of intersections exactly with the resultant method, the exponents of the non-zero terms were constrained between -20 and 0.

3.3 Analysis

The time that it takes to compute the order of intersections between a given line and a scene of quadric surfaces is expected to be linear in both the number of quadric surfaces and the number of accurate compar-



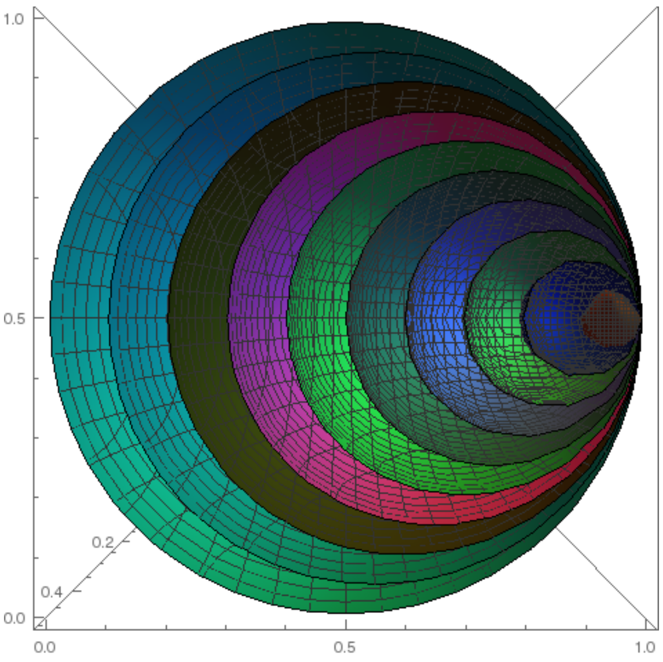


Figure 1: Test Scenes of 1331 Packed Spheres and 10 Nested Spheres, which is smallest of a family of eleven. Random lines in Packed Spheres have some chance of being near sphere contacts. Random lines in Nested Spheres are unlikely to, unless they are biased to pass by the near tangency.

isons made. Because performing accurate comparisons is so much more expensive than normal comparisons, we expect there to be a clear linear relation between the number of accurate comparisons performed and the time it takes to perform the sorting.

The number of quadrics, on the other hand, can significantly affect the number of intersections in the list to be sorted, especially in antagonistic scenes. However, most of the time spent sorting will be accounted for by the time spent making accurate comparisons, which we have already accounted for. Thus, the remaining time will instead come from computing the approximate roots, which is linear.

To analyze the Packed Spheres timing data, we used least squares to fit a line to the number of comparisons made and the timing data. A constant term was also computed for the time taken computing the approximate roots.

To analyze the set of Nested Spheres scenes, we aggregated the test results for the scenes so that we could use least squares to fit a plane to the number of comparisons made, the number of quadric surfaces, and the timing data. A constant term was also computed to catch any hidden initialization costs, though we expect this to contain mostly noise.

4 Experimental Results

The results of the experiments are shown in Table 1. The first thing to notice is that increasing the precision of a computation is not enough to guarantee that the result will be computed correctly. Despite increasing the precision of the computations to 24× the initial precision, the increased precision approximation still fails for 1044/11000 of the random lines in the Nested Spheres scenes. It did, however, perform significantly better than the original calculation, which failed for 8272/11000 of the lines. More lines are needed to find examples that cause errors in the Packed Spheres scene, but based on previous experiments, we can expect several to occur by the $k \cdot 100^{\text{th}}$ test.

In addition to guaranteeing correctness, the resultant method also performed well against the generic increased precision method. For the set of Nested Spheres scenes, it cost slightly more to compute the order of intersections on a time per quadric basis. The approximate computation with increased precision can cache intermediate values more effectively, reducing its cost.

The resultant method performed extremely well on the time per comparison basis, as it actually beat the increased precision method by more than it lost out on in the time per quadric basis in the Nested Spheres scenes, and the Packed Spheres scene on the Gentoo machine.

After removing the time per quadric basis in the tests with the Nested Spheres scenes, the constant term

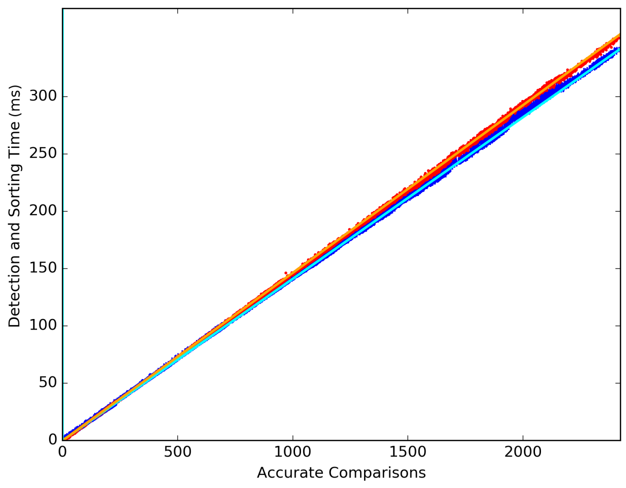


Figure 2: Evaluation Time for a Line (ms) vs. the Number of Comparisons; Sorting the intersections of 1k lines in each of the Nested Spheres test scenes on the Gentoo machine; **Red Dot's (above the bars): Approximation Method at** 24× **the Input Precision**; **Blue Bars (beneath the dots): Resultant Method**. The least squares coefficient for the time per quadrics has been subtracted out to better show the actual fit. The lines show the respective least squares fits without the quadric term.

appears somewhat nonsensical. From previous experiments, we have concluded that this is mostly noise, suggesting that we obtained most of the useful information from the measured times. This suggests the time per quadric is the main contributor to the constant time in the tests with the Packed Spheres scene as we expected.

Figure 2 shows a plot of the results from one of the tests. It appears to confirm our expectation that the time required is linearly correlated with the number of precision increases.

5 Conclusion

In this paper we showed how the resultant method can guarantee the correct order of line-quadric intersections at a similar cost to using an increased precision approximation method. We have also shown that naively using increased precision to improve accuracy is not enough to eliminate errors, and that one must take into account the operations being used and the ranges of the input.

We have assumed that we know the order of all roots except one pair. Even if one's application does not provide this information, for quadratic equations it is relatively easy to obtain using lower precision than it takes to compare roots. The zero of the derivative $x_i = -b_i/a_i$

Table 1: Analysis of the timing of the Approximate Comparison and Resultant Comparison. Timing data for 11k lines was analyzed for the Packed Spheres scene to find the coefficients of the best fitting lines. Timing data for 1k lines was analyzed for each of the set of 11 Nested Spheres scenes to find the coefficients of the best fitting planes. The dimensions are the number of quadric surfaces and the number of increased precision comparisons made.

Scene	Machine	Method	Errors	ms/Quadric	ms/Comp	Const ms	$\sum$ Residual (ms^2)
Nested Spheres	Arch	Approximate	8272	0.00425	0.000361	-0.0693	149.084
		Increased Prec.	1044	0.00554	0.105	-0.567	87655.1
		Resultant	—	0.00670	0.100	-0.746	80544.6
	Gentoo	Approximate	8244	0.00379	0.000313	-0.0519	34.4705
		Increased Prec.	1042	0.00484	0.146	-0.110	11944.9
		Resultant	—	0.00584	0.141	0.00485	19872.5
Packed Spheres	Arch	Approximate	0	–	0.00738	4.54	21.7059
		Increased Prec.	0	–	0.126	4.49	22.4387
		Resultant	—	–	0.130	4.51	23.5822
	Gentoo	Approximate	0	–	0.00180	4.37	3.75176
		Increased Prec.	0	–	0.156	4.37	3.76225
		Resultant	—	–	0.155	4.41	3.83604

separates r_i^- and r_i^+ by value of the discriminant. If $x_1 = x_2$ then comparing squared discriminants tells us all we need to know about root orders. When, wlog, $x_1 < x_2$, we use the signs of both quadratics at x_1 and x_2 to bound roots to intervals, and can again compare squared discriminants to reveal the order for all but one pair.

6 Acknowledgment

We thank David Griesheimer for discussions on this problem, and both NSF and Bettis Labs for their support of this research.

References

[1] W Dale Brownawell and Chee K Yap. "Lower bounds for zero-dimensional projections". In: *Proc. Int'l Symp on Symbolic and Algebraic Computation.* ACM. 2009, pp. 79–86.

[2] Michael Deakin. "Fast and Accurate Floating Point Algorithms". MA thesis. University of North Carolina at Chapel Hill, 2016.

[3] Michael Deakin and Jack Snoeyink. *Geometry: CCCG2016 Version.* May 2016. DOI: 10.5281/zenodo.57108. URL: http://dx.doi.org/10.5281/zenodo.57108.

[4] Erik D Demaine, Joseph SB Mitchell, and Joseph O'Rourke. *The open problems project, Problem 33: Sum of square roots.* http://cs.smith.edu/~orourke/TOPP/P33.html.

[5] Ioannis Z Emiris, Bernard Mourrain, and Elias P Tsigaridas. "The DMM bound: Multivariate (aggregate) separation bounds". In: *Proc. Int'l Symp on Symbolic and Algebraic Computation.* http://arxiv.org/abs/1005.5610. ACM. 2010, pp. 243–250.

[6] Laurent Fousse et al. "MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding". In: *ACM Trans. Math. Softw.* 33.2 (June 2007). ISSN: 0098-3500. DOI: 10.1145/1236463.1236468. URL: http://doi.acm.org/10.1145/1236463.1236468.

[7] Chee-Keng Yap. *Fundamental problems of algorithmic algebra.* Oxford University Press, 2000.

Adaptive Metrics for Adaptive Samples

Nicholas J. Cavanna
University of Connecticut

Donald R. Sheehy
University of Connecticut

Abstract

We generalize the local-feature size definition of adaptive sampling used in surface reconstruction to relate it to an alternative metric on Euclidean space. In the new metric, adaptive samples become uniform samples, making it simpler both to give adaptive sampling versions of homological inference results and to prove topological guarantees using the theory of critical points to distance functions.

1 From Surface Reconstruction to Homology Inference

To reconstruct a surface from a point set, one needs the sample to be sufficiently dense with respect to not just the local curvature of the surface, but also the distance to parts of the surface that are close in the embedding but far in geodesic distance. Otherwise, algorithms have no way of identifying which geometrically close sample points correspond to local neighborhoods in the surface. Adaptive sampling with respect to the so-called *local feature size* as introduced by Amenta and Bern [2] neatly characterizes such "good" samples and was then used in many later works on surface reconstruction with topological guarantees [7]. Such *adaptive samples* are in contrast to *uniform samples* for which a single parameter determines the density, usually driven by minimum of the local feature size and resulting in a much larger sample.

Later work on generalizations of surface reconstruction and homology inference related the topology of unions of balls centered at a sample $\widehat{X}$ near the unknown set X to the topology of X itself. A union of balls with a fixed radius can be viewed as a sublevel set of the distance function to $\widehat{X}$. If we have an adaptive sample, then we would like to scale the radii of the balls as well. However, if the sample is adaptive with respect to a local feature size defined as the distance to an unknown set L, another approximation $\widehat{L}$ near L is necessary. Indeed, one interpretation of some Voronoi-based surface reconstruction algorithms is that first an approximation $\widehat{L}$ to the medial axis L is computed from the Voronoi diagram of the sample $\widehat{X}$ of the unknown surface X.

We present a new perspective on adaptive samples. For *any* pair of disjoint, compact sets X and L, we define a metric on $\mathbb{R}^d \setminus L$ with the property that a uniform sample of X in the new metric corresponds to an adaptive sample in the Euclidean metric. This new metric can also be extended to an arbitrarily close Riemannian metric over the same domain. Our main motivation is to connect adaptive sampling theory to the critical point theory of distance functions used extensively to prove topological guarantees in topological data analysis [9, 4, 3]. That theory gives natural topological equivalences between sublevel sets of distance functions to compact sets in Riemannian metrics. Thus, we propose to use this new metric as the underlying ideal object and then relate it to a union of Euclidean balls constructed from approximations to X and L. Our metric can be viewed as a smoothed version of an adaptive metric used by Clarkson [5]. Our new formulation reveals connections with recent work on path planning [10, 1] and density-based distances [6].

2 Background

Let L and X be compact subsets of $\mathbb{R}^d$ with respect to the Euclidean metric. For $x, y \in \mathbb{R}^d$, define $\mathrm{Path}(x, y)$ to be the set of bounded piecewise-C_1 paths from x to y, parametrized by Euclidean arc-length. Similarly, $\mathrm{Path}(x, S) := \bigcup_{s \in S} \mathrm{Path}(x, s)$ denotes all paths from x to a set S.

For any compact set $L \subseteq \mathbb{R}^d$, define $f_L(\cdot) : \mathbb{R}^d \to \mathbb{R}$ by $f_L(x) := \min_{\ell \in L} \|x - \ell\|$. Define $\mathrm{d}^L(x, y) := \min_{\gamma \in \mathrm{Path}(x,y)} \int_\gamma \frac{dz}{f_L(z)}$. Note that d^L is a Riemannian metric on $\mathbb{R}^d \setminus L$. The length of a unit-speed path $\gamma : [0, a] \to \mathbb{R}^d$ is denoted as $|\gamma| := \int_\gamma dz = \int_0^a dt$.

For $y \in \mathbb{R}^d$, define $f_X^L(y) := d^L(y, X) = \min_{x \in X} \mathrm{d}^L(y, x)$, and $\widehat{f_X^L}(y) := \min_{x \in X} \frac{\|y-x\|}{f_L(x)}$.

Note that $f_X^L(\cdot)$ is a distance function, while $\widehat{f_X^L}(\cdot)$ is not. The latter function can be interpreted as a first-order approximation of the former.

The two sets resulting from the level sets of these functions are defined below, with the goal being to approximate $A_X^L(\cdot)$ by $B_{\widehat{X}}^{\widehat{L}}(\cdot)$, where $\widehat{L}$ and $\widehat{X}$ are approximations of L and X respectively.

Definition 1 *For any compact set $X \subset \mathbb{R}^d \setminus L$, for some compact set $L \subset \mathbb{R}^d$, the* α-offsets *with respect to*

d^L *are*

$$A_X^L(\alpha) := \{x \in \mathbb{R}^d \mid f_X^L(x) \leq \alpha\}.$$

Note the distance function $f_L(\cdot)$ can be transformed into an arbitrarily close smooth function $\tilde{f}_L(\cdot)$ [8], yielding a Riemannian metric $\tilde{\mathrm{d}}_L$ defined in an identical manner as d_L. From this, one has corresponding α-offsets $\tilde{A}_X^L(\alpha)$ that are arbitrarily close to $A_L^X(\alpha)$.

Definition 2 *For any compact set $X \subset \mathbb{R}^d \setminus L$, for some compact set $L \subset \mathbb{R}^d$, the* approximate α-offsets *with respect to* d^L *are*

$$B_X^L(\alpha) := (\widehat{f_X^L})^{-1}[0,\alpha] = \bigcup_{x \in X} \mathrm{ball}(x, \alpha f_L(x)).$$

A useful property of $f_X^L(\cdot)$ is that it a 1-Lipschitz function. In general, a function f between two metric spaces (X, d_X) and (Y, d_Y) is said to be k-Lipschitz if for all $x, y \in X$, $\mathrm{d}_Y(f(x), f(y)) \leq k\mathrm{d}_X(x,y)$.

Lemma 3 *$f_X^L(\cdot)$ is a 1-Lipschitz function from $(\mathbb{R}^d, \mathrm{d}^L)$ to $\mathbb{R}$.*

Proof. Fix any $a, b \in \mathbb{R}^d$. There exists $x \in X$ and $\gamma_1 \in \mathrm{Path}(a,x)$ such that $f_X^L(a) = \int_{\gamma_1} \frac{dz}{f_L(z)}$. Likewise, there exists $\gamma_2 \in \mathrm{Path}(a,b)$ such that $\mathrm{d}^L(a,b) = \int_{\gamma_2} \frac{dz}{f_L(z)}$.

This implies $\gamma_1 + \gamma_2 \in \mathrm{Path}(b, X)$, where $+$ in this case is the concatenation of paths in the usual sense.

Thus $f_X^L(b) \leq \int_{\gamma_1+\gamma_2} \frac{dz}{f_L(z)} \leq f_X^L(a) + \mathrm{d}^L(a,b)$. By symmetry of a and b, we obtain the other bound and we are done. □

We can extend $f_X^L(\cdot)$, a function measuring the distance from a point to a set, to the resulting Hausdorff distance, which is a metric between compact sets. This metric is useful for stating bounds on the quality, or uniformity, of a sample near a set.

Definition 4 *The* Hausdorff distance *between two compact sets $A, B \in (\mathbb{R}^d, \mathrm{d}^L)$ is defined as*

$$\mathrm{d}_H^L(A,B) = \max\{\min_{a \in A} f_B^L(a), \min_{b \in B} f_A^L(b)\}$$

or equivalently,

$$\mathrm{d}_H^L(A,B) = \min\{r \mid A \subseteq B_L^r \text{ and } B \subseteq A_L^r\}.$$

Using an assumption on the Hausdorff distance between a compact set and a sample of it, Lemma 5 shows their α-offsets can be included within each other at particular scales.

Lemma 5 *Consider $\widehat{X}, X \subseteq \mathbb{R}^d \setminus L$ be such that $\mathrm{d}_H^L(\widehat{X}, X) \leq \delta$. Then for all $\alpha \geq 0$, $A_X^L(\alpha) \subseteq A_{\widehat{X}}^L(\alpha+\delta)$ and $A_{\widehat{X}}^L(\alpha) \subseteq A_X^L(\alpha+\delta)$.*

Proof. Fix $y \in A_X^L(\alpha)$. By definition $f_X^L(y) \leq \alpha$, which implies that there exists $x \in X$ such that $\mathrm{d}^L(x,y) \leq \alpha$. $\mathrm{d}_H^L(\widehat{X}, X) \leq \delta$ which implies that for all $x \in X$, $f_{\widehat{X}}^L(x) \leq \delta$. Now by Lemma 3, $f_{\widehat{X}}^L(y) \leq f_{\widehat{X}}^L(x) + \mathrm{d}^L(x,y) \leq \delta + \alpha$, implying $y \in A_{\widehat{X}}^L(\alpha+\delta)$. By a symmetric argument, the other statement holds. □

The following is the definition of an adaptive sample we use, known as an ε-sample.

Definition 6 *Given compact set $L \subset \mathbb{R}^d$ and compact sets $X, \widehat{X} \subset \mathbb{R}^d \setminus L$ such that $\widehat{X} \subseteq X$, we say that $\widehat{X}$ is an* ε-sample *of X, for $\varepsilon \in [0,1)$, if for all $x \in X$, there exists $p \in \widehat{X}$ such that $\|x - p\| \leq \varepsilon f_L(x)$.*

This definition is closely related to that of the approximate α-offsets, because if $\widehat{X}$ is an ε-sample of X, then for all $x \in X$, $\mathrm{ball}(x, \varepsilon f_L(x)) \cap \widehat{X} \neq \emptyset$.

3 Adaptive Sampling

In this section, we prove that a uniform sample in the induced metric corresponds to an adaptive sample in the Euclidean metric and vice versa. The key to this proof is the following lemma about the relationship between the two metrics when just considering two points. This lemma will also be used for the more elaborate interleaving results of Section 4.

Lemma 7 *Let $L \subset \mathbb{R}^d$ be a compact set and let $a, b \in \mathbb{R}^d \setminus L$. Then, the following two statements hold for all $\delta \in [0,1)$.*

(i) If $\mathrm{d}^L(a,b) \leq \delta$ then $\frac{\|a-b\|}{f_L(a)} \leq \frac{\delta}{1-\delta}$.

(ii) If $\frac{\|a-b\|}{f_L(a)} \leq \delta$ then $\mathrm{d}^L(a,b) \leq \frac{\delta}{1-\delta}$.

Proof. To prove (i), we assume $\mathrm{d}^L(a,b) \leq \delta$. Let γ be the path in $\mathrm{Path}(a,b)$ such that $\mathrm{d}^L(a,b) = \int_\gamma \frac{dz}{f_L(z)} < \delta$. Then we have the following inequalities following from the Lipschitz property of f_L.

$$\begin{aligned}|\gamma| = \int_\gamma dz &= (f_L(a) + |\gamma|) \int_\gamma \frac{dz}{f_L(a) + |\gamma|} \\ &\leq (f_L(a) + |\gamma|) \int_\gamma \frac{dz}{f_L(z)} \\ &\leq (f_L(x) + |\gamma|)\delta\end{aligned}$$

It follows that $|\gamma| \leq \frac{\delta}{1-\delta} f_L(x)$. Because $\|a-b\|$ is the length of the shortest path between a and b in the Euclidean metric, we conclude $\|a - b\| \leq |\gamma| \leq \frac{\delta}{1-\delta} f_L(x)$.

Next we prove (ii). Assume $\frac{\|a-b\|}{f_L(a)} \leq \delta$. For all points z in the straight line segment $\overline{ab}$,

$$f_L(z) \geq f_L(a) - \|a-z\| \geq f_L(a) - \|a-b\| \geq (1-\delta) f_L(a).$$

This implies the following inequality.

$$\begin{aligned}
\mathrm{d}^L(a,b) &= \inf_{\gamma\in\mathrm{Path}(a,b)} \int_\gamma \frac{dz}{f_L(z)} \\
&\le \int_{ab} \frac{dz}{f_L(z)} \\
&\le \frac{1}{(1-\delta)f_L(a)} \int_{ab} dz \\
&= \frac{\|a-b\|}{(1-\delta)f_L(a)} \\
&\le \frac{\delta}{1-\delta}.
\end{aligned}$$

□

We can now state the main theorem relating adaptive samples in the Euclidean metric to uniform samples in the metric induced by a set L.

Theorem 8 *Let L and X be compact sets, let $\widehat{X} \subset X$ be a sample, and let $\varepsilon \in [0,1)$ be a constant. If $\widehat{X}$ is an ε-sample of X with respect to the distance to L, then $\mathrm{d}_H^L(X,\widehat{X}) \le \frac{\varepsilon}{1-\varepsilon}$. Also, if $\mathrm{d}_H^L(X,\widehat{X}) \le \varepsilon < \frac{1}{2}$, then $\widehat{X}$ is an $\frac{\varepsilon}{1-\varepsilon}$-sample of X with respect to the distance to L.*

Proof. Given $x \in X$, there exists $p \in \widehat{X}$ such that $\|x-p\| \le \varepsilon f_L(x)$. By Lemma 7, $\mathrm{d}^L(x,p) \le \frac{\varepsilon}{1-\varepsilon}$, so for all $x \in X$, $f_{\widehat{X}}^L(x) \le \frac{\varepsilon}{1-\varepsilon}$. As $\widehat{X} \subseteq X$, this proves $\mathrm{d}_H^L(\widehat{X},X) \le \frac{\varepsilon}{1-\varepsilon}$.

$\mathrm{d}_H^L(\widehat{X},X) \le \varepsilon < \frac{1}{2}$ implies that for all $x \in X$, $f_{\widehat{X}}^L(x) \le \varepsilon$, thus there exists $p \in \widehat{X}$ such that $\mathrm{d}^L(x,p) \le \varepsilon$, and thus by Lemma 7 $\|x-p\| \le \frac{\varepsilon}{1-\varepsilon} f_L(x)$. Since $\varepsilon < \frac{1}{2}$, then $\frac{\varepsilon}{1-\varepsilon} < 1$, so $\widehat{X}$ is an $\frac{\varepsilon}{1-\varepsilon}$-sample of X. □

4 Interleaving

A *filtration* is a nested family of sets. In this paper, we consider filtrations F parameterized by a real number $\alpha \ge 0$ so that $F(\alpha) \subset \mathbb{R}^d$ and whenever $\alpha < \beta$ we have $F(\alpha) \subseteq F(\beta)$. Often, our filtrations are *sublevel filtrations* of a real valued function $f : \mathbb{R}^d \to \mathbb{R}$. The sublevel filtration F corresponding to the function f is the defined as

$$F(\alpha) := \{x \in \mathbb{R}^d \mid f(x) \le \alpha\}.$$

Definition 9 *A pair of filtrations (F,G) is (h_1,h_2)-*interleaved *in an interval (s,t) if $F(r) \subseteq G(h_1(r))$ whenever $r, h_1(r) \in (s,t)$ and $G(r) \subseteq F(h_2(r))$ whenever $r, h_2(r) \in (s,t)$. We require that the functions h_1, h_2 be nondecreasing in (s,t).*

The following lemma gives us an easy iterative way to combine pairs of interleavings.

Lemma 10 *If (F,G) is (h_1,h_2)-interleaved in (s_1,t_1), and (G,H) is (h_3,h_4)-interleaved in (s_2,t_2), then (F,H) is $(h_3\circ h_1, h_2\circ h_4)$-interleaved in (s_3,t_3), where $s_3 = \max\{s1,s2\}$ and $t_3 = \min\{t_1,t_2\}$.*

Proof. If $r, h_3(h_1(r)) \in (s_3,t_3)$, then we have $F(r) \subseteq G(h_1(r)) \subseteq H(h_3(h_1(r)))$. Similarly, if $r, h_2(h_4(r)) \in (s_3,t_3)$, then $H(r) \subseteq G(h_4(r)) \subseteq F(h_2(h_4(r)))$. □

4.1 Approximating X with $\widehat{X}$

Ultimately, the goal is to relate A_X^L, the offsets in the induced metric, to $B_{\widehat{X}}^{\widehat{L}}$, the approximate offsets computed from approximations (or samples) to both X and L. This relationship will be given by an interleaving that is built up from an interleaving for each approximation step. For each of the following lemmas, let $L, \widehat{L} \subset \mathbb{R}^d$ and $X, \widehat{X} \subset \mathbb{R}^d \setminus (L \cup \widehat{L})$ be compact sets.

Lemma 11 *If $\mathrm{d}_H^L(\widehat{X},X) \le \varepsilon$, then $(A_X^L, A_{\widehat{X}}^L)$ is (h_1,h_1)-interleaved in $(0,\infty)$, where $h_1(r) = r+\varepsilon$.*

Proof. This lemma is a reinterpretation of Lemma 5 in the interleaving notation. □

4.2 Approximating the Induced Metric

It is much easier to use a union of Euclidean balls to model the sublevel sets of the distance function f_X^L. Below, we show that this is a reasonable approximation. The following results may also be viewed as a strengthening of the adaptive sampling result of the previous section (Theorem 8).

Lemma 12 *Given compact set $L \subset \mathbb{R}^d$, and compact set $X \subset \mathbb{R}^d \setminus L$, for $r \in [0,1)$, $A_X^L(r) \subseteq B_X^L(\frac{r}{1-r})$, and for $r \in [0,\frac{1}{2})$, $B_X^L(r) \subseteq A_X^L(\frac{r}{1-r})$.*

Proof. Take $y \in A_X^L(r)$ so that $f_X^L(y) \le r$. Thus there exists $x \in X$ such that $\mathrm{d}^L(x,y) \le r$. By Lemma 7, this implies that $\|x-y\| \le \frac{r}{1-r} f_L(x)$, which implies that $y \in B_X^L(\frac{r}{1-r})$.

Consider $y \in B_X^L(r)$. Thus $y \in \mathrm{ball}(x, rf_L(x))$, for some $x \in X$, so $\|x-y\| \le rf_L(x)$. Applying Lemma 7, we have then have that $\mathrm{d}^L(x,y) \le \frac{r}{1-r}$, and as $f_X^L(y) \le \mathrm{d}^L(x,y)$, $y \in A_X^L(\frac{r}{1-r})$. □

Corollary 13 *The pair $(A_{\widehat{X}}^L, B_{\widehat{X}}^L)$ are (h_2,h_2)-interleaved in $(0,1)$, where $h_2(r) = \frac{r}{1-r}$.*

Proof. This follows from combining the results of Lemma 12 into the interleaving notation.

□

4.3 Approximating L with $\widehat{L}$

Usually, the set L is unknown at the start and must be estimated from the input. For example in the case that L is the medial axis of X, there are several known techniques for approximating L by, for example, taking some vertices of the Voronoi diagram [2, 7]. We would like to give some sampling conditions that guarantee that allow us to replace L with an approximation $\widehat{L}$. Interestingly, the sampling conditions for $\widehat{X}$ are dual to those used for $\widehat{L}$. That is, we require $\mathrm{d}_H^{\widehat{X}}(L, \widehat{L}) \leq \varepsilon$, or, in other words, $\widehat{L}$ must be an adaptive sample with respect to the distance to $\widehat{X}$.

Lemma 14 *If $\mathrm{d}_H^{\widehat{X}}(L, \widehat{L}) \leq \delta < 1$, then $(B_{\widehat{X}}^{L}, B_{\widehat{X}}^{\widehat{L}})$ is (h_3, h_3)-interleaved in $(0, \infty)$, where $h_3(r) = \frac{r}{1-\delta}$.*

Proof. Fix any $x \in B_{\widehat{X}}^{L}(r)$. There is a point $p \in \widehat{X}$ such that $\frac{\|x-p\|}{f_L(p)} \leq r$. Moreover, there is a nearest point $z \in \widehat{L}$ to x, and so $f_{\widehat{L}}(p) = \|p - z\|$. Lemma 7 and the assumption that $\mathrm{d}_H^{\widehat{X}}(L, \widehat{L}) \leq \delta$ implies that there exists $y \in L$ such that

$$\|y - z\| \leq \frac{\delta}{1-\delta} f_{\widehat{X}}(z). \tag{1}$$

The definitions imply the following.

$$f_{\widehat{X}}(z) = \min_{q \in \widehat{X}} \|z - q\| \leq \|z - p\| = f_{\widehat{L}}(p). \tag{2}$$

So, we can bound $f_L(p)$ in terms of $f_{\widehat{L}}(p)$ as follows.

$$\begin{aligned} f_L(p) &\leq \|y - p\| && [y \in L] \\ &\leq \|y - z\| + \|z - p\| && \text{[triangle inequality]} \\ &\leq \frac{1}{1-\delta} f_{\widehat{L}}(p) && \text{[by (1) and (2)]} \end{aligned}$$

So,

$$\frac{\|x-p\|}{f_{\widehat{L}}(p)} \leq \frac{\|x-p\|}{(1-\delta) f_L(p)} \leq \frac{r}{1-\delta} = h_3(r).$$

Therefore, $x \in B_{\widehat{X}}^{\widehat{L}}(h_3(r))$ and so we conclude that $B_{\widehat{X}}^{L}(r) \subseteq B_{\widehat{X}}^{\widehat{L}}(h_3(r))$. The proof is symmetric to show that $B_{\widehat{X}}^{\widehat{L}}(r) \subseteq B_{\widehat{X}}^{L}(h_3(r))$ □

4.4 Putting it all together

We can now combine the interleavings established in Corollary 13, and Lemmas 11 & 14, using Lemma 10.

Theorem 15 *Let $L, \widehat{L} \subset \mathbb{R}^d$ and $X, \widehat{X} \subset \mathbb{R}^d \setminus (L \cup \widehat{L})$ be compact sets. If $\mathrm{d}_H^{\widehat{X}}(L, \widehat{L}) \leq \delta < 1$ and $\mathrm{d}_H^{L}(\widehat{X}, X) \leq \varepsilon < 1$, then $(A_X^L, B_{\widehat{X}}^{\widehat{L}})$ are (h_4, h_5)-interleaved in $(0, 1)$, where $h_4(r) = \frac{r+\varepsilon}{(1-r-\varepsilon)(1-\delta)}$ and $h_5(r) = \frac{r}{1-\delta-r} + \varepsilon$.*

Proof. By Lemma 10 along with the interleavings from Lemmas 11, 13, $(A_X^L, B_{\widehat{X}}^{L})$ is $(h_2 \circ h_1, h_1 \circ h_2)$-interleaved in $(0, 1)$. Combining this interleaving with the one resulting from Lemma 14 we get that $(A_X^L, B_{\widehat{X}}^{\widehat{L}})$ is $(h_3 \circ h_2 \circ h_1, h_1 \circ h_2 \circ h_3)$ interleaved in $(0, 1)$. Now we must compute $h_3 \circ h_2 \circ h_1$ and $h_1 \circ h_2 \circ h_3$.

$$\begin{aligned} (h_3 \circ h_2 \circ h_1)(r) = (h_3 \circ h_2)(r + \delta) &= h_3\left(\frac{r+\delta}{1-r-\delta}\right) \\ &= \frac{r+\delta}{(1-r-\delta)(1-\varepsilon)} \\ (h_1 \circ h_2 \circ h_3)(r) = (h_1 \circ h_2)\left(\frac{r}{1-\varepsilon}\right) &= h_1\left(\frac{r}{(1-\varepsilon)(1-\frac{r}{1-\varepsilon})}\right) \\ &= h_1\left(\frac{r}{1-\varepsilon-r}\right) \\ &= \frac{r}{1-\varepsilon-r} + \delta \end{aligned}$$

So we have that $h_4(r) = \frac{r+\delta}{(1-r-\delta)(1-\varepsilon)}$ and $h_5(r) = \frac{r}{1-\varepsilon-r} + \delta$. □

5 Conclusion

In our paper, we present results based on an alternative metric in Euclidean space that connect adaptive sampling and uniform sampling. With a metric comes a distance function with which one can apply classical results from critical point theory to infer topological properties of the underlying space, thus providing a connection between surface reconstruction (adaptive sampling) and homology inference (uniform sampling). Since one does not know the exact compact set X being reconstructed, nor the reference set L on which the adaptive sample is based, approximations $\widehat{X}$ and $\widehat{L}$ are needed.

We show in Theorem 8 that there is a precise relationship between samples that are uniformly taken with respect to d^L at some scale, to those same samples being adaptive in the Euclidean metric. In our main result, Theorem 15, we show that we can interleave the sublevel sets of our distance function under this alternative metric with the metric balls resulting from our approximation of the metric, assuming that both $\widehat{X}$ and $\widehat{L}$ are uniformly well-sampled with respect to the Hausdorff distance of d^L and $\mathrm{d}^{\widehat{X}}$. Using all approximations, albeit well-chosen ones, one can infer the behavior of the defined metric as well as the sublevel sets of it with respect to X.

There is a natural next step building off of this research that broadens its scope, background, and further applications. With the aforementioned critical point theory, these interleavings could be extended to homological guarantees about the compact set X in question.

An application of such a result could be that obstacle avoidance results could be reframed as obstacle exploration.

References

[1] P. K. Agarwal, K. Fox, and O. Salzman. An efficient algorithm for computing high quality paths amid polygonal obstacles. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1179–1192, 2016.

[2] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *SIGGRAPH*, pages 415–421, 1998.

[3] F. Chazal, D. Cohen-Steiner, and A. Lieutier. A sampling theory for compact sets in Euclidean space. *Discrete & Computational Geometry*, 41:461–479, 2009.

[4] F. Chazal and A. Lieutier. Smooth manifold reconstruction from noisy and non-uniform approximation with guarantees. *Computational Geometry: Theory and Applications*, 40:156–170, 2008.

[5] K. L. Clarkson. Building triangulations using ε-nets. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 326–335, 2006.

[6] M. B. Cohen, B. T. Fasy, G. L. Miller, A. Nayyeri, D. R. Sheehy, and A. Velingker. Approximating nearest neighbor distances. In *Proceedings of the Algorithms and Data Structures Symposium*, 2015.

[7] T. K. Dey. *Curve and Surface Reconstruction : Algorithms with Mathematical Analysis.* Cambridge University Press, 2007.

[8] R. Green and H. Wu. C^∞ approximations of convex, subharmonic, and plurisubharmonic functions. *Ann. Sci. École Norm. Sup.*, 12(1):47–84, 1979.

[9] K. Grove. Critical point theory for distance functions. In *Proceedings of the Symposia in Pure Mathematics*, volume 54, 1993.

[10] R. Wein, J. van den Berg, and D. Halperin. Planning high-quality paths and corridors amidst obstacles. *The International Journal of Robotics Research*, 27(11-12):1213–1231, November/December 2008.

Geometric Spanners Merging and its Applications

Davood Bakhshesh[*] Mohammad Farshi[†]

Abstract

Let $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ with $P_1 \cap P_2 = \emptyset$ be two t-spanners ($t > 1$) in the plane. One interesting question is that how one can merge these two t-spanners to make a t-spanner on $P_1 \cup P_2$ by only adding some edges between G_1 and G_2. In this paper, we propose an algorithm to merge G_1 and G_2 in $O(n_1 \log n_1 + (n_1 + n_2) \log n_2)$ time, where $n_1 = |P_1|, n_2 = |P_2|$ and without loss of generality we assume that $n_1 \leq n_2$. Furthermore, using the proposed algorithm, we present some divide and conquer algorithms to construct a t-spanner and a fault-tolerant t-spanner for a given point set in the plane.

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$ and $t > 1$ be a real number. A geometric graph is an edge-weighted graph on $S \subseteq \mathbb{R}^d$ such that the weight of each edge is the Euclidean distance between its endpoints. A geometric graph G with vertex set S is called a *t-spanner* for S, if for each two points p and q in S, there exists a path Q in G between p and q whose length is at most t times $|pq|$, the Euclidean distance between p and q. The length of a path is defined to be the sum of the weight of all edges on the path. The path Q is called a *t-spanner path* (or *t-path*) between p and q. We denote the length of path Q by $|Q|$. The *stretch factor* (or *dilation*) of G is the smallest value of t for which G is a t-spanner. The t-spanners were introduced by Peleg and Schäffer [10] in the scope of distributed computing and, then by Chew [6] in the scope of computational geometry, and they are applicable in many scopes such as graph theory, network topology design, distributed systems and robotics. We refer the reader to [5, 7, 8, 11] for reading about the t-spanners and their applications.

One of the useful properties of a network is the *region-fault tolerance*. A network is called region-fault tolerant if after removing vertices/edges of the network that lie in a region, the remaining part of the network keeps its good properties. In 2009, Abam et al. [2] introduced the concept of *region-fault tolerant spanner* for planar point sets. For a fault region F and a geometric graph G on a point set S, assume $G \ominus F$ is the remaining graph after removing the vertices of G that lie inside F and all edges that intersect F. For a set $\mathcal{F}$ of regions in the plane, an $\mathcal{F}$-fault tolerant t-spanner of S is a geometric graph G on S such that for any region $F \in \mathcal{F}$, the graph $G \ominus F$ is a t-spanner for $\mathcal{G}_c(S) \ominus F$, where $\mathcal{G}_c(S)$ is the complete geometric graph on S. They used the concept of *the semi-separated pair decomposition* (*SSPD*) which is a data structure similar to *the well-separated pair decomposition* (*WSPD*) to construct a $\mathcal{C}$-fault tolerant t-spanner with size of $O(n \log n)$ in $O(n \log^2 n)$ time for a set of n points in the plane, where $\mathcal{C}$ is any family of convex regions in the plane. For more details about SSPD and WSPD, we refer the reader to [1, 3, 4].

In the scope of geometric spanner networks, we usually face the following problem: given a point set S and a real number $t > 1$, construct a sparse t-spanner of S in optimal time. The problem of efficient construction of a sparse t-spanner has been studied extensively. One can see the major algorithms for building spanners in the book by Narasimhan and Smid [9]. Now, let G_1 and G_2 be two t-spanners of point sets P_1 and P_2, respectively. One may ask *how one constructs a sparse t-spanner on $P_1 \cup P_2$ by **only** connecting G_1 to G_2 using the small number of edges between them?* This question may be raised in the real world for example, connecting two countries by constructing some roads between them or connecting two water networks by creating some water pipelines.

We define the *spanners merging problem* as follows:

Definition 1 (Spanners Merging Problem) *Given two t-spanners $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ in the plane, construct a t-spanner G on the point set $P_1 \cup P_2$ by **only** adding small number of edges between G_1 and G_2.*

We emphasize that in the spanners merging problem, it is important that we construct the graph G by adding edges between G_1 and G_2. Otherwise one can use some well-known algorithms e.g. greedy algorithm, Yao graph, Θ-graph, on $P_1 \cup P_2$ to construct a sparse t-spanner on $P_1 \cup P_2$. Also, one can merge G_1 and G_2 by adding all edges between G_1 and G_2 which needs a lot of edges.

Throughout the paper, $t > 1$ will be assumed to be a real constant, $n_1 = |P_1|, n_2 = |P_2|$ and without loss of

[*]Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran dbakhshesh@gmail.com

[†]Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran, mfarshi@yazd.ac.ir

generality we assume $n_1 \leq n_2$. In this paper, we propose an algorithm that takes $O(n_1 \log n_1 + (n_1 + n_2) \log n_2)$ time to solve the spanners merging problem. Moreover, using the proposed algorithm, we propose some divide and conquer algorithms to construct a t-spanner and an $\mathcal{H}_k$-fault tolerant t-spanner for a given point set in the plane, where $\mathcal{H}_k$ is the family of half-planes such that their boundaries are parallel to some line of the finite set of specific lines with respect to parameter k (In the next sections, we will define $\mathcal{H}_k$ precisely). Note that the proposed divide and conquer algorithms to construct a t-spanner are not asymptotically optimal, and therefore they are not comparable with the current well-known algorithms such as Θ-graph, WSPD-spanner and etc. Hence, our purpose to propose these algorithms is only presenting an application of the spanners merging problem. Moreover, we show that the proposed divide and conquer algorithm can be used to construct an $\mathcal{H}_k$-fault tolerant t-spanner for a set of n points in the plane with size of $O(n)$ in $O(n^2 \log n)$ time, and an $\mathcal{H}_k$-fault tolerant t-spanner for a set of n points in the plane with size of $O(n \log n)$ in $O(n \log^2 n)$ time. As we mentioned, Abam et al. [2] using SSPD, presented an algorithm to construct a $\mathcal{C}$-fault tolerant t-spanner with size of $O(n \log n)$ in $O(n \log^2 n)$ time for a set of n points in the plane. Since in the proposed algorithm we do not need to compute the SSPD for the point set, we claim that for the regions of $\mathcal{H}_k$, the proposed algorithm in comparison with Abam et al.'s algorithm is simpler, and also it improves the size of the resulting graph in comparison with Abam et al.'s algorithm. It is notable that the our results can be easily generalized to higher dimensions.

2 Spanners merging

In the following, we propose an algorithm, denoted by MergeSpanners, to solve the spanners merging problem. The algorithm is based on the Θ-graph algorithm. Note that since the Θ-graph algorithm can be generalized to higher dimensions [9], the all results of this paper can be easily generalized to higher dimensions. Hence, in this paper, we focus on the space $\mathbb{R}^2$.

Let $k > 1$ be an integer. We partition the plane into k cones with angle $\frac{2\pi}{k}$ and apex origin. We denote the collection of these k cones by $\mathcal{C}_k$. For each cone $C \in \mathcal{C}_k$, let ℓ_C be an arbitrary fixed line through origin and that is contained in C, and for any point p in the plane, let $C_p := C + p$ be the cone that is obtained by translating C such that its apex is p, similarly we have $\ell_{C_p} := \ell_C + p$. The idea behind the algorithm to solve the spanners merging problem is as follows: for each cone $C \in \mathcal{C}_k$ and for each vertex p in G_1, we connect p to a point $r \in C_p \cap P_2$ whose orthogonal projection onto ℓ_{C_p} is closest to p. For more details on the algorithm, see Algorithm 1.

Algorithm 1: MergeSpanners(G_1, G_2, t)

```
input: t-spanners $G_1 = (P_1, E_1)$ and
       $G_2 = (P_2, E_2)$, and suppose that
       $|P_1| \leq |P_2|$.
output: t-spanner $G = (P_1 \cup P_2, E)$ which is
        obtained by merge of $G_1$ and $G_2$.
1  Select an angle $0 < \theta < \pi/4$ such that
   $\cos\theta - \sin\theta \geq 1/t$;
2  Partition the plane into $k = \frac{2\pi}{\theta}$ cones. We denote
   the set of cones that partition the plane by $\mathcal{C}_k$;
3  $E := E_1 \cup E_2$;
4  foreach cone $C$ in $\mathcal{C}_k$ do
5      foreach $p$ in $P_1$ do
6          $C_p := C + p$;
7          $r :=$ a point of $C_p \cap P_2$ whose orthogonal
           projection onto $\ell_{C_p}$ is closest to $p$;
           /* let $\ell_C$ be a fixed ray that
              emanates from the origin and that
              is contained in $C$ and
              $\ell_{C_p} := \ell_C + p$. The ray $\ell_C$ can be
              chosen arbitrarily.                     */
8          $E := E \cup \{(p, r)\}$;
           /* $(p, r)$ is an edge.                   */
9      end
10 end
11 return $G(P_1 \cup P_2, E)$;
```

In the following, we prove the correctness of algorithm MergeSpanners. First, we present a useful geometric lemma.

Lemma 1 ([9]) *Let $k \geq 8$ be an integer, let $\theta = \frac{2\pi}{k}$, let p and q be two distinct points in the plane, and let C be the cone of $\mathcal{C}_k$ such that $q \in C_p$. Let r be a point in C_p such that the orthogonal projection of r onto the line ℓ_{C_p} is at least as close to p as the orthogonal projection of q onto ℓ_{C_p}. Then, $|rq| \leq |pq| - (\cos\theta - \sin\theta)|pr|$.*

Theorem 1 *If G_1 and G_2 are two t-spanners of P_1 and P_2, respectively, then the graph G generated by algorithm* MergeSpanners(G_1, G_2, t) *is a t-spanner of $P_1 \cup P_2$ and is constructed by only adding at most kn_1 edges between G_1 and G_2.*

Proof. By Algorithm 1, it is clear that G is constructed by only adding at most kn_1 edges between G_1 and G_2. Now, we prove that G is a t-spanner of $P_1 \cup P_2$. Suppose that p and q are two distinct points of $P_1 \cup P_2$. We claim that there is a t-path Q between p and q in G.

If $p, q \in P_1$ or $p, q \in P_2$, then the claim holds by considering that G_1 and G_2 are t-spanners of P_1 and P_2, respectively. Now, suppose without loss of generality that $p \in P_1$ and $q \in P_2$, and let C be a cone in $\mathcal{C}_k$ such that

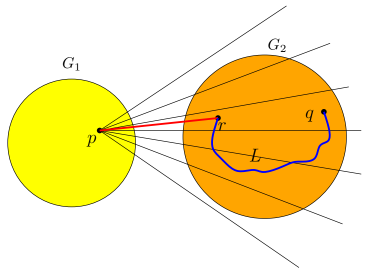


Figure 1: Illustrating of proof of Theorem 1.

$q \in C_p$. Note that according to the algorithm, we have $k = \frac{2\pi}{\theta}$ and $0 < \theta < \pi/4$ such that $\cos\theta - \sin\theta \geq 1/t$. Assume that the algorithm adds (p, r) to G, when it processes point p and cone C_p. If $r = q$, then clearly the claim holds. Now, suppose that $r \neq q$. Since G_2 is a t-spanner of P_2, there is a t-path L between r and q in G_2 (see Figure 1). Consider the path $Q := \{(p, r)\} \cup L$ between p and q in G. Using Lemma 1, we have

$$\begin{aligned}|Q| = |pr| + |L| &\leq |pr| + t|rq| \\ &\leq |pr| + t(|pq| - (\cos\theta - \sin\theta)|pr|) \\ &= t|pq| - (t(\cos\theta - \sin\theta) - 1)|pr| \\ &\leq t|pq| \quad (\text{since } \cos\theta - \sin\theta \geq 1/t).\end{aligned}$$

Hence, Q is a t-path between p and q in G. This completes the proof. □

For efficient implementation of algorithm MergeSpanners, we use the idea behind the efficient construction of the Θ-graph [9]. In the following, we present a plane sweep algorithm for the efficient implementation of MergeSpanners. We use the notations and terminologies in [9, Chapter 4].

Similar to the efficient implementation of the Θ-graph, we consider the problem Find-Leftmost-in-Translated-Halfplane as follows.

Problem (Find-Leftmost-in-Translated-Halfplane) [9]. Let h be a fixed nonvertical line through the origin. Maintain a set S of n points in a data structure that supports the following operations:

- MinAbove(p): Given a query point $p \in S$, compute a point with the minimum x-coordinate among all points in S that are above $h + p$.
- Insert(p): Insert an arbitrary point $p \in \mathbb{R}^2$ into S.
- Delete(p): Delete point p from S.

To solve the above problem, Narasimhan and Smid [9] used a binary search tree T. We refer the reader for more details on this problem and the tree T to the book [9].

Let D be the directed line orthogonal to the line h that passes through the origin. We define the following direction on D: D points toward the half-plane consisting of all points in $\mathbb{R}^2$ that are above h. The *order induced by* D on the points of S is defined as follows: let p and q be two points of S, and let p' and q' be their orthogonal projections onto D, respectively. If the vector $\overrightarrow{p'q'}$ has the same direction as D, then p is smaller than or equal to q in the order relation and denoted by $p \leq_D q$.

Lemma 2 [9, Lemma 4.1.9] *The binary search tree T solves the problem* Find-Leftmost-in-Translated-Halfplane. *It supports each of the operations* MinAbove, Insert, *and* Delete *in $O(\log n)$ time. This data structure has size $O(n)$ and can be built in $O(n \log n)$ time.*

Let C be a fixed cone of $\mathcal{C}_k$, and let h_1 and h_2 be the lines through the upper and lower bounding rays of C, respectively, and that we without loss of generality assume that the ray ℓ_C coincides with the positive x-axis. Let D_1 and D_2 be the lines through the origin that are orthogonal to h_1 and h_2, respectively. We define the following directions on D_1 and D_2: D_1 points toward the half-plane consisting of all points in $\mathbb{R}^2$ that are below h_1, and D_2 points toward the half-plane consisting of all points in $\mathbb{R}^2$ that are above h_2.

For efficient implementation of algorithm MergeSpanners, we propose an algorithm, denoted by PlaneSweepMerging (see Algorithm 2), that for each point $p \in P_1$, computes all edges (p, r) that correspond to C, where $r \in C_p \cap P_2$ and whose orthogonal projection onto ℓ_{C_p} is closest to p. If we repeat this for all k cones of $\mathcal{C}_k$, then we obtain all the new edges.

Let $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ be two geometric t-spanners such that $|P_1| \leq |P_2|$. For the sake of simplicity, we assume that $n_1 = |P_1|$ and $n_2 = |P_2|$. Now, we have

Theorem 2 *The running time of the algorithm* PlaneSweepMerging *is $O(n_1 \log n_1 + (n_1 + n_2) \log n_2)$.*

Proof. Let ℓ_i, k_i and $|T_i|$ be the values of ℓ, the number of points of L_2 which are inserted into T and the number of nodes of T in the i-th iteration of the algorithm, respectively. Clearly lines 5 and 6 of the algorithm take $O(n_1 \log n_1 + n_2 \log n_2)$ and $O(1)$ time, respectively. If we already sort the points of L_2 according to the order induced by D_1, using the binary search algorithm and by Lemma 2, the lines 9, 10 and 11 of the algorithm can be done in $O(\log \ell_i) + \sum_{j=1}^{k_i} O(\log(|T_i| + j)) = O(\log \ell_i) + O((k_i + 1)\log(k_i + 1 + |T_i|))$. Moreover, by Lemma 2, Line 16 takes $O(\log(|T_i|))$ time. Now, let $Time(n_1, n_2)$ be the time complexity of algorithm

PlaneSweepMerging. Hence, we have

$$\begin{aligned} Time(n_1, n_2) &= O(n_1 \log n_1 + n_2 \log n_2) + \sum_{i=1}^{n_1} O(\log \ell_i) \\ &+ \sum_{i=1}^{n_1} O((k_i + 1) \log(k_i + 1 + |T_i|)) \\ &+ \sum_{i=1}^{n_1} O(\log(|T_i|)). \end{aligned}$$

Since $|T_i| \le n_2$ and $\ell_i \le n_2$,

$$\begin{aligned} Time(n_1, n_2) &= O(n_1 \log n_1) + O(n_2 \log n_2) + O(n_1 \log n_2) \\ &+ O((n_1 + n_2) \log n_2) + O(n_1 \log n_2) \\ &= O(n_1 \log n_1 + (n_1 + n_2) \log n_2). \end{aligned}$$

This completes the proof. □

3 Applications

In this section, we present two applications of the spanners merging problem. First, we propose a divide and conquer algorithm to construct a geometric t-spanner for a given set of n points in the plane. Second, we propose a divide and conquer algorithm to construct an $\mathcal{H}_k$-fault tolerant t-spanner for a given point set in the plane, where $\mathcal{H}_k$ is the family of half-planes in the plane such that their boundary is parallel to any ray of any cone in $\mathcal{C}_k$.

3.1 Constructing a t-spanner

In this section, we propose some divide and conquer algorithms to construct a t-spanner for a given point set in the plane. The construction is as follows: let $c > 0$ be an integer. We first partition the point set into some subsets such that the cardinality of each subset is at most c. Then, we compute the complete graph on the each subset. Then, using algorithm PLANESWEEPMERGING (see Algorithm 2), we merge these subsets repeatedly. The way of merging theses subsets is important. In the following, we introduce two types of merging denoted by *serial merging*, *one-by-one merging*. Depending on the type of merging, the running time and size of the output differ.

Let S be set of n points in the plane and $c > 0$ be an integer constant. We without loss of generality suppose that the points are in general position in the sense that no tow points share the same x-coordinate. The partitioning is done by sorting the points of S according to the their x-coordinate and then separating the points of the sorted list as c-tuple members (see Algorithm 3). Figure 2 shows a set S partitioned with $c = 5$. Note that the time complexity of PARTITION is $O(n \log n)$. Now, let $S_1, S_2, \ldots, S_f$ be the partition of S, where f is the number of all subsets ($f \in O(n/c)$) and the cardinality of S_i is at most c, and let G_i be the complete graph of S_i. Clearly, G_i is a t-spanner of S_i, and the

Algorithm 2: PLANESWEEPMERGING(G_1, G_2, t)

```
   input: geometric $t$-spanners $G_1 = (P_1, E_1)$ and
          $G_2 = (P_2, E_2)$, and $t > 1$ (suppose that
          $|P_1| \le |P_2|$).
   output: the geometric $t$-spanner $G = (P_1 \cup P_2, E)$.
1  Select an angle $0 < \theta < \pi/4$ such that
   $\cos\theta - \sin\theta \ge 1/t$;
2  $k := 2\pi/\theta$;
3  $E := E_1 \cup E_2$;
4  foreach cone $C \in \mathcal{C}_k$ do
       /* Consider the lines $h_1, h_2, D_1, D_2$
          correspond to cone $C$.                    */
5      Sort the points of $P_1$ and $P_2$ according to the
       order induced by the directed line $D_1$ and $D_2$,
       respectively. Assume that the lists $L_1$ and $L_2$
       contain the sorted points of $P_1$ and $P_2$,
       respectively;
6      Initialize an empty data structure $T$ for solving
       problem FIND-LEFTMOST-IN-
       TRANSLATED-HALFPLANE using $h := h_2$;
7      $\ell := n_2$;
8      for $i = n_1$ down to 1 do
9          while $L_1[i] \le_{D_1} L_2[\ell]$ do
10             Insert point $L_2[\ell]$ into $T$;
11             $\ell := \ell - 1$;
12         end
13         if $\ell > 0$ then
14             $\ell := \ell + 1$;
15         end
           /* Note that the points
              $L_2[\ell], L_2[\ell+1] \ldots, L_2[n_2]$ are exactly
              the points of $P_2$ that are below
              the line $h_1 + L_1[i]$.              */
16         Find the point $r$ in $T$ that is above the line
           $h_2 + L_1[i]$ and whose $x$-coordinate is
           minimum; that is, it answers the query
           MINABOVE$(L_1[i])$;
17         $E := E \cup \{(L_1[i], r)\}$;
18     end
19 end
20 return $E$;
```

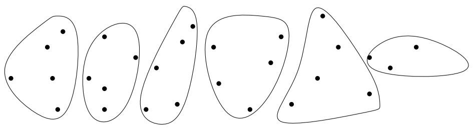


Figure 2: Partitioning with $c = 5$.

Algorithm 3: PARTITION(S, c)

```
1  Sort the points of $S$ increasingly according to their
   $x$-coordinate. Let $L$ be the sorted list;
2  for $h := 1$ to $\lfloor \frac{n}{c} \rfloor$ do
3      for $i = 1$ to $c$ do
4          Create new subset $S_h$;
5          Add $L[(h-1) \times c + i]$ into $S_h$;
6      end
7  end
8  if $n$ is not divisible by $c$ then
9      Create a new subset $S_h$ and add the remaining
       points of $S$ into $S_h$;
10 end
11 return all subsets $S_h$;
```

number of its edges is $O(c^2)$ and can be constructed in $O(c^2)$ time. In the following, we discuss about the two types of merging, in details.

3.1.1 Serial merging

In this merging, we first merge two graphs, then in each step we merge another graph with the resulting graph at the previous step. In particular, the serial merging is as follows: let G_i^* be the resulting graph of merging the graphs $G_1, \ldots, G_i$ using serial merging. Then, using algorithm PLANESWEEPMERGING (see Algorithm 2), we merge G_{i+1} with G_i^*. We call the final graph by $G = (S, E)$. By Theorem 1, it is clear that the graph G is a t-spanner for S. In the following, we compute the number of the edges of G, denoted by $|E|$, and the time complexity of the construction of G, denoted by $T(n)$. By Theorem 1, we have

$$|E| = \sum_{i=1}^{f} O(c^2) + \sum_{i=1}^{f-1} O(kc) = O((c+k)n), \quad (1)$$

and by Theorem 2, we have

$$\begin{aligned} T(n) &= \sum_{i=1}^{f} O(c^2) + \sum_{i=1}^{f-1} O\Big(c \log c + (i+1)c \log(ic)\Big) \\ &= O(fc^2) + O(cf^2 \log f) = O(cn + c^{-1}n^2 \log n). \end{aligned}$$

In the following, let S be a set of n points in the plane, c be a positive integer and $t > 1$ be a constant real number. Now, if in Equation (1), k be a constant integer, then we have

Theorem 3 *One can compute a t-spanner of S by a divide-and-conquer algorithm that splits the point set into c subsets in divide step and merging them in conquer step. The algorithm computes a t-spanner of S with $O(cn)$ edges in $O(cn + c^{-1}n^2 \log n)$ time.*

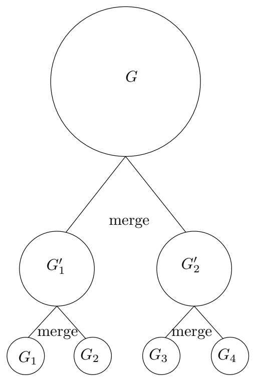


Figure 3: One-by-one merging.

In Theorem 3, if we choose $c = \log n$, then we have the following corollary.

Corollary 1 *One can compute a t-spanner of S by a divide-and-conquer algorithm that splits the point set into c subsets in divide step and merging them in conquer step. The algorithm computes a t-spanner of S with $O(n \log n)$ edges in $O(n^2)$ time.*

If c be a constant integer, then we have:

Corollary 2 *One can compute a t-spanner of S by a divide-and-conquer algorithm that splits the point set into c subsets in divide step and merging them in conquer step. The algorithm computes a t-spanner of S with $O(n)$ edges in $O(n^2 \log n)$ time.*

3.1.2 One-by-one Merging

One-by-one merging –similar to the merging sublists in the merge sort– is as follows: using the algorithm PLANESWEEPMERGING, we repeatedly merge G_i's to produce new subgraphs until there is only 1 subgraph remaining (see Figure 3). By Theorem 1, this will be a t-spanner of S. We denote the resulting graph by $G = (S, E)$. By Theorem 1, we have

$$\begin{aligned} |E| &= \sum_{i=1}^{f} O(c^2) + \sum_{i=1}^{\log f} O(\frac{f}{2^i}(k2^{i-1}c)) \\ &= O(fc^2) + O(kfc \log f) = O(cn + kn \log n) \\ &= O(kn \log n), \end{aligned}$$

and by Theorem 2, the time complexity $T(n)$ of the construction of G is

$$\begin{aligned} T(n) &= \sum_{i=1}^{f} O(c^2) + \sum_{i=1}^{\log f} O\left(\frac{3f}{2^i} 2^{i-1} c \log(2^{i-1}c)\right) \\ &= O(fc^2) + O(\frac{3fc}{2} \times \frac{\log f(\log f - 1)}{2}) \\ &= O(fc^2) + O(cf \log^2 f) = O(cn + n\log^2 n) \\ &= O(n \log^2 n). \end{aligned}$$

Now, we have:

Theorem 4 *One can compute a t-spanner of S by a divide-and-conquer algorithm that splits the point set into c subsets in divide step and merging them in conquer step. The algorithm computes a t-spanner of S with $O(n \log n)$ edges in $O(n \log^2 n)$ time.*

Note that this result improve the time complexity compared to the serial merging (see Corollary 1).

3.2 Constructing the region-fault tolerant spanners

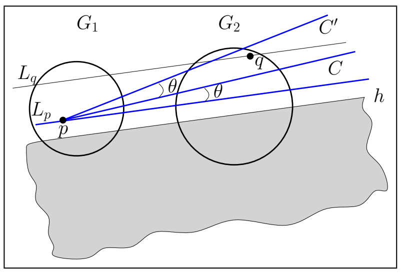
Let $k \geq 2$ be a real number and $\theta = \frac{2\pi}{k}$, and let $\mathcal{H}_k$ be the family of half-planes such that their boundary is parallel to any ray of any cone in $\mathcal{C}_k$.

In this section, we consider the problem of constructing an $\mathcal{H}_k$-fault tolerant t-spanner for a given point set in the plane. We first show the following theorem.

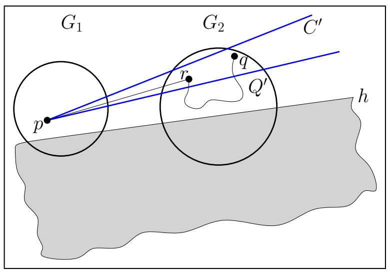
Theorem 5 *Let $G_1 = (P_1, E_1)$ and $G_2 = (P_2, E_2)$ be two $\mathcal{H}_k$-fault tolerant t-spanners, then the output G of the algorithm* MergeSpanners(G_1, G_2, t) *is an $\mathcal{H}_k$-fault tolerant t-spanner for $P_1 \cup P_2$.*

Proof. Let h be an arbitrary half-plane in $\mathcal{H}_k$. We show that for each pair of points $p, q \in P_1 \cup P_2$ outside h, there is a t-path Q connecting them in $G \ominus h$.

If $p, q \in P_1$ or $p, q \in P_2$, then the theorem holds by considering that G_1 and G_2 are $\mathcal{H}_k$-fault tolerant t-spanner. Now, suppose without loss of generality that $p \in P_1$ and $q \in P_2$, and suppose that L is a ray of a cone $C \in \mathcal{C}_k$ such that L is parallel to boundary of h. Now, consider L_p and L_q ($L_p := L + p$ and $L_q := L + q$). Since the boundary of h is parallel to L_p and L_q, we can suppose without loss of generality that the point q is in above line L_p (see Figure 4(a)) implies that there is a cone $C' \in \mathcal{C}_k$ such that $q \in C'_p$ and the boundary of h dose not intersect C'. Now, suppose that point r is the point that algorithm MergeSpanners adds it to the graph, when it processes the point p and cone C'_p. If $r = q$, then clearly the theorem holds. Now, suppose that $r \neq q$. Since G_2 is an $\mathcal{H}_k$-fault tolerant t-spanner, there is a t-path Q' between r and q in $G_2 \ominus h$ (see Figure 4(b)). Now, consider the path $Q := \{p, r\} \cup Q'$ in $G \ominus h$ connecting p and q in G. Then, using Lemma 1, we have



(a)



(b)

Figure 4: Illustrating of proof of Theorem 5.

$$\begin{aligned} |Q'| = |pr| + |L| &\leq |pr| + t|rq| \\ &\leq |pr| + t(|pq| - (\cos\theta - \sin\theta)|pr|) \\ &= t|pq| - (t(\cos\theta - \sin\theta) - 1)|pr| \\ &\leq t|pq|. \end{aligned}$$

Hence, Q is a t-path between p and q in $G \ominus h$. □

Note that according to Theorem 5 and since every complete graph is an $\mathcal{H}_k$-fault tolerant t-spanner, the output of the divide and conquer algorithm mentioned in previous section is an $\mathcal{H}_k$-fault tolerant t-spanner. Hence, as a corollary, we have:

Corollary 3 *There exist two divide and conquer algorithm to construct an $\mathcal{H}_k$-fault tolerant t-spanner for a given set of n points in the plane that ones takes $O(n^2 \log n)$ time and its output has $O(n)$ edges, and the other takes $O(n \log^2 n)$ time and its output has $O(n \log n)$ edges*

4 Conclusion

In this paper, we introduced the problem of merge of two geometric spanners, denoted by spanners merging problem, and then we proposed an algorithm that solves this problem in $O(n_1 \log n_1 + (n_1 + n_2) \log n_2)$ time. Furthermore, using the proposed algorithm, we proposed two divide and conquer algorithms to construct a t-spanner for a given point set in the plane. Moreover, we proposed two divide and conquer algorithms to construct an $\mathcal{H}_k$-fault tolerant t-spanner for a given set of n points in the plane that ones takes $O(n^2 \log n)$ time and its output has $O(n)$ edges, and the other takes $O(n \log^2 n)$ time and its output has $O(n \log n)$ edges.

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. *Computational Geometry*, 46(6):631 – 639, 2013.

[2] M. A. Abam, M. de Berg, M. Farshi, and J. Gudmundsson. Region-fault tolerant geometric spanners. *Discrete and Computational Geometry*, 41(4):556–582, 2009.

[3] M. A. Abam and S. Har-Peled. New constructions of SSPDs and their applications. *Computational Geometry*, 45(5Ũ6):200 – 214, 2012. Special issue: 26th Annual Symposium on Computation Geometry at Snowbird, Utah, USA.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[5] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Proceedings of the eighth Annual ACM Symposium on Computational Geometry*, pages 192–201. ACM, 1992.

[6] P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the second Annual ACM Symposium on Computational Geometry*, pages 169–177. ACM, 1986.

[7] D. Eppstein. Spanning trees and spanners. *Handbook of computational geometry*, pages 425–461, 1999.

[8] T. Lukovszki. *New results on geometric spanners and their applications.* Ph.D. thesis, Heinz Nixdorf Institute and Department of Mathematics and Computer Science, Paderborn University, Paderborn, Germany, 1999.

[9] G. Narasimhan and M. Smid. *Geometric spanner networks.* Cambridge University Press, 2007.

[10] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

[11] M. Smid. Closest point problems in computational geometry. *Handbook on Computational Geometry*, 1997.

A Faster Algorithm for the Minimum Red-Blue-Purple Spanning Graph Problem for Points on a Circle

Ahmad Biniaz* Prosenjit Bose* Ingo van Duijn † Anil Maheshwari* Michiel Smid*

Abstract

Consider a set of n points in the plane, each one of which is colored either red, blue, or purple. A red-blue-purple spanning graph (RBP spanning graph) is a graph whose vertices are the points and whose edges connect the points such that the subgraph induced by the red and purple points is connected, and the subgraph induced by the blue and purple points is connected. The minimum RBP spanning graph problem is to find an RBP spanning graph with minimum total edge length. We consider this problem for the case when the points are located on a circle. We present an algorithm that solves this problem in $O(n^2)$ time, improving upon the previous algorithm by a factor of $O(n)$.

1 Introduction

Let S be a set of n points in the plane that is partitioned into $\{R, B, P\}$. The points of R are colored red, the points of B are colored blue, and the points of P are colored purple. A *red-blue-purple spanning graph* (RBP spanning graph) on S is a graph whose vertices are the points of S and whose edges connect the points such that each of the subgraphs induced by $R \cup P$ and by $B \cup P$ are connected. In other words, if we remove the red points then the resulting subgraph is connected, and if we remove the blue points then the resulting subgraph is connected. One may think of the purple points belonging to both the red set and the blue set. The *minimum RBP spanning graph problem* is to compute an RBP spanning graph that has minimum weight (total edge length). See [3, 4] for applications of this problem. In this paper we consider the special case of this problem when the points of S are located on a circle.

In [3] it is claimed that the general case of the problem is NP-hard; this claim is based on a reduction from planar 3-SAT. However, in [4] it is claimed that the NP-hardness reduction of [3] is incorrect, and an $O(n^6)$-time exact algorithm for this problem is presented. The algorithm is based on the weighted matroid intersection. When the points of S are located on a line and given in sorted order, this problem can be solved in $O(n)$ time (see [3, 4]). If the points of S are located on a circle and given in circularly sorted order this problem can be solved in $O(n^3)$ time; specifically, it can be solved in $O(k^3 + n)$ time, where k is the number of purple points (see [3, 4]). For points on a circle, we present an algorithm that improves the running time to $O(k^2 + n)$.

2 Properties of Minimum RBP Spanning Graphs

In this section we review some properties of minimum RBP spanning graphs. Given a graph G with vertex set S, and a set $S' \subseteq S$, we denote by $G[S']$ the subgraph of G that is induced by S'.

For three sets R, B, and P of red, blue, and purple points, respectively, we denote by $G^*(R, B, P)$ a minimum RBP spanning graph on $R \cup B \cup P$; this is denoted by G^* when the triple (R, B, P) is clear from the context. As in [3, 4] we classify the edges of G^* into red, blue, and purple. An edge is *red* if it connects two red points, or a red point and a purple point. An edge is *blue* if it connects two blue points, or a blue point and a purple point. An edge is *purple* if it connects two purple points. Note that G^* does not contain any edge between a red point and a blue point. The subgraph $G^*[P]$ that is induced by the purple points is acyclic, because otherwise we could remove a purple edge from a cycle and reduce the weight of G^* without destroying the connectivity of $G^*[R \cup P]$ and $G^*[B \cup P]$. The subgraph $G^*[R \cup P]$ (resp. $G^*[B \cup P]$) is a spanning tree because otherwise we could remove a red edge (resp. blue edge) from a cycle without affecting the connectivity of $G^*[B \cup P]$ (resp. $G^*[R \cup P]$). We refer to $G^*[R \cup P]$ as the *red tree* and to $G^*[B \cup P]$ as the *blue tree.*

Every red edge in G^* is also an edge of a minimum spanning tree of $R \cup P$, because otherwise we could replace it by another red or purple edge of smaller weight. The corresponding statement holds for the blue edges. Thus, the red edges of G^* do not cross each other, and the blue edges of G^* do not cross each other. The corresponding statements do not hold for purple edges. There can be purple edges in G^* that are not present in any minimum spanning tree of the purple points. Moreover, a purple edge in G^* can cross $\Theta(|P|)$ other purple edges [3, 4]. In [3, 4] it is shown that the maximum

*School of Computer Science, Carleton University, Ottawa, Canada, ahmad.biniaz@gmail.com, {jit, anil, michiel} @scs.carleton.ca. Supported by NSERC.

†MADALGO, department of Computer Science, Aarhus University, Denmark, ivd@cs.au.dk. MADALGO is supported in part by DNRF84.

degree of a purple point in G^* is at most 18 and the maximum degree of a red point or a blue point is at most 6. Moreover, there exists an optimal graph in which the maximum degree of every purple point is at most 15 and the maximum degree of every red point or blue point is at most 5. The proofs for these degree constraints are inherited from the proofs of maximum degree constraints of minimum spanning trees of a point set in the plane.

3 The Algorithm

Let S be a set of n points on a circle C that are colored red, blue, or purple. Let R, B, and P denote the set of red, blue, and purple points of S, respectively. Let k denote the number of purple points, i.e., $k = |P|$. The problem is to compute a minimum RBP spanning graph for S. Although for points in the plane, and even for points in convex position, a purple edge can be crossed by other purple edges, for points on a circle, purple edges cannot be crossed by other purple edges. Based on this, Hurtado *et al.* [3, 4] presented a dynamic programming algorithm that solves this problem in $O(k^3 + n)$ time. We use a similar dynamic programming approach and improve the running time to $O(k^2 + n)$. First we review a lemma from [3, 4].

Lemma 1 (see [3, 4]) *Let S be a set of points on a circle, each one of which is colored either red, blue, or purple. Let G^* be a minimum RBP spanning graph for S. Then the following statements holds.*

1. *No purple edge of G^* can cross any other edge of G^*.*

2. *No red or blue edge of G^* can cross any segment between two purple points (which are not necessarily connected by an edge in G^*).*

3. *For any purple point p in S, let p' be the point on the circle diametrically opposite to p, and let SC be any of the two closed semicircles containing both p and p'. Then in G^*, p has at most one purple neighbor in SC, and thus at most two purple neighbors in total.*

3.1 The Dynamic Programming Algorithm

In this section we give an overview of the dynamic programming algorithm presented in [3, 4]. Assume that the points of S are circularly sorted. Let $p_1, \dots, p_k$ be the purple points in clockwise order. For any $1 \leqslant i \leqslant k$, let S_i be the set of red and blue points between p_i and p_{i+1}. Assume that all indices are taken modulo k.

Let G^* be a minimum RBP spanning graph for S. By Lemma 1 no edge of G^* that is incident to a point in S_i can cross segment $p_i p_{i+1}$ ($p_i p_{i+1}$ is not necessarily an edge of G^*). Thus, a solution for each set S_i can be computed independently. Moreover, this is analogous to the case when the points are on a line, and thus, it can be solved in linear time for all sets S_i.

For any two purple points p_i and p_j, Lemma 1 guarantees that if $p_i p_j$ is an edge in G^* then it cannot be crossed by any other edge of G^*. This introduces two independent subproblems, one to the left of the oriented segment $p_i p_j$, and one to the right. Each subproblem has four different types PC, RC, BC, and NC. In the PC-type, p_i and p_j are connected in both red and blue subgraphs. In the RC-type, p_i and p_j are connected in the red subgraph but disconnected in the blue subgraph; any solution for this type must connect p_i and p_j in its blue subgraph. In the BC-type, p_i and p_j are disconnected in the red subgraph but connected in the blue subgraph. In the NC-type, p_i and p_j are neither connected in the red subgraph nor in the blue subgraph. The algorithm maintains four tables, PC, RC, BC, and NC, each of size $O(k^2)$, that are indexed by pairs of purple points. Each entry $[i, j]$ of each table, stores the length of a minimum RBP spanning graph of the corresponding type for the point set $\{p_i, p_{i+1}, \dots, p_j\}$. Based on this, the length of an optimal solution can be found as

$$\min_{2 \leqslant j \leqslant k} \{PC[1, j] + NC[j, 1], NC[1, j] + PC[j, 1], RC[1, j] + BC[j, 1], BC[1, j] + RC[j, 1]\}.$$

Let $\Lambda \in \{P, R, B, N\}$. The entries of each table are filled in order, so that when it is time to compute the value of an entry $\Lambda[i, j]$, all the entries corresponding to smaller problems, i.e., subproblems introduced by purple pairs to the left of the oriented segment $p_i p_j$, have already been computed. In order to fill entry $\Lambda C[i, j]$, where $1 \leqslant i < j \leqslant k$, the following two cases are considered, and the one with minimum cost will be stored in $\Lambda C[i, j]$. See [3, 4] for more details.

1. p_i is connected to some purple point(s) in an optimal solution of the subproblem (i, j); recall that by Lemma 1, p_i can be connected to at most two purple points. Let p_h be the one in the sequence $p_{i+1}, \dots, p_j$ that is closer to p_j, see Figure 1(a). Note that p_i and p_h are connected in both red and blue subgraphs. Therefore, p_h and p_j must be connected in the same way as p_i and p_j. Since we do not know p_h, we try all possible candidates and keep the one minimizing the cost, i.e., $\Lambda C[i, j] = \min_{i+1 \leqslant h \leqslant j} \{PC[i, h] + \Lambda C[h, j] + |p_i p_h|\}$. This case takes $O(j - i)$ time.

2. p_i is not connected to other purple points in any optimal solution of the subproblem (i, j). Now Consider p_{i+1}. By Lemma 1, in an optimal solution no edge can cross the segment $p_i p_{i+1}$. Since no purple

edge is incident to p_i, the segment $p_{i+1}p_j$ cannot be crossed either; see Figure 1(b). Therefore, an optimal solution for the subproblem (i, j) can be computed by combining the solutions associated to the subproblems $(i, i+1)$ and $(i+1, j)$. See [3, 4] for more details. This case takes $O(1)$ time.

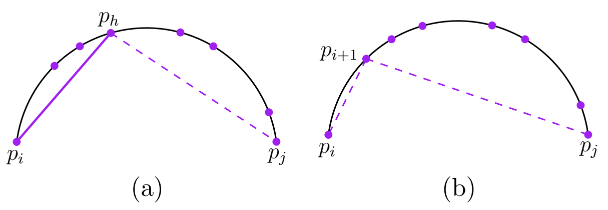


Figure 1: Solving subproblem (i, j): (a) p_i is connected to a purple point p_h, and (b) p_i is not connected to any purple point.

Based on the description above, the total running time of the algorithm is $O(k^3 + n)$.

3.2 Improving the Running Time

In this section we show how to improve the running time of the algorithm presented in Section 3.1 to $O(k^2 + n)$. First we prove Lemma 2 which plays an important role in this regard.

A *chord* of a circle is a straight line segment whose endpoints lie on the circle. For any two points p and q on C let $SC(p, q)$ denote the smaller arc of C that is determined by the chord pq.

Lemma 2 *Let S be a set of red, blue, and purple points located on a circle. Let P be the set of purple points. Let a and b be any two points of P such that $SC(a, b)$ contains at least two points of $P \setminus \{a, b\}$. Let $a^*, b^* \in P$ be the purple neighbors of a and b on $SC(a, b)$, respectively.*

1. *If $|aa^*|+|bb^*| < |ab|$, then ab does not belong to any minimum RBP spanning graph for S.*

2. *If $|aa^*|+|bb^*| = |ab|$, then there exists a minimum RBP spanning graph of S that does not contain ab.*

Proof. First we prove statement 1 of the lemma. The proof is by contradiction. Assume there exists a minimum RBP spanning graph G^* for S that contains ab. Recall that G^* consists of a red tree and a blue tree; moreover, the purple edges of G^* belong to both trees. Let R_a and R_b be the two red trees obtained by removing ab from G^*, such that $a \in R_a$ and $b \in R_b$. Let B_a and B_b be the two blue trees obtained in a similar way.

Claim 1: *It is not possible to have $a^* \in R_b$ and $b^* \in R_a$, or $a^* \in B_b$ and $b^* \in B_a$.*

We prove this claim for the case where $a^* \in R_b$ and $b^* \in R_a$; the proof for the other case is similar. By Lemma 1 (item 2) no red edge or blue edge can "jump" over a^* or b^*. Thus, in order to have $a^* \in R_b$ and $b^* \in R_a$ there must be two purple edges in G^* that cross; this contradicts Lemma 1 (item 1). This proves the claim.

By Claim 1, $a^* \in R_a$ or $b^* \in R_b$. Without loss of generality assume that $a^* \in R_a$. If $a^* \in B_a$, then by replacing the edge ab in G^* with the purple edge a^*b we obtain a valid RBP spanning graph that is smaller than G^* (note that a^*b is shorter than ab); see Figure 2(a). This contradicts the minimality of G^*. Assume that $a^* \in B_b$; see Figure 2(b). Then by Claim 1, we have $b^* \in B_b$. If $b^* \in R_b$, then by replacing the edge ab with the purple edge ab^* we obtain a valid RBP spanning graph that is smaller than G^*; see Figure 2(b). This contradicts the minimality of G^*. Assume that $b^* \in R_a$; see Figure 2(c). Then, by replacing ab with aa^* and bb^* we obtain a valid RBP spanning graph that is smaller than G^*. This contradicts the minimality of G^*.

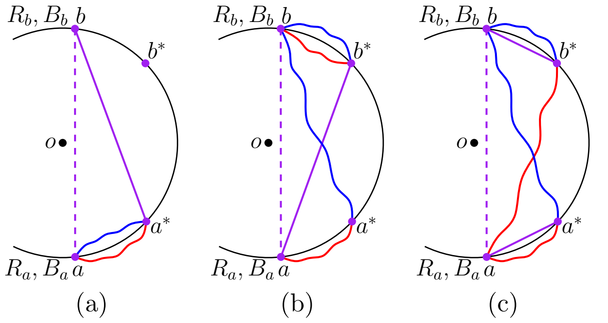


Figure 2: Proof of Lemma 2: (a) $a^* \in R_a$ and $a^* \in B_a$. (b) $a^* \in R_a$ and $a^* \in B_b$. (c) $a^* \in B_b$ and $b^* \in R_a$.

Now we prove statement 2 of the lemma. As we have seen in the proof of statement 1, if $|aa^*|+|bb^*| = |ab|$, in all cases we obtain an RBP spanning graph that is smaller than G^*, except for the case when we replace ab with aa^* and bb^*. Let G' be the graph that is obtained after replacing all such kind of edges. Since $|aa^*|+|bb^*| = |ab|$, G' has a weight equal to the weight of G^*. Moreover, G' does not contain ab. Thus, G' is a spanning graph that satisfies statement 2 of the lemma. □

We prove the following theorem in Section 4.

Theorem 3 *Let P be a set of points on a circle C. Let E_2 be the set of edges that contains an edge ab if and only if $a, b \in P$ and $|aa^*|+|bb^*| > |ab|$, where a^* and b^* are two points of P that are neighbors of a and b on the smaller arc of C that is determined by the chord ab, respectively. Then, no three edges of E_2 can pairwise cross.*

Theorem 4 *Let S be a set of n red, blue, and purple points located on a circle, and angularly sorted. A minimum RBP spanning graph of S can be computed in $O(k^2+n)$ time, where k is the number of purple points.*

Proof. We define three sets of edges, E_0, E_1, and E_2, on the purple points as follows. Let a and b be any pair of purple points. If $SC(a,b)$ has no point of $P \setminus \{a,b\}$ then add ab to E_0. If $SC(a,b)$ contains exactly one point of $P \setminus \{a,b\}$ then add ab to E_1. Let E_2 be the set of purple edges that is defined in the statement of Theorem 3. Let $E_P = E_0 \cup E_1 \cup E_2$. As a consequence of Lemma 2 there exists an optimal solution in which all the purple edges belong to E_P. Thus, in case 1 of the dynamic programming algorithm, instead of looking at all pairs (p_i, p_h) it is enough to only consider the pairs (p_i, p_h) that are connected by an edge in E_P. Each pair (p_i, p_h) is considered only for the subproblems that have p_i or p_h as an endpoint; the number of such subproblems is $O(k)$. Thus, the total time we spend for case 1 is $O(k|E_P|)$. Therefore the total running time of the algorithm is $O(k|E_P|+n)$.

Note that E_P can be computed in $O(k^2)$ time in the preprocessing phase. We are going to show that $|E_P|= O(k)$; this will complete the proof of the theorem. Each of E_0 and E_1 contains $k = |P|$ edges. By Theorem 3 no three edges of E_2 pairwise cross. Agarwal *et al.* [1] have shown that any graph with n vertices that can be drawn in the plane such that no three edges pairwise cross, has $O(n)$ edges. Thus, E_2 has $O(k)$ edges. Therefore, $|E_P|= O(k)$. □

4 Proof of Theorem 3

In this section we prove Theorem 3. First we prove some lemmas that will be used in the proof of the theorem.

4.1 Preliminary Results

Lemma 5 *Let aa' and bb' be two intersecting chords of a circle C such that the center of C is to the left of the oriented segment aa' and to the right of the oriented segment bb'. Then, $|ab|< |a'b'|$.*

Proof. Let o be the center of C. Let $\alpha = \angle aob$ and $\alpha' = \angle a'ob'$. The triangles $\triangle abo$ and $\triangle a'b'o$ are isosceles. Based on this and assuming that the radius of C is 1, we have $|ab|= 2\sin\left(\frac{\alpha}{2}\right)$ and $|a'b'|= 2\sin\left(\frac{\alpha'}{2}\right)$. See Figure 3. Let Q be the convex quadrilateral with vertices a, b, a', and b'. If o does not lie in Q (see Figure 3(a)), then we have $\alpha < \alpha'$. This implies that $|ab|< |a'b'|$. Assume o lies in Q; see Figure 3(b). Let a'' and b'' be two points on C such that aa'' and bb'' are two diameters of C. Let $\beta = \angle a''ob''$. Note that $\beta = \alpha$. Moreover, we have $\beta < \alpha'$. This implies that that $\alpha < \alpha'$, and consequently $|ab|< |a'b'|$. □

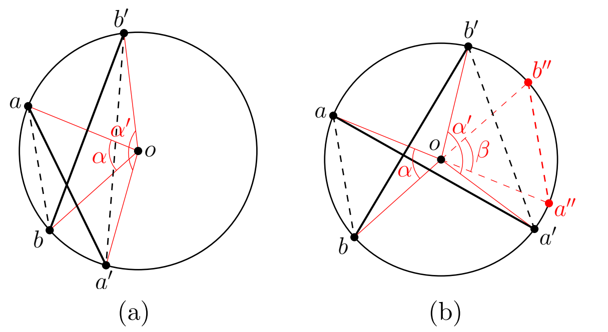


Figure 3: Proof of Lemma 5: (a) o does not lie in convex quadrilateral a, b, a', b', and (b) o lies in convex quadrilateral a, b, a', b'.

Lemma 6 *Let b, a, and p be three points on a circle C, in clockwise order, such that the center of C is to the right side of the oriented segment bp. Let b'' be the point such that bb'' is a diameter of C. Let p' be a point on $SC(p, b'')$. Then $|ap'|-|ap|> |bp'|-|bp|$.*

Proof. Refer to Figure 4. Let $C(a, |ap|)$ be the circle of radius $|ap|$ that is centered at a, and let $C(b, |bp|)$ be the circle of radius $|bp|$ that is centered at b. Since a, p, and p' are on the same side of the line through bb'', we have $|ap'|> |ap|$ and $|bp'|> |bp|$. Let a' be the intersection point of ap' with $C(a, |ap|)$, and let b' be the intersection point of bp' with $C(b, |bp|)$. Then $|aa'|= |ap|$ and $|bb'|= |bp|$. Thus, $|ap'|-|ap|= |a'p'|$ and $|bp'|-|bp|= |b'p'|$. In order to prove the statement of the lemma it suffices to show that $|a'p'|> |b'p'|$. Let ℓ be the line that is tangent to $C(b, |bp|)$ at b'. Observe that a' and p' are on different sides of ℓ. Thus, in triangle $\triangle a'b'p'$, the angle $\angle a'b'p'$ is at least $\frac{\pi}{2}$. This implies that $a'p'$ is the longest side of $\triangle a'b'p'$. Therefore, $|a'p'|> |b'p'|$; this completes the proof of the lemma. □

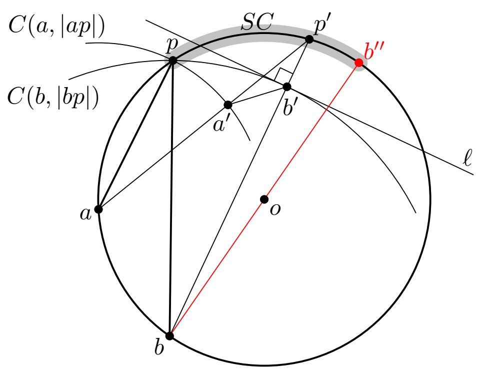


Figure 4: Proof of Lemma 6.

Lemma 6 can be restated as follows. If we fix the position of a and b, then by moving p towards b'', the length of ap increases more than the length of bp.

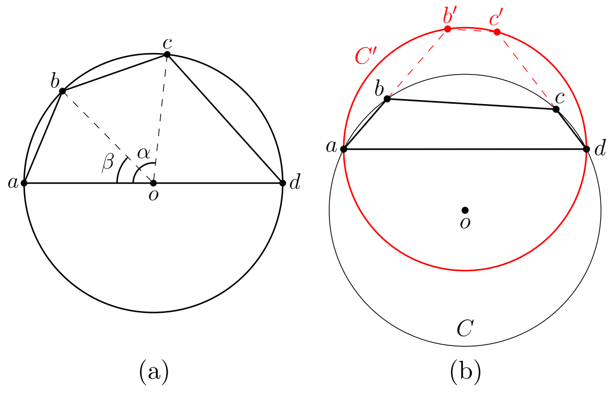


Figure 5: (a) Proof of Lemma 7 and Lemma 9. (b) Proof of Lemma 9.

Lemma 7 *Let $0 < \alpha \leqslant \pi$ be fixed, and let C be a circle that is centered at o. Let a, b, and c be three points on C, in clockwise order, such that $\angle aoc = \alpha$. Then, $|ab|+|bc|$ is maximum when $\angle aob = \angle boc = \frac{\alpha}{2}$.*

Proof. Let $f = |ab|+|bc|$, and let $\beta = \angle aob$. See Figure 5(a). Since the triangles $\triangle aob$ and $\triangle aoc$ are isosceles, we have $f = 2\sin\left(\frac{\beta}{2}\right) + 2\sin\left(\frac{\alpha-\beta}{2}\right)$. By taking the derivative of f with respect to β, we can see that f is maximum when $\beta = \frac{\alpha}{2}$, and thus, $\angle aob = \angle boc = \frac{\alpha}{2}$. $\square$

The following is a corollary of Lemma 7.

Corollary 1 *Let C be a circle that is centered at o. Let a, b, and c be three points on C, in clockwise order, such that $\angle aoc \leqslant \frac{2\pi}{3}$. Then $|ab|+|bc| \leqslant |ao|+|co|$.*

Proof. By Lemma 7, $|ab|+|bc|$ is maximum when $\angle aob = \angle boc$, and thus, both these angles are at most $\frac{\pi}{3}$. This implies that $|ab| \leqslant |ao|$ and $|bc| \leqslant |co|$, which proves the claim. $\square$

The following theorem is a restatement of Theorem 7.11 in [2].

Theorem 8 (See [2]) *If C_1 and C_2 are convex polygonal regions with $C_1 \subseteq C_2$, then the length of the boundary of C_1 is at most the length of the boundary of C_2.*

Lemma 9 *Let a, b, c, and d be four points on a circle C, in clockwise order, such that the center of C is on or to the right side of the oriented segment ad. Then $|ab|+|bc|+|cd| \leqslant \frac{3}{2} \cdot |ad|$.*

Proof. Without loss of generality assume C is centered at o and has radius 1. We consider two cases: (i) o is on ad, and (ii) o is not on ad. First, we prove case (i). Then, we show how to reduce case (ii) to case (i). Assume o is on ad, that is, ad is a diameter of C, and thus, $|ad| = 2$. See Figure 5(a). If we fix the position of c on C, then by Lemma 7, $|ab|+|bc|$ is maximum when $\angle aob = \angle boc$. If we fix the position of b on C, then by Lemma 7, $|bc|+|cd|$ is maximum when $\angle boc = \angle cod$. Therefore, $|ab|+|bc|+|cd|$ is maximum when $\angle aob = \angle boc = \angle cod = \frac{\pi}{3}$, and thus, $|ab| = |bc| = |cd| = |ao| = |od|$. This implies the statement of the lemma for case (i).

Now we show how to handle case (ii). Assume o is not on ad, and thus, ad is not a diameter of C. We show how to reduce this case to case (i). Follow Figure 5(b). Let C' be the circle with diameter ad. Since C and C' intersect only at the two points a and d, we argue that b and c are in the interior of C'. Extend ab and dc to intersect C' at b' and c', respectively. Now we consider two cases depending on whether bb' and cc' intersect or not. In the former case, let o' be the intersection point of bb' and cc'. By Theorem 8 we have $|ab|+|bc|+|cd| \leqslant |ao'|+|o'd|$. Since o' is in the interior of C' then $|ao'|+|o'd| \leqslant \sqrt{2} \cdot |ad|$; and we are done with this case. In the latter case, by Theorem 8 we have $|ab|+|bc|+|cd| \leqslant |ab'|+|b'c'|+|c'd|$. As we have seen in case (i), $|ab'|+|b'c'|+|c'd| \leqslant \frac{3}{2} \cdot |ad|$; which completes the proof of the lemma. $\square$

4.2 Proof of Theorem

Recall that P is a set of points on a circle C. The edge set E_2 contains an edge ab if and only if $a, b \in P$ and $|aa^*|+|bb^*| > |ab|$, where a^* and b^* are the two points of P that are neighbors of a and b on the smaller arc of C that is determined by the chord ab, respectively. Without loss of generality assume C is centered at o and has radius 1. Based on the definition of E_2, the following observation is valid.

Observation 1 *For any edge $ab \in E_2$, we have $|aa^*| > \frac{1}{2} \cdot |ab|$ or $|bb^*| > \frac{1}{2} \cdot |ab|$.*

Corollary 2 *For any edge $ab \in E_2$, we have $|a^*b^*| < \frac{1}{2} \cdot |ab|$.*

Proof. Since $ab \in E_2$, we have $|aa^*|+|bb^*| > |ab|$. By Lemma 9 we have $|aa^*|+|a^*b^*|+|b^*b| \leqslant \frac{3}{2} \cdot |ab|$ where a, a^*, b^*, and b play the role of a, b, c, and d, respectively. This implies that $|a^*b^*| < \frac{1}{2} \cdot |ab|$. $\square$

Now we have all the tools that we need to prove Theorem 3. For the sake of contradiction assume that three edges aa', bb', and cc' of E_2 are pairwise crossing. Observe that if we remove all points of P except a, b, c, a', b', c', and then recompute E_2, the edges aa', bb', and cc' will remain in E_2. Thus, without loss of generality we assume that $P = \{a, b, c, a', b', c'\}$. Moreover, assume a, b, c, a', b', c' appear in clockwise order on C. Let $\triangle$ be the triangle whose vertices are the intersection points of aa', bb', and cc'. We differentiate between the following two cases: (i) o is in the interior

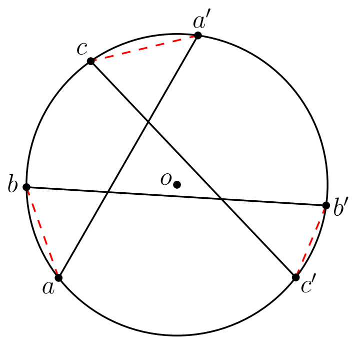


Figure 6: Proof of case (i) in Theorem 3.

of $\triangle$, and (ii) o is not in the interior of $\triangle$. We will get contradictions in both cases.

First we handle case (i). Refer to Figure 6. Since aa', bb', and cc' are edges of E_2, we have $|ab|+|a'c|>|aa'|$, $|ab|+|b'c'|>|bb'|$, and $|b'c'|+|a'c|>|cc'|$. By adding up these three inequalities, we get

$$|ab|+|a'c|+|b'c'|>\frac{|aa'|+|bb'|+|cc'|}{2}. \qquad (1)$$

By Lemma 5 we have $|ab|<|a'b'|$, $|a'c|<|ac'|$, and $|b'c'|<|bc|$. Adding up these three inequalities implies

$$|ab|+|a'c|+|b'c'|<|a'b'|+|ac'|+|bc|. \qquad (2)$$

In view of Corollary 2, we have $|bc|<\frac{1}{2}\cdot|aa'|$, $|ac'|<\frac{1}{2}\cdot|bb'|$, and $|a'b'|<\frac{1}{2}\cdot|cc'|$. This implies

$$|bc|+|ac'|+|a'b'|<\frac{|aa'|+|bb'|+|cc'|}{2}.$$

This and Inequality (2) imply

$$|ab|+|a'c|+|b'c'|<\frac{|aa'|+|bb'|+|cc'|}{2},$$

which contradicts Inequality (1); this is a contradiction for case (i).

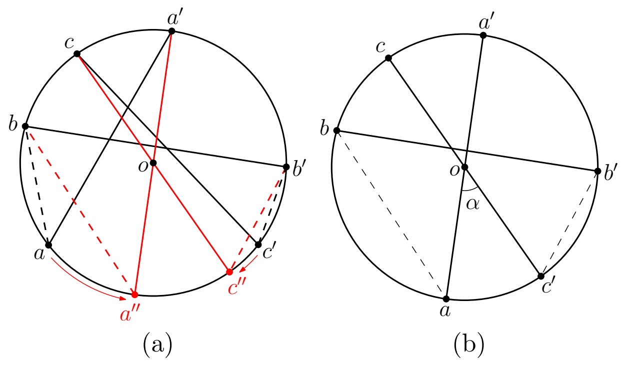


Figure 7: Proof of case (ii) in Theorem 3.

Now we are going to handle case (ii) where o is not in the interior of the triangle formed by the intersection of aa', bb', and cc'. Without loss of generality assume that o is on or to the right side of all the oriented segments aa', bb', and cc'; see Figure 7(a). Since aa', bb', and cc' are edges of E_2, we have $|ab|+|a'c|>|aa'|$, $|bc|+|a'b'|>|bb'|$, and $|a'c|+|b'c'|>|cc'|$. By adding up these three inequalities, we get

$$|ab|+|bc|+|a'b'|+|b'c'|+2|a'c|>|aa'|+|bb'|+|cc'|. \qquad (3)$$

Let a'' and c'' be the two points on C such that $a'a''$ and cc'' are diameters of C. By Lemma 6, $|a''b|-|ab|>|a''a'|-|aa'|$ and $|b'c''|-|b'c'|>|cc''|-|cc'|$. By adding these two inequalities to Inequality (3) we get

$$|a''b|+|bc|+|a'b'|+|b'c''|+2|a'c|>|a''a'|+|bb'|+|cc''|. \qquad (4)$$

Thus, if o is on or to the right side of all the oriented segments aa', bb', and cc', then Inequality (4) is valid. In fact, Inequality (4) is the same as Inequality (3) where a'' and c'' play the role of a and c', respectively. Therefore, without loss of generality, from now on we assume that a is on a'' and c' is on c'', that is, aa' and cc' are two diameters of C.

Let $\alpha=\angle aoc'=\angle coa'$. We claim that $\alpha\leqslant\frac{\pi}{3}$. Assume $\alpha>\frac{\pi}{3}$. This implies that $\angle aoc\leqslant\frac{2\pi}{3}$ and $\angle a'oc'\leqslant\frac{2\pi}{3}$. By Corollary 1 we have $|ab|+|bc|\leqslant|ao|+|oc|$ and $|a'b'|+|b'c'|\leqslant|a'o|+|oc'|$. As a consequence of Corollary 2, for edge bb' we have $2|a'c|<|bb'|$. By adding these three inequalities we get

$$\begin{aligned}&|ab|+|bc|+|a'b'|+|b'c'|+2|a'c|\\&<|ao|+|oc|+|a'o|+|oc'|+|bb'|=|aa'|+|bb'|+|cc'|,\end{aligned}$$

which contradicts Inequality (3). Therefore, $\alpha\leqslant\frac{\pi}{3}$.

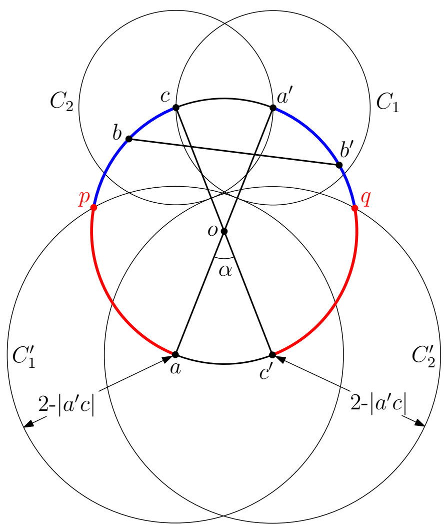


Figure 8: Proof of case (ii) in Theorem 3.

Recall that C has radius 1, and thus, $|aa'|=|cc'|=2$. Consider two circles $C_1=C(a',|a'c|)$ and $C_2=$

$C(c, |a'c|)$. Let C_1' be the circle that is centered at a and touches C_1, i.e., C_1' has radius $2 - |a'c|$. Similarly, let C_2' be the circle that is centered at c' and touches C_2. See Figure 8. Let $\widehat{ac}$ (resp. $\widehat{a'c'}$) be the smaller arc of C that has endpoints a to c (resp. a' and c'). Let p be the intersection point of C_1' and $\widehat{ac}$. Let $\widehat{ap}$ and $\widehat{pc}$ be the two sub-arcs of $\widehat{ac}$. Similarly, let q be the intersection point of C_2' and $\widehat{a'c'}$, and let $\widehat{a'q}$ and $\widehat{qc'}$ be the two sub-arcs of $\widehat{a'c'}$.

If b is in the interior of $\widehat{ap}$ then $|ab|+|a'c| < 2 = |aa'|$, which contradicts the existence of aa' in E_2. Thus $b \in \widehat{pc}$, and similarly, $b' \in \widehat{a'q}$. We are going to show that $|bc|+|a'b'| \leqslant |bb'|$; this will contradict the existence of bb' in E_2.

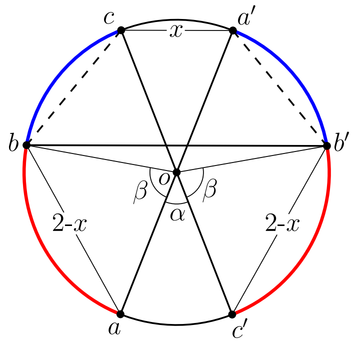


Figure 9: Proof of case (ii) in Theorem 3.

Since $bb' \in E_2$ we have $|bc|+|a'b'| > |bb'|$. By Lemma 6 if we move b towards p, then $|bc|$ increases more than $|bb'|$. Similarly, if we move b' towards q, then $|a'b'|$ increases more than $|bb'|$. Therefore, $|bc|+|a'b'| > |bb'|$ holds after moving b to p, and b' to q. Thus, from now we assume $b = p$ and $b' = q$. See Figure 9. Note that all the triangles $\triangle aob$, $\triangle boc$, $\triangle coa'$, $\triangle a'ob'$, $\triangle b'oc'$, and $\triangle bob'$ are isosceles. Let $x = |a'c|$, and thus, $|ab| = |b'c'| = 2 - |a'c| = 2 - x$. Note that $x = 2\sin\left(\frac{\alpha}{2}\right)$. Let $\beta = \angle aob = \angle b'oc'$. Then, $\beta = 2\arcsin\left(\frac{2-x}{2}\right) = 2\arcsin\left(1 - \sin\left(\frac{\alpha}{2}\right)\right)$. Note that $\angle bob' = 2\pi - \alpha - 2\beta$. Thus,

$$|bb'| = 2\sin\left(\pi - \frac{\alpha}{2} - \beta\right) = 2\sin\left(\frac{\alpha}{2} + \beta\right).$$

Moreover, $\angle boc = \angle a'ob' = \pi - \alpha - \beta$. Thus,

$$|bc| = |a'b'| = 2\sin\left(\frac{\pi - \alpha - \beta}{2}\right) = 2\cos\left(\frac{\alpha + \beta}{2}\right).$$

Now we show that $|bc|+|a'b'| \leqslant |bb'|$, which contradicts the existence of bb' in E_2. In order to show this, it suffices to prove that

$$4\cos\left(\frac{\alpha+\beta}{2}\right) \leqslant 2\sin\left(\frac{\alpha}{2} + \beta\right), \tag{5}$$

where $\beta = 2\arcsin\left(1 - \sin\left(\frac{\alpha}{2}\right)\right)$, for all $0 < \alpha \leqslant \frac{\pi}{3}$. Inequality (5) simplifies to

$$\begin{aligned} &2\sqrt{1 - \sin^2\left(\frac{\alpha}{2}\right)}\sqrt{1 - \left(1 - \sin\left(\frac{\alpha}{2}\right)\right)^2} \\ &+ 2\left(1 - \sin\left(\frac{\alpha}{2}\right)\right)^2 + \sin\left(\frac{\alpha}{2}\right) \\ &\leqslant 3, \end{aligned} \tag{6}$$

where $0 < \alpha \leqslant \frac{\pi}{3}$. Let $u = \sin\left(\frac{\alpha}{2}\right)$. Then, Inequality (6) simplifies to

$$4u^2 - 4u + 1 \geqslant 0, \tag{7}$$

where $0 \leqslant u \leqslant \frac{1}{2}$; it is easy to verify that Inequality (7) is valid in this range of u. This contradicts the fact that $|bc|+|a'b'| > |bb'|$, and hence the existence of bb' in E_2; this is a contradiction for case (ii). This completes the proof of Theorem 3.

References

[1] P. K. Agarwal, B. Aronov, J. Pach, R. Pollack, and M. Sharir. Quasi-planar graphs have a linear number of edges. *Combinatorica*, 17(1):1–9, 1997.

[2] R. V. Benson. *Euclidean geometry and convexity*. McGraw-Hill, 1966.

[3] F. Hurtado, M. Korman, M. J. van Kreveld, M. Löffler, V. S. Adinolfi, R. I. Silveira, and B. Speckmann. Colored spanning graphs for set visualization. In *21st Int. Symp. on Graph Drawing*, pages 280–291, 2013.

[4] F. Hurtado, M. Korman, M. J. van Kreveld, M. Löffler, V. Sacristán, A. Shioura, R. I. Silveira, B. Speckmann, and T. Tokuyama. Colored spanning graphs for set visualization. *Comput. Geom., special issue in Memoriam: Ferran Hurtado*, page to appear.

Partitions of planar point sets into polygons

Ajit Arvind Diwan [*] Bodhayan Roy [†]

Abstract

In this paper, we characterize planar point sets that can be partitioned into disjoint polygons of arbitrarily specified sizes. We provide an algorithm to construct such a partition, if it exists, in polynomial time. We show that this problem is equivalent to finding a specified 2-factor in the visibility graph of the point set. The characterization for the case where all cycles have length 3 also translates to finding a K_3-factor of the visibility graph of the point set. We show that the generalized problem of finding a K_k-factor of the visibility graph of a given point set for $k \geq 5$ is NP-hard.

1 Introduction

Partitioning of point sets is a well studied topic in Computational Geometry. Let P be a finite set of points in the plane. A partition of P is called a *convex partition* if P is partitioned into j subsets $S_1, S_2, \dots, S_j$ such that all the points of S_i form the vertices of a convex polygon [8]. Problems concerning such partitions have been studied, and bounds on the number of sets required for such a disjoint partition have been established [8, 14, 1]. In this paper, we study the following two related partitions of P, and the equivalent problem on the visibility graph of P.

A partition of P into subsets $S_1, S_2, \dots, S_j$ is said to be a *cycle partition* of P, when the points of each S_i can be joined by straight line segments to form a simple polygon, i.e. no S_i has all points collinear. We say that two points p_i and p_j of P are *visible* to each other if the line segment p_ip_j does not contain any other point of P. If a point $p_k \in P$ lies on the segment p_ip_j connecting two points p_i and p_j in P, we say that p_k blocks the visibility between p_i and p_j, and p_k is called a *blocker* in P. A partition of P into subsets $S_1, S_2, \dots, S_j$ is said to be a *clique partition* of P, when all the points of each S_i are mutually visible, i.e. no S_i has three collinear points, and no two points of S_i are blocked by points from any other S_l.

The *visibility graph* (also called the *point visibility graph* denoted as PVG) of P is defined by associating a vertex v_i with each point p_i of P and such that (v_i, v_j) is an undirected edge of the PVG if p_i and p_j are visible to each other. Point visibility graphs have been studied in the contexts of construction [3, 4], recognition [6, 7, 2, 12], connectivity [10], chromatic number and clique number [9, 11].

Let H be a connected graph. For a given graph G, an H-factor of G is a spanning subgraph of G whose components are isomorphic to H. Thus, a C_k-factor of G is a spanning subgraph of G whose components are isomorphic to the cycle on k vertices. Similarly, a *K_k-factor* of G is a spanning subgraph of G whose components are isomorphic to the clique on k vertices. To decide whether or not a graph G on kn vertices has a K_k-factor or a C_k-factor is NP-hard for $k \geq 3$ [5].

We say that a cycle partition of a point set is *disjoint* when no two of the polygons enclosed by the cycles intersect with respect to vertices, edges or area. In this paper, in Section 2, we study disjoint cycle partitions of point sets. In Section 2.1 we study the special case where all cycles are of length 3. We provide a necessary and sufficient condition for this case. The condition also shows that all point sets that admit any partition into cliques of size 3, also admit such a disjoint cycle partition. In Section 2.2 we study the generalized disjoint cycle partitions except the case mentioned above. We provide a different necessary and sufficient condition for it, thereby completely characterizing all point sets that admit a disjoint cycle partition. In Section 2.3 we show that a point set admits a disjoint cycle partition if and only if its visibility graph admits a corresponding 2-factor. In Section 3 we study the problem of clique partitions of point sets. If all cliques are of size 3, then this problem is the same as the special case in cycle partition. We prove that for all cliques of size k, $k \geq 5$, the problem becomes NP-hard. Finally, in Section 4, we conclude with some remarks and open questions.

[*]Department of Computer Science and Engineering, Indian Institute of Technology Bombay, aad@cse.iitb.ac.in

[†]Department of Computer Science and Engineering, Indian Institute of Technology Bombay, broy@cse.iitb.ac.in

2 Disjoint cycle partitions

Let P be a given set of finitely many points in the plane. Denote the convex hull of a point set P by $CH(P)$. It is well-known that every point set with an even number of points has a non-crossing matching. In this section, we consider the existence of disjoint polygons of specified sizes in a given point set. l

2.1 Disjoint triangle partition

A cycle partition of P is said to be a *triangle partition* if all cycles have length 3. A subset I of P such that no two points of I are visible from each other is called an *independent set* of P. Observe that I also induces an independent set in the visibility graph of P. Here we provide a characterization of all point sets that admit a disjoint triangle partition. We first characterize sets of $3n$ points that contain an independent set of size $n+1$.

Lemma 1 *Let P be a set of $3n$ points that contains an independent set I of size $n+1$. Then one of the following must hold:*

1. *The points in I are collinear.*
2. *The points in I occur on the boundary of $CH(P)$ and $CH(I) = CH(P)$. $CH(P)$ has at most 4 vertices and the boundary of $CH(P)$ contains exactly $2n+2$ points of P, with every alternate point in I. Further, every subset of 5 points in I must contain 3 collinear points.*

Proof. Consider a maximal plane graph G with vertex set the points in I. Suppose there are h points of I on the boundary of $CH(I)$. Then the number of edges in G is $3n-h$. Since I is an independent set in P, every such edge in G must contain a blocker in P which is not in I. Since there are at most $2n-1$ such blockers, we must have $3n-h \leq 2n-1$, or $h \geq n+1$. Thus all points of I are on the boundary of $CH(I)$, and every edge in G must contain exactly one blocker. This implies that $CH(P) = CH(I)$, and there are exactly $2n+2$ points of P on the boundary of $CH(P)$, with every alternate point in I.

Suppose there are 5 points in I such that no 3 are collinear. Since all points in I are on the boundary of $CH(I)$, we can choose 5 such points such that their convex hull does not contain any other point of I (Figure 1 (a)). Now choose edges in the graph G such that these 5 points form a strictly convex polygon in G. There will be exactly two edges of G and two blockers contained inside this polygon. However, two blockers cannot block the visibility between all pairs of non-adjacent vertices of a strictly convex pentagon, contradicting the fact that I is an independent set (Figure 1 (b)).

This also implies that $CH(P)$ must have at most 4 vertices, otherwise we can find a subset of 5 points in I such that no 3 are collinear. □

Theorem 2 *A set P of $3n$ points in the plane admits a disjoint triangle partition iff P does not contain an independent set of size $n+1$.*

Proof. Suppose P admits a disjoint triangle partition and contains an independent set of size $n+1$. Then some two points in the independent set must be in the same triangle in the triangle partition. Since the triangles are disjoint, these two points must be visible to each other, contradicting the fact that they are in an independent set.

Suppose P does not contain an independent set of size $n+1$. We show that P has a disjoint triangle partition. The proof is by induction on n. For $n=1$, this is trivial. Suppose $n \geq 2$.

Let p_i be any vertex of $CH(P)$, p_j the point in P that follows p_i on the boundary of $CH(P)$ in clockwise order, and p_k the point that precedes p_j on the boundary of $CH(P \setminus \{p_i\})$. Note that all points in $P \setminus \{p_i\}$ cannot be collinear, otherwise there are $2n+1$ collinear points in P. Denote the triangle $p_ip_jp_k$ by $\Delta(p_i)$ and let $P' = P \setminus \{p_i, p_j, p_k\}$.

Clearly the triangle $\Delta(p_i)$ is disjoint from $CH(P')$. If P' does not contain an independent set of size n, then by induction, P' has a disjoint triangle partition, which, along with the triangle $\Delta(p_i)$, gives a disjoint triangle partition of P. Suppose P' contains an independent set of size n. Then P' must satisfy one of the conditions given by Lemma 1.

First suppose P' contains $2n-1$ collinear points on some line L. Since P has no independent set of size $n+1$, L can contain at most $2n$ points of P. Suppose P has $2n$ collinear points and let these be $p_1, p_2, \dots, p_{2n}$ in left to right order along L. We now form a disjoint triangle partition of P directly. For $i=1$ to n, we choose the triangle p_{2i-1}, p_{2i}, q_i, where q_i is a point in P not in L, that has not been included in any earlier triangle, such that the angle p_{2i-1}, p_{2i}, q_i is as small as possible, and subject to this condition, q_i is as close to p_{2i} as possible. This choice of q_i ensures that the triangles chosen are disjoint.

If P has $2n-1$ collinear points, we use the same procedure for $i=1$ to $n-1$. Now we are left with p_{2n-1} and two points p, q that are not in L. If both p, q are on the same side of L and p, q, p_{2n-1} are not collinear, they form a triangle to complete the disjoint triangle partition. If they are on different sides of L, say p is above L, we consider the triangle $p_{2n-3}, p_{2n-2}, q_{n-1}$ formed earlier. If q_{n-1} is above L, we replace this triangle by p_{2n-3}, p_{2n-2}, q and add the triangle p, q_{n-1}, p_{2n-1}. A similar argument holds if q_{n-1} is below L.

Finally, suppose p, q are on the same side of L, say above, and q lies on the segment pp_{2n-1}. If q_{n-1} lies below L, we use the triangles p_{2n-3}, p, q and $p_{2n-2}, p_{2n-1}, q_{n-1}$. If q_{n-1} is above L but does not lie in the segment $p_{2n-3}p$, we use the triangles p_{2n-3}, q_{n-1}, p and p_{2n-2}, p_{2n-1}, q. The only remaining possibility is that q_{n-1} blocks p_{2n-3} from p. Now consider the previous triangle $p_{2n-5}, p_{2n-4}, q_{n-2}$. By a similar argument, if q_{n-2} does not block p from p_{2n-5}, we can modify the triangles to find a disjoint triangle partition. Continuing this way, we either get a disjoint triangle partition, or p must be blocked from p_{2i-1} by q_i for $1 \leq i \leq n-1$. But this implies P contains an independent set of size $n+1$, a contradiction.

Now suppose $CH(P')$ satisfies the second condition in Lemma 1. Suppose $CH(P')$ has 3 vertices p, q, r in clockwise order. We consider different cases based on the possible vertices of $CH(P)$.

Suppose p, q, r are also vertices of $CH(P)$ and p_i occurs between q and r, in clockwise order, on the boundary of $CH(P)$. Let P'' be the point set obtained after deleting the triangle $\Delta(p)$. Suppose P'' has an independent set of size n, otherwise we can apply induction. Then the edge qp_i of $CH(P'')$ must contain the point p_k, and the edge p_ir must contain p_j. The independent set of size n in P'' must contain all points from the independent set of size n in P', except p. This implies p must be visible to p_i, otherwise we get an independent set of size $n+1$ in P. Now let P''' be the point set obtained after deleting the triangle $\Delta(r)$. Since $CH(P''')$ has 2 vertices p and p_i that are visible to each other, it cannot contain an independent set of size n. Again, we can apply induction.

Suppose p and q are vertices of $CH(P)$ but r is not. If r does not lie on the boundary of $CH(P)$, then P'' has two convex hull vertices p_i and p_j that are visible to each other, and hence cannot contain an independent set of size n. On the other hand, if r is on the boundary of $CH(P)$, and if P'' contains an independent set of size n, then since p and p_i are not visible to each other, P contains an independent set of size $n+1$, a contradiction.

Finally, suppose p is a vertex of $CH(P)$ but neither q nor r is a vertex of $CH(P)$. If $\Delta(p)$ does not contain p_i, then $CH(P'')$ has two vertices p_i and p_j that are visible to each other. A symmetrical argument can be used if p_j does not precede p on the boundary of $CH(P)$. The only other possibility is that the boundary of $CH(P)$ contains only the points $p.p_i, p_j$. Now if p_k does not lie in the segment p_iq, we delete the triangle p_iqp_k. Again, the remaining point set has two vertices p, p_j, visible to each other. If p_k lies in p_iq, we delete the triangle p_jqp_k. Thus in all cases, the remaining point set cannot have an independent set of size n, and we can apply induction.

The arguments in the case when $CH(P')$ has 4 vertices are very similar to the case when $CH(P')$ has 3 vertices. We omit them here. □

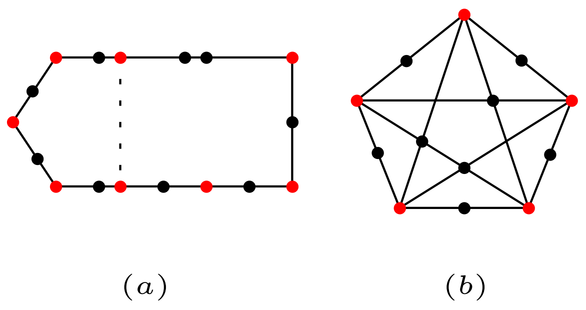


Figure 1: (a) A point set P where there are five points of I with no three collinear. The points in the interior of $CH(P)$ are not shown. The points of I are coloured red. (b) A pentagon with five vertices from I.

2.2 Generalized cycle partitions

Let S be a given set of cycles $\{C_1, C_2, \ldots, C_l\}$, not all C_i of length 3. Let L_i be the length of $C_i \in S$. In this section, we show that it is possible to partition a point set P into disjoint cycles of S if and only if P does not have $\sum_{i=1}^{l} L_i - l + 1$ collinear points.

Lemma 3 *If P has $\sum_{i=1}^{l} L_i$ points, not all collinear, then it is possible to separate out C_i from P so that $CH(C_i)$ and $CH(P \setminus C_i)$ are disjoint.*

Proof. Let p be any vertex of $CH(P)$, p_0 the point of P that precedes p on the boundary of $CH(P)$, and q the point that follows p. Then p_0 and q are vertices of $CH(P \setminus \{p\})$ and let $p_0, p_1, p_2, \ldots, p_k = q$ be the points of P that occur between p_0 and q on the boundary of $CH(P \setminus \{p\}$. If $k \geq L_i - 2$, we choose C_i to be the cycle $p, p_0, p_1, p_2, \ldots, p_{L_i-2}$ (Figure 2 (a)). If $k < L_i - 2$, then $L_i > 3$. We delete the points $p, p_1, \ldots, p_{k-1}$, and in the remaining set

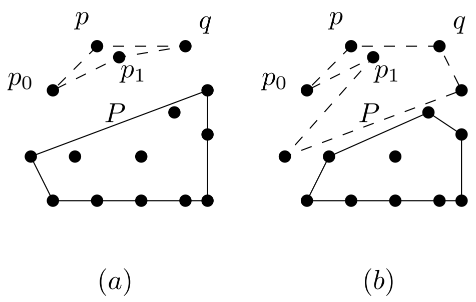


Figure 2: (a) A C_4 separated out from P. (b) A C_6 is separated out from P

of P' of points, find a cycle C'_i of length $L_i - k$, using the same procedure, starting with the vertex q. Then $C'_i \cup \{p, p_1, \ldots, p_{k-1}$ give the required cycle C_i (Figure 2 (b)). □

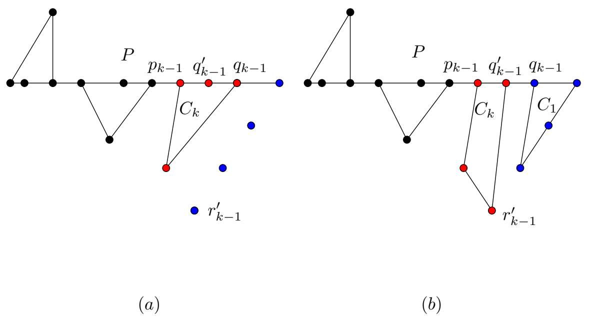


Figure 3: (a) There is only one remaining point of P on the line. (b) One point of C_k is freed to be used for C_1.

Theorem 4 *Let $C_1, C_2, \ldots, C_k$ be a collection of cycles of lengths $L_1, L_2, \ldots, L_k$ such that $L_k \geq 4$. A set P of $L = \sum_{i=1}^{k} L_i$ points admits a disjoint cycle partition into cycles of lengths $L_1, L_2, \ldots, L_k$ iff it does not contain $L - k + 1$ collinear points.*

Proof. If P contains $L - k + 1$ collinear points on some line, then there are at most $k - 1$ points of P not in the line. Thus in any partition of P into k parts, some part must contain all points in the line. Thus P cannot have a cycle partition into k cycles of lengths $L_1, \ldots, L_k$.

We prove the converse by induction on k. If $k = 1$, the result is trivial. Suppose $k \geq 2$. By Lemma 3, we can find a cycle C_1 of length L_1 in P, such that C_1 and $CH(P \setminus C_1)$ are disjoint. If $P \setminus C_1$ does not contain $L - L_1 - (k - 1) + 1$ collinear points, then by induction, it has a disjoint cycle partition into $k - 1$ cycles of lengths $L_2, \ldots, L_k$. This gives the required disjoint cycle partition of P.

Suppose $P \setminus C_1$, and hence P itself contains $L - L_1 - k + 2$ collinear points. Note that by assumption P contains at most $L - k$ collinear points. Order these collinear points in left to right order along the line containing them, and group them into k groups in left to right order, with the i^{th} group containing $L_{i+1} - 1$ points, for $1 \leq i < k$. All the remaining points are placed in the k^{th} group. Note that there is at least one, and at most $L_1 - 1$ points in the k^{th} group. There are at least k points not in the line. Let p_i denote the leftmost point of the i^{th} group and q_i the rightmost.

We now form a disjoint cycle partition of P directly, using the same method as in Theorem 2. For $i = 1$ to $k - 1$, let r_i be a point not in the line, that has not be included in any earlier cycle, such that the angle $p_i q_i r_i$ is minimum, and subject to this condition r_i is nearest to q_i. Let C_{i+1} be the cycle formed by r_i and all points in the line that lie between p_i and q_i (inclusive). Thus length of C_{i+1} is L_{i+1}.

Now there are L_1 points remaining, at least one of which is on the line, and at least one is not in the line. If there are two or more points remaining on the line, we can connect the remaining points by line-segments to form a cycle of length L_1 that is disjoint from the earlier cycles.

Suppose there is only one point on the line (Figure 3 (a)). Now consider the cycle C_k formed earlier. By assumption, it has length at least 4, so there is a point q'_{k-1} immediately to the left of q_{k-1} on the line, such that $q'_{k-1} \neq p_{k-1}$. Let r'_{k-1} be a remaining point such that the angle $p_{k-1} q'_{k-1} r'_{k-1}$ is minimum, and subject to this r'_{k-1} is nearest to q'_{k-1}. Modify C_k by replacing q_{k-1} by r'_{k-1}. Now there are two points remaining on the line and we can complete the cycle C_1 using the remaining points (Figure 3 (b)). □

2.3 2-factors of point visibility graphs

Here we study the relationship between cycle partition of point sets and 2-factors of their visibility graphs.

Lemma 5 *Given a set of cycles $S = \{C_1, C_2, \ldots, C_l\}$, not all C_i of length 3, a point visibility graph G admits the 2-factor with cycles of length specified by S if and only if G does not have any induced path on $(\sum_{i=1}^{l} L_i) - l + 1$ vertices, where L_i is the length of the cycle C_i.*

Proof. If such a path exists, then one of the cyles must have all its vertices from the induced path, a contradiction. If such a path does not exist, then no point set P corresponding to G can have $(\sum_{i=1}^{l} L_i) - l + 1$ collinear points. Thus, P must have a disjoint cycle partition corresponding to S, which corresponds to a 2-factor of G. $\square$

Corollary 6 *If a point set P has l points on a line and at most k points outside of it, where $l \geq 2k+2$, then every point set with the same visibility graph as P, has at least l collinear points.*

3 Clique partition

A characterization analogous to that in 2 does not work for clique partitions where $k \geq 4$. Consider the graph drawn on a point set in Figure 4. Due to its structure, it is called a *slanted grid graph* [12]. A slanted grid graph on x points has a maximum independent size only of $O(\sqrt{x})$. The slanted grid graph in the figure has forty-four points with $k = 4$ and $n = 11$, and a maximum independent set of size five. However, it cannot be partitioned into copies of K_4, because p_1 and p_2 are not contained in any K_4. adjacent to triangles. In this section, we

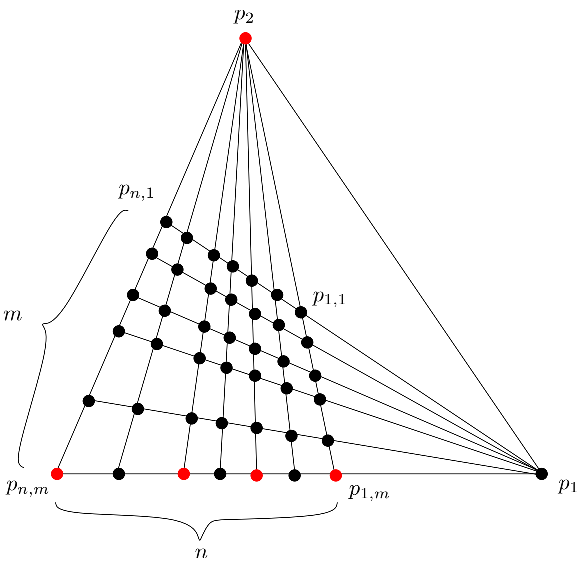


Figure 4: A slanted grid graph on forty-four points. A maximum independent set is coloured in red.

show that the problem of partitioning a given point set in the plane, into k-cliques for $k \geq 5$, is NP-hard. To show this, we reduce *3-occurrence SAT* to the problem. A 3-occurrence SAT formula is a SAT formula where each variable occurs at most 3 times. The 3-occurrence SAT problem is known to be NP-hard [13].

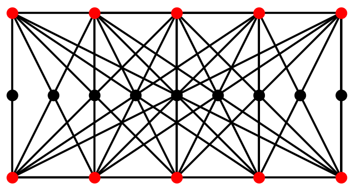


Figure 5: A partial grid graph.

3.1 Construction of the reduction

We provide a reduction of 3-occurrence SAT to partitioning a point set into copies of K_5. We start with any given 3-occurrence SAT formula θ, with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$. Wlog we assume that there is no variable in θ whose positive or negative literals solely constitute all of its occurrences. We also assume that each clause C_i, $2 \leq i \leq m-1$, has all variables different from those in C_{i-1} and C_{i+1}. This assumption is valid because any given 3-occurrence SAT formula can be transformed to such a formula by adding a linear number of variables and clauses, with every variable and its negation occuring at least once each. Let n_1 and n_2 be the number of variables that occur twice and thrice in θ, respectively, so that $n_1 + n_2 = n$.

We now construct a point set P from θ so that a partition of P into 5-cliques is possible if and only if θ is satisfiable. We do the following:

(a) Let $v = 3n_1 + 4n_2 + n - 1$, $b_n = 2(m-1)n_1 + (m-2)n_1 + 2(m-1)n_2 + 2(m-2)n_2 + m(n-1) = 4mn - 5n + n_2m - 2n_2$. Let $e = b_n - 2v + 3m - 1$, $b = e + m - 1$ and $c = 2(v - 2m) + 2e$.

(b) Call the x-axis the *clause line*. Starting from the origin, from left to right, place m points on the clause-line, unit distance apart. Each such point is called a *clause-point*, the k^{th} point representing C_k, and denoted as cp_k.

(c) Consider the horizontal line with y-coordinate -2 and call it the *extra-line*. Starting from the y-axis from left to right, we place e points on the extra-line, each a unit distance apart.

(d) Consider the horizontal line with y-coordinate -1 and call it the *blocking-line*. Starting from the y-axis from left to right, we place b points on the blocking-line, each half a unit distance apart.

(e) Now consider the horizontal line with y-coordinate -1.5 and call it the *variable-line*. Let

x' be the x-coordinate of the point of intersection of the variable-line and the line passing through the leftmost clause-point and the rightmost point on the blocking-line. Starting from the point $(x'+1, -1.5)$ from left to right we place points on the variable-line, representing the variables of θ as follows.

(e.i) If x_i and $\bar{x}_i$ occur in C_j and C_k respectively, then we place three points, x_i^1, $x_i^2 = \bar{x}_i{}^1$ and $\bar{x}_i{}^2$ to the right of all points placed so far on the variable-line, in the same consecutive order. We block the points x_i^1 and $\bar{x}_i{}^2$ from all points on the clause-line other than cp_j and cp_k respectively. We block the point x_i^2 from all points on the clause-line other than cp_j and cp_k.

(e.ii) Wlog if there are two occurrences of x_i and one occurrrence of $\bar{x}_i$, then we place four points, x_i^1, $x_i^2 = \bar{x}_i{}^1$, $x_i^3 = \bar{x}_i{}^2$ and x_i^4 to the right of all points placeed so far on the variable-line, in the same consecutive order. Suppose that x_i occurs in C_j and C_k, and $\bar{x}_i$ occurs in C_l. We block the points x_i^1 and x_i^4 from all points on the clause-line other than cp_j and cp_k respectively. We block the point x_i^2 from all points on the clause-line other than cp_j and cp_l. We block the point x_i^3 from all points on the clause-line other than cp_k and cp_l.

(f) On the variable-line, introduce a new *variable-blocker point* after all the points corresponding to a particular variable have been placed, except for the rightmost set of points corresponding to a variable (i.e. points corresponding to x_n). Thus, there are $n-1$ variable-blocker points in total. Thus, now there are a total of v points on the variable-line. Introduce blockers on the blocking-line so that no clause-point sees any of the variable-blocker points.

(g) Perturb the points on the variable-line slightly so that for each such point, the corresponding blocking vertex blocking it from a point on the clause-line, blocks only a single pair of vertices. Thus there are b_n blockers in total used for the last two steps.

(h) Add c more points to the clause-line, all to the right of the clause-points, such that they see all points not on the clause-line.

3.2 Properties of the constructed point set

We have the following lemmas based on the construction.

Lemma 7 *The construction of the point set can be completed in polynomial time.*

Proof. The points placed on lattice points have integer coordinates. The length of these coordinates are only $O(log\ mn)$. The blockers are placed between the intersection of the blocking-line and the line passing through two such points. So, the coordinates of the blockers are also of length $O(log\ n)$. After a blocker is placed on the blocking-line, it can coincide with some already placed blocker or block one or more pairs of points which are required to be visible. There are $O(mn)$ blockers in total. So there are only $O(mn)$ such undesirable positions. We first divide the variable-line into $4n$ intervals. Each of these intervals we further divide into mn intervals. Clearly, the coordinates of the endpoints of the intervals are of length $O(log\ mn)$. Each of the perturbations can be achieved by assigning these coordinates to the variable-points. Hence the whole construction can be achived in $O(mn\ log\ mn)$ time. □

Now we study a related structure related to our construction. Consider a partial grid P_g on the lines $y = 1$, $y = 0$ and $y = -1$. We call these three lines the *top*, *middle* and *bottom* horizontal lines of P_g. The partial grid starts from the y-axis and lies on the right side of it. The points of the top and bottom horizontal lines of P_g are only allowed to have nonnegative integer coordinates. For every point with coordinates (x, y) in the top and bottom horizontal lines of P_g, where $x \neq 0$, there must also be a point with coordinates $(x-1, y)$ in P_g. The points on the middle line of P_g are only allowed to have coordinates of the form $(x, \frac{y}{2})$, where x and y are nonnegative integers. For every point on the middle line of P_g with coordinates $(\frac{x}{2}, 0)$, where $x \neq 0$, there must also be a point with coordinates $(\frac{x-1}{2}, 0)$. Suppose P_g has p points on $y = 1$ and q points on $y = -1$. Then we have the following lemma.

Lemma 8 *A total of $p+q-1$ points of P_g on $y = 0$ are necessary and sufficient to block all points of P_g on $y = 1$ from all points of P_g on $y = -1$.*

Proof. Consider points on the top and bottom lines on P_g having coordinates $(x_1, 1)$ and $(x_2, -1)$ respectively. The point of intersection of the middle line and the line segment joining these two points is $(\frac{x_1+x_2}{2}, 0)$. Since $x_1 + x_2 \leq p + q$, this point is already in P_g (Figure 5). Now let $(x_3, 0)$ be the coordinates of a point of P_g on the middle line and wlog let $p \geq q$. This point blocks the points of P_g with coordinates $(2x_3, 1)$ and $(0, -1)$. □

Lemma 9 *In our construction for the reduction, no clause-point can see any extra-point.*

Proof. The clause-points, extra-points, and the blockers on lattice points of the blocking-line induce a partial grid. By Lemma 8, no clause-point can see any extra-point. □

Lemma 10 *P_c can be K_5-partitioned if and only if θ has a satisfying assignment.*

Proof. Suppose ξ is a K_5 partition of P_c. By Lemma 9, no clause-point can see any extra-point. So, a clause-point can see only its adjacent clause-points, all the blocking-points, and two variable-points for each of the variables that occur in its clause. Suppose that a clause-point is a part of a K_5. In the formula θ, no consecutive clauses have the same variables. So, its two adjacent clause-points see neither each other, nor any of the variable points corresponding to the clause-point. Hence, the K_5 can contain only (a) the clause-point, (b) only two of the variable-points corresponding to the same literal, since all the variable points are collinear and the points corresponding to each variable are separated by a variable-blocker point, and (c) only two blocking-points, since all the blocking-points are collinear.

Given such a K_5-partition of P_c, the corresponding satisfying assignment of θ can be constructed as follows. If some clause-point for C_i takes the variable-points for x_j in its K_5, then assign 1 to x_j. If $\bar{x_j}$ does not occur in θ then we are done for C_i. Otherwise, since θ is an instance of 3-occurrence SAT, $\bar{x_j}$ can occur in at most two clauses. But by construction of P_c, a variable-point from each of the pairs of variable-points representing the occurrences of $\bar{x_j}$, will coincide with the one of the variable-points representing x_j that the clause-point of C_i took. So, no clause-point can include variable-points representing $\bar{x_i}$ in their K_5. An analogus reasoning holds if the clause-point for some C_i takes the variable-point for $\bar{x_i}$. Hence there is no conflict in assigning truth values to variables using the method described above.

Now we prove the other direction of the lemma. Consider a satisfying assignment of θ. In P_c, start with the clause-point of C_1. The corresponding K_5 will contain the clause-point, the two leftmost points on the blocking-line, and the variable-points corresponding to a literal that is assigned 1 in $C_1 \in \theta$. Similarly, for C_i, use the $(2i-1)^{th}$ and $2i^{th}$ leftmost points on the blocking-line, and the variable-points corresponding to a literal that is assigned 1 in $C_i \in \theta$. Now each clause-point is in its respective K_5.

Now there are $v-2m$ and $b+b_n-2m$ remaining points on the variable-line and blocking-line respectively. Form a K_5 for each of the remaining points on the variable-line with two consecutive points on the blocking-line and clause-line, always choosing the leftmost points available. After this is done, $b+b_n-2m-2(v-2m)$ and $c-2m-2(v-2m)$ points remain available on the blocking-line and clause-line respectively. Form a K_5 each for the points on the extra-line, with two consecutive available points each from the blocking-line and clause-line. Due to the values of c, e, b, and b_n, all the points of P_c are exhausted. □

Theorem 11 *For all $k \geq 5$, the K_k-partition problem for point sets on the plane is NP-hard.*

Proof. For $k = 5$ we have Lemma 10. Suppose that K is a greater odd number $5 + 2x$, then for a given instance of 3-occurrence SAT, first produce P_c for K_5-partition. Let P_c have $5y$ points. Parallel to the clause-line and above it, draw x lines of $2y$ points each, such that each new point is visible from every other new point and all points of P_c.

Now consider the other case where K is any greater even number $5+2x-1$. First we discuss the case where $k = 6$. We assign new values to b, c and e. Let $b = b_n$, $c = 2b_n - v$ and $e = 2b_n - m$. Modify the construction for $k = 5$ by intially placing b points on the blocking-line, each a unit distance apart. Observe that, due to the above placement, a clause-point is visible from an extra-point if and only if the parity of their x-coordinates is different. So, since no clause-point sees to consecutive points on the extra-line, a clause-point can be placed into the same K_6 with only one extra-point. Also, this makes $b = \lfloor \frac{m+e}{2} \rfloor$. We ensure that the parity of e and m are the same, so the above relation becomes $b = \frac{m+e}{2}$. As before, whenever possible, we choose the leftmost free points. Also, as before, this K_6 can contain only two blocking-points and variable-points each. After all the clause-points are exhausted thus, the number of available points on the variable-line, blocking-line and extra-line, are $v-2m$, $b+b_n-2m$ and $e-m$ respectively. After this, we place the remaining variable-points into copies of K_6, using two blocking-points, two extra-points and one new point from the clause-line. So, now the number of available points on the clause-line, blocking-line and extra-line, are $c-(v-2m)$, $b+b_n-2m-2(v-2m)$ and $e-m-2(v-2m)$ respectively. We take two points each from these three lines and form copies of K_6. This is possible due to the new values of b, c and e.

If k is any greater even number $5+2x-1 = 6+2x-2$, add $x-1$ new lines as in the case before. □

4 Concluding Remarks

We have solved the problem of partitioning point sets into a set of polygons whose sizes are given, and proved analogous results for their visibility graphs. For clique partitions, when the sizes of given cliques are at least 5, we have shown the problem to be NP-hard. Our triangle-partition method indeed gives the solution a clique partition into triangles, but the result for $k \geq 4$ remains unknown. The related problem of partitioning a point set into *convex* polygons also remains unsolved.

References

[1] O. Aichholzer, C. Huemer, S. Kappes, B. Speckmann, and C. D. Tóth. Decompositions, partitions, and coverings with convex polygons and pseudo-triangles. *Graphs and Combinatorics*, 23(5):481–507, 2007.

[2] J. Cardinal and U. Hoffmann. Recognition and complexity of point visibility graphs. *Symposium of Computational Geometry*, pages 171–185, 2015.

[3] B. Chazelle, L. J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[4] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[6] S. K. Ghosh and P. P. Goswami. Unsolved problems in visibility graphs of points, segments and polygons. *ACM Computing Surveys*, 46(2):22:1–22:29, December, 2013.

[7] S. K. Ghosh and B. Roy. Some results on point visibility graphs. In *Proceedings of the Eighth International Workshop on Algorithms and Computation*, volume 8344 of *Lecture Notes in Computer Science*, pages 163–175. Springer-Verlag, 2014.

[8] K. Hosono. On convex decompositions of a planar point set. *Discrete Mathematics*, 309(6):1714–1717, 2009.

[9] J. Kára, A. Pór, and D. R. Wood. On the Chromatic Number of the Visibility Graph of a Set of Points in the Plane. *Discrete & Computational Geometry*, 34(3):497–506, 2005.

[10] M. S. Payne, A. Pór, P. Valtr, and D. R. Wood. On the connectivity of visibility graphs. *Discrete & Computational Geometry*, 48(3):669–681, 2012.

[11] F. Pfender. Visibility graphs of point sets in the plane. *Discrete & Computational Geometry*, 39(1):455–459, 2008.

[12] B. Roy. Point visibility graph recognition is NP-hard. *International Journal of Computational Geometry and Applications*, 26(1):1–32, 2016.

[13] C. A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 2:85–89, 1984.

[14] M. Urabe. On a partition into convex polygons. *Discrete Applied Mathematics*, 64(2):179–191, 1996.

Counting Convex k-gons in an Arrangement of Line Segments

Martin Fink* Neeraj Kumar* Subhash Suri*

Abstract

Let $\mathcal{A}(S)$ be the arrangement formed by a set of n line segments S in the plane. A subset of arrangement vertices $p_1, p_2, \ldots, p_k$ is called a *convex k-gon* of $\mathcal{A}(S)$ if $(p_1, p_2, \ldots, p_k)$ forms a convex polygon and each of its sides, namely, (p_i, p_{i+1}) is part of an input segment. We want to count the number of distinct convex k-gons in the arrangement $\mathcal{A}(S)$, of which there can be $\Theta(n^k)$ in the worst-case. We present an $O(n \log n + mn)$ time algorithm, for any fixed constant k, where m is the number of pairwise segment intersections. We can also *report* all the convex k-gons in time $O(n \log n + mn + |K|)$, where K is the output set. We also prove that the k-gon counting problem is 3SUM-hard for $k = 3$ and $k = 4$.

1 Introduction

We consider the problem of counting, and enumerating, all convex k-gons formed by the arrangement $\mathcal{A}(S)$ of a set S of n line segments in the plane. A set of vertices $p_1, p_2, \ldots, p_k$ of the arrangement $\mathcal{A}(S)$ is called a *convex k-gon* if $(p_1, p_2, \ldots, p_k)$ forms a convex polygon and each of its sides $(p_1, p_2), (p_2, p_3), \ldots, (p_{k-1}, p_k), (p_k, p_1)$ is part of an input segment. We note that such a k-gon is not necessarily a *face* of the arrangement, and in general there can be $\Theta(n^k)$ convex k-gons, for any fixed k. We are interested in the problem of *counting* these k-gons. That is, given a set of n line segments in the plane, how many convex k-gons exist in their arrangement?

Surprisingly, this natural-sounding problem appears not to have been explored in computational geometry. We are motivated by an application in computer vision where the case of counting, and enumerating, *convex quadrilaterals* arises. Specifically, the arrangement is the camera image representing linear boundaries of objects in the scene, and the goal is to estimate (the counting problem) the number of "rectangular" objects in the input scene, which may represent important features such as desks, door frames, walls etc. Due to the perspective transformation, the rectangles in the scene map to convex quadrilaterals in the image.

The computational problem then becomes the following: can we find all convex quadrilaterals in the n-segment arrangement in better than the naive $O(n^4)$ time? Counting convex k-gons is a natural generalization of this problem. We call this the *k-gon reporting* problem, which leads to a natural *counting* version of the problem, where we are just interested in counting the number of k-gons formed by the segments. Unlike the segment intersection problem [2, 3], in which the maximum number of intersecting line segments is $O(n^2)$, the number of combinatorially distinct k-gons in an n-segment arrangement can be $\Omega(n^k)$. See Fig. 1. Therefore, it is desirable to be able to count the k-gons in time much faster than the number of combinatorially distinct k-gons.

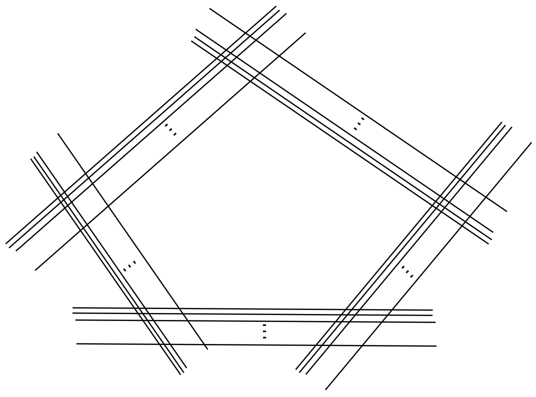


Figure 1: An arrangement of n segments with $\Omega(n^k)$ convex k-gons. Each of the $k = 5$ groups contains $\lfloor \frac{n}{k} \rfloor$ segments.

Our Contribution. We present a sweep-line algorithm for counting the number of k-gons in worst-case time $O(n \log n + mn)$, using $O(n^2)$ space, for any constant k, where m is the number of pairwise segment intersections. The algorithm works for non-constant values of k as well, but in that case takes $O(n \log n + mn^2)$ time and $O(n^3)$ space. In either case, the running time is independent of k.

By maintaining additional information during the counting algorithm, we can also recover all the k-gons, in worst-case time $O(n \log n + mn + |K|)$ time, using $O(mn + n^2)$ space, where K is the output.

Finally, we show that counting the number of triangles and the number of quadrilaterals are both 3SUM-hard, suggesting that a running time significantly better than $O(n^2)$ is unlikely.

*Department of Computer Science, University of California, Santa Barbara, {fink|neeraj|suri}@cs.ucsb.edu

Related Work. The problem of counting convex k-gons in a set of n points has been considered by several researchers [9, 8, 7]. The algorithm in [9] achieves a running time of $O(n^{k-2})$, which was then improved to $O(n^{\lceil k/2 \rceil})$ in [8]. This bound was improved significantly to $O(n^3)$ by Mitchell et al. [7] using dynamic programming. In [4], Eppstein et al. study the related problem of finding minimum area k-gons for point sets.

Some results are also known for the restricted problem of counting faces in line arrangements. For instance, every arrangement of lines (or pseudolines) in the plane results in $\Omega(n)$ triangular faces [5].

Another related problem is one of counting and reporting simple cycles of given length in a graph. In general, the time bound for counting the cycles is exponential in k [1], however for $k \leq 7$ the problem can be solved in $O(n^{2.376})$ time. These cycle counting results in graphs, however, do not solve our k-gon problem because of the *convexity* constraint. Indeed, an arrangement $\mathcal{A}(S)$ of segments can be easily viewed as a graph, whose vertices are the segments and whose edges correspond to pairs of intersecting segments. However, cycles in this graph are not necessarily convex polygons. An exception is the case of triangles which are always convex: every 3-cycle correspond to a triangle in the segment arrangement, but only for non-degenerate input, namely, no three segments intersecting in a common point.

2 Counting Convex k-gons

Let P be a convex k-gon in the arrangement $\mathcal{A}(S)$ formed by the n line segments of S.[1] Let L be a vertical line intersecting P. Since P is convex, L can only intersect two sides of P. The *span* of P with respect to L is the (ordered) pair of segments of P that intersect L. Although the number of k-gons can be exponential in k, the number of distinct spans is only quadratic.

Observation 1 *There are $O(n^2)$ distinct spans among all k-gons of $\mathcal{A}(S)$ with respect to a vertical line L.*

In other words, Observation 1 tells us that although there could be $\Omega(n^k)$ k-gons, at a given vertical line L, all k-gons intersecting L can be assigned to one of the $O(n^2)$ distinct segment pairs. This suggests the existence of a natural sweep line based approach for the counting problem. The key idea is to keep track of convex *open* polygons with up to k sides as we sweep a vertical line L across the arrangement. When sweeping over an intersection, some open polygons may become closed k-gons, which we must count, while other open polygons can be extended using the intersection vertex, and new open polygons start growing at the intersection. We start by fixing some notation.

[1]For the rest of the paper, we drop the qualifier "convex" and simply refer to P as a k-gon.

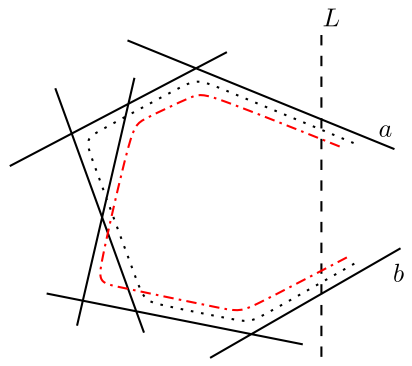


Figure 2: Open 5-gons: $\Sigma(a, b, 5)$ at a given sweep line L; two members of the set are shown by dotted lines.

Notation. Observe that when we are sweeping a vertical line L across the arrangement, at a given x-coordinate x_L, we may have come across two types of potential k-gons:

- *Closed k-gons:* these are the k-gons all of whose sides are to the left of the line L.
- *Open j-gons:* these are j-gons, for $j \leq k$, whose j sides lie (partially or fully) to the left of L. More precisely, L intersects the two open sides of these convex polygons and their remaining $j-2$ sides are to the left of L. See Fig. 2.

Observe that open j-gons are only potential candidates for closed k-gons; not all of them necessarily become k-gons.

At an x-coordinate x_L, we can now represent an open j-gon P by the triplet (a, b, j), where a and b are the segments forming the top and bottom sides of P at x_L and j is the number of sides we have seen so far. Note that $2 \leq j \leq k$ and this includes the sides of P formed by the segments a and b. The triplet (a, b, j) succinctly combines all open j-gons for which a and b are the open sides intersecting the line L; we let $\Sigma(a, b, j)$ denote the set of these open j-gons, and we let $\sigma(a, b, j) = |\Sigma(a, b, j)|$.

2.1 Algorithm

Our algorithm moves a sweep line L across the arrangement $\mathcal{A}(S)$ with the segment intersections as key event points. For the sake of simplicity, we assume no degeneracies for now, that is, every intersection involves exactly two line segments of S. (We will show how to handle degenerate cases later in this section.) We maintain an array of counters $\sigma(a, b, j)$ which keep track of all open j-gons whose span is (a, b) on line L, for all $2 \leq j \leq k$. A global counter keeps track of all the closed k-gons that have been encountered already. In the following, we explain these steps in detail.

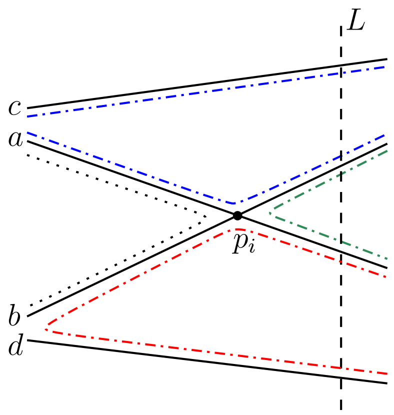


Figure 3: After the intersection of a and b, the sweep line L gets new open polygons for $\Sigma(b,a,2)$, $\Sigma(c,b,j+1)$, and $\Sigma(a,d,j+1)$ shown in dash dotted; parent polygons in $\Sigma(c,a,j)$ and $\Sigma(b,d,j)$ are still active.

1. Set *count* $= 0$ and $\sigma(a,b,j) = 0$ for all segments a, b and $2 \le j \le k$.

2. Compute all m intersections of the n line segments using [2] and order them from left to right.

3. Process the intersections one by one from left to right, moving the sweep line L accordingly.

   When L contains the intersection of segments a and b (where to the left of L, a is above b), we perform the following updates. See Fig. 3.

   (a) Open k-gons of $\Sigma(a,b,k)$ become closed k-gons. We update the count:
   $$count \mathrel{+}= \sigma(a,b,k)$$

   (b) An open 2-gon of b and a begins at this intersection. We initialize the count:
   $$\sigma(b,a,2) = 1$$

   (c) For all $2 \le j < k$, each open j-gon with a as the lower side can be extended to an open $j+1$-gon with b as the lower side. Let S'_L be the set of segments intersecting sweep line L *above* the intersection point (a,b). We update the count:
   $$\forall c \in S'_L \quad \sigma(c,b,j+1) \mathrel{+}= \sigma(c,a,j)$$

   Similarly, for each $2 \le j < k$, each open j-gon with b as the upper side can be extended to an open $j+1$-gon with a as the upper side. Let S''_L be the set of segments that intersect sweep line L *below* the intersection point (a,b).
   $$\forall d \in S''_L \quad \sigma(a,d,j+1) \mathrel{+}= \sigma(b,d,j)$$

4. Return *count*.

Correctness. Let S_L be the set of all segments that intersect the sweep line L. Let A_L be the set of all possible spans for the k-gons intersecting L. That is, $A_L = \{(a,b) \mid a,b \in S_L,\ a \text{ above } b \text{ on } L\}$. As the sweep line moves from left to right we maintain the following invariant:

> *count* is the total number of closed k-gons to the left of L; and $\sigma(a,b,j)$ is the number of open j-gons with span (a,b) on L, for all $(a,b) \in A_L$ and $2 \le j \le k$.

The invariant is trivially satisfied before processing the first intersection. For the general case, after moving the sweep line L over the intersection (a,b), the segments a and b switch their vertical order. Therefore,

1. A_L no longer includes the span (a,b).

2. *count* now includes the new k-gons that complete at the intersection (Step 3a); these correspond to the open k-gons counted by $\sigma(a,b,k)$.

3. A_L includes the new span (b,a). Right after the crossing, the open 2-gon formed by b and a is the only polygon with that span, covered by $\sigma(b,a,2) = 1$.

4. Open polygons with span $(c,b) \in A_L$ can now also use the vertex (a,b). Any such open j-gon ($j \le k$) must consist of the new intersection and an open $j-1$-gon with span (c,a) right before the intersection (compare Step 3c).

5. Analogously, open polygons with span $(a,d) \in A_L$ can now use vertex (a,b). Such a new open j-gon ($j \le k$) consists of the new intersection and an open $j-1$-gon with span (b,d).

It is easy to see that the algorithm maintains the invariant after processing each intersection. Hence, when eventually the sweep line L is right of all intersections, *count* is the total number of k-gons.

Analysis. Computing and storing all m intersections takes $O((n+m)\log n)$ time and $O(m+n)$ space [2]. Since we perform $O(n)$ updates for each of the m events, the total running time for our algorithm is $O(n \log n + mn)$. The total space requirement is $O(n^2)$ since we store information for all pairs of segments that may intersect the sweep line.

Handling Degenerate Cases. We now show how to extend our algorithm so that it can also handle segment arrangement with degeneracies, that is, with three or more segments intersecting in a single point. (For parallel segments, we do not need to do anything special). For an intersection point p_i of a set S_i of more than two segments, we first update the number of closed k-gons for every pair of segments in S_i. For extending open j-gons ($2 \leq j < k$), we need to be a bit more careful. Since we do not want degenerate k-gons, we should only extend the j-gons which we have seen before the current intersection. More precisely, we compute the updates for every pair of segments in S_i, and apply them collectively. One way to achieve this is to process the updates in Step 3(b) and 3(c) in decreasing order of j as follows:

- For j in $k-1$ down to 3 perform updates in Step 3(c) for every segment pair in S_i.
- Perform updates in Step 3(b) for every segment pair in S_i.

Observe that these modifications do not affect the overall runtime since $m \in O(n^2)$ is the number of pairwise intersections.

3 Reporting Convex k-gons

We now turn to the problem of reporting all the k-gons in an arrangement of n line segments. We solve this problem by extending our algorithm for counting k-gons. The key idea is to keep track of how the values $\sigma(a, b, j)$ are updated as we move the sweep line across the arrangement, and to remember the values that contributed to the total number. Recall that the total number of k-gons formed by n segments can be $\Omega(n^k)$. Therefore, we would like the total running time to be linear in size of the output. We start by describing a *reporting graph* that will help us reconstruct all k-gons.

Reporting Graph. Our reporting graph is a *labeled* directed acyclic graph $G = (V, E, \mathcal{L})$. Its vertices represent the sets of polygons $\Sigma(a, b, j)$ and its edges keep track of how these sets grow. The function $\mathcal{L} \colon E \to \mathbb{N}$ assigns a label $\mathcal{L}(e)$ to each edge $e \in E$. The label is a timestamp and represents the intersection at which the edge was created.

To construct the digraph G, we extend the counting algorithm from Section 2.1 as follows:

1. For every pair (a, b) of segments and $2 \leq j \leq k$ add a vertex (a, b, j) to G.
2. Define $Q = \emptyset$ to be the set that keeps track of closed k-gons grouped by their rightmost vertex.
3. Refer to step 3 of the counting algorithm. Suppose the sweep line L is currently at the i^{th} intersection event (a, b). Recall that S'_L and S''_L are respectively the sets of segments that intersect L above and below the intersection point (a, b). We modify the reporting graph as follows; see Fig. 4 for an example.
   (a) If $\sigma(a, b, k) > 0$, insert (a, b, k) to Q.
   (b) For all values $2 \leq j < k$ and each segment $c \in S'_L$ with $\sigma(c, a, j) > 0$, create an edge $\big((c, b, j+1), (c, a, j)\big)$.
   (c) Similarly, for all $2 \leq j < k$ and a segment $d \in S''_L$ with $\sigma(b, d, j) > 0$, create an edge $\big((a, d, j+1), (b, d, j)\big)$.

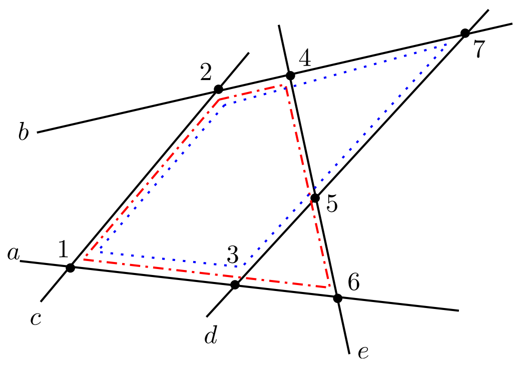
   Label each edge e created for this intersection event with the timestamp $\mathcal{L}(e) = i$.

The reporting graph G has the following properties.

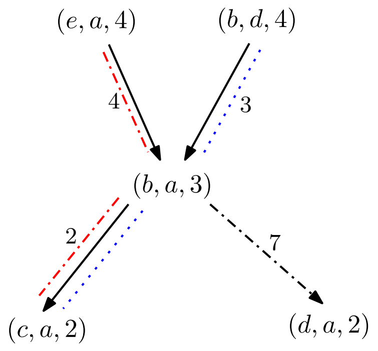
- G has $O(n^2)$ vertices and $O(mn)$ edges. The vertices of the form (a, b, k) have in-degree zero (sources) and vertices of the form $(a, b, 2)$ have out-degree zero (sinks).
- An edge of G represent the extension of an open j-gon into a $j+1$-gon ($2 \leq j < k$), with the intersection point (a, b) being the newly added corner. As a result, we get two new sets of $j+1$-gons: one with b as the lower side and another with a as the upper side.
- There is exactly one label on every edge since two segments can only intersect once.
- Each complete k-gon corresponds to a path that starts at a vertex in Q (a source in G) and ends at a sink of G. Since G is acyclic, every such path has exactly $k-2$ edges. The intersection points corresponding to these $k-2$ edges along with the two intersection points for the source and sink will be the k vertices of the output k-gon.

Enumerating all k-gons. With the reporting graph G, enumerating all k-gons seems pretty straightforward. We can simply start at vertices in Q one by one and recursively explore all distinct paths to sinks. However, there is one small caveat. Since the segments may continue to grow further after we close a k-gon, it is possible that a vertex v of G gets an additional successor w' after it got a predecessor u; see Fig. 4. Observe that in such a case, the path $(u \to v \to w' \to \cdots)$ does not correspond to a valid k-gon since the corresponding vertices are not ordered chronologically.

In order to fix this we can use the timestamps of the edges: we only recurse using the edges whose timestamp is smaller than that of the parent edge. Because of our construction, we are guaranteed to find at least one such edge. Moreover, since the edges are added in the order of their timestamps, we can stop at the first edge for which the timestamp is higher than the parent value. This way we spend no extra time on objects that are not a member of our output set. Consequently, we get a running time of $O(n \log n + mn)$ for constructing the G, and $O(|K|)$ time for reporting the k-gons that form our



(a) Arrangement of five line segments. Numbers 1 through 7 indicate intersections from left to right events. The two valid 4-gons formed by the segments are shown by red and blue dotted lines.



(b) Reporting graph. The edge shown as dash-dotted was added after its predecessors and therefore does not contribute to a valid 4-gon. The other two valid dotted paths to the sink $(c, a, 2)$ represent the closed 4-gons with vertices $(1, 2, 4, 6)$ and $(1, 2, 3, 7)$.

Figure 4: An arrangement and the corresponding reporting graph.

output set K. Since G has $O(n^2)$ vertices and $O(mn)$ edges, the total space requirement is $O(n^2 + mn)$.

4 3SUM-Hardness

In this section, we show that counting the number of triangles in an arrangement of straight-line segments is at least as hard as the 3SUM problem. Since it is widely believed that 3SUM cannot be solved in $o(n^2)$ time, this also holds for the problem of counting triangles.

Theorem 1 *Counting the number of triangles in an arrangement of straight-line segments is 3SUM-hard.*

Proof. We reduce the problem Point-on-3-Lines to counting triangles. Gajentaan and Overmars showed that Point-on-3-Lines is as hard as 3SUM [6]. In Point-on-3-Lines one has to decide whether a given arrangement of straight-lines contains a point in which at least three lines intersect. It is easy to see that the problem remains 3SUM-hard even if no pair of lines is parallel. We transform such an arrangement of lines (with no parallel pairs) to an arrangement of straight-line segments by shortening all lines to segments. We must ensure that all crossings of lines are preserved as crossings of the corresponding segments. To this end, we determine the bounding box of the line arrangement, which is not hard to achieve in $O(n \log n)$ time.

Consider the resulting arrangement. Since it contains all crossings, each triple of segments forms a triangle unless either the three segments intersect in a single point, or (ii) two of the segments are parallel—which cannot happen since the input lines did not contain parallel pairs. Therefore, the arrangement of segments contains $\binom{n}{3}$ triangles if and only if there is no point in which three or more lines intersect.

We have seen that we can check the existence of a point lying on at least three lines by counting the triangles in the arrangement of segments. Furthermore, transforming the instance and determining the number α needed only constant time. Hence, an $o(n^2)$-time algorithm for counting triangles in segment arrangements implies an $o(n^2)$-time algorithm for Point-on-3-Lines. □

We can use almost the same 3SUM-hardness proof for convex quadrilaterals rather than triangles. Observe that any arrangement of four straight-lines (without parallel pairs) forms exactly one quadrilateral face unless three of the lines meet in a point. Hence, the number of quadrilaterals is $\binom{n}{4}$ if and only if there is no triple of lines meeting in a point.

Theorem 2 *Counting the number of convex quadrilaterals in an arrangement of straight-line segments is 3SUM-hard.*

Unfortunately, for larger values of k, e.g., $k = 5$ the hardness reduction does not seem easy to adjust. The problem is that not every set of five straight lines forms a 5-gon, even if they are in general position.

5 Conclusion

We introduced the problem of counting and reporting k-gons in an arrangement of line segments, and presented an $O(n \log n + mn)$ time algorithm for counting all the k-gons, for any fixed constant k, where m is the number of intersecting segment pairs. Our algorithm for reporting all the k-gons runs in time $O(n \log n + mn + |K|)$, where K is the output set. We also prove that the k-gon counting problem is 3SUM-hard for $k = 3$ and $k = 4$.

References

[1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 100(9):643–647, 1979.

[3] B. Chazelle. Reporting and counting segment intersections. *Journal of Computer and System Sciences*, 32(2):156–182, 1986.

[4] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete & Computational Geometry*, 7(1):45–58, 1992.

[5] S. Felsner and K. Krieger. Triangles in euclidean arrangements. In *Graph-Theoretic Concepts in Computer Science*, pages 137–148. Springer, 1998.

[6] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165 – 185, 1995.

[7] J. S. Mitchell, G. Rote, G. Sundaram, and G. Woeginger. Counting convex polygons in planar point sets. *Information Processing Letters*, 56(1):45–49, 1995.

[8] G. Rote and G. Woeginger. Counting convex k-gons in planar point sets. *Information Processing Letters*, 41(4):191–194, 1992.

[9] G. Rote, G. Woeginger, Z. Binhai, and W. Zhengyan. Counting k-subsets and convex k-gons in the plane. *Information Processing Letters*, 38(3):149–151, 1991.

A Fast 2-Approximation Algorithm for Guarding Orthogonal Terrains

Yangdi Lyu[*] Alper Üngör[*]

Abstract

Terrain Guarding Problem(TGP), which is known to be NP-complete, asks to find a smallest set of guard locations on a terrain T such that every point on T is visible by a guard. Here, we study this problem on 1.5D orthogonal terrains where the edges are bound to be horizontal or vertical. We propose a 2-approximation algorithm that runs in $O(n \log m)$ time, where n and m are the sizes of input and output, respectively. This is an improvement over the previous best algorithm, which is a 2-approximation with $O(n^2)$ running time.

1 Introduction

Optimal placement of antennas, cameras, and light sources on terrains is important for communication network, security, and architectural design applications. Even a consideration of the problem on 1.5D terrains is useful whenever the domain is a highway, street, or a hallway. Moreover, this simpler version plays a role on the complexity analysis and algorithm design for the guarding problem on higher dimensional terrains.

A 1.5D terrain T is an x-monotone polygonal chain consists of n vertices $v_i \in \mathbb{R}^2$, for $i = 1, 2, \ldots, n$ and $n-1$ edges $e_i = \overline{v_i v_{i+1}}$ for $i = 1, 2, \ldots, n-1$. T is called an *orthogonal terrain* if all its edges are either horizontal or vertical, and there are no two consecutive horizontal/vertical edges. For two vertices $p, q \in T$, we say p is left of q, denoted as $p < q$, if $p.x < q.x$. The vertices of T are indexed from left to right, so $v_{i+1} \nless v_i$. For $p, q \in T$, p can *see* q if the line segment $\overline{pq}$ is never strictly below the terrain T.

Given a terrain T, a guarding candidate set $G \subseteq T$ and a witness set $W \subseteq T$, terrain guarding problem TGP(G, W) is to find the minimum guarding set $G^* \subseteq G$ such that each point in W is seen by at least one point in G^*. For orthogonal terrains, we refer to this problem as OTGP. Here, we focus on solving OTGP$(V(T), V(T))$ where both the guarding candidate set and the witness set are the vertices of the terrain, i.e., $G = W = V(T)$.

[*]Dept. of Computer & Info. Sci. & Eng., University of Florida, {yangdi, ungor}@cise.ufl.edu

1.1 Related Work

The terrain guarding problem is closely related to the well known Art Gallery Problem [12] of finding the minimum set of positions to guard a polygon. The first result was obtained by Chvátal: $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary to guard a polygon of n vertices. Art Gallery Problem was shown to be NP-hard: on simple polygons [10], on simple orthogonal polygons [13], and on monotone polygons [9]. Moreover, it was shown to be APX-hard on simple polygons [3].

Terrain Guarding Problem for general 1.5D terrains is shown to be NP-hard by a reduction from Planar 3sat [8]. Ben-Moshe *et al.* [1] gave the first $O(1)$-approximation algorithm. Elbassioni *et al.* [4] gave an improvement by showing that LP rounding results in a 4-approximation for TGP(G, W) if $G \cap W = \emptyset$ (a 5-approximation otherwise). A local search based PTAS is also proposed for TGP [5, 6].

For orthogonal terrains, Katz and Roisman [7] gave a 2-approximation algorithm that runs in $O(n^2)$ time, by computing a minimum clique cover in chordal graphs. Recently, Durocher *et al.* [2] and Mehrabi [14] studied the orthogonal terrain guarding problem under *directed visibility* where two vertices u, v are considered to see each other only if the interior of the segment uv is strictly above the terrain. Under this restricted definition, no reflex vertex of the input terrain T can see convex vertices both on its left and right side. This property simplifies the problem, and leads to a linear time greedy exact algorithm. Under standard visibility, Durocher *et al.* [2] also observed that the hardness result for TGP in [8] does not apply for orthogonal terrains, leaving the complexity of OTGP open.

1.2 Our Contribution

The local search based PTAS can be used for approximation of orthogonal terrain guarding problem, but the running time makes it cumbersome for practical use. It takes $n^{O(\alpha/\epsilon^2)}$ time to achieve $O(1+\epsilon)$ approximation, where α is a suitably large constant [6]. Katz and Roisman [7] subdivided the orthogonal terrain guarding problem into two sub-problems and reduce each problem to the problem of computing a minimum clique cover in chordal graphs, where computing the chordal graph takes $O(n^2)$ time. Our algorithm borrows the idea of subdividing the problem into two problems, but avoids computing the chordal graph to reduce the running time and achieve the same approximation factor.

2 Preliminaries

An input terrain is *standard* if it begins and ends with vertical edges.

2.1 Standardization

In this section, we will show how to transform an input terrain which begins and ends with horizontal edges to a standard terrain. For other terrains, it is similar.

First we extend the terrain by adding two edges. Let the leftmost vertex be u, and the rightmost vertex be v. We add two vertical edges $\overline{uu'}$ and $\overline{vv'}$ with infinitesimal length to both of them. Each newly added vertex is the upper endpoint of its new edge. Let the extended terrain be T'. We have $V(T') = V(T) \cup \{u', v'\}$.

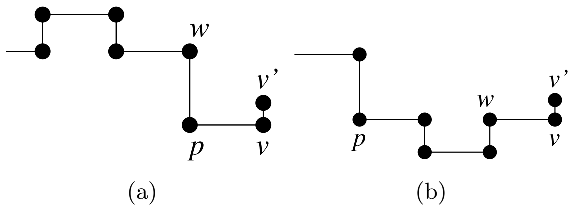


Figure 1: (a) the vertex next to v is a convex vertex (p). (b) the vertex next to v is a reflex vertex (w). If v' is a guard, we can replace v' with w.

Lemma 1 *The cardinality of the optimal solution for OTGP($V(T), V(T)$) is the same as the cardinality of the optimal solution for OTGP($V(T'), V(T')$), and we can easily transform from the solution of the latter to the solution of the former.*

Proof. Let G be an optimal solution for OTGP($V(T), V(T)$), G' be an optimal solution for OTGP($V(T'), V(T')$). Suppose $g \in G$ can see u, g can also see u', so u' is seen by G. Similarly, v' is also seen by G. $G \subseteq V(T) \subset V(T')$, we have G is a solution for OTGP($V(T'), V(T')$). $|G'| \leqslant |G|$.

If neither of u' and v' is in G', then $G' \subseteq V(T)$ and G' can see $V(T)$, so G' is a solution for OTGP($V(T), V(T)$). $|G| \leqslant |G'|$. If $v' \in G'$, there are two cases depending on the vertex next to v. If the vertex next to v is a left convex vertex as in Figure 1a. v' can only see p, w and v, so we can replace v' with w. It is easy to see that w is not in G', otherwise we get a better solution than G' for OTGP($V(T'), V(T')$), it is a contradiction. Similarly we can find a replacement for v' when the vertex next to v is a reflex vertex, see Figure 1b. We can also find a replacement for u' if $u' \in G'$ symmetrically . Suppose we get an optimal solution G'' for OTGP($V(T'), V(T')$) after replacements. It is easy to see that $G'' \subseteq V(T)$ and G'' can see $V(T)$, so G'' is an solution for OTGP($V(T), V(T)$). $|G| \leqslant |G''| = |G'|$.

Thus we have $|G| = |G'|$, and we have also shown how to transform from G' to G.

□

From now on, we will assume that the input terrain is standard.

2.2 Definitions

$V(T)$ is split into two disjoint subsets as reflex vertices $V_r(T)$ and convex vertices $V_c(T)$. Walking along the orthogonal terrain T from left to right, a vertex v is *convex(reflex)* if we turn left(right) at v. Each subset is further split into two subsets depending on whether a vertex is on the left or on the right side of its incident horizontal edge. Specifically, walking along T from left to right, a vertex v is *left(right)* if we walk from a vertical(horizontal) edge to a horizontal(vertical) edge at v. So, $V(T)$ is split into four disjoint subsets: left reflex vertices $V_{lr}(T)$, right reflex vertices $V_{rr}(T)$, left convex vertices $V_{lc}(T)$, and right convex vertices $V_{rc}(T)$, see Figure 2. The first and the last vertices of T can also be labelled simply by considering dummy horizontal edges incident to them.

For each $v \in V_c(T)$, *upper vertex* of v, $U(v) \in V_r(T)$ is the reflex vertex that shares a common vertical edge with v, see Figure 2. As T begins and ends with vertical edges, $U(v)$ for each convex vertex v is well defined.

For each $v \in V_{lc}(T)$, *right horizon* of v, $R(v) \in V_r(T)$ is the rightmost reflex vertex that can see v, see Figure 2. This definition is similar to that of $R(v)$ by Durocher *et al.* [2] except that a left convex vertex cannot be seen by right reflex vertices under directed visibility but it can be seen by them under standard visibility.

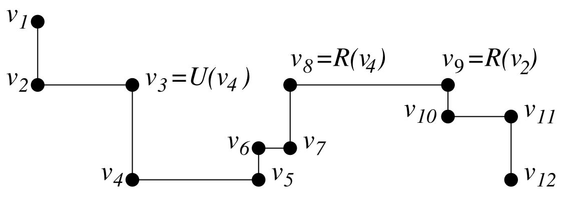


Figure 2: $V_{lc}(T) = \{v_2, v_4, v_{10}, v_{12}\}$, $V_{rc}(T) = \{v_5, v_7\}$, $V_{lr}(T) = \{v_6, v_8\}$, $V_{rr}(T) = \{v_1, v_3, v_9, v_{11}\}$

The following definition adopted from Hurtado *et al.* [11] will be key part of the sweepline algorithm presented in the next section that sweeps the terrain from right to left.

Definition 1 [11] *Given a reflex vertex p_i and a vertex $v_k \in T$, the ray with origin p_i and vector $\overrightarrow{p_i v_k}$ is called a* shadow ray *if: (i) p_i sees v_k; (ii) p_i does not see the points of T immediately to the left of v_k.*

For each shadow ray $\overrightarrow{p_i v_k}$, v_k is called the *obstacle* of p_i, $obs(p_i)$. By definition, there may be multiple shadow rays for each vertex p, corresponding to different obstacles. Our sweepline algorithm relies on the following definition to identify a unique shadow ray (and its obstacle). The shadow ray of p with respect to the sweep line at event w, $sr_w(p)$, is defined as the highest shadow ray of p whose obstacle is to the right of the sweep line at event w, see Figure 3. In the following sections, a shadow ray of p refers to the shadow ray of p with respect to the current sweep line. If two shadow rays, $\overrightarrow{pu}$ and $\overrightarrow{qv}$, intersect and the intersection is not p or q, we define this intersection as an *interior intersection* of these two shadow rays. In our algorithm, lower envelope of shadow rays is maintained to extract some essential visibility information efficiently.

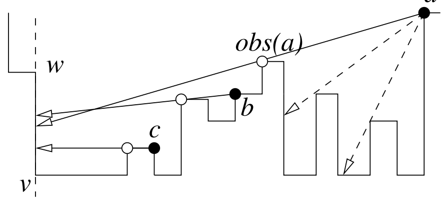


Figure 3: The shadow rays of a, b and c with respect to sweep line w. Obstacles are denoted by empty circles.

2.3 Properties of Orthogonal Terrains

The following claim called *the order claim* was proved by Ben-Moshe *et al.* [1], and holds in 1.5D general terrains.

Lemma 2 [1] *Let $p < q < r < s$ be four points on terrain T. If p sees r, and q sees s, then p sees s.*

The following claims were proved by Katz and Roisman [7] for orthogonal terrains.

Lemma 3 [7] *Let T be an orthogonal terrain and $v \in V_{lc}(T)$. If a point p in T can see v, then $p \nless v$.*

Lemma 4 [7] *If a set G of points on orthogonal terrain T guards a subset $V' \subseteq V_c(T)$, then there exists a subset $G' \subseteq V_r(T)$, such that G' guards V' and $|G'| \leqslant |G|$.*

Lemma 5 [7] *If $G \subseteq V(T)$ guards all the convex vertices of an orthogonal terrain T (i.e., G guards the set $V_c(T)$), then G guards all the vertices of T.*

3 Approximation Algorithm

Given an orthogonal terrain T, our algorithm computes a subset of $V(T)$ that can guard all vertices of T, and we prove that the output of our algorithm is at most twice the size of the optimal solution for OTGP$(V(T), V(T))$.

By Lemmas 4 and 5, our problem can be reduced to OTGP$(V_r(T), V_c(T))$ [7]. Let $G^* \subseteq V_r(T)$ be an optimal solution for OTGP$(V_r(T), V_c(T))$, G^* can guard all convex vertices. So, of course, G^* can guard all left convex vertices, i.e., G^* has at least the same size as the optimal solution for OTGP$(V_r(T), V_{lc}(T))$. The same is true for $V_{rc}(T)$, the right convex vertices.

Our algorithm first computes the optimal solutions for OTGP$(V_r(T), V_{lc}(T))$ and OTGP$(V_r(T), V_{rc}(T))$, then take the union of these two sets. Our solution can guard all convex vertices, and has the size at most twice as G^*, which means it is a 2-approximation.

In the following sections, we will present a sweep line algorithm that computes the optimal solution for OTGP$(V_r(T), V_{lc}(T))$. The right convex vertices part is symmetric.

3.1 Data Structures

Our algorithm sweeps the terrain from right to left and puts each left convex vertex u into an associated list of a unique reflex vertex v, called $L(v)$. When the algorithm terminates, the set of all vertices with non-empty associated lists forms the solution, with each reflex vertex responsible to guard all left convex vertices in its associated list. In addition to the associated lists, following data structures are used:

(1) A modified stack $\mathcal{MS}$ to store a set of all reflex vertices each with a non-empty associated list that can potentially guard more left convex vertices beyond the sweep line. In addition to the standard stack operations (Top, Pop, Push), this modified data structure also supports deletion from any place in the stack given a pointer to that element. Along with each vertex in $\mathcal{MS}$, we also dynamically maintain its obstacle which defines the unique shadow ray with respect to the current sweep line.

(2) A heap, $\mathcal{H}$, to maintain the interior intersections of shadow rays of vertices adjacent in $\mathcal{MS}$.

(3) An event queue $\mathcal{EQ}$ that consists of two components, a list $\mathcal{EQ}_T$ to keep all vertices of T, and a pointer $\mathcal{EQ}_I$ for $\mathcal{H}$. Next event is the rightmost vertex/intersection from $\mathcal{EQ}_T$ and $\mathcal{EQ}_I$. After handling an event, we delete it from the corresponding component of the queue.

(4) A standard stack, $\mathcal{UHS}$, to store the upper chain of the convex hull (upper hull for short) used for computing right horizons $R(v)$.

For each vertex v in $\mathcal{MS}$, we keep two pointers for the shadow ray intersections with its two neighbors. Pointers corresponding to missing neighbors/intersections are set to null. Symmetrically, for each intersection in $\mathcal{H}$, we use two pointers to reach the origins of the corresponding shadow rays in $\mathcal{MS}$.

3.2 Computing Right Horizons

To compute $R(v)$, the rightmost vertex visible from a left convex vertex v, we use the sweep line algorithm for computing the upper hull of a point set.

Lemma 6 *Let v be a left convex vertex. If v is the rightmost vertex on terrain T, $R(v) = U(v)$. Otherwise, $R(v)$ is the vertex right next to v on the upper hull of all vertices to the right of v together with v.*

Proof. If v is the rightmost vertex, it is easy to see that $U(v)$ is the rightmost reflex vertex that can see v, i.e., $R(v) = U(v)$. Otherwise, v is always on the upper hull of the considered vertices since it is the leftmost one. There must be some vertex to the right of v on the upper hull, because the rightmost vertex is always on the upper hull. Let p be the vertex next to v on the upper hull, so $\overline{vp}$ is nowhere below the terrain, i.e., p can see v. For any vertex q to the right of p, as a property of upper hull, we have p higher than $\overline{qv}$, which means q cannot see v. So $R(v) = p$. □

With the upper hull of the swept vertices maintained in $\mathcal{UHS}$, $R(v)$ of a vertex v on the sweep line can be found in constant time. Since T is x-monotone, $\mathcal{UHS}$ can be maintained in linear time in total over all events.

3.3 Sweep Line Algorithm

Our algorithm which sweeps the terrain from right to left is depicted below. Handling of each event consists of updates on the relevant data structures, described below after Observation 1 which motivates the first step in handling a right reflex vertex event.

Algorithm 1 TERRAIN–SWEEPING

```
1: Initialize $\mathcal{H}$, $\mathcal{MS}$ and all $L(v)$ to be empty
2: Initialize $\mathcal{EQ}$ using $T$ and $\mathcal{H}$
3: while $\mathcal{EQ}_T \neq \emptyset$ do
4:     Let $v$ be next event in $\mathcal{E}$
5:     if $v \in V(T)$ then
6:         Update $\mathcal{UHS}$
7:         Handle the vertex $v$
8:     else
9:         Handle the intersection $v$
10:    end if
11: end while
12: Return $\{g \mid L(g) \neq \emptyset\}$
```

Observation 1 *A right reflex vertex can see at least one left convex vertex which is right below it, and at most two left convex vertices.*

1. Left convex vertex v (Line 7):

   (i) Repeatedly Pop($\mathcal{MS}$), until Top($\mathcal{MS}$) can see v or Top($\mathcal{MS}$) is to the right of $R(v)$.

   (ii) If Top($\mathcal{MS}$) sees v, add v to L(Top($\mathcal{MS}$)). Otherwise, Push $R(v)$ to $\mathcal{MS}$, add v to $L(R(v))$, and set *obs*$(R(v))$ be v, see Figure 4a. $\overrightarrow{R(v)v}$ is called a *dummy shadow ray.*

2. Right convex vertex v (Line 7): the only update is to $\mathcal{UHS}$ (in Line 6), so nothing to be done in Line 7.

3. Left reflex vertex v (Line 7):

   (i) Repeatedly Pop($\mathcal{MS}$) until Top($\mathcal{MS}$) cannot see v. Push back the last popped vertex that can see v, and update its obstacle to be v, see Figure 4b.

   (ii) Whenever deleting a vertex from $\mathcal{MS}$, remove its corresponding intersections from $\mathcal{H}$. For the vertex that is pushed to $\mathcal{MS}$, insert the shadow ray intersection with its neighbor to $\mathcal{H}$ and set the corresponding pointers.

4. Right reflex vertex v (Line 7):

   (i) Let u=Top($\mathcal{MS}$). Iteratively Pop($\mathcal{MS}$) if Top($\mathcal{MS}$) is lower than v. If u is lower than v and there is only one vertex p in $L(u)$, delete p from $L(u)$, add p to $L(v)$, and push v to $\mathcal{MS}$, see Figure 4c. To correctly compute the intersections introduced by the new vertex v in $\mathcal{MS}$, we set *obs*(v) one step ahead to be the vertex who shares the same horizontal edge with v.

   (ii) Delete all vertices in $\mathcal{MS}$ that can see v except for the rightmost one.

   (iii) Update intersections in $\mathcal{H}$ as in 3(ii).

5. Intersection v (Line 9):

   (i) If intersection v is above terrain T, delete all vertices from $\mathcal{MS}$, whose shadow rays are incident in v, except for the rightmost one, see Figure 4d.

   (ii) Update the intersections and pointers as in 3(ii)

3.4 Correctness

We say a stack satisfies *left to right order* if the vertices in the stack from top to bottom are ordered from left to right on the terrain. We say a stack satisfies *lower to higher order* if the vertices in the stack from top to bottom are ordered from lower to higher on the terrain. If the stack satisfies both left to right order and lower to higher order, we say the stack is *in order.*

Lemma 7 *$\mathcal{MS}$ is always in order throughout Algorithm 1. The slope of each shadow ray is never negative, i.e., for each vertex u in $\mathcal{MS}$, obs(u) is never higher than u.*

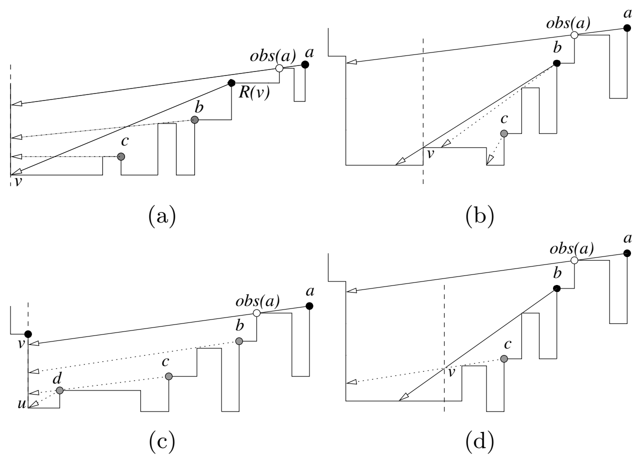


Figure 4: (a) $v \in V_{lc}(T)$: remove reflex vertices from $\mathcal{MS}$ that are to the left of $R(v)$ and cannot see v, and add dummy shadow ray. (b) $v \in V_{lr}(T)$: remove all vertices from $\mathcal{MS}$ that can see v except the rightmost one. (c) $v \in V_{rr}(T)$: delete all vertices that are lower than v. If $L(d)$ contains only one vertex, push v. (d) Intersection v: delete all vertices whose shadow rays are incident in v except the rightmost one.

Proof. (By induction.) Initially, $\mathcal{MS}$ is empty. So the base case is trivial. Suppose before sweeping to event v, $\mathcal{MS}$ is in order and $obs(u)$ is no higher than u for each u in $\mathcal{MS}$.

(1) If v is a left convex vertex, there are two cases. (i) If there exists any vertex in $\mathcal{MS}$ that can see v, we only pop vertices from $\mathcal{MS}$, so it is still in order. (ii) If no vertex can see v, all vertices to the left of $R(v)$ are deleted, and $R(v)$ is pushed into $\mathcal{MS}$. So, the left to right order is maintained. Next, we need to prove that all remaining vertices in $\mathcal{MS}$ are no lower than $R(v)$. Suppose there exists such vertex u in $\mathcal{MS}$ that is lower than $R(v)$. Then a walk from $R(v)$ to u on the terrain must go down a right reflex vertex w that is higher than u. It is easy to see that u cannot see any left convex vertex between $R(v)$ and w, so it must have been pushed to $\mathcal{MS}$ before the sweep line reached w. However, when the sweep line arrives at w, u is deleted from $\mathcal{MS}$ as it is lower than w as case (4) shows. It is a contradiction. So all the other vertices are higher than $R(v)$. Also it is easy to see that the slope of dummy shadow ray $\overrightarrow{R(v)v}$ is positive.

(2) If v is a right convex vertex, the only operation is updating the upper hull, $\mathcal{MS}$ remains the same.

(3) If v is a left reflex vertex, we delete some vertices from $\mathcal{MS}$ and update the obstacle of a vertex p to be v. As p can see v and v is a left reflex vertex, v is no lower than p.

(4) If v is a right reflex vertex, as $\mathcal{MS}$ is in order by induction, step 4(i) ensures all the vertices that are lower than v are deleted. Then if we push v back to $\mathcal{MS}$, it is in order. Our newly introduced shadow ray is horizontal and the remaining operations are deletions.

(5) If v is an intersection, we only delete some vertices from $\mathcal{MS}$.

Other than these events, $\mathcal{MS}$ will not change. So we can conclude that $\mathcal{MS}$ is always in order and the slope of each shadow ray is never negative. □

As a result of this lemma along with the definition of shadow ray, we can see that the obstacles can only be left reflex vertices except for the dummy shadow rays.

Lemma 8 *For each vertex v in $\mathcal{MS}$, v and $obs(v)$ correctly define $sr_w(v)$ where w is the current event. Shadow rays of vertices in $\mathcal{MS}$ have no pairwise interior intersections to the right of w, and are ordered from lower to higher corresponding to the order of their origins in $\mathcal{MS}$, with the lowest shadow ray corresponding to Top($\mathcal{MS}$).*

Proof. (By induction.) Initially, $\mathcal{MS}$ is empty, hence the base case is trivial. Suppose before dealing with event w the claim holds.

(1) w is a left convex vertex: The shadow rays remain the same if the lowest shadow ray can see w. Otherwise, the vertices lower than $R(w)$ are deleted, and $R(w)$ is pushed into $\mathcal{MS}$ with $obs(R(w)) = w$. Let u be the vertex next to Top($\mathcal{MS}$). By definition, $sr_w(u)$ should be no lower than $R(w)$. u cannot see w as it is to the right of $R(w)$, i.e., $sr_w(u)$ is higher than w. So, $sr_w(u)$ is higher than $\overrightarrow{R(w)w}$ and no interior intersection is introduced to the right of w. The lemma holds.

(2) w is a right convex vertex: The shadow rays remain the same.

(3) w is a left reflex vertex: It is the only place we may need to update obstacles to keep the shadow rays correct. As the shadow rays are in order from lower to higher, all the vertices that can see w are near the top of $\mathcal{MS}$ and are consecutive. So, our algorithm correctly finds all shadow rays that need to be updated. We delete all of them except the highest shadow ray which correspond to the rightmost vertex v in $\mathcal{MS}$ that is visible from w, then update $sr_w(v)$. Similar to the arguments in case (1), $sr_w(v)$ is lower than the shadow rays of all vertices in $\mathcal{MS}$.

(4) w is a right reflex vertex: The only place to push a vertex to $\mathcal{MS}$ is the first step and it can only push w. Suppose w is pushed into $\mathcal{MS}$. In the second step, if w is higher than the shadow ray of p to its right in $\mathcal{MS}$, we will delete w from $\mathcal{MS}$. Otherwise the shadow ray of w is also lower than the shadow rays of all the other vertices in $\mathcal{MS}$.

(5) w is an intersection: Under the induction assumption, the rightmost intersection appears between shadow rays of adjacent vertices in $\mathcal{MS}$. The way we maintain

$\mathcal{H}$ ensures w as the rightmost intersection. All shadow rays incident in w are deleted except one, so w disappears. ☐

We say a point $p \in V_r(T)$ dominates point $q \in V_r(T)$, if p can see every point $v \in V_{lc}(T)$ to the left of the sweep line that is visible by q.

Lemma 9 *All vertices deleted from $\mathcal{MS}$ are either dominated by some vertex in $\mathcal{MS}$ at the end of current iteration, or cannot see any left convex vertex to the left of current sweep line.*

Proof. Consider five types of event v:

(1) v is a left convex vertex: We prove that all deleted vertices are dominated by the vertex whose associated list contains v at the end of current iteration. Let this vertex be p. As $\mathcal{MS}$ is in order, any deleted vertex u is to the left of p and to the right of v. Suppose u can see q to the left of the sweep line. So we have $q < v < u < p$, q can see u, and v can see p. According to Lemma 2, q can see p; hence, p dominates u.

(2) v is a right convex vertex: No vertex is deleted.

(3) v is a left reflex vertex: Let p be the rightmost vertex in $\mathcal{MS}$ that can see v. We prove that all deleted vertices are dominated by p. Any deleted vertex u must see v. Hence, $v < u < p$. Using a proof similar to case (1) and Lemma 2, we conclude that p dominates u.

(4) v is a right reflex vertex: All vertices deleted in the first step are lower than v, so they cannot see any left convex vertex to the left of the sweep line. Similar to case (3), all vertices deleted in the second step are dominated by the rightmost one in $\mathcal{MS}$ that can see v.

(5) v is an intersection: We prove that all deleted vertices are dominated by the rightmost vertex p in $\mathcal{MS}$ whose shadow ray crosses v. As $\mathcal{MS}$ is in order, any deleted vertex u is lower than and to the left of p. Suppose u can see q to the left of the sweep line, i.e., $q < v$. Segment $\overline{qu}$ is nowhere below the terrain T and intersects segment $\overline{vp}$ in its interior. So $\overline{qp}$ is nowhere below the terrain T, which means p can see q, see Figure 5. ☐

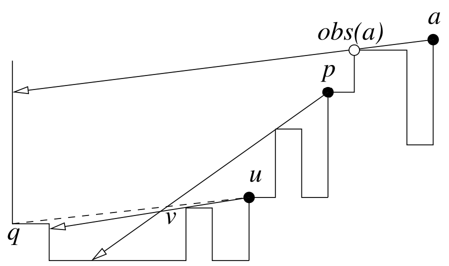


Figure 5: intersection v: u is dominated by p.

Applying Lemma 9, we can get the following corollary.

Corollary 10 *For any $v \in V_{lc}(T)$, if v is seen by some vertex in $\mathcal{MS}$ before the sweep line reaches v, then v is seen by some vertex in $\mathcal{MS}$ when the sweep line arrives at v.*

Let our solution be set G, so we have for each $g \in G$, $L(g)$ is not empty.

Lemma 11 *For each left convex vertex v, there is a unique $g \in G$ such that $v \in L(g)$.*

Proof. Before the sweep reaches v, v is not added to the list of any vertex. When the sweep line arrives at v, v is added to some list $L(u)$. After that, the only operation that may change the list containing v is the first step in handling a right reflex vertex. If $L(u)$ contains some vertex other than v, v will be in $L(u)$ till the end of the algorithm. If $L(u)$ contains only v, when u is popped in the first step of handling right reflex vertex w, v will be deleted from $L(u)$ and added to $L(w)$, then it will never change. In either case, when the algorithm terminates, there is a unique $g \in G$ such that $v \in L(g)$. ☐

Optimality of G will be based on the following set definition also appearing in [2]. Let $F = \{v | v \text{ is the first left convex vertex in } L(g), \text{ for each } g \in G\}$. Observe that the sizes of the sets of F and G are the same, i.e., $|F| = |G|$. Moreover, for any vertex $v \in F$, we know that when the sweep line arrives at v, there is no vertex in $\mathcal{MS}$ that can see v. A lemma similar to the following is given in [2] except that their definition of visibility allows only left reflex vertices to see left convex vertices, stated as case 1 here.

Lemma 12 *For any two vertices $u, v \in F$, there are no reflex vertices that can see both of them.*

Proof. To prove by contradiction, suppose w is a reflex vertex that can see both $u, v \in F$. Without loss of generality, let $u < v$, so we visit v first. Then we prove that there exists some vertex in $\mathcal{MS}$ that can see u before the sweep line reaches u.

Case 1: w is a left reflex vertex: We have $u < v < w$. By definition of $R(v)$, $w \leqslant R(v)$. Using Lemma 2, $R(v)$ can see u. When we visit v, we add $R(v)$ to $\mathcal{MS}$.

Case 2: w is a right reflex vertex: w should be $U(v)$. It is easy to see that $R(v)$ is Top($\mathcal{MS}$) when the sweep line arrives at w. If $R(v)$ is higher than $U(v)$, $\overline{R(v)u}$ is nowhere below the terrain T, $R(v)$ can see u. Otherwise, $R(v)$ is popped in the first step as it is lower than w, and w is pushed into $\mathcal{MS}$ as $L(R(v))$ contains only v.

In either case there exists some vertex in $\mathcal{MS}$ that can see u before the sweep line reaches u. By Corollary 10, there exists some vertex in $\mathcal{MS}$ when the sweep line arrives at u, which contradicts that $u \in F$. ☐

Lemma 11 implies that the optimal solution of OTGP($V_r(T), V_{lc}(T)$) has at least $|F|$ reflex vertices. Our solution can see all left convex vertices and has size $|G| = |F|$. So we have the following result.

Lemma 13 *Algorithm 1 computes the optimal solution for OTGP*$(V_r(T), V_{lc}(T))$.

Symmetrically we can compute the optimal solution for OTGP$(V_r(T), V_{rc}(T))$, leading to a 2-approximation algorithm for the OTGP$(V(T), V(T))$.

3.5 Running Time

Let k be the size of $\mathcal{MS}$, and t be the number of vertices with non-empty lists outside $\mathcal{MS}$. It is easy to see that the summation of k and t never decreases and eventually it will be m, where m is the output size. As the number of intersections of shadow rays of adjacent vertices in $\mathcal{MS}$ is less than k, the size of $\mathcal{H}$ is $O(m)$. Note that t is increased by at least 1 when handling each intersection. Thus there are $O(m)$ intersection events. Then we analyse the running time associated with each data structure.

(1) $\mathcal{UHS}$. Maintenance of upper hull takes $O(n)$ total time.

(2) $\mathcal{MS}$. The running time is proportional to the cost of stack insertions and deletions. Each deleted vertex when handling right reflex vertex v is lower than v and all the other deleted vertices are dominated by some vertex in $\mathcal{MS}$ by Lemma 9. So, the deleted vertices cannot be inserted again in future iterations, i.e., there are at most n insertions and n deletions. The total running time is $O(n)$.

(3) $\mathcal{H}$ and $\mathcal{EQ}$. There are four cases. (i) Get the next event. If the next event is from $\mathcal{EQ}_T$, it takes constant time and there are n such events, so it takes $O(n)$ time in total; if next event is from $\mathcal{EQ}_I$, it takes $O(\log m)$ time and there are $O(m)$ intersections, so it takes $O(m \log m)$ in total. (ii) Insert vertices into $\mathcal{MS}$. There are $O(n)$ insertions and constant number of new intersections with each insertion, so the time complexity is $O(n \log m)$ in total. (iii) Delete vertices from $\mathcal{MS}$. Similar to case (ii). (iv) Update obstacles. We need to update at most one obstacle at any left reflex vertex, along with two deletions and one insertion with $\mathcal{H}$. As there are $O(n)$ left reflex vertices, the total running time is $O(n \log m)$.

Overall, the running time is $O(n \log m)$.

References

[1] B. Ben-Moshe, M.J. Katz, and J.S.B. Mitchell. A constant–factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[2] S. Durocher, P.C. Li, and S. Mehrabi. Guarding orthogonal terrains. In *Proc. of 27th Canadian Conf. on Comp. Geometry*, 220–227, 2015.

[3] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31:79–113, 2001.

[4] K. Elbassioni, E. Krohn, D. Matijević, J. Mestre, and D. Ševerdija. Improved approximations for guarding 1.5-D terrains. *Algorithmica*, 60:451–463, 2011.

[5] S. Friedrichs, M. Hemmer, and C. Schmidt. A PTAS for the continuous 1.5D terrain guarding problem. In *Proc. of 26th CCCG*, 367–373, 2014.

[6] M. Gibson, G. Kanade, E. Krohn, and K.Varadarajan. An Approximation Scheme for Terrain Guarding. *Proc. APPROX*, Springer-LNCS:5687, 140–148, 2009.

[7] M. J. Katz and G. S. Roisman. On guarding the vertices of rectilinear domains. *Computational Geometry*, 39(3):219 – 228, 2008.

[8] J. King and E. Krohn. Terrain guarding is NP-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

[9] E. A. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

[10] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[11] F. Hurtado, M. Löffler, I. Matos, V. Sacristán, M. Saumell, R. I. Silveira, and F. Staals. Terrain visibility with multiple viewpoints. *International Journal of Computational Geometry & Applications*, 24(4):275–306, 2014.

[12] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford Univ. Press, New York, NY, USA, 1987.

[13] D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995.

[14] S. Mehrabi. Guarding the Vertices of an Orthogonal Terrain using Vertex Guards. http://arxiv.org/abs/1512.08292, 2015.

An Iterative Refinement Scheme of Dominating Guards and Witnesses for Art Gallery Problems

Eyüp Serdar Ayaz[†]   Alper Üngör[*]

Abstract

The art gallery problem asks to find a smallest set of guards in a polygon such that every point in the polygon is visible by a guard. This problem can be formulated as an instance of the well-known set cover problem and also its dual the hitting set problem. We present an iterative refinement scheme based on this dual formulation. Two of the main ingredients of this scheme are the witness sets and the dominating guard sets. Here, we extend some recent results and algorithms for computing minimal witness sets to those for computing dominating guard sets. In particular, an $O(n^5)$ time algorithm is presented for computing dominating guard sets.

1 Introduction

The Art Gallery Problem (AGP) is one of the best known geometric algorithms problem with many applications, e.g., security, surveillance, sensor networks, and architectural design. It was originally proposed by Klee to Chvátal in 1973 as a challenge to find the point locations of a minimum number of guards such that each point in the wall of an art gallery is seen by at least one guard [13]. In general, however, the gallery interior needs to be guarded too. Chvátal has proven that $\lfloor n/3 \rfloor$ guards are always sufficient and occasionally necessary on simple polygons with n vertices for the generic version which also holds for the original problem [6]. Later Fisk has simplified the proof with an elegant triangulation and 3-coloring scheme [10]. Both the original [17] and the generic [18] versions of art gallery problem are proven to be NP-hard. Note that it is not known whether these problems are in NP or not, since the guard positions are not known to be represented polynomial length over the input size. It is clear that a set of points that guards the interior of a polygon can also guard its boundary, but the reverse proposition is not true. In fact, the number of guards in the optimum solution for these two version can differ. Over the years, many different versions of the art gallery problem have been studied, such as the terrain guarding, guarding with mobile guards, and gallery domains with holes or in high dimensions [19, 21].

There is strong similarity between the set cover problem (*SCP*), its dual the hitting set problem (*HSP*) and AGP. We can reduce the art gallery problem to either of them by defining the points in a polygon as the universal set and the visibility polygons of those points as the collection of sets in a set system. For discrete set systems, while both the set cover and the hitting set problems are NP-complete [15], they admit a greedy approximation algorithm [7]. The discretization of the set system for AGP is a viable method that can take advantage of the approximation algorithms designed for SCP and HSP. In fact, to the best of our knowledge, all known approximation algorithms for the generic art gallery problem are based on its formulation as SCP or HSP [12].

In the set cover problem formulation, instead of all the sets given in a set system, we can consider only the sets that are not proper subsets of other sets. Similarly, in the hitting set problem formulation, considering to hit only the sets that have no proper subset in the set system is sufficient to hit all the sets. However, the number of possible guards is uncountably many and so is the number of their visibility polygons. Moreover, checking the inclusion relation between two sets takes $\Theta(\min(n, m))$ time where n and m are the number elements in the sets. Here, we exploit the geometric properties of visibility polygons of points in a simple polygon to find the inclusion minimal and maximal of sets.

A *witness set* W of a polygon P is defined as a set such that any set G that guards W also guards P [8]. The minimal witness sets correspond to inclusion minimal visibility sets for polygons. Chwa et al. [8] have presented an algorithm that calculates a minimal witness set when a polygon admits witness sets with finitely many points. Unfortunately, very few polygons have witness sets consisting of finitely many points. Later, an $O(n^4)$ algorithm is presented to find a (near)-minimal witness set (of points, line segments and interior regions) for a simple polygon [1].

In this paper, we formulate a dual version of the witnessability concept based on the inclusion relations of visibility sets called dominating guards. Previously domination concept for visibility polygons have been studied on discrete art gallery problems [2, 20] (as dominant regions and light atomic visibility polygons respectively) and watchman tours [3, 5]. However, we are not aware of any previous study on dominating guards

[*]CISE Department, University of Florida, Gainesville [ayaz,ungor]@cise.ufl.edu

where the possible guard positions are not limited to be finite. Here we define the same concept on infinitely many points and their corresponding visibility polygons. We propose an algorithm to find a dominating guards set in $O(n^5)$ time. Then, we use both witness sets and dominating guard sets to define an iterative refinement scheme for the art gallery problem.

The organization of the paper is as follows. In section 2, we review the witnessing concept and model the art gallery problem. In section 3, we propose an algorithm to find the dominating guards. In section 4, we extend the dominating guard concepts with specified input. In section 5, we outline our refinement scheme.

2 Preliminaries

The input for the art gallery problem is a simple polygon $P \subset \mathbb{R}^2$ with n vertices. We assume that no three vertices are co-linear for the sake of simplicity. Let $int(P)$, and ∂P denote the interior and the boundary of P, respectively. $P = int(P) \cup \partial$, so when we say a point p is in P, that means p is either in $int(P)$ or on ∂P. It is straight-forward to come up with an $O(n)$-time algorithm to find an optimal solution for AGP on star-shaped polygons, so we assume P is not star-shaped for the rest of the paper. Two points $p, q \in P$ *see* each other if the whole line segment $\overline{pq}$ is in P. If a point p in P sees a reflex vertex v of P and the ray $\overrightarrow{pv}$ continues in P after hitting v then p *sees past* v. For an edge e of P, the closure of the half-plane defined by $\overleftrightarrow{e}$ on the same side as P at the immediate neighborhood of e is denoted as $l^c(e)$. For two points $p, v \in P$ such that p sees past v, the closure of the half-plane defined by $\overleftrightarrow{pv}$ on the same side as P at the immediate neighborhood of v is denoted as $l^c(p, v)$.

The set of points in P that can be seen from a point $p \in P$ is called the *visibility polygon* of p, denoted as $\mathcal{V}(p)$. The set of points that sees all the points in a polygon is called the *kernel* of P. The set of points that can see every point in $\mathcal{V}(p)$ is called the *visibility kernel* of p denoted as $\mathcal{VK}(p)$ (See Figure 1). In fact $\mathcal{VK}(p)$ is the kernel of $\mathcal{V}(p)$. Any point that sees p also sees $\mathcal{VK}(p)$ [8].

Let p be a point on P. $E(p)$ denotes the set of edges of P of which p sees at least one interior point. $R(p)$ denotes the set of reflex vertices of P that are seen past from p. Then, as proven by Chwa et al. [8] $\mathcal{VK}(p)$ can be computed as the intersection of $O(n)$ half-planes:

$$\mathcal{VK}(p) = \bigcap_{v \in R(p)} l^c(p, v) \cap \bigcap_{e \in E(p)} l^c(e) \tag{1}$$

2.1 Witness sets

A finite set of objects $W \subseteq P$ is defined as a *witness set* if, for any set of guards G in P, $W \subseteq \bigcup_{g \in G} \mathcal{V}(g)$ implies $\bigcup_{g \in G} \mathcal{V}(g) = P$, i.e., if G *guards* W then G also guards P.

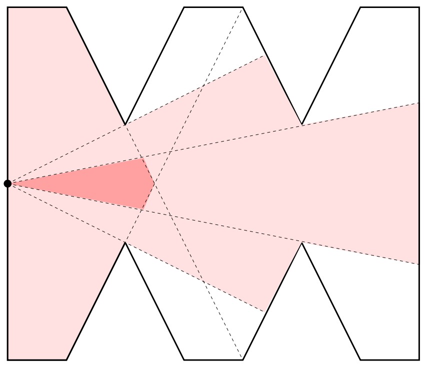


Figure 1: $\mathcal{V}(p)$ is the dark and light shaded areas. $\mathcal{VK}(p)$ is the dark shaded area. p sees past all the reflex vertices in this case.

Theorem 1 (Chwa et. al. [8]) *A point set W is a witness set for a polygon P if and only if $\bigcup_{p \in W} \mathcal{VK}(p) = P$. Also the following statements are equivalent for $p, q \in P$:*
(i) *p witnesses q;* (ii) *$q \in \mathcal{VK}(p)$;* (iii) *$\mathcal{VK}(q) \subseteq \mathcal{VK}(p)$.*

A witness set W for P is said to be *minimal* if there exist no proper subset of W that witnesses P. If there exists a minimal witness set for P, then P is a *minimalizable* polygon. Recently, a characterization of witness sets for simple polygons is given in [1]. Their algorithm finds a minimal witness sets for minimalizable polygons. For polygons that are not minimalizable, their algorithm finds a witness set called *near-minimal witness set*, that has a minimal component of points, line segments and regions and non-minimal component of infinitesimially small line segment incident to reflex vertices, called *ϵ-witnesses* (See Figure 2).

Lemma 2 *Let p, q be in P. Then we have:*
(i) *$\mathcal{VK}(p) = \mathcal{VK}(q)$ iff $\mathcal{V}(p) = \mathcal{V}(q)$*
(ii) *$\mathcal{VK}(p) \subset \mathcal{VK}(q)$ iff $\mathcal{V}(q) \subset \mathcal{V}(p)$*

Proof. (i) $\Leftarrow$: The kernels of $\mathcal{V}(p)$ and $\mathcal{V}(q)$ are equal when $\mathcal{V}(p)$ and $\mathcal{V}(q)$ are equal.

$\Rightarrow$: $p, q \in \mathcal{VK}(p) = \mathcal{VK}(q)$. Therefore any point that sees p also sees q and any point that sees q also sees p.

(ii) $\Leftarrow$: Since $\mathcal{V}(q) \subset \mathcal{V}(p)$, any point that sees q also sees p. Therefore $p \in \mathcal{VK}(q)$ which means $\mathcal{VK}(p) \subseteq \mathcal{VK}(q)$. There exists a point in $\mathcal{V}(p) \backslash \mathcal{V}(q)$, so $q \notin \mathcal{VK}(p)$, but it has to be in $\mathcal{VK}(q)$.

$\Rightarrow$: $p \in \mathcal{VK}(p)$, so $p \in \mathcal{VK}(q)$, which means $\mathcal{V}(q) \subseteq \mathcal{V}(p)$. If $q \in \mathcal{VK}(p)$, then $\mathcal{VK}(p) = \mathcal{VK}(q)$, which is a contradiction. So $q \notin \mathcal{VK}(p)$. Then there exist a point that sees p but not q. $\square$

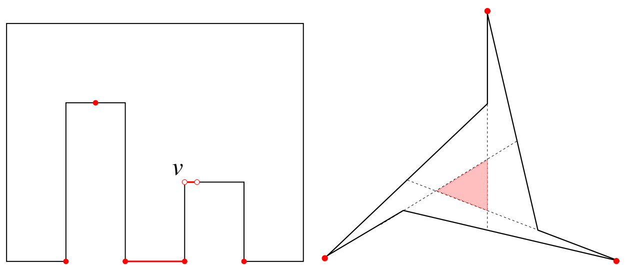


Figure 2: Red points lines and regions indicate the witnesses. (Left) polygon is not a minimalizable polygon. It is necessary to include an infinitesimally small line segment (ϵ-witness) next to the reflex vertex v. (Right) polygon is a minimalizable polygon

Minimal witness sets are defined in terms of visibility kernels of witnesses or visibility polygons of guards in [1, 8]. Using Lemma 2, we can formulate the (near)-minimal witness sets using visibility polygons of witnesses:

Lemma 3 *W is a (near)-minimal witness set for P iff we have the following properties:*
(i) *Let p, q be two distinct points in W. Then $\mathcal{V}(p)$ cannot be a subset of $\mathcal{V}(q)$ unless p and q are on the same ϵ-witness.*
(ii) *Let p be a point in $P \setminus W$. Then $\exists q \in W$ such that $\mathcal{V}(q) \subseteq \mathcal{V}(p)$.*

Proof. $\Rightarrow$: (i) Suppose $\mathcal{V}(p) \subseteq \mathcal{V}(q)$. From Lemma 2, $\mathcal{VK}(q) \subseteq \mathcal{VK}(p)$, which is a contradiction for the minimality condition. (ii) Suppose $\nexists q \in W$ such that $\mathcal{V}(q) \subseteq \mathcal{V}(p)$. Then, from Lemma 2, we there $\nexists q \in W$ such that $p \in \mathcal{VK}(q)$ which is a contradiction with Theorem 1.

$\Leftarrow$: We need to prove that the witnessing condition, $\bigcup_{p\in W} \mathcal{VK}(W) = P$ and the minimality condition, $\forall p, q \in W$ such that p, q are distinct points not on a same ϵ-witness, we have $p \notin \mathcal{VK}(q)$. The witnessing condition is satisfied by (ii) and the minimality condition is satisfied by (i) using Lemma 2. □

2.2 AGP formulation as SCP and HSP

The art gallery problem can be defined with two parameters: AGP(X, Y), where $X, Y \subseteq P$, X is the set of possible guard locations and Y is the set of points to be guarded [9]. AGP(P, P) is the interior guarding problem whereas AGP$(P, \partial P)$ is the wall guarding.

A pair $(U, \mathcal{R})$ consisting of a universal set U, and a collection of subsets $\mathcal{R}$ such that each element of $\mathcal{R}$ is a subset of U is called a *set system*. Given a set system $(U, \mathcal{R})$ and a subset $S \subseteq U$, we define the operation $\mathcal{R} \sqcap S$ as $\{S \cap X | X \in \mathcal{R}\}$.

Given a set system $(U, \mathcal{R})$, a *set cover* is a subset $\mathcal{C}$ of $\mathcal{R}$ where $\bigcup_{x\in\mathcal{C}} x = U$. *Set cover problem*, denoted as *SCP*$(U, \mathcal{R})$, is the quest for finding the cover with minimum elements.

Given a set system $(U, \mathcal{R})$, a *hitting set* is a subset H of U such that for all $S \in \mathcal{R}$, $S \cap H \neq \emptyset$. *Hitting set problem*, denoted as *HSP*$(U, \mathcal{R})$, is the quest for finding the hitting set with minimum elements.

Given a point set $S \subseteq P$, $\mathcal{VS}(S)$ represents the collection of the visibility polygons of the points in S, i.e., $\mathcal{VS}(S) = \{\mathcal{V}(p) | p \in S\}$.

Then, AGP(X, Y) can be formulated as instances of SCP [9] or HSP [16]:

$$\text{AGP}(X, Y) = \text{SCP}(Y, \mathcal{VS}(X) \sqcap Y) \tag{2}$$

$$\text{AGP}(X, Y) = \text{HSP}(X, \mathcal{VS}(Y) \sqcap X) \tag{3}$$

Here, we have a nice symmetry of SCP and HSP, since every point p in a polygon P has a correspoinding set, $\mathcal{V}(p)$, consisting of fellow points in P. Moreover, visibility is a reflexive relation, i.e., $\mathcal{V}(p) = \mathcal{V}^{-1}(p)$ where $\mathcal{V}^{-1}(p)$ denotes the set of points in P that sees p. Therefore we can use the same function, $\mathcal{VS}(S)$, to find the collection of sets corresponding to a subset of points $S \subseteq P$ for both SCP and HSP. For the generic version of art gallery problem, AGP(P, P), the universal set is the same for both SCP and HSP and the cardinality of the universal set is equal to the cardinality of the number of subsets in the set system. Here, the dual formulation with infinitely many points and sets enables an iterative refinement for both X and Y.

$$\begin{tikzcd}
 & \mathcal{V}(p_1) & \\
\mathcal{V}(p_2) \arrow[ur] & \mathcal{V}(p_3) \arrow[u] & \mathcal{V}(p_4) \arrow[ul] \\
\mathcal{V}(p_5) \arrow[ur] & \mathcal{V}(p_6) \arrow[u] & \mathcal{V}(p_7) \\
 & \mathcal{V}(p_8) \arrow[ul] \arrow[u] &
\end{tikzcd}$$

Figure 3: A simple example for the partially ordered set of visibility polygons where $p_i \in P$ and $\mathcal{V}(p_i) \to \mathcal{V}(p_j)$ means $\mathcal{V}(p_i) \subset \mathcal{V}(p_j)$ for $1 \leq i, j \leq 8$. Inclusion minimal sets are $\mathcal{V}(p_2), \mathcal{V}(p_4), \mathcal{V}(p_7)$ and $\mathcal{V}(p_8)$, corresponding to the witness set $\{p_2, p_4, p_7, p_8\}$. Inclusion maximal sets are $\mathcal{V}(p_1)$ and $\mathcal{V}(p_7)$, corresponding to the dominating guard set $\{p_1, p_7\}$.

We exploit the geometric properties of piecewise linear models to calculate the inclusion minimal and inclusion maximal of the poset $(\mathcal{VS}(P), \subseteq)$. The inclusion maximal set of $(\mathcal{VS}(P), \subseteq)$ corresponds to the dominating guard set and inclusion minimal set of $(\mathcal{VS}(Y), \subseteq)$ corresponds to the witness set. The Visibility polygon of a point p in a witness set has no proper subset in $\mathcal{VS}(P)$ unless p is on an ϵ-witness. This is an important property while solving the HSP formulation of AGP, since hitting the sets that are corresponding to witness points is sufficient to hit all the sets.

3 Dominating guard sets

A *dominating point* is a point $p \in P$, where $\nexists q \in P$ such that $\mathcal{V}(p) \subset \mathcal{V}(q)$ and $p \neq q$. The *dominating guard set* for P is the all dominating points in P, denoted as $\mathcal{DG}(P)$. The poset $(\mathcal{VS}(P), \subseteq)$ is defined on the set of visibility polygons, hence there is a unique inclusion maximal set for visibility polygons. But we define $\mathcal{DG}(P)$ using the points in P instead of their visibility polygons which creates redundant elements when two dominating points have identical visibility polygons. To get rid of the redundant points in $\mathcal{DG}(P)$, we define *minimal dominating guard set* (denoted as $\mathcal{DG}^*(P)$) as a set of points $\mathcal{DG}^*(P) \subseteq \mathcal{DG}(P)$, such that for all distinct elements $p, q \in \mathcal{DG}^*(P)$, $\mathcal{V}(p) \neq \mathcal{V}(q)$ and $\mathcal{VS}(\mathcal{DG}^*(P)) = \mathcal{VS}(\mathcal{DG}(P))$.

Then we have the following property:

Lemma 4 *Let p be a point in $P \setminus \mathcal{DG}^*(P)$. Then $\exists q \in \mathcal{DG}^*(P)$ such that $\mathcal{V}(p) \subseteq \mathcal{V}(q)$.*

Proof. If $p \in \mathcal{DG}(P)$, then $\mathcal{V}(p) \in \mathcal{VS}(\mathcal{DG}(P)) = \mathcal{VS}(\mathcal{DG}^*(P))$.

If $p \notin \mathcal{DG}(P)$, then there exists an element $p' \in P$ such that $\mathcal{V}(p) \subset \mathcal{V}(p')$. If $p' \notin \mathcal{DG}(P)$, then there exists an element $p'' \in P$ such that $\mathcal{V}(p') \subset \mathcal{V}(p'')$. We can iterate this until we reach an element in $\mathcal{DG}(P)$, which can be replaced with another element with an identical visibility polygon in $\mathcal{DG}^*(P)$. □

Using $\mathcal{DG}^*(P)$ we can reduce the number of the sets in SCP formulation, since any element p of a guard set is either in $\mathcal{DG}^*(P)$ or we can replace p with an element of $\mathcal{DG}^*(P)$ by Lemma 4.

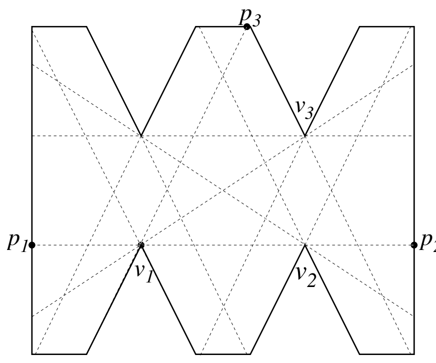


Figure 4: Subdivision of a polygon P, $\mathcal{A}(P)$. $\overline{v_1p_3}$ is of Type 1, $\overline{v_1p_1}$ and $\overline{v_2p_2}$ are of Type 2, $\overline{v_1v_2}$ and $\overline{v_1p_3}$ are of Type 3.

We subdivide P into regions in order to calculate $\mathcal{DG}^*(P)$. The subdivision is created through an arrangement consisting of three types of line segments:

1. Extensions of the edges at reflex vertices.
2. Extensions of line segments between two reflex vertices that see each other.
3. The line segments between two reflex vertices that see past each other.

We denote this subdivision as $\mathcal{A}(P)$. We call the maximal contiguous regions that do not include line segments of the subdivision and ∂P as *cells of* $\mathcal{A}(P)$. The intersection points of the line segments among themselves and with ∂P are called the *vertices of* $\mathcal{A}(P)$ and the open line segments between two vertices of $\mathcal{A}(P)$ are called the edges of $\mathcal{A}(P)$. Note that there are at most $O(n^4)$ vertices, edges and cells of $\mathcal{A}(P)$. (See Figure 4).

Lemma 5 *Let p and q be two points in the same cell of $\mathcal{A}(P)$. Then $R(p) = R(q)$. Similarly, if p and q are points on the same edge of $\mathcal{A}(P)$ we have $R(p) = R(q)$.*

Proof. Let p and q are in the same cell. Suppose $R(p) \neq R(q)$. Without loss of generality, let $r \in R(p)$ and $r \notin R(q)$. Then either q does not see r or q does not see past r: If q does not see r, then there exists a reflex vertex that is seen past by r in the region between the rays $\overrightarrow{rp}$ and $\overrightarrow{rq}$. This puts p and q in different cells, because there is a Type 2 line segment between them. If q sees but does not see past r, then there is a Type 1 line segment between them. Contradiction. Same analysis can be made when p and q are on the same edge of $\mathcal{A}(P)$. □

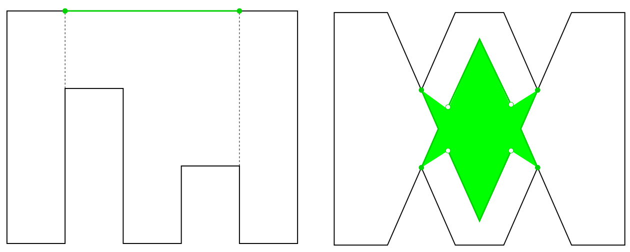


Figure 5: Green indicates $\mathcal{DG}(P)$ which is equal to $\mathcal{DG}^*(P)$ in both left and right polygons.

Lemma 6 *A point p is a dominating point iff $\forall q \in \mathcal{VK}(p)$ we have $\mathcal{V}(p) = \mathcal{V}(q)$.*

Proof. $\Rightarrow$: Suppose $q \in \mathcal{VK}(p)$ such that $\mathcal{V}(p) \neq \mathcal{V}(q)$. From Theorem 1, we have $\mathcal{VK}(q) \subseteq \mathcal{VK}(p)$ which makes $\mathcal{V}(p) \subset \mathcal{V}(q)$ by Lemma 2. This is a contradiction from the definition of a dominating point.

$\Leftarrow$: Suppose p is not a dominating point. Then there is a point q such that $\mathcal{V}(p) \subset \mathcal{V}(q)$. It follows that $\mathcal{VK}(q) \subset \mathcal{VK}(p)$ from Lemma 2 and $q \in \mathcal{VK}(p)$ from Theorem 1. This is a contradiction of the statement. □

Lemma 7 *Let p be a dominating point in a non-star-shaped polygon. Then one of the following statements is correct:*
(i) *$\mathcal{VK}(p) = \{p\}$.*
(ii) *Let $q \in \mathcal{VK}(p)$ and $q \neq p$. Then p and q are on a type 3 line segment in $\mathcal{A}(P)$. Moreover the endpoints of that type 3 line segment are the only reflex vertices either of them see past.*

Proof. Let us assume that neither of the cases correct. Then we have $q \in \mathcal{VK}(p)$ that is not on a line segment between two reflex vertices they see past. From Lemma 6, we have $\mathcal{V}(p) = \mathcal{V}(q)$, hence $R(p) = R(q)$ and $\mathcal{VK}(p) = \mathcal{VK}(q)$. If $R(p) = \emptyset$, then p is in the kernel of P which is not star-shaped, contradiction. Let $r \in R(p)$ that is not co-linear with p and q. Then either $p \notin l^c(p, r)$ or $q \notin l^c(p, r)$ which causes a contradiction from $\mathcal{VK}(p) = \mathcal{VK}(q)$ and (1). □

Lemma 8 *If a point in a cell of $\mathcal{A}(P)$ is a dominating point, all points in that cell are also dominating points.*

Proof. Let p, q be two points in a cell. Let p be a dominating point. From Lemma 7, $\mathcal{VK}(p) = \{p\}$. Suppose that q is not a dominating point. Then $\mathcal{VK}(q) \neq \{q\}$. From (1) and Lemma 5, we have $\{q\} \subset \bigcap_{v \in R(p)} l^c(q, v)$ and $\{p\} = \bigcap_{v \in R(p)} l^c(p, v)$. Let $r_1, r_2 \in R(p)$ be the vertices such that $\bigcap_{v \in R(p)} l^c(q, v) = l^c(q, r_1) \cap l^c(q, r_2)$. Then $\{p\} \subset l^c(p, r_1) \cap l^c(p, r_2)$, since otherwise p and q will be in different sides of the Type 2 line segment induced by $\overline{r_1 r_2}$. Then there exists $r_3 \in R(p)$ such that $\{p\} = l^c(p, r_1) \cap l^c(p, r_2) \cap l^c(p, r_3)$. Then $\{q\} = l^c(q, r_1) \cap l^c(q, r_2) \cap l^c(q, r_3)$, since otherwise p and q will be in different sides of the Type 2 line segment induced by $\overline{r_1 r_3}$ or $\overline{r_2 r_3}$ which is a contradiction. □

Lemma 9 *If a point on an edge of $\mathcal{A}(P)$ is a dominating point, all points on that edge are also dominating points.*

Proof. Let p, q be two points on an edge of $\mathcal{A}(P)$. Let p be a dominating point. If $\mathcal{VK}(p) = \{p\}$ we can make a similar analysis as the proof of Lemma 8.

Assume that $\mathcal{VK}(p) \neq \{p\}$. Then from Lemma 7, p, q on a Type 3 line segment and $|R(p)| = 2$. Let $R(p) = R(q) = \{r_1, r_2\}$. From (1), we have $\mathcal{VK}(p) \subseteq \overline{r_1 r_2}$. Since p and q are on the same edge of $\mathcal{A}(P)$, they see the same edges of P of which extension can intersect $\overline{r_1 r_2}$. Therefore $\mathcal{VK}(p) = \mathcal{VK}(q)$ from (1). □

We first calculate $\mathcal{A}(P)$ in $O(n^4)$ time using the algorithm described in [4]. We can calculate the visibility polygon of a point in $O(n)$ time using the algorithm in [14]. Then, for each cell, we check whether $\mathcal{VK}(p) = \{p\}$ for an arbitrary point in that cell. If so, the whole cell is in $\mathcal{DG}^*(P)$ by Lemma 8. The same test can be made for type 1 and 2 line segments and ∂P. For type 3 line segments, check whether a point p in the edge $\mathcal{VK}(p) = \{p\}$ or $\mathcal{VK}(p) = \overline{r_1 r_2}$ where $R(p) = \{r_1, r_2\}$. If $\mathcal{VK}(p) = \{p\}$, all points on the edge are in $\mathcal{DG}^*(P)$. If $\mathcal{VK}(p) = \overline{r_1 r_2}$, then all points on the edge are in $\mathcal{DG}(P)$, but only an arbitrary point is in $\mathcal{DG}^*(P)$. Since there are $O(n^4)$ for each of them, the total time complexity of calculating a minimal dominating guard set takes $O(n^5)$ time.

4 Dominating guard sets for parametrized AGP

In the previous section, we find the minimal dominating guard set for AGP(P, P). We can use the algorithm given in [1] to find the (near-)minimal witness set W which can further refine the dominating guard set (See Figure 5 (left) and 6). Also we may be given another problem AGP(X, Y) where X or Y are subsets of P, such as AGP$(P, \partial P)$. Therefore, we have given a set of possible guard and witness positions X, Y respectively and we have another poset $(\mathcal{VS}(X) \sqcap Y, \subseteq)$ to find the inclusion maximal. Let X and Y be a collection of points, line segments and convex regions. Let us assume the complexity of X and Y as m and k respectively.

For a set of points S, the set of points that are visible from all points in S is called *strong visibility set* [11], denoted as $\mathcal{SV}(S)$, i.e., $\mathcal{SV}(S) = \bigcap_{p \in S} \mathcal{V}(p)$. For a point $p \in P$ and sets $X, Y \subseteq P$, we denote $\mathcal{SV}(\mathcal{V}(p) \cap Y) \cap X$ as $\mathcal{SVV}(p, X, Y)$.

Corollary 10 *Let us have two sets $S_1, S_2 \subseteq P$. If $S_1 \subset S_2$, then $\mathcal{SV}(S_2) \subseteq \mathcal{SV}(S_1)$.*

In Section 3, we check the hierarchy of the visibility polygons in the poset $(\mathcal{VS}(P), \subseteq)$ using the visibility kernels. For a potential guard p, every point in $\mathcal{V}(p)$ is necessarily to be guarded. So we get the kernel of $\mathcal{V}(p)$ to find other points that sees every point p see. Here, we need to cover only the points in $\mathcal{V}(p) \cap Y$. Therefore the points that see all points in $\mathcal{V}(p) \cap Y$, which is the strong visibility set of $\mathcal{V}(p) \cap Y$. So instead of the visibility kernel of p we use $\mathcal{SVV}(p, X, Y)$ which is equal to $\mathcal{VK}(p)$ if $X = Y = P$. Now, we need to extend the subdivision and the lemmas we have presented before.

Lemma 11 *Let us have two sets $X, Y_1, Y_2 \subseteq P$ such that $Y_1 \subseteq Y_2$. For any point $p \in X$ we have $\mathcal{SVV}(p, X, Y_2) \subseteq \mathcal{SVV}(p, X, Y_2)$.*

Corollary 12 *For any point $p \in P$ and any set $X \subseteq P$ we have $\mathcal{VK}(p) \subseteq \mathcal{SVV}(p, P, X)$.*

A *dominating point for AGP(X, Y)* is a point $p \in P$, where $\nexists q \in X$ such that $\mathcal{V}(p) \cap Y \subset \mathcal{V}(q) \cap Y$ and $p \neq q$. The *dominating guard set for AGP(X, Y)* for

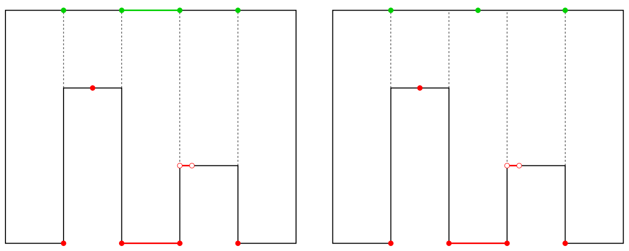


Figure 6: Red indicates the near-minimal witness set W. Green indicates $\mathcal{DG}(P, P, W)$ on the left and $\mathcal{DG}^*(P, P, W)$ on the right.

P is a set of points $D \subseteq X$, such that each element of D is a dominating point and D has all the dominating points in X, denoted as $\mathcal{DG}(P, X, Y)$. We define *minimal dominating guard set for $AGP(X, Y)$* as a set of points $S \subseteq \mathcal{DG}(P, X, Y)$, such that for all distinct elements $p, q \in S$, $\mathcal{V}(p) \cap Y \neq \mathcal{V}(q) \cap Y$ and $\mathcal{VS}(S) = \mathcal{VS}(\mathcal{DG}(P, X, Y))$. We denote a minimal dominating guard set as $\mathcal{DG}^*(P, X, Y)$.

Then we describe another subdivision of P into cells that are atomic in terms of $\mathcal{SVV}(p, X, Y)$. The subdivision is created through an arrangement consisting of two types of line segments:

1. For each vertex v of X and each reflex vertex r in $R(v)$, shoot a ray $\overrightarrow{vr}$ if $\overrightarrow{vr}$ hits Y after passing r until it hits ∂P.

2. For each vertex v of Y and each reflex vertex r in $R(v)$, shoot a ray $\overrightarrow{vr}$ if $\overrightarrow{vr}$ hits X after passing r until it hits ∂P.

3. For each pair of reflex vertices r_1, r_2 that see past each other, shoot rays $\overrightarrow{r_1r_2}$ and $\overrightarrow{r_2r_1}$ if $\overrightarrow{r_1r_2}$ hits X after passing r_2 and $\overrightarrow{r_2r_1}$ hits Y after passing r_1 until they hit ∂P.

We denote the subdivision as $\mathcal{A}(P, X, Y)$. The cells, vertices and edges of $\mathcal{A}(P, X, Y)$ are defined similar to those of $\mathcal{A}(P)$ in Section 3. This time there are at most $O(n^2(m + k)^2)$ vertices, edges and cells of $\mathcal{A}(P, X, Y)$ where m is the number of connected components in Y.

Lemma 13 *A point $p \in X$ is a dominating point for $AGP(X, Y)$ iff $\forall q \in \mathcal{SVV}(p, X, Y)$ we have $\mathcal{SVV}(p, X, Y) = \mathcal{SVV}(q, X, Y)$.*

Lemma 14 *If a point on an edge of $\mathcal{A}(P, X, Y)$ is a dominating point for $AGP(X, Y)$, all the points on that edge are also dominating points for $AGP(X, Y)$. If a point in a cell of $\mathcal{A}(P, X, Y)$ is a dominating point, all the points on that cell are also dominating points, i.e., cells and line segments are atomic in terms of dominating property.*

We use a similar algorithm as the previous section here where we analyze the inclusion relations of $\mathcal{SVV}(p, X, Y)$ instead of $\mathcal{VK}(p)$. We use the strong visibility algorithms described in [11].

5 The iterative refinement scheme

Here we present an iterative heuristic scheme for AGP(P, P). In the pseudocode depicted below, Lines 2 and 3 serves as an initial minimal witness set and minimal dominating guard set so that an equivalent simpler problem, AGP(G, W), is considered. Then we try iteratively reducing the witness set W with respect to the current dominating guard set G and reducing the dominating guard set G with respect to the current witness set W. At each iteration, we make a note of the witness points that can be seen by only one point *ep* in the dominating guard set. Those points in the dominating guard set are essential to the solution, (i.e., any optimum solution that does not include *ep* can be replaced by a solution that has *ep* with the same number of points) and hence they can be added to the guard set and the visible points in W from *ep* can be removed from W. An algorithm for computing Line 2 is presented in [1]. Section 3 and 4 present algorithms for computing Lines 3 and 7 respectively. An extension akin to the extension from Section 3 to 4 of the algorithm can be used for Line 6.

AGP refinement
```
1:  function REFINEMENT($P$)
2:      $W \leftarrow$ MinWitness$(P, P)$
3:      $G \leftarrow$ DominatingGuards$(P, P)$
4:      $EG \leftarrow \emptyset$ // Essential guards
5:      while $G$ or $W$ changing do
6:          $W \leftarrow$ MinWitness$(P, G)$
7:          $G \leftarrow$ DominatingGuards$(P, W)$
8:          $Z \leftarrow \{p \in W || \mathcal{V}(p) \cap G| = 1\}$
9:          $EG \leftarrow EG \cup Z$
10:         $W \leftarrow W \setminus \bigcup_{p \in Z} \mathcal{V}(p)$
11:     end while
12: end function
```

As it stands the above refinement strategy is a heuristic, it would be good to establish a strong upper bound on the number of iterations. An analysis of the complexity of evolving arrangements computed in Lines 6 and 7 will be crucial on establishing a time bound for the iterative refinement scheme.

References

[1] E. S. Ayaz and A. Üngör. Minimal witness sets for art gallery problems. In *European Workshop on Computational Geometry (EuroCG), Book of Abstracts*, pages 195–198, 2016.

[2] A. Bottino and A. Laurentini. A nearly optimal sensor placement algorithm for boundary coverage. *Pattern Recognition*, 41(11):3343–3355, 2008.

[3] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. In *ISAAC '93 Proceedings*, pages 58–67, Berlin, Heidelberg, 1993. Springer.

[4] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, Jan. 1992.

[5] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6(1):9–31, 1991.

[6] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39 – 41, 1975.

[7] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[8] K. Chwa, B. Jo, C. Knauer, E. Moet, R. van Oostrum, and C. Shin. Guarding art galleries by guarding witnesses. *Int. J. Comput. Geometry Appl.*, 16(2-3):205–226, 2006.

[9] P. J. de Rezende, C. C. de Souza, S. Friedrichs, M. Hemmer, A. Kröller, and D. C. Tozoni. Engineering art galleries. *CoRR*, abs/1410.8720, 2014.

[10] S. Fisk. A short proof of chvátal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.

[11] S. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, NY, USA, 2007.

[12] S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718 – 722, 2010.

[13] R. Honsberger. *Mathematical Gems II*. Mathematical Association of America, 1976.

[14] B. Joe and R. B. Simpson. Corrections to lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.

[15] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, Boston, MA, 1972. Springer US.

[16] J. King. Fast vertex guarding for polygons with and without holes. *Computational Geometry*, 46(3):219 – 231, 2013.

[17] A. Laurentini. Guarding the walls of an art gallery. *The Visual Computer*, 15(6):265278, 1999.

[18] D. Lee and A. Lin. Computational complexity of art gallery problems. *Information Theory, IEEE Transactions on*, 32(2):276–282, Mar 1986.

[19] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

[20] D. C. Tozoni, P. de Rezende, and C. de Souza. The quest for optimal solutions for the art gallery problem: A practical iterative algorithm. In *Experimental Algorithms*, LNCS v. 7933, pages 320–336. Springer, 2013.

[21] J. Urrutia. Art gallery and illumination problems. In *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

Minimizing the Solid Angle Sum of Orthogonal Polyhedra and Guarding them with $\frac{\pi}{2}$-Edge Guards

I. Aldana-Galván[*] J.L. Álvarez-Rebollar[†] J.C. Catana-Salazar[*] M. Jiménez-Salinas[*]
E. Solís-Villarreal[*] J. Urrutia[‡]

Abstract

We give a characterization for the orthogonal polyhedron in $\mathbb{R}^3$ that minimizes the sum of its internal solid angles, and prove that its minimum angle sum is $(n-4)\pi$ and their maximum angle sum is $(3n-24)\pi$. We generalize to $\mathbb{R}^3$ the well-known result that in an orthogonal polygon with n vertices, $(n+4)/2$ of them are convex and $(n-4)/2$ of them are reflex. We define a vertex of a polyhedron to be convex on the faces if it is convex or straight in all the faces where it participates, and to be reflex on the faces otherwise. If a polyhedron with n vertices and genus g has k vertices of degree greater than 3 (in its 1-skeleton), we prove that it has $(n+8-8g+3k)/2$ vertices that are convex on the faces and $(n-8+8g-3k)/2$ vertices that are reflex on the faces. Finally, we prove that if the orthogonal polyhedron has k_4 vertices of degree 4, k_6 vertices of degree 6, genus g and h_m holes on its faces, then we can guard it using at most $(11e-k_4-3k_6-12g-24h_m+12)/72$ $\frac{\pi}{2}$-edge guards (i.e., having a visibility angle of $\pi/2$ towards the interior of the polyhedron), improving the bound given by Viglietta et al in [14] for open edge guards.

1 Introduction

In the plane, to measure the interior angle of a polygon at a vertex v, we usually consider a small enough circle centered at v and not containing any other vertices of the polygon, measure the length of the portion of the circle that lies inside the polygon, and then divide it by the radius. In this way, we can have angles that vary between 0 and 2π. It has been well-known since antiquity that the sum of the angles of a triangle is π. Since a simple polygon of n vertices can be partitioned into $n-2$ triangles using diagonals, the sum of the internal angles of a polygon is $(n-2)\pi$. We extend these ideas to polyhedra in $\mathbb{R}^3$.

We measure the interior solid angles of a polyhedron in a vertex v in an analogous way to the plane. We consider a small enough sphere centered at v, measure the area of the portion of the sphere that lies within the polyhedron, and then divide it by the square of the radius. In this way, we have solid angles that vary between 0 and 4π since the area of a unit sphere is 4π.

For summing interior angles in polyhedra we cannot use the same approach that was used for polygons. This approach would consist in tetrahedralizing a polyhedron and summing the solid angles of all the resulting tetrahedra. However, there exist examples of polyhedra that cannot be tetrahedralized; for example, the Schönhardt polyhedron [12]. It is also known that the sum of the solid angles of a tetrahedron can take any value between 0 and 2π [7].

These examples show that in general polyhedra, the sum of their solid angles is not constant and their vertices can have interior angles that are arbitrarily small. However, it is an interesting question to find the minimum and the maximum sums of the internal solid angles of an orthogonal polyhedron. This sum cannot be arbitrarily small because the internal solid angle of each vertex is at least $\pi/2$. We show in this paper that the lower and upper bounds for the sum of angles of an orthogonal polyhedron with n vertices are $(n-4)\pi$ and $(3n-24)\pi$ respectively. We also give the classification of the families of orthogonal polyhedra achieving these bounds.

We consider that a vertex of a polyhedron is convex on the faces if it is a convex or a straight vertex in all the faces where it participates, and it is reflex on the faces otherwise. If a polyhedron with n vertices has k vertices of degree greater than 3 in its *1-skeleton* (i.e., the set of edges and vertices of the polyhedron), we prove that it has $(n+8+3k)/2$ vertices that are convex on the faces and $(n-8-3k)/2$ vertices that are reflex on the faces.

We apply this result to address a variant of the Art Gallery Problem in orthogonal polyhedra. Most of the research on art gallery problems has been focused on polygons on the plane. For example, it is well

[*]Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Ciudad de México, México, ialdana@ciencias.unam.mx, {j.catanas, m.jimenez, solis_e}@uxmcc2.iimas.unam.mx

[†]Posgrado en Ciencias Mateméticas, Universidad Nacional Autónoma de México, Ciudad de México, México, chepomich1306@gmail.com

[‡]Instituto de Mateméticas, Universidad Nacional Autónoma de México, Ciudad de México, México, urrutia@matem.unam.mx

known that every simple polygon with n vertices can be guarded with at most $\lfloor n/3 \rfloor$ vertex guards [4], and for orthogonal polygons $\lfloor n/4 \rfloor$ vertex guards are always sufficient to guard the polygon [8]. Estivill-Castro and Urrutia [6] showed that every orthogonal polygon can be guarded with at most $3(n-1)/8$ *orthogonal floodlights*; that is, vertex guards that have an angle of vision of $\pi/2$. Later in [1] it was proved that $(3n + 4(h-1))/8$ orthogonal floodlights are always sufficient to guard an orthogonal polygon with n vertices and h holes.

For orthogonal polyhedra with e edges in $\mathbb{R}^3$, it was conjectured that $e/12$ edge guards are always sufficient to guard any polyhedron [13]. Benbernou *et al.* [14] showed that every polyhedron can always be guarded by $(11/72)e - g/6 - 1$ *open edge guards* (i.e., excluding their endpoints).

For general polyhedra, Cano *et al.* [3] showed that any polyhedron can always be guarded by $(27/32)e$ edge guards, and if the faces are all triangles the bound improves to $(29/36)e$. For general polyhedra it is conjectured that every simply connected polyhedron can be guarded with $e/6$ edge guards [13].

We say that a $\frac{\pi}{2}$*-edge guard* is a guard located on an edge of the polyhedron, occupying all the edge with an angle of vision of $\frac{\pi}{2}$. An interior point p of the polyhedron is guarded by an edge e if the segment s, described by the shortest distance between p and e, is perpendicular to e, s is contained in the visibility angle of e, and s is completely contained in the interior of the polyhedron.

The variant of the art gallery problem we address is the following: Given an orthogonal polyhedron P in $\mathbb{R}^3$, choose a minimum set of $\frac{\pi}{2}$-edge guards located on the edges of P such that any interior point of P is guarded. We prove that if P has k_4 vertices of degree 4, k_6 vertices of degree 6, genus g and h_m holes on its faces, then we can guard it using at most $(11e - k_4 - 3k_6 - 12g - 24h_m + 12)/72$ $\frac{\pi}{2}$-edge guards.

2 Orthogonal Polyhedra

A *polyhedron* in $\mathbb{R}^3$ is a compact set bounded by a piecewise linear manifold. A *face* of a polyhedron is a maximal planar subset of its boundary whose interior is connected and non-empty. A polyhedron is *orthogonal* if all of its faces are parallel to the xy, xz or yz planes. Faces of a polyhedron can be polygons with holes, and if the polyhedron is orthogonal, then its faces and its holes are also orthogonal. A *vertex* of a polyhedron is a vertex of any of its faces. An *edge* is a minimal positive-length straight line segment shared by two faces and joining two vertices of the polyhedron.

2.1 Vertex Characterization in Orthogonal Polyhedra

Let P be an orthogonal polyhedron in $\mathbb{R}^3$. We classify the vertices of P by its interior solid angles. A vertex x of P is classified as *1-octant* if its interior solid angle is $\pi/2$ (see Figure 1a), and *3-octant* if its interior solid angle is $3\pi/2$ (see Figure 1b). The *4-octant*, *5-octant* and *7-octant* vertices are defined in a similar way, as illustrated in Figures 1c, 1d, 1e and 1f respectively.

In an orthogonal polygon we have three kinds of vertices; convex, reflex and straight. A vertex is convex if it has an interior angle of $\pi/2$, reflex if it has an interior angle of $3\pi/2$ and straight if it has an angle of π.

We say that a vertex is *convex on the faces* if it participates on each of its incident faces as a convex or a straight vertex. Thus the 1-octant, 4-octant, and 7-octant vertices are convex on the faces. We say that a vertex is *reflex on the faces* if it participates as a reflex vertex on exactly one of its incident faces. Thus the 3-octant and 5-octant vertices are reflex on the faces. We will refer to a convex vertex on the faces (resp. reflex vertex on the faces) as a convex vertex (resp. reflex vertex) unless stated otherwise. Since we can have straight vertices on the faces of a polyhedron, we extend our concept of orthogonal polygon in order to allow them to have straight vertices, too.

The *genus* g of a connected orientable surface is the integer representing the maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected [9].

In our main result we use the Euler-Poincaire's formula, which states that for any polyhedron of genus g with f faces, e edges, v vertices and a total of h holes on its faces, the identity $v - e - h + f = 2 - 2g$ holds. A proof of this theorem can be found in [11].

Next, we prove the following theorem:

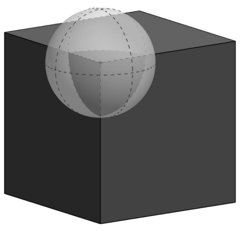
Theorem 1 *Let P be an orthogonal polyhedron in $\mathbb{R}^3$ homeomorphic to the sphere with $n = 2k$ vertices and a connected and 3-regular 1-skeleton. Then P has $(n+8)/2$ convex vertices and $(n-8)/2$ reflex vertices.*

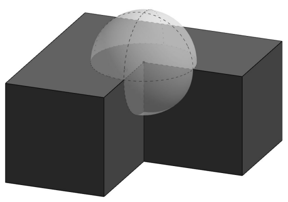
Proof. Since each vertex has degree 3, the number of edges e is $3k$. By Euler's formula, the number of faces f is $k+2$. The number of reflex vertices in an orthogonal polygon is $(n-4)/2$, so the number of reflex vertices on each face of P is $(V_i - 4)/2$, where V_i is the number of vertices on the i^{th} face of P. Then the number of reflex vertices of P is

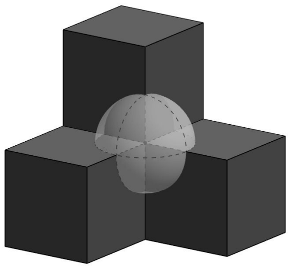
$$r = \sum_{i=1}^{k+2} \frac{V_i - 4}{2}. \qquad (1)$$

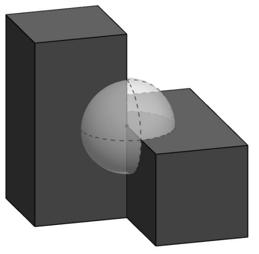
Solving equation (1), we have

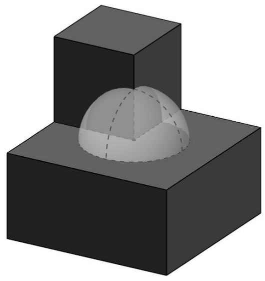
$$2r = \sum_{i=1}^{k+2} V_i - \sum_{i=1}^{k+2} 4.$$


(a) 1-octant vertex


(b) 3-octant vertex


(c) 4-octant vertex


(d) 4-octant vertex


(e) 5-octant vertex

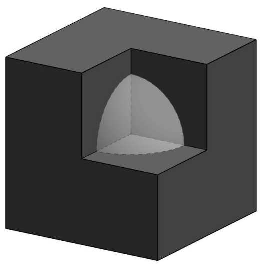

(f) 7-octant vertex

Figure 1: Vertex classification for orthogonal polyhedra

As each vertex belongs to three faces, it is counted three times when adding up the first sum;

$$2r = 6k - 4(k+2)$$
$$r = k - 4.$$

Since $n = 2k$, $r = (n-8)/2$, and since $n = c + r$, $c = (n+8)/2$. $\square$

This result tells us exactly the number of convex and reflex vertices of the family of orthogonal polyhedra that are connected and 3-regular in its 1-skeleton. Next, we eliminate the restriction of a connected 1-skeleton by considering the number of holes on the faces of P and including the genus of the polyhedron. We will also include the 4-octant vertices. Note that these vertices do not have degree 3, but degree 4 or 6. Some of the 4-octant vertices look like straight angles on some faces of the polyhedron.

We introduce two lemmas that will help us to incorporate the 4-octant vertices in the count of convex and reflex vertices of an orthogonal polyhedron.

Lemma 2 *In an orthogonal polygon with n vertices of which s are straight, the number of reflex vertices is $r = (n-s-4)/2$ and the number of convex vertices is $c = (n-s+4)/2$.*

Proof. Since the sum of the internal angles of a simple polygon is $\pi(n-2)$; and the angle of each convex vertex is $\pi/2$, of each reflex vertex $3\pi/2$, and of each straight angle π,

$$\pi(n-2) = \left(\frac{\pi}{2}\right)c + \left(\frac{3\pi}{2}\right)r + (\pi)s.$$

Solving for c and replacing in $n = c + r + s$ yields $n = 2r + s + 4$. Therefore, $r = (n-s-4)/2$ and $c = (n-s+4)/2$. $\square$

If the polygon has holes, we have the next lemma.

Lemma 3 *In an orthogonal polygon P with n vertices, h holes, and a total of s straight vertices, the number of reflex vertices is $(n-s+4h-4)/2$ and the number of convex vertices is $(n-s-4h+4)/2$.*

Proof. Note that a hole is an orthogonal polygon such that its convex vertices are reflex in P, its reflex vertices are convex in P, and its straight vertices are straight in P. Thus, using Lemma 2, we have that if m is the number of vertices in P without the holes, s_m of which are straight, and each hole has n_i vertices, s_i of which are straight, then the number of reflex vertices of P is

$$r = \left(\sum_{i=1}^{h} \frac{n_i - s_i + 4}{2}\right) + \frac{m - s_m - 4}{2} = \frac{n - s + 4h - 4}{2}.$$

Then it follows automatically that the number of convex vertices in P is $(n-s-4h+4)/2$. $\square$

Let k_3 be the vertices of degree 3, k_4 the vertices of degree 4, and k_6 the vertices of degree 6 in the 1-skeleton of a polyhedron.

We are ready to give one of our main results.

Theorem 4 *Let P be an orthogonal polyhedron in $\mathbb{R}^3$ with $n = k_3 + k_4 + k_6$ vertices and arbitrary genus g. Then P has $(n - 3(k_4 + k_6) + 8g - 8)/2$ reflex vertices and $(n + 3(k_4 + k_6) - 8g + 8)/2$ convex vertices.*

Proof. The number of edges e is $3k_3/2 + 2k_4 + 3k_6$. Using the Euler-Poincaré formula, the number of faces f is $k_3/2 + k_4 + 2k_6 + 2 + h - 2g$. By Lemma 3, the number of reflex vertices in P is

$$r = \sum_{i=1}^{f} \frac{V_i - s_i + 4h_i - 4}{2}, \qquad (2)$$

where V_i is the number of vertices and s_i is the number of straight vertices and h_i is the number of holes on the ith face of P.

Solving Equation (2), we have

$$2r = \sum_{i=1}^{f} V_i - \sum_{i=1}^{f} s_i + \sum_{i=1}^{f} 4h_i + \sum_{i=1}^{f} 4.$$

In the first sum we count the total number of vertices: the k_3 vertices are counted three times, the k_4 vertices are counted four times and the k_6 vertices are counted six times. The second sum counts the total number of straight vertices but there are only k_4 vertices and they are counted two times. The third sum gives the total number of holes in P. Then we have

$$2r = k_3 - 2k_4 - 2k_6 - 8 + 8g. \quad (3)$$

Since $n = k_3 + k_4 + k_6$, we obtain

$$r = (n - 3(k_4 + k_6) + 8g - 8)/2.$$

Since $n = c + r$, $c = (n + 3(k_4 + k_6) + 8 - 8g)/2$. □

This generalizes the well known result that the number of convex and reflex vertices of an orthogonal polygon with n vertices are respectively $(n + 4)/2$ and $(n - 4)/2$, see $\mathbb{R}^2$ [10].

2.2 Minimizing the Solid Angle Sum of Orthogonal Polyhedra

Let V_i be the number of i-octant vertices, $i = 1, 3, 4, 5, 7$. The angle sum of an orthogonal polyhedron is

$$S = \frac{\pi}{2}V_1 + \frac{3\pi}{2}V_3 + 2\pi V_4 + \frac{5\pi}{2}V_5 + \frac{7\pi}{2}V_7. \quad (4)$$

Since an orthogonal polyhedron has n vertices,

$$V_1 + V_3 + V_4 + V_5 + V_7 = n. \quad (5)$$

We use the polyhedral version of Gauss-Bonnet's theorem to calculate the curvature of the polyhedron [5]. Observe that the angle deficit for 1-octant and 7-octant vertices is $\pi/2$, the angle deficit for 3-octant and 5-octant vertices is $-\pi/2$ and the angle deficit for 4-octant vertices is $-\pi$. Applying Gauss-Bonnet's theorem, where g is the genus of the polyhedron, we get

$$\frac{\pi}{2}(V_1 + V_7) - \frac{\pi}{2}(V_3 + 2V_4 + V_5) = 4\pi - 4\pi g \quad (6)$$

Multiplying (5) by π and subtracting (6) we obtain:

$$\frac{\pi}{2}V_1 + \frac{3\pi}{2}V_3 + 2\pi V_4 + \frac{3\pi}{2}V_5 + \frac{\pi}{2}V_7 = n\pi - 4\pi + 4\pi g \quad (7)$$

Adding $\pi V_5 + 3\pi V_7$ to both sides of (7) yields:

$$\frac{\pi}{2}V_1 + \frac{3\pi}{2}V_3 + 2\pi V_4 + \frac{5\pi}{2}V_5 + \frac{7\pi}{2}V_7 = \pi n - 4\pi + 4\pi g + \pi V_5 + 3\pi V_7 \quad (8)$$

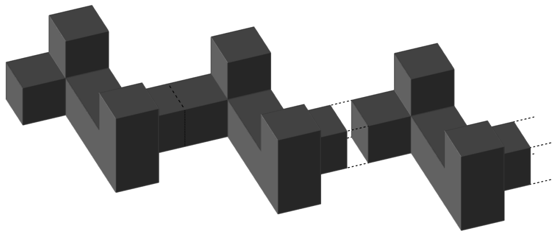


Figure 2: Family of Polyhedra that minimise the solid angle sum.

The left side of (8) corresponds to the angle sum:

$$S = \pi(n - 4 + 4g + V_5 + 3V_7) \quad (9)$$

Thus (9) is minimized when V_5 and V_7 are both equal to zero. The next result follows.

Theorem 5 *The minimum solid angle sum of orthogonal polyhedra is $(n - 4)\pi$ and is achieved by polyhedra having only 1-octant, 3-octant and 4-octant vertices.*

Figure 2 shows an example that achieves the bound of Theorem 5.

The maximum solid angle sum is reached when we maximize the number of V_7 and V_5 vertices in (9). In order to do this, we observe that any orthogonal polyhedra P always has at least eight 1-vertices, and if it is not a box, it has at least eight 1-vertices and four 3-vertices, or it has ten 1-vertices and two 3-vertices. The best case arises when P has exactly eight 1-vertices and four 3-vertices. This can be achieved by *carving out of a box* a polyhedra with $m = n - 8$ vertices that minimizes the sum of its angles, as shown in Figure 3.

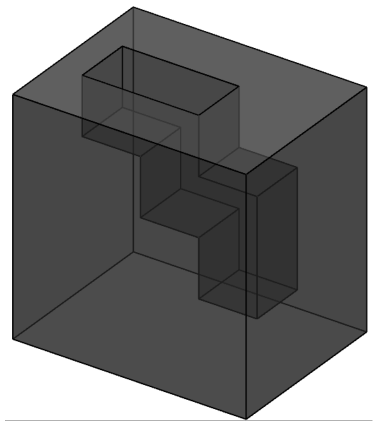


Figure 3: An Orthogonal Polyhedron that maximize its solid angle sum.

Theorem 6 *The maximum solid angle sum of orthogonal polyhedra is $(3n - 24)\pi$.*

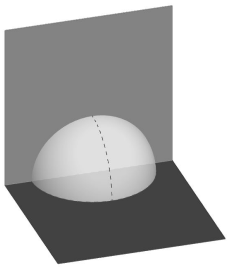
3 Guarding Polyhedra

We say that an α*-edge guard* is a guard located on an edge of a polyhedron, occupying the entire edge with an angle of vision towards the interior of the polyhedron of size α. In this section we will deal with $\alpha = \frac{\pi}{2}$.

An interior point p of a polyhedron is guarded by an α-edge guard e if the segment pr, where r is the closest point to p in e, is perpendicular to e, pr is contained in the α-visibility angle of e, and the interior of pr is contained in the interior of the polyhedron.

We apply the results obtained in the previous section to address the following variation of the Art Gallery Problem: Given an orthogonal polyhedron P in $\mathbb{R}^3$, select a set of $\frac{\pi}{2}$-edge guards located on the edges of P that guards P.

Note that P has two kinds of edges: convex edges that cover an internal solid angle of two octants, see Figure 4a, and reflex edges that cover an internal solid angle of six octants, see Figure 4b.


(a) 2-octant edge

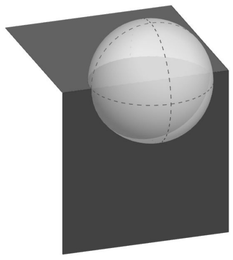

(b) 6-octant edge

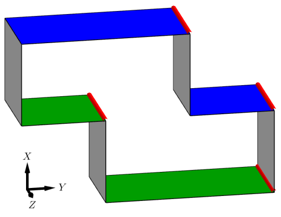
Figure 4: Types of edges in orthogonal polyhedra

It is easy to see that to guard P it is sufficient to place one $\frac{\pi}{2}$-edge guard on each convex edge, and two $\frac{\pi}{2}$-edge guards, in opposite directions, on every reflex edge. In fact, we can also guard P by applying the previous rule only to edges parallel to the $\mathcal{X}$-axis, the $\mathcal{Y}$-axis, or the $\mathcal{Z}$-axis. This follows from the results proved in [2]. For the sake of completeness we describe briefly how to prove this.

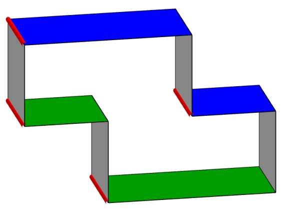
Consider all the faces of P parallel to $\mathcal{XZ}$ and $\mathcal{YZ}$ planes. We call a face of P incident to e, a *top face* f, if for any interior point q of f there is an $\epsilon > 0$ such that any point at distance less than or equal to ϵ from q, and below f belongs to the interior of P. *Right*, *bottom*, and *left* faces are defined in a similar way, see Figure 5.

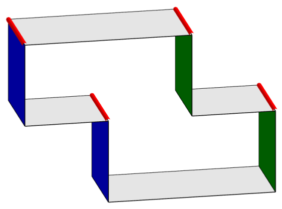
Let e be an edge parallel to the $\mathcal{Z}$ axis. Given a top (bottom) face f, we call an edge of f a *right edge* if there is an $\epsilon > 0$ such that any point at distance less than or equal to ϵ from the mid-point of e, and the left of e belongs to the interior of f. A *left edge* is defined in a similar way. Given a right (left) face f, the *top* and *bottom edges* are defined similarly to the left and right edges, see Figure 5.

We define the placement rules for $\frac{\pi}{2}$-edge guards at the edges of P parallel to the $\mathcal{Z}$ axis, as follows: In the *top-right* rule at each right edge of each top face of P, and at each top edge of each right face of P we place a $\frac{\pi}{2}$-edge guard whose angle of illumination covers the interval of directions $\frac{3\pi}{2}$ to 2π. We define three extra rules, the *top-left* rule, *bottom-right* rule, and *bottom-left* rule in a similar way by rotating our polyhedra 90, 180 and 270 degrees with respect to the $\mathcal{Z}$-axis, and applying the top-left rule to the polyhedron obtained from P after applying these rotations.



(a)


(b)


(c)

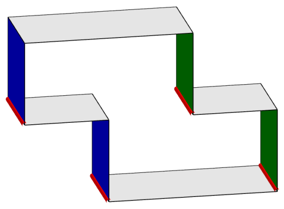

(d)

Figure 5: Figures (a) and (b) show top faces in blue and bottom faces in green. Figures (c) and (d) show left faces in blue and right faces in green. Figures (a), (b), (c) and (d) show right, left, top and bottom edges respectively.

Now we prove the following Lemma:

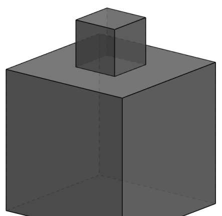
Lemma 7 *Let P be an orthogonal polyhedron with genus g and h holes on its faces. Then P can be guarded by the $\frac{\pi}{2}$-edge guards placed by any of the following rules: top-right, top-left, bottom-right and bottom-left.*

Proof. We prove our result for the top-right rule, the other rules can be proved in a similar fashion. Let p be a point in P and let β be the plane parallel to the $\mathcal{XY}$ plane containing p. Let Q be the intersection of P with β. Q consists of a set of orthogonal polygons contained in β. It is straightforward to see that the top right rule places $\frac{\pi}{2}$-vertex guards as in the top-right illumination rule in [2] which illuminates, and thus guards p. Our result follows. □

Some faces of an orthogonal polyhedron P may have holes in them. When these holes appear, the 1-skeleton of P may become disconnected, for an example see Figure 3. In that example we "carved out" an orthogonal polyhedron H from a box in the middle of one of its faces, call it f. Observe that the k-vertices of H became $8-k$-vertices in P, except for those lying in f, in that case 1-octant vertices of H became 3-octant vertices of P, and 3-octant vertices of H become 1-octant vertices of P, (*i.e.* the convex vertices become reflex and the reflex vertices become convex), see also Figure 6b. Observe that at least four of the vertices of H in f are reflex, and that two of the edges incident to them, are convex, and one is reflex. Thus our guarding rules place only four edge guards on these edges. This will be used in the proof of our next Theorem, as this will allow us to save four edges per each hole in which we carved an orthogonal polyhedron (in that proof we place five edges in the edges of a reflex vertex of degree three).

There is a second case in which the 1-skeleton of P becomes disconnected, and this happens when instead of carving out an orthogonal polyhedron H, we kind of "glue" it in the middle of a face f of P, see Figure 6a. In this case it is easy to see that when we apply the guarding rules to P described above, the points of P in H will be guarded by edges in H, and the edges in $P-H$ can be guarded with edges in the 1-skeleton of $P-H$. This implies that the edges of H in f can be considered as convex edges when applying the guarding rules described above. Thus we save at least four edge guards, one for each reflex edge of H in f.

In both cases we save at least four guards per hole.


(a)

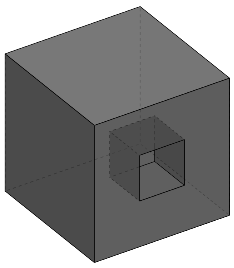

(b)

Figure 6: (a) Two "glued" orthogonal polyhedron. (b) An orthogonal polyhedron carved out of another one.

Theorem 8 *Let P be an orthogonal polyhedron with n vertices, k_4 of them are of degree 4, k_6 of degree 6, e edges, genus g and h_m holes in the faces of P. Then $(11e - k_4 - 3k_6 - 12g - 24h_m + 12)/72$ $\frac{\pi}{2}$-edge guards are always sufficient to guard the interior of P.*

Proof. First we look at the type of vertices of the polyhedron P, and describe the number of convex and reflex edges that each kind of vertex is incident to.

Each 1-octant vertex is incident to three convex edges. Each 3-octant vertex is incident to two convex edges and one reflex edge. Each 4-octant vertex with degree four, is incident to two convex edges and two reflex edges. Each 4-octant vertex with degree six, is incident to three convex edges and three reflex edges. Each 5-octant vertex is incident to one convex edge and two reflex edges. Finally, each 7-octant vertex is incident to three reflex edges.

By the Theorem 4, P has $c = (n+3(k_4+k_6)-8g+8)/2$ convex vertices and $r = (n - 3(k_4 + k_6) + 8g - 8)/2$ reflex vertices. Note that according to our definition, 4-octant vertices, whether they have degree four or six are convex. Then, P has $c = (n+k_4+k_6-8g+8)/2$ convex vertices, k_4 4-octant vertices of degree four, k_6 4-octant vertices of degree six, and $r = (n-3(k_4+k_6)+8g-8)/2$ reflex vertices.

In the worst case every convex vertex is adjacent to three reflex edges, every 4-octant vertex of degree four is adjacent to two reflex edges and two convex edges, every 4-octant vertex of degree six is adjacent to three reflex edges and three convex edges, and every reflex vertex is incident to two reflex edges and one convex edge.

If we place guards on every edge of P then, we have $(6c + 6k_4 + 9k_6 + 5r)/2$ $\frac{\pi}{2}$-edge guards in total. We can divide the number of $\frac{\pi}{2}$-edge guards by three and four, since it is sufficient to choose one of the three axis directions, and we only need to choose the smallest of the four guarding rules used in this direction, then we obtain $(6c + 6k_6 + 9k_6 + 5r)/24$. Substituting c and r in the above equation, we have a total of $(11n + 3k_4 + 9k_6 + 8)/48$ $\frac{\pi}{2}$-edge guards.

As P has h_m holes on its faces, and for each of them we save four edge guards we conclude that the total number of $\frac{\pi}{2}$-edge guards in P is $(11n+3k_4+9k_6-8g-16h_m+8)/48$. If we substitute $n = (2e - k_4 - 3k_6)/3$ in the number of $\frac{\pi}{2}$-edge guards, then we finally obtain that $(11e-k_4-3k_6-12g-24h_m+12)/72$ $\frac{\pi}{2}$-edge guards are always sufficient to guard the interior of P.

□

References

[1] J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia. *Illumination with orthogonal floodlights*, pages 362–371. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[2] J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia. Illumination of orthogonal polygons with orthogonal floodlights. *International Journal of Computational Geometry & Applications*, 8(01):25–38, 1998.

[3] J. Cano, C. D. Tóth, and J. Urrutia. Edge guards for polyhedra in 3-space. In *CCCG*, pages 155–160, 2012.

[4] V. Chvatal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[5] S. L. Devadoss and J. O'Rourke. *Discrete and computational geometry.* Princeton University Press, 2011.

[6] V. Estivill-Castro and J. Urrutia. Optimal floodlight illumination of orthogonal art galleries. In *CCCG*, pages 81–86, 1994.

[7] J. Gaddum. The sums of the dihedral and trihedral angles in a tetrahedron. *The American Mathematical Monthly*, 59(6):370–371, 1952.

[8] J. Kahn, M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983.

[9] J. Munkres. *Topology*. Featured Titles for Topology Series. Prentice Hall, Incorporated, 2000.

[10] J. O'Rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[11] H. Poincaré. Sur la généralisation d'un théoreme d'euler relatif aux polyedres. *Comptes Rendus de Séances de l'Academie des Sciences*, 117:144, 1893.

[12] E. Schönhardt. Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98(1):309–312, 1928.

[13] J. Urrutia. Art gallery and illumination problems. In J.-R. S. Urrutia, editor, *Handbook of Computational Geometry*, pages 973 – 1027. North-Holland, Amsterdam, 2000.

[14] G. Viglietta, N. Benbernou, E. D. Demaine, M. L. Demaine, A. Kurdia, J. O'Rourke, G. T. Toussaint, and J. Urrutia. Edge-guarding orthogonal polyhedra. In *CCCG*, 2011.

The Planar Slope Number

Udo Hoffmann*

Abstract

The *planar slope number* of a planar graph G is defined as the minimum number of slopes that is required for a crossing-free straight-line drawing of G. We show that determining the planar slope number is hard in the *existential theory of the reals*. We point out consequences for drawings that minimize the planar slope number.

1 Introduction

The *slope number* of a non-degenerate straight-line drawing D of a graph G is defined to be the number of distinct slopes that is used to draw the edges of G in D. The minimum slope number of all straight-line drawings of G is the slope number of G. Similarly, the *planar slope number* of a planar graph G is the minimum slope number over all planar straight-line drawings of G.

In this paper, we consider the computational complexity of computing the planar slope number. In Section 2, we show that determining the *planar slope number* of a graph is hard in the *existential theory of the reals*, i.e., as hard as deciding the solvability of a polynomial inequality system over the reals. Furthermore, it is complete in the *existential theory of the rationals* (and thus possibly undecidable) to decide whether a planar graph has a drawing on the grid that minimizes the planar slope number. However, for each fixed k, deciding whether the (planar) slope number is at most k is in NP. A consequence of this result is that deciding if the planar slope number of a bounded degree graph is at most k is in NP. Afterwards, in Section 3, we point out consequences for drawings that minimize the slope number: There are planar graphs such that each drawing that minimizes the planar slope number requires irrational coordinates for the vertices and slopes of the edges. In Section 4 we point out open problems in connection to the slope number.

1.1 Background

The slope number of a graph has mainly been studied for the relation between the maximum degree of a graph and the slope number: A simple lower bound for the slope number of a graph G is $\lceil \Delta(G)/2 \rceil$, where $\Delta(G)$ denotes the maximum degree of G, since at most two edges of the same slope are incident to one vertex. The main work in this area deals with the question, whether the slope number of a graph is also bounded from above by a function in the maximum degree. This was answered negatively [BMW06, PP06, DSW07] by examples of families of graphs of maximum degree 5 with arbitrarily large slope number. In contrast, Keszegh, Pach, and Pálvölgyi have shown that the planar slope number is bounded by an exponential function in the maximum degree [KPP13]. For partial planar 3-trees [JJK$^+$13] this bound has been improved to a polynomial upper bound of $O(\Delta^5)$ and for outerplanar graphs [KMW14] to a linear upper bound of $\Delta - 1$ (for $\Delta \geq 4$) for outerplanar drawings.

From the computational point of view, it is known to be NP-complete to decide whether a graph has slope number 2 [FHH$^+$93], and it is NP-complete to decide whether a planar graph has planar slope number 2 [GT01]. Thus both problems, computing the slope number and the planar slope number, are NP-hard. We characterize the planar slope number problem as hard in the *existential theory of the reals*.

The *existential theory of the reals* ($\exists\mathbb{R}$) is a complexity class defined by the following complete problem: Given a quantifier-free formula $F(x_1, \ldots, x_n)$ that contains logic connections of polynomial equalities and inequalities in the variables $x_1, \ldots, x_n$ with integer coefficients, is there an assignment of real values to the variables such that the formula is satisfied? This problem can be reduced to deciding the solvability of a polynomial inequality system over the reals. Starting with Mnëv's universality theorem [Mnë88] many geometric problems have been shown to be hard in $\exists\mathbb{R}$. Mnëv's universality theorem states that for each *semialgebraic set* V there exists an *order type* (or by duality, a line arrangement) whose *realization space* is *stably equivalent* to V. From a computational point of view it is important that the realization space is empty if and only if V is empty.

Some $\exists\mathbb{R}$-complete problems include pseudoline stretchability [Mnë88, Sho91], recognition of segment intersection graphs [KM94], realizability of planar graphs and linkages [Sch12], realizing abstract 4-polytopes [RGZ95], point visibility graph recognition [CH15], and many more, see [Car15] for an

*Université libre de Bruxelles (ULB) `hoffmann.odu@googlemail.com` The results in this paper are part of the authors dissertation at TU Berlin, supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structure' (GRK 1408) overview.

The *existential theory of the rationals* ($\exists\mathbb{Q}$) is defined similarly to $\exists\mathbb{R}$, but restricted to rational solutions. When asking for geometric representations on the integer grid for $\exists\mathbb{R}$-hard problems it turns out that $\exists\mathbb{Q}$ is the right complexity class because of scaling arguments. It is an open problem if $\exists\mathbb{Q}$ is decidable. The class $\exists\mathbb{R}$ is decidable in PSPACE [Can88], while the *existential theory of the integers* is undecidable by the negative answer to *Hilbert's tenth problem* due to Matiyasevich [Mat70].

We want to point out that the problem of deciding if the (planar) slope number is at most k is contained in $\exists\mathbb{R}$. This can be easily shown by encoding the coordinates as well as the k allowed slopes in variables. The same holds for drawings on the grid and $\exists\mathbb{Q}$. In the following we only mention hardness results because we consider optimization problems and not decision problems.

Our hardness proofs are based on the problem of *pseudoline stretchability*: Given a collection of x-monotone curves that extend infinitely in positive and negative x-direction such that any two curves intersect pairwise exactly once, is there a homeomorphism of the plane that maps the curves onto lines? Or in other words, is there a collection of lines with the same intersection pattern as in the collection of curves. We call the collection of curves a *pseudoline arrangement*; it is *stretchable* if the described homeomorphism exists. We call the stretched collection of curves a *line arrangement*. A (pseudo)line arrangement is *simple* if no three lines/curves intersect in a common point. The stretchability of simple pseudoline arrangements is also hard in $\exists\mathbb{R}$. For a good overview on the $\exists\mathbb{R}$-reduction for the stretchability problem we refer to [Mat14]. We point out that stretchability of non-simple pseudoline arrangements with rational coordinates is complete in $\exists\mathbb{Q}$ [Stu87], while simple line arrangements can always be perturbed onto rational coordinates.

2 Computational complexity

In this section we consider the computational complexity of the planar slope number.

2.1 $\exists\mathbb{R}$-hardness

In this subsection, we show that computing the planar slope number is $\exists\mathbb{R}$-hard. The general idea is to construct an (almost) 3-connected planar graph G_L that contains the edges and vertices of a pseudoline arrangement L. Consequently, the pseudoline arrangement L can be found in each planar drawing of G_L by drawing the pseudolines on the corresponding edges. The degree of each vertex of the arrangement in G_L is equal to the even maximum degree Δ. Any two consecutive edges of one pseudoline are opposite edges at some vertex of the arrangement. By the following proposition, the existence a drawing of G_L with slope number $\Delta/2$ implies that L is stretchable.

Proposition 1 *Let G be a planar graph with even maximum degree Δ, and let D be a planar straight-line drawing of G with slope number $\Delta/2$. Each pair of opposite edges of a vertex of degree Δ in D has the same slope.*

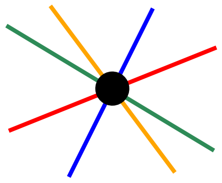


Figure 1: Opposite edges of a degree 8 vertex in drawing of slope number 4 have the same slope.

Proof. Let v be a vertex of degree Δ. Each slope of the drawing D appears exactly twice among the edges that are incident to v. The edges with the same slope are opposite in D. □

For the proof of the following theorem we proceed to construct such a graph G_L from a pseudoline arrangement L that has a drawing D with $\Delta/2$ slopes if and only if L is stretchable.

Theorem 2 *Deciding if the planar slope number of a planar graph with even maximum degree Δ is $\Delta/2$ is complete in $\exists\mathbb{R}$.*

Proof. We prove the theorem by reducing the stretchability of a pseudoline arrangement to the problem of deciding whether the planar slope number of a graph is

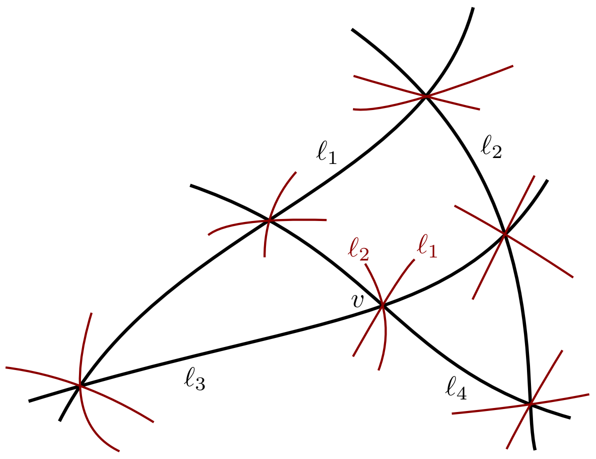


Figure 2: Adding a star (brown) on each vertex of the black arrangement.

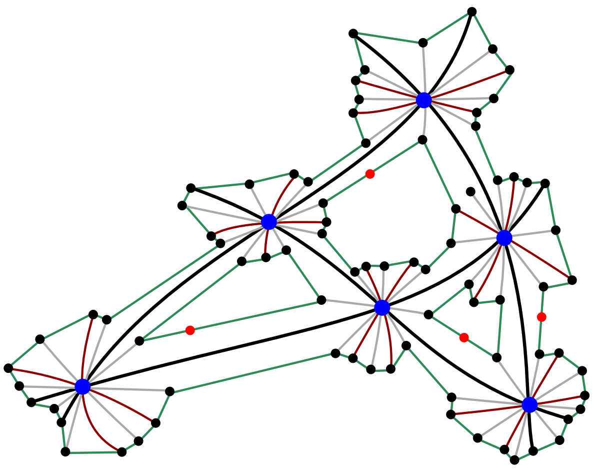


Figure 3: Constructing the graph G_L from the stars, including the grey intermediate slopes and the red subdivision vertices.

$\Delta/2$. Therefore, let L be an arrangement of n pseudolines.

We note that we can determine the order of slopes of the lines in a stretched realization of L from the pseudoline arrangement, namely as the order in which the lines appear while traversing the adjacent unbounded faces. We use this observation to speak about the *slope of a pseudoline* and apply it in the following construction (see Figure 2): In the pseudoline arrangement L we draw a *star of pseudolines* on each vertex of the arrangement, i.e., for each pseudoline ℓ that is not incident to a vertex v of the arrangement we draw a pseudosegment that indicates which faces around v a pseudoline of the slope of ℓ through v intersects.

Now, we cut the pseudolines in the unbounded faces and define a planar graph by placing a vertex on each endpoint of a pseudosegment. We can already observe that the *embedding* we constructed can be drawn straight-linn with n slopes if and only if the arrangement is stretchable. We modify this construction to obtain a 3-connected graph as shown in Figure 3: In addition to the pseudosegment of each slope of a pseudoline of L we add a star of intermediate slopes, one slope between each two consecutive slopes of pseudolines. We connect the leaf vertices of the stars in each face (including the one unbounded face) such that they form a cycle. We pick one edge per face cycle that connects two leaves of different stars and subdivide these edges. We call this planar graph G_L. After contracting the subdivision vertices the graph G_L is 3-connected. Thus Proposition 1 implies that the opposite edges, which originate from one pseudoline lie on one line, and thus a drawing with n slopes gives a realization of the line arrangement by drawing the lines along the edges.

So it remains to show that there exists a drawing D with slope number n if L is stretchable. Therefore, we consider a realization R of L as a line arrangement. We draw the vertices and edges of G_L on the corresponding edges and vertices of R. We choose the intermediate slopes and place a star containing all the $2n$ slopes on each vertex of the arrangement. The cycle in an inner face f is realized by drawing a polygon with sides parallel to the boundary of f such that on each corner of a polygon lies a vertex of the cycle. This can be done in the following way. We draw the polygon in the face clockwise, starting from one point close to the boundary on the counterclockwise first ray of one vertex v_1. We draw the first edges of the cycle following parallel to the boundary of the face in clockwise order and place a vertex of the cycle on the intersection point of the segments of the star and the polygon. When we reach the counterclockwise last ray of the vertex v_2 we continue with a line parallel to the second boundary edge. We follow this procedure until we reach the counterclockwise last ray of the last vertex. To close the last edge of the polygon we have drawn in the face we use the subdivision vertex as shown in Figure 5. The cycle surrounding the outer face can be drawn with the same method as indicated in Figure 4. This concludes the proof that there exists a drawing of G_L with n slopes if and only if L is stretchable. □

2.2 Drawings on the grid.

Lemma 3 *The graph G_L constructed in the proof of Theorem 2 has a drawing with slope number $\Delta/2$ with rational coordinates if and only if L has a realization with rational coordinates.*

Proof. In the proof of Theorem 2 we have shown that we can realize L on a subset of vertices and edges of a slope minimizing drawing. Thus L has a rational realization if and only if there is a drawing of G_L with

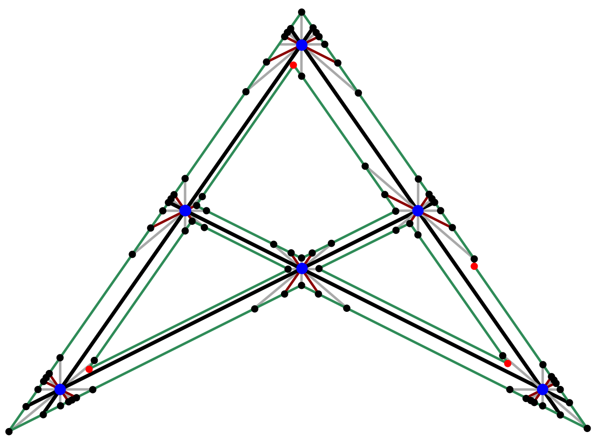


Figure 4: A drawing with slope number 8 of G_L of the arrangement L in Figure 2.

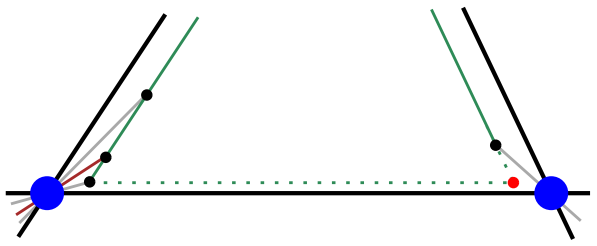


Figure 5: Using the subdivision vertex (red) to close the face cycle with few slopes.

slope number $\Delta/2$, where the vertices and edges of the arrangement graph lie on rational coordinates. Thus, to conclude this proof, we only have to show that we can draw the cycles in the inner faces on rational coordinates. This is simply done by choosing rational intermediate slopes and a rational coordinate for the first vertex we draw in the polygon. Then all vertices of the cycle lie on the intersection points of rational lines, and thus have rational coordinate. $\square$

From the lemma above and the fact that deciding the realizability of a non-simple line arrangement is complete in $\exists\mathbb{Q}$ [Stu87] we obtain the following theorem.

Theorem 4 *Deciding whether a planar graph G has a drawing on the grid with slope number $\Delta/2$ is complete in $\exists\mathbb{Q}$.*

2.3 Bounded slope number and bounded degree.

In contrast to the previous results, we show that we can decide for a fixed k in non-deterministic polynomial time whether a planar graph can be drawn with at most k slopes.

Theorem 5 *For each fixed k the decision problem whether a graph G has planar slope number or slope number at most k is in NP.*

Proof. We give a proof based on the NP membership of the recognition of segment intersection graphs that can be represented by at most k slopes for the segments [KM94][Theorem 1.1.ii.c] by Kratochvíl and Matoušek. They show that deciding the realizability of an arrangement of segments using at most k slopes can be decided in polynomial time.

To show that deciding whether the planar slope number is at most k is in NP we guess the embedding of the graph and which of the edges use the same slope. With this information we can use the result of Kratochvíl and Matoušek to decide in polynomial time whether the arrangement of edges can be realized using at most k slopes.

For the non-planar slope number we guess the complete arrangement of edges and the partition of the edges into common slopes. $\square$

Let G_Δ be the set of planar graphs with maximum degree at most Δ. We use the theorem above and the fact that deciding if a graph in G_Δ has slope number at most k is in NP.

Theorem 6 *Deciding whether a planar graph $G \in G_\Delta$ has planar slope number at most k is in NP.*

Proof. By [KPP13] there exists a function $f(\Delta)$, such that each graph in G_Δ has planar slope number at most $f(\Delta)$. To decide whether the graph G has planar slope number k we return true if $k \geq f(\Delta)$. Otherwise, if $k < f(\Delta)$, we can decide if the planar slope number is at most k by Theorem 5, since k is bounded by the constant $f(\Delta)$. $\square$

3 Consequences of the hardness

In this section we point out consequences of $\exists\mathbb{R}$-hardness of computing the planar slope number and $\exists\mathbb{Q}$-hardness of deciding whether there is a drawing on the grid that achieves this slope number.

The fact that there are non-simple line arrangements that are known to have irrational coordinates in each representation [Grü03] directly translates into the following result.

Corollary 7 *There are planar graphs such that each planar drawing that minimizes the slope number has at least one vertex with an irrational coordinate.*

Even if a line arrangement is stretchable with rational coordinates, there are arrangements that require an doubly exponential representation size [GPS90]. By the observation that the graph G_L has $|L|^3$ vertices, we obtain the following corollary.

Corollary 8 *For each $n \in \mathbb{N}$, there is a planar graph G_n on n vertices such that each planar drawing of G_n on a grid that minimizes the slope number requires a grid of size $2^{2^{\Omega(\sqrt[3]{|V(G)|})}}$.*

We want to point out that giving a reasonable (a.k.a. computable) upper bound on the grid size in the corollary above, is strongly connected with the decidability of $\exists\mathbb{Q}$.

Theorem 9 *Assume $\exists\mathbb{Q}$ is undecidable. Then there is no computable function f such that every graph G, that has a slope number minimizing drawing on the grid, can be drawn with this slope number on a grid of size $f(|V(G)|) \times f(|V(G)|)$.*

Proof. Assume the function f exists. Then compute $f(|V(G)|)$ and try each combination of coordinates of vertices of G on a grid of size $f(|V(G)|) \times f(|V(G)|)$ and check whether a straight-line drawing with those vertex coordinates gives a drawing of the given slope number. This procedure finds a drawing on the grid that minimizes the planar slope number by the assumption that f gives an upper bound on the grid size of such a drawing. Thus we have just given an algorithm that finds such a drawing of minimum planar slope number on the grid if it exists, which is contradiction to the assumed undecidability of $\exists\mathbb{Q}$ by Theorem 4. $\square$

4 Conclusion and open problems.

We have settled the computational complexity of determining the planar slope number. It is an open problem whether the (non-planar) slope number is also $\exists\mathbb{R}$-hard. A further open problem is to give a better bound on the function $f(\Delta)$ that bounds the planar slope number of graphs of degree Δ. The bound on $f(\Delta)$ in [KPP13] is exponential in Δ and uses the non-constructive proof for a touching disc representation, where the radii of touching discs are bounded by a constant factor by [MP94]. They give the idea of a non-deterministic algorithm to obtain a planar drawing using $f(\Delta)$ slopes. It is open whether the bound can be improved and can be turned in a polynomial algorithm.

References

[BMW06] János Barát, Jiri Matoušek, and David R. Wood. Bounded-degree graphs have arbitrarily large geometric thickness. *European Journal of Combinatorics*, 13:R3, 2006.

[Can88] John Canny. Some algebraic and geometric computations in PSPACE. In *STOC*, pages 460–467. ACM, 1988.

[Car15] Jean Cardinal. Computational geometry column 62. *ACM SIGACT News*, 46:69–78, 2015.

[CH15] Jean Cardinal and Udo Hoffmann. Recognition and complexity of point visibility graphs. In *SoCG*, volume 34 of *LIPIcs*, pages 171–185. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015.

[DSW07] Vida Dujmović, Matthew Suderman, and David R. Wood. Graph drawings with few slopes. *Computational Geometry*, 38:181–193, 2007.

[FHH$^+$93] Michael Formann, Torben Hagerup, James Haralambides, Michael Kaufmann, Frank T. Leighton, Antonios Symvonis, Emo Welzl, and Gerhard J. Woeginger. Drawing graphs in the plane with high resolution. *SIAM Journal on Computing*, 22:1035–1052, 1993.

[GPS90] Jacob E. Goodman, Richard Pollack, and Bernd Sturmfels. The intrinsic spread of a configuration in $\mathbb{R}^d$. *Journal of the American Mathematical Society*, 3:639–651, 1990.

[Grü03] Branko Grünbaum. *Convex Polytopes*, volume 221 of *Graduate Texts in Mathematics*. Springer-Verlag, 2003.

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31:601–625, 2001.

[JJK$^+$13] Vít Jelínek, Eva Jelínková, Jan Kratochvíl, Bernard Lidický, Marek Tesař, and Tomáš Vyskočil. The planar slope number of planar partial 3-trees of bounded degree. *Graphs and Combinatorics*, 29:981–1005, 2013.

[KM94] Jan Kratochvíl and Jirí Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62:289–315, 1994.

[KMW14] Kolja Knauer, Piotr Micek, and Bartosz Walczak. Outerplanar graph drawings with few slopes. *Computational Geometry*, 47:614–624, 2014.

[KPP13] Balázs Keszegh, János Pach, and Dömötör Pálvölgyi. Drawing planar graphs of bounded degree with few slopes. *SIAM Journal on Discrete Mathematics*, 27:1171–1183, 2013.

[Mat70] Yuri V. Matiyasevich. Enumerable sets are diophantine. *Doklady Akademii Nauk SSSR*, 191:279–282, 1970.

[Mat14] Jiří Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$. *arXiv:1406.2636*, 2014.

[Mnë88] Nicolai E. Mnëv. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In *Topology and Geometry – Rohlin Seminar*, LNM, pages 527–543. Springer, 1988.

[MP94] Seth Malitz and Achilleas Papakostas. On the angular resolution of planar graphs. *SIAM Journal on Discrete Mathematics*, 7:172–183, 1994.

[PP06] János Pach and Dömötör Pálvölgyi. Bounded-degree graphs can have arbitrarily large slope numbers. *European Journal of Combinatorics*, 13:N1, 2006.

[RGZ95] Jürgen Richter-Gebert and Günter M. Ziegler. Realization spaces of 4-polytopes are universal. *Bulletin of the American Mathematical Society*, 32:403–412, 1995.

[Sch12] Marcus Schaefer. Realizability of graphs and linkages. In János Pach, editor, *Thirty Essays on Geometric Graph Theory*. Springer, 2012.

[Sho91] Peter W. Shor. Stretchability of pseudolines is NP-hard. *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, 4:531–554, 1991.

[Stu87] Bernd Sturmfels. On the decidability of diophantine problems in combinatorial geometry. *Bulletin of the American Mathematical Society*, 17:121–124, 1987.

Epsilon-covering: a greedy optimal algorithm for simple shapes

Tuong-Bach Nguyen [*] Isabelle Sivignon [*]

Abstract

Unions of balls are widely used shape representations. Given a shape, computing a union of balls that is both accurate in some sense and of small cardinality is thus a challenging problem. In this work, accuracy is ensured by imposing that the union of balls, called *covering*, is included in the shape and covers a parameterized core set (namely the erosion) of the shape. For a family of simple shapes, we propose a polynomial-time greedy algorithm that computes a covering of minimum cardinality for a given shape.

1 Introduction

Unions of balls are common shape representations, useful for instance to describe molecules in biochemistry [4], to quickly detect collisions [3] between shapes or to derive higher-level representations. The ubiquity of unions of balls is largely due to the existence of provably good conversion algorithms that allow us to derive them from various representations such as point clouds and polygonal meshes [6]. However, the union of balls output by the conversion process provides only an approximation of the original shape.

In a previous work [2], we introduced a novel way, called ε-covering, of controlling the geometric error between a given input shape and a selected union of balls. The idea is to impose that the union of balls covers a core set of the shape *and* does not cross over an outer set. This problem falls in the family of geometric set cover problems, where the goal is to minimize the number of balls. We proved that, in the general case, computing an ε-covering of minimum cardinality is an NP-complete problem. Other approaches, related to the maximum k-cover problem, aim at maximising the coverage for a fixed number of balls [4], but the problem is also NP-complete.

In this work, we consider the family of input shapes that are themselves 2D unions of balls with a tree-like structure. An ε-covering of such a shape is a simplified union of balls. We present a polynomial-time algorithm that computes an ε-covering of minimum cardinality for this specific family. To do so, we rely on the medial axis structure of unions of balls, and show how to continuously sweep its pencils in order to get a correct and optimal result.

[*]Univ. Grenoble Alpes, GIPSA-Lab, F-38000 Grenoble, France CNRS, GIPSA-Lab, F-38000 Grenoble, France
tuong-bach.nguyen@gipsa-lab.grenoble-inp.fr
isabelle.sivignon@gipsa-lab.grenoble-inp.fr

2 Statement of the result

In this paper, $\mathbb{R}^2$ is endowed with the Euclidean distance. For any point c and real $r > 0$, we denote by $b(c,r)$ the closed ball of center c and radius r. For any subset $S \subseteq \mathbb{R}^2$, we respectively denote its closure, interior, complement, boundary, and medial axis by $\overline{S}$, $\mathring{S}$, S^c, ∂S, and $\mathrm{MA}\,(S)$. Let $\varepsilon > 0$ be a real number. The erosion of S (by ε) is $S^{\ominus\varepsilon} = \{y \mid b(y,\varepsilon) \subseteq S\}$. For any collection of balls $\mathscr{B}$, we write $\bigcup \mathscr{B} = \cup_{b\in\mathscr{B}} b$.

Definition 1 An (inner) **ε-covering** of S is a collection of balls $\mathscr{B}$ such that $S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B} \subseteq S$.

For given S and ε, there exist many ε-coverings of S, with different cardinalities. We say that an ε-covering is **optimal** if it achieves minimum cardinality. In general, finding such an optimal ε-covering for S is an NP-complete problem [2], but we focus here on simple shapes S and prove the following result:

Theorem 1 There is a polynomial-time algorithm to compute optimal ε-coverings for finite unions of balls whose medial axis is cycle-free.

In Section 3, we present some results on the structure of unions of balls, before describing the principle of our algorithm in Section 4. Section 5 expands on some practical considerations required for the algorithm, and Section 6 is dedicated to proving that it indeed achieves the claimed result.

3 Union of balls

3.1 Medial axis and pencils

In order to specify our algorithm, we must elaborate on the structure of unions of balls, in particular that of their medial axis. Recall that a ball $b \subseteq S$ is a medial ball if its boundary ∂b intersects ∂S at least twice. The medial axis $\mathrm{MA}\,(S)$ is the collection of the centers of these medial balls. Owing to the structure theorem of the medial axis of a union of balls [1], we know that for a finite union of balls S, $\mathrm{MA}\,(S)$ is a finite collection of line segments. We also know that each of these line segments coincides with a pencil of balls [4] in the following sense.

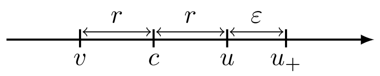
Borrowing the terminology used in [7], an elliptic pencil can be characterized by two points $u, v \in \mathbb{R}^2$: it is the family of all balls whose boundary goes through u and v. The collection of their centers forms a line. In this paper, we only manipulate elliptic pencil segments, that are subsets of elliptic pencils whose collection of centers forms a segment instead of a line. From here on, we will not consider any proper line pencil, and thus we refer to these elliptic pencil segments simply as pencils. As such, the pencils we consider always have two endpoint balls b_1 and b_2. We denote the pencil they generate by $[b_1 b_2]$. A basic property of a pencil is that the domain it covers, that is the collection of all points covered by some ball of the pencil, is the union of b_1 and b_2, $\bigcup [b_1 b_2] = b_1 \cup b_2$.

Linking back to the previous remark, $\mathrm{MA}(S)$ is a collection of pencils. Indeed, for each segment of the medial axis, there are two points $u, v \in \partial S$, such that any medial ball centered at a point of that segment contains both u and v in its boundary. Thus, each segment of $\mathrm{MA}(S)$ coincides with a pencil. Hence a union of balls can always be interpreted as a union of pencil domains (see Figure 1 for an illustration).

The medial axis of a closed shape and its erosion always satisfy the below inclusion.

Proposition 2 $\mathrm{MA}(S^{\ominus\varepsilon}) \subseteq \mathrm{MA}(S)$

Proof. Consider $c \in \mathrm{MA}(S^{\ominus\varepsilon})$ and $b = b(c, r) \subseteq S^{\ominus\varepsilon}$ medial in $S^{\ominus\varepsilon}$. Let $b_+ = b(c, r+\varepsilon)$. We prove that b_+ is medial in S which implies that $c \in \mathrm{MA}(S)$.
First, note that $b_+ = \cup_{y \in b} b(y, \varepsilon) \subseteq S$, hence if ∂b_+ intersects ∂S at least twice, it is medial in S. By definition of b, there are at least two points $u \neq v$ in the intersection of ∂b and $\partial(S^{\ominus\varepsilon})$. Since $u, v \in \partial(S^{\ominus\varepsilon})$, there are $u_+, v_+ \in \partial S$ such that $\|u - u_+\| = \varepsilon = \|v - v_+\|$. By triangular inequality we have $\|c - u_+\| \leq \|c - u\| + \|u - u_+\| = r + \varepsilon$, hence $u_+ \in b_+$. Since $u_+ \in \partial S$, necessarily $u_+ \notin \mathring{b}$ and we have $\|c - u_+\| \geq r + \varepsilon$. Therefore $\|c - u_+\| = r + \varepsilon$, thus $u_+ \in \partial b_+ \cap \partial S$. Also, we have equality in a triangular equality, hence c, u and u_+ are aligned. Likewise, $v_+ \in \partial b_+ \cap \partial S$, and c, v and v_+ are also aligned. Thus, if $u_+ \neq v_+$, then b_+ is medial in S. By contradiction, assume that we have $u_+ = v_+$. Then, c, u, v and u_+ must be aligned. Because $u \neq v$, we have the below situation.



Hence $\|v - v_+\| = \|v - u_+\| = 2r + \varepsilon = \varepsilon$. This implies $r = 0$ and $u = v$, which is impossible. Therefore, $u_+ \neq v_+$ and b_+ is medial in S. □

Thus for unions of balls, $\mathrm{MA}(S^{\ominus\varepsilon})$ splits $\mathrm{MA}(S)$ into two finite collections of line segments: segments that are part of both $\mathrm{MA}(S)$ and $\mathrm{MA}(S^{\ominus\varepsilon})$, and segments that are exclusively part of $\mathrm{MA}(S)$. We respectively refer to them as **eroded** and **non-eroded** pencils.

3.2 Partial ordering on medial balls

$\mathrm{MA}(S)$ being a collection of segments, it can be viewed as the embedding of a graph in $\mathbb{R}^2$. By assumption on the class of shapes considered, $\mathrm{MA}(S)$ is cycle-free, hence it is a forest. Since we can process each tree of the forest independently, we assume without loss of generality that $\mathrm{MA}(S)$ is a tree. By picking any point x of $\mathrm{MA}(S)$ as a root, we obtain an orientation of $\mathrm{MA}(S)$ which induces a partial order on $\mathrm{MA}(S)$. Indeed, we simply have to orient all the edges of $\mathrm{MA}(S)$ from the leaves to the root x. We denote by T the resulting oriented tree. The structure represented by T is at times called anti-arborescence or in-tree, and can also be viewed as a directed acyclic graph with a unique sink. For any $y, z \in \mathrm{MA}(S)$, we say that y is T-smaller than or equal to z, and note $y \leq_T z$, if z belongs to the unique path from y to the root x of T. We also use the usual order symbols and notions such as being T-larger or equal to, $\geq_T$, or also being strictly T-smaller, $<_T$.

Note that this T-order is valid for all points of $\mathrm{MA}(S)$, and not simply vertices of T. Because points of $\mathrm{MA}(S)$ are centers of medial balls of S and $S^{\ominus\varepsilon}$, this T-order extends to medial balls. Specifically, we can T-compare two medial balls of either S or $S^{\ominus\varepsilon}$, but also a medial ball of S with a medial ball of $S^{\ominus\varepsilon}$.

Since T only induces a partial order, we say that two balls that cannot be ordered by T are ***T*-unrelated**. An additional useful notion is that of T-maximal ball for a collection: given a collection of balls $\mathscr{B}$, $b \in \mathscr{B}$ is ***T*-maximal** in $\mathscr{B}$ if for all $b' \in \mathscr{B}$, either $b' \leq_T b$ or b and b' are T-unrelated. Likewise, b is ***T*-minimal** in $\mathscr{B}$ if for all $b' \in \mathscr{B}$, either $b' \geq_T b$ or b and b' are T-unrelated. Finally, we extend the notion of degree for any point $c \in \mathrm{MA}(S)$. By convention, if c is not a vertex of T but an inner edge point, we say that c has degree 2. We denote the degree by $\deg(c)$.

4 Algorithm

4.1 Principle

For our proposed algorithm, the partial ordering we introduced allows the definition of clear start and end points, as well as a measure of progress. Indeed, let $b_0 = b(c_0, r_0)$ be a medial ball of S. Its center c_0 splits $\mathrm{MA}(S)$ into $\deg(c_0)$ connected components. We denote these components by $\mathrm{branch}(c_0, i)$, $1 \leq i \leq \deg(c_0)$. For our purpose, we want to express the domain covered by balls centered at points of these components of $\mathrm{MA}(S)$, but not covered by b_0. First we define the collection of related medial balls, $\mathscr{C}(b_0, i) = \{b(c, r) \text{ medial in } S \mid c \in \mathrm{branch}(c_0, i)\}$. Then the domain of each $\mathscr{C}(b_0, i)$ is $C(b_0, i) = \bigcup \mathscr{C}(b_0, i) \setminus b_0$. With these notations, b_0 also splits S into the different $C(b_0, i)$'s, and $S \setminus b_0 = \cup_{i=1}^{\deg(c_0)} C(b_0, i)$. Unless c_0 is the

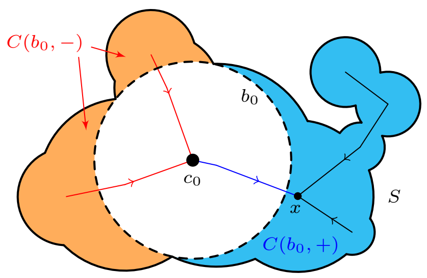


Figure 1: T-large and T-small components of a medial ball. x is the root of T. Red segments are points T-smaller than c_0, blue ones are points T-larger, black ones are T-unrelated.

root of T, one of these domains corresponds to balls T-larger than or T-unrelated to b_0. The other $\deg(c_0)-1$ domains corresponds to balls T-smaller than b_0. To promote clarity, we refer to the former domain simply as the ***T*-large component**, and denote it by $\boldsymbol{C(b_0,+)}$ (see Figure 1). As for the later domains, we refer to their union as the ***T*-small component** and note $\boldsymbol{C(b_0,-)}$. Hence, we have $S \setminus b_0 = C(b_0,+) \cup C(b_0,-)$. From the definition, we also deduce the following:

Proposition 3 *If $b_1 \leq_T b_2$, then $C(b_1,-) \subseteq C(b_2,-)$ and $C(b_1,+) \supseteq C(b_2,+)$.*

Now assume we want to traverse and sweep S with a medial ball, starting from a leaf of T, toward its root. When we reach a medial ball b_0, then at that moment, $C(b_0,-) \cup b_0$ corresponds to the domain of S that was swept by our medial ball, and $C(b_0,+)$ to the domain of S that was not swept by it. Our approach is based on this particular decomposition of S. We want to use a greedy approach to iteratively compute an ε-covering of the T-small component $C(b_0,-)$. Because T may have several leaves, it is necessary to extend the above definitions of T-small and T-large components to collections $\mathscr{B}$ of medial balls, while preserving the interpretation that $C(\mathscr{B},-) \cup (\bigcup \mathscr{B})$ is the domain of S already processed, and $C(\mathscr{B},+)$ is the domain of S that has not been processed yet. The domain already processed for a collection of balls should thus be the union of the domains already processed by some $b \in \mathscr{B}$. Hence the T-small component of $\mathscr{B}$ is $C(\mathscr{B},-) = \cup_{b\in\mathscr{B}} C(b,-)$. Likewise, the domain that still needs to be processed for $\mathscr{B}$ should be the intersection, over all $b \in \mathscr{B}$, of the domains to be processed for b. Hence the T-large component of $\mathscr{B}$ is $C(\mathscr{B},+) = \cap_{b\in\mathscr{B}} C(b,+)$. Owing to Proposition 3, these definitions emphasize the importance of the T-maximal balls of $\mathscr{B}$. Let $\boldsymbol{T\text{-}\max(\mathscr{B})}$ be the collection of these T-maximal balls. Then $C(\mathscr{B},+) = C(T\text{-}\max(\mathscr{B}),+)$ and likewise $C(\mathscr{B},-) = C(T\text{-}\max(\mathscr{B}),-)$.

To formalize the procedure presented above, we require two more definitions.

Definition 2 Let $\mathscr{B}$ be a collection of medial balls in S. We say that $\mathscr{B}$ is a ***T*-small *ε*-covering** of S if it covers $S^{\ominus\varepsilon}$ in its T-small component $C(\mathscr{B},-)$, that is if $C(\mathscr{B},-) \cap S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B}$.

Note that every ε-covering is also a T-small ε-covering of S. As such, we employ the term partial T-small ε-covering if we need to distinguish from complete ε-coverings.

Definition 3 Let $\mathscr{B}$ be a partial T-small ε-covering of S, and b_0 be medial in S. We say that b_0 is a **candidate** ball with respect to $\mathscr{B}$, if $\mathscr{B}_0 = \mathscr{B} \cup \{b_0\}$ is also a T-small ε-covering of S, and $S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B}_0 \subsetneq S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B}$.

The strict inclusion $S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B}_0 \subsetneq S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B}$ ensures that $\mathscr{B}_0$ is closer to being a complete ε-covering than $\mathscr{B}$. Hence, for any partial T-small ε-covering, iteratively adding a candidate to the collection ensure that at some point we will obtain a complete ε-covering. Because any partial T-small ε-covering always have an infinity of candidates, we elect to add only T-maximal candidates, that is T-maximal among the collection of candidates.

4.2 Specification

Our algorithm is based on a loop over the collection of all eroded and non-eroded pencils of $\mathrm{MA}(S)$ in a topological order. Since $\mathrm{MA}(S^{\ominus\varepsilon}) \subseteq \mathrm{MA}(S)$, we can simultaneously sweep S and $S^{\ominus\varepsilon}$. With our partial ordering on $\mathrm{MA}(S)$, a topological order of its vertices $(v_1,\ldots,v_{n+1})$ is such that for $i \leq j$, then either $v_i \leq_T v_j$, or v_i and v_j are T-unrelated. Besides the root, each vertex is incident to exactly one pencil composed of T-larger points of $\mathrm{MA}(S)$, hence any topological ordering of vertices induces an ordering of pencils $([v_1\widehat{v}_1],\ldots,[v_n\widehat{v}_n])$, where $\widehat{v}_i \in \{v_1,\ldots,v_n\}$. This ordering of pencils thus satisfies that for $i<j$, then either $v_i <_T \widehat{v}_i \leq_T v_j <_T \widehat{v}_j$, or $\widehat{v}_i$ and v_j are T-unrelated.

As we loop over the pencils, we maintain a collection of medial balls $\mathscr{B}$ which is a T-small ε-covering, while looking for T-maximal candidates to add to said collection. When we process a pencil, we compute a collection of constraints to pass on to the next incident pencil. A constraint is a point or circular arc that any ball T-larger than the pencil (that is T-larger than any ball of that pencil) must contain in order to be a candidate for $\mathscr{B}$. Hence, if no T-maximal candidate is found in the currently processed pencil, the set of constraints it will pass on to its incident T-larger pencil is the collection of all constraints it itself inherited from incident T-smaller pencils, plus new constraints specific to the current pencil. Then, we can compute the T-maximal ball that contains all these constraints. If it is not the T-large endpoint of the pencil, we call it **critical**. We claim that for well chosen constraints, critical balls are T-maximal candidates. The overall approach is summed up in Algorithm 1.

Algorithm 1 Greedy ε-covering

```
Input: A finite union of balls $S$
Output: An $\varepsilon$-covering $\mathscr{B}$ of $S$
 1: Compute a topological ordering of $\mathrm{MA}(S)$
 2: $\mathscr{B} \leftarrow \varnothing$
 3: Loop over all pencils in topological order
 4:     Retrieve incident constraints
 5:     Search for a critical ball in the pencil
 6:     If a critical ball $b$ is found then
 7:         $\mathscr{B} \leftarrow \mathscr{B} \cup \{b\}$
 8:     end If
 9:     Compute the constraints to pass on
10: end Loop
11: Return $\mathscr{B}$
```

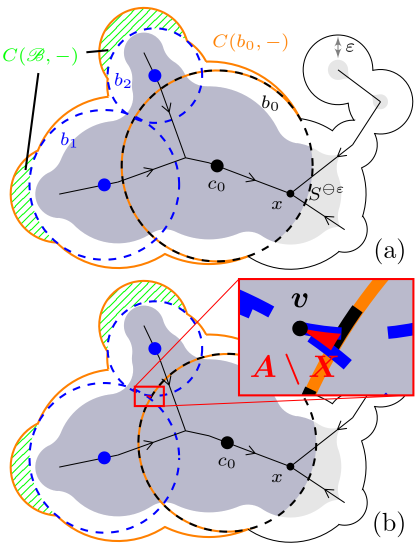


Figure 2: A is the dark shaded area, with $\mathscr{B} = \{b_1, b_2\}$ and $X = b_0 \cup b_1 \cup b_2$. In (a) b_0 is a candidate to $\mathscr{B}$, but in (b) vertex v does not satisfy condition (ii) and $A \setminus X$ (in red) is non empty.

All that remains is to explicit constraints which guarantee that critical balls are indeed T-maximal candidates. Consider a candidate b_0 to the collection $\mathscr{B}$. We can ignore balls which are T-unrelated to b_0, thus let $\mathscr{B}' = \{b \in \mathscr{B},\ b \leq_T b_0\}$. By definition, $\mathscr{B}'_0 = \mathscr{B}' \cup \{b_0\}$ is a T-small ε-covering. The domain of $S^{\ominus\varepsilon}$ covered by $C(\mathscr{B}'_0, -) \cup (\bigcup \mathscr{B}'_0)$ but not by $C(\mathscr{B}', -)$ is contained in $(\bigcup \mathscr{B}'_0) \setminus C(\mathscr{B}', -)$. Let that former domain be

$$\begin{aligned} A &= \left(\left(C(\mathscr{B}'_0, -) \cup \left(\bigcup \mathscr{B}'_0\right)\right) \setminus C\left(\mathscr{B}', -\right)\right) \cap S^{\ominus\varepsilon} \\ &= ((C(b_0, -) \cup b_0) \setminus C(\mathscr{B}', -)) \cap S^{\ominus\varepsilon} \end{aligned}$$

and the latter be

$$X = \left(\bigcup \mathscr{B}'_0\right) \setminus C(\mathscr{B}', -) = b_0 \cup \left(\bigcup T\text{-max}\left(\mathscr{B}'\right)\right).$$

See Figure 2a. A and X both vary depending on $\mathscr{B}$ and b_0, and in light of the previous remark, b_0 is a candidate to $\mathscr{B}$ if and only if $A \subseteq X$. Explicitly ensuring and verifying that we indeed have the inclusion $A \subseteq X$ is non trivial. Instead, we claim that it is sufficient to ensure the two conditions:

(i) $\partial A \subseteq X$,

(ii) $\forall$ vertex $v \in \partial X$, $\exists N_v$ an open neighbourhood of v, such that $N_v \cap A \subseteq X$.

Note that both ∂A and ∂X are finite collections of circular arcs, whose intersections we call vertices of the boundary. Since these boundaries have a finite combinatorial structure, (i) and (ii) are more readily verifiable and can be enforced. Also, as illustrated in Figure 2b, condition (i) by itself is insufficient to ensure that b_0 is a valid candidate. We take as constraints for Algorithm 1 any arc of ∂A not in $\bigcup \mathscr{B}'$, as well as any vertex of $\partial\left(\bigcup T\text{-max}\left(\mathscr{B}'\right)\right)$ whose neighbourhood in A is not fully contained in $\bigcup \mathscr{B}'$. This ensures that critical balls can be computed as per Section 5 and fulfill conditions (i) and (ii). Both conditions are necessary to have the inclusion $A \subseteq X$. We prove in Section 6 that they are also sufficient.

5 Computation of critical balls

As stated previously, each pencil inherits a collection of constraints that are either a singleton point or a circular arc. Because of Proposition 4, which will be stated and explained in detail later, when we sweep a pencil from an endpoint to the other, a point that exited the sweep ball will never re-enter it, and a point that entered the sweep ball will never exit it. Thus, to each constraint corresponds a **single-constraint critical** ball b_{crit}: any ball strictly T-larger than b_{crit} cannot fully contain the constraint, hence cannot be a candidate, while any ball T-smaller than or equal to b_{crit} will always contain the constraint and may be a candidate. Therefore, the critical ball for the overall collection of constraints is the T-maximal ball that is T-smaller than all single-constraint critical balls, that is the unique T-minimal ball amongst all single-constraint critical balls. From here on, we omit the qualifier "single-constraint". Also, because we parameterize balls of a pencil by an interpolation value λ, we call critical λ an interpolation value which corresponds to a critical ball.

The next sections present how to compute a critical ball given a single constraint. Section 5.1 presents a very useful property of pencils and how to handle point constraints, while Section 5.2 deals with arc constraints.

5.1 Point inclusion

Given a ball $b = b(c, r)$ and a point y, there are many ways to test whether y belongs to b or not. By definition, we can compare the distance from y to c, to the radius of b. Here, we rely on the **power** of y with respect to ball b, which by definition is $\mathbf{pow(y,b)} = \|y-c\|^2 - r^2$. Hence, y belongs to ball b if and only if $\text{pow}(y,b) \le 0$.

Consider now a pencil of balls $[b_1 b_2]$, with $b_i = b(c_i, r_i)$. For $\lambda \in [0,1]$, we denote by $b_\lambda = b(c_\lambda, r_\lambda)$ the ball of pencil $[b_1 b_2]$ that is centered at $c_\lambda = \lambda c_1 + (1-\lambda)c_2$. We argue that then, for any point y, the power of y with respect to b_λ is the linear interpolation of its power with respect to b_1 and b_2.

Proposition 4

$$\text{pow}(y, b_\lambda) = \lambda\,\text{pow}(y, b_1) + (1-\lambda)\,\text{pow}(y, b_2)$$

The power of a point with respect to balls of the pencil is thus linear in λ and may only change sign once. This justifies our earlier claim that when sweeping a pencil, a point can exit or enter the sweep ball only once.

Proof. By definition, all the balls of a pencil $[b_1 b_2]$ share exactly two points $\{u, v\} = \partial b_1 \cap \partial b_2$ on their boundary. We have $r_\lambda^2 = \|u - c_\lambda\|^2$. Hence:

$$\begin{aligned}\text{pow}(y, b_\lambda) &= \|y-c_\lambda\|^2 - \|u-c_\lambda\|^2 \\ &= \|y\|^2 - 2\langle c_\lambda, y\rangle + \|c_\lambda\|^2 \\ &\quad - \left(\|u\|^2 - 2\langle c_\lambda, u\rangle + \|c_\lambda\|^2\right) \\ &= 2\langle c_\lambda, u-y\rangle + \|y\|^2 - \|u\|^2\end{aligned}$$

Note that this identity also holds for b_1 and b_2 since they coincide with b_λ, for $\lambda \in \{0,1\}$.

$$\begin{aligned}\text{pow}(y, b_\lambda) &= 2\langle \lambda c_1 + (1-\lambda)c_2, u-y\rangle + \|y\|^2 - \|u\|^2 \\ &= \lambda\left(2\langle c_1, u-y\rangle + \|y\|^2 - \|u\|^2\right) \\ &\quad + (1-\lambda)\left(2\langle c_2, u-y\rangle + \|y\|^2 - \|u\|^2\right) \\ &= \lambda\,\text{pow}(y, b_1) + (1-\lambda)\,\text{pow}(y, b_2)\end{aligned}$$

□

This proof actually extends to all balls of the complete line pencil. From there, given two endpoint balls of a segment pencil and a single constraint point, we can easily compute the value λ_{crit} for which $b_{\lambda_{\text{crit}}}$ will be critical. From the design of Algorithm 1, and assuming that $b_1 \le_T b_2$, we will always have $\lambda_{\text{crit}} \ge 0$. If we happen to have $\lambda_{\text{crit}} > 1$, then the segment pencil does not contain any critical ball for that constraint point, since all balls of the pencil contain that constraint point.

5.2 Arc inclusion

For arc constraints, we use the same approach of computing a critical value for λ. If all of these critical λ's are strictly larger than 1, then the pencil does not contain any T-maximal candidate. If any are less than or equal to 1, then the minimum value yields a T-maximal candidate. All that remains is thus being able to compute such a critical λ for arc constraints. To do so, we first need to compute this critical value for a ball.

5.2.1 Critical λ for balls

Consider a constraint ball $b = b(c, r)$. We want to compute the critical values of λ for which b is fully contained in b_λ. Hence:

$$\begin{aligned}b \subseteq b_\lambda &\iff r_\lambda \ge \|c - c_\lambda\| + r \\ &\iff r_\lambda^2 \ge \|c-c_\lambda\|^2 + r^2 + 2r\|c-c_\lambda\| \\ &\iff -\left(\|c-c_\lambda\|^2 - r_\lambda^2\right) - r^2 \ge 2r\|c-c_\lambda\| \\ &\iff -\text{pow}(c, b_\lambda) - r^2 \ge 2r\|c-c_\lambda\| \\ &\iff \left(-\text{pow}(c,b_\lambda) - r^2\right)^2 \ge 4r^2\|c-c_\lambda\|^2 \\ &\qquad\quad \& \quad -\text{pow}(c, b_\lambda) - r^2 \ge 0\end{aligned}$$

Since $\text{pow}(c_0, b_\lambda)$ is linear in λ as per Proposition 4, we introduce two constants A and B such that $A\lambda + B = \text{pow}(c, b_\lambda) + r^2$ to simplify notations. We have

$$\begin{aligned}A &= \text{pow}(c, b_1) - \text{pow}(c, b_2) \\ B &= \text{pow}(c, b_2) + r^2.\end{aligned}$$

We focus on the first inequality, $(A\lambda + B)^2 \ge 4r^2\|c - c_\lambda\|^2$. For the right hand side, the factor $\|c - c_\lambda\|^2$ can be developed as follows

$$\begin{aligned}&\|c_\lambda - c\|^2 \\ &= \|\lambda(c_1 - c_2) + c_2 - c\|^2 \\ &= \lambda^2\|c_1 - c_2\|^2 + 2\lambda\langle c_1 - c_2, c_2 - c\rangle + \|c_2 - c\|^2\end{aligned}$$

and thus, it is quadratic in λ. Therefore

$$b \subseteq b_\lambda \iff C\lambda^2 + D\lambda + E \ge 0 \quad \& \quad A\lambda + B \le 0$$

where

$$\begin{aligned}C &= A^2 - 4r^2\|c_1 - c_2\|^2 \\ D &= 2AB - 8r^2\langle c_1 - c_2, c_2 - c\rangle \\ E &= B^2 - 4r^2\|c_2 - c\|^2\end{aligned}$$

A, B, C, D and E are constant with respect to λ and thus the inclusion test can be reduced to a sign analysis of polynomials of degree 2 and 1. From the roots of these polynomials, we can thus derive the critical values of λ.

5.2.2 Critical λ for arcs

In order to compute the critical λ value for an arc constraint e, we first require the critical value for the ball b supporting that arc, that is the ball such that $e \subseteq \partial b$. Let λ' be the critical value for that supporting ball.

With λ', we can then compute the tangency point y between b and $b_{\lambda'}$. This particular point, in addition with the endpoints of arc e, are sufficient to compute the critical value of e. Indeed, for $\lambda \leq \lambda'$, the whole ball b is contained in b_λ, hence we also have $e \subseteq b_\lambda$. For $\lambda > \lambda'$, two cases arise. Either $y \in e$ or $y \notin e$. In the former, we have $y \in e \setminus b_\lambda$ hence the critical value for the arc e is In the former, we have $y \in e$ and $y \notin b_\lambda$ hence the critical value for the arc e is λ' itself. In the latter, note that b_λ splits ∂b into two connected components, one which is inside b_λ, and the other outside. By Proposition 4, the outside component will always contain y. For $\lambda > \lambda_{\text{crit}}$, any point of e not in b_λ must be path-connected in ∂b to y. Therefore, some endpoint of e also belongs to the outside connected component. It immediately follows that if both endpoints of e actually belong to b_λ, then the whole arc e is also contained in b_λ. Therefore, the critical value for e is in this case equal to the smallest critical value of its endpoints.

6 Correctness of the algorithm

6.1 Convergence to an ε-covering

Lemma 5 *Let b_0 and $\mathscr{B}$ such that b_0 fulfills both conditions (i) and (ii). Then b_0 is a candidate for $\mathscr{B}$.*

Proof. Let A and X be defined from b_0 and $\mathscr{B}$. Consider $H = A \setminus X = A \cap X^{\mathsf{c}}$. By contradiction assume that $H \neq \varnothing$. First, notice that $\partial H \subseteq \partial X$. Indeed, $\partial H = \partial(A \cap X^{\mathsf{c}}) \subseteq \left(\partial A \cap \overline{X^{\mathsf{c}}}\right) \cup \left(\partial\left(X^{\mathsf{c}}\right) \cap \overline{A}\right)$. By condition (i), $\partial A \subseteq X$ and we have $\partial A \cap \overline{X^{\mathsf{c}}} \subseteq \partial X$. Also, $\partial\left(X^{\mathsf{c}}\right) = \partial X$. Hence $\partial H \subseteq \partial X$. Let H_i be a connected component of H. Since $\partial H_i \subseteq \partial H$, hence $\partial H_i \subseteq \partial X$. We have $H_i \subseteq X^{\mathsf{c}}$ connected, with $\partial H_i \subseteq \partial X$. Thus, H_i is actually a connected component of X^{c}. H_i being bounded, it is commonly called a hole of X. As a hole of X, any vertex of ∂H_i is also a vertex of ∂X. Let v be a vertex of ∂H_i. For all open neighbourhood N_v of v, we have $N_v \cap H \neq \varnothing$. Since $H = A \cap X^{\mathsf{c}}$, we deduce $\varnothing \neq N_v \cap A \nsubseteq X$. This contradicts (ii) and is impossible. Therefore, $H = \varnothing$, $A \subseteq X$, and b_0 is indeed a candidate for $\mathscr{B}$. □

From the above lemma, Algorithm 1 indeed finds candidates and eventually converges to an ε-covering. We now show that it converges in polynomial time.

Lemma 6 *Algorithm 1 converges to an ε-covering in $\mathcal{O}\left(|\mathrm{MA}\,(S)|^2\right)$.*

Proof. Let $n = |\mathrm{MA}\,(S)|$. First, we show that Algorithm 1 outputs a collection $\mathscr{B}_{\text{algo}}$ with size at most $2n$, and then that the complexity is at most quadratic.

For any partial T-small ε-covering $\mathscr{B}$, let $b \in T\text{-max}\,(\mathscr{B})$. There is a unique pencil incident to b that contains balls T-larger than b. Let $b_0 >_T b$ be the T-large endpoint of that pencil. We show that b_0 is always a candidate to $\mathscr{B}$. Though b_0 may not be a T-maximal candidate, this still implies that there can be at most two medial balls from the same pencil in $\mathscr{B}_{\text{algo}}$.
Let $\mathscr{B}_0 = \mathscr{B} \cup \{b_0\}$. Because b and b_0 belong to the same pencil, $C(\mathscr{B}_0, -)$ and $C(\mathscr{B}, -)$ only differ in the domain of S covered by the pencil $[bb_0]$. This implies the equalities $C(\mathscr{B}_0, -) \setminus C(\mathscr{B}, -) = C(b_0, -) \setminus C(b, -) = b \setminus b_0$. Hence, we obtain

$$A = ((b \setminus b_0) \cup b_0) \cap S^{\ominus\varepsilon} \subseteq b \cup b_0,$$

thus b_0 is a candidate for $\mathscr{B}$. Therefore, $|\mathscr{B}_{\text{algo}}| \leq 2n$.

We now analyse the complexity. Computing a topological ordering [5] is linear in n. Enforcing a single constraint takes constant time (see Section 5), hence the time expanded to search for a critical ball depends on the number of constraints. This number cannot exceed the combinatorial complexity of the boundaries of $S^{\ominus\varepsilon}$ and $\bigcup \mathscr{B}_{\text{algo}}$. The former is linear in n. As for the latter, it is linear in the size of $\mathscr{B}_{\text{algo}}$, which is itself linear in n. Thus, Algorithm 1 is quadratic in n. □

6.2 Convergence to an optimal solution

In order to prove that our algorithm reaches an optimal, we rely on several intermediate results. We introduce a sequence of three lemmas, the last of which we reformulate, through two corollaries, in terms of candidate ball to a partial T-small ε-covering. Then, we finally prove Proposition 12.

Lemma 7 *Consider a finite union of balls S, and a ball b such that $b \nsubseteq S$. Let $S' = S \cup b$. Then, $\mathring{S}' = \mathring{S} \cup \mathring{b}$.*

Lemma 8 *Consider a finite union of balls S such that $\mathrm{MA}\,(S)$ is a tree. Then $\overline{S^{\mathsf{c}}}$ is path-connected.*

Proof. Let $\mathscr{S}$ be the collection of medial balls in S which are centered at a vertex of $\mathrm{MA}\,(S)$. $\mathscr{S}$ has finite cardinality and we have $\bigcup \mathscr{S} = S$. We proceed by induction on $n = |\mathscr{S}|$, and henceforth use the notation $S_n = \bigcup \mathscr{S}_n$ for collection of balls with cardinality n. For $n = 1$, S_1 is only a ball, the property is verified. Now consider a collection $\mathscr{S}_{n+1}$, and let $b \in \mathscr{S}_{n+1}$ such that b is centered on a leaf of $\mathrm{MA}\,(S_{n+1})$. Let $\mathscr{S}_n = \mathscr{S}_{n+1} \setminus \{b\}$ and $S_n = \bigcup \mathscr{S}_n$.

Using Lemma 7, we deduce that $\overline{S^{\mathsf{c}}_{n+1}} = \overline{S^{\mathsf{c}}_n} \cap \overline{b^{\mathsf{c}}}$. Consider $y, z \in \overline{S^{\mathsf{c}}_{n+1}}$. By induction assumption, we know that $\overline{S^{\mathsf{c}}_n}$ is path-connected, hence there is a path $\gamma \subseteq \overline{S^{\mathsf{c}}_n}$ connecting y and z. We build from γ another path $\gamma' \subseteq \overline{S^{\mathsf{c}}_{n+1}}$ that connects y and z. If $\gamma \subseteq \overline{b^{\mathsf{c}}}$ we already have $\gamma \subseteq \overline{S^{\mathsf{c}}_{n+1}}$. Otherwise let $\pi_y, \pi_z \in \gamma \cap \partial b$ such that γ does not meet b between y and π_y, and likewise between z and π_z. Because $\pi_y, \pi_z \in \overline{S^{\mathsf{c}}_n}$, they cannot be in $\mathring{S}_n$. Let $e = \partial b \setminus \mathring{S}_n$. e is a path-connected circular arc, and is the contribution of ∂b to ∂S_{n+1}. We have $\pi_y, \pi_z \in e$. Since e is path-connected and contained in $\overline{S^{\mathsf{c}}_{n+1}}$, we can complete γ' by following e to go from π_y to π_z, showing that $\overline{S^{\mathsf{c}}_{n+1}}$ is path-connected. $\square$

Lemma 9 *Consider b_0 a medial ball of S. $C(b_0,+)$ and $C(b_0,-)$ are interior disjoint.*

Proof. By contradiction, assume that $\mathring{C}(b_0,+)$ and $\mathring{C}(b_0,-)$ are not disjoint. From there, we exhibit two paths $\gamma \subseteq \mathring{S}$ and $\delta \subseteq \overline{S^{\mathsf{c}}}$ with non empty intersection, which is impossible.

Let $y \in \mathring{C}(b_0,+) \cap \mathring{C}(b_0,-)$. Without loss of generality, we assume that $y \notin \mathrm{MA}\,(S)$. Necessarily, there are two medial balls of S, b_+ and b_-, such that:

$$\begin{aligned} b_+ &\subseteq C(b_0,+) \cup b_0 \\ b_- &\subseteq C(b_0,-) \cup b_0 \\ y &\in \left(\mathring{b}_+ \cap \mathring{b}_-\right) \setminus b_0. \end{aligned}$$

Let c_+ and c_- be the respective centers of b_+ and b_-. We have segment $[c_+y] \subseteq \mathring{S}$ and likewise segment $[c_-y] \subseteq \mathring{S}$. There is also a path in $\mathrm{MA}\,(S) \subseteq \mathring{S}$ connecting c_+ and c_-. Hence there exists a Jordan curve $\gamma \subseteq \mathrm{MA}\,(S) \cup [c_+y] \cup [c_-y] \subseteq \mathring{S}$. By the Jordan-Brouwer separation theorem, γ has a well defined interior and exterior. Consider now $e = \partial b_0 \cap \overline{C(b_0,+)}$. It is a path-connected circular arc. Its two endpoints are vertices of ∂S, hence we have $\partial e \subseteq \overline{S^{\mathsf{c}}}$. Moreover, one of those endpoints lies in the interior of γ, while the other lies in the exterior of γ. See Figure 3 for a schematic representation. However by Lemma 8, $\overline{S^{\mathsf{c}}}$ is path-connected. Therefore, let $\delta \subseteq \overline{S^{\mathsf{c}}}$ be a path connecting the endpoints of e. Since the interior and exterior of γ are separated, necessarily $\varnothing \neq \gamma \cap \delta \subseteq \mathring{S} \cap \overline{S^{\mathsf{c}}} = \varnothing$, hence the contradiction. $\square$

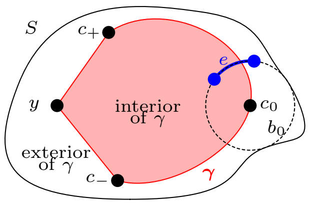


Figure 3: Schematic representation for Lemma 9.

Though Lemmas 7 and 8 are quite remote from the result we want to prove, in essence Lemma 9 implies that whatever medial ball we chose in $C(b_0,+)$, it cannot contribute to cover $S^{\ominus\varepsilon}$ in $C(b_0,-)$. That is why we can process each of these components separately in a greedy way and still achieve a global optimal solution. Formally, we rely on the following corollaries.

Corollary 10 *Let b_0 be a medial ball in S. Then we have the three identities:*

$$\begin{aligned} \left(C(b_0,-) \cup b_0\right)^{\mathsf{c}} \cap S^{\ominus\varepsilon} &= C(b_0,+) \cap S^{\ominus\varepsilon} \\ \left(C(b_0,+) \cup b_0\right)^{\mathsf{c}} \cap S^{\ominus\varepsilon} &= C(b_0,-) \cap S^{\ominus\varepsilon} \\ \left(C(b_0,+) \cup C(b_0,-)\right)^{\mathsf{c}} \cap S^{\ominus\varepsilon} &= b_0 \cap S^{\ominus\varepsilon} \end{aligned}$$

Corollary 10 simply states that when restricted to $S^{\ominus\varepsilon}$, the complement of $C(b_0,+)$, $C(b_0,-)$, and b_0, is the union of the other two subsets.

Proof. We only prove the first equality. We have $S^{\ominus\varepsilon} \subseteq \mathring{S}$. Additionally by Lemma 9, $\mathring{S}$ is the disjoint union of $C(b_0,-) \cap \mathring{S}$, $b_0 \cap \mathring{S}$, and $C(b_0,+) \cap \mathring{S}$. Hence,

$$\begin{aligned} S^{\ominus\varepsilon} \cap \left(C(b_0,-) \cup b_0\right)^{\mathsf{c}} &= S^{\ominus\varepsilon} \cap \mathring{S} \cap \left(C(b_0,-) \cup b_0\right)^{\mathsf{c}} \\ &= S^{\ominus\varepsilon} \cap \mathring{S} \cap C(b_0,+) \\ &= S^{\ominus\varepsilon} \cap C(b_0,+). \end{aligned}$$

$\square$

Corollary 11 *Consider an ε-covering $\mathscr{B}$. Let $\mathscr{B}_- \subsetneq \mathscr{B}$ be a partial T-small ε-covering, and let b_0 be any T-maximal candidate to $\mathscr{B}_-$. Then $\mathscr{B} \setminus \mathscr{B}_-$ contains a candidate to $\mathscr{B}_-$ that is T-smaller than or equal to b_0.*

Proof. Let $\mathscr{B}_+ = \mathscr{B} \setminus \mathscr{B}_-$. First we prove that $\mathscr{B}_+$ always contains candidates to $\mathscr{B}_-$, and then that one of these candidates is T-smaller than or equal to b_0.

By contradiction, assume that $\mathscr{B}_+$ is void of candidate to $\mathscr{B}_-$. Consider $b' \in \mathscr{B}_+$, T-minimal in $\mathscr{B}_+$. By T-minimality of b', for all $b \in \mathscr{B}_+$, $b \subseteq b' \cup C(b',+)$. Thus $\bigcup \mathscr{B}_+ \subseteq b' \cup C(b',+)$. By Corollary 10, we deduce that $S^{\ominus\varepsilon} \cap C(b',-) \subseteq S^{\ominus\varepsilon} \cap \left(\bigcup \mathscr{B}_+\right)^{\mathsf{c}} = S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B}_+$. By assumption, b' cannot be a candidate to $\mathscr{B}_-$, hence by Definitions 2 and 3 $(S^{\ominus\varepsilon} \cap C(b',-)) \setminus (b' \cup (\bigcup \mathscr{B}_-))$ is non empty. With the previous inclusion, we get the following development.

$$\begin{aligned} &\left(S^{\ominus\varepsilon} \cap C(b',-)\right) \setminus \left(b' \cup \left(\bigcup \mathscr{B}_-\right)\right) \\ &= \left(S^{\ominus\varepsilon} \cap C(b',-)\right) \setminus \left(b' \cup \left(\bigcup \mathscr{B}_-\right) \cup \left(\bigcup \mathscr{B}_+\right)\right) \\ &= \left(S^{\ominus\varepsilon} \cap C(b',-)\right) \setminus \bigcup \mathscr{B} \\ &\subseteq S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B} \end{aligned}$$

Hence, $S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B} \neq \varnothing$ which is impossible since $\mathscr{B}$ is an ε-covering. Thus, $\mathscr{B}_+$ contains a candidate to $\mathscr{B}_-$.

Because $\mathscr{B}_-$ may have several distinct T-maximal candidates, $\mathscr{B}_+$ may only contain candidates T-unrelated to b_0. By contradiction assume $\mathscr{B}_+$ is void of candidate to $\mathscr{B}_-$ T-smaller than or equal to b_0. Since $\mathscr{B}_+$ contains at least one candidate to $\mathscr{B}_-$, b_0 cannot be centered at the root of T. By Proposition 3, any ball T-smaller than or equal to b_0 is a candidate to $\mathscr{B}_-$. Hence

$\mathscr{B}_+$ only contains balls that are either strictly T-larger than b_0, or T-unrelated to b_0. Therefore, there exists a medial ball $b' >_T b_0$, with b' also strictly T-smaller or T-unrelated to balls in $\mathscr{B}_+$. By T-maximality of b_0, b' cannot be a candidate to $\mathscr{B}_-$. Once again, we have a ball b', T-minimal in $\mathscr{B}'_+ = \mathscr{B}_+ \cup \{b'\}$ which is not a candidate. The same development as above using Corollary 10 yields $S^{\ominus\varepsilon} \setminus \bigcup \mathscr{B} \neq \varnothing$, which is still impossible. Therefore, $\mathscr{B}_+$ must contain a candidate to $\mathscr{B}_-$ that is T-smaller than or equal to b_0. □

Proposition 12 *Algorithm 1 converges to an optimal ε-covering.*

Proof. We denote by $\mathscr{B}_{\text{algo}}$ the ε-covering found by Algorithm 1. We number the balls of $\mathscr{B}_{\text{algo}}$ by $b_1, \ldots, b_k$, such that for $i \leq j$, b_i was found before b_j. Consider any optimal ε-covering $\mathscr{B}_{\text{opt}}$. We assume without loss of generality that $\mathscr{B}_{\text{opt}}$ only contains medial balls. Indeed, S is a finite union of balls, hence any ball $b \in \mathscr{B}_{\text{opt}}$ is wholly contained in a medial ball. Using consecutive substitutions, we want to build a finite sequence of ε-coverings $\mathscr{B}_0, \ldots, \mathscr{B}_k$ that satisfies the properties:

(a) $\mathscr{B}_0 = \mathscr{B}_{\text{opt}}$,

(b) $|\mathscr{B}_{i+1}| = |\mathscr{B}_i|$, $\forall i \in [\![0, k-1]\!]$,

(c) $\{b_1, \ldots, b_i\} \subseteq \mathscr{B}_i$, $\forall i \in [\![1, k]\!]$,

If such a sequence exists, we immediately deduce that $|\mathscr{B}_{\text{algo}}| = |\mathscr{B}_{\text{opt}}|$, and $\mathscr{B}_{\text{algo}}$ is also optimal.

We proceed by induction. Assume that for $0 \leq i < k$, we have built $\mathscr{B}_0, \ldots, \mathscr{B}_i$ with the above properties. Consider b_{i+1}. Let $\mathscr{B}_- = \{b_1, \ldots, b_i\}$. By construction, $\mathscr{B}_- \subsetneq \mathscr{B}_i$. Let $\mathscr{B}_+ = \mathscr{B}_i \setminus \mathscr{B}_-$. Algorithm 1 guarantees that $\mathscr{B}_-$ is a partial T-small ε-covering and that b_{i+1} is a T-maximal candidate for $\mathscr{B}_-$. Hence we can apply Corollary 11 and there is a candidate b to $\mathscr{B}_-$, such that $b \in \mathscr{B}_+$ and $b \leq_T b_{i+1}$. Then, let $\mathscr{B}_{i+1} = (\mathscr{B}_i \cup \{b_{i+1}\}) \setminus \{b\}$. $\mathscr{B}_{i+1}$ satisfies both properties (b) and (c), we must prove that it is also an ε-covering. To do so, it suffices to prove that both $C(b_{i+1}, -) \cap S^{\ominus\varepsilon}$ and $C(b_{i+1}, +) \cap S^{\ominus\varepsilon}$ are contained in $\bigcup \mathscr{B}_{i+1}$. Because b_{i+1} is a candidate to $\mathscr{B}_-$, we have $C(b_{i+1}, -) \cap S^{\ominus\varepsilon} \subseteq b_{i+1} \cup (\bigcup \mathscr{B}_-) \subseteq \bigcup \mathscr{B}_{i+1}$. Also, $b \leq_T b_{i+1}$, hence by Proposition 3 we have $C(b_{i+1}, +) \subseteq C(b, +)$. This implies $C(b_{i+1}, +) \cap S^{\ominus\varepsilon} \subseteq C(b, +) \cap S^{\ominus\varepsilon} \subseteq (\bigcup \mathscr{B}_i) \setminus b \subseteq \bigcup \mathscr{B}_{i+1}$. Thus, $\mathscr{B}_{i+1}$ is an ε-covering, and $\mathscr{B}_{\text{algo}}$ is an optimal ε-covering. □

7 Discussion

Consider the following more general covering definition, where $S^{\oplus\varepsilon'} = \cup_{y \in S} b(y, \varepsilon')$ is the dilation of S (by ε'):

Definition 4 An **$\varepsilon'\varepsilon$-covering** of S is a collection of balls $\mathscr{B}$ such that $S^{\ominus\varepsilon} \subseteq \bigcup \mathscr{B} \subseteq S^{\oplus\varepsilon'}$.

The algorithm presented computes in polynomial time an optimal $\varepsilon'\varepsilon$-covering of any union of balls S such that MA (S) is a forest and MA $(S^{\ominus\varepsilon}) \subseteq$ MA$(S^{\ominus\varepsilon'})$.

An interesting perspective is to build on this work to design a heuristic algorithm where both conditions on the medial axis are relaxed.

References

[1] N. Amenta and R. K. Kolluri. The medial axis of a union of balls. *Computational Geometry*, 20(1):25–37, 2001.

[2] D. Attali, T.-B. Nguyen, and I. Sivignon. Epsilon-covering is NP-complete. In *EuroCG*, Lugano, Switzerland, Mar. 2016.

[3] G. Bradshaw and C. O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics (TOG)*, 23(1):1–26, 2004.

[4] F. Cazals, T. Dreyfus, S. Sachdeva, and N. Shah. Greedy geometric algorithms for collection of balls, with applications to geometric approximation and molecular coarse-graining. In *Computer Graphics Forum*, volume 33, pages 1–17. Wiley Online Lib., 2014.

[5] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* 2001.

[6] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3d geometry. In *ACM Transactions on Graphics (TOG)*, volume 29, page 101. ACM, 2010.

[7] H. Schwerdtfeger. *Geometry of complex numbers: circle geometry, Moebius transformation, non-euclidean geometry.* Courier Corporation, 1979.

Geometric Unique Set Cover on Unit Disks and Unit Squares

Saeed Mehrabi*

Abstract

We study the Unique Set Cover problem on unit disks and unit squares. For a given set P of n points and a set D of m geometric objects both in the plane, the objective of the Unique Set Cover problem is to select a subset $D' \subseteq D$ of objects such that every point in P is covered by at least one object in D' and the number of points covered uniquely is maximized, where a point is *covered uniquely* if the point is covered by exactly one object in D'. In this paper, (i) we show that the Unique Set Cover is NP-hard on both unit disks and unit squares, and (ii) we give a PTAS for this problem on unit squares by applying the *mod-one* approach of Chan and Hu (Comput. Geom. 48(5), 2015).

1 Introduction

Consider a set P of n points and a set D of m geometric objects both in the plane. For a subset $S \subseteq D$ of objects, we say that a point $p \subseteq P$ is *covered uniquely* by S if there is exactly one object in S that covers p. In the Unique Set Cover problem, the objective is to compute a subset $S \subseteq D$ of objects so as to cover all points in P and to maximize the number of points covered uniquely by S.

There is a slightly different problem, called Unique Cover, where the input is the same as the Unique Set Cover problem and the objective is to compute a subset $S \subseteq D$ that maximizes the number of points covered uniquely; note that not all the points in P are required to be covered in an instance of the Unique Cover problem. Both Unique Set Cover and Unique Cover belong to the larger class of the Unit Cover problem in which we are given the same input and the objective is to compute a minimum-cardinality subset $S \subseteq D$ so as to cover all points in P. Indeed, Unique Set Cover combines the objectives of the Unique Cover and Unit Cover problems.

In this paper, we are interested in the Unique Set Cover problem on unit disks and unit squares. Formally, given a set P of n points and a set D of m unit disks (resp., unit squares) both in the plane, the objective of the Unique Disk (resp., Square) Set Cover problem is to find a subset $S \subseteq D$ that covers all points in P and that maximizes the number of points covered uniquely. Figure 1 shows an instance of, e.g., the Unique Disk Set Cover problem and an optimal solution for this instance.

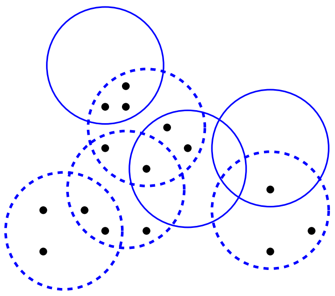


Figure 1: An instance of the Unique Disk Set Cover problem with $n = 15$ and $m = 7$, and an optimal solution, where 11 points are covered uniquely by the 4 dashed disks.

The Unique Cover problem was first studied by Erlebach and van Leeuwen [5] on unit disks and is motivated by its applications in wireless communication networks, where the broadcasting range of equivalent base stations have been frequently modelled as unit disks. Providers of wireless networks often consider having a number of base stations to service their customers. However, if a customer receives signals from too many base stations, then the customer may receive no service at all due to the resulting interference. As such, each customer is ideally serviced by exactly one base station and we want such a service to be provided to as many customers as possible. Our motivation for studying the Unique Set Cover problem is that what if in addition to providing service to as many customers as possible by exactly one base station, we still want to do service *all* the customers as well.

Related Work. The Unit Cover problem is a well-known NP-hard problem on unit disks and unit squares. Mustafa and Ray [11] gave the first PTAS for the Unit Cover problem on both unit disks and unit squares using a *local search technique*. This technique was also discovered independently by Chan and Har-Peled [1] who gave the first PTAS for the maximum independent set problem on pseudo-disks in the plane. Demaine et al. [3] introduced the non-geometric variant of the Unique Cover problem and gave a polynomial-time $O(\log n)$-

*Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. smehrabi@uwaterloo.ca

Problem	Unit Squares	Unit Disks
Unit Cover	NP-hard, PTAS [11]	NP-hard, PTAS [11]
Unique Cover	NP-hard [5], PTAS [9]	NP-hard [5], 4.31-approximation [8]
Unique Set Cover	**NP-hard** [Theorem 2]	**NP-hard** [Theorem 3]
	PTAS [Theorem 6]	

Table 1: A summary of previous and new results; the new results are shown in bold.

approximation algorithm for the problem, where n is the number of elements of the universe in the corresponding set system. Erlebach and van Leeuwen [5], who were first to study the geometric version of the Unique Cover problem, showed that Unique Cover problem is NP-hard on both unit disks and unit squares (see also [12]).

By combining dynamic programming and the shifting strategy of Hochbaum and Maass [7], Erlebach and van Leeuwen [6] gave the first PTAS for the weighted version of the Unit Cover problem on unit squares, where each unit square in D is associated with a positive value as *weight* and the objective is to cover the points in P so as to minimize the total weight of the unit squares selected.[1] This approach was also used by Ito et al. [9] who gave a PTAS for the Unique Cover problem on unit squares. However, this technique does not seem to work for unit disks. In fact, Ito et al. [8] used this approach to give a polynomial-time approximation algorithm for the Unique Cover problem on unit disks with approximation factor $\sigma < 4.3095 + \epsilon$, where $\epsilon > 0$ is any fixed constant. Moreover, this approach involves sophisticated dynamic programming. Finally, by introducing a *mod-one transformation*, Chan and Hu [2] gave a PTAS for the Red-Blue Unit-Square Cover problem, which is defined as follows: given a red point set R, a blue point set B and a set of unit squares in the plane, the objective is to select a subset of the unit squares so as to cover all the blue points while minimizing the number of red points covered.

Our Results. In this paper, we first show that both Unique Disk Set Cover and Unique Square Set Cover are NP-hard. We note that the Unique Cover problem is shown to be NP-hard on unit disks and unit squares [5]. However, their hardness, which is based on a reduction from the Independent Set problem on planar graphs of maximum degree 3, does not apply to proving the hardness of the Unique Set Cover problem mainly because all points in P are required to be covered in an instance of the Unique Set Cover problem. We instead show a reduction from a variant of the planar 3SAT problem to prove the NP-hardness of Unique Set Cover on both unit disks and unit squares.

[1]The local search technique of Mustafa and Ray [11] does not apply to the weighted version of these problems.

As our second result, we show that the *mod-one* approach of Chan and Hu [2] provides a PTAS for the Unique Square Set Cover problem. The only difference is that instead of storing 4-tuples of unit squares when solving the dynamic programming, we store 6-tuples of unit squares and show that they suffice to capture the necessary information (we will discuss this approach in more details in Section 3). See Table 1 for a summary of previous and new results.

We first prove the NP-hardness of the problems in Section 2 and will then give a PTAS for the problem on unit squares in Section 3. Finally, we conclude the paper with a discussion on open problems in Section 4.

2 NP-Hardness

In this section, we first show that the Unique Disk Set Cover is NP-hard; the NP-hardness of the Unique Square Set Cover is proved analogously and we will discuss it at the end of this section.

Unique Disk Set Cover
Input. A set P of n points, a set D of m unit disks, and an integer $k > 0$.
Output. Does there exist a set $S \subseteq D$ that covers all points in P and that covers at least k points uniquely?

To prove the hardness of the Unique Disk Set Cover, we show a reduction from the Planar Variable Restricted 3SAT (Planar VR3SAT, for short) problem. Planar VR3SAT is a constrained version of 3SAT in which each variable can appear in at most three clauses and the corresponding *variable-clause graph* is planar. Efrat et al. [4] showed that Planar VR3SAT is NP-hard.

Let $I_{\mathtt{SAT}}$ be an instance of Planar VR3SAT with K clauses $C_1, C_2, \ldots, C_K$ and N variables $X_1, X_2, \ldots, X_N$; we denote the two literals of a variable X_i by x_i and $\overline{x_i}$. We construct an instance $I_{\mathtt{USC}}$ of the Unique Disk Set Cover problem such that $I_{\mathtt{USC}}$ has a solution with at least $c \cdot (K+1)$ points covered unqiuely, for some c that we will determine its value later, if and only if $I_{\mathtt{SAT}}$ is satisfiable. Given $I_{\mathtt{SAT}}$, we first construct the variable-clause graph G of $I_{\mathtt{SAT}}$ in the non-crossing comb-shape form of Knuth and Raghunathan [10]. Without loss of generality, we assume that the variable vertices lie on a vertical line and the clause vertices are connected from

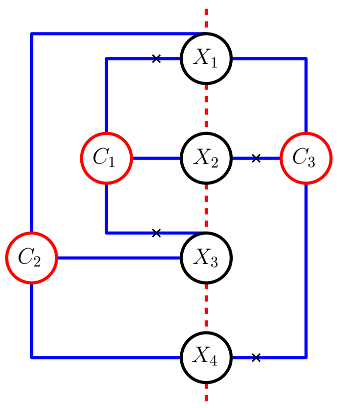


Figure 2: An instance of the Planar VR3SAT problem in the comb-shape form of Knuth and Raghunathan [10]. Crosses on the edges indicate negations; for example, $C_1 = (\overline{x_1} \vee x_2 \vee \overline{x_3})$.

left or right of that line; see Figure 2 for an illustration. Note that each variable appears in at most three clauses.

Gadgets. For each variable $X_i \in I_{\texttt{SAT}}$, we replace the corresponding variable vertex in G with a single unit disk containing three *groups of points* each of which consists of $K+1$ points (where K is the number of clauses in $I_{\texttt{SAT}}$). We call each such groups of points a *cloud*. If a literal of X_i appears in a clause, then the variable gadget of X_i is connected to the corresponding clause gadget by a chain of unit disks such that (i) the first unit disk in the chain covers exactly one of the clouds in the variable unit disk, and (ii) every two consecutive unit disks in the chain share a cloud. We call such a chain of unit disks a *wire*; see Figure 3(Right) for an illustration. A cloud shared between two unit disks in a wire has also $K+1$ points. We call the unit disk of a wire that shares a cloud with the variable unit disk a *start disk*. For the clause gadget, where three wires meet, we make the last three unit disks (each of which arriving from one of the wires) to have a non-empty intersection region in which we insert one single point; see Figure 3(Left). We call this point a *clause point*.

Consider the cloud shared between a variable unit disk and a start disk. If all the clouds in the corresponding wire are covered uniquely, then exactly one of the variable unit disk and the start disk must be selected to cover the cloud shared between them. Consequently, depending on whether the variable unit disk is selected, we can decide whether the clause point of the clause gadget on the other end of the wire is covered by the last unit disk of this wire. That is, we can create two ways of covering the clouds of the wire uniquely, one of which will not be able to cover the corresponding clause point. It is clear from the variable unit disk shown in Figure 3 that if the variable unit disk is selected (resp., is not selected), then the clause point is covered (resp.,

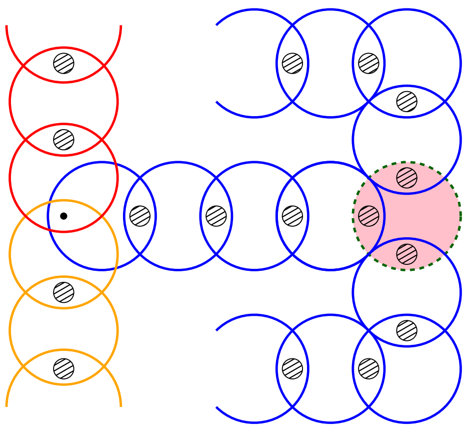


Figure 3: An illustration of the gadgets used in our reduction. **Right.** The filled (pink) unit disk indicates a variable unit disk and each small (rising) shaded circle indicates a cloud consisting of $K+1$ points. Each variable unit disk shares three clouds with the start disks of the three wires that connect the variable unit disk to the clauses in which it appears. **Left.** A clause gadget determined by the non-empty intersection of the last three unit disks of the three wires arriving from the literals of this clause, and the corresponding clause point.

is not covered) by the last unit disk of the corresponding wire. We remark that this is always doable, even for the wires that require one bend, by adjusting the number of unit disks in the wire. See the top wire shown in Figure 4 for an example. We set a variable to true or false depending on whether its corresponding variable unit disk is selected or not to cover the clouds that it contains.

Finally, we need a gadget for negation literals. Suppose that the literal $\overline{x_i}$ appears in a clause, for some variable X_i in $I_{\texttt{SAT}}$. To indicate the negation literal, we add two additional unit disks besides the wire connecting the variable unit disk to the corresponding clause gadget; see the two (falling and green) shaded unit disks in the middle wire (i.e., the wire corresponding to X_j) shown in Figure 4. We call these two newly-added unit disks the *negation pair*. Observe that the negation pair share a cloud and each of them covers one of the previously existed clouds in the current wire. The construction of the negation pair ensures that if the variable unit disk is selected (resp., is not selected), i.e., it is set to true (resp., false), then the clause point is not covered (resp., is covered). The same gadget can be used for the negation literals whose corresponding wires have a bend; see for instance the lowest wire shown in Figure 4.

Construction Details. Clearly, our construction allows the wires to be connected to a variable unit disk from both left and right of the variable unit disk; we just

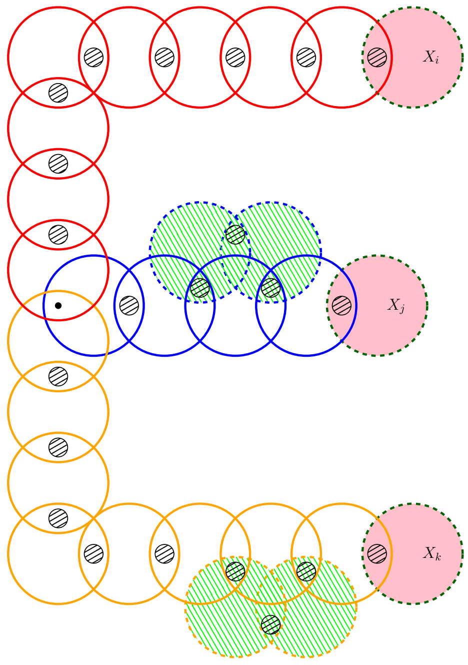


Figure 4: A complete illustration of a clause gadget $C = (x_i \vee \overline{x_j} \vee \overline{x_k})$ and its corresponding variable gadgets. The existence of a negation pair indicates that literal $\overline{x_j}$ (or literal $\overline{x_k}$) appears in C.

need to move the corresponding clouds inside the variable unit disk accordingly. By scaling and making the drawing of the wires consistent with the edges of G, we can ensure that the unit disks of different wires will never intersect. Note that we can have the scaling so as to increase the number of unit disks in any wire by only a constant factor. Hence, by this and from the construction, we have that the number of disks in the instance $I_{\mathtt{USC}}$ is polynomial in terms of K and N. To see the value of n, the number of points in $I_{\mathtt{USC}}$, we set c to the number of clouds used in our construction. Since each cloud contains $K+1$ points, the total number of points in $I_{\mathtt{USC}}$ is $n = c \cdot (K+1) + K$ as we also have K clause points. Therefore, the total complexity of the constructed instance $I_{\mathtt{USC}}$ is polynomial in K and N. Moreover, it is not hard to see that $I_{\mathtt{USC}}$ can be constructed in polynomial time. We now show the following result.

Lemma 1 *There exists a feasible solution S for $I_{\mathtt{USC}}$ that covers at least $c \cdot (K+1)$ points uniquely if and only if $I_{\mathtt{SAT}}$ is satisfiable.*

Proof. ($\Rightarrow$) Let S be a feasible solution for $I_{\mathtt{USC}}$ that covers at least $c \cdot (K+1)$ points. First, by the choice of c and the fact that we have exactly K clause points and any other point belongs to some cloud, we conclude that all the clouds in $I_{\mathtt{USC}}$ must be covered uniquely by S. For each variable X_i, where $1 \leq i \leq N$, we set the variable X_i to true if its corresponding unit disk is in S; otherwise, we set X_i to false. To show that this results in a truth assignment, suppose for a contradiction that there exists a clause C that is not satisfied by this assignment. Take any variable $X \in C$. If $x \in C$ (resp., $\overline{x} \in C$), then the variable X is set to false (resp., true) by the assignment and so the variable unit disk corresponding to X is not in S (resp., is in S). Consequently, it follows from the construction that the point clause of C is not covered by the wire arriving from X. This means that none of the wires arriving at C will cover its point clause — this is a contradiction to the feasibility of S.

($\Leftarrow$) Given a truth assignment for $I_{\mathtt{SAT}}$, we construct a feasible solution S for $I_{\mathtt{USC}}$ covering at least $c \cdot (K+1)$ points uniquely as follows. For each variable X_i in $I_{\mathtt{SAT}}$, where $1 \leq i \leq N$: if X_i is set to true, then we add the corresponding variable unit disk into S; otherwise, we add the start disks intersecting this variable unit disk into S. Consequently, every other unit disks of the corresponding wires are added into S in such a way that each cloud is covered uniquely by one of the unit disks along the wire. Clearly, all clouds are covered by S. Moreover, S also covers all the clause points because the only way to have a clause C satisfied is to have at least one of the wires arriving at C covering the clause point of C. Finally, by adding every other unit disk of each wire into S, we ensure that all the clouds are covered uniquely by S. Since we have c clouds each of which consists of $K+1$ points, we conclude that S is a feasible solution that covers at least $c \cdot (K+1)$ points in $I_{\mathtt{USC}}$ uniquely. □

By Lemma 1, we have the following theorem.

Theorem 2 *The* Unique Disk Set Cover *is* NP-*hard.*

NP-**Hardness for Unit Squares.** The proof for the NP-hardness of the Unique Square Set Cover problem is almost identical to the one shown above for unit disks. In what follows, we mainly describe the gadgets for unit squares so as then one can verify that the hardness follows.

Starting by the comb-shape form of Knuth and Raghunathan [10] for an instance of the Planar VR3SAT problem, we replace each variable vertex with a unit square and will then have three (unit-square type) wires connecting the variable unit square to the corresponding clause gadgets. This is again doable even if we have one bend along the wire; see Figure 5 for an illustration. Moreover, we can make the three wires arriving at a clause to have a non-empty intersection region in which we insert our single clause point. Finally, the negation pair is constructed in an analogous way. See Figure 5. Note that more unit squares are used (than unit disks) in a wire connecting a variable unit square to

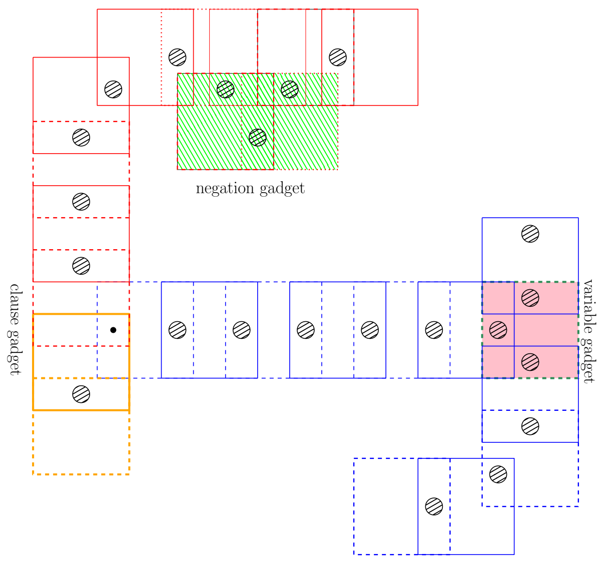


Figure 5: An illustration of the gadgets for unit squares. Solid and dashed unit squares alternate for better visibility.

its corresponding clauses. However, we can still ensure to have the scaling step to have a polynomial number of unit squares in each wire.

One can verify that the construction is again polynomial in K and N, and we can prove a lemma similar to Lemma 1 for this new construction. So, we have the following result.

Theorem 3 *The* Unique Square Set Cover *is* NP-*hard.*

3 PTAS

In this section, we give a PTAS for the Unique Square Set Cover problem by applying the mod-one approach of Chan and Hu [2]; we first describe this approach.

Recall the dynamic programming involed in the PTASes developed by Ito et al. [9], and Erlebach and van Leeuwen [6]. The dynamic program is essentially based on the line-sweep paradigm by considering points and squares from left to right, and extending the uniquely covered region sequentially. However, adding one square can influence squares that were already chosen and so we need to keep track of the squares that are possibly influenced by a newly-added square. The complication stems mainly from the fact that a new square may influence too many squares. This requires keeping track of the changes on the two separate chains induced by the boundary of the current squares. The mod-one approach avoids this complication by transforming these chains into two chains that are connected at the corner points.

For a set $S = \{s_1, \dots, s_t\}$ of t unit squares containing a common point, where $s_1, \dots, s_t$ are arranged in increasing x-order of their centres, the set S is called a *monotone set* if the centres of $s_1, \dots, s_t$ are in increasing or decreasing y-order. The boundary of the union of the squares in a monotone set S consists of two monotone chains, called *complementary chains.* In the *mod-one transformation*, a point (x, y) in the plane is mapped to the point $(x \mod 1, y \mod 1)$, where $x \mod 1$ denotes the fractional part of the real number x. Applying the mod-one to a monotone set transforms the squares in such a way that the two complementary chains are mapped to two monotone chains that are connected at the corner points. See Figure 6 for an illustration.

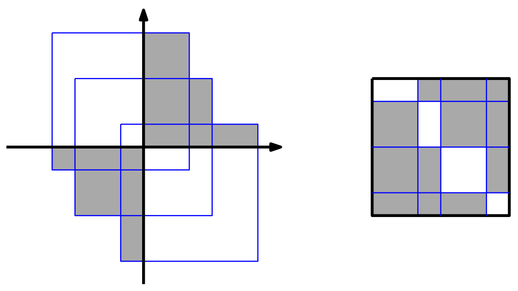


Figure 6: A monotone set consisting of three unit squares (left), and the resulting set after applying the mod-one transformation (right).

We now show that the mod-one approach provides a PTAS for the Unique Square Set Cover problem. The plan is to first give an exact dynamic programming algorithm for a special variant of the problem, where the points in P are all inside a $k \times k$ square for some constant k, and then to apply the shifting strategy of Hochbaum and Maass [7] to obtain our PTAS. Note that this follows the framework of [2] with the only difference that we store 6-tuples of unit squares, instead of storing 4-tuples, when solving the dynamic programming.

Lemma 4 *Let OPT denote an optimal solution for an instance of the* Unique Square Set Cover *problem in which the set P is inside a $k \times k$ square, for some constant k. Then, OPT can be decomposed into $O(k^2)$ monotone sets.*

Proof. This lemma is essentially proved in [2], but for a different problem. The idea is to draw a unit side-length grid over the $k \times k$ square and then show that each unit square appears in the boundary of the union of unit square containing a fixed grid point p. By dividing the plane into four quadrants at p and grouping the unit squares containing p based on their contribution to the part of the union boundary in quadrants, OPT is decomposed into $4(k + 1)^2$ monotone sets. $\square$

We now give an exact dynamic programming algorithm for the variant of the Unique Square Set

Cover problem, where all points of P are inside a $k \times k$ square for some constant k.

Theorem 5 *For any instance of* Unique Square Set Cover *problem in which the set P is inside a $k \times k$ square for a constant k, the optimal solution can be computed in $O(n \cdot m^{O(k^2)})$ time.*

Proof. We describe the dynamic programming algorithm by a state-transition diagram. We store 6-tuples of unit squares in each state. More specifically, a state is defined to consist of (i) a vertical sweep line ℓ that passes through a corner of an input square, after taking mod 1, and (ii) $O(k^2)$ 6-tuples of unit squares of the form $(s_{\texttt{start}}, s_{\texttt{prev'}}, s_{\texttt{prev}}, s_{\texttt{curr}}, s_{\texttt{curr'}}, s_{\texttt{end}})$ given that $s_{\texttt{start}}$, $s_{\texttt{prev'}}$, $s_{\texttt{prev}}$, $s_{\texttt{curr}}$, $s_{\texttt{curr'}}$, and $s_{\texttt{end}}$ are in the increasing order of x-coordinate, they form a monotone set, and ℓ lies between the corners of $s_{\texttt{prev}}$ and $s_{\texttt{curr}}$ mod 1.

Observe that each 6-tuple corresponds to a monotone set S: $s_{\texttt{start}}$ and $s_{\texttt{end}}$ represent the start and end squares of S and $s_{\texttt{prev}}$ and $s_{\texttt{curr}}$ represent the squares of the two complementary chains of S, after taking mod 1, that are intersected by the sweep line ℓ. Moreover, we define $s_{\texttt{prev'}}$ and $s_{\texttt{curr'}}$ as the predecessor of $s_{\texttt{prev}}$ and the successor of $s_{\texttt{curr}}$, respectively. We now define a transition between two states and its cost function. Given a state A, we create a transition from A to a new state B as follows. Let $(s_{\texttt{start}}, s_{\texttt{prev'}}, s_{\texttt{prev}}, s_{\texttt{curr}}, s_{\texttt{curr'}}, s_{\texttt{end}})$ be the 6-tuple such that the corner point of $s_{\texttt{curr}}$ has the smallest x-coordinate mod 1. First, the new sweep line ℓ' is located at the corner of $s_{\texttt{curr}}$. Next, we replace this 6-tuple by a new 6-tuple $(s_{\texttt{start}}, s_{\texttt{prev}}, s_{\texttt{curr}}, s_{\texttt{curr'}}, s', s_{\texttt{end}})$, where s' is a unit square that satisfies the conditions of a state. B is now set to this new state with having all other 6-tuples left unchanged. To see the cost of this transition, let x (resp., y) be the number of points in P that lie between ℓ and ℓ', after taking mod 1, and are not covered (resp., are covered uniquely) by the squares from the $O(k^2)$ 6-tuples of unit squares, before taking mod 1. If $x > 0$, then we remove this transition from the diagram (since all points must be covered). Otherwise, we set the cost of this transition to y.

The problem is then reduced to finding the longest path in this state-transition diagram for which we can appropriately add the start and end states. Since the diagram is a directed acyclic graph, we can construct the diagram and find the longest path in $O(n \cdot m^{O(k^2)})$ time. □

We can now apply the shifting strategy of Hochbaum and Maass [7] to obtain our PTAS. The proof of the following theorem is much similar to the one given in [2] and so we omit it here.

Theorem 6 *For any fixed constant $\epsilon > 0$, there exists a polynomial time $(1 + \epsilon)$-approximation algorithm for the* Unique Square Set Cover *problem.*

4 Conclusion

In this paper, we showed that the Unique Disk Set Cover and Unique Square Set Cover problems are both NP-hard and gave a PTAS for the Unique Square Set Cover problem. Our PTAS is based on the mod-one transformation of Chan and Hu [2], which does not apply to unit disks. Giving a PTAS for the Unique Disk Set Cover remains open.

References

[1] T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

[2] T. M. Chan and N. Hu. Geometric red-blue set cover for unit squares and related problems. *Comput. Geom.*, 48(5):380–385, 2015.

[3] E. D. Demaine, U. Feige, M. Hajiaghayi, and M. R. Salavatipour. Combination can be hard: Approximability of the unique coverage problem. *SIAM J. Comput.*, 38(4):1464–1483, 2008.

[4] 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.

[5] T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 1267–1276, 2008.

[6] T. Erlebach and E. J. van Leeuwen. PTAS for weighted set cover on unit squares. In *proceedings of Approx., Random., and Comb. Opt. (APPROX-RANDOM 2010)*, pages 166–177, 2010.

[7] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[8] T. Ito, S. Nakano, Y. Okamoto, Y. Otachi, R. Uehara, T. Uno, and Y. Uno. A 4.31-approximation for the geometric unique coverage problem on unit disks. *Theor. Comput. Sci.*, 544:14–31, 2014.

[9] T. Ito, S. Nakano, Y. Okamoto, Y. Otachi, R. Uehara, T. Uno, and Y. Uno. A polynomial-time approximation scheme for the geometric unique coverage problem on unit squares. *Comput. Geom.*, 51:25–39, 2016.

[10] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Discrete Math.*, 5(3):422–427, 1992.

[11] N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[12] E. J. van Leeuwen. *Optimization and Approximation on Systems on Geometric Objects.* PhD thesis, University of Amsterdam, Amsterdam, Netherlands, 2009.

Stabbing Line Segments with Disks and Related Problems

Raghunath Reddy M[*] Apurva Mudgal[†‡]

Abstract

We show that the problem of stabbing a set of line segments with minimum number of unit disks is APX-hard. However, we give a PTAS when no two line segments intersect. The approximation hardness of the problem remains the same, even when the objective is to stab unit disks with line segments. In addition to this, we show that stabbing circles with circles is also APX-hard. On the other hand, we show that there exists a PTAS for stabbing disks with disks.

1 Introduction

We are given two sets of objects $\mathcal{X}$ and $\mathcal{Y}$. One has to find a minimum cardinality subset $\mathcal{Y}'$ of $\mathcal{Y}$ such that for each $X \in \mathcal{X}$, there is an object $Y \in \mathcal{Y}'$ and $X \cap Y \neq \emptyset$. We call this problem as *stabbing objects with objects*. We consider sets $\mathcal{X}, \mathcal{Y}$ of objects like disks, line segments, circles. We say an object Y stabs (hits) another object X iff $X \cap Y \neq \emptyset$. An example is shown in Figure 1.

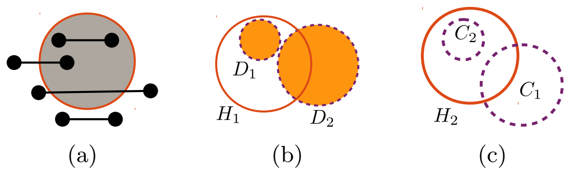


Figure 1: (a) All the line segments except the bottom one are hit by the disk. (b) Disk H_1 hits both the disks D_1 and D_2. (c) Circle H_2 hits circle C_1 but not circle C_2 since the boundaries of H_1 and C_2 do not intersect.

Following problems are considered in this paper.

Problem 1: Stabbing Line Segments with Disks (*Stab-LS-Disks*). Let $\mathcal{X}$ be a set of line segments and $\mathcal{Y}$ be a set of disks.

Problem 2: Stabbing Disks with Line Segments (*Stab-Disks-LS*). Let $\mathcal{X}$ be a set of disks and $\mathcal{Y}$ be a set of line segments.

Problem 3: Stabbing Circles with Circles (*Stab-Cir-Cir*). Let both $\mathcal{X}$ and $\mathcal{Y}$ be the sets of circles.

Problem 4: Stabbing Disks with Disks (*Stab-Disk-Disk*). Let both $\mathcal{X}$ and $\mathcal{Y}$ be the sets of disks.

We can note that both the geometric set cover and hitting set problems which are known to be NP-hard [7] with unit disks and points respectively, are special cases of *Stab-LS-Disks* and *Stab-Disks-LS* respectively. Further, these problems are also a special case of *Stab-Disk-Disk*. Therefore, the problems *Stab-LS-Disks*, *Stab-Disks-LS*, and *Stab-Disk-Disk* are also NP-hard. Further, in discrete case, *Stab-Cir-Cir* is known to be APX-hard [9].

Results.

1. *Stab-LS-Disks* is APX-hard, even if the disks in $\mathcal{Y}$ are unit disks and we are allowed to choose their positions. However, we give a PTAS for a special case of *Stab-LS-Disks*, when no two line segments in $\mathcal{X}$ intersect.
2. *Stab-Disks-LS* is APX-hard, even for unit disks. There exists a PTAS for a special case where no two line segments intersect.
3. *Stab-Cir-Cir* is APX-hard, even if we are allowed to choose the position of the (unit) circles in $\mathcal{Y}$.
4. There exists a PTAS for *Stab-Disk-Disk.*

Previous Work. For geometric set cover with disks, a PTAS is given by Mustafa and Ray [11]. Later Aschner et al. [2], Wan et al. [12] also gave a PTAS for the same problem and the results hold true for geometric set cover with squares also. For weighted geometric set cover with unit squares, Erlebach et al. gave a PTAS [6].

On the other hand, Chaplick et al. [5] proved that stabbing polygonal chains with rays is APX-hard. Katz et al. [10] proved that stabbing horizontal line segments with vertical line segments is NP-complete, whereas the problem lies in P when the hitting objects are vertical lines.

[*]Department of Computer Science & Engineering, Indian Intitute of Technology Ropar, Rupnagar, India - 140001, raghunath.reddy@iitrpr.ac.in

[†]Department of Computer Science & Engineering, Indian Intitute of Technology Ropar, Rupnagar, India - 140001, apurva@iitrpr.ac.in

[‡]Partially supported by grant No. SB/FTP/ETA-434/2012 under DST-SERB Fast Track Scheme for Young Scientist

2 Approximation Hardness

2.1 Stabbing Axis-Parallel Line Segments with Unit Disks

In this section, we show that *Stab-LS-Disks* is APX-hard by giving a polynomial-time reduction from $MAX-3SAT(5)$ (optimization version of $3SAT$ where every variable can occur in at most 5 clauses) which is known to be APX-hard [1]. In fact, we show that a special case of *Stab-LS-Disks* is APX-hard, where each line segment in $\mathcal{X}$ is axis-parallel, and all the disks in $\mathcal{Y}$ are unit disks. Further, we assume that we can choose the positions of the disks in $\mathcal{Y}$.

Let ϕ be an instance of $MAX-3SAT(5)$ with n variables and m clauses. Let $x_1, x_2, \ldots, x_n$ be the variables in the instance ϕ. Since every variable x_i can occur in at most 5 clauses and every clause contains exactly three literals, $5n \leq 3m$.

Consider $C_1, C_2, \ldots, C_m$ to be an ordering of clauses. We say two clauses C_l and C_k ($1 \leq l < k \leq m$) to be *consecutive occurring clauses* with respect to variable x_i ($i = 1, 2, \ldots, n$) if x_i occurs in both C_l and C_k but not in any one of $C_{l+1}, \ldots, C_{k-1}$.

We now create the instance I_ϕ of $Stab-LS-Disks$ for the instance ϕ as follows:

Clause gadget: For every clause C_j ($j = 1, 2, \ldots, m$), consider a long horizontal line segment H_j^c. Place all the horizontal line segments $H_1^c, H_2^c, \ldots, H_m^c$ in the same order (top to down), with vertical gap greater than 4 between H_j^c and H_{j+1}^c for every $j = 1, 2, \ldots, m-1$. We call the line segments $H_1^c, H_2^c, \ldots, H_m^c$ as clause line segments.

Variable gadget: For each variable x_i ($i = 1, 2, \ldots, n$), we form a cycle $\mathcal{L}_i$ of horizontal and vertical line segments as follows:

- Consider two (small) horizontal line segments, say H_i^t and H_i^b. We place the line segment H_i^t above the line segment H_1^c with vertical distance greater than 4. Similarly, we place the line segment H_i^b below the line segment H_m^c with vertical distance greater than 4.
- Consider two long vertical line segments, V_i^{left} and V_i^{right}. We place them at a horizontal distance greater than 2 and a unit disk can hit (i) both V_i^{left} and H_i^t, (ii) both V_i^{right} and H_i^t, (iii) both V_i^{left} and H_i^b, and (iv) both V_i^{right} and H_i^b. Hence the vertical line segments V_i^{left} and V_i^{right} intersect all the clause line segments $H_1^c, H_2^c, \ldots, H_m^c$.

Place all the cycles $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_n$ such that no unit disk can hit two line segments from different cycles. For the outline of the cycles $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$, see Figure 2.

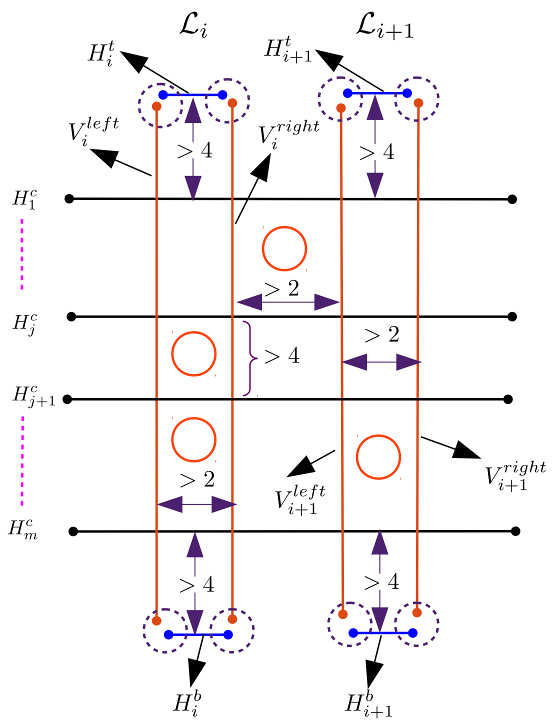


Figure 2: Outline of cycles $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$.

We now break the line segments V_i^{left} and V_i^{right}, for $i = 1, 2, \ldots, n$, into small pieces as follows:

Step 1 Suppose the variable x_i occurs in clause C_j. We break both line segments V_i^{left} and V_i^{right} at the intersection point with clause line segment H_j^c. In total, V_i^{left} and V_i^{right} breaks into at most 6 pieces each. Moreover, only two consecutive pieces can be hit by a unit disk. Further, if C_l and C_k are two consecutive occurring clauses with respect to x_i, then there is a unique piece (say $V_{i,(l,k)}^{left}$) of V_i^{left} and a unique piece (say $V_{i,(l,k)}^{right}$) of V_i^{right} between both clause line segments H_l^c and H_k^c.

Step 2 If the variable x_i occurs as the same literal in two consecutive occurring clauses C_l and C_k, then we break each $V_{i,(l,k)}^{left}$ and $V_{i,(l,k)}^{right}$ into two pieces such that only the new two pieces can be hit by a unit disk. See Figure 3.

Step 3 Further, we break the top most piece of V_i^{right} into two pieces before H_1^c and place them such that a unit disk can hit only these two pieces. Similarly, we break bottom most piece of V_i^{right} into two pieces below H_m^c and place them such that a disk can hit only these two pieces.

An example is given in Figure 4.

Let $l_i = |\mathcal{L}_i|$ for $i = 1, 2, \ldots, n$. Then the number of line segments, say L, in the instance I_ϕ is $\Sigma_{i=1}^n l_i + m$.

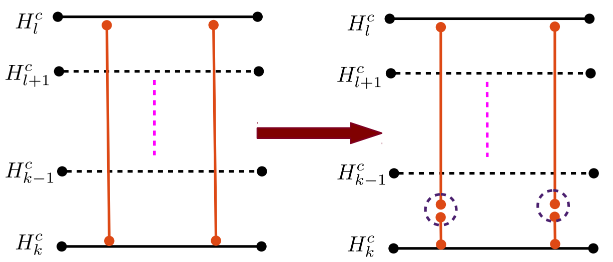


Figure 3: Variable x_i occurs in clauses C_l and C_k with same sign and does not occur in clauses $C_{l+1}, \ldots, C_{k-1}$.

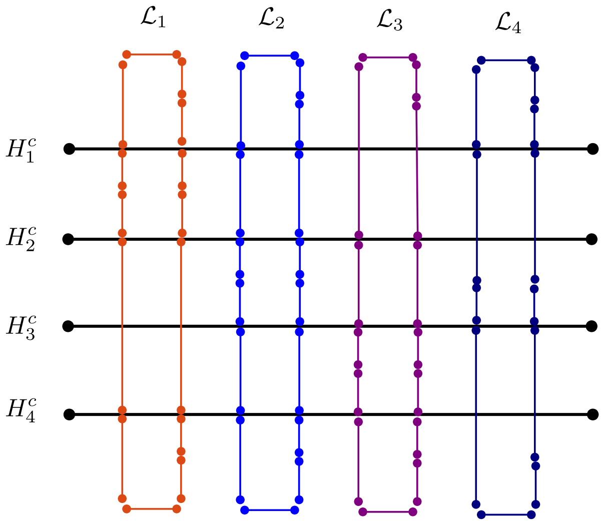


Figure 4: An instance of I_ϕ corresponding to $3SAT(5)$ instance ϕ with clauses $C_1 = x_1 \vee \neg x_2 \vee x_4$, $C_2 = x_1 \vee x_2 \vee x_3$, $C_3 = x_2 \vee \neg x_3 \vee x_4$, and $C_4 = \neg x_1 \vee \neg x_2 \vee \neg x_3$.

From the construction, we can note that $|\mathcal{L}_i| \leq 24$ for $i = 1, 2, \ldots, n$. Moreover, $|\mathcal{L}_i|$ is even $(i = 1, 2, \ldots, n)$. Then $L = \Sigma_{i=1}^n l_i + m \leq 24n + m$. Hence the reduction completes in $O(n + m)$ time.

In the instance I_ϕ, if the variable x_i occurs in clause C_l then the number of pieces of V_i^{right} and V_i^{left} above H_l^c (alternatively, below H_l^c) differ by one. Further, in cycle $\mathcal{L}_i$ $(i = 1, 2, \ldots, n)$, only two consecutive line segments can be hit by a unit disk. Hence we have the following lemma.

Lemma 1 *There exist two minimum size sets of unit disks of size $l_i/2$ each, which hits all line segments in $\mathcal{L}_i$ $(i = 1, 2, \ldots, n)$.*

Let $\mathcal{D}_i$ be a set of $l_i/2$ unit disks which can hit all the line segments in $\mathcal{L}_i$ $(i = 1, 2, \ldots, n)$. Then every unit disk in $\mathcal{D}_i$ hits exactly two line segments from $\mathcal{L}_i$ and every line segment in $\mathcal{L}_i$ hit by exactly one disk in $\mathcal{D}_i$. Moreover, a disk $D \in \mathcal{D}_i$ can hit a clause line segment H_l^C only if the variable x_i occurs in clause C_l.

Lemma 2 *Let the variable x_i occur in a clause C_l. Then exactly zero or two disks in $\mathcal{D}_i$ hit the clause line segment H_l^c.*

We partition the set of disks $\mathcal{D}_i$ into $\mathcal{D}_i^{left}$ and $\mathcal{D}_i^{right}$ which hit the pieces of V_i^{left} and V_i^{right} respectively.

Lemma 3 *The disks in $\mathcal{D}_i^{left}$ (and $\mathcal{D}_i^{right}$) hit either only all the clause line segments in which x_i occurs as positive literal or only all the clause line segments in which x_i occurs as negative literal.*

Proof. Without loss of generality, we assume that the variable x_i occurs in the clauses $C_{l_1}, C_{l_2}, \ldots, C_{l_5}$ $(1 \leq l_1 < l_2 < \cdots < l_5 \leq m)$. Assume that x_i occurs in the clause C_{l_1} as a positive literal. Other case is similar.

Let the variable x_i occur in the clause C_{l_2} as a positive literal. Then there exist four consecutive line segments V_1, V_2, V_3, V_4 pieces of V_i^{left} such that (i) only the line segments V_1, V_2, and $H_{l_1}^c$ can be hit by a disk D_1, (ii) only both line segments V_2 and V_3 can be hit by a disk D_2, and (iii) only the line segments V_3, V_4 and $H_{l_2}^c$ can be hit by a unit disk D_3. See Figure 5(a). But the set $\mathcal{D}_i^{left}$ contains either both D_1 and D_3 or only D_2. Hence either both clause line segments $H_{l_1}^c$ and $H_{l_2}^c$ or none is hit by disks in $\mathcal{D}_i^{left}$.

Let the variable x_i occur in clause C_{l_2} as a negative literal. Then there exist three consecutive line segments V_1, V_2, V_3 pieces of V_i^{left} such that (i) only the line segments V_1, V_2, and $H_{l_1}^c$ can be hit by a unit disk D_1 and (ii) only the line segments V_2, V_3, and $H_{l_2}^c$ can be hit by a unit disk D_2. See Figure 5(b). But the set $\mathcal{D}_i^{left}$ contains either D_1 or D_2. Hence only one of the clause line segment among $H_{l_1}^c$ and $H_{l_2}^c$ can be hit by disks in $\mathcal{D}_i^{left}$.

By repeating the above argument for clauses $C_{l_2}, C_{l_3}, C_{l_4}$ we can prove the lemma. □

From Lemma 2 and Lemma 3, disks in $\mathcal{D}_i$ hit either only clause line segments in which the variable x_i occurs as a positive literal or only clause line segments in which the variable x_i occurs as a negative literal. In the first case, we assign truth value *True* to x_i and in other case, we assign truth value *False* to x_i.

Lemma 4 *ϕ will have a satisfying assignment if and only if I_ϕ has an optimum solution of size $\Sigma_{i=1}^n l_i/2$.*

Proof. Suppose ϕ has a satisfying assignment $(s_1, s_2, \ldots, s_n)$ where $s_i \in \{True, False\}$ for $i = 1, 2, \ldots, n$. For every variable x_i $(i = 1, 2, \ldots, n)$, we pick the set of disks $\mathcal{D}_i$ corresponding to the truth assignment s_i. The disks in $\mathcal{D}_i$ hit all the line segments in $\mathcal{L}_i$ and $|\mathcal{D}_i| = l_i/2$ for $i = 1, 2, \ldots, n$. Since $(s_1, s_2, \ldots, s_n)$ is a satisfying assignment, there exists a literal in every clause C_j $(j = 1, 2, \ldots, m)$

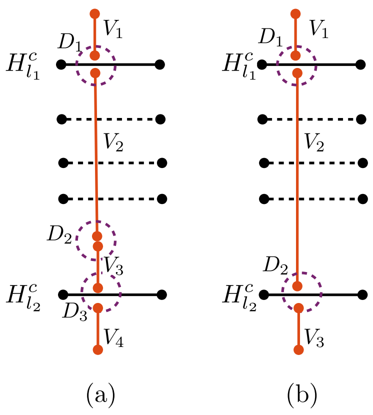


Figure 5: Variable gadget segments between clause segments corresponding to two consecutive occurring clauses C_{l_1} and C_{l_2} for variable x_i: (a) x_i is positive in both clauses and (b) x_i is positive in C_{l_1} and negative in C_{l_2}.

which is true and disks corresponding to that literal will hit the clause line segment H^c_j. Hence $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \mathcal{D}_n$ is an optimum solution to an instance I_ϕ and $|\mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \mathcal{D}_n| = \Sigma^n_{i=1} l_i/2$.

Let $\mathcal{D}$ be an optimal solution to an instance I_ϕ of size $\Sigma^n_{i=1} l_i/2$. Partition the set $\mathcal{D}$ into $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n$ and R, where $\mathcal{D}_i \subseteq \mathcal{D}$ is the set of disks which hit the line segments in the cycle $\mathcal{L}_i$ $(i = 1, 2, \dots, n)$ and R is the set of remaining disks.

$$\begin{aligned}\text{Therefore, } |\mathcal{D}| &= \Sigma^n_{i=1}|\mathcal{D}_i| + |R| \\ &= \Sigma^n_{i=1}|l_i/2| + |R|\end{aligned}$$

Hence $|R| = 0$, which implies $|\mathcal{D}_i| = \Sigma^n_{i=1} l_i/2$, and the disks in $\mathcal{D}_i$ must be a set of disks corresponding to some truth value of the variable x_i. Since $\mathcal{D}$ hits all the line segments in the instance I_ϕ, then every clause line segment H^c_j $(j = 1, 2, \dots, m)$ is also hit by some disk $D \in \mathcal{D}_i$ such that x_i is in C_j. Then we assign the truth value to x_i such that the value of literal of x_i is true. Hence the clause C_j is also true. Therefore, by considering all such assignments, every clause $C_1, C_2, \dots, C_m$ will be satisfied. □

Theorem 5 *If $OPT(\phi) = m$ then $OPT(I_\phi) = \Sigma^m_{i=1} l_i/2$ and If $OPT(\phi) < (1-\epsilon)m$ then $OPT(I_\phi) > (1+\epsilon')\Sigma^m_{i=1} l_i/2$, where $\epsilon' = \epsilon/36$.*

Proof. First case follows from Lemma 4. We now prove the latter case. Let $\mathcal{D}$ be a feasible solution to the instance I_ϕ. Without loss of generality, we assume that (i) any disk in $\mathcal{D}$ which intersects clause line segment H^c_l also intersects line segments in the cycle $\mathcal{L}_i$ for some variable x_i which occurs in C_l and (ii) no disk in $\mathcal{D}$ can hit only one line segment from $\mathcal{L}_i$ for some variable x_i.

Partition the set $\mathcal{D}$ into $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n$ such that $\mathcal{D}_i \subseteq \mathcal{D}$ is the set of disks hitting the line segments in $\mathcal{L}_i$. Let $V_1 = \{x_i \mid |\mathcal{D}_i| = l_i/2\}$ and $V_2 = \{x_i \mid |\mathcal{D}_i| > l_i/2\}$. Then all the disks in $\mathcal{D}_i$, for $x_i \in V_1$, are corresponding to one of the truth assignment *True or False*. Let $\mathcal{D}'_1 = \mathcal{D}_{i_1} \cup \mathcal{D}_{i_2} \cup \dots \cup \mathcal{D}_{i_k}$ where $x_{i_1}, x_{i_2}, \dots, x_{i_k} \in V_1$ and $\mathcal{D}'_2 = \mathcal{D} \setminus \mathcal{D}'_1$.

For $x_i \in V_1$, we assign the truth value according to the disks in $\mathcal{D}_i$. Then only less than $(1-\epsilon)m$ clauses will be satisfied, since no assignment satisfies more than this number. Hence less than $(1-\epsilon)m$ clause line segments can be hit by disks in $\mathcal{D}'_1$. Therefore, greater than ϵm clause line segments have to be hit by disks in $\mathcal{D}'_2$. Further, $|\mathcal{D}'_2| = \Sigma_{i:x_i \in V_2} \frac{l_i}{2} + \beta$, where $\beta \geq |V_2|$.

We know that every variable is in at most 5 clauses. Even if we pick all the disks in $\mathcal{D}_i$ corresponding to a variable $x_i \in V_2$ then the disks can hit at most 5 clause line segments. Hence if we even pick all disks in the set $\mathcal{D}'_2$, these will hit at most $5|V_2|$ clause line segments, which must be greater than ϵm. Thus $5|V_2| > \epsilon m$. Therefore $\beta \geq |V_2| > \frac{\epsilon m}{5}$.

$$\begin{aligned}\text{Therefore, } |\mathcal{D}| &= |\mathcal{D}'_1| + |\mathcal{D}'_2| \\ &\geq \tfrac{L'}{2} + \beta \text{ where } L' = \Sigma^n_{i=1} l_i/2 \\ &> \tfrac{L'}{2} + \tfrac{\epsilon m}{5} \\ &= \tfrac{L'}{2}(1 + \tfrac{2\epsilon m}{5L'}) \\ &= \tfrac{L'}{2}(1 + \tfrac{2\epsilon m}{5\times 24n}),\ (\ L' \leq 24n) \\ &= \tfrac{L'}{2}(1 + \tfrac{2\epsilon(5n/3)}{5\times 24n})\ ,\ (5n \leq 3m) \\ &= \tfrac{L'}{2}(1 + \tfrac{\epsilon}{36}) \\ &= \tfrac{L'}{2}(1 + \epsilon'), \text{ where } \epsilon' = \tfrac{\epsilon}{36}\end{aligned}$$

Therefore, if $OPT(\phi) < (1-\epsilon)m$ then the optimal solution for instance I_ϕ will have more than $\frac{L'}{2}(1+\epsilon')$ disks, where $\epsilon' = \frac{\epsilon}{36}$. □

2.2 Stabbing Circles with Unit Circles

Theorem 6 *In continuous case, Stab-Cir-Cir is APX-hard.*

Proof. In the above reduction, $MAX-3SAT(5)$ to *Stab-LS-Disk*, we replace every clause line segment with a big circle and place these circles in a nested fashion with sufficient gaps. Formally, let $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m$ be the circles corresponding to the clauses $C_1, C_2, \dots, C_m$. All clause circles $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m$ have a common center, radius of $\mathcal{C}_1$ is very large, and radius of $\mathcal{C}_{i+1}$ is α more than that of $\mathcal{C}_i$ where α is a fixed constant greater than 4. Since the radius of $\mathcal{C}_1$ is very large, we can find a rectangular region where every clause circle looks like a line segment and we can place all the variable cycles in that region. To place a variable cycle, we replace every vertical line segment in the cycle with a circle. The left and right portions of the cycle are placed far apart so that no unit circle can hit both portions. Finally, two circles corresponding to the horizontal line segments at the top and bottom are added. Consecutive variable cycles are placed far apart and the rest of the analysis is similar to the reduction from $MAX - 3SAT(5)$ to *Stab-LS-Disk*. □

2.3 Stabbing Unit Disks with Axis-Parallel Line Segments

We now show that *Stab-Disks-LS* is APX-hard, even if the disks in $\mathcal{X}$ are unit disks and line segments in $\mathcal{Y}$ are axis-parallel. Our reduction uses a special case of set cover, called $SPECIAL - 3SC$. Chan and Grant [4] introduced this problem and proved that it is APX-hard. We give the definition of $SPECIAL - 3SC$ below (we use the same terminology of [4]).

Definition 2.1 *(SPECIAL-3SC) [4] Let* $\mathcal{U} = \{a_1, a_2, \ldots, a_n, w_1, w_2, \ldots, w_m, x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m, z_1, z_2, \ldots, z_m\}$ *be a universe with* $2n = 3m$*. Let* $\mathcal{S}$ *be a collection of* $5m$ *subsets of* $\mathcal{U}$ *such that for each* $1 \le p \le m$ *we can find three integers* $1 \le i < j < k \le n$ *with the sets* $\{a_i, w_p\}$, $\{w_p, x_p\}$, $\{a_j, x_p, y_p\}$, $\{y_p, z_p\}$, *and* $\{a_k, z_p\}$ *in* $\mathcal{S}$*. Further, exactly two sets contain the element* a_t $(1 \le t \le n)$*.*
Set cover with range space $(\mathcal{U}, \mathcal{S})$ *is called SPECIAL-3SC.*

Theorem 7 *Stab-Disks-LS with unit disks and axis-parallel line segments is APX-hard.*

Proof. We will find an instance I' of *Stab-Disks-LS* for every instance I of *SPECIAL-3SC* and will have a one-to-one correspondence between solutions of I and I'.

Consider a unit disk for every element in the universe $\mathcal{U}$ and a horizontal or vertical line segment for every set in $\mathcal{S}$. We place all the unit disks corresponding to $a_1, a_2, \ldots, a_n$ such that their centers are on the same horizontal line with sufficiently large gap. There are n consecutive horizontal strips where the p-th strip contains the disks corresponding to w_p, x_p, y_p, z_p. Within the p-th strip, centers of the disks corresponding to w_p and x_p are on the same horizontal line, the centers of the disks corresponding to x_p and y_p are on the same vertical line, and the centers of the disks corresponding to y_p and z_p are on the same horizontal line (see Figure 6). We know that each of the elements a_i, a_j, a_k appear in exactly two sets of $\mathcal{S}$. For element a_i, if p is the smallest index such that the set $\{a_i, w_p\}$ is in $\mathcal{S}$ then place the center of disk corresponding to the element w_p to the left of the center of the disk corresponding to the element a_i. Otherwise place the center of the disk corresponding to w_p to the right of the center of the disk corresponding to a_i. Do the same for the other two elements a_j and a_k. Finally, add horizontal and vertical line segments as shown in Figure 6 for the five sets $\{a_i, w_p\}$, $\{w_p, x_p\}$, $\{x_p, y_p, a_j\}$, $\{y_p, z_p\}$, and $\{a_k, z_p\}$. □

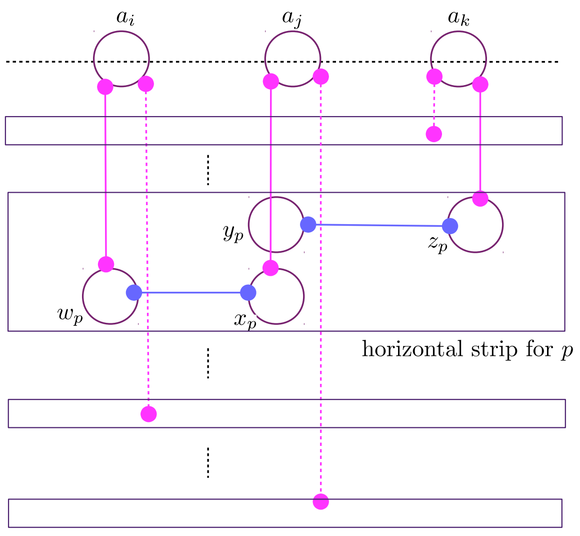


Figure 6: Embedding an instance of *SPECIAL-3SC* into *Stab-Disks-LS*.

3 Polynomial-Time Approximation Schemes

In this section, we apply local search method given in Algorithm 1 which is similar to [11]. As noted by Mustafa and Ray [11], Steps $2 - 5$ will be repeated at most $|\mathcal{Y}|$ times and finding $\mathcal{S}_1$ and $\mathcal{S}_2$ in each such repetition requires us to go over $\binom{|\mathcal{Y}'|}{l}\binom{|\mathcal{Y}'|}{l-1}$ possibilities.

Algorithm 1

```
1: Let $\mathcal{Y}' \subseteq \mathcal{Y}$ be a set of objects which stabs all objects in $\mathcal{X}$.
2: Consider $\mathcal{S}_1 \subseteq \mathcal{Y}'$ and $\mathcal{S}_2 \subseteq \mathcal{Y}$ such that $|\mathcal{S}_1| = l$ and $|\mathcal{S}_2| < l$.
3: if $(\mathcal{Y}' \setminus \mathcal{S}_1) \cup \mathcal{S}_2$ stabs all the objects in $\mathcal{X}$ then
4:     $\mathcal{Y}' \leftarrow (\mathcal{Y}' \setminus \mathcal{S}_1) \cup \mathcal{S}_2$
5: end if
6: Repeat steps 2- 5 till no such $\mathcal{S}_1$ and $\mathcal{S}_2$ exist.
```

Let $\mathcal{R}$ be an optimum solution and $\mathcal{B}$ be a solution returned by Algorithm 1. Without loss of generality, assume that $\mathcal{R} \cap \mathcal{B} = \emptyset$.

Definition 3.1 (Locality Condition [11]) *Let* $\mathcal{G} = (\mathcal{R} \cup \mathcal{B}, \mathcal{E})$ *be a planar bipartite graph with edges between* $\mathcal{R}$ *and* $\mathcal{B}$ *such that for every object* X *in* $\mathcal{X}$ *there exists an edge* (R, B) *in* $\mathcal{E}$ *such that both* $R \in \mathcal{R}$ *and* $B \in \mathcal{B}$ *stab the object* X*. We say that* $\mathcal{G}$ *satisfies locality condition.*

Lemma 8 *[11] Let* $\mathcal{R} \cup \mathcal{B}$ *satisfy the locality condition. Then* $|\mathcal{B}| \le (1 + \epsilon)|\mathcal{R}|$ *where* $\epsilon = O(1/\sqrt{l})$*.*

3.1 A Special Case of *Stab-LS-Disks*

In this section, we consider $\mathcal{X}$ as a set of non-intersecting line segments and $\mathcal{Y}$ as a set of disks. Apply Algorithm 1. Let $\mathcal{R}$ be an optimum solution and $\mathcal{B}$ be the solution return by Algorithm 1. Further, assume $\mathcal{R} \cap \mathcal{B} = \emptyset$. In the following, we call the disks in $\mathcal{R}$ and $\mathcal{B}$ as red and blue disks respectively. Our main contribution is showing that $\mathcal{R} \cup \mathcal{B}$ satisfies locality condition.

3.1.1 Existence of $\mathcal{G} = (\mathcal{R} \cup \mathcal{B}, E)$

In the following, we use $cen(D)$ and $rad(D)$ for center and radius of disk D respectively. We define *weighted Voronoi diagram* similar to [8]. For a disk D, we define $cell(D)$ as the set of points on the plane which are closest to the boundary of D than any other disk boundary. In the following, we use the following properties given by Gibson et al. [8]: (i) Cells are star-shaped and (ii) $cen(D)$ is in only $cell(D)$.

Let $\mathcal{L} = \mathcal{X}$. Partition the set of line segments $\mathcal{L}$ into $\mathcal{L}_p$ and $\mathcal{L}_l$ as follows: for $L \in \mathcal{L}$, if there exist two disks $R \in \mathcal{R}$ and $B \in \mathcal{B}$ such that $L \cap R \cap B \neq \emptyset$ then $L \in \mathcal{L}_p$, otherwise $L \in \mathcal{L}_l$.

In the following, we construct a planar bipartite graph $\mathcal{G} = (\mathcal{R} \cup \mathcal{B}, \mathcal{E}_1 \cup \mathcal{E}_2)$ which satisfies the locality condition for every $L \in \mathcal{L}$. Our construction of $\mathcal{E}_1$ is on the same lines as Wan et al. [12]. Find the weighted Voronoi diagram of disks in $\mathcal{R} \cup \mathcal{B}$. Let $\mathcal{E}_1$ be the set of edges between red and blue cells in the dual graph of this weighted Voronoi diagram. Hence we have the following lemma.

Lemma 9 *[12] The graph $\mathcal{G}_1 = (\mathcal{R} \cup \mathcal{B}, \mathcal{E}_1)$ is planar and bipartite. For each $L \in \mathcal{L}_p$ there exists an edge $(R, B) \in \mathcal{E}_1$ where both $R \in \mathcal{R}$ and $B \in \mathcal{B}$ stab the line segment L at a common point.*

In the following, we use $\overline{ab}$ to represent the line segment with end points a and b.

We now add a set of edges $\mathcal{E}_2$ to the graph $\mathcal{G}$ such that the locality condition for the line segments in $\mathcal{L}_l$ is also satisfied. For every $L \in \mathcal{L}_l$, we find a minimal portion $L' = \overline{rb}$ of line segment L such that $r \in R \cap L, r \in cell(R)$ and $b \in B \cap L, b \in cell(B)$ for some $R \in \mathcal{R}$ and $B \in \mathcal{B}$. We can note that such R and B exist since both $\mathcal{R}$ and $\mathcal{B}$ stab line segment L. Further, the points r and b are on the boundaries of disks R and B respectively. We will add the edge $\overline{cen(R)\ r} \cup \overline{rb} \cup \overline{b\ cen(B)}$ to graph $\mathcal{G}$. We call the set of all such edges $\mathcal{E}_2$. Then we have the following lemma.

Lemma 10 *In the graph $\mathcal{G} = (\mathcal{R} \cup \mathcal{B}, \mathcal{E}_1 \cup \mathcal{E}_2)$, for every $L \in \mathcal{L}$ there exists an edge between some $R \in \mathcal{R}$ and $B \in \mathcal{B}$ such that both R and B hit the line segment L.*

Lemma 11 *The graph $\mathcal{G} = (\mathcal{R} \cup \mathcal{B}, \mathcal{E}_1 \cup \mathcal{E}_2)$ is planar.*

Proof. From Lemma 9, it is clear that no two edges in $\mathcal{E}_1$ intersect. We now prove that no two edges in $\mathcal{E}_2$ intersect. Let e_1 and e_2 be two edges in $\mathcal{E}_2$ which intersect. Assume $e_1 = \overline{cen(R_1)\ r_1} \cup \overline{r_1 b_1} \cup \overline{b_1\ cen(B_1)}$ and $e_2 = \overline{cen(R_2)\ r_2} \cup \overline{r_2 b_2} \cup \overline{b_2\ cen(B_2)}$ for some $R_1, R_2 \in \mathcal{R}$ and $B_1, B_2 \in \mathcal{B}$. Hence $\overline{cen(R_1)\ r_1} \in cell(R_1)$, $\overline{b_1\ cen(B_1)} \in cell(B_1)$, $\overline{cen(R_2)\ r_2} \in cell(R_2)$, and $\overline{b_2\ cen(B_2)} \in cell(B_2)$.

1. $\overline{r_1 b_1}$ and $\overline{r_2 b_2}$ do not intersect since no two line segments in $\mathcal{L}$ intersect.

2. Let $\overline{r_1 b_1}$ intersect $\overline{cen(R_2)\ r_2}$ (or $\overline{b_2\ cen(B_2)}$) at a point p. Then p lies in $cell(R_2)$ (or $cell(B_2)$ respectively). Hence disk R_2 (or B_2 respectively) must stab $\overline{r_1 b_1}$, which is not true. Therefore, $\overline{r_1 b_1}$ does not intersect $\overline{cen(R_2)\ r_2}$ (or $\overline{b_2\ cen(B_2)}$). Similarly, $\overline{r_2 b_2}$ does not intersect $\overline{cen(R_1)\ r_1}$ or $\overline{b_1\ cen(B_1)}$.

3. Further, let $\overline{cen(R_1)\ r_1}$ intersect $\overline{cen(R_2)\ r_2}$ at a point p. Then $cell(R_1)$ and $cell(R_2)$ must have p as a common point which cannot be an interior point of both cells. Further, if p was a boundary point of both $cell(R_1)$ and $cell(R_2)$ then $p = r_1 = r_2$. This is not possible since segments in $\mathcal{L}$ are disjoint. Hence, $\overline{cen(R_1)\ r_1}$ and $\overline{cen(R_2)\ r_2}$ do not intersect. Other cases are symmetric.

Therefore, no two edges in $\mathcal{E}_2$ intersect.

Finally, we will complete the proof by showing that no edge in $\mathcal{E}_1$ intersects an edge in $\mathcal{E}_2$. Let $e_1 = \overline{cen(R_1)\ x} \cup \overline{x\ cen(B_1)} \in \mathcal{E}_1$ where $x \in cell(R_1) \cap cell(B_1)$. Let $e_2 = \overline{cen(R_2)\ r_2} \cup \overline{r_2 b_2} \cup \overline{b_2\ cen(B_2)} \in \mathcal{E}_2$. Other than the above cases, only following two cases are possible which we rule out.

1. $x = r_2$ or $x = b_2$. If $x = r_2$, the disks R_1 and B_1 would have stabbed the line segment L, for which we added e_2 in $\mathcal{E}_2$, at a common point x. But then $L \notin \mathcal{L}_p$, a contradiction. Similarly, $x = b_2$ can be ruled out.

2. $x = cen(R_2)$ or $x = cen(B_2)$. If $x = cen(R_2)$, then x is in only $cell(R_2)$ and cannot be a common point for $cell(R_1)$ and $cell(B_1)$. Hence $x \neq cen(R_2)$. Similarly, $x \neq cen(B_2)$.

Hence the edges e_1 and e_2 cannot intersect. □

3.2 PTAS for *Stab-Disk-Disk*

Theorem 12 *Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets of disks. There exists a PTAS for finding a minimum size subset $\mathcal{Y}' \subseteq \mathcal{Y}$ which stabs all disks in $\mathcal{X}$.*

Proof. Proof is similar to Wan et al. [12]. The dual of weighted Voronoi diagram of disks in $\mathcal{R} \cup \mathcal{B}$ gives a planar bipartite graph which satisfies the locality condition. □

3.3 A Special case of *Stab-Disks-LS*

Theorem 13 *Let $\mathcal{X}$ be a set of disks and $\mathcal{Y}$ be a set of non-intersecting line segments. Then there exists a PTAS to find a minimum cardinality subset $\mathcal{Y}'$ of $\mathcal{Y}$ which stabs all the disks in $\mathcal{X}$.*

Proof. On the same lines of Wan et al. [12], we can prove that the dual of Voronoi diagram of line segments [3] in $\mathcal{R} \cup \mathcal{B}$ is a planar bipartite graph which satisfies the locality condition. $\square$

Acknowledgement

We thank anonymous reviewers for their valuable suggestions which helped to improve the quality of the paper.

References

[1] S. Arora and C. Lund. Hardness of Approximations. *In Approximation Algorithms for NP-hard Problems, Dorit Hochbaum, Ed. PWS Publishing , 1996.*

[2] R. Aschner, M. J. Katz, G. Morgenstern, and Y. Yuditsky. Approximation Schemes for Covering and Packing. WALCOM, pages 89–100, 2013.

[3] F. Aurenhammer. Voronoi diagrams —a Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[4] T. M. Chan and E. Grant. Exact Algorithms and APX-hardness Results for Geometric Packing and Covering Problems. *Computational Geometry*, 47(2):112–124, Feb. 2014.

[5] S. Chaplick, E. Cohen, and G. Morgenstern. Stabbing Polygonal Chains with Rays is Hard to Approximate. *CCCG 2013*.

[6] T. Erlebach and E. J. Van Leeuwen. PTAS for Weighted Set Cover on Unit Squares. APPROX/RANDOM'10, pages 166–177, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal Packing and Covering in the Plane are NP-complete. *Information processing letters*, 12(3):133–137, 1981.

[8] M. Gibson and I. A. Pirwani. Algorithms for Dominating Set in Disk Graphs: Breaking the *log n* Barrier. *ESA 2010: 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010.*, pages 243–254, 2010.

[9] S. Har-Peled. Being Fat and Friendly is Not Enough. *CoRR*, abs/0908.2369, 2009.

[10] M. J. Katz, J. S. Mitchell, and Y. Nir. Orthogonal Segment Stabbing. *Computational Geometry*, 30(2):197 – 205, 2005.

[11] N. H. Mustafa and S. Ray. Improved Results on Geometric Hitting Set Problems. *Discrete & Computational Geometry*, 44(4):883–895, Sept. 2010.

[12] P. Wan, X. Xu, and Z. Wang. Wireless Coverage with Disparate Ranges. In *Proceedings of the 12th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc 2011, Paris, France, May 16-20, 2011*, page 11, 2011.

Critical Placements of a Square or Circle amidst Trajectories for Junction Detection

Ingo van Duijn[*] Irina Kostitsyna[†] Marc van Kreveld[‡] Maarten Löffler[§]

Abstract

Motivated by automated junction recognition in tracking data, we study a problem of placing a square or disc of fixed size in an arrangement of lines or line segments in the plane. We let distances among the intersection points of the lines and line segments with the square or circle define a clustering, and study the complexity of *critical* placements for this clustering. Here critical means that arbitrarily small movements of the placement change the clustering.

A parameter ε defines the granularity of the clustering. Without any assumptions on ε, the critical placements have a trivial $\mathcal{O}(n^4)$ upper bound. When the square or circle has unit size and $0 < \varepsilon < 1$ is given, we show a refined $\mathcal{O}(n^2/\varepsilon^2)$ bound, which is tight in the worst case.

We use our combinatorial bounds to design efficient algorithms to compute junctions. As a proof of concept for our algorithms we have a prototype implementation that showcases their application in a basic visualization of a set of n trajectories and their k most important junctions.

1 Introduction

Many analysis problems in geography have an inherent scale component: the "granularity" or "coarseness" at which the data is studied. The most direct way to model spatial scale in geographic problems is by using a fixed-size neighborhood of locations, such as a fixed-size square or circle. For example, *population density* can be studied at the scale of a city or at the scale of a country, where one may consider the population in units with an area of $10^4\ m^2$ or $25\ km^2$, respectively. There are many other cases where local geographic phenomena are studied at different spatial scales. The Geographical Analysis Machine is an example of a system that supports such analyses on point data sets [10].

In computational geometry, the problem of computing the placement of a square or circle to optimize some measure has received considerable attention. For a set of n points in the plane, one can compute the (fixed-size, fixed-orientation) square that maximizes the number of points inside in $\mathcal{O}(n \log n)$ time (expand every point to a square, and the problem becomes finding a point in the maximum number of squares which is solved by a sweep). Mount et al. [9] study the overlap function of two simple polygons under translation and show, among other things, that one can compute the placement of a square that maximizes the area of overlap with a simple polygon with n vertices in $\mathcal{O}(n^2)$ time. For a weighted subdivision, one can compute a placement that maximizes the weighted area inside in $\mathcal{O}(n^2)$ time as well [2]; this problem is motivated by clustering in aggregated data. In the context of diagram placement on maps, various other measures to minimize or maximize when placing squares were considered, like the total length of border overlap [16].

Our interest lies in a problem concerning trajectory data. A trajectory is represented by a sequence of points with associated time stamps, and models the movement of an entity through space; we will assume in this paper that the movement space is two-dimensional. The identification of "interesting regions" in the plane defined by a collection of trajectories has been studied in several papers recently. These regions can be characterized as meeting places [4], popular or interesting places [1, 5, 11, 12], and stop regions [8]. In several cases, interesting regions are also defined as squares of fixed size, placed suitably. The more algorithmic papers show such regions of interest can typically be computed in $\mathcal{O}(n^2)$ or $\mathcal{O}(n^3)$ time; what is possible of course depends on the precise definition of the problem. There are many other types of problems that can be formulated with trajectory data. For overviews, see [3, 7].

Besides exact algorithms for optimal square or circle placement, approximation algorithms have been developed for several problems on trajectory and other data, see e.g. [4, 6, 15].

Motivation and problem description. We consider a problem on trajectories related to common movements and changes of movement directions at certain places. Imagine a large open space like a town square, a large entrance hall, or a grass field. People tend to traverse such areas in ways that are not random, and the places where a decision is made and possibly a change of direction is initiated may be specific. Also for data like ant tracks, the identification of places where tracks go differ-

[*]MADALGO, Aarhus University, ivd@cs.au.dk
[†]Université libre de Bruxelles, irina.kostitsyna@ulb.ac.be
[‡]Utrecht University m.j.vankreveld@uu.nl
[§]Utrecht University m.loffler@uu.nl

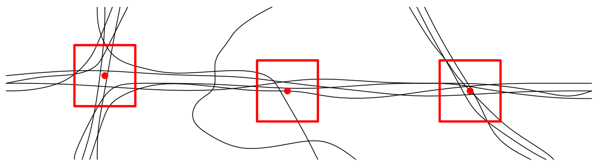


Figure 1: A set of trajectories and square placements at junctions of varying significance.

ent ways is of interest. Without going into details, these observations motivate us to study placements of a square or circle of fixed size such that bundles of incoming and outgoing entities arise.

Consider the tracks in Figure 1. We observe that to define places where the tracks of entities cross and where decisions are made, we can use the placement of a square of a certain size and where the tracks enter and leave the square. We are interested in placements that give rise to bundles: large subsets of tracks that enter and leave the square at a similar place on its boundary, and with a gap to the next location along the boundary where this happens. The left square in the figure has five bundles where one bundle consists of only one trajectory, and the middle and right square have only two bundles (albeit with different "topology"). We see that the left and right squares indicate regions that should be considered more significant than the middle one for being a junction. The main difference between the left and right junctions is that on the left, decisions are made to go straight or change direction, whereas on the right, all moving entities went straight and no different decisions were taken.

We study an abstracted version of this placement problem and ignore many of the practical issues that our simplified definition of a junction would have. Later in the paper we briefly address such issues by giving a slightly more involved definition of a junction. The majority of the paper concentrates on an abstract combinatorial and computational problem that lies at the core of junction detection.

Let $\mathcal{T}$ be a collection of trajectories, let $\square_p$ be the boundary of a unit square S centered at p placed axis-aligned amidst the trajectories, and let ε be a positive real constant. Let $Q = \mathcal{T} \cap \square_p$ be the set of all intersections of trajectories with the boundary of the square (here we can ignore the time component of trajectories; they are considered polygonal lines). Two points in Q are *ε-close* if their distance along $\square_p$ is at most ε. $\mathcal{T}$ and $\square_p$ give rise to a *clustering* of Q on $\square_p$ by the transitive closure relation of ε-closeness. Different clusters are separated by a distance larger than ε along $\square_p$. A single cluster of Q corresponds to parts of trajectories that enter or leave S in each other's proximity (in Figure 1 from left to right the squares define four, three, and four clusters respectively).

Now consider the two-dimensional space of all placements of a square of fixed size by choosing its center as its reference point. We say that a placement q is *critical*, if any arbitrarily small neighborhood of q contains points inducing different clusters. This can be due to a change in size of a cluster on $\square_q$ or to a change in the clustering. The latter corresponds to placements where the distance between two points of $\mathcal{T} \cap \square_p$ on $\square_p$ is exactly ε, with no other points of $\mathcal{T} \cap \square_p$ in between. Since noncritical points are part of a region that defines the same clustering, one can think of the set of critical points to be the boundary between these regions.

Results and organization. In Section 2 we analyze the complexity of the space of critical placements for fixed-size squares in an arrangement of n lines. We show that the placement space has $\mathcal{O}(n^2/\varepsilon^2)$ total complexity in the worst case, and can be constructed in $\mathcal{O}(n^2 \log n + k)$ time where k is the true complexity (the latter appearing in the full version [14] due to space constraints). In Section 4, we show that these results are tight by presenting an explicit construction that exhibits the worst-case behavior. In Section 5 we discuss our application to junction detection further and show output from a prototype implementation. We conclude in Section 6. In the full version of the paper [14] we further show how to extend our approach to more realistic settings, such as placing a square on an arrangement of line segments or placing a circle rather than a square.

2 Square on lines

We begin by studying the simplest version of the problem: placing a unit square over an arrangement of lines. The lines "cut" the boundary of the square into several pieces. We are interested in all placements of the square such that one of these pieces has length exactly ε, in which case we call the piece an *ε-segment* and the placement *critical*. When a piece contains one of the corners of the square, its length is the sum of its two incident line segments. Note that this definition of a critical placement is a simplification of the one defined earlier in terms of clusters; it only considers merging and splitting clusters. In the definition from the introduction, a placement is also critical if a corner of the square coincides with a line. However, these placements are simply four translates of the input lines, so the rest of this section focuses on the harder critical placements as defined here.

2.1 Placement space

Let $\mathcal{L}$ be a set of lines and denote by $\mathcal{A}$ the arrangement of $\mathcal{L}$. For a placement of the square on $\mathcal{A}$, consider all cells that contain part of its boundary. To characterize all critical placements, we look at how the square can be "moved around" such that a given cell of $\mathcal{A}$ contains an ε-segment throughout the motion. For instance, in Figure 2 on the left, we can move the square slightly left and right such that there is always an ε-segment in the same cell. We use the intuition of moving the square to argue that the critical placements can be characterized as a set of line segments, and then prove an upper bound on how many times these line segments can intersect.

Definition 1 *Let $\mathcal{L}$ be a set of lines and $\square_p$ be the boundary of an axis-aligned unit square whose center is denoted by p, and let $\varepsilon > 0$ be a constant. A placement of $\square_p$ (or p) is an ε-*placement *if at least one connected component in $\square_p \setminus \mathcal{L}$ has length exactly ε. An ε-*segment *is a connected component of $\square_p \setminus \mathcal{L}$ with length exactly ε.*

Consider the example from the figure again. When moving p to the right we maintain the indicated ε-segment; this reduces p's movement to one degree of freedom. It can happen that when moving p, another ε-segment is created in another cell. This means that this ε-placement gives rise to two ε-segments. We assume $\mathcal{L}$ to be in general position such that no two ε-segments give p the same allowed movement (a condition that is met after perturbing the input). Therefore no more than two ε-segments will ever occur simultaneously.

Since ε-segments are part of $\square_p$, it is convenient to fix a point p' on $\square_p$ and look at the movement of p' instead of p. Thus, by moving the square such that an ε-segment is maintained inside a cell c in $\mathcal{A}$, the point p' traces a curve. If we consider only those parts of this curve corresponding to placements where p' lies on the ε-segment, we observe that this subcurve is contained in c. For instance, in Figure 2 on the bottom right, the fixed point p' can be moved vertically (p and the square move accordingly) exactly between the intersection points with $\mathcal{L}$. To facilitate our analysis, we will choose a set of fixed points on $\square_p$ such that any ε-segment will contain exactly one of these points. For ease of presentation we assume that $\frac{1}{\varepsilon}$ is integer, so that we can place exactly $\frac{1}{\varepsilon}$ fixed points with distance ε apart along $\square_p$. If it is not integer, we can pick fixed points such that an ε-segment contains one or two such points, in which case our analysis overcounts by a constant factor.

Since these points are fixed with respect to p, we define them in terms of *translation vectors*. We write $\tau(X)$ for the translation by τ of any object X. The inverse of τ is denoted τ^{-1}. Thus, in Figure 2, $p' = \tau(p)$.

Definition 2 *Let $\frac{1}{\varepsilon}$ be integer. T_ε is the set of $\frac{4}{\varepsilon}$ vectors such that $\square_p \setminus \{\tau(p) \mid \tau \in T_\varepsilon\}$ is a set of open line segments each of length ε, and T_ε includes all vectors σ such that $\sigma(p)$ is a corner of $\square_p$.*

Let c be a cell of $\mathcal{A}$ and $\tau \in T_\varepsilon$ a vector. We denote by $S(c, \tau)$ the set of all critical placements p such that

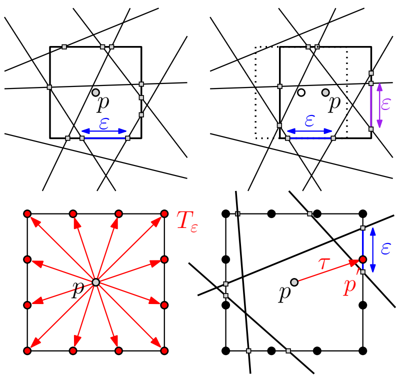


Figure 2: Top: placement of square with one ε-segment (Left) and two ε-segments (Right). Bottom: the set of vectors T_ε (Left) and an ε-placement p with the vector in T_p indicated (Right).

$\tau(p)$ lies on an ε-segment in c. That is, $\tau(S(c, \tau))$ is the set of curves traced out by $\tau(p)$ under our previous interpretation of moving p'. We note that some cells are "too small" to contain an ε-segment, in which case $S(c, \tau)$ is empty; other cells may contain up to two disconnected curves (see Figure 3). As shorthand notation, let $S(\tau)$ denote the union of all $S(c, \tau)$ over all cells in $\mathcal{A}$.

If $\tau(p)$ is not a corner of $\square_p$, then $\tau(S(c, \tau))$ coincides with (one[1] or) two parallel axis-aligned ε-segments contained in c. Therefore, the number of such segments is bounded by twice the number of cells in the arrangement of lines. If $\tau(p)$ *is* a corner of $\square_p$, the shape of $S(c, \tau)$ is more complex. This means that a similar bound on $S(\tau)$ as before is not as easy to achieve. Figure 3 shows the construction of $S(c, \tau)$ when $\tau(p)$ is the upper right corner of $\square_p$. The curve inside c shows exactly how $\tau(p)$ can be moved such that c contains an ε-segment containing $\tau(p)$. Translated by the inverse of τ, we get the actual ε-placements of p corresponding to the particular cell c and vector τ. Note that the figure also shows the only case when $S(c, \tau)$ can be disconnected, i.e. when c contains an acute angle in the same "direction" as τ. We will show that for the four corner vectors τ, the curves in $S(\tau)$ have properties that allow us to bound the complexity of the space of all ε-placements.

Lemma 1 *For $\tau \in T_\varepsilon$ and a cell c in $\mathcal{A}$, $S(c, \tau)$ consists of at most two connected subsets of a piecewise linear and convex curve.*

Proof. If $\tau(p)$ is not a corner point of $\square_p$ then $S(c, \tau)$ is either empty or consists of two single line segments.

Without loss of generality let τ translate p to the upper right corner of $\square_p$. Because the result must hold for arbitrary ε, we define the following function f related to

[1]Degenerate case where the width or height of c is exactly ε.

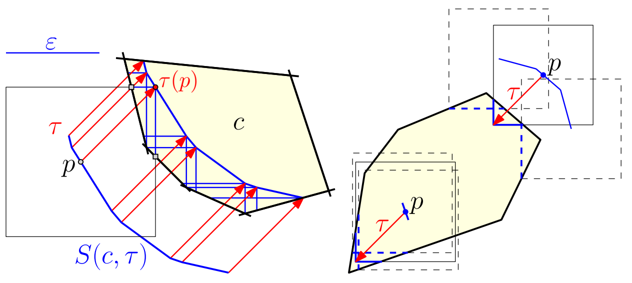


Figure 3: Left: construction of the curve $S(c,\tau)$. Right: $S(c,\tau)$ can consist of multiple pieces.

the notion of an ε-segment, but without being dependent on ε.

For a point a, let x_a be the distance to the closest line in $\mathcal{L}$ left of a, and let y_a be the distance to the closest line below a; define $f(a) = x_a + y_a$. Let f_c be f restricted to the points in a cell c of $\mathcal{A}$, and consider two points a and b inside c. For a point v halfway between a and b, we have $x_v \geq (x_a + x_b)/2$ and $y_v \geq (y_a + y_b)/2$ by the convexity of c. It follows that $f_c(v) \geq (f_c(a) + f_c(b))/2$. Thus, f_c is a concave function inside c, and taking the level set of a concave function yields a convex curve (possibly extending outside c). Observe that the set of all points $a \in c$ such that $f(a) = \varepsilon$ is the same as $S(c,\tau)$ and is a level set of f_c, so it is convex. Similarly, f_c is piecewise linear and hence, so is $S(c,\tau)$. Since there are at most 4 points on the boundary of c where f_c has value ε, $S(c,\tau)$ consists of at most two connected components. □

2.2 Complexity

Lemma 2 *For all $\tau \in T_\varepsilon$, $S(\tau)$ consists of $\mathcal{O}(n^2)$ line segments.*

Proof. For any τ not corresponding to a corner, $S(\tau)$ contains at most two horizontal or two vertical line segments for each cell of $\mathcal{A}$. Next, assume that τ corresponds to a corner of $\square_p$, say, the upper right corner. Let c be a cell with k edges in $\mathcal{A}$, and let p be an ε-placement such that $\tau(p) \in c$. For a fixed y-coordinate of p, there are at most two ε-placements with $\tau(p) \in c$ by convexity of the cell c. Furthermore, for each vertex of $S(c,\tau)$, the point $\tau(p)$ aligns horizontally or vertically with a vertex of c. It follows that the complexity of $S(c,\tau)$ is $\mathcal{O}(k)$. Summing over the complexities of all cells in the arrangement gives the bound. □

We are interested in the complexity of the placement space of $\square_p$, and it is therefore important to know how many times two sets of curves $S(\tau)$ and $S(\sigma)$ intersect, for $\tau, \sigma \in T_\varepsilon$. An intersection of these two sets corresponds to a placement of $\square_p$ such that $\tau(p)$ and $\sigma(p)$ both lie on an ε-segment of $\square_p$. The following lemma shows that the number of intersections is bounded by the complexity of $\mathcal{A}$.

Lemma 3 *For distinct $\tau, \sigma \in T_\varepsilon$, the intersection of $S(\tau)$ and $S(\sigma)$ contains $\mathcal{O}(n^2)$ points.*

Proof. Let p be an ε-placement such that $p \in S(\tau) \cap S(\sigma)$, for distinct $\tau, \sigma \in T_\varepsilon$. Let $\mathcal{A}' = \mathcal{A}\left(\tau^{-1}(\mathcal{L}) \cup \sigma^{-1}(\mathcal{L})\right)$ be the arrangement of two copies of $\mathcal{L}$ translated by the inverses of τ and σ. Let c denote the cell in $\mathcal{A}'$ that contains p, and let c_τ and c_σ denote the cells in $\mathcal{A}$ that respectively contain $\tau(p)$ and $\sigma(p)$ (see Figure 4). Observe that $c = \tau^{-1}(c_\tau) \cap \sigma^{-1}(c_\sigma)$.

Let $s(\tau)$ denote the line segment in $S(c_\tau, \tau)$ on which p lies. Distinguish the following two cases for p: either at least one end point of $s(\tau)$ or $s(\sigma)$ lies in c, or none. In the first case, without loss of generality assume that one end point of $s(\tau)$ lies in c. By Lemma 1, $S(c_\sigma, \sigma)$ consists of at most two convex curves, hence $s(\tau)$ intersects $S(c_\sigma, \sigma)$ at most four times. Because $S(c_\sigma, \sigma)$ is the only part of $S(\sigma)$ intersecting c, it also holds that the part of $s(\tau)$ inside c intersects $S(\sigma)$ only four times. Thus, all intersections of this kind are bounded by the number of vertices in $S(\tau)$, which by Lemma 2 is $\mathcal{O}(n^2)$

In the other case, no end point of $s(\tau)$ or $s(\sigma)$ lies in c. Consider an edge e of c in $\mathcal{A}'$ that intersects $s(\tau)$. Since e intersects $s(\tau)$ it must be an edge of c_σ. Therefore, it intersects $S(c_\tau, \tau)$ and thus $S(\tau)$ at most twice.

Therefore, the number of intersections of this kind is bounded by the number of edges in $\mathcal{A}'$, which is $\mathcal{O}(n^2)$. □

Since there are $\mathcal{O}(\frac{1}{\varepsilon^2})$ pairs of vectors in T_ε, we immediately obtain:

Theorem 4 *Given a set of n lines $\mathcal{L}$, the arrangement of ε-placements of a unit square has complexity $\mathcal{O}(\frac{n^2}{\varepsilon^2})$.*

3 Extensions

Here we describe how to extend the general approach when placing a square on lines to different, more realistic settings. We describe what happens when we replace the line arrangement with an arrangement of line *segments*, denoted $\mathcal{A}(\mathcal{T})$. Additionally, we describe how to adapt the approach when placing a circle rather than a square.

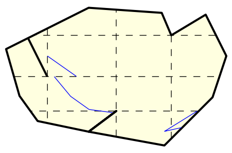


Figure 5: A non-convex cell c, its subdivision, and the (blue) curves $S(c,\tau)$.

3.1 Square on line segments

The definition of an ε-placement remains the same when we place a square amidst line segments, and again we study the complexity of the ε-placement space. In a

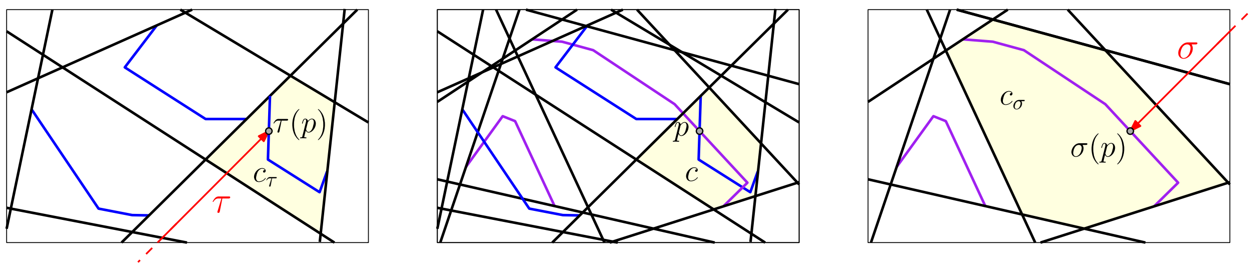


Figure 4: Example arrangement with ε-placements for two vectors τ and σ. Left: $\mathcal{L}$ and $\tau(S(\tau))$. Right: $\mathcal{L}$ and $\sigma(S(\sigma))$. Center: Both combined under appropriate translations.

nonconvex cell c of the arrangement, the curve $S(c, \tau)$ is no longer a piecewise linear convex curve, but it can have several disconnected pieces; see Figure 5.

Theorem 5 *Given a set of line segments $\mathcal{T}$, the arrangement of ε-placements of a unit square has complexity $\mathcal{O}(\frac{n^2}{\varepsilon^2})$.*

Proof. For analysis purposes, we imagine that any nonconvex cell c of $\mathcal{A}(\mathcal{T})$ is partitioned into convex subcells as follows. For each endpoint of a line segment, shoot a ray in each orthogonal direction inside the cell until it hits another line segment. Use these rays to subdivide the cell into convex subcells, see Figure 5. If a cell c has k endpoints in its boundary, it is partitioned into $\mathcal{O}(k^2)$ subcells. Within each subcell of c, the curve $S(c, \tau)$ has the same properties as in a convex cell, and hence we can apply the same arguments as before. The total number of endpoints is $\mathcal{O}(n)$, and hence the total number of subcells analyzed in the arrangement $\mathcal{A}(\mathcal{T})$ is $\mathcal{O}(n^2)$. The bound follows. $\square$

3.2 Circle on lines

We now place a unit circle, $\bigcirc_p$, on $\mathcal{L}$, rather than a square. We again define a set of translation vectors T_ε to points ε-spaced on the boundary of $\bigcirc_p$, this time assuming $2\pi/\varepsilon$ is an integer. Consider cell c in the arrangement $\mathcal{A}$. The segments in $S(c, \tau)$ are no longer straight, but are generally elliptic.

Lemma 6 *The ε-placements of a circle such that the corresponding ε-segments lie in c form a collection of segments and elliptic arcs.*

Proof. Consider two lines ℓ_1 and $\ell_2 \in \mathcal{L}$. Consider a coordinate system with the origin in the intersection point of ℓ_1 and ℓ_2 and oriented such that ℓ_1 has equation $y = ax$, and ℓ_2 has equation $y = -ax$. Consider an ε-placement of a unit circle such that the ε-segment is in the I- and IV-quadrants below line ℓ_1 and above ℓ_2 as in Figure 6.

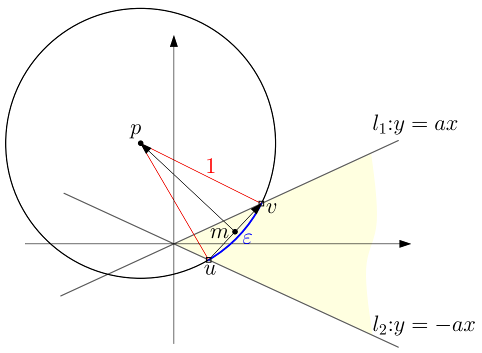


Figure 6: Two lines $\ell_1, \ell_2 \in \mathcal{L}$ and an ε-placement of p such that the corresponding ε-segment is in the highlighted region.

Let u be the intersection point of the circle and line ℓ_2, v be the intersection of the circle and ℓ_1, and m be a middle point between u and v. From the following equations:

$$|uv|^2 = 2 - 2\cos\varepsilon\,,$$
$$|up| = 1\,,$$
$$|vp| = 1\,,$$
$$\vec{up} \times \vec{mp} = |up| \cdot |mp|\,,$$

we can derive that the segment of the ε-placement curve that corresponds to all such ε-segments that have one end point on line ℓ_1 and another end on ℓ_2 is an arc of an ellipse given by the following equation (refer to Figure 7):

$$\frac{a^2x^2}{\left(\sin\left(\frac{\varepsilon}{2}\right) - a\cos\left(\frac{\varepsilon}{2}\right)\right)^2} + \frac{y^2}{\left(a\sin\left(\frac{\varepsilon}{2}\right) + \cos\left(\frac{\varepsilon}{2}\right)\right)^2} = 1\,.$$

$\square$

From this analysis we see that a sequence of adjacent arcs is not necessarily convex: when $a > \tan\frac{\varepsilon}{2}$ the point p is tracing the left arc of an ellipse (the convex part), when $a < \tan\frac{\varepsilon}{2}$ point p is tracing the right arc of an ellipse (the concave part), and when $a = \tan\frac{\varepsilon}{2}$ the ellipse

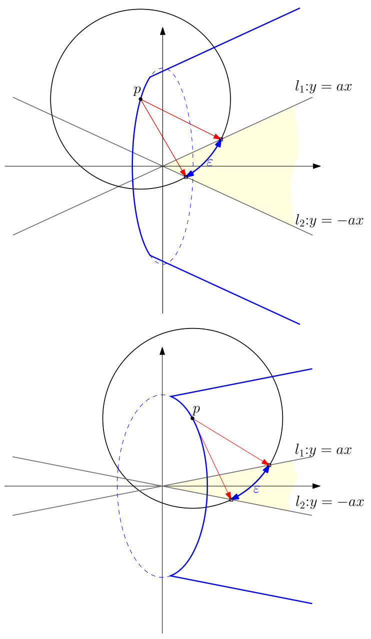


Figure 7: Two lines $\ell_1, \ell_2 \in \mathcal{L}$ and the curve (blue) that p traces over all ε-placements such that the corresponding ε-segments are within the highlighted region.

degenerates to a straight segment. Nonetheless, we can show that this cannot happen arbitrarily often.

Lemma 7 *For $\varepsilon < 1$, $S(c, \tau)$ consists of a constant number of piece-wise elliptic convex curves.*

Proof. Consider the sequence of angles $a_1, a_2, \ldots, a_k$ between consecutive edges of c. Because c is convex, we know that $\sum_{i=1}^{k}(\pi - a_i) = 2\pi$. For sufficiently small ε ($\varepsilon < 1$ suffices), this implies that no more than two angles a_i can be smaller than $\pi - \tan\frac{\varepsilon}{2}$.

By the previous lemma, these at most two angles correspond to concave elliptic arcs, which we report as separate curves. The remainder of the arcs and segments formed by boundary of c is subdivided by these gaps into at most two piece-wise elliptic convex curves.

As in the case for squares (refer to Lemma 1), at most two disconnected curves, each of which can be decomposed into at most four convex subcurves, may exist for any given τ. To see this, consider any line with direction τ that intersects c in a segment s. As we slide $\tau(p)$ along s, the length of the boundary piece of $\bigcirc_p$ containing the point $\tau(p)$ in c changes as a concave function. Hence, the piece is an ε-segment at most twice. (Note that, in contrast to the square case, it is essential that s has orientation τ.) $\square$

To bound the complexity of the placement space, it is sufficient to bound the number of intersection points between $S(\tau)$ and $S(\sigma)$.

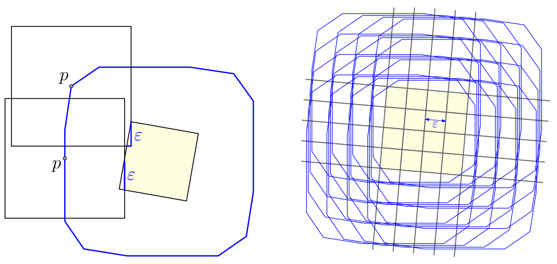


Figure 8: Lower bound construction for squares. (left) Point p traces an "offset" around square cell. (right) A $\frac{n}{2} \times \frac{n}{2}$ grid formed by lines from $\mathcal{L}$. Each cell has size $\mathcal{O}(\varepsilon)$.

Lemma 8 *For distinct $\tau, \sigma \in T_\varepsilon$, the intersection of $S(\tau)$ and $S(\sigma)$ contains $\mathcal{O}(n^2)$ points.*

Proof. Let $\mathcal{A}'$ be the overlay of $\tau^{-1}(\mathcal{A})$ and $\sigma^{-1}(\mathcal{A})$: the arrangement of two copies of $\mathcal{L}$ translated by the inverses of τ and σ. Let c be a cell in $\mathcal{A}'$, let c_τ and c_σ denote the cells in $\mathcal{A}$ which respectively contain $\tau(p)$ and $\sigma(p)$. Observe that $c = \tau^{-1}(c_\tau) \cap \sigma^{-1}(c_\sigma)$.

Now, by Lemma 7, c_τ and c_σ both contain a constant number of (pieces of) convex curves. Each curve piece itself may consist of many elliptic arcs. We observe that a convex curve, consisting of k elliptic arcs, and a second convex curve, consisting of h elliptic arcs, may cause at most $\mathcal{O}(k + h)$ intersections. We thus charge the intersections to the pieces of $S(\tau)$ or $S(\sigma)$, and note that each piece is charged at most constantly often. $\square$

As in the case of squares, we have $\mathcal{O}(1/\varepsilon^2)$ distinct translation vectors, so we can bound the total complexity by $\mathcal{O}(n^2/\varepsilon^2)$.

Theorem 9 *Given a set of n lines $\mathcal{L}$, the arrangement of ε-placements of a unit circle has complexity $\mathcal{O}(\frac{n^2}{\varepsilon^2})$.*

4 Lower bounds

By a worst case construction we show a tight bound on the complexity of the placement space of a square (the construction for placing a circle is analogous). Consider a (slightly tilted) square cell of size $\mathcal{O}(\varepsilon)$. The curve traced by p of all ε-placements of a unit side square such that an ε-segment is inside the cell is shown in Figure 8. It forms an "offset" curve around the cell of width ≈ 2. Place $\frac{n}{2}$ almost horizontal lines and $\frac{n}{2}$ almost vertical lines to form a grid with cells of size $\mathcal{O}(\varepsilon)$ (consider all lines slightly tilted). Figure 8 shows this construction.

A trivial upper bound[2] on the complexity of the placement space is $\mathcal{O}(n^4)$, which by our construction is worst

[2]If ε is arbitrarily small, the arrangement of the placement space is a collection of n^2 unit squares which can intersect pairwise.

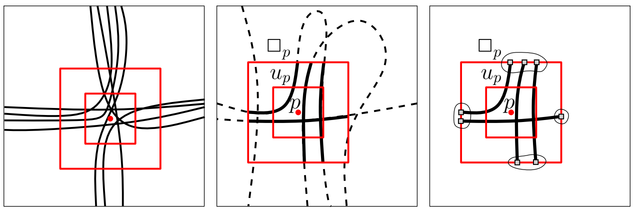


Figure 9: Left: Junction-like point with its two concentric squares. Middle: Solid subtrajectories are salient for the junction. Right: ε-clustering of the salient subtrajectory endpoints, revealing four arms.

case tight if $\varepsilon = \mathcal{O}(\frac{1}{n})$. If ε is bounded from below, the parametrized complexity of the arrangement is also worst-case tight.

Corollary 10 *Given a set of n lines $\mathcal{L}$ and a parameter $\varepsilon = \Omega(\frac{1}{n})$, the arrangement of ε-placements has a worst case complexity $\Omega(\frac{n^2}{\varepsilon^2})$.*

5 Applications to junction detection

While junctions are normally features of (road) networks, our objective is to extract junctions purely based on trajectory data. This allows us to identify regions that serve as junctions in open spaces like city squares, or to identify places where animals pass by and choose one or the other direction.

We describe basic properties of a junction, which motivate corresponding definitions. Our approach is to treat any point in the plane as a potential junction; we define when a point is *junction-like* depending on the trajectories in its neighborhood. We show that the arrangement from Section 2 can be used to partition the plane into regions of points that are similarly junction-like, that is to say they have the same basic properties. In the full version of the paper [14] we present—as a proof of concept—the output of a prototype implementation run on idealized data.

Properties of a junction. A junction is a region where trajectories enter and leave in a limited number of directions or routes, called *arms* of the junction. While the moving entities initiate a direction to leave the junction in the core area, the arms are only "visible" somewhat further away from the core; see Figure 9. This motivates the use of two concentric squares to decide whether a point is junction-like. The way in which the trajectories enter and leave the two squares determines whether the point is junction-like and how significant that junction is. Since we can scale the plane with all trajectories, we can assume that the larger square is unit-size. We let the smaller square have side length $\frac{1}{2}$.

For a point p in the plane, let u_p and $\square_p$ denote the boundaries of squares with side length $\frac{1}{2}$ and 1, respectively, both centered at p.

Definition 3 *A (sub)trajectory T is* salient *for a point p if it lies completely inside $\square_p$, it intersects u_p, and its endpoints lie on $\square_p$.*

From a set $\mathcal{T}$ of trajectories we consider all subtrajectories that are salient; see Figure 9. Such subtrajectories should be maximal: their endpoints must be actual crossings with the boundary of $\square_p$ and not just contacts. A single trajectory in $\mathcal{T}$ can have multiple salient subtrajectories.

Definition 4 *Given a set of points Q on the boundary of a square $\square_p$ and a constant $\varepsilon > 0$, an* ε-clustering *is a partitioning of Q such that for any two points $q, q' \in Q$ belonging to the same cluster, there exists a sequence of points $q = q_1, \ldots, q_j = q'$, all in Q, such that q_i and q_{i+1} for $1 \le i < j$ have distance at most ε along $\square_p$.*

Definition 5 *Let p be a point in the plane, let $\mathcal{T}$ be a set of trajectories, let $\varepsilon > 0$, and let $Q_p(\mathcal{T})$ be the set of endpoints of all salient subtrajectories from $\mathcal{T}$. A point p is* junction-like *if an ε-clustering of $Q_p(\mathcal{T})$ has at least three clusters.*

When we move the point p over the plane, these clusters can grow, shrink, merge and split. A cluster may for instance split when two consecutive intersection points from $Q_p(\mathcal{T})$ become more separated than ε. A cluster may shrink because a subtrajectory is no longer salient, which then may also cause a split of a cluster.

It should be clear that we can compute a subdivision of the plane into maximal cells where the ε-clustering is the same. It should also be clear that the theory of Section 2 discusses a simplified version of this problem where salience is not considered.

Implementation and results. There are many ways to obtain junctions from the junction-like points and their ε-clusterings. There are also many ways to attach a significance to a junction, based on the number of clusters and their sizes. Moreover, we may want to distinguish between *real junctions* and *crossings*, where a crossing is a junction-like region with four arms where all trajectories go straight, and a real junction has several different splits and merges of trajectories over the clusters. For both types we can define their significance in various ways. This is beyond the scope of this paper; see [13, 14] for further discussions.

6 Conclusions and further research

We analyzed the complexity of the placement space of a unit square or circle in an arrangement of lines or line segments, a problem that lies at the core of junction detection in trajectory analysis. Our results include upper bounds that improve on the naive $O(n^4)$ bound by showing that two of the linear factors in fact only need to depend on $1/\varepsilon$. Increasing the number of trajectories is not a reason to increase the granularity of the clustering, so in practice we expect $1/\varepsilon$ to be much smaller than n, if not constant. The resulting $\Theta(n^2/\varepsilon^2)$ bound is tight in the worst case. The combinatorial problem is interesting from the theoretical perspective because it combines arrangements with distances in the arrangement.

A prototype implementation applies the result to junction detection. While the implementation demonstrates the value of the approach and indicates that the time complexity is not prohibitive, several simplifications of the problem have been made and no extensive experiments have been performed. In a follow-up study, it would be interesting to augment the implementation with circles or different shapes, and to run the algorithm on real-world trajectory data from various sources.

From a theoretical perspective, it would be interesting to explore further extensions of the approach (i.e. curved trajectories, trajectories embedded in higher-dimensional spaces), or apply it to other distance based arrangements.

Acknowledgments

M.L. is supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. MADALGO Center for Massive Data Algorithmics is supported in part by the Danish National Research Foundation grant DNRF 84.

References

[1] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *Int. J. Comput. Geometry Appl.*, 20(1):19–42, 2010.

[2] K. Buchin, M. Buchin, M. van Kreveld, M. Löffler, J. Luo, and R. I. Silveira. Processing aggregated data: the location of clusters in health data. *GeoInformatica*, 16(3):497–521, 2012.

[3] J. Gudmundsson, P. Laube, and T. Wolle. Movement patterns in spatio-temporal data. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, pages 726–732. Springer, 2008.

[4] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *14th ACM International Symposium on Geographic Information Systems*, pages 35–42, 2006.

[5] J. Gudmundsson, M. van Kreveld, and F. Staals. Algorithms for hotspot computation on trajectory data. In *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 134–143, 2013.

[6] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing circle. *Algorithmica*, 41(3):147–157, 2005.

[7] H. Jeung, M. Yiu, and C. Jensen. Trajectory pattern mining. In Y. Zheng and X. Zhou, editors, *Computing with Spatial Trajectories*, pages 143–177. Springer, 2011.

[8] B. Moreno, V.C. Times, C. Renso, and V. Bogorny. Looking inside the stops of trajectories of moving objects. In *Proc. XI Brazilian Symposium on Geoinformatics*, pages 9–20. MCT/INPE, 2010.

[9] D. M. Mount, R. Silverman, and A. Y. Wu. On the area of overlap of translated polygons. *Computer Vision and Image Understanding*, 64(1):53–61, 1996.

[10] S. Openshaw, M. Charlton, C. Wymer, and A. Craft. A mark 1 geographical analysis machine for the automated analysis of point data sets. *International Journal of Geographical Information System*, 1(4):335–358, 1987.

[11] A.T. Palma, V. Bogorny, B. Kuijpers, and L. Otávio Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proc. 2008 ACM Symposium on Applied Computing*, pages 863–868, 2008.

[12] S. Tiwari and S. Kaushik. Mining popular places in a geo-spatial region based on GPS data using semantic information. In *Proc. 8th Workshop on Databases in Networked Information Systems*, volume 7813 of *LNCS*, pages 262–276, 2013.

[13] I. van Duijn. Pattern extraction in trajectories and its use in enriching visualisations. Master's thesis, Department of Information and Computing Sciences, Utrecht University, 2014. http://dspace.library.uu.nl/handle/1874/294075.

[14] I. van Duijn, I. Kostitsyna, M. van Kreveld, and M. Löffler. Critical placements of a square or circle amidst trajectories for junction detection. *CoRR*, abs/1607.05347, 2016.

[15] S. van Hagen and M. van Kreveld. Placing text boxes on graphs. In *Graph Drawing, 16th International Symposium*, volume 5417 of *Lecture Notes in Computer Science*, pages 284–295. Springer, 2008.

[16] M. van Kreveld, É. Schramm, and A. Wolff. Algorithms for the placement of diagrams on maps. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 222–231. ACM, 2004.

Smallest Paths with Restricted Orientations in Simple Polygons*

Jillian Dicker† Joseph G. Peters†

Abstract

Finding a *smallest path* - a path that minimizes both length and link distance - between two points in a polygon has been previously studied in both the unrestricted and rectilinear cases. In this paper we bridge the gap between the unrestricted and rectilinear cases by considering paths whose segments are limited to a given set of orientations. We prove that a smallest path between two points in an arbitrarily-oriented simple polygon always exists when the path is restricted to two arbitrary orientations. We also show that when three or more orientations are allowed, a smallest path may not exist.

1 Introduction

We consider the problem of finding a *smallest path* - a path that simultaneously achieves minimum length and minimum link distance (number of segments in the path) - connecting two points s and t where both the path and the points are contained in an arbitrarily-oriented simple polygon. Past results for this problem typically focussed on either the case where the orientations of the segments of the path are not restricted, or the case where the path is constrained to be rectilinear (consisting of only horizontal and vertical segments).

Here we consider a restricted set $\mathcal{O}$ of orientations and seek a path whose line segments are oriented according to $\mathcal{O}$. A path consisting of segments with orientations contained in $\mathcal{O}$ is called an $\mathcal{O}$-path. In this paper we unify previous results by exploring the problem of finding a smallest $\mathcal{O}$-path connecting two points s and t.

An application of smallest paths is in VLSI design where it is desirable to minimize both the length of a path and the number of *vias* (i.e. places where a path changes direction) [8]. In the past, VLSI design was restricted to orthogonal orientations, but now it is often possible to use a larger (finite) number of orientations. Thus it is now practical to define a set $\mathcal{O}$ of orientations for which we aim to find smallest $\mathcal{O}$-paths between pairs of points. Another application is in the area of motion planning where a robot with limited directions of travel could select directions that minimize path length.

Restricted Orientation Geometry, introduced by Guting [4] and expanded upon by Rawlins [11], attempts to bridge the gap between arbitrarily-oriented geometry and rectilinear geometry. While this area remains relatively unstudied, there have been some developments concerning minimum length and minimum link distance paths where the orientations of the path segments are restricted to some set $\mathcal{O}$. Finding a minimum length $\mathcal{O}$-path that avoids a set of non-intersecting polygonal obstacles has been solved by Reich [12]. Nilsson et al [10] gave an algorithm that finds a minimum length $\mathcal{O}$-path that avoids a set of $\mathcal{O}$-oriented polygonal obstacles when $\mathcal{O}$ is constrained to three allowable orientations. Mitchell et al [9] provided an algorithm that finds a minimum-link $\mathcal{O}$-path between two points in an arbitrarily-oriented polygon with polygonal obstacles.

Several results proving that a smallest rectilinear path always exists between two points in a polygon have been published. McDonald [7] showed that a smallest rectilinear path always exists between any two points in a simple polygon, and provided an algorithm to find such a path. De Berg [2] and McDonald and Peters [8] independently proved a similar result for the less general problem of finding a rectilinear path between two points in a simple rectilinear polygon. Maheshwari and Sack [6] later found another algorithm to find a smallest rectilinear path between two points in a rectilinear polygon.

In a paper more closely related to this paper, Hershberger and Snoeyink [5] looked at finding a minimum length $\mathcal{O}$-path between two points in a simple $\mathcal{O}$-oriented polygon and proved that a smallest path always exists when there are no more than three *directions* allowed. However we must be careful to note the difference between the terms *direction* and *orientation*; each orientation has two associated directions. For example, the rectilinear case has two orientations - horizontal and vertical, but four directions - up, down, left and right. Therefore the proof given in [5] does not prove that there always exists a smallest path for two orientations.

Speckmann and Verbeek investigated homotopic routing in both the rectilinear [13] and $\mathcal{O}$-oriented [14] cases. Two paths are homotopic if one can be "continuously deformed" into the other without passing over any obstacles. In [14], it is shown that for every path through $\mathcal{O}$-oriented obstacles, there is a smallest "smooth" $\mathcal{O}$-oriented path that is homotopic. (A "smooth" path does not skip an orientation when it changes direction.) Two differences from our results are that we consider paths that are contained in arbitrarily-

*This work was supported in part by NSERC of Canada.

†School of Computing Science, Simon Fraser University, Burnaby, BC, Canada

oriented polygons and our polygons are simple (so no obstacles).

A number of papers have used *extreme vertices* - vertices that any minimum length rectilinear path must contain - as a way to find minimum length rectilinear paths [1, 7, 16]. Yang et al [16] defined an *extreme sequence* to be a list of extreme vertices in the order that they must be traversed between s and t. They used extreme sequences to investigate the problem of finding two non-crossing smallest rectilinear paths that connect two pairs of points in a rectilinear polygon. In this paper, we find smallest paths by finding an extreme sequence and then finding shortest paths between each consecutive pair of extreme vertices in the sequence.

We will prove that for any given pair of orientations $\mathcal{O} = \{\theta_1, \theta_2\}$, there must exist a smallest $\mathcal{O}$-path between two points in any arbitrarily-oriented simple polygon P where neither of the two points is on the boundary of P. Additionally, we show that for any set of orientations with $|\mathcal{O}| \geq 3$, there does not always exist a smallest path.

2 Definitions

Definition 1 *A polygon P is defined by a set of n vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a set of line segments $E = \{e_1, e_2, \ldots, e_n\}$ where $e_j = \overline{v_j\, v_{j+1}}$, $1 \leq j \leq n-1$, and $e_n = \overline{v_n\, v_1}$. Let the vertices be labelled in a counterclockwise order around the boundary of P which is denoted $bd(P)$.*

Definition 2 *A* simple polygon *$P = (V, E)$ is a polygon where E is not self-intersecting and no three consecutive vertices are co-linear. In this paper, all polygons are assumed to be closed and simple. A polygon contains all points in $bd(P)$ and the region inside $bd(P)$.*

Definition 3 *For a point z in the plane, z_x and z_y denote the x- the y-coordinates of z, respectively. The* length of a line segment *$\overline{xy}$, denoted by $\ell(\overline{xy})$, is the Euclidean length of $\overline{xy}$.*

Definition 4 *A line segment $l = \overline{uv}$ where $u_y < v_y$ is* α-oriented *if $\alpha \in (0, \pi]$ is the counterclockwise angle l makes with the horizontal ray beginning at u travelling in the positive direction (see Figure 1). A line segment l is* ϕ-directed *if $\phi \in (0, 2\pi]$ is the counterclockwise angle $\overline{uv}$ makes with a horizontal ray beginning at u travelling in the positive direction. A line segment is said to be* $\mathcal{O}$-oriented *if its orientation is contained in $\mathcal{O}$. Let $\theta(u, v)$ denote the orientation of the line $\overline{uv}$ and let $\theta_{dir}(u, v)$ denote the direction of the line $\overline{uv}$ from u to v.*

Definition 5 *An* (α, β)-path *consists of a finite sequence of line segments where each consecutive pair of segments shares an endpoint and each segment is either α-oriented or β-oriented. An* $\mathcal{O}$-path *is a path consisting of a finite number of $\mathcal{O}$-oriented line segments. Let $\ell_{\alpha,\beta}(u, v)$ denote the length of the shortest (α, β)-path between u and v.*

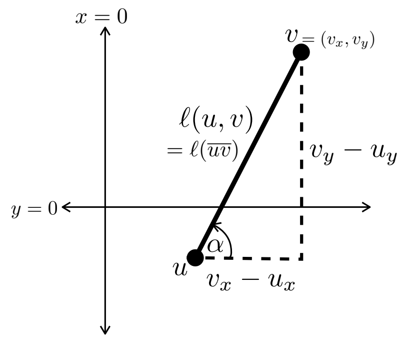


Figure 1: The orientation of $\overline{uv}$ is α, that is $\theta(u, v) = \alpha$. The direction of $\overline{vu}$ is $\theta_{dir}(v, u) = \pi + \alpha$.

Definition 6 *For any point v in polygon P and $\phi \in (0, 2\pi]$, let the* projection point *from v onto P, denoted $Pr_\phi(v, P)$, be $x \in bd(P)$ such that $\overline{vx}$ is contained in P, is ϕ-directed, and is as long as possible. If $Pr_\phi(v, P) = v$, the projection point is said to be* degenerate *(see Figure 2). When the context is clear, we will use $Pr_\phi(v)$ to mean $Pr_\phi(v, P)$.*

Definition 7 *For simple polygon P, a* chord *is a line segment $\overline{pq}$ that is contained in P with $p, q \in bd(P)$. A* maximal chord *is a chord that is as long as possible. For any $v \in V$ and $\phi \in (0, 2\pi]$, let $C(v, \phi)$ denote the chord $\overline{vp}$ in P where $p = Pr_\phi(v)$ (see Figure 2).*

Definition 8 *Let $S = \{s_1, \ldots, s_k\}$ with $s_i = \overline{v_i\, v_{i+1}}$, $1 \leq i \leq k$, be a set of line segments that define a path. For two points $p, q \in S$, let the path along S from p to q be denoted $S(p \to q)$. If p and q both lie on the same segment, that is $\exists s_i \in S$, $p, q \in s_i$, then $S(p \to q) = \overline{pq}$. Otherwise p and q are on distinct segments $s_i, s_j \in S$ and $S(p \to q) = \overline{p\, v_{i+1}} \cup \{s_{i+1}, \ldots, s_{j-1}\} \cup \overline{v_j\, q}$. See Figure 2.*

Definition 9 *A* U-turn *is a path consisting of three line segments s_1, s_2, and s_3 such that s_1 and s_3 lie on the same side of the line containing s_2. A* staircase path *is a path containing no U-turns. See Figure 3.*

3 Smallest Paths for Two Orientations

In this section, we prove our main result that for any simple polygon P, any two points $s, t \in P \setminus bd(P)$, and any two orientations α and β, there is a smallest (α, β)-path contained in P between s and t. We begin by

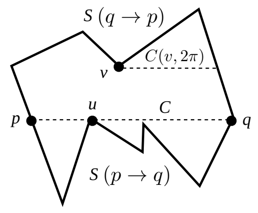


Figure 2: The points p and q are the projection points $Pr_{\pi}(u)$ and $Pr_{2\pi}(u)$, respectively. The projection point $Pr_{\frac{3\pi}{2}}(u)$ is degenerate. $C(v, 2\pi)$ is a chord but is not a maximal chord, while C is the π-oriented maximal chord through u. The boundary of the polygon is made up of two subpaths $S(p \to q)$ and $S(q \to p)$.

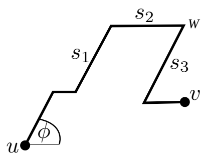


Figure 3: The segments s_1, s_2, s_3 form a U-turn on the (ϕ, π)-path from u to v. The subpath from u to w is a staircase path.

defining the concept of an *extreme vertex* and showing that any shortest $s - t$ path must include all extreme vertices.

Definition 10 *For $\alpha, \beta \in (0, 2\pi]$ where $\alpha \neq \beta$, and $v \in V$, P is divided into three regions by the chords $C(v, \alpha)$ and $C(v, \beta)$ if and only if both $Pr_{\alpha}(v)$ and $Pr_{\beta}(v)$ are non-degenerate. A region is called the* middle region *if it is bounded by both chords. The other two regions are called* end regions*. See Figure 4. The chords themselves are considered to belong to the end regions and each end region is* induced *by the chord that bounds it.*

Definition 11 *A vertex $v \in V$ is* extreme *with respect to orientation $\alpha \in (0, \pi]$ if s and t are in different end regions induced by $C(v, \alpha)$ and $C(v, \alpha + \pi)$. See Figure 4. A vertex $v \in V$ is extreme with respect to $\mathcal{O}$ if v is extreme with respect to some $\theta_i \in \mathcal{O}$.*

Definition 12 *Given $\mathcal{O}$, let $\theta_1, \theta_2, \ldots, \theta_n$ be the sequence of all orientations in $\mathcal{O}$ ordered counterclockwise. The orientations $\theta_i, \theta_j \in \mathcal{O}$ that are before and after ϕ in the counterclockwise cyclic order are the* neighbouring orientations *of ϕ.*

Lemma 1 [15] *For any two points x, y let $\theta_i, \theta_j \in \mathcal{O}$ be the two neighbouring orientations of $\theta(x, y)$ where $\theta_i = \theta_j = \theta(x, y)$ if $\theta(x, y) \in \mathcal{O}$. Then a shortest path (not necessarily in P) between x and y is a staircase path consisting of segments with orientations θ_i and θ_j.*

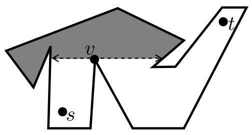


Figure 4: The shaded region is the middle region induced by $C(v, \pi)$ and $C(v, 2\pi)$, and the unshaded regions are the end regions induced by the chords. The vertex v is extreme with respect to π since s and t are in different end regions.

Lemma 2 *If S is a shortest path between two points s and t, then for any two points $x, y \in S$ the subpath $S(x \to y)$ is a shortest path between x and y.*

Proof. If $S(x \to y)$ is not a shortest path between x and y, then there is some other path Q between x and y that is shorter. It follows that the path $S(s \to x) \cup Q \cup S(y \to t)$ is shorter than S, which means that S was not a shortest path between s and t. $\square$

Theorem 3 (Extreme Point Theorem). *Given points $s, t \in P$, if $v \in V$ is an extreme vertex with respect to $\mathcal{O}$ then a shortest $\mathcal{O}$-path from s to t must contain v.*

Proof. Let S be a shortest $\mathcal{O}$-path from s to t. Since v is extreme, s and t are in different end regions induced by two chords in P. This means that any path from s to t must pass through both chords. Let x be the point where S first intersects the first chord, and x' be the last point where S intersects the second chord. Note that by definition of an extreme point, the directions of the two chords are π apart and both intersect v, so they will actually form a single chord across P. This means that the line $\overline{xx'}$ will lie entirely in P. Now S is made up of three subpaths: $S(s \to x)$, $S(x \to x')$, and $S(x' \to t)$. By Lemma 2, a shortest path is made up of shortest subpaths, so $S(x \to x') = \overline{xx'}$ which clearly contains v. $\square$

Lemma 4 *For the set of all vertices that are extreme with respect to $\mathcal{O}$, there is exactly one order in which the vertices are traversed on any shortest $\mathcal{O}$-path from s to t.*

Proof. Since all extreme vertices must be included on any shortest path from s to t, there is at least one order. Let X be the sequence comprised of the extreme vertices in that order together with $s, t \in X$ as the first and last points in X, and let S be a shortest path from s to t. If X contains less than two extreme vertices, then we are done. Otherwise, take any two extreme vertices

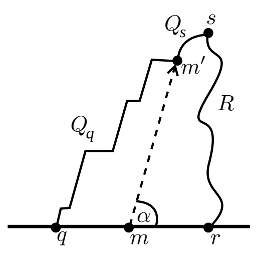


Figure 5: Configuration described in Lemma 5.

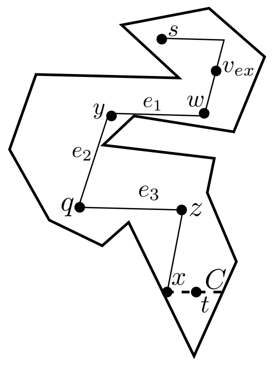


Figure 6: Configuration described in Lemma 7.

$u, v \in X$ where u precedes v in X. Since S is a shortest path, the subpaths $S(s \to u)$, $S(u \to v)$, and $S(v \to t)$ are all shortest subpaths. Suppose that there exists a shortest path S' that traverses u and v in the opposite order. Then $S'(s \to v)$, $S'(v \to u)$, and $S'(u \to t)$ are all shortest subpaths. Since $S'(s \to v)$ is a shortest subpath, it is just as short as $S(s \to u) \cup S(u \to v)$ and we can construct a shortest path $S'(s \to v) \cup S(v \to t)$ from s to t that doesn't contain u. $\square$

To prove that there always exists a smallest (θ_1, θ_2)-path from s to t, we first prove that there always exists a smallest (θ_1, θ_2)-path from s to any θ_1-oriented or θ_2-oriented chord. Then we will be able to construct a smallest (θ_1, θ_2)-path from s to t using the smallest path from s to a chord containing t. This is the same technique as used in McDonald's thesis [7].

Lemma 5 *For any horizontal maximal chord C in P, every shortest (π, α)-path from s to C ends at the same point on C.*

Proof. Suppose that there are two paths Q, R that are both shortest (π, α)-paths from s to C. Let $q, r \in C$ be the points at which Q and R intersect C, and let $m \in C$ be a point between q and r. Let P' be the polygon (not necessarily simple) defined by the boundaries $Q \cup R \cup \overline{qr}$ and let m' be the projection point $Pr_\alpha(m, P')$ or $Pr_{\alpha+\pi}(m, P')$, whichever is non-degenerate. We know that m' cannot be on the line $\overline{qr}$ because the line $\overline{qr}$ is not α-oriented, and therefore $m' \in Q \cup R$. Without loss of generality, suppose that $m' \in Q$ (see Figure 5). By Lemma 1, the length of the shortest (π, α)-path from m' to q is equal to $\ell(\overline{m'm}) + \ell(\overline{mq})$. Let $Q_s = Q(s \to m')$, $Q_q = Q(m' \to q)$ and let Q_{new} denote the path $Q_s \cup \overline{m'm}$, which is a (π, α)-path from s to C contained in P. Then the length of Q_{new} is:

$$\begin{aligned}\ell_{\pi,\alpha}(Q_{new}) &= \ell_{\pi,\alpha}(Q_s) + \ell_{\pi,\alpha}(m', m)\\ &< \ell_{\pi,\alpha}(Q_s) + \ell_{\pi,\alpha}(m', m) + \ell_{\pi,\alpha}(m, q)\\ &\leq \ell_{\pi,\alpha}(Q_s) + \ell_{\pi,\alpha}(Q_q)\\ &= \ell_{\pi,\alpha}(Q)\\ &= \ell_{\pi,\alpha}(R)\end{aligned}$$

and therefore Q_{new} is shorter than both Q and R, so neither were shortest paths from s to C. $\square$

Lemma 6 *A shortest $\mathcal{O}$-path between s and t that intersects an $\mathcal{O}$-oriented chord C does so in one continuous piece.*

Proof. Let S be a shortest $\mathcal{O}$-path between s and t, let $\overline{pq}$ be a part of S contained entirely on C, and suppose that S intersects C at some point $x \notin \overline{pq}$. S consists of either the subpaths $S(s \to x) \cup S(x \to p) \cup S(p \to t)$ or $S(s \to q) \cup S(q \to x) \cup S(x \to t)$). By Lemma 2 the subpath $S(x \to p)$ (resp. $S(q \to x)$) must be a shortest path from x to p (resp. q to x), and since $\theta(x, p) = \theta(C) \in \mathcal{O}$ (resp. $\theta(q, x) = \theta(C) \in \mathcal{O}$) the shortest path is the straight line $\overline{xp}$ (resp. $\overline{qx}$). This contradicts the assumption that $x \notin \overline{pq}$. $\square$

Lemma 7 *For points $s, t \in P$ and $\alpha \in (0, \pi]$, let C be the horizontal maximal chord containing t and let S be a shortest (π, α)-path from s to C ending at some point $x \in C$ plus the segment $\overline{xt}$. Every U-turn in S contains an extreme vertex.*

Proof. Let $e_1 = \overline{wy}$, $e_2 = \overline{yq}$, and $e_3 = \overline{qz}$ be a U-turn in S with e_1 closest to s on S. Suppose that e_2 is $(\alpha + \pi)$-directed as shown in Figure 6. The proofs for the three other possible directions of e_2 are similar. First we show that there must be a vertex in e_2. By way of contradiction, suppose that there exists some $x' \in e_3$ with $x' \neq q$ such that the line from x' to $Pr_\alpha(x')$ includes a point $x'' \in e_1$. Since $\overline{x''x'}$ is an α-oriented line connecting e_1 and e_3, it has the same length as e_2 and $x'' \neq y$.

1. If $e_3 \not\subseteq C$, the length of the path $S(s \to x'') \cup \overline{x''x'} \cup S(x' \to x)$ is less than the length of $S(s \to x)$, so $S(s \to x)$ was not a shortest path from s to C.

2. If $e_3 \subseteq C$, the length of the path $S(s \to x'') \cup \overline{x''x'}$ is less than the length of $S(s \to x)$, so $S(s \to x)$ was not a shortest path from s to C.

Thus x' cannot exist and there must exist some vertex $v \in V$ on e_2. Now we show that any vertex $v \in e_2$ is extreme. Consider the α-oriented chords $C_1 = C(v, \alpha)$ and $C_2 = C(v, \alpha + \pi)$. $\overline{vy} \in C_1$ and $\overline{vq} \in C_2$, so both chords are of non-zero length and divide P into two end regions and a middle region. Since $e_2 \in C_1 \cup C_2$, no other part of S can intersect $C_1 \cup C_2$ by Lemma 6. Since e_1 lies in the end region of P induced by C_1, $S(s \to y)$ lies entirely in that end region. Since e_3 lies in the end region of P induced by C_2, $S(q \to t)$ lies entirely in that end region. Therefore v is an extreme vertex. □

Lemma 8 *For points $s, t \in P$ and $\alpha \in (0, \pi]$, a shortest (π, α)-path from s to the horizontal maximal chord containing t plus the straight line on the chord to t is a shortest (π, α)-path from s to t.*

Proof. Let C be the horizontal maximal chord containing t and let S be a shortest (π, α)-path from s to C ending at some point $x \in C$ plus the segment $\overline{xt}$. Let R be an arbitrary shortest (α, π)-path from s to t. If there are no U-turns in S, then S is a shortest path from s to t by Lemma 1. Otherwise S contains at least one U-turn, so let v' be a vertex contained on a U-turn of S such that $S(v' \to t)$ is a staircase path. By Lemma 7 we know that v' is extreme with respect to α or π, so v' must be contained in R. Since both $S(s \to x)$ and R are shortest paths, they are made up of shortest subpaths by Lemma 2. Thus both $S(s \to v')$ and $R(s \to v')$ must be shortest paths from s to v' and so have equal length. Furthermore, since $S(v' \to t)$ contains no U-turns, it is a shortest (π, α)-path from v' to t and therefore $R(v' \to t)$ has the same length as $S(v' \to t)$. Therefore S is a shortest path from s to t. □

Lemma 9 *For a horizontal maximal chord C in P, $s \in P$, and $\alpha \in (0, \pi]$, there exists a smallest (π, α)-path from s to C.*

Proof. A minimum-link (π, α)-path from s to C always exists [9]. We prove that a smallest path exists by induction.
Basis: The theorem is obviously true when there are two or fewer links in the minimum-link path from s to C.
Inductive Hypothesis: Assume that we can find a smallest path from s to C when there are k or fewer links in a minimum-link path from s to C.
Inductive Step: Let Q be a minimum-link path from s to C with $k+1$ segments. The $k+1^{st}$ segment of Q must be α-oriented since otherwise we could shorten Q by removing it. Therefore the k^{th} segment of Q is π-oriented, and we can draw a maximal chord C' that contains the k^{th} segment of Q. By the inductive hypothesis, there is a smallest path Q' from s to C' with $k - 1$ segments. Let $q \in C'$ be the point at which Q' ends. See Figure 7.

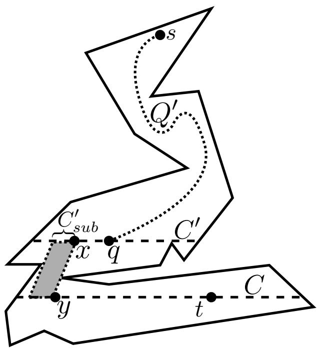


Figure 7: Configuration described in Lemma 9.

Let C'_{sub} be the set of points on C' such that one α-oriented segment contained in P can connect C' and C. Note that there is at least one point in C'_{sub}, namely the point at which the $k+1^{st}$ segment of Q intersects C'. Furthermore, $q \notin C'_{sub}$ since otherwise Q would not be a minimum-link path. Let $x \in C'_{sub}$ be the closest point to q, and let y be the point at which the α-oriented segment from x intersects C.
Now consider the path $S = (Q' \cup \overline{qx} \cup \overline{xy})$. By Lemma 5, $q \in C'$ is the unique ending point on the shortest path from s to C', and by Lemma 8 we know that the shortest path from s to x has length equal to the length of Q' plus the length of the segment $\overline{qx}$. Additionally, the distance from C' to C via a single link is constant. Therefore S is both a shortest path and a minimum-link path. □

Theorem 10 (Smallest Path Theorem). *For any points $s, t \in P \setminus bd(P)$ and $\theta_1, \theta_2 \in (0, \pi]$, there exists a smallest (θ_1, θ_2)-path between s and t.*

Proof. Let Q be a minimum-link path from s to t. Let k be the number of segments in Q and let the orientation of the final segment of Q be θ_1. Now without loss of generality we can rotate the problem so that $\theta_1 = \pi$. Let C be the horizontal maximal chord in P such that the last segment in Q is contained in C. By Lemma 9, there is a smallest path S from s to C. The path S has $(k-1)$ links and ends at some point $x \in C$. By Lemma 8, $S \cup \overline{xt}$ is a shortest path from s to t. Furthermore, $S \cup \overline{xt}$ has k links, which is minimum. □

4 Smallest Paths for More Than Two Orientations

For a set of more than two orientations, a smallest path does not always exist. To show this, we prove two lemmas concerning shortest $\mathcal{O}$-paths between two points. Then examples for which a smallest path is not possible will be shown for $m \geq 3$ allowable orientations.

Definition 13 *In a shortest $\mathcal{O}$-path S between s and t, two points $u, v \in S$ are* consecutive extreme vertices *if $u, v \in V$ are extreme and there is no other extreme vertex v_{mid} such that $v_{mid} \in S(u \to v)$.*

Lemma 11 *If $u, v \in V$ are consecutive extreme vertices on a shortest $\mathcal{O}$-path S in P and $\theta(u, v) \in \mathcal{O}$, then $\overline{uv}$ must be contained in P.*

Proof. We can simplify the proof, without loss of generality, by rotating the problem so that $\theta_{dir}(u, v) = 2\pi$. The subpath $Q = S(u \to v)$ is a shortest path from u to v contained in P by Lemma 2. If Q is a straight line from u to v then $\theta(u, v) \in \mathcal{O}$ and $\overline{uv}$ is contained entirely in P. Otherwise Q contains at least one point with y-coordinate different than $u_y = v_y$. Let z be a point in Q that maximizes $|z_y - u_y|$ and consider the π-oriented maximal chord C through z with endpoints $Pr_{\pi}(z)$ and $Pr_{2\pi}(z)$. By Lemma 6, Q intersects C in one continuous piece, say $\overline{xx'}$ with $\theta_{dir}(x, x') = 2\pi$. Furthermore, both $Q(u \to x)$ and $Q(x' \to v)$ are shortest paths and both lie entirely on the same side of C. It follows that $\overline{xx'}$ must contain an extreme vertex that is preventing $Q(u \to x)$ and $Q(x' \to v)$ from being shorter, so u and v are not consecutive extreme vertices. □

Lemma 12 *For any two consecutive extreme vertices u, v, a shortest $\mathcal{O}$-path contained in P from u to v is either the line $\overline{uv}$, if $\theta(u, v) \in \mathcal{O}$, or a staircase path consisting of two or more segments with orientations in $\mathcal{O}$ that are the neighbouring orientations of $\theta(u, v)$.*

Proof. Let $\theta_i, \theta_j \in \mathcal{O}$ be the neighbouring orientations of $\theta(u, v)$. If $\theta(u, v) \in \mathcal{O}$ then by Lemma 11 the line $\overline{uv}$ is contained in P and thus the shortest path in P between u and v is $\overline{uv}$. Otherwise we are looking for a staircase (θ_i, θ_j)-path in P from u to v. Let R be a shortest (θ_i, θ_j)-path from u to v in P and let S be a shortest $\mathcal{O}$-path in P from u to v. R includes all extreme vertices with respect to θ_i and θ_j by Theorem 3. S also includes all of these extreme vertices because $\theta_i, \theta_j \in \mathcal{O}$. Since u and v are consecutive extreme vertices with respect to $\mathcal{O}$ on S, there is no other extreme vertex between u and v in R. By Lemma 7, every U-turn in R contains an extreme vertex. Thus there are no U-turns in R so R is a staircase (θ_i, θ_j)-path and R is a shortest $\mathcal{O}$-path by Lemma 1. □

Theorem 13 *For three or more allowable orientations, there does not always exist a smallest path from s to t.*

Proof. Figure 8 shows an example where there is no smallest path from s to t for $m = 3$. By Lemma 12, a shortest path from s to v is a staircase (θ_1, θ_3)-path and the shortest path from v to t is a staircase (θ_2, θ_3)-path. Therefore a shortest path from s to t must consist of at least three segments since $\theta(s, v)$ and $\theta(v, t)$ have different neighbouring orientations. However, there is a (θ_1, θ_2)-path from s to t that consists of two segments. Therefore there is no $\mathcal{O}$-path connecting s and t that minimizes length and link distance simultaneously. For $m > 3$, we add orientations $\theta_4, \ldots, \theta_m \in \Upsilon$ between θ_1 and θ_2 to the example of Figure 8. Since the neighbouring orientations of $\theta(s, v)$ and $\theta(v, t)$ don't change, the shortest $\mathcal{O}$-path connecting s and t doesn't change. Similarly, there still exists a 2-link path from s to t. Therefore there is no smallest $\mathcal{O}$-path connecting s and t for $\mathcal{O} = \{\theta_1, \ldots, \theta_m\}$. □

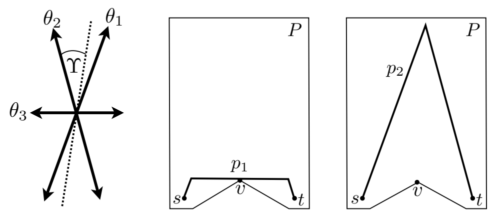


Figure 8: There is no smallest $(\theta_1, \theta_2, \theta_3)$-path between s and t. Any minimum length path has at least three segments.

5 Conclusions

We proved that a smallest path always exists when the path between two points in a polygon is restricted to a set $\mathcal{O}$ of two arbitrary orientations. We also showed that when $\mathcal{O}$ contains three or more orientations, a smallest path may not exist. The results in [14] show that when the paths are restricted to be "smooth" then smallest $\mathcal{O}$-paths always exist in $\mathcal{O}$-oriented polygons.

Acknowledgements

We thank the reviewers for a careful reading of our manuscript and constructive suggestions.

References

[1] K. Clarkson, S. Kapoor, and P. M. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n(logn)^{3/2})$ time. In *Proc. 33rd Ann. Symp. of Foundations on Computer Science*, pages 251–257, 1987.

[2] M. de Berg. On rectilinear link distance. *Computational Geometry - Theory and Applications*, 1(1):13–34, 1991.

[3] J. N. Dicker. Finding shortest s-t paths with a restricted number of orientations. Master's thesis, Simon Fraser University, April 2012.

[4] R. H. Guting. *Conquering Contours: Efficient Algorithms for Computational Geometry.* PhD thesis, Universitat Dortmund, 1983.

[5] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry - Theory and Applications*, 4(2):63–97, 1994.

[6] A. Maheshwari and J.-R. Sack. Simple optimal algorithms for rectilinear link path and polygon separation problems. *Parallel Processing Letters*, 9(1):31–42, 1999.

[7] K. M. McDonald. Smallest paths in polygons. Master's thesis, Simon Fraser University, May 1989.

[8] K. M. McDonald and J. G. Peters. Smallest paths in simple rectilinear polygons. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):864–875, 1992.

[9] J. S. B. Mitchell, V. Polishchuk, and M. Sysikaski. Minimum-link paths revisited. *Computational Geometry: Theory and Applications*, 47(6):651–667, 2014.

[10] B. J. Nilsson, T. Ottman, S. Schuierer, and C. Icking. Restricted orientation computational geometry. In *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, pages 148–185, 1992.

[11] G. J. E. Rawlins. *Explorations in Restricted Orientation Geometry.* PhD thesis, University of Waterloo, 1987.

[12] G. Reich. Finitely-oriented shortest paths in the presence of polygonal obstacles. Technical report, Institut fur Informatik, Universitat Freiburg, 1991.

[13] B. Speckmann and K. Verbeek. Homotopic rectilinear routing with few links and thick edges. In *LATIN 2010: Theoretical Informatics: 9th Latin American Symposium*, pages 468–479, 2010.

[14] K. Verbeek. Homotopic $\mathcal{C}$-oriented routing. In *Proc. 20th Int. Conf. on Graph Drawing*, pages 272–278, 2013.

[15] P. Widmayer, Y. F. Wu, and C. K. Wong. On some distance problems in fixed orientations. *SIAM Journal on Computing*, 16(4):728–746, 1987.

[16] C. D. Yang, D. T. Lee, and C. K. Wong. The smallest pair of noncrossing paths in a rectilinear polygon. *IEEE Trans. on Computers*, 46(8):930–941, 1997.

Exact Solutions for the Geometric Firefighter Problem[*]

Mauricio J. O. Zambon[†] Pedro J. de Rezende[†] Cid C. de Souza[†]

Abstract

We consider the Geometric Firefighter Problem (GFP) recently proposed by Klein et al. in [6], where several problem variants were proven $\mathbb{NP}$-hard. Despite this negative result, the design of algorithms for the GFP remains relevant as they may help reveal geometric properties of exact solutions. This paper proposes a method for finding guaranteed optimal solutions and reports on empirical tests over 900 polygons of up to 300 vertices. Almost all benchmark instances were solved within three minutes on a standard off-the-shelf computer. This was made possible by a savvy combination of geometric preprocessing, integer programming modeling and heuristics proposed here.

1 Introduction

Well known by the graph theoretic community (see [4], [2], [3] and the references therein), the *Firefighter Problem* may be described as a discrete time-lapse simulation on a graph $G = (V, E)$ of a firefighter trying to prevent areas adjacent to a fire from burning. At time t_0, a single vertex $r \in V$ starts to burn. At the same time, a firefighter can select a vertex $v \in V$ to be shielded. After a vertex starts to burn or is protected, it can no longer change states. At each subsequent time t_i, $i \geq 1$, the fire spreads to the (immediate) neighborhood of every burning vertex, while the firefighter chooses another vertex to be saved. For finite graphs, the process ends when all vertices are either burning or saved. Among the many interesting optimization objectives for this problem, one can seek to minimize the number of burned vertices or the number of clock ticks required to stop the fire from spreading further.

A natural variation of this problem on the euclidean plane was proposed by Klein et al. ([6]), called the *Geometric Firefighter Problem* (GFP). In the GFP, the object of study is a simple polygon P, while a fire originates at a predetermined point r in P. We are then interested in saving regions within P from burning, which is accomplished by choosing barriers to be built from a pre-specified static set. To complete the description of the problem, we are also given the (constant) speed at which the fire spreads from point r, as well as the (constant) construction speed of the barriers. How these speeds are related affects the construction of the barriers in the sense that, if the fire reaches any yet unconstructed point of a barrier, that barrier succumbs and ceases to exist. More formally, we are interested in the following variation:

Geometric Firefighter Problem: Let P be a simple polygon, $r \in P$ the fire starting point, B a set of curves within P, and v_f and v_b the fire and barriers construction speeds, respectively. Find the barriers in B that should be fully constructed in order to maximize the area of P protected from the fire. Here, we assume v_f and v_b to be constant and that the barriers have to be built one at a time.

Previous works. The only reference to the GFP that we are aware of is [6] where the authors formulate the problem and prove its $\mathbb{NP}$-hardness for the cases where P is a simple polygon and B is a set of diagonals of equal length; or P is a convex polygon and B is the set of all of its diagonals; or P is a star-shaped polygon and there are no restrictions on the set of barriers. The article also presents an $\sim$11.65-approximation algorithm for the GFP, as well as the description of a simpler problem, although also $\mathbb{NP}$-hard called *Budget Fence Problem*. In this variant, we seek to enclose a given static region inside P subject to a bounded budget on the total length of the fence built. An additional requirement is that the constructed barriers have pairwise disjoint interiors. Also, in [5], Klein et al. examine the akin case of a circularly spreading fire on the plane at unit speed and the question of fire containment in light of various barrier construction speeds, paths, strategies, etc.

Our contribution. Notwithstanding the hardness of the GFP, the following question begs to be addressed: how efficiently can we solve instances of the GFP of increasing sizes in practice? Here, we present a practical study on solving the GFP when the set of barriers is composed only of straight line segments inside P. We report experiments with polygons of up to 300 vertices.

This paper is organized as follows. Additional definitions are introduced in Section 2. Section 3, describes the preprocessing phases and an Integer Programming (IP) model for the GFP, followed by implementation details and experimental results in Section 4. Final remarks and conclusions are the object of Section 5.

[*]This research was supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) #311140/2014-9, #477692/2012-5; #304727/2014-8; #141583/2016-9; Coordenação de Aperfeiçoamento de Pessoal de Ensino Superior (Capes) #1458729; and FAEPEX/UNICAMP.

[†]{mauricio.zambon|rezende|cid}@ic.unicamp.br

2 Preliminaries

In this section, we present additional definitions used in the later description of the preprocessing phase required for generating the IP model presented in Section 3.

We consider throughout this paper that the set of barriers B is compliant with the linearity condition described in [6], which states that barriers must not cooperate to protect a region, i.e., for any subset $B' \subset B$, the area protected by the union of the barriers in B' must be equal to the union of the areas protected individually by each of those barriers.

Deadline. An essential concept for the undermentioned model is the notion of deadline. Let $f_b(t)$ be a function defined for a barrier $b \in B$ where $f_b(t) = \mathtt{T}$ (true) iff b can be successfully built provided its construction starts at time t, and $\mathtt{F}$ (false) otherwise. A barrier b is *successfully built* iff the fire does not reach any yet unconstructed point of b. We define the *deadline* of a barrier b as $\delta_b = \max\{t \mid f_b(t) = \mathtt{T}\} + p_b$, where p_b is the time required to build b. Notice that since $f_b(t)$ does not take into account the direction for building b, the definition of δ_b actually considers both directions.

3 Algorithm

We are now ready to describe the complete algorithm for solving the GFP, which is composed by two main parts: a preprocessing phase and a solver phase. Preprocessing starts by computing the deadlines for each barrier in B with which we can obtain a primal feasible solution. The purpose of this solution is twofold. First, it is used by the barrier elimination algorithm, and later as a warm start for the IP solver. For analysis purposes, we will split preprocessing into two stages: S, the deadline computation and the primal algorithm, and BE, the barrier elimination algorithm. Once preprocessing is completed, the IP model is prepared and loaded onto the solver.

Computing deadlines. To understand how to calculate the deadlines for line segment barriers, consider Figure 1 where barrier b has endpoints b_s and b_e, and supporting line γ_b. For simplicity, consider that r, the fire source, sees b, and that c, the closest point to r on γ_b belongs to b (the following reasoning can easily be extended to the case where $c \in \gamma_b - b$).

Although we need to take both directions of construction of b into account when computing δ_b, we will consider the direction $\overrightarrow{b_s b_e}$ here, as the steps are essentially the same otherwise. When the fire first reaches b, at point c, it starts spreading in both directions around c. At this point, the building process must have reached c, i.e., $\overline{b_s c}$ has already been constructed. Let us examine the segment $\overline{cb_e}$. Let b_t be the point of $\overline{cb_e}$ reached by the fire at time t. From the triangle $\triangle rcb_t$ we get $(v_f \cdot t)^2 = d(r,c)^2 + d(c,b_t)^2$, where $d(a,b)$ is the euclidean distance between a and b. Hence, $d(c,b_t) = (v_f^2 \cdot t^2 - d(r,c)^2)^{1/2}$. In summary, $d(c,b_t) = s_{cb_e}(t)$, where $s_\sigma(t)$ denotes the length of the segment σ burned by the fire at time t.

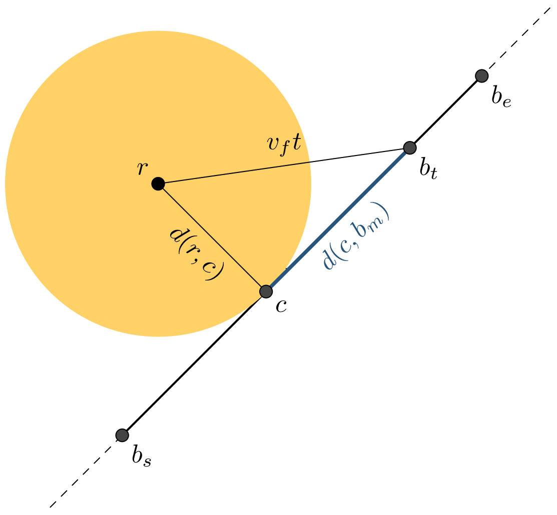


Figure 1: Computing deadlines.

A similar expression for the length $\ell_{cb_e}(t) = v_b \cdot t$ of the constructed portion of the barrier at time t (starting at c) can be deduced. We now need to guarantee that $\ell_{cb_e}(t) \geq s_{cb_e}(t)$, for all $\tau_c \leq t \leq \tau_{b_e}$, where τ_p is the time for the fire, from r, to reach p. We can rewrite $\delta_b = \max\{t \mid \ell_{cb_e} \geq s_{cb_e}(t) \text{ and } \tau_c \leq t \leq \tau_{b_e}\} - p_{b_s c} + p_b$, i.e., the maximum t for which $f_{cb_e}(t) = \mathtt{T}$ minus the time to build $\overline{b_s c}$ plus the time to build the whole barrier. This computation narrows down to a few closed formulas, which we must omit due to space restrictions.

Note that a reflex vertex that occludes a (portion of a) barrier b from r becomes a fire source to be considered in all above calculations. Moreover, since b ends up being partitioned into smaller segments based on which of such fire sources sees it, the value of δ_b is set to be the smallest of the deadlines of these segments. Accordingly, the shortest path distances from r to the fire sources have to be taken into account as well.

Primal solution. To obtain a feasible primal solution, we employ two algorithms. The first one is a greedy heuristic, and the second is the ~11.65-approximation algorithm from [6]. For the greedy algorithm, consider the arrangement $A = P \cup B$. Since the faces in A are atomic with respect to the set B, we can associate each barrier to the set of faces it protects. The algorithm works as follows. At each iteration, it tries to fit into the current construction schedule the barrier that shields the largest unprotected area. Once a barrier is processed, it is never considered again, regardless of whether it was added to the solution or not.

Barrier elimination algorithm. As it will become clear later, the cardinality of the set of barriers directly impacts the size of the IP model. For this reason, methods that might allow us to discard from consideration barriers that cannot appear in an optimal solution are worth investigating. Barriers whose time constraints impede them from being constructed are clearly disposable.

Furthermore, we designed an algorithm that discards barriers based on the value of a primal solution (such as those obtained with the aforementioned algorithms). Recall that our objective is to maximize the area within P protected from the fire. Given a feasible solution, we know how much of the polygon will be burned under that solution. Denote by A_b the area of said burned region. Assuming that we do not build any unnecessary barriers, i.e., those that do not benefit to the current solution, we can also compute a lower bound for the area that will be burned if we decide to build a given barrier b. To do this, we create a directed acyclic graph G, whose vertices are the barriers in B, and with an edge between u and v, if barrier u *dominates* barrier v, i.e., v is fully contained in the region guarded by u. Once G is set up, for each barrier $b \in B$, we build an arrangement M_b composed by only the barriers that do not dominate b, i.e., the barriers that could stop the fire from reaching b. Then, provided we decide to erect b, a lower bound to the burned area is that of the face f_r in M_b that contains the fire source r. If the area of f_r is greater than A_b we may safely discard b since we already know a solution that is better than any that requires building b. Notice that there is also no need to include in M_b any barriers dominated by b and we can further improve the arrangement construction by not including any dominated barrier as well, since they will not help design f_r.

The overall time complexity of the preprocessing phase depends first on determining the visibility from each reflex vertex to each of the $|B|$ barriers, which can be done in $O(|B||P|^2 \log |P|)$ time[1]. Secondly, the computation of deadlines[2] can be performed in $O(|B||P|)$ time. Thirdly, the complexity of the algorithm responsible for determining which barriers protect each of the $O(|B|^2)$ faces of the arrangement is $O(|B| \log |P|)$ per barrier for a total of $O(|B|^3 \log |P|)$ time. Lastly, the barrier elimination phase has time complexity $O(|B|^3)$.

Integer Programming Model. Inequalities (1)–(10) describe an IP model for solving the GFP. The following variables are used: one *binary* variable x_f for each $f \in A$ (the arrangement composed by $P \cup B$), such that $x_f = 1$ iff face f is shielded from the fire. For each $i \in B$, the *binary* variable $y_i = 1$ iff barrier i is used. To enforce ordering on the construction of the barriers, for each pair of barriers (i, j), $i \neq j$, the *binary* variable $\nu_{ij} = 1$ means that barrier i will be built before j, and $= 0$ if either the construction of j precedes the construction of i or if i or j is not built at all. Finally, the continuous variable c_i, one for each barrier, indicates the time that barrier i finishes being built, whenever it is constructed.

The objective function (1) aims to maximize the shielded area of the polygon by maximizing the sum of the areas of the saved faces. The set of inequalities (2) forces that any protected face must have at least one barrier shielding it from the fire. Restrictions (3) and (4) ensure the total ordering of the construction of all barriers in a solution by requiring that for all pairs (i, j) of barriers constructed, either ν_{ij} or ν_{ji} is set. Inequalities (5) and (6) forbid any unconstructed barrier to be part of the total ordering. Constraints (7) guarantee that if barrier i is constructed before barrier j, then j cannot be completed before the instant i is finished plus j's own building time (here, M is a large enough value, *e.g.*, the sum of all barrier construction times). Constraints (8) and (9) force the finishing time of i to be null if it is not built or, else, to be between its construction time p_i and its deadline δ_i. Lastly, restrictions (10) guarantee the integrality of the variables x_f, y_i and ν_{ij}.

$$z = \max \sum_{f \in A} a_f x_f \tag{1}$$

$$\sum_{\substack{b \in B| \\ f \in S(b)}} y_i \geq x_f \qquad \forall f \in A \tag{2}$$

$$\nu_{ij} + \nu_{ji} \geq y_j - (1 - y_i) \qquad \forall i, j \in B, i < j \tag{3}$$

$$\nu_{ij} + \nu_{ji} \leq 1 \qquad \forall i, j \in B, i < j \tag{4}$$

$$\nu_{ij} \leq y_i \qquad \forall i, j \in B, i \neq j \tag{5}$$

$$\nu_{ji} \leq y_i \qquad \forall i, j \in B, i \neq j \tag{6}$$

$$c_j \geq c_i + p_j - M(1 - \nu_{ij}) \qquad \forall i, j \in B, i \neq j \tag{7}$$

$$c_i \geq p_i y_i \qquad \forall i \in B \tag{8}$$

$$c_i \leq \delta_i y_i \qquad \forall i \in B \tag{9}$$

$$x_f, y_i, \nu_{ij} \in \{0, 1\} \qquad \forall f \in A, \forall i, j \in B \tag{10}$$

4 Experimental Analysis

In this section, we report on test data and results regarding exact solutions for the GFP.

Environment. The machine used features an Intel® Xeon® CPU E5-2420, with 12 cores at 1.90GHz each, and 32GB of RAM. The code was written in C++ and compiled with GCC 5.3.0. The libraries used were CGAL 4.7 for geometric data structures and algorithms combined with the GMP 5.1.3 library for fixed and arbitraty precision numbers, Boost library 1.60.0 for data structures, and the Integer Programming solver IBM ILOG CPLEX 12.6.1.

Implementation details. The calculation of the deadlines described in Section 2 involves the computa-

[1]Throughout this analysis, we consider arithmetic operations on arbitrary precision rational numbers realizable in $O(1)$ time.

[2]Here, square roots of fixed precision numbers are regarded as having unit cost. See Implementation Details in Section 4.

tion of square roots. Even though we are seeking exact solutions for the GFP, numerical exact computation of square roots is certainly a very time consuming task in practice. To attenuate this problem, i.e., for the sake of efficiency, we renounce arbitrary precision in favor of a more realistic approach, yet equally precise for practical purposes. All root computations are done using a finite precision number type whose mantissa precision is set to 128 bits (more than twice that of the `double` number type). That number type is then converted to an arbitrary precision rational number type that is used for the rest of the code. Besides, notice that due to CPLEX's architecture, any attempt to employ exact square roots would, in the end, be ill-founded.

Also, when analyzing the data reported in this section, one should be aware of aspects regarding the use of multi-threaded processes. While the solver is able and was allowed to use all available threads (up to 12 in our environment), the preprocessing phases were run on a single thread since they could not be parallelized due to the arbitrary precision rational numbers library not being thread-safe.

Instances. The polygons used for the tests were obtained from the publicly available benchmark `AGP2009a` in [1]. In total, we assembled a set of 900 instances as follows. For each of the sizes 100, 200 and 300 vertices, we selected 30 simple and 30 orthogonal polygons. From each of these 180 polygons, five instances were generated with barrier construction speed $v_b = 2$, and five different fire speeds $v_f \in \{1, 2, 4, 8, 16\}$. The fire source was randomly generated inside the polygon and was the same for each of these sets of five instances. Hence, a total of 450 instances based on simple polygons, and 450 based on orthogonal polygons were created. To increase the complexity of the instances, we opted for the set of barriers to include *all* internal diagonals of the polygons as also considered in [6].

Testing conditions. Initially, during the Start-up phase (S), we discard all barriers that cannot be constructed due to time constraints. Next, we divide our experiments into two strategies that differ only with respect to additional barrier elimination. We either keep all remaining barriers and simply run the IP solver on the resulting model (Strategy 1 or SIP), or apply the algorithm for eliminating barriers based on a primal viable solution (see Section 3) before applying the solver to the reduced model (Strategy 2 or SBEIP). In SBEIP, we actually employ the best primal solution obtained from the greedy or the approximation algorithm described before. In either case, the best primal solution is also used as a warm start for the solver. Furthermore, we disabled all CPLEX's probing algorithms since tests show that they consume a large amount of time, with no observable efficiency improvement. Lastly, a time limit of 30 minutes was set for the solver for each instance.

Results. All of the orthogonal polygon based instances were solved by both strategies to exactness. In the case of simple polygons, the SBEIP strategy solved all but two instances to optimality. Another nine instances produced IP models too large to be loaded into memory. The SIP strategy could not solve to optimality the same two instances, while a full 61 (out of 450) other ones were too large for the available memory. This suggests that the simple polygon based instances are harder in practice, and that SBEIP is a more powerful approach.

Let us first analyze the computing times for SIP and SBEIP subsequent to the start-up time T_S (common to both strategies). These are denoted $T_1(\text{IP})$ and $T_{\text{BE}} + T_2(\text{IP})$, respectively. The former is only the time for the solver to optimize the larger IP model and the latter is the sum of the times for barrier elimination and for the solver to find a solution for the reduced model. Table 1 shows $T_1(\text{IP})$ and $T_{\text{BE}}+T_2(\text{IP})$ for simple polygons. For orthogonal polygons, we present only the times $T_{\text{BE}}+T_2(\text{IP})$ in Table 2, and omit the times $T_1(\text{IP})$ due to their similarity to the simple polygon counterparts. Firstly, we draw the reader's attention to the standard deviations, which often significantly surpass the mean values (see Figure 2 for a typical distribution of the measured times). This suggests that polygon size and fire propagation speed are not determinant factors for evaluating the hardness of an instance. Nonetheless, note that the mean of $T_1(\text{IP})$ in any cell exceeds the corresponding entry for $T_{\text{BE}} + T_2(\text{IP})$. More precisely, $T_{\text{BE}}+T_2(\text{IP})$ is less than $T_1(\text{IP})$ for 73.7% of the instances on simple polygons, and for 95.3% of the instances on orthogonal polygons. Moreover, for the cases where $T_{\text{BE}}+T_2(\text{IP}) \leq T_1(\text{IP})$, the first was $82.2 \pm 24.7\%$ smaller than the second for simple polygons and $89.1 \pm 16.9\%$ smaller for orthogonal polygons. On the other hand, when $T_{\text{BE}} + T_2(\text{IP}) > T_1(\text{IP})$, the difference was only $18.6 \pm 15.0\%$ for simple polygons and $11.5 \pm 9.6\%$ for orthogonal ones.

Now, to compare the effectiveness of the two algorithms used to compute primal feasible solutions, Table 3 shows the average percentage gap between these and the optimal solutions for all 439 simple polygon based instances with known optima. The column labeled *Max* displays the average of the best primal solutions from the two algorithms revealing that they are quite good in practice. Notice also that the greedy algorithm does better when $v_f \in \{1, 2, 16\}$ while the approximation algorithm surpasses it for $v_f \in \{4, 8\}$. Similar results were obtained for the orthogonal cases but exhibiting them here seemed superfluous.

Since the number of diagonals on the simple polygons of, say, 300 vertices was on average over 1000, it is interesting to determine how effective the two barrier elimination procedures are. Table 4 shows the percentages of barriers discarded (w.r.t. the original numbers) by

v_f	$\lvert P\rvert = 100$	$\lvert P\rvert = 200$	$\lvert P\rvert = 300$
T_1(IP) in seconds			
1	19.7 ± 17.9	125.5 ± 44.2	207.4 ± 85.3
2	18.6 ± 11.8	141.2 ± 64.4	248.7 ± 104.1
4	9.2 ± 3.9	128.9 ± 133.3	278.3 ± 109.5
8	3.1 ± 1.8	61.9 ± 56.8	291.8 ± 286.9
16	0.5 ± 0.4	13.0 ± 8.3	85.1 ± 68.2
$T_{BE} + T_2$(IP) in seconds			
1	8.4 ± 20.4	16.9 ± 24.5	43.4 ± 93.2
2	9.5 ± 15.4	51.8 ± 92.1	45.7 ± 81.7
4	4.5 ± 6.4	95.0 ± 183.4	87.9 ± 158.6
8	1.3 ± 1.8	23.0 ± 56.6	168.8 ± 337.0
16	0.2 ± 0.1	5.4 ± 9.6	47.7 ± 76.7

Table 1: T_1(IP) and $T_{BE} + T_2$(IP) for simple polygons.

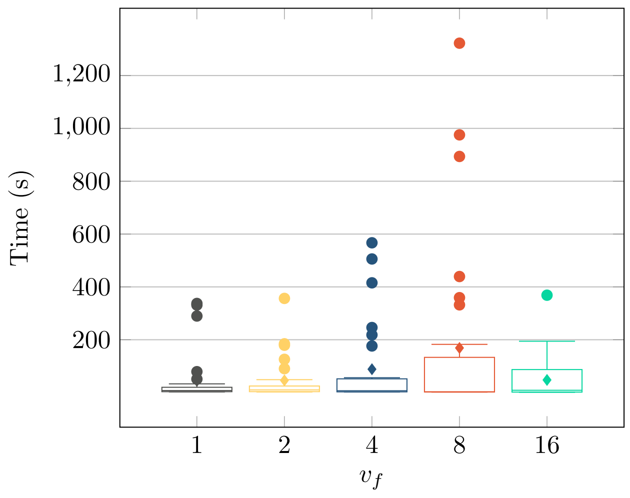


Figure 2: Boxplot showing time measurement distribution of T_{BE}+T_2(IP) for simple polygons with 300 vertices (♦ = average).

v_f	$\lvert P\rvert = 100$	$\lvert P\rvert = 200$	$\lvert P\rvert = 300$
1	1.1 ± 1.7	2.9 ± 3.2	4.3 ± 9.8
2	2.7 ± 5.3	11.1 ± 30.2	6.4 ± 16.2
4	1.3 ± 2.0	30.7 ± 103.8	12.2 ± 37.2
8	0.9 ± 1.5	10.6 ± 24.5	13.5 ± 39.7
16	0.3 ± 0.4	3.1 ± 5.2	27.4 ± 99.5

Table 2: T_{BE}+T_2(IP) (seconds) for orthogonal polygons.

infeasibility (during the Start-up phase) and the ones discarded by the BE algorithm. Low fire propagation speed (compared to barrier construction speed) leads elimination by infeasibility to have better performance, while high fire speed induces BE to have a clear advantage due to the high quality of the primal solutions.

v_f	Greedy	Approx.	*Max*
1	$0.6 \pm 1.5\%$	$2.1 \pm 2.8\%$	$0.4 \pm 0.7\%$
2	$1.8 \pm 3.7\%$	$2.3 \pm 3.3\%$	$1.1 \pm 2.1\%$
4	$3.5 \pm 5.6\%$	$3.1 \pm 4.7\%$	$2.2 \pm 3.7\%$
8	$8.2 \pm 12.5\%$	$5.2 \pm 7.8\%$	$3.4 \pm 6.2\%$
16	$7.5 \pm 11.2\%$	$7.8 \pm 12.7\%$	$4.1 \pm 6.6\%$

Table 3: Average percentage gap from the optimal solutions (for simple polygons).

	Simple		Orthogonal	
v_f	Infeasible	BE	Infeasible	BE
1	$1.7 \pm 1.6\%$	$75.5 \pm 25.3\%$	$2.8 \pm 2.9\%$	$85.3 \pm 14.9\%$
2	$4.0 \pm 3.4\%$	$63.6 \pm 31.2\%$	$5.8 \pm 5.6\%$	$77.1 \pm 24.1\%$
4	$12.6 \pm 8.2\%$	$52.9 \pm 33.9\%$	$18.1 \pm 10.6\%$	$67.9 \pm 26.6\%$
8	$27.4 \pm 13.6\%$	$41.8 \pm 28.3\%$	$40.0 \pm 15.1\%$	$57.8 \pm 27.3\%$
16	$50.2 \pm 17.0\%$	$21.1 \pm 18.3\%$	$67.2 \pm 14.6\%$	$37.8 \pm 22.7\%$

Table 4: Percentage of barriers removed for infeasibility and by the BE algorithm.

Lastly, let us analyze the relative contributions of the Start-up (S), barrier elimination (BE) and solver (IP) phases. Figure 3 shows the average times for T_S, T_{BE} and T_2(IP), for simple polygon based instances of sizes 100, 200, 300, grouped by $v_f \in \{1, 2, 4, 8, 16\}$. T_S grows with the size of the instance as expected. The primal solution based barrier elimination's computing time (T_{BE}) is comparatively small but tests showed that its standard deviation is very high, ranging from 39.6 (for $|P| = 100$ and $v_f = 16$) to as high as 142.0 seconds. This behavior is explained by the following observations. First, BE is highly dependent on the cardinality of the set of barriers and the arrangement inside the polygon. So, the randomness of the given simple polygons leads to a very diverse set of barriers, however, the BE algorithm allows for shortcuts that easily bypass barriers dominated by others already discarded.

A broader analysis, though, involves comparing the average solver time T_{IP} to the average total preprocessing time $T_S + T_{BE}$ for the SBEIP strategy. Often T_{IP} is lower than the time for the whole preprocessing phase. Still, we see that this relative performance can fluctuate with the fire propagation speed. We also presume that a more consistent behavior would be observed if we could scale the polygon sizes much higher, since all preprocessing steps are polynomially bounded while an IP solver is inherently not (unless $\mathbb{P}$=$\mathbb{NP}$).

The data in Table 5 corroborates and quantifies some of these observations by showing that the average of the ratios $T_{IP}/(T_S + T_{BE})$ is indeed small for the polygon sizes tested. This stems from the fact that the preprocessing phase does a great deal of the hard work by filtering out most inessential barriers (by infeasibility and through algorithm BE), leaving a much leaner IP model.

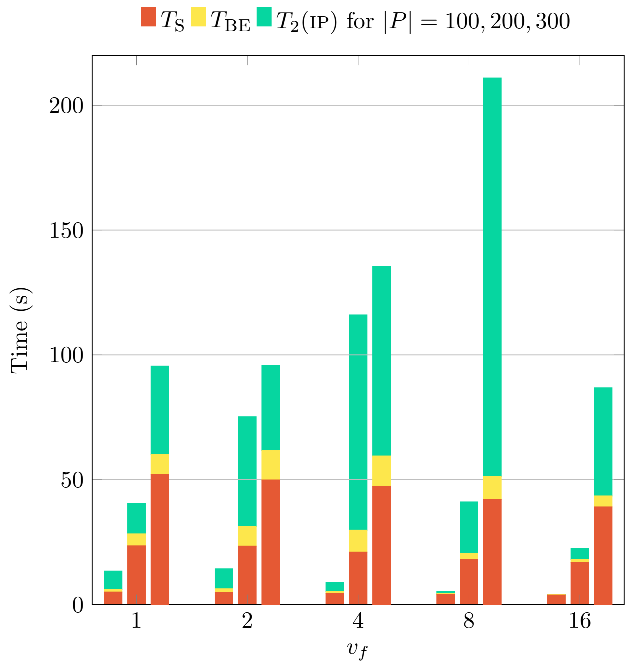


Figure 3: Averages of T_S, T_{BE}, $T_2(IP)$ for simple polygons.

v_f	$\|P\| = 100$	$\|P\| = 200$	$\|P\| = 300$
Simple			
1	0.91 ± 2.11	0.35 ± 0.48	0.38 ± 0.81
2	1.05 ± 1.59	0.94 ± 1.46	0.39 ± 0.62
4	0.55 ± 0.77	2.22 ± 4.70	0.85 ± 1.50
8	0.22 ± 0.41	0.76 ± 1.95	2.46 ± 5.34
16	0.03 ± 0.03	0.21 ± 0.39	0.90 ± 1.51
Orthogonal			
1	0.28 ± 0.47	0.18 ± 0.23	0.12 ± 0.38
2	0.74 ± 1.41	0.65 ± 1.91	0.20 ± 0.64
4	0.62 ± 1.24	2.16 ± 7.10	0.43 ± 1.36
8	0.59 ± 1.11	1.18 ± 2.84	0.44 ± 1.19
16	0.19 ± 0.29	0.35 ± 0.64	1.27 ± 4.57

Table 5: Averages of the ratio $T_2(IP)/(T_S + T_{BE})$.

We emphasize that although the results presented in this section are for instances whose sets of barriers are composed only by polygon diagonals, both the preprocessing and the IP model may be utilized for an arbitrary set of barriers formed by straight line segment that respect the linearity condition described in Section 2.

The attentive reader must suspect that the fire source location alone may heavily impact the complexity of an instance. To substantiate this perception, consider Figure 4. It depicts two instances generated from the same polygon of 40 vertices, using the same values for v_f and v_b, but with different fire source location r. We solved this instance after disabling the barrier elimination algorithm (BE) and we did not use a warm start for the IP solver. Notice that the solver time $T(IP) = 342.5$ s for the instance in Figure 4a is more than five times $T(IP) = 62.8$ s for the instance in Figure 4b.

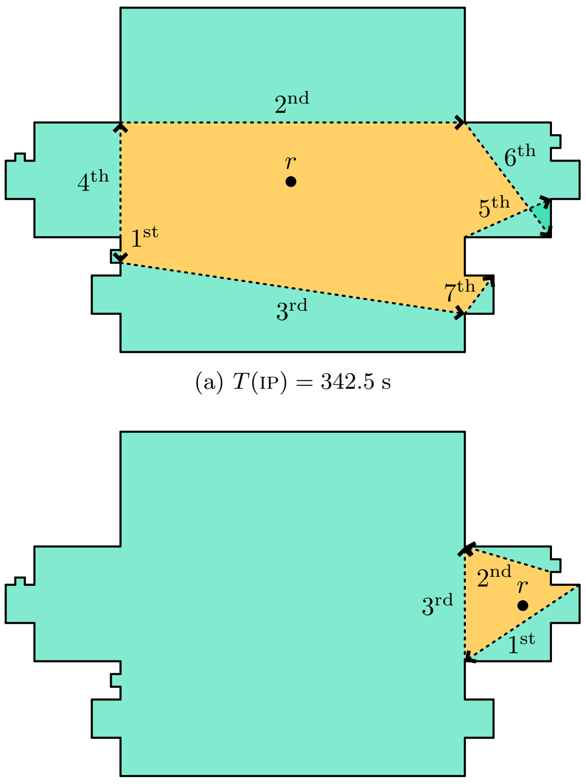


Figure 4: Optimal solutions, their barrier construction order and directions for two instances: same polygon and same set of barriers (all diagonals), same v_f and v_b; *distinct fire sources trigger contrasting solver times.*

5 Conclusion

In this paper, we presented methods to solve the GFP to exactness in practice for instances of up to 300 vertices. In the formulation introduced here, the set of barriers is composed only of line segments. We reported experimental results over a set of 900 instances and show that discarding barriers based on the value of a known primal solution can be very effective and greatly improves the overall computing time. Data also show that it is not simple to decide *a priori* on the hardness of an instance based solely on primary characteristics of the input such as the size of the polygon and the speed of fire propagation and of barrier construction. The initial location of the fire source seems to play a significant role, as well.

Many directions may be considered for improving the methods presented here in order to solve even larger instances. In particular, identifying geometric properties of a given instance that could lead to strengthening the constraints of the model is paramount. Moreover, polygons with holes, multiple sources of fire, time constraints for firefighters to move about, concurrent barrier constructions and piecewise linear barriers (or even curves) are some of the aspects worth investigating.

References

[1] M. C. Couto, P. J. de Rezende, and C. C. de Souza. Instances for the Art Gallery Problem, 2009. www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery.

[2] S. Finbow and G. MacGillivray. The firefighter problem: A survey of results, directions and questions. In *Australasian Journal of Combinatorics*, volume 43, pages 57–77. 2009.

[3] C. García-Martínez, C. Blum, F. Rodriguez, and M. Lozano. The firefighter problem: Empirical results on random graphs. *Computers & Operations Research*, 60:55–66, 2015.

[4] B. Hartnell. Firefighter! an application of domination. presentation. In *25th Manitoba Conference on Combinatorial Mathematics and Computing, Canada*, 1995.

[5] R. Klein, E. Langetepe, and C. Levcopoulos. A Fire Fighters Problem. In L. Arge and J. Pach, editors, *31st International Symposium on Computational Geometry*, volume 34 of *LIPIcs*, pages 768–780, Germany, 2015.

[6] R. Klein, C. Levcopoulos, and A. Lingas. Approximation algorithms for the geometric firefighter and budget fence problems. In A. Pardo and A. Viola, editors, *LATIN*, volume 8392 of *LNCS*, pages 261–272. Springer, 2014.

Progressive Alignment of Shapes

Ashwin Gopinath[*] David Kirkpatrick[†] Paul W. K. Rothemund[‡] Chris Thachuk[§]

Abstract

We introduce a natural property of shape pairs, that, starting from any initial overlapping configuration, they can be brought into a unique configuration of maximum overlap by a continuous motion that monotonically increases their overlap. The identification of such shapes is motivated by applications of self-assembly, driven by molecular forces, in nanofabrication processes.

1 Introduction

Certain shapes, a disk is the simplest example, have the property that, starting from any initial placement with non-zero overlap with a target placement, there is a continuous rigid motion (a simple translation between centres, in the case of a disk) that takes the shape to its target placement with monotonically increasing overlap. We are interested in designing–and certifying–such self-aligning shapes. More generally, we are interested in shape-target pairs, under the additional restriction that the motion terminates in a unique placement that maximizes the overlap with the target. (Because of their rotational symmetry, disks clearly do *not* satisfy this restriction.)

We formalize the notion of shape alignment with some preliminary definitions. A (planar) *shape* is a connected bounded subset of $\Re^2$. A *placement* λ of a shape A is a proper rigid transformation of A (expressed as a translation and rotation of A, with respect to its centre of mass). We will frequently refer to both λ and the resulting shape, denoted A_λ, as a placement of A. Given a target shape T, the *T-overlap* of a placement A_λ is just area of intersection of T with A_λ. Placements with non-zero T-overlap are said to be *T-proximate*. A placement A_λ is locally (resp. globally) *optimal* with respect to target shape T if the T-overlap of A_λ is locally (resp. globally) maximum in the space of all placements of A. A *T-alignment* of shape A (or simply *alignment*, when T is understood), from placement λ_0 to placement λ_1, is a continuous function μ from $[0, 1]$ to T-proximate placements of A, with $\mu(0) = \lambda_0$ and $\mu(1) = \lambda_1$. An alignment μ of A is *T-progressive* if the T-overlap of $A_{\mu(j)}$ exceeds that of $A_{\mu(i)}$, for all $0 \le i < j \le 1$. The pair (T, A) is an *optimally aligning pair* if there is a T-progressive alignment of A to a unique globally-optimal placement, from every T-proximate placement of A. (Note that in this case A has exactly one locally optimal placement.) If (A, A) is an optimally aligning pair then we say that A is a *self-aligning shape.*

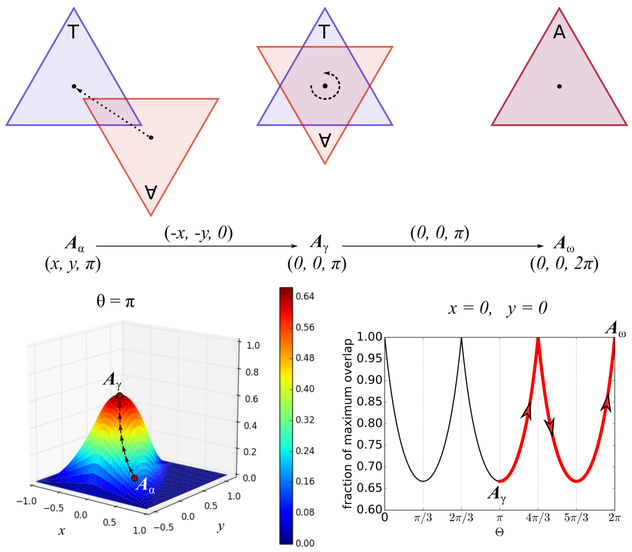


Figure 1: a two phase alignment pathway $\Pi = A_\alpha, A_\gamma, A_\omega$ for an equilateral triangle A and target placement T. In the first phase, shape A is translated (from its initial placement A_α to placement A_γ) along the vector that brings its centre coincident with that of T. In the second phase, A is rotated by π until it coincides with T (placement A_ω). Shown above are the relative placements of A and T. Shown below is the relevant projection of the alignment landscape that contains the pathway Π (the directed path highlighted in red).

We begin with a simple example to illustrate some of the concepts defined above (see Fig. 1). Consider an equilateral triangle A, with sides of length 1, centred at $(0, 0)$. An alignment of A, from some initial placement A_α (with rotation π) to some final placement A_ω (with rotation 2π), is a continuous sequence of placements, from A_α to A_ω.

The shape-pair (T, A) defines an *alignment landscape*

[*]Bioengineering, California Institute of Technology, ashwing@caltech.edu

[†]Department of Computer Science, University of British Columbia, kirk@cs.ubc.ca

[‡]Bioengineering and Computing & Mathematical Sciences, California Institute of Technology, pwkr@caltech.edu

[§]Computing & Mathematical Sciences, California Institute of Technology, thachuk@caltech.edu

in 4-dimensional space. An example pathway in that landscape is illustrated in Fig. 1. Due to the 3-fold rotational symmetry of A, and since A and T are congruent, the alignment landscape has three global maxima where A has been rotated by angle $\theta \in \{0, 2\pi/3, 4\pi/3\}$. After the centre of A has become coincident with that of T (placement A_γ) the alignment path rotates A through a first and then a second global maximum. Note that the example path is *not* T-progressive, but the prefix of the path ending at the placement with rotation $4\pi/3$ is progressive.

Observe that the absence of self-similarity alone is not sufficient to guarantee that a shape is self-aligning (see Fig. 2).

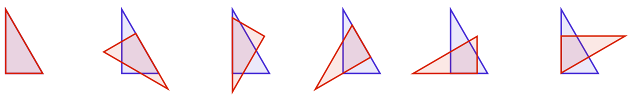


Figure 2: a right triangle is not self-aligning since many placements are local maxima.

This leaves us with the question: what shapes are self-aligning or, more generally, what shape-pairs are optimally aligning? One strategy is to separately model each specific alignment landscape with sufficient precision to effectively rule out the existence of undesirable local maxima. Our goal is to address this question using more precise geometric arguments that (i) bring more clarity to our understanding (in terms of parameter settings for certain families of shapes) of the shape characteristics that support self-alignment, and (ii) help to inspire the design of novel shapes with other desirable characteristics.

In the next section, we describe some of the motivation for the study of optimally-aligning shape-pairs that derives from applications in nanofabrication. We also review some related work and identify the key differences in the analytic approach used in this paper.

Section 3 sets out our results on optimally-aligning shape-pairs in a staged fashion, starting with shapes that progressively align using pure translations or pure rotations, and moving on to those whose progressive alignment draws on a combination of these motions.

Our presentation is more illustrative than comprehensive, hopefully raising more interesting questions than we have settled.

2 Background and motivation

The problem of designing/characterizing self-aligning shapes, or, more generally, optimally aligning shape pairs, is a natural geometric problem, worthy of study without further motivation. However, it turns out to have significant practical importance as well. Consider the task, arising in the context of nanofabrication, of producing a particular pattern on a flat surface. The pattern may be printed or etched (*e.g.* lithography), or individual components placed precisely until the pattern emerges. These are examples of pattern assembly directed by an external process. In contrast, autonomous *self-assembly* is the process by which a designed pattern emerges by assembling itself from constituent parts.

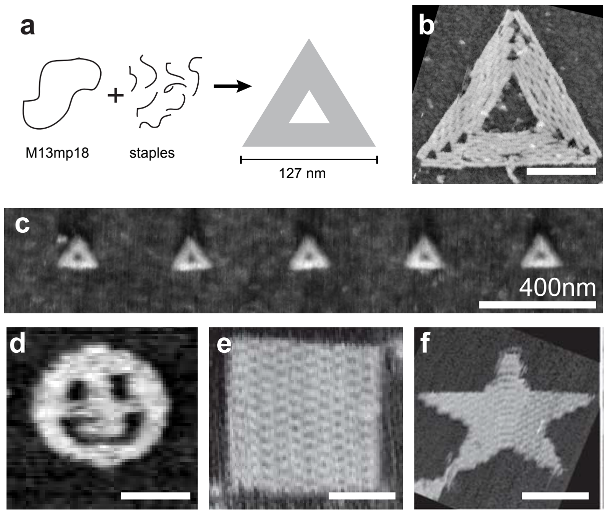


Figure 3: (a) DNA origami technology can fabricate 2D shapes on the nanoscale, such as a triangular annulus. (b) An atomic force microscope image of the triangular annulus. (c) Triangular annuli in a lattice arrangement on a surface. (d)–(f) Atomic force microscope images of other shapes realized with DNA origami. Scale bars in (b) and (d)–(f) are 50 nanometers.

A very successful example of self-assembly, driven by molecular forces, is DNA origami [6] which has become a foundational technology in nanoscience [2]. The process involves a long *scaffold* sequence of DNA — typically a circular plasmid — and a collection of short *staple* sequences that are complementary to two or more regions on the scaffold. When annealed in an appropriate buffer solution, each staple strand "pinches" regions of the scaffold together according to their designed complementarity. Typically, this process can result in the assembly of $\approx 10^{10}$ copies of a designed 2D or 3D shape, with feature resolution of 6 nanometers (nm) [6]. In contrast, the smallest resolution of any feature in current CMOS processes is 14 nm.

Efforts have sought to place these DNA origami on a surface. Kershner *et al.* [4] and Gopinath & Rothemund [3] demonstrated a combination of directed- and self-assembly by creating a surface with a regular lattice of triangular patches, via ebeam lithography, to which triangular shaped origami could bind (see Fig. 3c).

That work was the original motivation for our study of the progressive alignment of shapes since (i) origami can initially land (from solution) onto the surface[1] in any orientation or translation relative to a patch, and (ii) the electrostatic force between patch and origami, coupled with stochastic perturbation, drives a process to improve the binding between the pair. To ensure an origami cannot become "stuck" in a degenerate placement, the energy landscape exhibited by the patch-origami pair should have the following property: from *all* initial placements there should exist a path, to one or more of the designated final placements, that monotonically increases the number of chemical bonds (in this case salt-bridges) between the origami and surface. We model this in the shape alignment problem by ensuring there is a progressive alignment from any initial placement to a designated final placement.

While equilateral triangles have been demonstrated experimentally to place well, each origami may be in one of three final states due to the 3-fold rotational symmetry of the patch-origami pair. It is natural to ask whether there exists a patch-origami pair that has a *unique* local maximum of overlap. Such a pair would enable placement of a single molecule on a surface, with 6 nm precision, since molecules can be attached to an origami at this same resolution. In this way, DNA origami acts as a molecular *breadboard.*

Böhringer and co-authors [5, 7, 1] encountered the problem of designing self-aligning shapes in the context of the study of surface-tension driven self-assembly of micro-components in microelectromechanical systems (MEMS). Their analysis is based on a first-order approximation of the energy model that reduces the problem to the self-alignment question posed above. They performed an experimental evaluation of a wide variety of shapes, and an exact analysis of a family of shapes formed by addition and subtraction of solid disks.

Their exact analysis led to a complete characterization of the shapes, called death-stars below, that are guaranteed to self-align. In a nutshell, this analysis relies on the simplicity of regions formed by intersecting disks to give a precise expression for the area of intersection of death-stars in a specified placement, as well as its derivative with respect to a specified direction of motion. This approach is simply not feasible for more complicated shapes that necessarily arise in our intended application. Furthermore, it is focused on optimizing certain shape parameters, ultimately through the numerical solution of certain systems of constraints, that, for the kind of simple alignment paths that arise in practice, are rather easy to approximate using more straightforward geometric arguments.

The analysis techniques used in this paper are based on a geometric description of motion paths, and can be used to show a larger class of shape-pairs have the requisite properties. This is important in the context of DNA origami where shapes are rasterized; those whose boundary has low curvature can be approximated well, those with high curvature cannot. Our approach is to view the area of intersection of a target shape T and a placement A_λ of a movable shape A as an integral of infinitesimal strips (referred to as *cuts*) parallel to a specified direction of motion. Then, to confirm that motion from A_λ in this specified direction leads to an increase in the T-overlap, it suffices to analyze and accumulate the change within each cut. (In effect we differ from the previous approach by simply interchanging the order of integration and differentiation.) The utility/versatility of this approach stems from the fact that the boundary of $T \cap A_\lambda$ can have a simple description in terms of portions of the boundary of T and A_λ. Thus, the change within each cut can be characterized by the boundary-type of the endpoints of the intervals of $T \cap A_\lambda$ that live within that cut. We make this approach more concrete in the next section.

3 Examples of certified progressive alignments

3.1 Progressive alignment by translation

Consider again the case of a simple disk shape. Fig. 4 (left) illustrates a typical T-proximate placement of a red disk A relative to a target placement T (blue). If A is translated horizontally, bringing its centre closer to that of T, the area of T-overlap clearly increases. One way of seeing, and quantifying, this is to imagine slicing the overlap lens into infinitescimal cuts, parallel to the direction of translation, and to analyse the (instantaneous) change of length of each cut. As illustrated, every cut is bounded on the left by the boundary of A and on the right by the boundary of T, and so each cut increases in length in proportion to the length of the translation of A. It follows that the rate of increase in overlap is proportional to the vertical extent (i.e. the length of the vertical projection) of that portion of the boundary of A that forms part of the left boundary of the intersection.

Note that if shape A is replaced by any shape whose smallest enclosing disk coincides with A, then we can, in a similar way, measure the rate of increase in overlap by the length of the vertical projection of that portion of the boundary of A that forms part of the left boundary of the intersection, minus the (necessarily smaller) length of the vertical projection of that portion of the boundary of A that forms part of the right boundary of the intersection.

To see a slightly more general situation, consider the case of an annulus (see Fig. 4 (right)). In this case the intersection in a typical T-proximate placement is a region that, when sliced into infinitescimal horizontal cuts as before, has some cuts bounded on the left by the

[1]A 2D (flat) origami can be designed to ensure that only one of its two faces can stick to the surface.

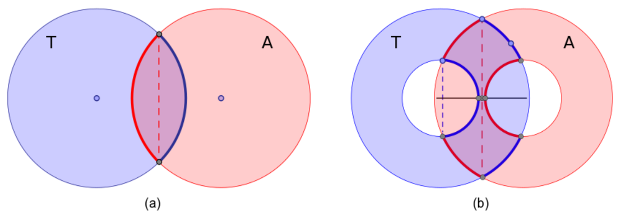


Figure 4: (a) disk and (b) annulus placements.

boundary of A and others by the boundary of T. Similarly for the right end of the cuts. As before, cuts that are bounded on the left by the boundary of A and on the right by the boundary of T increase in length as A is translated left. However, just the opposite is true for cuts with opposite bounding conditions. Furthermore, cuts bounded on both ends by the boundary of the same shape do not change in length under infinitescimal motion of A. Thus we can measure the total rate of change by accumulating the length of the vertical projection of the portion(s) of the boundary of A that coincide with the left boundary of the intersection, minus the length of the vertical projection of the portion(s) of the boundary of A that coincide with the right boundary of the intersection, plus the length of the vertical projection of the portion(s) of the boundary of T that coincide with the right boundary of the intersection, minus the length of the vertical projection of the portion(s) of the boundary of T that coincide with the left boundary of the intersection.

To ensure that a translation taking the centre of A onto the centre of T is progressive (i.e. the rate of change is always positive) for an annulus, it suffices to choose the radius of the inner circle of the annulus in such a way that for all T-proximate placements the height of the lens (red dashed segment) is at least twice the height of the inner and outer circle intersection (dashed blue segment). It is easy to confirm that, subject to this constraint, the rate of change is minimized when the annulus "holes" are disjoint. It follows that, with this same choice of inner radius, the same annulus and target, with arbitrary content inside the inner circles, will progressively align to a configuration in which the annulus and target centres coincide.

3.2 Progressive alignment by rotation

Of course, even if a shape is self-aligning, it is only coincidentally possible to achieve this by translation alone. Fortunately, our analysis of progressive alignments using pure translation has a direct counterpart for pure rotations. As before it is helpful to illustrate this with a simple example. Fig. 5 (left) illustrates a proximate placement of a yin-yang shape A (red) with respect to a congruent target T (blue). It is not surprising that translation to a configuration (Fig. 5 (right)) in which the centres coincide is not progressive. Nevertheless, once the centres coincide a rotation about the centre suffices to complete the alignment in a progressive fashion.

In the case of the T-proximate yin-yang configuration in Fig. 5 (right), observe that at every radius r no larger than the outer radius the intersection of the circle of radius r (the counterpart of an infinitesimal strip) with the current placement-target overlap is an arc bounded on the counter-clockwise end by a portion of the boundary of A and on the clockwise end by a portion of the boundary of T. Thus a counter-clockwise rotation of A will increase the length of all such arcs, ensuring that the intersection increases until it reaches it unique maximum.

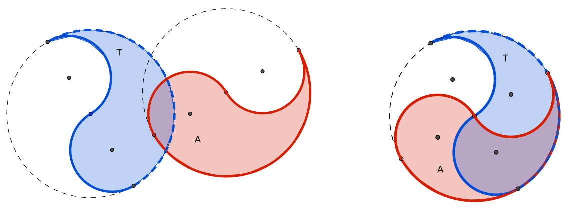


Figure 5: yin-yang shape placements.

Of course, many other shapes, including a half circle, enjoy this same progressive rotational alignment property. We have illustrated the yin-yang shape because its central, but non-reflective, symmetry turns out to be useful in certain applications.

3.3 Progressive alignment by translation then rotation

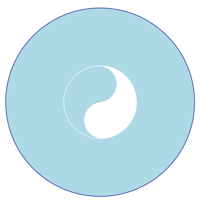


Figure 6: a self-aligning shape.

Combining the observations of the previous two sections, it follows directly that an annulus with a sufficiently small inner radius whose hole circumscribes a copy of the yin-yang shape (see Fig. 6) is self-aligning.

3.3.1 Self-alignment of disk with offset hole

Another way to realize a self-aligning shape is to break the symmetry of an annulus and offset the "hole". As previously noted, such a family of shapes was the focus of a detailed analysis by Böhringer et al. [5, 7]. Here we observe that a straightforward application of the approach discussed in the previous two subsections allows us to draw conclusions about the specifications for such shapes, and generalizations, that ensure the self-aligning property.

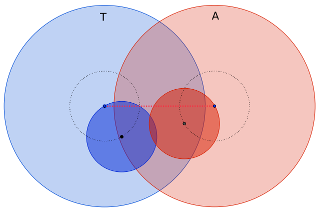


Figure 7: a generic death-star placement.

For obvious reasons we refer to the unit-radius disk with an offset hole of radius r as a "death-star" (although our intended application involves a realization at a considerably different scale). Fig. 7 illustrates a death-star A (red) in a proximate placement with respect to a target T (blue). Here holes are depicted in bold color to help in visualizing their interaction.

We will maintain the invariant that the star centres lie on the x-axis. Our progressive alignment in all cases consists of two phases. Phase 1 is a simple translation, taking the centre of star A onto the centre of star T. Phase 2 is a rotation of star A about its centre so as to reduce the separation of the hole centres to zero. It is easy to see that, provided the holes overlap the star centre, phase 2 monotonically increases the star overlap, since the hole intersection increases monotonically. Thus it remains to analyze the behaviour of phase 1.

We first observe that in the absence of intersections between holes and the opposite star, which is unavoidable when the distance between the star centres is at least $2r + 1$, a translation of A towards T increases the intersection of the stars in proportion to the height of the lens formed by the intersection of the disks A and T (just as in the case of intersecting disks discussed earlier). More generally, for smaller star separations, when the holes may intersect their opposite star, we can, as before, view the star intersection (which of course is a subset of the lens) as being made up of infinitesimal cuts, whose length may increase, decrease or remain unchanged as a result of such a translation.

As described in the analysis of annulus alignments, cuts are of four types depending on the boundary types of their left and right delimiters. Cuts delimited on the left by a portion of the A star boundary and on the right by a portion of the T star boundary, increase in length in proportion to the length of the translation. Cuts delimited on the left and right by portions of the boundary of the same star do not change length with such translations. Finally, cuts delimited on the left by a portion of the T star boundary and on the right by a portion of the A star boundary, decrease in length in proportion to the length of the translation. Since the right (respectively, left) boundary of the A disk (respectively, T disk) is disjoint from the lens, it follows that every shrinking cut connects a point on the right boundary of the T hole to a point on the left boundary of the A hole, and every unchanging cut involves a point on the right boundary of the T hole or a point on the left boundary of the A hole. Thus there is an expanding cut at every height that intersects the lens, but neither of the hole boundaries within the lens. Hence the star intersection increases at least in proportion to the height of the lens minus the sum of lengths of the vertical projections of the hole boundaries within the lens. (Here we have used the fact that cuts that intersect both hole boundaries are double counted in the vertical projections. Furthermore, we have not discounted the possible hole intersection, which can only reduce the shrinking cuts, or those portions of the right boundary of the B hole, or the left boundary of the B hole, that intersect the lens, which can only add to the expanding cuts.)

Thus, it remains to demonstrate that in all configurations the height of the lens exceeds the sum of lengths of the vertical projections of the hole boundaries within the lens. Note that this is obvious if the distance between A and T is less than $4r$ (a necessary condition for the holes to intersect) provided r is chosen so that the height of the lens is at least $4r$, in this case.

When the distance between A and T is at least $4r$, we observe that the length, as well as the vertical projection, of both hole boundaries within the lens is maximized when the distance between the centre of A and the centre of the T hole (respectively, the centre of T and the centre of the A hole) is minimized, i.e. the hole centres lie on the x-axis (see Fig.8). So, we hereafter study configurations in which both hole centres lie on the x-axis. Note: this does not suggest that our transformation rotates the stars to realize such a configuration; in fact such a rotation would in general reduce the start intersection.

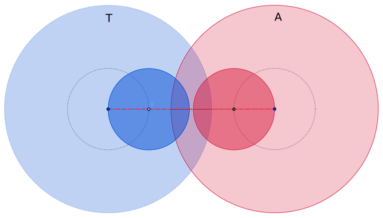


Figure 8: a placement when both hole centres lie on the x-axis.

It remains to analyse the rate of change of the star overlap as A is translated to make its centre coincident with that of T. As in the case of the annulus, this can be achieved by studying the length of the vertical projections of the portions of the boundary of A and T that delimit the left and right boundaries of $A \cap T$. Since these projections change in a monotonic way with a decrease in the separation of A and T, it is straightforward to establish a largest value for r that will ensure that this translation phase is progressive.

As with the annulus we note that if the star A is modified in such a way that its hole is a proper subset of the A hole then the translation of this shape to T remains progressive. Of course, there is no guarantee that the second (rotation) phase would remain progressive with such a modification, but our argument for progressive rotational alignments allows us to make such an assertion in certain special cases, for example when the A hole is reduced to an inscribed square.

3.3.2 Progressive alignment of regular polygons

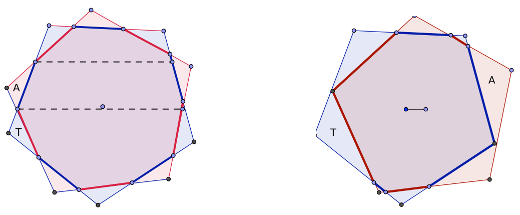


Figure 9: placements of regular 5-gons.

As a final example of the application of our approach to the certification of progressive alignments using translation and rotation, we consider the natural question of whether every k-regular polygon A progressively aligns to one of its maximally overlapped configurations with a congruent target T (see Fig. 9). It is straightforward to see that this is true if the intial and final configurations share a common centre. Thus it suffices to argue that, for any T-proximate placement A_λ, a translation taking A to a placement A_0 whose centre coincides with the centre of T, is progressive.

To confirm that this is the case we imagine any translation taking A from a centre-aligned placement A_0 in any direction, and argue that the overlap $A \cap T$ decreases with distance. This follows from several simple observations. First, we note that the boundary of the intersection $A_0 \cap T$ is semi-regular convex polygon consisting of $2k$ equal length sides drawn in alternating fashion from the sides of A_0 and T. (This is easily demonstrated by appealing to mirror symmetry across any line through the common centre and any point of intersection of the boundaries of A_0 and T.)

It follows from this that contracting the sides of the boundary of $A_0 \cap T$ that correspond to boundary edges of A_0 (respectively T) forms a regular k-gon. Thus, when we accumulate the vertical projections of portions of the boundaries of A_0 and T that determine the boundary of $A_0 \cap T$ we get a net change of zero for any instantaneous translation from the placement A_0.

The second observation is that, for any placement formed by an infinitescimal translation from placement A_0, the magnitude of the associated projections that count positively in the accumulated projection length decrease with distance, and those that contribute negatively increase. This demonstrates that for all placements in the immediate neighbourhood of a centre-aligned placement the derivative of the change of overlap, with respect to distance, is negative, confirming that any centre-aligned placement is a local maximum with respect to all translations.

To confirm that every centre-aligned placement is in fact a *global* maximum, with respect to all translations, we note that the second observation extends to translations that take A from A_0 to any placement whose symmetric difference with T contains three or more components. (To demonstrate this we cut the overlapped placements into strips, parallel to the direction of translation, bounded by lines through each of the polygon vertices, and accumulate the change of overlap in each strip separately.)

Finally, we note that if A has been translated to a T-proximate placement whose symmetric difference with T has two components, then an argument analogous to that used in the limiting case of disks (recall Fig. 4(a)) shows that such placements can be improved by any infinitescimal translation in the direction that takes the centres back into alignment.

4 Conclusion and Future Work

We have identified an interesting property of planar shapes that models a desirable attribute of self-assembling nanofabricated structures. We have identified a simple approach to the certification of this progressive alignment property and illustrated it with some fundamental examples, some of which are beyond the reach of earlier approaches, and all of which serve as potential building blocks for the design of shapes with other desirable properties in a nanofabrication context.

Our study of self-aligning shapes is still in a preliminary state. One indication of this is that at this point we still do not know if there exist convex shapes that are self-aligning. Furthermore, while we have reasonable tools for certifying progressive alignments that are composed of pure translations and pure rotations, these do not allow us to address arbitrary rigid motions, which presumably could be required for some progressive alignments.

Acknowledgments

The authors are supported by the Natural Sciences and Engineering Research Council of Canada (DK), a Banting Fellowship (CT), the Office of Naval Research (award N000141410702) (AG, PWKR), the US National Science Foundation (award 1317694) (AG, PWKR), and the Molecular Programming Project (http://molecular-programming.org) (AG, CT, PWKR). We thank W. Evans for helpful discussions and the anonymous reviewers for their feedback.

References

[1] K. F. Böhringer, U. Srinivasan, and R. T. Howe. Modeling of capillary forces and binding sites for fluidic self-assembly. In *Micro Electro Mechanical Systems, 2001. MEMS 2001. The 14th IEEE International Conference on*, pages 369–374. IEEE, 2001.

[2] Editorial. Returning to the fold. *Nature Materials*, 15(3):245–245, 2016.

[3] A. Gopinath and P. W. Rothemund. Optimized assembly and covalent coupling of single-molecule DNA origami nanoarrays. *ACS nano*, 8(12):12030–12040, 2014.

[4] R. J. Kershner, L. D. Bozano, C. M. Micheel, A. M. Hung, A. R. Fornof, J. N. Cha, C. T. Rettner, M. Bersani, J. Frommer, P. W. Rothemund, et al. Placement and orientation of individual DNA shapes on lithographically patterned surfaces. *Nature Nanotechnology*, 4(9):557–561, 2009.

[5] S.-H. Liang, X. Xiong, and K. F. Böhringer. Towards optimal designs for self-alignment in surface tension driven micro-assembly. In *Micro Electro Mechanical Systems, 2004. 17th IEEE International Conference on.(MEMS)*, pages 9–12. IEEE, 2004.

[6] P. W. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[7] X. Xiong, S.-H. Liang, and K. F. Böhringer. Geometric binding site design for surface-tension driven self-assembly. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1141–1148. IEEE, 2004.

Rectangle-of-influence triangulations

Therese Biedl* Anna Lubiw* Saeed Mehrabi* Sander Verdonschot†

1 Background

The concept of *rectangle-of-influence (RI) drawings* is old and well-studied in the area of graph drawing. A graph has such a drawing if we can assign points to its vertices such that for every edge (v, w) the *supporting rectangle* (i.e., the smallest closed axis-aligned rectangle $R(v, w)$ containing v and w) contains no other points. In the original setup, the graph had to have an edge for every pair of points with an empty supporting rectangle (the *strong* model, see e.g. [8]). Later papers focus on *weak* RI-drawings, where for every edge the supporting rectangle must be empty, but not all such edges must exist. Of particular interest are planar weak RI-drawings of planar graphs, since these can always be deformed to reside on an $n \times n$-integer grid. See e.g. [9, 11, 1].

Our Results: In this paper, we take two computational geometry problems—how to triangulate a point set and how to flip between two geometric triangulations—and apply them in the setting of weak rectangle-of-influence drawings. In particular, we show that any point set can be triangulated (with some exceptions near the convex hull edges, which we show to be necessary) such that the resulting planar straight-line graph (PSLG) is an RI-drawing. Next, we turn to the problem of *flipping* among geometric triangulations, i.e., converting one triangulation into another through the operation of flipping the diagonal of one quadrangle. We show that any RI-triangulation can be converted into any other RI-triangulation by $O(n^2)$ such flipping-operations (and $\Omega(n^2)$ flips are required for some RI-triangulations). Moreover, all intermediate triangulations are also RI-triangulations. The main idea is that the L_∞-Delaunay-triangulation (defined below) is an RI-triangulation; it hence suffices to find one flip-operation that gets the RI-triangulation "closer" to the L_∞-Delaunay-triangulation in some sense. We also study how to flip from any triangulation to an RI-triangulation while getting "closer".

Existing literature: Triangulating point sets and polygons is one of the standard problems in computational geometry. Any set of n points can be triangulated in $O(n \log n)$ time (e.g. by computing the Delaunay triangulation), and the interior of a polygon can be triangulated in $O(n)$ time [4]. The Delaunay triangulation has been generalized to other "unit discs". In particular, for any convex compact shape C, the C-Delaunay-triangulation is a triangulation such that for every edge (v, w) there exists a homothet of C that contains v and w and no other points [5]. Aurenhammer and Paulini [2] studied a number of their properties. If C is a unit square (i.e., the unit-circle in the L_∞-metric), then this triangulation is called the *L_∞-Delaunay-triangulation.* The L_∞-Delaunay-triangulation can be computed in $O(n \log n)$ time [5]. Observe that the L_∞-Delaunay-triangulation is an RI-triangulation, but an RI-triangulation need not be a C-Delaunay-triangulation for any C because the supporting rectangles are not necessarily homothets of each other or expandable into such.

Flipping among triangulations is also a well-studied problem; see [3] for an overview of many variants and existing results. It is very easy to see that any (geometric) triangulation T_1 can be flipped into any other triangulation T_2 via the intermediary of the Delaunay triangulation T_D: We can always find a flip that gets us closer to the Delaunay triangulation (in the sense that some angle-sum increases), so keep flipping from T_1 until we reach T_D. Also compute the flips from T_2 to T_D, and reversing these flips and combining the two flip-sequences then gives the result. For C-Delaunay-triangulations, a similar result holds: we can always flip to get "closer" to the C-Delaunay-triangulation [2].

Notation: Let P be a set of n points that we assume to be in *general position* in the sense that no two points are on a horizontal or vertical line, and no 4 points are on a square. For any two points $p, q \in P$, define the *supporting rectangle* $R(p, q)$ to be the minimal axis-aligned rectangle containing p and q. Define a *supporting square* $S(p, q)$ to be a minimal axis-aligned square containing p and q; $S(p, q)$ is not unique. For any two points $p, q \in P$, we call a supporting rectangle/square of (p, q) *empty* if it contains no points of P other than p and q.

An edge (p, q) between points of P is called an *RI-edge* (*L_∞-edge*) if $R(p, q)$ is empty (resp., some supporting square $S(p, q)$ is empty). Note that an L_∞-edge is an RI-edge. An *RI-polygon* is a polygon for which every edge is an RI-edge. A planar straight-line graph (PSLG) is a *triangulation* if every interior face

*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, {alubiw,biedl, s2mehrabi}@uwaterloo.ca. TB and AL supported by NSERC.

†Computational Geometry Lab, School of Computer Science, Carleton University, Ottawa, Ontario, Canada. sander@cg.scs.carleton.ca

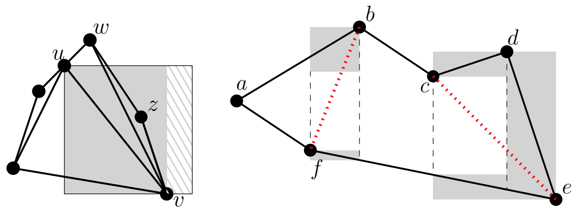


Figure 1: (Left) A triangulation. (v,w) is not locally RI. (u,v) is locally L_∞ (therefore locally RI), but not globally L_∞. (Right) A polygon (solid) with its trapezoidation (thin dashed). Edge (b,f) would be added with the first method, edge (c,e) with the second.

of the PSLG is a triangle. An *RI-triangulation* (L_∞-*triangulation*) is a triangulation for which every edge is an RI-edge (L_∞-edge). We sometimes use "unrestricted" triangulation for a triangulation that need not be an RI-triangulation. A triangulation is *maximal* if it contains as many edges as possible while staying within the additional requirements that we impose. Thus, a *maximal RI-triangulation* is a triangulation for which every edge is an RI-edge and no RI-edge can be added without violating planarity; *maximal* L_∞*-triangulation* is defined similarly.

For an edge (u,v) in a triangulation, vertex w is *facing* (u,v) if there exists an interior face $\{u,v,w\}$. Every interior edge has exactly two vertices facing it. We say that (u,v) is *locally RI* (resp. *locally* L_∞) if $R(u,v)$ (resp.. some supporting square $S(u,v)$) contains none of the vertices facing (u,v). Sometimes we say that an RI-edge (L_∞-edge) is *globally RI* (*globally* L_∞).

2 RI-triangulating an RI-polygon

We first show that any RI-polygon P can be *RI-triangulated*, i.e., made into a triangulation by adding only RI-edges in its interior (presuming no extra points are inside P). To do so, first find a trapezoidation of P, i.e., extend vertical subdivision lines from all vertices. This can be done in linear time [4].

We add RI-edges in two ways. First, check whether there is any trapezoid that has vertical subdivision lines on both its left and right sides, and for which the two vertices v,w that caused these lines were not adjacent yet. If so, add edge (v,w). This is an RI-edge since $R(v,w)$ is contained within the supporting rectangles of the top and bottom edge of T and the interior of P, all of which are empty. See edge (b,f) in Fig. 1.

Secondly, if no such trapezoid exists, then any remaining face is x*-monotone*, i.e., it consists of two x-monotone chains from left to right. Since the first method does not apply, one of the two chains is a single edge. Consider one such piece with (say) the bottom chain a single edge (v,w). All vertices in the top chain are outside $R(v,w)$ and hence have larger y-coordinate than v,w. Let u be a local maximum in the top chain, and connect the neighbors of u. The new edge is an RI-edge, because its supporting rectangle is contained in the union of the ones of the edges incident to u as well as $R(v,w)$ and the interior of P. See edge (c,e) in Fig. 1 (with $v{=}f, w{=}e, u{=}d$).

So we add RI-edges until all interior faces are triangles. This takes linear time (once the trapezoidation is found), since finding the local minimum/maximum for the second rule can be done in $O(1)$ amortized time.

Theorem 1 *Every RI-polygon can be triangulated using only RI-edges in linear time.*

3 Outer face considerations

The remaining sections deal with triangulations of point sets, rather than polygons. Here there arise some complications at the outer face. For any maximal (unrestricted) triangulation of P, the outer face consists of the convex hull $CH(P)$. If some edge of $CH(P)$ is not an RI-edge, then it obviously cannot be in an RI-triangulation. We begin by characterizing the outer face of any RI-triangulation.

The maxima-hull: We need some definitions that are closely related to the rectilinear convex hull (see e.g. [10]) and the maxima of a set of vectors (see e.g. [7]). Define the *first quadrant* of a point p to be the set $\{(x,y) : x \geq x(p), y \geq y(p)\}$. Define the *first-quadrant chain* C_I to be all those points p in P for which the first quadrant relative to p contains no other points of P; we sort these points by increasing x-coordinate (hence by decreasing y-coordinate), and connect them with straight-line segments in this order. Similarly define three other chains C_{II}, C_{III}, C_{IV} using the three other types of quadrants. Note that C_I and C_{II} share one endpoint (the one with maximum x-coordinate), and similarly for the other chains, so we can combine the four polygonal chains into one closed polygonal chain that we call the *maxima-hull* $MH(P)$. Note that some chains may have edges in common, but no two edges cross, so $MH(P)$ may self-overlap but it has a well-defined interior region. See Fig. 2.

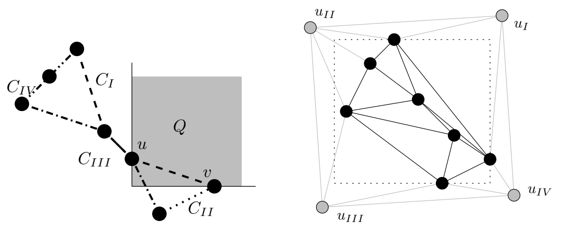


Figure 2: The maxima-hull of a set of points, and how to add corner-points and edges to them.

We need a few observations:

Lemma 2 *Let (u, v) be any edge on the first-quadrant chain C_I, say $x(u) < x(v)$. Let Q be the first quadrant of point $(x(u), y(v))$, i.e., it has u and v on its boundary. Then no point (other than u, v) is in Q.*

Proof. See Fig. 2. Assume to the contrary that Q contained points other than u, v, and let w be the one that maximizes the x-coordinate. Since the first quadrants relative to u and v are empty, w must be in $R(u, v)$. By general position we have $x(u) < x(w) < x(v)$ and $y(v) < y(w) < y(u)$. The first quadrant of w cannot contain any other point, for all such points would be to the right of w (contradicting the choice of w). So point w should have been in C_I, contradicting that u, v were consecutive in C_I. □

Lemma 3 *For any point set P, the maxima-hull consists of L_∞-edges (hence RI-edges).*

Proof. Note that for any edge (u, v) on C_I, rectangle $R(u, v)$ is part of this first quadrant, so it is empty, and we can expand it into a square supporting u, v that is empty. So any edge on C_I is an L_∞-edge. Similar arguments hold for the other three quadrant-chains. □

Lemma 4 *For any point set P, any RI-edge (u, v) is within the region bounded by $MH(P)$.*

Proof. We aim to show that any RI-edge (u, v) is on or below $C_I \cup C_{II}$. Similar proofs in the other three directions show that (u, v) is either on or enclosed by the maxima-hull as desired.

Consider the maximal vertical strip S containing (u, v). This strip must intersect $C_I \cup C_{II}$, since the rightmost point of P is in C_I and the leftmost of P is in C_{II}. Let C be the part of $C_I \cup C_{II}$ within S, say its ends are p_u (on a vertical line with u) and p_v (on a vertical line with v). See also Fig. 3. Applying Lemma 2 (or the equivalent for second quadrants) to the edge containing p_u shows that the vertical ray upward from p_u contains no other points of P. So either $p_u = u$, or p_u is above u. Similarly $p_v = v$ or p_v is above v.

In what follows, we use the term "vertex of C" for a point on C that is also a point of P, while "point of C" refers to an arbitrary point that belongs to C. If C has no vertices, then it is a single line segment $\overline{p_u p_v}$, and by the above (u, v) is below that. So assume that C has vertices.

Since C (as part of $C_I \cup C_{II}$) consists of an increasing chain followed a decreasing chain, its minima (with respect to y-coordinate) appear at the ends. Since p_v is above v, therefore all vertices of C have y-coordinate at least $y(v)$. Since none of these vertices are inside $R(u, v)$ (recall that (u, v) is an RI-edge), they in fact must have y-coordinate at least $y(u)$, hence be above $R(u, v)$. In consequence the only edge of C that can intersect $R(u, v)$ is the edge incident to p_v, but this edge is also above (u, v) since p_v is above v. So all of C is above (u, v) as desired. □

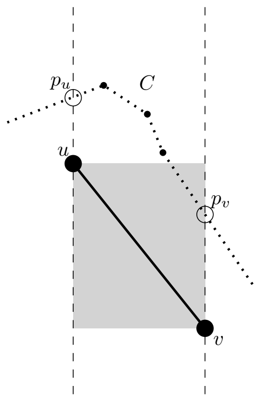


Figure 3: For the proof of Lemma 4.

.

Combining the last two lemmas gives:

Corollary 1 *An RI-triangulation is maximal if and only if its outer face consists of the maxima-hull.*

Adding corner-points: It will be cumbersome to deal directly with edges that are on the convex hull but not RI-edges. To simplify our life, we add points as follows. For any point set P, let P^+ be the set obtained by adding four *corner-points* $u_I, u_{II}, u_{III}, u_{IV}$ that form an axis-aligned rectangle (we rotate it slightly to be in general position) and are outside the bounding box of P, with u_i in the ith quadrant relative to all points of P. See Fig. 2. Notice that the convex hull of P^+ consists of L_∞-edges.

Lemma 5 *Let T^+ be a maximal RI-triangulation of P^+, and let $T := T^+ - \{u_I, u_{II}, u_{III}, u_{IV}\}$. Then T is a maximal RI-triangulation of T.*

Proof. Clearly any edge of T is an RI-edge, so we only need to argue maximality. Assume (p, u_I) is an edge in T^+ for some $p \neq u_{II}, u_{III}, u_{IV}$. Then $R(p, u_I)$ contains no other point, and is to the right and/or above u_{II}, u_{III}, u_{IV}. So expanding $R(p, u_I)$ into the first quadrant of p does not add points of P, hence p is on the maxima-hull. So removing $\{u_I, u_{II}, u_{III}, u_{IV}\}$ from T^+ leaves an RI-triangulation where the outer face is the maxima-hull. By Corollary 1 this is maximal. □

4 RI-triangulating a point set

In this section, we study how to find a maximal RI-triangulation of a given point set. It is obvious that this exists (for example the L^∞-Delaunay triangulation will do), but our algorithm is especially simple.

Theorem 6 *A maximal RI-triangulation of a point set P can be computed in $O(n \log n)$ time, or $O(n)$ time if P is sorted by x-coordinate.*

Proof. As before, add corner-points $u_I, u_{II}, u_{III}, u_{IV}$ to obtain point set P^+. Sort the points by x-coordinates, and add an edge between any two consecutive points; these are clearly RI-edges. Also add the cycle $u_I, u_{II}, u_{III}, u_{IV}$; these are also RI-edges. Now we have a PSLG whose faces consist of RI-polygons. Triangulate each polygon with Theorem 1. We obtain an RI-triangulation T^+ of P^+, and it is clearly maximal since all convex-hull edges are in it. By Lemma 5, deleting the four corner-points gives the result. □

5 Flipping and RI-triangulations

In this section, we investigate flipping while maintaining RI-triangulations or (if we start with an unrestricted one) getting closer to an RI-triangulation. The natural "intermediary" here is the L_∞-Delaunay triangulation, which is an RI-triangulation. So we show that any triangulation can be flipped to an RI-triangulation while getting "closer", and then that any RI-triangulation can be flipped to the L_∞-Delaunay triangulation while remaining an RI-triangulation throughout.

5.1 Flipping to an RI-triangulation

Let P^+ be a point set for which any convex hull edge is an RI-edge (we will argue later how to remove this assumption). Let T be an arbitrary triangulation of P^+. A *bad triangle* $\{u, v, w\}$ in T is a face $\{u, v, w\}$ such that $v \in R(u, w)$. After possible rotation, assume that u is in the 2nd quadrant and w is in the 4th quadrant relative to v, with edge (u, w) above v. Define the *special region* of bad triangle $\{u, v, w\}$ to be all points p above (u, w) with $x(u) \leq x(p) \leq x(v)$ (including u) and all points p to the right of (u, w) with $y(v) \leq x(p) \leq y(w)$ (including w). See Fig. 4(a). The definition is symmetric for the other three possible rotations of a bad triangle. Now define for any triangulation T the potential function $\Phi(T)$ to be the sum, over all bad triangles $\{u, v, w\}$, of the number of points in the special region of $\{u, v, w\}$.

Lemma 7 *Let T be any triangulation of P^+ with $\Phi(T) > 0$. Then there exists an edge in T that we can flip so that the resulting triangulation T' satisfies $\Phi(T') < \Phi(T)$.*

Proof. Since $\Phi(T) > 0$, it must have at least one bad triangle, and hence edges that are not locally RI. Of all those edges, let (u, w) be the one that maximizes the L_1-distance between its endpoints, thus $|x(u) - x(w)| + |y(u) - y(w)|$ is maximum among all edges that are not locally RI. Since convex hull edges are RI-edges, we know that (u, w) is an interior edge and has two facing vertices. Let v be a vertex facing (u, w) that is in $R(u, w)$, and assume again that (after possible rotation) the bad triangle $\{u, v, w\}$ has u (or w) in the 2nd (4th) quadrant of v with (u, w) above v.

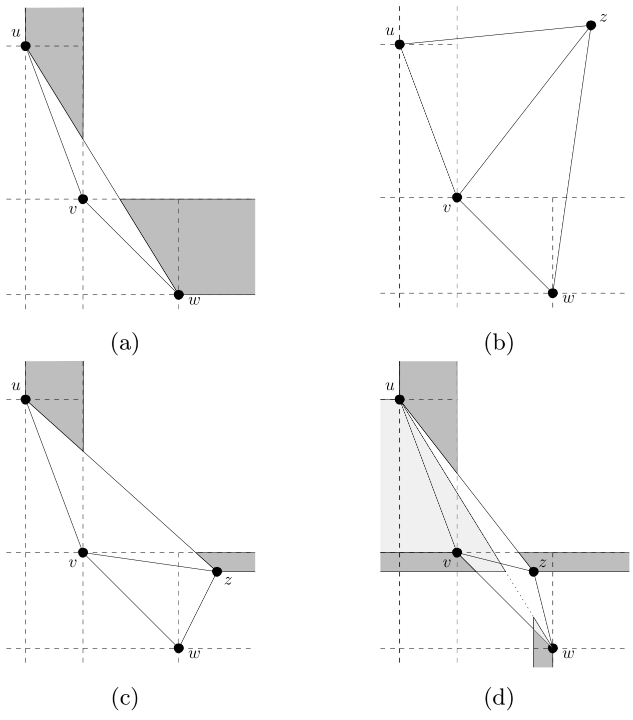


Figure 4: (a) The special region (shaded) of bad triangle $\{u, v, w\}$. (b-d) Possible positions of the other facing vertex z. Light gray regions were in the special region of the bad triangle $\{u, z, w\}$.

Let z be the other vertex facing (u, w). We claim that $x(z) > x(u)$. We know that z is above the line through (u, w) since $\{z, u, w\}$ is a face. If we had $x(z) < x(u)$, then $u \in R(z, w)$, and so (z, w) is not locally RI but its L_1-distance is longer than the one of (u, w), contradicting our choice of edge (u, w).

Therefore we know $x(z) > x(u)$, and symmetrically one argues $y(z) > y(w)$. Notice that therefore quadrilateral $\{u, v, w, z\}$ is convex and so edge (u, w) is flippable. We claim that regardless of the position of z, doing this flip improves the potential function. To show this, we distinguish where z is located

- $x(z) > x(v), y(z) > y(v)$ (see Fig. 4(b)): In this case, neither of the two new triangles $\{u, z, v\}$ and $\{v, z, w\}$ is bad. Since triangle $\{u, v, w\}$ used to be bad, Φ decreases by at least 2.
- $x(z) > x(w), y(z) < y(v)$ (see Fig. 4(c)): In this case, the new triangle $\{u, z, v\}$ is bad, but $\{v, z, w\}$ is not. The special region of triangle $\{u, v, z\}$ is a strict subset of the one for triangle $\{u, v, w\}$, and in particular, excludes w. Hence Φ decreases.

- $z \in R(v,w)$ (see Fig. 4(d)): In this case, both new triangles $\{u,v,z\}$ and $\{v,w,z\}$ are bad. But both triangles $\{u,v,w\}$ and $\{u,z,w\}$ that were removed were bad, and the special regions of the new triangles are strict subsets of the special regions of the old triangles that, in particular, contain u and w only once, instead of twice. So again Φ decreases.

The cases for $x(z) < x(v), y(z) > y(u)$ and for $z \in R(u,v)$ are symmetric. □

Note in particular that if $\Phi(T) = 0$, then it has no bad triangles (because any bad triangle has at least two points in its special region); therefore it has no edge that is not locally RI.

Lemma 8 *Let T be a triangulation such that all edges are locally RI and all outer face edges are globally RI. Then all edges are globally RI.*

Proof. Suppose that some edges of T are not globally RI, hence contain points inside their supporting rectangles. Let $e = (u,v)$ be the edge with the closest (with respect to Euclidean distance) such point, say z. Since e is an interior edge by assumption, it has two facing vertices; let w be the vertex facing e that is on the same side of the supporting line of e as z is. Suppose that z and w lie right of e, and u is the left endpoint of e; see Fig. 1. Observe that w must lie either above $R(u,v)$ (within the same x-range) or to the right of $R(u,v)$ (within the same y-rage), else some edge of $\{u,v,w\}$ would not be locally RI or $\{u,v,w\}$ would not be a face. Assume w lies strictly above $R(u,v)$, within the same x-range. Then (v,w) is also not globally RI, since z lies in $R(v,w)$. But z is closer to (v,w) than to e, contradicting our choice of e. □

Putting the two lemmas together, we can hence flip from any triangulation to an RI-triangulation while steadily improving the potential-function. Initially there are $O(n)$ bad triangles, each of which defines a special region containing at most n points, so $\Phi(T) \in O(n^2)$. Each flip improves the function, and we are done when it is 0, which means that the number of flips is $O(n^2)$. We summarize:

Lemma 9 *Any maximal triangulation of P^+ can be converted into an RI-triangulation of P^+ using $O(n^2)$ flips.*

We can argue that this bound is sometimes tight.

Lemma 10 *There are triangulations that cannot be converted into an RI-triangulation with $o(n^2)$ flips.*

Proof. Consider a set of points spread evenly over two opposing convex chains, such that the only possible RI-edges between the two chains connect point i on the first chain to point i and $i+1$ on the opposing chain (see Fig. 5). The edges connecting consecutive points on each chain are not intersected by any other possible edge, which implies that these edges are present in every triangulation and can never be flipped. Thus, the region between the two chains is independent of the outside regions. Further, by construction, this center region has a unique RI-triangulation (Fig. 5, right). Now consider a triangulation that includes the long diagonal connecting one end of the first convex chain to the opposite end of the other chain (Fig. 5, left). Transforming this triangulation into the unique RI-triangulation requires $\Omega(n^2)$ flips, by an argument analogous to the one given by Hurtado et al. [6, Theorem 3.8] □

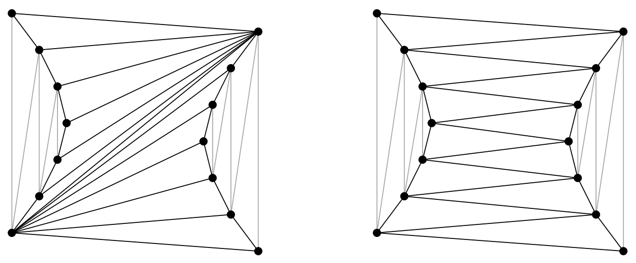


Figure 5: Turning the triangulation on the left into an RI-triangulation requires $\Omega(n^2)$ flips for the region between the chains to become the only possible RI-triangulation shown on the right.

Arbitrary point sets: It remains to argue how to handle point sets P where not all convex-hull edges are RI-edges. Assume we are given a maximal triangulation T of P, and we would like to flip that to a maximal RI-triangulation T_R. Add corner-points $u_I, u_{II}, u_{III}, u_{IV}$ as before to obtain point set P^+ and connect them in a cycle. Connect u_i (for $i \in \{I, II, III, IV\}$) to all points p on the convex hull of P for which quadrant i contains only p and u_i. This gives a triangulation T^+ of P^+ because the convex hull is the outer face T. See Fig. 2. The convex hull of P^+ consists of RI-edges, so there exists a sequence σ of flips that turns T^+ into a maximal RI-triangulation T_R^+.

Lemma 11 *Throughout flip-sequence σ, all edges incident to corner-points are RI-edges. Therefore any edge being flipped is not incident to a corner-point.*

Proof. The first claim implies the second because flipped edges were not locally RI. We prove the first claim for u_I only. Apart from the edges to u_{II} and u_{IV}, vertex u_I has an edge only to a point p for which the first quadrant is empty, so $R(u_I, p)$ is empty and the claim holds. We now show that any time we flip an edge (u,w) such that the new edge is (v, u_I) for some u, v, w, this new edge is an RI edge. Since u_I is outside the bounding box of P, it is outside $R(u,w)$. Since

(u,w) was not locally RI (by our choice of edges to flip), therefore $v \in R(u,w)$. In the naming of Fig. 4, we have $u_I = z$ and the case of Fig. 4(b) applies since u_I is in the first quadrant of v. We already knew that in this case (v,z) is locally RI. But since $z = u_I$, edge (v,z) is globally RI: $R(v,u_I)$ lies within the union of $R(u,u_I)$ and $R(w,u_I)$ (both empty, because these edges are incident to u_I and hence RI-edges), and the interior of triangles $\{u,v,w\}$ and $\{u,u_I,w\}$ (which are faces). □

We now create a sequence of flips and edge deletions for T that leads to a maximal RI-triangulation by mirroring the flip-sequence of T^+. We maintain the claim that at any time triangulation T equals T^+ with the corner-points removed. Clearly this holds initially. Say the next flip for T^+ was to flip (u,w) to (v,z). We know $u,w \in P$. If $v,z \in P$, then (by induction) the 4-cycle u,v,w,z that exists in T^+ also exists in T, and so we can do the exact same flip in T and the claim holds. Else, one of v,z is a corner-point. Do an edge-deletion in T, i.e., remove edge (u,w) without adding a new one. The claim still holds since one end of (v,z) is not in P.

We end with a triangulation T_R of P that equals T_R^+ with the corner-points removed. By Lemma 5, this is a maximal RI-triangulation.

Theorem 12 *We can convert any maximal triangulation into an RI-triangulation by doing $O(n^2)$ flips and $O(n)$ edge-deletions.*

5.2 Flipping between RI-triangulations

As explained earlier, to flip between maximal RI-triangulations while maintaining an RI-triangulation, it suffices to show that every maximal RI-triangulation can be flipped into the L_∞-Delaunay-triangulation. To prove this, we use again a potential-function argument, but with a different function. We need the following:

Lemma 13 *[2] Let T be a maximal triangulation where all edges are locally L_∞. Then all edges are globally L_∞.*

Our potential function depends on having fixed, for every edge (u,v) of the current triangulation, a particular supporting square $S(u,v)$, and counting the number of points of P in it. We define $\Psi(T)$ to be the sum, over all edges (u,v) of the number of points in $S(u,v)$. When we flip, we are free to choose a supporting square for the new edge.

Lemma 14 *Let T be a maximal RI-triangulation that is not the L_∞-Delaunay triangulation. There exists an edge e such that flipping e results in an RI-triangulation T' and we can assign a supporting square to e such that $\Psi(T') < \Psi(T)$.*

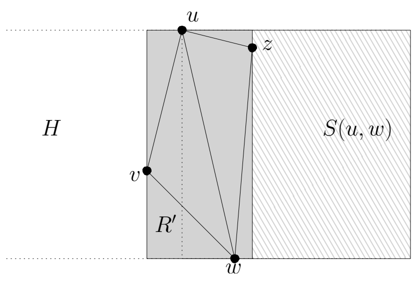


Figure 6: Finding a supporting square for (v,x).

Proof. By Lemma 13 T has some edge (u,w) that is not locally L_∞. Up to symmetry, we may assume that the height Y of $R(u,w)$ is no smaller than its width. All supporting squares of (u,w) are then in the horizontal strip H of height Y between u and w.

Since T is a maximal RI-triangulation, its outer face is the maxima-hull and consists of L_∞-edges. So (u,w) is an interior edge. Let v and z the two vertices facing (u,w). We claim that both v and z must be in strip H. To see this, recall that we have an RI-triangulation, and hence rectangle $R(u,w)$ is empty. This rectangle bisects strip H. So if, say, v is not in H, then at most one facing vertex is in H, which means that to one side of $R(u,w)$ in H there is no facing vertex of (u,v). We could hence pick a supporting square S' of (u,w) that consists of $R(u,w)$ extended to that side. Then S' contains neither v nor z, and (u,w) would be locally L_∞, a contradiction.

So both v and z are in strip H. By planarity they must be on opposite sides of $R(u,w)$, say v is to the left and z is to the right. Quadrangle $\{u,v,w,z\}$ hence is convex and edge (u,v) is flippable. Define R' be the minimum rectangle containing u,v,w,z, and notice that it is contained in the union of the supporting rectangles of the edges $(u,v),(v,w),(u,w),(u,z),(z,w)$. Since we had an RI-triangulation, therefore R' contains no points other than these 4. Also u,v,w,z are all on the boundary of R'. Therefore $R(v,z)$ is empty and the new edge (v,z) is an RI-edge.

Now we explain how to find a supporting square for (v,z). Let X be the width of R', and notice that $X < Y$, since $X = Y$ would imply four points on a square and $X > Y$ would mean that some square within R' supports (u,w) and does not contain v,z, contradicting that (u,w) is not locally L_∞. Now consider the union of R' and the square $S(u,w)$ that was used as supporting square for (u,w). Since (u,w) was not locally L_∞, at least one of $\{v,z\}$ is in $S(u,w)$, hence $S(u,w) \cup R'$ contains at most one more point than $S(u,w)$. Let R'' be the rectangle obtained by shrinking $S(u,w) \cup R'$ to height $Y - \varepsilon$ in such a way that $u,w \notin R''$. We choose ε so small that v and z remain in R'' and such that $Y - \varepsilon > X$. So R'' contains at least one fewer points than $S(u,w)$. Finally shrink R'' in width until it is a

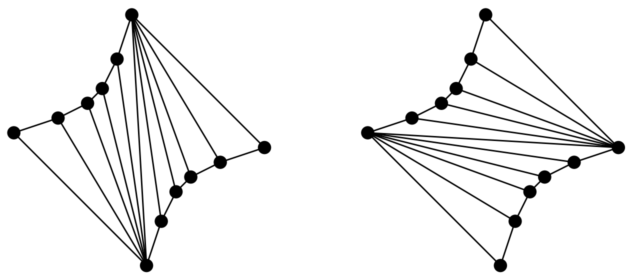


Figure 7: A pair of RI-triangulations such that $\Omega(n^2)$ flips are required to transform one into the other.

square; we can do this and retain v and z in it, since R'' is taller than R' is wide. Using the resulting square for $S(v, z)$ decreases Ψ as desired. □

Theorem 15 *Any maximal RI-triangulation T can be converted into any other maximal RI-triangulation T' by doing $O(n^2)$ flips, and all intermediate triangulations are maximal RI-triangulations.*

Proof. As before it suffices to argue this if T' is the L_∞-Delaunay triangulation. Compute an arbitrary set of supporting squares for T. Initially there are $O(n)$ edges in the triangulation, each of which has at most n points in its supporting square, so $\Psi(T) \in O(n^2)$. Applying the above flip means that with $O(n^2)$ flips we get to the L_∞-Delaunay-triangulation while maintaining an RI-triangulation. □

This bound is tight. Fig. 7 shows two RI-triangulations of a point set that forms two convex chains. Hurtado et al. [6, Theorem 3.8] showed that their flip distance is $\Omega(n^2)$ even without the restriction of using only RI-triangulations.

References

[1] S. Alamdari and T. Biedl. Open rectangle-of-influence drawings of non-triangulated planar graphs. In *Proc. GD'12*, pages 102–113, 2013.

[2] F. Aurenhammer and G. Paulini. On shape Delaunay tessellations. *Inf. Process. Lett.*, 114(10):535–541, 2014.

[3] P. Bose and F. Hurtado. Flips in planar graphs. *Comput. Geom.*, 42(1):60–80, 2009.

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

[5] R. Drysdale. A practical algorithm for computing the Delaunay triangulation for convex distance functions. In *Proc. SODA '90*, pages 159–168, 1990.

[6] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete Comput. Geom.*, 22(3):333–346, 1999.

[7] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[8] G. Liotta, A. Lubiw, H. Meijer, and S. Whitesides. The rectangle of influence drawability problem. *Comput. Geom.*, 10(1):1–22, 1998.

[9] K. Miura, T. Matsuno, and T. Nishizeki. Open rectangle-of-influence drawings of inner triangulated plane graphs. *Discrete Comput. Geom.*, 41(4):643–670, 2009.

[10] T. Ottmann, E. Soisalon-Soininen, and D. Wood. On the definition and computation of rectlinear convex hulls. *Inf. Sci.*, 33:157–171, 1984.

[11] S. Sadasivam and H. Zhang. Closed rectangle-of-influence drawings for irreducible triangulations. *Comput. Geom.*, 44(1):9–19, 2011.

Maximum Area Rectangle Separating Red and Blue Points[*]

Bogdan Armaselu[†] Ovidiu Daescu[‡]

Abstract

Given a set of n red points R and a set of m blue points B, we study the problem of finding the rectangle that contains all the red points, the minimum number of blue points and has the largest area. We call such rectangle a *maximum separating rectangle*. First, we address the planar axis-aligned version, then the more general planar version when the rectangle need not be axis-aligned. Finally, we study the 3D axis-aligned version. For the 2D axis-aligned version, we give an $O(m \log m + n)$ time algorithm. The running time reduces to $O(m+n)$ if the points are pre-sorted by one of the coordinates. For the 2D non axis-aligned version, we give an $O(m^3 + n \log n)$ time approach. For the axis-aligned 3D version, we have an algorithm that runs in $O(m^2(m+n))$ time.

1 Introduction

Consider two sets of points, R and B in 2D or 3D. R contains n points calleed *red points* and B sontains m points called *blue points*. The problem we study in this paper, called the *Maximum Area Rectangle Separating Red and Blue Points* problem, is to find a (hyper-)rectangle that contains all the red points, the minimum number of blue points, and has the largest area. We call such a (hyper-)rectangle a *maximum separating (hyper-)rectangle*. Examples of maximum separating rectangles are illustrated in Figures 1, for the planar axis-aligned case, as well as the planar non axis-aligned case.

Applications that require separation of bi-color points could benefit from efficient results to this problem. For instance, consider we are given a tissue containing a tumor, where the tumor cells are specified by their coordinates. The coordinates of healthy cells are also given. The goal is to separate the tumor cells from the healthy cells, for surgical removal or radiation treatment. Another application would be in city planning, where blue points represent buildings and red points represent monuments and the goal is to build a park that contains the monuments and as few buildings as possible.

[*]This research was partially supported by NSF award IIP1439718 and CPRIT award RP150164

[†]Department of Computer Science, University of Texas at Dallas, bxa120530@utdallas.edu

[‡]Department of Computer Science, University of Texas at Dallas, bxa120530@utdallas.edu

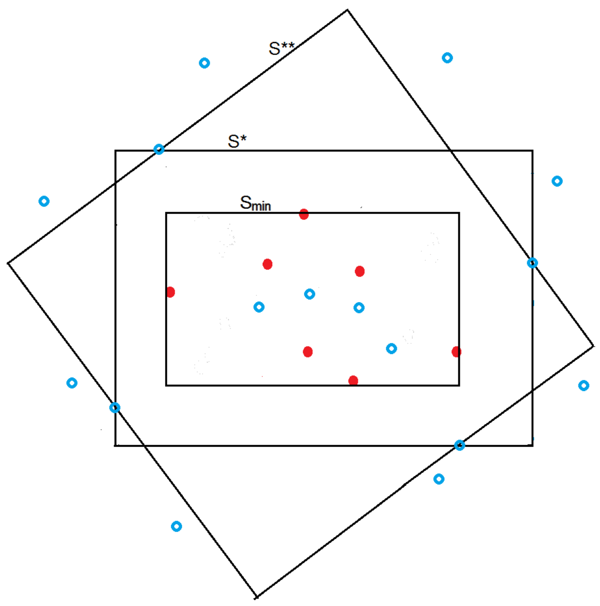


Figure 1: Red points in R are shown in solid circles and blue points in B are shown in empty circles. The minimum enclosing rectangle S_{min} of all red points, the maximum axis-aligned separating rectangle S^*, and the maximum non axis-aligned separating rectangle S^{**}, are shown.

1.1 Related Work

Mukhopadhyay et. al [8] studied the problem of finding the maximum empty rectangle among a point set of n planar points bounded by a given axis-aligned rectangle. For the axis-aligned case, they present an algorithm that runs in $O(n)$ time, assuming points are pre-sorted by one of the coordinates. For finding the arbitrary oriented optimal rectangle, they give an $O(n^3)$-time algorithm. Chaudhuri et. al [4] also study the problem of finding the maximum empty rectangle of arbitrary orientation. They give an algorithm that requires $O(n^3)$ time and $O(n^2)$ worst-case space. Their results seem to be the most relevant to our problem.

More recetntly, Dumitrescu and Jiang [5] considered the problem of computing the maximum-volume axis-aligned d-dimensional box that is empty with respect to a given point set P in d dimensions. The target box is restricted to be contained within a given axis-aligned box R. For this problem, they give the first known FPTAS, which computes a box of a volume at least $(1-\epsilon)OPT$, for an arbitrary ϵ, where OPT is the volume of the optimal box. Their algorithm runs in $O((22d\epsilon^{-2})^d \cdot n\log^d n)$ time.

In [11], Toussaint studies the Rotating Callipers problem, in which the goal is to find the smallest area enclosing rectangle of a given convex polygon. They give an algorithm that takes $O(n\log n)$ time, or $O(n)$ if the vertices are pre-sorted.

Kaplan and Sharir study the problem of finding the maximal empty axis-aligned rectangle containing a query point [9]. They design an algorithm to answer queries in $O(\log^4 n)$ time, with $O(n\alpha(n)\log^4 n)$ time and $O(n\alpha(n)\log^3 n)$ space for preprocessing. Here $\alpha(n)$ is the inverse of the Ackermann's function. In a different paper, they also solve the disk version of the problem (finding the maximal empty disk containing query point) [10]. Their approach takes $O(\log^2 n)$ query and $O(n\log^2 n)$ preprocessing time, using a $O(n\log n)$ space data structure.

Separability of two point sets using various separators is a well known problem. Megiddo et. al [7] study the hyperplane separability of point sets in $\mathbb{R}^d$ in $d \geq 2$ dimensions. They show how to decide one-hyperplane separability using linear programming in polynomial time. They also prove that the problem of separating two point sets by k lines is NP-complete. Aronov et. al [2] consider four metrics to evaluate misclassification of points of two sets by a linear separator. One of them is the number of misclassified points (which is somewhat related to our problem). For this error metric, they present an $O(n^d)$-time algoritm. Separating point sets with circles has also been considered. The problem of finding the minimum area separating circle among red and blue points was studied by O'Rourke et. al [6]. They solve the decision problem in linear time and give an $O(n\log n)$ time algorithm for computing the minimum separating circle, if it exists. When points sets cannot be separated by a circle, Bitner and Daescu et al [3] give two algorithms that run in $O(mn\log m + n\log n)$ (respectively, $O(mn + m^{1.5}\log^{O(1)} m)$) time. Armaselu and Daescu studied the dynamic version of the problem [1].

1.2 Our Results

In Section 2, we study the axis-aligned version of the problem and give an $O(m\log m + n)$ time algorithm. The running time reduces to $O(m+n)$ if the points are pre-sorted by one of the coordinates. Then, in Section 3, we address the non-axis aligned version and show how to solve it in $O(m^3 + n\log n)$ time. To do that, we use a polar sweep-like approach and exploit a few properties of the problem. Our algorithm uses an enumeration approach that identifies a subset of all maximal rectangles containing the convex hull of all red points, so it will find all optimal solutions. In Section 4, we study the 3D axis-aligned case and we give an algorithm that is based on computing a set of staircases (in the horizontal plane) for each pair of blue points outside the minimum enclosing box of all red points. This algorithm runs in $O(m^2(m+n))$ time.

2 Algorithm for the axis-aligned version

We begin by describing some properties of the bounded optimal solution.

Observation 1 *The maximum axis-aligned separating rectangle S must contain at least one blue point on each of its sides.*

Consider a quad Q of 4 non-colinear blue points q_1, q_2, q_3, q_4 in this order. Two vertical lines going through two of these points and two horizontal lines going through the other two of those points define a rectangle S. We say that Q *defines* S.

Definition 2.1. We say that points in Q satisfy the *extremum condition*, if they are in convex position and no point is extremal according to both coordinates (e.g. none is both rightmost and lowest).

The use of the notion of extremum condition will be revealed later on.

Definition 2.2. Consider a vertical strip formed by the two vertical lines bounding R to the left and right, as well as a horizontal strip formed by the parallel lines bounding R above and below. The *minimum blue rectangle S_{min}* is the intersection between the vertical and horizontal strips (refer to Figure 2).

We start with the minimum blue rectangle S_{min} enclosing all red points. For each side of S_{min}, we slide it outwards parallel to itself until it hits a blue point (if no such point exists, then the solution is unbounded). Denote by S_{max} the resuting rectangle (shown in Figure 1). Unbounded solutions can be easily determined in linear time, so from now on we assume bounded solutions. If the interior of $S_{max} \setminus S_{min}$ does not contain blue points, S_{max} is the optimal solution, and we are done. We discard the blue points contained in S_{min}, as well as the blue points outside of S_{max}, from B. The set of remaining blue points is partitioned into 4 disjoint subsets $B_1, \ldots, B_4$, each containing points that are located in a rectangle formed by right upper (left upper, left lower, right lower) corners of S_{min} and S_{max} (see Figure 2 for details).

Consider the points of B_1 and sort them by X coordinate. A point $p_0 = (x_0, y_0) \in B_1$ is a candidate to be a part of the optimal solution only if there are no points $p = (x, y) \in B_1$ with $x < x_0$ and $y < y_0$. We only leave such possible candidate blue points in B_1 and discard the rest. The first point in B_1 is such a possible candidate. The elements of B_1 form a sequence

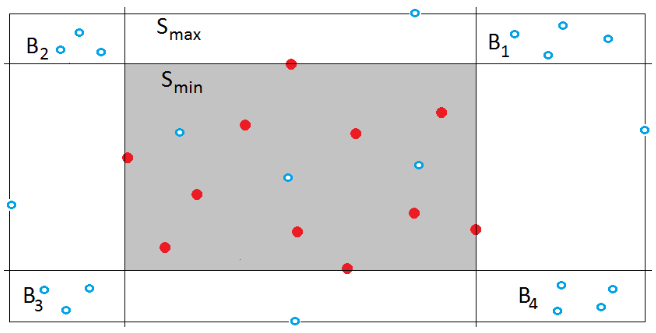


Figure 2: The minimum blue rectangle S_{min}, the rectangle S_{max} which bounds the solution space, and the subsets $B_1, \ldots, B_4$.

that is ordered non-decreasingly by X coordinate and non-increasingly by Y coordinate, as shown in Figure 3. Note that these points form a staircase. The sets B_2, B_3, B_4 are treated appropriately in a similar way. The 4 staircases can be found in $O(m \log m)$ time [8].

While the staircase construction approach has been used before [8], there are differences between how we use it in this paper and how it was used in [8]. Specifically, note that a maximum B-empty rectangle computed as in [8] may not contain all red points.

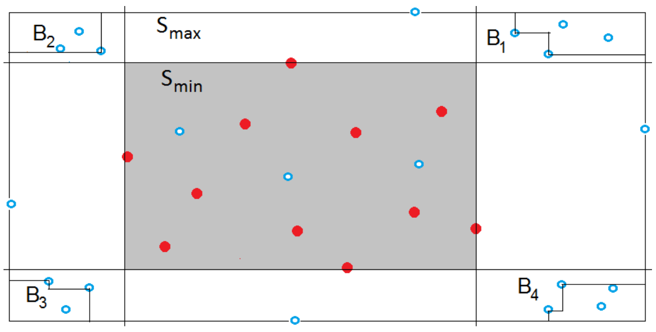


Figure 3: The set of possible candidate points defining a solution. In each $B_i, i = 1, 2, 3, 4$, they are ordered by X, then by Y and form a staircase

We now consider tuples of four points from $B_i, i = 1, 2, 3, 4$. Note that, if points of a tuple Q do not satisfy the extremum condition, Q cannot define an optimal solution. Otherwise, Q defines a rectangle S constructed according to the following rule: horizontal lines pass through points with extremal Y coordinates, vertical lines pass through points with extremal X coordinates. If a rectangle contains S_{min}, does not contain any blue points other than those in S_{min}, and cannot be expanded in any direction, then it is a candidate to be an optimal solution. Consider the maximum Y coordinate point $p_y \in Q$, which belongs to $B_1 \cup B_2$. We present the solution for the case when $p_y \in B_1$ (symmetric arguments holds for $p_y \in B_2$). Let $x(p)$ (respectively, $y(p)$) denote the X (Y) coordinate of point p. Let $X(p - -q)$ $(Y(p - -q))$ denote the set of blue points with X (Y) coordinates larger than that of p and smaller than that of q.

Let p^i_k denote the k-th point of the staircase of $B_i, i = 1, 2, 3, 4$.

Observation 2 *Given a point $p^1_k \in B_1$ and a point $p \in B_1 \cup B_4$, such that $x(p^1_k) < x(p), y(p^1_k) > y(p)$, and $x(p) \leq x(p^1_{k+1})$, there exists a unique rectangle S that has p^1_k on its top side, p on its right side, and two other points of B on its other sides, that is a candidate to be an optimal solution.*

This rectangle can be constructed by drawing a horizontal line l_1 through p^1_k, a vertical line l_2 through p, a horizontal line l_3 through point $p^4_l \in B_4$ that is closest to the line l_2 (and to the left of it) and a vertical line passing through the point $p^2_j \in B_2 \cup B_3$ located between l_1 and l_3 and closest to l_2. See Figure 4 for details.

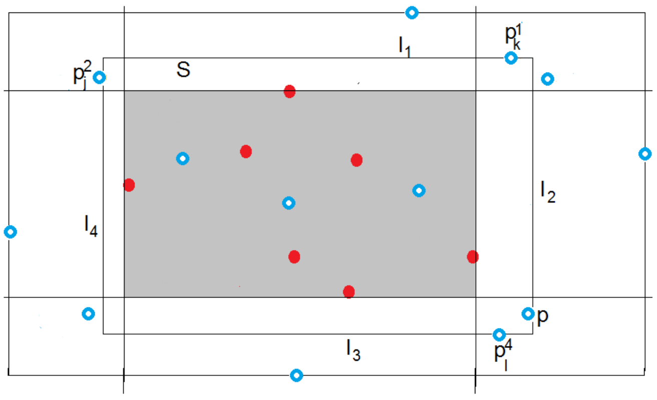


Figure 4: The unique candidate rectangle S defined by points $p^1_k \in B_1, p \in B_1 \cup B_4$ with $x(p^1_k) < x(p), y(p^1_k) > y(p)$

Lemma 1 *Out of all tuples that contain a point $p^1_k \in B_1$ as the highest point, there is at least one and at most $O(t_k)$ tuples determining rectangles that can be an optimal solution, where $t_k = |X(p^1_k - -p^1_{k+1})| + 1$.*

Proof. For a tuple Q with points satisfying the extremum condition, let p_r be the rightmost point. If $x(p_r) > x(p^1_{k+1})$, the rectangle that Q defines contains p^1_{k+1} and thus will have at least one more blue point than S_{min}. The right side of the rectangle containing p^1_k on its top side can either pass through $p^1_{k+1} \in B_1$ or one of the points of B_4 in $X(p^1_k - -p^1_{k+1})$. For each of these points we construct a rectangle as described in Observation 2, then select the rectangle with the maximum area from the set of constructed rectangles and add its defining points to Q (all rectangles in $X(p^1_k - -p^1_{k+1})$ can have the maximum area, hence the $O(t_k)$ bound). $\square$

Lemma 2 *For all points in $B_1 \cup B_2$ (resp., $B_3 \cup B_4$), there are at most $O(m)$ tuples defining candidate rectangles.*

Proof. We ignore the superscript in p_k^1 and let t_k be defined as in Lemma 1, for any index k. The number of tuples is at most $N = \sum_{p_k \in B_1 \cup B_2} t_k$. Since for any k, l, the sets $X(p_k - -p_{k+1}), X(p_l - -p_{l+1})$ are disjoint, $N = |X(p_1 - -p_2) \cup \cdots \cup X(p_{m_1 - 1} - -p_{m_1})| + m_1 = |X(p_1 - -p_{m_1})| + m_1 = O(m)$ (where $m_1 = |B_1 \cup B_2|$). □

In order to find the optimal solution, we iteratively take each point of $B_1 \cup B_2$ and construct the candidate rectangle for it, according to Lemma 1. After that, the rectangle with the maximum area is returned as the solution. Finding S_{min} takes $O(n)$ time, finding S_{max} takes $O(m)$ time, and computing the staircases of B takes $O(m \log m)$ time, as the points need to be ordered by X (Y) coordinate (if blue points are initially available in sorted order by one of the coordinates, this becomes $O(m)$). After that, $O(m)$ extra time is required to find S^*, the maximium separating rectangle. At each iterative step we take advantage of coordinate monotonicity of points in B_i and keep a pointer to a previously accessed point, thus avoiding spending logarithmic time in one iteration. Thus, we have proved the following result.

Theorem 3 *The axis-aligned version of the maximum-area separating rectangle problem can be solved in $O(m \log m + n)$ time. The running time reduces to $O(m + n)$ if the blue points are presorted.*

3 Algorithm for the non-axis-aligned version

Without loss of generality, we assume that there is no blue point within $CH(R)$, the convex hull of all red points. For a given angle ϕ, let L_ϕ denote the two lines of slope ϕ tangent to $CH(R)$ on opposite sides. L_ϕ defines a strip containing $CH(R)$. Similarly, let $L_\phi^\perp$ denote the two lines orthogonal to the lines in L_ϕ and tangent to $CH(R)$ on opposite sides. $L_\phi^\perp$ also defines a strip containing $CH(R)$. Note that the intersection of the lines in L_ϕ and $L_\phi^\perp$ define $S_{min,\phi}$, the smallest rectangle containing $CH(R)$ at orientation ϕ. Let $(L, L^\perp, \phi)$ denote the set of lines in L_ϕ and $L_\phi^\perp$. Throughout the paper, whenever understood, we are going to remove ϕ from our notation. That is, we can use $(L, L^\perp)$ without specifying the angle ϕ.

3.1 Optimal rotation ranges

$L, L^\perp$, and $CH(R)$ divide the plane into 12 regions. Four of these regions are inside $S_{min,\phi}$ and denoted as $C_{NW}, C_{SW}, C_{SE}, C_{NE}$. Another four of them are within the union of the two strips defined by $(L, L^\perp)$, denoted by B_N, B_W, B_S, B_E. Finally, the remaining four are defined by $(L, L^\perp)$ and the corners of $S_{min,\phi}$ and denoted by $B_{NW}, B_{SW}, B_{SE}, B_{NE}$. Figure 5 illustrates this. For a given angle ϕ, the 8 regions outside $S_{min,\phi}$ will serve the same purpose as they did in Section 2.

Our general approach is as follows. Starting from a given angle ϕ (initially $\phi = 0$), we rotate $(L, L^\perp)$ until one of the lines hits a blue point p at some angle ϕ'. When we rotate past angle ϕ', p defines an enter event for one of the 12 regions and an exit event for an adjacent region. Angle ϕ' is called a *region event* and point p is called a *region event point*. See figure 6 for an illustration. Notice that, in the open rotation interval (ϕ, ϕ'), there is no change in the number of blue points in any of the 12 regions mentioned earlier. We keep rotating until $\phi \geq \pi/2$.

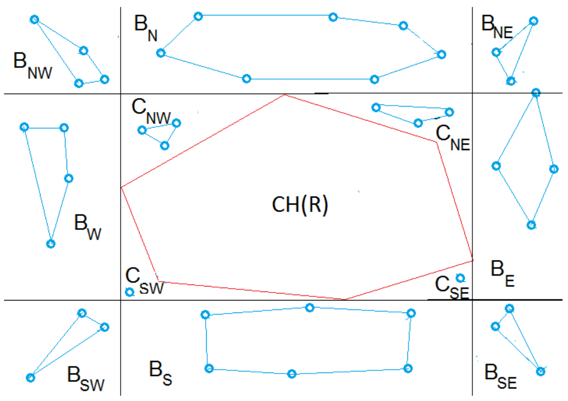


Figure 5: The 12 regions and their convex hull

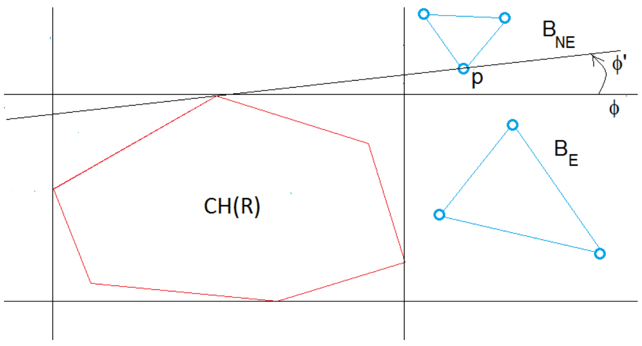


Figure 6: Blue point p enters B_E and exit B_{NE}, defining a region event ϕ'

Each blue point defines two region event points and the angle ϕ for each of these event points can be found in $O(\log n)$ time by means of a binary search on $CH(R)$. We add these $2m$ angles together with the n slopes of the edges on $CH(R)$ to the unit circle of directions. This computation takes $O(m \log n + n)$ time.

Let $(l, l^\perp)$ denote the directions of $(L, L^\perp)$ on the circle of directions. We are going to rotate $(l, l^\perp)$ between consecutive directions defined by region event points.

Definition 3.1. We call *elementary rotation range*, a range of angles $[\phi, \phi']$ defined by two consecutive region event points.

Definition 3.2. We call *optimal rotation range*, an elementary rotation range $[\phi, \phi']$ in which the number of blue points inside $S_{min,\alpha}$ is minimal, $\forall\alpha : \phi \leq \alpha \leq \phi'$.

From now on, we focus only on optimal rotation ranges. These ranges can be found in $O(n+m)$ time by counting the points inside $S_{min,\phi}$ at each region event point.

For each angle ϕ of rotation, there is a maximum area separating rectangle S_ϕ with polar angle ϕ.

Observation 3 *Within any elementary rotation range $[\phi, \phi']$, the number of blue points inside S_{min} is constant. Hence, the number of blue points inside S_α, say N_α, is constant, $\forall\alpha : \phi \leq \alpha \leq \phi'$.*

Observation 4 *The range $[0, \pi/2]$ of rotation of $(L, L^\perp)$ sweeps all blue points that can be swept by a full $[0, 2\pi]$ rotation range.*

3.2 Finding the optimal rectangle within an elementary rotation range

We now give some insight into the structure of the problem within an elementary rotation range $[\phi, \phi']$. Note that the points defining S_α might change as α changes, $\phi < \alpha < \phi'$.

For any angle $\alpha \in [\phi, \phi']$, consider the blue points closest to each edge of $S_{min,\alpha}$ in quadrants q_N, q_W, q_S, q_E, called *bounding blue points*. The *bounding sides* are line segments parallel to $S_{min,\alpha}$ that go through the bounding blue points. Also consider the staircases $S_{NW}, S_{SW}, S_{SE}, S_{NE}$ for the corresponding 4 quadrants. During rotation within an optimal range $[\phi, \phi']$, the staircases may gain or lose blue points. In addition, the bounding blue points q_N, q_W, q_S, q_E may change.

Definition 3.5. An angle $\alpha \in [\phi, \phi']$ is called a *blue edge event*, if the closest blue point p to some edge of $S_{min,\alpha}$ changes to q at angle α. Point q is called *blue edge event point*. See Figure 7 for an illustration.

Note that region events can also be blue edge events, as a blue point moving from one region to another may become a bounding blue point.

Definition 3.6. An angle $\alpha \in [\phi, \phi']$ is called an *insertion event*, if some blue point q (called *insertion event point*) is included in some staircase at angle α. See Figure 8 for an illustration.

Definition 3.7. An angle $\alpha \in [\phi, \phi']$ is called an *deletion event*, if some blue point q (called *deletion event point*) is excluded from some staircase at angle α. See Figure 9 for an illustration.

At insertion and deletion events, the staircase needs to be updated.

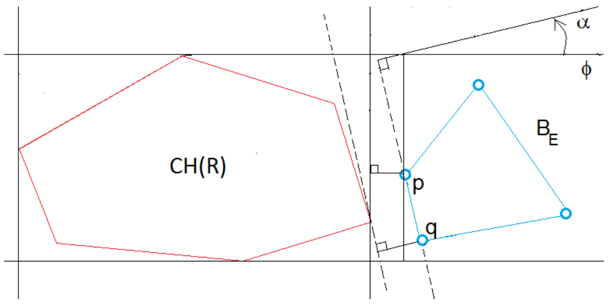


Figure 7: Closest blue point p to some edge of $S_{min,\alpha}$ changes to q at angle α, which is thus a blue edge event

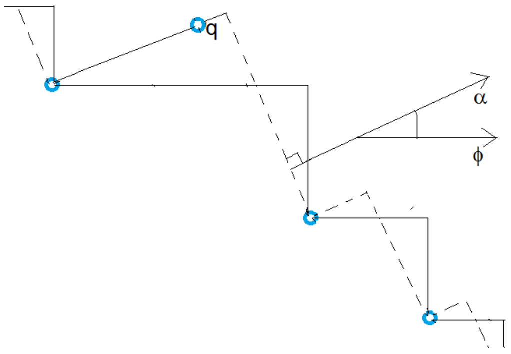


Figure 8: Blue point q appears on the staircase at rotation angle α, which is thus an insertion event

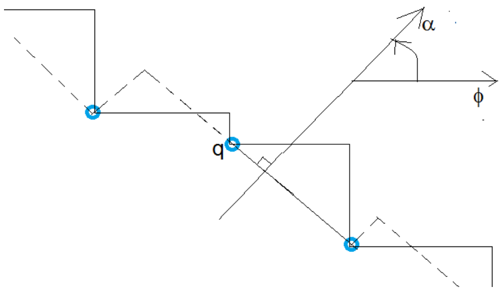


Figure 9: Blue point q disappears from the staircase at rotation angle α, which is thus a deletion event

Let the current strip rotation be $\alpha \in [\phi, \phi']$. To compute blue edge events, we take the smallest angle α' made by each side of $S_{min,\alpha}$ with the closest edge (in counter-clockwise order) of its corresponding blue hull, one of $CH(B_E), CH(B_N), CH(B_W), CH(B_S)$. We keep track of blue edge events as we sweep the unit circle of directions with the angle α. We then set $\alpha = \alpha + \alpha'$ and repeat the process.

Lemma 4 *All blue edge events in* $[0, \pi/2]$ *that are not region events can be treated in* $O(m \log m)$ *time.*

For each blue point, during rotation, we store the index of the region it belongs to. At region events, two changes may occur to a staircase.

1) Points from one of B_N, B_W, B_S, B_E may appear on an adjacent staircase.

2) A staircase may lose points, which enter an adjacent region, one of B_N, B_W, B_S, B_E. One of them may become bounding blue point for the region it enters.

We "reduce" the problem of finding the maximum separating rectangle within an elementary rotation range to the problem of computing the maximum empty rectangles (MER's) among the blue point set. In a nutshell, for all relevant blue points p_i, we do the following.

1. Compute the necessary staircases of p_i within angle ranges of interest (see Figure 10 for an example)

2. Find the MER's bounded by p_i and its staircases, by "intersecting" the staircase approach in [8] with our algorithm within optimal ranges, and selecting only those containing $CH(R)$ (see Figure 10).

All the relevant blue points are in $S_{NE} \cup S_{NW} \cup S_{SW} \cup S_{SE} \cup \{q_N, q_W, q_S, q_E\}$ (blue points that may appear on these staircases at insertion events are also considered). Note that, by performing these two steps for any blue point p_i inside $S_{min,\phi}$, throughout the whole elementary rotation range, the resulting rectangles will intersect $CH(R)$. Also, for any p_i not on $S_{NE} \cup S_{NW} \cup S_{SW} \cup S_{SE}$, the resulting rectangle will contain extra blue points. Thus, we disregard them.

All staircases of p_i are computed and maintained using the approach in [8].

Due to space limitation, we leave further details to the full version of the paper.

We have come to the following result.

Theorem 5 *The non axis-aligned maximum separating retcangle of a set of red points* R *and a set of blue points* B *can be found in* $O(m^3 + n \log n)$ *time.*

Proof. We first compute $CH(R)$ in $O(n \log n)$ time. After that, we spend $O(m \log n + n)$ time to find all optimal rotation ranges, as shown earlier. Computing the overall staircases takes $O(m \log m)$ time per rotation range, or $O(m^2 \log m)$ in total. There are $O(m^2)$ insertion and deletion events, and each can be handled

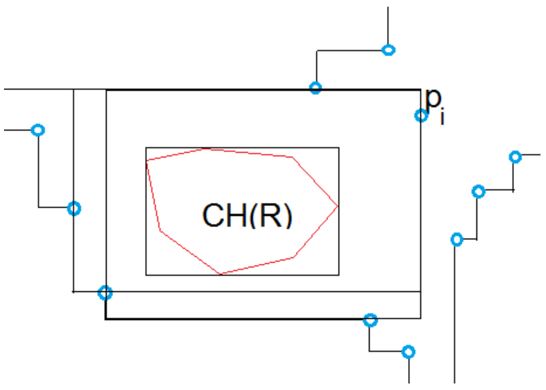


Figure 10: Staircases of p_i are computed, then the rectangles bounded by the staricases and p_i and containing $CH(R)$ are computed. For this location of p_i, there is no need to coompute the NE staircase

in $O(m)$ time. Computing and updating the staircase for each relevant p_i in each rotation range takes $O(m)$ time. That is $O(m^3)$ for all rotation ranges. As noted in [8], the time needed to compute the largest B-empty rectangle is $O(m^3 + r)$, where r is the number of B-empty rectangle ranges. They also prove $r = O(m^3)$ in total. In our case, we select those rectangle ranges included in an optimal rotation range and trimmed to the respective quadrants, in which the maximal rectangle contains $CH(R)$. Thus, the result follows. □

Note that our algorithm will find all optimal solutions. It would be nice to argue a sub-cubic running time.

However, we can show that the number of maximal B-empty rectangles within an elementary rotation range can be $\Theta(m^3)$, using a construction similar to the one in [4].

4 Algorithms for the 3D axis-aligned version

Given a set of blue points B and a set of red points R in the 3D space, the goal is to find the axis aligned box of maximum volume enclosing R and having the fewest blue points.

We first make some important observations, then we describe our algorithm.

4.1 Preliminaries

We extend the definitions from Sections 2 and 3 to the 3D case. Specifically, a *separating box* is a box enclosing all red points and the fewest blue points. A *maximum separating box* is a separating box of maximum volume. The *minimum blue box* is the axis aligned box enclosing R having minimum volume.

First, compute the minimum blue box S_{min}, and discard the blue points inside S_{min}, as they cannot be avoided. Then, consider a 3D-strip defined on S_{min}, consisting of an X axis, a Y axis, and a Z axis strip. Similarly, to the 2D case, we extend the minimum blue box in each of the 6 directions until each side passes through a blue point. If this is not possible (the box can be extended in some direction without ever hitting a blue point), then the problem is unbounded and the optimal box has infinite volume. Hence, from now on, we assume that the problem is bounded. Denote the resulting box by S_{max}. It is easy to see that the volume of the maximum separating box is no larger than $volume(S_{max})$. We therefore discard blue points outside S_{max}.

A *candidate box* is a box S with $S_{min} \subseteq S \subseteq S_{max}$ that has a blue point on each face.

We subdivide B into 8 subsets $B^+_{NE}, B^+_{NW}, B^+_{SW}, B^+_{SE}, B^-_{NE}, B^-_{NW}, B^-_{SW}, B^-_{SE}$. Here B^+_{NE} is the subset of points within the box bounded by the top NE corners of S_{min}, S_{max} (the other subsets are defined in a similar manner). Note that no blue point in any of the 8 subsets of B lies within the 3D-strip.

Definition 4.1. A point $p \in B^+_{NW}$ *directly dominates* another point $q \in B^+_{NW}$, if $x_p \geq x_q, y_p \leq y_q, z_p \leq z_q$, and there is no other point $r \in B^+_{NW}$ such that $x_p \geq x_r \geq x_q, y_p \leq y_r \leq y_q, z_p \leq z_r \leq z_q$.

The definition can be extended as follows.

Definition 4.2. A point $p \in B^+_{NW}$ *directly dominates* a point $q \in B^+_{NE}$, if $x_p \leq x_q, y_p \leq y_q, z_p \leq z_q$, and there is no other point $r \in B$ such that $x_p \leq x_r \leq x_q, y_p \leq y_r \leq y_q, z_p \leq z_r \leq z_q$. The definition can be extended to points from the other pairs of subsets of B, changing x, y, z coordinate orders accordingly.

Figure 11 illustrates the concept of direct domination.

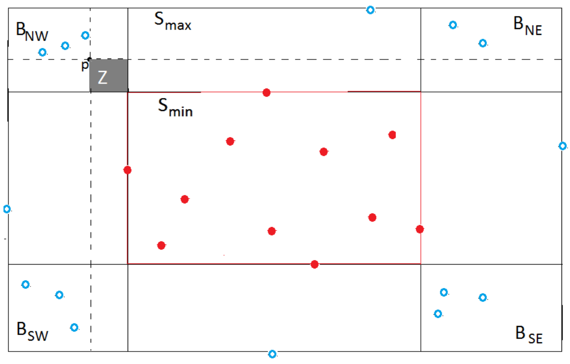


Figure 11: Points from $B_{NW}, B_{SW}, B_{SE}, B_{NE}$ directly dominated by point $p \in B^+_{NW}$ are shown. If projection of some point $q \in B^+_{NW}$ below p is in the gray area Z, then p can be ignored

For $p \in B$, denote by $D(p)$ the set of points in B directly dominated by p and project $D(p)$ on the xOy plane. Also denote by $X(p)$ the rectilinear polygon formed by connecting the points in the projected $D(p)$ as follows. First, the points $q \in D(p)$ are sorted by the angle between the line through the center of S_{min} and q and the X axis. Note that, for instance, in B^+_{NW} the points of $D(p)$ are ordered non-increasingly by X coordinate and non-decreasingly by Y coordinate (in the other subsets of $B^+, D(p)$ is ordered accordingly). Then, each two adjacent points $q, r \in D(p)$ are connected by two line segments, based on the subset they belong to. Specifically, if $q, r \in B^+_{NW}$ and q is before r in $D(p)$, then connect q to a horizontal segment to the west of q and r to a vertical segment to the north of r such that these segments meet. The first point of $D(p)$ within B^+_{NW} is connected to the north edge of the projected S_{max} and the last point is connected to the west edge. The cases where r, q are in the other subsets of B^+ are treated similarly. Refer to Figure 12 for an illustration.

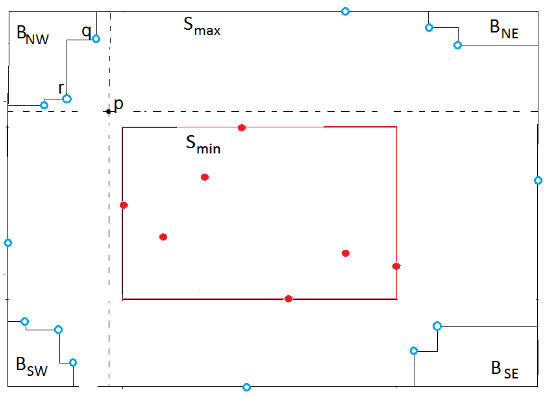


Figure 12: $X(p)$

For any point $p \in B^+(B^-)$ and for any candidate box supported by p on the top (bottom) face, the 4 corners of the top (bottom) face must be inside $X(p)$ (otherwise the box would contain some points of $D(p)$). Let Z be the rectangle having the projection of p as one corner and the closest corner of the projection of S_{min} as the other corner (see Figure 11). If the projection of some point $q \in B^+(B^-)$ below (above) p is inside Z, then p cannot be a bounding point for a candidate box, as otherwise it would be forced to contain q.

4.2 Solution

For any point $p \in B^+_{NW}$, we consider a set $T(p)$ of 4 rectangles $T_i(p), i = 1, 2, 3, 4$, defined as follows. Consider the projection of p on the xOy plane, as in the previous section. Abusing notation, refer to the projection of a point q as simply q, and the projection of a box S as the rectangle S. Consider the two lines l^1_p, l^2_p through p (l^p_1

is X-aligned and l_2^p is Y-aligned). Then each rectangle of $T(p)$ is bounded by two of the lines that form the boundary of S_{min}. In addition, $T_1(p)$ is bounded by l_1^p and the left side of S_{max}, $T_2(p)$ is bounded by l_2^p and the south side of S_{max}, $T_3(p)$ is bounded by l_1^p and the right side of S_{max} and $T_4(p)$ is bounded by l_2^p and the right side of S_{max}. Figure 13 shows how these rectangles are defined. If there exists a blue point q below p within $T_i(p)$ for some $i = 1, 2, 3, 4$, then $X(p)$ is retracted so that the edge flush with the side of S_{max} closest to q is replaced by the edge through q (updating adjacent edges accordingly and adding q to $X(p)$, as shown in Figure 13. The argument also holds for $p \in B_j^+, p \in B_j^-$ for any other $j \in \{NW, NE, SW, SE\}$.

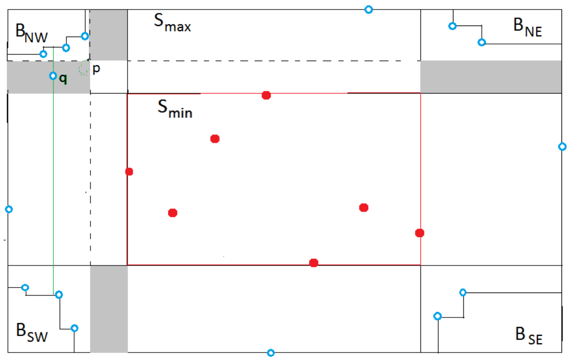


Figure 13: Rectangles of $T(p)$ are shaded in gray. If projection of some point q is within some of these rectangles, it redefines $X(p)$

For each pair of points $p \in B^+, q \in B^-$, we compute the largest box bounded above by p and below by q. We check whether there exists a point $r \in B^+$ below p such that the projection of r is within the Z region defined by p, or $r \in B^-$ above q such that the projection of r is within the Z region defined by q. If there exists such r, then we disregard the pair p, q. We construct the chains $X(p), X(q)$ as described above. Let C be the staircase chain of projections of points in $X(p) \cup X(q)$ on xOy that are directly dominated by the projection of p. We then resort to solving the 2D axis-aligned version of the problem with C as blue point set in $O(m + n)$ time (assuming B is pre-sorted by x, y, z). We can prove the following result.

Theorem 6 *The maximum axis-aligned separating box of a set of red points R and a set of blue points B can be found in $O(m^2(m + n))$ time.*

Proof. We first spend $O(m + n)$ time to compute S_{min}, S_{max}. There are $O(m^2)$ pairs of points $p \in B^+, q \in B^-$. After sorting B by x, then by y, then by z coordinates in $O(m \log m)$ time, each pair (p, q) is treated in $O(m+n)$ time. Hence, the result follows. $\square$

5 Conclusion and Remarks

We addressed the problem of finding the maximum area separating rectangle of red and blue points. We considered three cases: the axis-aligned 2D case and 3D case, as well as the arbitrary orientation planar case. For the axis-aligned planar case, we presented a near-linear time algorithm. We also presented an $O(m^3 + n \log n)$ time algorithm for the arbitrary oriented planar case. Finally, we gave an $O(m^2(m + n))$ time algorithm for the 3D case. We leave open improving the time for arbitrary orientation for the 2D case.

Acknowledgement

The authors would like to thank Dr. Anastasia Kurdia for her preliminary work on the 2D and 3D axis-aligned cases.

References

[1] B. Armaselu and O. Daescu, Dynamic minimum bichromatic separating circle, COCOA 2015: 688-697

[2] B. Aronov and D. Garijo, Y. Núñez, D. Rappaport. C. Seara and J. Urrutia, Measuring the error of linear separators on linearly inseparable data, Discrete Applied Mathematics 160(10-11): 1441-1452 (2012)

[3] S. Bitner, Y. K. Cheung, and O. Daescu, Minimum separating circle for bichromatic points in the plane, ISVD '2010

[4] J. Chaudhuri, S. C. Nandy and S. Das, Largest empty rectangle among a point set, J. Algorithms, 46, Vol. 1, pp. 54-78, 2003

[5] A. Dumitrescu and M. Jiang, On the largest empty axis-parallel box amidst n points, Algorithmica 66(2): 225-248 (2013)

[6] S. Kosaraju, J. O'Rourke and N. Megiddo, Computing circular separability, Journal of Discrete Computational Geometry, 1:105113, 1986

[7] N. Megiddo, On the complexity of Polyhedral Separability, Journal of Diescrete Computational Geometry, 1988, vol. 3, pp. 325-337

[8] A. Mukhopadhyay, S.V. Rao, Computing a Largest Empty Arbitrary Oriented Rectangle. Theory and Implementation, International Journal of Computational Geometry & Applications Vol. 13, No. 3 (2003) 257-271

[9] H. Kaplan, M. Sharir, Finding the Maximal Empty Rectangle Containing a Query Point, CoRR abs/1106.3628 (2011)

[10] H. Kaplan, M. Sharir, Finding the Maximal Empty Disk Containing a Query Point, Symposium on Computational Geometry 2012: 287-292

[11] G. Toussaint, Solving Geometric Problems with the Rotating Calipers, IEEE MELECON83

Characterizing minimum-length coordinated motions for two discs

David Kirkpatrick[*] Paul Liu[†]

Abstract

We study the problem of planning coordinated motions for two disc robots in an otherwise obstacle-free plane. We give a characterization of collision-avoiding motions that minimize the total trace length of the disc centres, for all initial and final configurations of the robots. The individual traces are composed of at most six (straight or circular-arc) segments, and their total length can be expressed as a simple integral with a closed form solution depending only on the initial and final configuration of the robots.

1 Introduction

We consider the problem of planning collision-free motions for two disc robots of arbitrary radius in an otherwise obstacle-free environment. Given two discs $\mathbb{A}$ and $\mathbb{B}$ in the plane, with specified initial and final configurations, we seek a shortest collision-free motion taking $\mathbb{A}$ and $\mathbb{B}$ from their initial to their final configurations. The length of such a motion is defined to be the length sum of paths traced by the centres of $\mathbb{A}$ and $\mathbb{B}$.

The consideration of disc robots in motion planning has amassed a substantial body of research, the bulk of which is focused on the feasibility, rather than optimality, of motions. Schwartz and Sharir [5] were the first to study motion planning for k discs among n polygonal obstacles. For $k = 2$, they developed an $\mathcal{O}(n^3)$ algorithm (later improved to $\mathcal{O}(n^2)$ [6, 9]) to determine if a collision-free motion connecting two specified configurations is feasible. When the number of robots k is unbounded, Spirakis and Yap [7] showed that even determining feasibility is NP-hard for disc robots, although the proof relies on the robots having different radii.

When considering the optimality of a coordinated motion, one could adopt several different cost measures. Algorithms that optimize motions with respect to total distance traced by the centers of the two robots, the measure adopted in this paper, are typically numerical or iterative in nature, with no precise performance bounds [8]. If the robots are velocity constrained, another natural measure would be realization time [1]. Note that the coordination required to minimize distance is not the same as that required to minimize time.

This paper makes several novel contributions to the understanding of minimum-length coordinated motions. We first characterize all configurations that admit straight-line optimal motions. For all other configurations, the motion from initial to final configuration involves either a net clockwise or counter-clockwise turn in the relative position of the discs. In this case, our results describe either (i) a single optimal motion, or (ii) two feasible motions, of which one is optimal among all net clockwise motions and the other is optimal among all net counter-clockwise motions. For (ii), one of the two motions will be globally optimal. The motions we describe are composed of a constant number of straight segments and circular arcs of radius s, the sum of the disc radii. The total path length can be expressed as a simple integral depending only on the initial and final positions of the discs.

Our general approach is based on the Cauchy surface area formula, which was first applied to motion planning by Icking et al. [3] to establish the optimality of motions of a directed line segment in the plane, where distance is measured by the length sum of the paths traced by the two endpoints of the segment. (The problem of optimizing the motion of a line segment, even in the absence of obstacles, has a rich history; see [3], and references therein.) Of course, the problem of moving a directed line segment of length s corresponds exactly to the coordinated motion of two discs with radius sum s constrained to remain in contact throughout the motion. Hence the coordinated motion of two discs with radius sum s is equivalent to the problem of moving an "extensible" line segment that can extend freely but has minimum length s. As such, our results also generalize those of Icking et al. [3]. Although we use some of the same tools introduced by Icking et al., our generalization is non-trivial; the path-embedding argument that lies at the heart of the proof of Icking et al. depends in an essential way on the assumption that the rod length is fixed throughout the motion.

The rest of the paper is organized as follows. In Section 2 we outline some basic definitions. Section 3 summarizes the general structure of our proofs, and the main proof and algorithm is presented in Section 4. It is the (unavoidable) nature of our results that they involve an extensive case analysis, far more than can be dealt with comprehensively in eight pages. Our objec-

[*]Department of Computer Science, UBC, kirk@cs.ubc.ca
[†]Department of Computer Science, UBC, paul.liu.ubc@gmail.com

tive is to provide sufficient detail on a small number of typical cases to convey the general nature of our constructions, together with indications of where extraordinary treatment is required. A longer version of this paper (see [4]) provides more comprehensive proofs; full details will appear in the forthcoming MSc thesis of the second author.

2 Background

We want to describe collision-free coordinated motions, for a pair of disc robots, between their initial and final configurations. We specify the (instantaneous) **position** of a disc by the location (in $\Re^2$) of its centre. A **placement** of a disc pair $(\mathbb{A}, \mathbb{B})$ is a pair (A, B), where A (resp. B) denotes the position of $\mathbb{A}$ (resp. $\mathbb{B}$). A placement (A, B) is said to be **compatible** if $||A - B|| \geq s$.

A **trajectory** $\xi_{\mathbb{A}}$ of a disc $\mathbb{A}$ from a position A_0 to a position A_1 is a continuous, rectifiable curve $\xi_{\mathbb{A}} : [0, 1] \to \Re^2$, where $\xi_{\mathbb{A}}(0) = A_0$, $\xi_{\mathbb{A}}(1) = A_1$. The length $\ell(\xi_{\mathbb{A}})$ of a trajectory $\xi_{\mathbb{A}}$ is simply the arc-length of its trace.

A **(coordinated) motion** m of a disc pair $(\mathbb{A}, \mathbb{B})$ from a placement (A_0, B_0) to a placement (A_1, B_1) is a pair $(\xi_{\mathbb{A}}, \xi_{\mathbb{B}})$, where $\xi_{\mathbb{A}}$ (resp. $\xi_{\mathbb{B}}$) is a trajectory of $\mathbb{A}$ (resp. $\mathbb{B}$) from position A_0 to a position A_1 (resp. position B_0 to a position B_1). A motion is said to be **compatible** if all of its associated placements are compatible. Unless otherwise specified, all placements and motions that arise in this paper are compatible.

The **separation** between two placements (A_0, B_0) and (A_1, B_1) is simply $||A_1 - A_0|| + ||B_1 - B_0||$, the net Euclidean distance traversed by discs $\mathbb{A}$ and $\mathbb{B}$ in moving from placement (A_0, B_0) to placement (A_1, B_1). The separation clearly provides a lower bound on the **length** $\ell(m)$ of any motion m between placements (A_0, B_0) and (A_1, B_1), defined as the sum of the lengths of its associated trajectories. Finally, the **(collision-free) distance** $d(P_0, P_1)$ between two placements P_0 and P_1 is the minimum possible length over all compatible motions m from P_0 to P_1. We refer to any compatible motion m between P_0 and P_1 satisfying $\ell(m) = d(P_0, P_1)$ as an **optimal** motion from P_0 to P_1.

3 The general approach

Suppose that the discs have initial placement $P_0 = (A_0, B_0)$ and final placement $P_1 = (A_1, B_1)$, and let $m = (\xi_{\mathbb{A}}, \xi_{\mathbb{B}})$ be any motion from P_0 to P_1. Denote by $\widehat{\xi_{\mathbb{A}}}$ (resp. $\widehat{\xi_{\mathbb{B}}}$) the closed curve defining the boundary of the convex hull of $\xi_{\mathbb{A}}$ (resp. $\xi_{\mathbb{B}}$). Since $\xi_{\mathbb{A}}$ (resp. $\xi_{\mathbb{B}}$), together with the segment $\overline{A_0A_1}$ (resp. $\overline{B_0B_1}$), forms a closed curve whose convex hull has boundary $\widehat{\xi_{\mathbb{A}}}$ (resp. $\widehat{\xi_{\mathbb{B}}}$), it follows from convexity that:

$$\ell(\xi_{\mathbb{A}}) \geq \ell(\widehat{\xi_{\mathbb{A}}}) - \ell(\overline{A_0A_1}) \quad \text{and} \quad \ell(\xi_{\mathbb{B}}) \geq \ell(\widehat{\xi_{\mathbb{B}}}) - \ell(\overline{B_0B_1}) \tag{1}$$

Given a placement P, we refer to the angle formed by the vector from $\mathbb{B}[P]$ to $\mathbb{A}[P]$ with respect to the x-axis as the *angle* of the placement P. Let $[\theta_0, \theta_1]$ be the range of angles counter-clockwise between the angle of P_0 and P_1.

Observation 1 *Let m be any motion from P_0 to P_1, and let I denote the range of angles realized by the set of placements in m. Then $[\theta_0, \theta_1] \subseteq I$ or $S^1 - [\theta_0, \theta_1] \subseteq I$.*

We use Observation 1 to categorize the motions we describe into *net clockwise* and *net counter-clockwise* motions. Net clockwise motions satisfy $S^1 - [\theta_0, \theta_1] \subseteq I$ and net counter-clockwise motions satisfy $[\theta_0, \theta_1] \subseteq I$.

Since any motion is either net clockwise or net counter-clockwise (or both) it suffices to optimize over net clockwise and net counter-clockwise motions separately. The following lemma sets out sufficient conditions for net (counter-)clockwise motions to be optimal.

Lemma 1 *Let $m = (\xi_{\mathbb{A}}, \xi_{\mathbb{B}})$ be any net (counter-)clockwise motion from P_0 to P_1 satisfying the following properties:*

1. $\widehat{\xi_{\mathbb{A}}} = \xi_{\mathbb{A}} \cup \overline{A_0A_1}$ *and* $\widehat{\xi_{\mathbb{B}}} = \xi_{\mathbb{B}} \cup \overline{B_0B_1}$; *and*
2. $\ell(\widehat{\xi_{\mathbb{A}}}) + \ell(\widehat{\xi_{\mathbb{B}}})$ *is minimized over all possible net (counter-)clockwise motions.*

Then m is a shortest net (counter-)clockwise motion from P_0 to P_1.

Proof. Let $m' = (\xi'_{\mathbb{A}}, \xi'_{\mathbb{B}})$ be any net (counter-)clockwise motion from P_0 to P_1. It follows from property 1 that $\ell(m) = \ell(\widehat{\xi_{\mathbb{A}}}) - \ell(\overline{A_0A_1}) + \ell(\widehat{\xi_{\mathbb{B}}}) - \ell(\overline{B_0B_1})$. Furthermore, from 2 we know that $\ell(\widehat{\xi_{\mathbb{A}}}) + \ell(\widehat{\xi_{\mathbb{B}}}) \leq \ell(\widehat{\xi'_{\mathbb{A}}}) + \ell(\widehat{\xi'_{\mathbb{B}}})$. Thus, using inequality (1), we have

$$\ell(m) \leq \ell(\widehat{\xi'_{\mathbb{A}}}) - \ell(\overline{A_0A_1}) + \ell(\widehat{\xi'_{\mathbb{B}}}) - \ell(\overline{B_0B_1}) \leq \ell(m'). \quad \square$$

When a net (counter-)clockwise motion satisfies the two properties of Lemma 1, we say it is *(counter-)clockwise optimal.* Figure 1 illustrates two motions from the placement (A_0, B_0) to the placement (A_1, B_1). The blue motion, where $\mathbb{B}$ first pivots about A_0 and moves to B_1, followed by $\mathbb{A}$ moving from A_0 to A_1, is counter-clockwise optimal. The yellow motion, where $\mathbb{A}$ first pivots about B_0 and moves to A_1, followed by $\mathbb{B}$ moving from B_0 to B_1, is clockwise optimal (as one can check following the proofs of Section 4). However, only the yellow motion is optimal.

While property 1 of Lemma 1 is typically easy to verify, property 2 is less straightforward. Fortunately, an application of Cauchy's surface area formula (Theorem 2) suffices in all cases of interest. Theorem 2 allows us

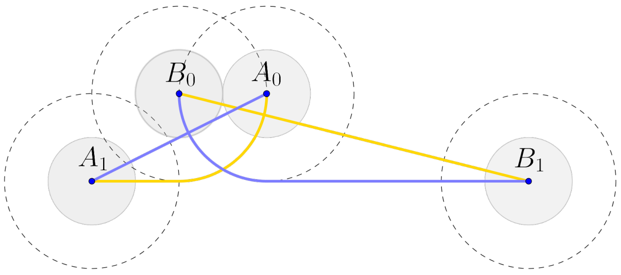


Figure 1: Clockwise (yellow) and counter-clockwise (blue) motions satisfying the two properties of Lemma 1.

to translate the problem of measuring lengths of curves into a problem of measuring the support functions of $\widehat{\xi_{\mathbb{A}}}$ and $\widehat{\xi_{\mathbb{B}}}$ at certain critical angles. As it turns out, checking the properties of a curve's support function is much easier than those of it's length.

Definition 1 *Let C be a closed curve. The* ***support function*** *$h_C : S^1 \to \mathbb{R}$ of C is defined as*

$$h_C(\theta) = \sup\{x\cos\theta + y\sin\theta : (x,y) \in C\}.$$

For an angle θ, the set points that realize the supremum above are called ***support points****, and the line oriented at angle $\frac{\pi}{2} + \theta$ going through the support points is called the* ***support line*** *(see Figure 2).*

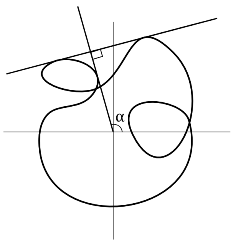


Figure 2: The (two) support points and support line at angle θ of a given curve.

Theorem 2 *(Cauchy's surface area formula [2, Section 5.3]) Let C be a closed convex curve in the plane and h_C be the support function of C. Then*

$$\ell(C) = \int_0^{2\pi} h_C(\theta)d\theta. \tag{2}$$

As noted in [3], it follows from Theorem 2 that we can bound the length of two convex curves in the plane:

Corollary 3 *Let C_1 and C_2 be closed convex curves in the plane. Then the sum of their lengths can be expressed as follows:*

$$\ell(C_1) + \ell(C_2) = \int_0^{2\pi} (h_1(\theta) + h_2(\pi+\theta))\, d\theta, \tag{3}$$

where h_i is the support function of C_i.

In order to argue the optimality of our motions, we use the following observation that provides a bound on support functions in specified ranges. Let $h_{\mathbb{A}}$ (resp. $h_{\mathbb{B}}$) denote the support function of $\widehat{\xi_{\mathbb{A}}}$ (resp. $\widehat{\xi_{\mathbb{B}}}$), and let $h_{\mathbb{AB}}(\theta)$ denote the sum $h_{\mathbb{A}}(\theta) + h_{\mathbb{B}}(\pi+\theta)$. Recall that s is the radii sum of the two disks.

Observation 2 *Let P_0 and P_1 be two configurations and let $[\theta_0, \theta_1]$ be the range of angles counter-clockwise between the angles of P_0 and P_1. Then, for all net counter-clockwise motions from P_0 to P_1, and $\theta \in [\theta_0, \theta_1]$, $h_{\mathbb{AB}}(\theta) \geq s$. Similarly, for all net clockwise motions and $\theta \in S^1 - [\theta_0, \theta_1]$, $h_{\mathbb{AB}}(\theta) \geq s$.*

For all support angles, the support function $h_{\mathbb{A}}$ (resp. $h_{\mathbb{B}}$) is lower bounded by the support function $H_{\mathbb{A}}$ (resp. $H_{\mathbb{B}}$) of $\overline{A_0A_1}$ (resp. $\overline{B_0B_1}$), since $\overline{A_0A_1} \subset \widehat{\xi_{\mathbb{A}}}$ (resp. $\overline{B_0B_1} \subset \widehat{\xi_{\mathbb{B}}}$). For all of our specified motions, it will be the case that whenever this lower bound is not achieved, the one given by Observation 2 is.

In the next section we give explicit constructions of optimal motions for many initial-final configuration pairs. This includes, of course, all those whose associated trajectories correspond to two straight segments, what we refer to as **straight-line motions**. In other cases, we construct both the clockwise and counter-clockwise optimal motions, one of which must be optimal among all motions.

4 Optimal paths for two discs

Our constructions of shortest (counter-)clockwise motions can be summarized by the following theorem:

Theorem 4 *Let $\mathbb{A}$ and $\mathbb{B}$ be two discs with radius sum s in an obstacle-free plane with arbitrary initial and final placements $P_0 = (A_0, B_0)$ and $P_1 = (A_1, B_1)$. Then there is a shortest motion from P_0 to P_1 composed of at most three circular arcs of radius s and straight segments.*

We devote this entire section to the identification and exhaustively treatment of various cases of Theorem 4. The paths that we identify in each case also allow us to provide the following unified characterization of the optimal path length, covering all cases:

Corollary 5 *Let $H_{\mathbb{A}}$ and $H_{\mathbb{B}}$ be the support functions of the segments $\overline{A_0A_1}$ and $\overline{B_0B_1}$ respectively, $H_{\mathbb{AB}}(\theta) := H_{\mathbb{A}}(\theta)+H_{\mathbb{B}}(\pi+\theta)$, and m be an optimal motion between P_0 and P_1. Let $[\theta_0,\theta_1]$ be the range of angles counter-clockwise between P_0 and P_1. Then*

$$\ell(m) = \min\Big(\int_0^{2\pi} \max(H_{\mathbb{AB}}(\theta), s\cdot \mathbb{1}_{[\theta_0,\theta_1]})\,d\theta, \int_0^{2\pi} \max(H_{\mathbb{AB}}(\theta), s\cdot \mathbb{1}_{S^1-[\theta_0,\theta_1]})\,d\theta \Big)$$

where $\mathbb{1}_{[a,b]}$ is the indicator function of the interval $[a,b]$.

Though the expression in Corollary 5 looks daunting, the only difference between the two integrals is the indicator function used. The support functions themselves can be expressed in closed form and the integrals are clearly lower bounds on the path length by Corollary 3 and Observation 2. We emphasize that the integrals can be expressed in closed form if needed, albeit with some cases involved.

We now introduce some additional tools that will help us classify the initial and final placements into different cases.

Let p and q be arbitrary points. We denote by $\mathrm{circ}_s(p)$ the open disk of radius s centred at point p. In addition we use (i) $\mathrm{corr}_s(p,q)$ to denote the corridor formed by the Minkowski sum of the line segment $\overline{pq}$ and an open disk of radius s, and (ii) $\mathrm{cone}_s(p,q)$ to denote the cone formed by all half-lines from p that intersect $\mathrm{circ}_s(q)$.

If disc $\mathbb{A}$ is centred at location A then $\mathrm{circ}_s(A)$ corresponds to the locations forbidden to the centre of disc $\mathbb{B}$ in a compatible placement. Note that, if point $A \notin \mathrm{corr}_s(B_0,B_1)$ it is possible to translate $\mathbb{B}$ from B_0 to B_1 without intersecting disc $\mathbb{A}$ with centre at point A.

What follows is a case analysis of various scenarios for the initial and final placements. We first classify the cases by the containment of A_0, A_1, B_0, B_1 within $\mathrm{corr}_s(A_0,A_1)$ and $\mathrm{corr}_s(B_0,B_1)$ (see Table 1). While there appears to be 16 cases—since each point is either contained within a corridor or not—they cluster into just three disjoint collections, referred to as Cases 1, 2 and 3. These are further reduced by symmetries which include (i) interchanging the initial and final placements and (ii) switching the roles of $\mathbb{A}$ and $\mathbb{B}$.

In all cases our specified motion has a common form – with a possible switch of the roles of $\mathbb{A}$ and $\mathbb{B}$. We identify an intermediate position A_{int} (possibly A_0 or A_1) and perform the following sequence of (locally shortest) moves:

1. Move $\mathbb{A}$ from A_0 to A_{int}, avoiding $\mathrm{circ}_s(B_0)$;
2. Move $\mathbb{B}$ from B_0 to B_1, avoiding $\mathrm{circ}_s(A_{\text{int}})$; then
3. Move $\mathbb{A}$ from A_{int} to A_1, while avoiding $\mathrm{circ}_s(B_1)$.

Case	$A_0 \in \mathrm{corr}_s(B_0,B_1)$	$A_1 \in \mathrm{corr}_s(B_0,B_1)$	$B_0 \in \mathrm{corr}_s(A_0,A_1)$	$B_1 \in \mathrm{corr}_s(A_0,A_1)$
1a	`false`	*	*	`false`
1b	*	`false`	`false`	*
2a	`true`	*	`true`	*
2b	*	`true`	*	`true`
3a	`true`	`true`	`false`	`false`
3b	`false`	`false`	`true`	`true`

Table 1: All cases of possible motions. The $*$ entries mean that the specified condition could be `true` or `false`.

Without loss of generality, assume that our initial and final configurations have been *normalized* as follows: B_0 and B_1 lie on the x-axis with B_0 at the origin, B_1 right of B_0. In all but a few special cases, we will only examine motions that are net counter-clockwise; net clockwise optimal motions can be obtained by reflecting the initial and final placements across the x-axis and then examining net counter-clockwise motions.

The net counter-clockwise orientation of our proposed motion $m = (\xi_{\mathbb{A}}, \xi_{\mathbb{B}})$ as well as the fact that $\overline{A_0A_1}$ (resp. $\overline{B_0B_1}$) are part of $\widehat{\xi_{\mathbb{A}}}$ (resp. $\widehat{\xi_{\mathbb{B}}}$) will typically be straightforward to verify. Hence the bulk of our arguments will use Corollary 3 to show that $\ell(\widehat{\xi_{\mathbb{A}}}) + \ell(\widehat{\xi_{\mathbb{B}}})$ is minimized.

4.1 Case 1

It suffices to treat Case 1a, as Case 1b reduces to Case 1a by either symmetry (i) or (ii). In Case 1a, $A_0 \notin \mathrm{corr}_s(B_0,B_1)$, so on the first step we translate $\mathbb{B}$ from B_0 to B_1 in a straight line without touching $\mathbb{A}$. At this point $\mathbb{A}$ can move freely in a straight line from A_0 to A_1, as $B_1 \notin \mathrm{corr}_s(A_0,A_1)$. As we shall see through examining the other cases, Case 1 is the only situation where a straight-line motion is possible.

4.2 Case 2

It suffices to treat Case 2a since Case 2b reduces to Case 2a by symmetry (i); thus we assume that $A_0 \in \mathrm{corr}_s(B_0,B_1)$ and $B_0 \in \mathrm{corr}_s(A_0,A_1)$.

There are many subcases to consider, depending on the location of A_1 and B_1 relative to A_0. We start with the simplest of these, those that arise when $\mathrm{cone}_s(A_0,B_0)$ and $\mathrm{circ}_s(B_1)$ are disjoint (see Figure 3).

4.2.1 $\mathrm{cone}_s(A_0,B_0)$ and $\mathrm{circ}_s(B_1)$ do not intersect

In this case, one can construct zones I-IV in Figure 3 in which each zone has a net counter-clockwise optimal motion of a different form. The zones are constructed through the following tangents and curves:

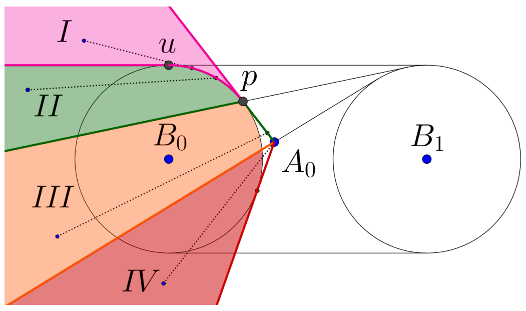


Figure 3: Zones of Case 2 (Section 4.2). Typical paths in each zone are shown as dotted lines.

1. The horizontal tangent through the uppermost point u of $\text{circ}_s(B_0)$. This tangent and the arc of $\text{circ}_s(B_0)$ between u and p (where p is the upper tangent point between A_0 and $\text{circ}_s(B_0)$) separates zone I and II.
2. The tangent through p to $\text{circ}_s(B_1)$. This tangent separates zone II and III.
3. The tangent line from A_0 to $\text{circ}_s(B_1)$. This tangent separates zone III and IV.

Note that zone III and IV may be empty, if the position of A_0 lies below the line tangent to the bottom of $\text{circ}_s(B_0)$ and the top of $\text{circ}_s(B_1)$.

For each zone we specify the location of the intermediate point A_{int} as follows:
zone I: A_{int} is the point A_1.
zone II: A_{int} is the rightmost point of intersection between the tangent from A_1 to $\text{circ}_s(B_1)$ and $\text{circ}_s(B_0)$.
zone III: A_{int} the point of intersection of the tangent from A_1 to $\text{circ}_s(B_1)$ and the tangent from A_0 to $\text{circ}_s(B_0)$.
zone IV: A_{int} is the point A_0.

We then define points T_0 and T_1 which are the lower points of tangency to $\text{circ}_s(A_{\text{int}})$ from B_0 and B_1 respectively. Our three-step generic motion involves:

1. Moving $\mathbb{A}$ on the shortest path from A_0 to A_{int}, avoiding $\text{circ}_s(B_0)$. This may involve rotating $\mathbb{A}$ counter-clockwise about B_0 in a range of angles $[\alpha_0, \alpha_1]$.
2. Moving $\mathbb{B}$ from B_0 to B_1 avoiding $\text{circ}_s(A_{\text{int}})$. This involves translating $\mathbb{B}$ from B_0 to T_0, rotating $\mathbb{B}$ counter-clockwise about A_{int} from T_0 to T_1 in a range of angles $[\beta_0, \beta_1]$, and then translating $\mathbb{B}$ from T_1 to B_1.
3. Translating $\mathbb{A}$ from A_{int} to A_1 (collision-free by the disjointness of $\text{cone}_s(A_0, B_0)$ and $\text{circ}_s(B_1)$).

Claim 1 *The motions described above are optimal.*

Proof. It is easy to check that property 1 of Lemma 1 is satisfied. To show that property 2 holds as well we verify that $h_{\mathbb{AB}}(\theta)$ matches its lower bound and then use Corollary 3. We check that, for all angles θ, either $h_{\mathbb{AB}}(\theta) = s$ or is determined by $\mathbb{A}$ and $\mathbb{B}$ in their initial or final position.

If the range of angles $[\alpha_0, \alpha_1]$ is non-empty then, by construction, α_0 is normal to the right tangent from A_0 to $\text{circ}_s(B_0)$ and α_1 is normal to the left tangent from A_{int} to $\text{circ}_s(B_0)$. This ensures that for the range of angles $[\alpha_0, \alpha_1]$, B_0 is a support point of $h_{\mathbb{B}}(\theta + \pi)$ while the support point of $h_{\mathbb{A}}(\theta)$ lies on the arc of the circle traversed by $\mathbb{A}$. Hence $h_{\mathbb{AB}}(\theta) = s$ for $\theta \in [\alpha_0, \alpha_1]$.

Similarly, if the range of angles $[\beta_0, \beta_1]$ is non-empty then, by construction, β_0 is normal to the right tangent from B_0 to $\text{circ}_s(A_{\text{int}})$ and β_1 is normal to the left tangent from B_1 to $\text{circ}_s(A_{\text{int}})$ (equivalently, the left tangent from A_{int} to $\text{circ}_s(B_1)$). This ensures that for the range of angles $[\beta_0, \beta_1]$, A_{int} is the support point of $h_{\mathbb{A}}(\theta)$ while the support point of $h_{\mathbb{B}}(\theta + \pi)$ lies on the arc of the circle traversed by $\mathbb{B}$. Hence $h_{\mathbb{AB}}(\theta) = s$ for $\theta \in [\beta_0, \beta_1]$.

For angles in $S^1 - [\beta_0, \beta_1] - [\alpha_0, \alpha_1]$, it is easy to confirm that either A_0 or A_1 must be one support point, and either B_0 or B_1 must be the other. $\square$

Note that if A_1 is replaced by any point on the segment joining A_{int} and A_1 the optimal motion is unchanged, except for a shortening of the third move. Consequently, although they do not fall under case 2 according to our classification, configurations in which A_1 lies in the small white triangle at the apex of $\text{cone}_s(A_0, B_0)$ (see Figure 3) are completely covered by the analysis above.

4.2.2 $\text{cone}_s(A_0, B_0)$ and $\text{circ}_s(B_1)$ intersect

Although conceptually very similar (for the most part), the optimal motions that arise when $\text{cone}_s(A_0, B_0)$ and $\text{circ}_s(B_1)$ intersect are complicated by the fact that the path from A_{int} to A_1 must avoid $\text{circ}_s(B_1)$. In particular, placements of A_{int} inside $\text{circ}_s(B_1)$ are forbidden.

The modifications we require to the paths in Section 4.2.1 are to those originating in zone IV. Previously, the intermediate point A_{int} in zone IV was always the point A_0. However in exceptional cases, such as when $A_0 \in \text{circ}_s(B_1)$, A_0 is infeasible as an intermediate point. Let t be this intermediate point, which we define below depending on the relative position of B_0, B_1, and A_0.

Let p be the tangent point between A_0 and $\text{circ}_s(B_0)$. If $\text{circ}_s(B_0)$ and $\text{circ}_s(B_1)$ intersects, let q be that intersection point. If $A_0 \in \text{circ}_s(B_1)$, let r be the intersection point between $\text{circ}_s(B_1)$ and the line through A_0 and p.

- If q exists and p is clockwise of q on $\text{circ}_s(B_0)$, then let t be q.

- If q exists and p is counter-clockwise of q on $\mathrm{circ}_s(B_0)$, then let t be r if r exists, and let t be A_0 otherwise.

By using this choice of A_{int} for zone IV and taking care of the exceptional configurations in Section 4.5, one can show through similar proofs to those in Section 4.2.1 that the zone IV motion is net counter-clockwise optimal.

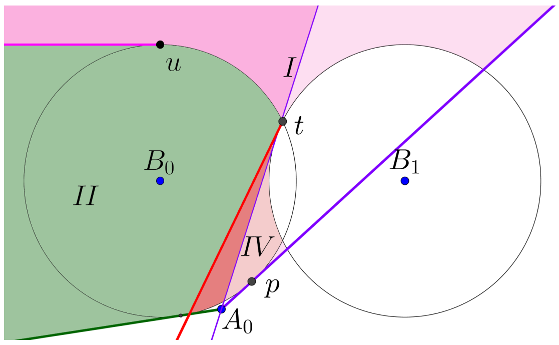


Figure 4: Zone classification when $\mathrm{cone}_s(A_0, B_0)$ and $\mathrm{circ}_s(B_1)$ intersect. In this case we use the tangent to $\mathrm{circ}_s(B_1)$ at t to form the left boundary of zone IV. Regions addressed by Lemma 6 are coloured in lighter shades.

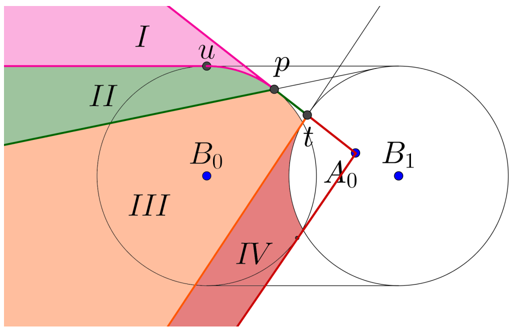


Figure 5: Zone classification when $\mathrm{cone}_s(A_0, B_0)$ and $\mathrm{circ}_s(B_1)$ intersect and $A_0 \in \mathrm{circ}_s(B_1)$.

A more serious complication arises when A_0 occupies an extreme position in the bottom of $\mathrm{corr}_s(B_0, B_1)$ (see Fig. 4). In this situation, when A_1 lies within both $\mathrm{cone}_s(A_0, B_0)$ and $\mathrm{cone}_s(A_0, B_1)$, the motion suggested by our previous zone I analysis does not satisfy the properties needed to support a proof of optimality using Lemma 1. Fortunately, this corresponds to a special case of a more general scenario in which we can prove that there exists a clockwise-optimal motion that is globally optimal (making it unnecessary to try and construct a counter-clockwise optimal motion). Furthermore, the clockwise motion will correspond to a

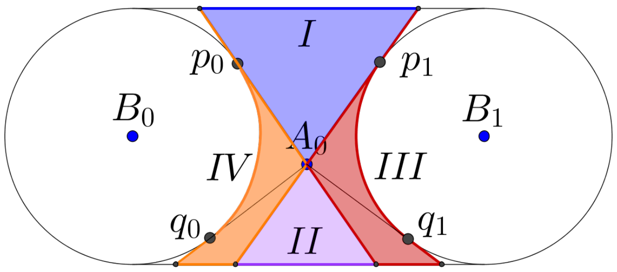


Figure 6: Zones of Case 3 (Section 4.3). Typical paths in each zone are shown as dotted lines.

vertically reflected version of a counter-clockwise motion that we've proven optimal above. A technical lemma (Lemma 6) describing this scenario is given in Section 4.4.

4.3 Case 3

Given the results of our Case 1 and Case 2 analysis (and its symmetric counterpart when the initial and final configurations are interchanged), we can assume that (i) both A_0 and A_1 lie in $\mathrm{corr}_s(B_0, B_1)$.

After removing cases in which the motion is counter-clockwise optimal (by Lemma 6), Case 3 becomes highly constrained. We focus on situations where the motion is not known to be clockwise optimal.

4.3.1 Counter-clockwise optimal motions

Figure 6 illustrates the zone definition for all cases where the optimal motion is not known to be net-clockwise. The zones I-IV of Figure 6 are defined by the following curves:

1. The two upper tangents from A_0 to $\mathrm{circ}_s(B_0)$ and $\mathrm{circ}_s(B_1)$ (through tangent points p_i). These tangents separate zone I from the rest of the zones. The tangent from A_0 to p_1 forms the left boundary of zone II if A_0 is below the tangent from the bottom of $\mathrm{circ}_s(B_0)$ to the top of $\mathrm{circ}_s(B_1)$. The tangent from A_0 to p_0 forms part of the right boundary of zone II.
2. The two horizontal tangents from $\mathrm{circ}_s(B_0)$.
3. The lower tangent from A_0 to $\mathrm{circ}_s(B_0)$ and $\mathrm{circ}_s(B_1)$ (through tangent points q_i). The tangent from A_0 to q_1 (resp. q_0) form part of the right (resp. left) boundary for zone III (resp. zone IV). The tangent from A_0 to q_0 forms the left boundary of zone II if A_0 is above the tangent from below $\mathrm{circ}_s(B_0)$ to above $\mathrm{circ}_s(B_1)$.
4. The arc of $\mathrm{circ}_s(B_0)$ (resp. $\mathrm{circ}_s(B_1)$) from p_0 to t_0 (resp. p_1 to t_1). If the tangent from A_0 to p_0 (resp. to p_1) does not intersect $\mathrm{circ}_s(B_1)$ (resp.

circ$_s(B_0)$), then t_0 is q_0 (resp. t_1 is q_1). Otherwise, t_0 (resp. t_1) is the intersection point.

5. The arc of circ$_s(B_0)$ (resp. circ$_s(B_1)$) from t_0 to q_0 (resp. t_1 to q_1). These arcs forms part of the left and right boundaries of zone II.

We now specify, for each zone, the location of A_{int}, and define T_0 and T_1 to be the lower tangent points of B_0 and B_1 to circ$_s(A_{\text{int}})$ respectively.
zone I: A_{int} is the point A_1.
zone II: A_{int} is the point A_0.
zone III: A_{int} is the intersection point of the tangent from A_1 to the circ$_s(B_0)$ and the tangent from A_0 to circ$_s(B_1)$.
zone IV: A_{int} is the intersection point of the tangent from A_0 to the circ$_s(B_0)$ and the tangent from A_1 to circ$_s(B_1)$.

Our generic three-stage motion then becomes:

1. Move $\mathbb{A}$ on a straight line from A_0 to A_{int}
2. Move $\mathbb{B}$ from B_0 to B_1 avoiding circ$_s(A_{\text{int}})$. This involves moving $\mathbb{B}$ to T_0, rotating $\mathbb{B}$ counter-clockwise about A_0 to T_1 in a range of angles $[\beta_0, \beta_1]$, and then moving $\mathbb{B}$ from T_1 to B_1.
3. Move $\mathbb{A}$ on a straight line motion from A_{int} to A_1.

Note that in zone IV of Figure 6, all optimal counter-clockwise motions are of exactly the same from as zone III of Case 2.

Claim 2 *The motions described above are optimal.*

Proof. As before, it is easy to check that property 1 of Lemma 1 is satisfied. To show that property 2 holds as well we verify that $h_{\mathbb{AB}}(\theta)$ matches its lower bound and then use Corollary 3. We check that, for all directions θ, either $h_{\mathbb{AB}}(\theta) = s$ or the support function is determined by $\mathbb{A}$ and $\mathbb{B}$ in their initial or final position.

By construction, β_0 is normal to the right tangent from B_0 to circ$_s(A_{\text{int}})$ (equivalently, the right tangent from A_{int} to circ$_s(B_0)$) and β_1 is normal to the left tangent from B_1 to circ$_s(A_{\text{int}})$ (equivalently, the left tangent from A_{int} to circ$_s(B_1)$). This ensures that for the range of angles $[\beta_0, \beta_1]$, A_{int} is one support point while the other support point lies on the arc of the circle traversed by $\mathbb{B}$. Hence $h_{\mathbb{AB}}(\theta) = s$ for $\theta \in [\beta_0, \beta_1]$.

For angles in $S^1 - [\beta_0, \beta_1]$, it is easy to confirm that either A_0 or A_1 must be one support point, and either B_0 or B_1 must be the other. □

In Figures 8 and 9 of Section 4.4, we show how some exceptional cases are handled by Lemma 6, such as the case where the circles are intersecting.

4.4 Clockwise-minimal motions

Lemma 6 describes a set of placements for which the optimal motion is net clockwise. This lemma will deal with subcases within Case 2 and Case 3 for which the demonstration of net counter-clockwise optimal motions seems beyond the reach of Cauchy's surface area formula. The proof of this lemma involves the explicit construction of a clockwise motion that is at least as good as any counter-clockwise motion. The proof is elementary, but lengthy. For this reason, we provide the proof in [4].

Lemma 6 *Suppose that $A_0, A_1 \in corr_s(B_0, B_1)$ and $cone_s(A_i, B_i)$ intersects $circ_s(B_j)$ for any $i, j \in \{0, 1\}$ with $i \neq j$. Let H_i be the half-space under the tangent of $cone_s(A_i, B_i)$ intersecting $circ_s(B_j)$ and suppose that $A_j \in H_i$. If A_i is below the line connecting B_0 with B_1, then the optimal motion must be net clockwise. Otherwise the optimal motion is net counter-clockwise.*

4.5 Exceptional configurations in Case 2

When cone$_s(A_0, B_0)$ and circ$_s(B_1)$ intersect, our arguments from Section 4.2 can be insufficient in a variety of ways. For example, when A_0 is as depicted in Figure 4 and A_1 is situated under the lower intersection point of circ$_s(B_0)$ and circ$_s(B_1)$, the zone I motion we previously outlined would first move $\mathbb{A}$ from A_0 to A_1, and then rotate $\mathbb{B}$ from B_0 to B_1. However, one can verify that this motion does not minimize the support function while B is rotating, as the support point of $h_{\mathbb{A}}$ while B is rotating switches from A_1 to A_0 before the rotation is complete.

Another example can be seen in Figure 7. Suppose A_1 is situated right of circ$_s(B_1)$ and above the lower horizontal tangent of circ$_s(B_1)$. Then the zone IV motion previously outlined would first rotate $\mathbb{B}$ from B_0 to B_1, and then rotate $\mathbb{A}$ from A_0 to A_1 about circ$_s(B_1)$. However, during this last rotation, the support function is not minimized, as the support point of $h_{\mathbb{B}}$ switches from B_1 to B_0 once $\mathbb{A}$ rotates past $3\pi/2$.

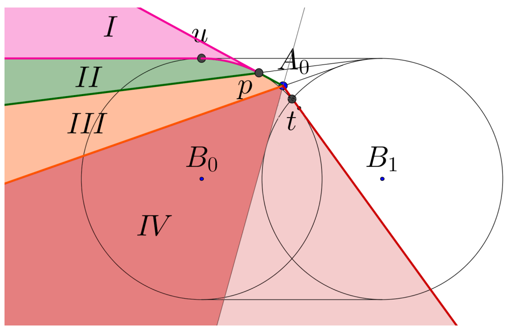


Figure 7: Zone classification when cone$_s(A_0, B_0)$ and circ$_s(B_1)$ intersect. Regions addressed by Lemma 6 are coloured in lighter shades.

Fortunately, both of these inefficiencies are addressed by Lemma 6, as in both cases the optimal motion is net clockwise. By an exhaustive treatment, one can verify that the only cases where our Case 2 arguments fail are once for which the optimal motion is net clockwise.

4.6 Exceptional configurations in Case 3

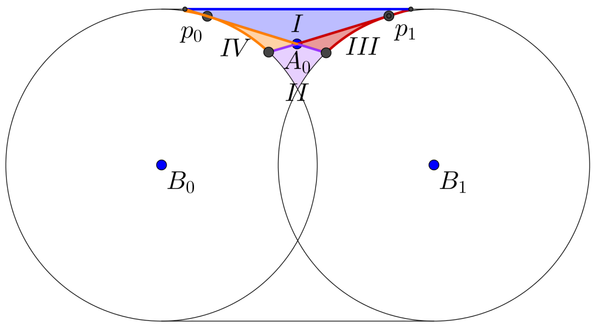


Figure 8: Case 3, when s-circ(B_0) and s-circ(B_1) intersect. A_0 and A_1 are both above the intersection.

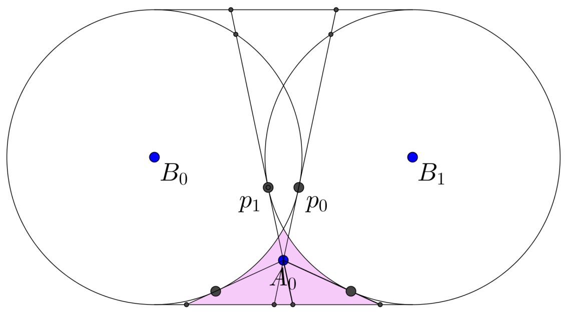


Figure 9: Case 3, when s-circ(B_0) and s-circ(B_1) intersect. A_0 and A_1 are both below the intersection.

As stated in Section 4.3, Case 3 is highly constrained, and the zones are simpler than those in Case 2. When circ$_s(B_0)$ and circ$_s(B_1)$ intersect, the description of the zones change slightly, and certain configurations become impossible. For example, when s-circ(B_0) and s-circ(B_1) intersect, observe that the constraints force either A_0 and A_1 to be both above the B circles, or both below. If one of the A_i's were above and the other below, then we would be in Case 2. Figures 8 and 9 outline two cases in which the zones differ from our previous description.

In Figure 8, the zones are constructed similarly to Figure 6. However, the lower boundary of zones II-IV is now formed by the boundaries of circ$_s(B_0)$ and circ$_s(B_1)$ above their intersection point. The motions in zones I-IV, as well as proofs of optimality, remain valid for all four zones.

In Figure 9, we note that the optimal motion is clockwise by Lemma 6. After vertically reflecting Figure 9 to obtain the clockwise optimal motion, note that it becomes Figure 8 exactly, and thus is already handled by our previous analysis.

As far as we know, our tools are limited to the case when the robots are discs in 2D. Indeed, when the robots are spheres in 3D, even if the initial and final positions of the robot are coplanar, we are unsure if the shortest path stays within the plane (except in special cases). The 3D extension of the problem as well as the 2D problem with obstacles remain subjects for future exploration.

References

[1] Yui-Bin Chen and Doug Ierardi. Optimal motion planning for a rod in the plane subject to velocity constraints. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, SCG '93, pages 143–152, New York, NY, USA, 1993. ACM.

[2] H. G. Eggleston. *Convexity.* Cambridge Tracts in Mathematics and Mathematical Physics, No. 47. Cambridge University Press, New York, 1958.

[3] Christian Icking, Günter Rote, Emo Welzl, and Chee-Keng Yap. Shortest paths for line segments. *Algorithmica*, 10(2-4):182–200, 1993.

[4] D. Kirkpatrick and P. Liu. Characterizing minimum-length coordinated motions for two discs. *ArXiv e-prints*, July 2016.

[5] Jacob T Schwartz and Micha Sharir. On the piano movers' problem: III. coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983.

[6] Micha Sharir and Shmuel Sifrony. Coordinated motion planning for two independent robots. *Ann. Math. Artif. Intell.*, 3(1):107–130, 1991.

[7] Paul G. Spirakis and Chee-Keng Yap. Strong NP-hardness of moving many discs. *Inf. Process. Lett.*, 19(1):55–59, 1984.

[8] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3260–3267, 2011.

[9] Chee Yap. *Coordinating the motion of several discs.* Robotics Report. Department of Computer Science, New York University, 1984.

Maximizing the Sum of Radii of Disjoint Balls or Disks

David Eppstein*

Abstract

Finding nonoverlapping balls with given centers in any metric space, maximizing the sum of radii of the balls, can be expressed as a linear program. Its dual linear program expresses the problem of finding a minimum-weight set of cycles (allowing 2-cycles) covering all vertices in a complete geometric graph. For points in a Euclidean space of any finite dimension d, this graph can be replaced by a sparse subgraph obeying a separator theorem. This graph structure leads to an algorithm for finding the optimum set of balls in time $O(n^{2-1/d})$, improving the $O(n^3)$ time of a naive cycle cover algorithm.

1 Introduction

Researchers of map labeling (the placement of non-overlapping text labels on maps or other visualizations) have studied various problems of assigning shapes of maximum size to a given set of points in the plane [5,13,16,21]. One simple case of this problem, finding circles centered at the given points that maximize the minimum radius, can be solved by finding the closest pair of points and setting all radii equal to half of this pair's distance. However, this measure of solution quality penalizes the label sizes even for points far away from the closest pair, where larger radii could be used without overlap. On the other hand, an L_2 measure of solution quality (maximizing the sum of disk areas) would also be unsatisfactory: even with only two points to be labeled, the L_2 solution would assign zero radius to one point. In this paper, we study the L_1 version of this problem, in which we are given as input n points and must maximize the sum of radii of disjoint disks centered at those points. Two points can be optimally labeled by two equal-radius disks, and this criterion has the largest value of p among L_p criteria (maximizing sums of pth powers of the radii) that allow this equal-radius solution.

The same problem can be generalized to arbitrary metric spaces, with some care about definitions. We may specify a ball by its center (a point in the space) and radius (a real number); we consider balls to be distinct when they have different centers or radii, even when they contain the same sets of points. For a *geodesic* metric space (in which every two points can be joined by an isometrically embedded line segment), a pair of balls has a nonempty intersection if and only if their sum of radii equals or exceeds the distance of their centers. In other metric spaces (for instance finite metric spaces) two balls may have larger sum of radii than their center distance without intersecting. Nevertheless we will say that two balls in an arbitrary metric space *overlap* when their sum of radii exceeds the center distance, and are *nonoverlapping* otherwise. If the sum of radii equals the center distance, we sat that the two balls *touch.*

Then in any metric space, finding a set of nonoverlapping metric balls with n given center points p_i, maximizing the sum of non-negative radii r_i, can be expressed as a linear program. The objective is to maximize the linear function $\sum r_i$, subject to constraints that each pair of balls remain nonoverlapping, i.e. that $r_i + r_j \le d(p_i, p_j)$. This linear program has two variables per constraint, a well-studied special case of linear programming. But although strongly polynomial algorithms for feasibility with two variables per constraint are known [4,12], they do not extend to optimization, and their running time is higher than might be desired. Therefore, it remains of interest to find a purely combinatorial algorithm for the problem, with as low a running time as possible.

In this paper, we provide such a combinatorial algorithm, running in cubic time for general metrics and subquadratic time for low-dimensional Euclidean spaces.

1.1 New results

We prove the following results:

- Finding metric balls with maximum sum of radii is equivalent under linear programming duality to finding a minimum-length set of cycles (allowing 2-cycles) that cover all vertices of the complete geometric graph on the given centers. The maximum sum of radii equals half of the minimum total cycle length. By reducing cycle covers to weighted matching, the optimal set of balls in any metric space can be constructed in cubic time.

- For points in Euclidean spaces of bounded dimension d, the edges of the optimal cycle cover can be found in a subgraph of the complete geometric graph, the intersection graph of nearest neighbor balls of the points. This graph is sparse, having $O(n)$ edges with a constant of proportionality depending singly-exponentially in the dimension. Moreover, it obeys a

*Computer Science Department, University of California, Irvine; Irvine, CA, 92697, USA; eppstein@uci.edu. Research supported in part by NSF grant CCF-1228639.

separator theorem with separators of size $O(n^{1-1/d})$, and the graph and its separator decomposition can be constructed in time $O(n \log n)$.

- A separator-based divide and conquer weighted matching algorithm can find an optimal cycle cover and set of nonoverlapping balls in time $O(n^{2-1/d})$. In particular, we can find an optimal set of disks in the plane in time $O(n^{3/2})$.

To avoid issues of numerical precision we consider only strongly-polynomial-time algorithms, in a model of computation in which distance computation, arithmetic, and comparisons take constant time per operation.

1.2 Related research

Along with the map labeling research discussed above, researchers have studied other types of geometric optimization problems in which the optimization criterion is a sum of radii of balls or circles. These include finding a set of k balls with given centers, drawn from a larger set of n points, that cover all points and minimize the sum of radii of the balls [10], finding a connected set of disks in the plane with given centers that minimize the sum of radii [2], and finding both a collection of disks centered at a subset of input points that covers all input points, and a tour connecting the disk centers, minimizing a combination of disk radii and tour length [1].

2 Equivalence to cycle cover

Let p_i ($i = 0, \dots n-1$) be a set of points in a metric space, with distances $d(p_i, p_j)$. The maximum sum of radii problem can be expressed as a linear program:

$$\text{maximize } \sum r_i$$

subject to the inequality constraints

$$\begin{aligned} \forall i: \quad & r_i \geq 0 \\ \forall i, j: \quad & r_i + r_j \leq d(p_i, p_j). \end{aligned}$$

By linear programming duality [20, Ch. 12] this has the same value as the dual linear program:

$$\text{minimize } \sum w_{ij} d(p_i, p_j)$$

subject to the inequality constraints

$$\begin{aligned} \forall i, j: \quad & w_{ij} \geq 0 \\ \forall i: \quad & \sum_j w_{ij} \geq 1 \end{aligned}$$

That is, we must find non-negative weights w_{ij} for the edges of a complete geometric graph such that each vertex has incident edge weights totaling at least one, and minimizing the weighted sum of edge lengths.

Lemma 1 *There is an optimal solution to the dual linear program described above in which the incident edge weights at each vertex total exactly one.*

Proof. Define the excess of a vertex to be the total weight of its incident edges minus one. If any edge has weight greater than one, its weight can be reduced to exactly one, improving the quality of the solution without changing its feasibility. Otherwise, if any vertex has positive excess, we can subtract an equal positive weight from two of its incident edges and add the same weight to the edge between the other endpoints of these edges. By the triangle inequality, this change does not worsen the solution, and again it does not change its feasibility. By making such changes we can reduce the total excess, maintaining feasibility, until eventually all vertices have excess zero. □

This dual program (either in the form first given above, or with the restriction that the weights at each vertex sum to exactly one according to Lemma 1) is the linear programming relaxation of the problem of finding a minimum weight perfect matching in the complete geometric graph. Such a relaxation is primarily used for bipartite graphs, for which it is exact [20, Ex. 12.7]. For non-bipartite graphs such as the complete graph, it has a half-integral optimal solution in which all weights w_{ij} belong to $\{0, 1/2, 1\}$ [20, Ex. 14.8]. Doubling the weights to make them integers, and interpreting each doubled weight as an edge multiplicity, gives us a combinatorial description of the dual solution as a multiset of edges in which (by Lemma 1) each vertex has degree two. That is, we have the following result:

Theorem 2 *The maximum sum of radii of nonoverlapping balls, centered at points p_i of a metric space, equals half of the minimum total edge length of a collection of vertex-disjoint cycles (allowing 2-cycles) spanning the complete geometric graph on the points p_i with edge lengths equal to the distances between the edge endpoints.*

We call such a collection of cycles a *minimum cycle cover.* Theorem 2 can be expressed more briefly as stating that half of the minimum cycle cover length equals the

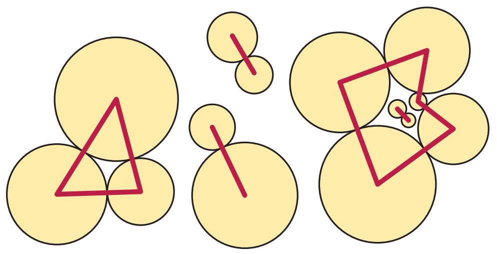


Figure 1: Nonoverlapping disks that (by Corollary 3) maximize the sum of radii for their centers.

maximum sum of radii of nonoverlapping balls. There always exists a minimum cycle cover in which all cycles of more than two edges have odd length, for any long even cycle can be partitioned into two disjoint matchings, and at least one of these matchings gives a covering of the same vertices by 2-cycles that is at least as good. Therefore, from now on we will assume that our cycle covers have no long even cycles.

This result also gives us an easy-to-test optimality condition for the maximum sum of radii problem, that applies to inputs in general position (without extra touching balls beyond the ones required by the solution). In what follows, the *touching graph* of a family of nonoverlapping balls has a vertex for each ball and an edge connecting each two balls that touch. It may differ from the intersection graph in spaces that are not geodesic.

Corollary 3 *Suppose that no two balls in a given family of balls overlap, and that each connected component of the touching graph of the balls is an odd cycle or an isolated edge. Then this family of balls has the maximum sum of radii of any family with the same centers.*

Proof. The touching graph of the balls (viewing each isolated edge as a 2-cycle) gives a cycle cover of length half the sum of radii of the given balls. By Theorem 2 no cycle cover can be shorter and no system of balls with the same centers can have a larger sum of radii. $\square$

Figure 1 shows a family of disks meeting the conditions of the corollary, together with their touching graph.

3 Cycle covers from matchings

The *bipartite double cover* $2G$ of a graph G is the tensor product $G \times K_2$, a bipartite graph with two copies of each vertex of G (one of each color) and two copies of each edge of G (one for each pair of oppositely-colored copies of the endpoint of the edge). We use the same edge weights in G and $2G$.

Lemma 4 *Every perfect matching in $2G$ corresponds (under the mapping that takes each vertex of $2G$ to the corresponding vertex in G) to a cycle vertex cover with equal total length in G. Every cycle vertex cover in G comes from a perfect matching in $2G$ in this way.*

See Figure 2 for an example. Each cycle of more than two vertices in G has two different representations as a set of matched edges in $2G$, but this ambiguity is not a problem. We can find a minimum cycle vertex cover in G by finding a minimum weight perfect matching in $2G$.

A minimum weight perfect matching on a bipartite graph of n vertices and m edges can be found in time $O(mn + n^2 \log n)$ [9]. (Better times are known when the weights are small integers [6].) By solving this problem on the doubled complete geometric graph, the optimal sum of radii can be computed in time $O(n^3)$. However, this is still slower than we would like for Euclidean spaces, and does not yet tell us the radii of the individual balls.

4 From cycle covers to balls

Each odd cycle in a minimum cycle cover of the complete geometric graph corresponds to a unique system of balls that maximizes the sum of radii for those points. Let the cycle have vertices $p_0, p_1, \dots p_{k-1}$ and edge lengths $\ell_i = d(p_i, p_{i+1 \bmod k})$. Then for any j with $0 \le j < i$ set

$$r_j = \sum_i \frac{\pm \ell_i}{2},$$

choosing the signs so that the two edges adjacent to p_j have positive sign and every other point p_i is incident to edges of opposite sign.

We omit the proof that this formula gives us a unique solution, because instead we use a more general solution that also provides radii for the points in 2-cycles. The 2-cycles in the cycle cover cause us more trouble, because their radii may be constrained by other nearby points but are not in general uniquely determined. For instance, in Figure 3, each of the three pairs of touching circles must have nearly-equal radii to avoid overlaps with nearby circles.

To describe how we transform a minimum weight perfect matching problem on $2K_n$ into a feasible system of balls with maximum sum of radii, we need to look

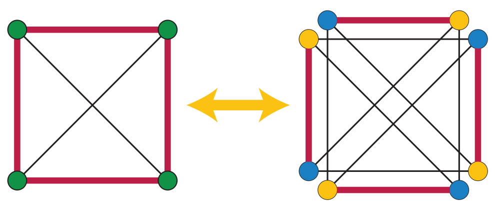


Figure 2: The correspondence between a cycle cover of a graph G (left, in this case, a complete graph K_4) and a matching in $2G$ (right).

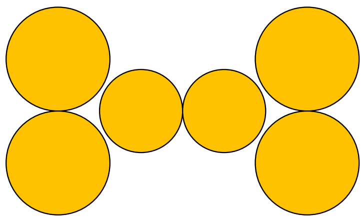


Figure 3: Pairs of disks corresponding to 2-cycles in the minimum cycle cover may constrain the radii of other nearby pairs of disks.

more deeply into the details of the Hungarian algorithm for minimum weight perfect matching. The version of this algorithm described by Tarjan [17, Secs. 8.4 & 9.1] maintains a matching (initially empty) which it uses to orient the graph. Unmatched edges are directed from one side of the bipartition to the other, and matched edges are directed in the opposite direction. In each iteration the algorithm finds a minimum-cost alternating path between two unmatched vertices; here, the cost of a path is the sum of lengths of unmatched edges, minus the sum of lengths of matched edges. It uses this path to extend the matching by one edge.

Because matched edges are subtracted from the path length, each path search involves negative-weight edges. However, the algorithm also maintains a system of non-negative weights that are equivalent (in the sense of having the same shortest paths), allowing Dijkstra's algorithm to be used, and adjusts these weights to keep them non-negative after each iteration. Each iteration increases the number of matched edges, so there are $O(n)$ iterations. The time per iteration can be bounded by the time for Dijkstra's algorithm. Using Fibonacci heaps this gives a total running time of $O(mn + n^2 \log n)$ [9].

To adjust edge weights we maintain dual variables: a real number for each vertex of the bipartite graph. We subtract these variables from the length of each incident unmatched edge, and add these variables to the (negated) length of each incident matched edge. In order for the adjusted edge weights to be non-negative, the algorithm maintains an invariant that the length of each unmatched edge is at least the sum of the dual variables at its endpoints, and each matched edge length equals the sum of the dual variables at its endpoints.

Recall that the graph $2K_n$ to which we apply this algorithm has two vertices for each input point p_i, one of each color (say red and blue). We may visualize the dual variables at these two vertices as two balls (again, one red and one blue) both centered at point p_i. Then the invariants maintained by the Hungarian algorithm can be expressed in our terms as stating that pairs of balls of opposite colors cannot overlap unless they have the same center, and that each matched edge comes from a touching pair of oppositely colored balls (Figure 4). However, this visualization may be somewhat misleading, as we allow the dual variables to be negative.

Lemma 5 *If we are given the dual variables found by the Hungarian algorithm for minimum weight perfect matching in the bipartite graph $2K_n$, we can construct from them a system of nonoverlapping balls with maximum sum of radii, in linear time.*

Proof. Let the two dual variables for point p_i be a_i and b_i, and calculate the radius of the ball centered at p_i to be the average $r_i = (a_i + b_i)/2$ of the two dual variables. For each i and j, we have (by assumption) $a_i + b_j \le d(p_i, p_j)$ and $b_i + a_j \le d(p_i, p_j)$, and averaging these two inequalities gives us that $r_i + r_j \le d(p_i, p_j)$. That is, the balls with radius r_i are non-overlapping.

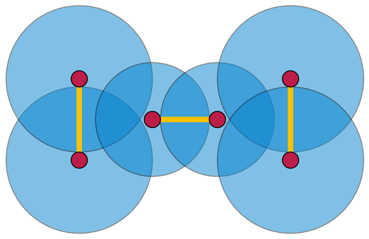


Figure 4: The dual variables for the Hungarian matching algorithm on the bipartite graph $2K_n$ can be visualized as red and blue balls with no bichromatic overlaps, touching at each matched pair.

Each edge of the cycle cover of K_n has two balls of opposite color at its endpoints whose radii sum to the edge length. For each cycle of the cycle cover of K_n, each ball centered at a cycle vertex will contribute to this equality for exactly one of the two incident cycle edges, so the sum of the radii of the blue and red balls centered at the cycle vertices equals the length of the cycle. Therefore, when we average these red and blue radii to give the radii r_i of a single system of balls, the sum of the averaged radii equals half the length of the cycle. Thus, over the whole input, the sum of all the radii r_i equals half the length of the cycle cover. The optimality of this system of nonoverlapping balls then follows from Theorem 2. $\square$

Combining Lemma 4, Lemma 5, and the known algorithms for minimum weight perfect matching in bipartite graphs gives us the following result.

Theorem 6 *Given n points in an arbitrary metric space, we can construct a system of nonoverlapping balls centered at those points, maximizing the sum of radii of the balls, in time $O(n^3)$.*

5 Euclidean speedup

To speed up the search for balls with maximum sum of radii in low-dimensional Euclidean spaces, we replace the complete geometric graph of the previous sections with a sparser graph that contains the optimal cover and behaves the same with respect to testing whether systems of balls are nonoverlapping. Given a system of points p_i in a metric space, let the *nearest neighbor distance* δ_i of each point p_i be

$$\delta_i = \min_{j \neq i} d(p_i, p_j).$$

In Euclidean spaces of bounded dimension, these distances can be found for all points in total time

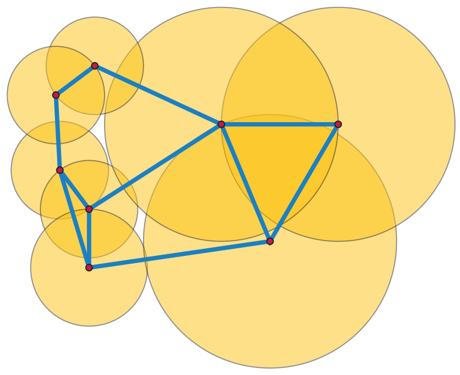


Figure 5: The nearest neighbor balls of a set of points and their nearest neighbor overlap graph $\mathcal{N}$.

$O(n \log n)$ [3, 19]. Define the *nearest-neighbor overlap graph* $\mathcal{N}$ (Figure 5) to be the graph whose vertices are the points p_i, with an edge between two points p_i and p_j when the balls with radius equal to the nearest neighbor distance overlap or touch. That is, p_i and p_j are adjacent exactly when

$$d(p_i, p_j) \leq \delta_i + \delta_j.$$

Lemma 7 *Each edge of the minimum cycle cover of the complete geometric graph belongs to $\mathcal{N}$.*

Proof. Each edge of the cycle cover can be represented by touching balls in a system of nonoverlapping balls. Because each ball centered at p_i in any system of nonoverlapping balls centered at the given points has radius at most δ_i (else it would overlap the ball of the nearest neighbor), two balls of such a system can touch only if the larger balls with radius δ_i touch or overlap. □

Lemma 8 *A system of balls centered at the points p_i is nonoverlapping if and only if, for each edge of $\mathcal{N}$, the two endpoints of the edge are nonoverlapping.*

Proof. A ball centered at p_i with radius greater than δ_i overlaps the ball centered at the nearest neighbor of p_i, which is necessarily adjacent to p_i in $\mathcal{N}$. Otherwise, if all balls have radius at most δ_i, then they can overlap only if their centers are adjacent in $\mathcal{N}$. □

By Lemma 7 we can find our minimum cycle cover by applying a weighted matching algorithm to $2\mathcal{N}$. The system of radii formed by averaging the dual variables of the matching in $2\mathcal{N}$ will give us an optimal set of nonoverlapping balls, just as it did in $2K_n$. For, the averaged radii will achieve the optimal value and will have no overlaps that are detected by the edges of $\mathcal{N}$; by Lemma 8 there can be no other overlaps. It remains to show that we can find a matching (and its dual variables) quickly in this graph. For this we need some geometry, since in arbitrary metric spaces $\mathcal{N}$ can be complete.

The *ply* of a system of (overlapping) balls is the maximum number of balls that have a nonempty intersection. The *kissing number* of a space is the maximum number of unit balls that can touch a single unit ball; it is single-exponential in the dimension for Euclidean spaces. The following lemma is from [18, Lem. 3.2].

Lemma 9 *The nearest neighbor balls of a finite set of points in d-dimensional Euclidean space have ply at most the kissing number of the space.*

It follows that $\mathcal{N}$ is sparse, having only $O(n)$ edges. We can measure this more precisely by its *degeneracy*, the minimum number Δ such that the vertices of the graph can be ordered into a sequence with each vertex having at most Δ neighbors that are later than it in the sequence. The following lemma is from [18, Thm. 4.2].

Lemma 10 *The intersection graph of any system of balls with bounded ply has bounded degeneracy.*

This immediately gives a speedup from $O(n^3)$ to $O(n^2 \log n)$, by applying the $O(mn + n^2 \log n)$ time bound for matching on the sparse bipartite graph $2\mathcal{N}$ and using the degeneracy bound to substitute $m \leq n\Delta = O(n)$. But we can do better using more graph structure. Define a *separator* of an n-vertex graph to be a subset of the vertices the removal of which allows the graph to be partitioned into two subgraphs, disconnected from each other and each having at most sn vertices for a constant $s < 1$. The following result is from [8]:

Lemma 11 *The intersection graphs of systems of balls with bounded ply in Euclidean spaces of dimension d have separators of size $O(n^{1-1/d})$ that can be found deterministically in linear time from the balls. By recursively constructing a hierarchy of separators, the intersection graph of a system of n balls can be found from the balls in time $O(n \log n)$.*

Another way of expressing the existence of small separators (of size a fractional power of n) is that the intersection graphs of bounded-ply systems of balls, such as $\mathcal{N}$, are graphs of *polynomial expansion* [7, 15]. We remark that any hierarchy of separators for $\mathcal{N}$ can be transformed into a hierarchy of separators for $2\mathcal{N}$, simply by including both copies of each vertex of a separator for $\mathcal{N}$ in the corresponding separator for $2\mathcal{N}$.

An efficient algorithm for weighted matching in graphs with small separators is given by [14, Sec. 7]. They directly consider only planar graphs, but their method (if it works) would also work for other graphs that have separator hierarchies. However, they consider a different variant of matching, maximum weight matching on non-bipartite graphs, and are not explicit in how they maintain and update the analogue for that problem of the dual variables that we need. They also do not take advantage of more recent advances in shortest path algorithms for graphs with separators [11]. In an appendix we describe an algorithm based on similar ideas that maintains the dual variables that we need and is faster by a logarithmic factor, proving the following result:

Lemma 12 *Let G be a bipartite graph given together with a separator hierarchy in which each subgraph of p vertices has a separator of size $O(p^c)$, for some constant c with $0 < c < 1$. Then one can find both a minimum weight perfect matching of G, and the dual variables of the matching, in time $O(n^{1+c})$.*

Putting the results of this section together, we have our main theorem:

Theorem 13 *Given n points in a Euclidean space of constant dimension $d \geq 2$, we can construct a system of balls centered at the given points, with maximum sum of radii, in time $O(n^{2-1/d})$.*

Proof. We find the nearest neighbors of all points, construct graph $\mathcal{N}$ and its separator hierarchy, apply the matching algorithm of Lemma 12 to $2\mathcal{N}$ (using separators formed by doubling the separators in $\mathcal{N}$), and average the dual variables from the two copies of each input point to give a single radius for each point. □

References

[1] H. Alt, E. M. Arkin, H. Brönnimann, J. Erickson, S. P. Fekete, C. Knauer, J. Lenchner, J. S. B. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. *Proc. 22nd Symp. Computational Geometry (SoCG 2006)*, pp. 449–458, 2006, doi:10.1145/1137856.1137922, MR2389355.

[2] E. W. Chambers, S. P. Fekete, H.-F. Hoffmann, D. Marinakis, J. S. B. Mitchell, V. Srinivasan, U. Stege, and S. Whitesides. Connecting a set of circles with minimum sum of radii. *Proc. 12th Int. Symp. Algorithms and Data Structures (WADS 2011)*, pp. 183–194. Springer, LNCS 6844, 2011, doi:10.1007/978-3-642-22300-6_16, MR2863135.

[3] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. *Proc. 24th IEEE Symp. Foundations of Computer Science (FOCS '83)*, pp. 226–232, 1983, doi:10.1109/SFCS.1983.16.

[4] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. *SIAM J. Comput.* 23(6):1313–1347, 1994, doi:10.1137/S0097539791256325, MR1303338.

[5] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. Map labeling and its generalizations. *Proc. 8th ACM-SIAM Symp. on Discrete Algorithms (SODA '97)*, pp. 148–157, 1997, MR1447660.

[6] R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. *Proc. 23rd ACM-SIAM Symp. on Discrete Algorithms (SODA '12)*, pp. 1413–1424, 2012, MR3205301.

[7] Z. Dvořák and S. Norin. Strongly sublinear separators and polynomial expansion. Electronic preprint arxiv:1504.04821, 2015.

[8] D. Eppstein, G. L. Miller, and S.-H. Teng. A deterministic linear time algorithm for geometric separators and its applications. *Fundamenta Informaticae* 22(4):309–331, 1995, MR1360950.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34(3):596–615, 1987, doi:10.1145/28869.28874, MR904195.

[10] M. Gibson, G. Kanade, E. Krohn, I. A. Pirwani, and K. Varadarajan. On metric clustering to minimize the sum of radii. *Algorithmica* 57(3):484–498, 2010, doi:10.1007/s00453-009-9282-7, MR2609050.

[11] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. System Sci.* 55(1):3–23, 1997, doi:10.1006/jcss.1997.1493, MR1473046.

[12] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.* 23(6):1179–1192, 1994, doi:10.1137/S0097539793251876, MR1303329.

[13] M. Jiang, S. Bereg, Z. Qin, and B. Zhu. New bounds on map labeling with circular labels. *Proc. 15th Int. Symp. Algorithms and Computation (ISAAC 2004)*, pp. 606–617. Springer, LNCS 3341, 2004, doi:10.1007/978-3-540-30551-4_53, MR2158365.

[14] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.* 9(3), 1980, doi:10.1137/0209046, MR584516.

[15] J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms.* Algorithms and Combinatorics 28. Springer, 2012, doi:10.1007/978-3-642-27875-4, MR2920058.

[16] T. Strijk and A. Wolff. Labeling points with circles. *Int. J. Comput. Geom. Appl.* 11(02):181–195, 2001, doi:10.1142/s0218195901000444, MR1831031.

[17] R. E. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics 44. SIAM, 1983, doi:10.1137/1.9781611970265, MR826534.

[18] S.-H. Teng. *Points, spheres, and separators: a unified approach to graph partitioning.* Ph.D. thesis, Carnegie-Mellon Univ., School of Computer Science, 1991, MR2687483.

[19] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.* 4(2):101–115, 1989, doi:10.1007/BF02187718, MR973540.

[20] V. V. Vazirani. *Approximation Algorithms.* Springer, 2001, doi:10.1007/978-3-662-04565-7, MR1851303.

[21] F. Wagner and A. Wolff. A practical map labeling algorithm. *Comp. Geom. Th. & Appl.* 7(5-6):387–404, 1997, doi:10.1016/s0925-7721(96)00007-7, MR1447248.

On the Number of Forceless Hint Sequences in Paint-by-Numbers Puzzles

D. Berend[*][†] S. Zucker[‡]

Abstract

In this paper we consider a question raised in [4] regarding Paint-by-Numbers puzzles. *Paint-by-Numbers* is a classic logic puzzle in which the squares of a $p \times n$ grid are to be colored black or white, according to hints, consisting of the length of the blocks of consecutive black squares in each row and column.

Mullen [4] studied the asymptotic probability that a random hint sequence of a row will determine uniquely the color of at least some of the squares in a row, and was able to give lower and upper bounds on this probability. In this paper we tighten his bounds.

1 Introduction

Paint-by-Numbers is a classical logic puzzle, in which cells in a grid must be colored black or left uncolored (or, equivalently, colored white) according to numbers at the side of the grid. In this puzzle, a rectangular grid is given, with a finite sequence of numbers above each column and alongside each row. These numbers indicate the lengths of the blocks of consecutive black squares in the columns and rows, respectively. For example, the sequence 3,2 signifies that in this column (or row) there are three adjacent black squares, space of at least one uncolored square, and then two adjacent black squares. Note that there may be any number of uncolored squares before the first block of black squares and/or any number of uncolored squares after the last block. Thus, if the given sequence is of length k, then in addition to the blocks of black squares, there must be at least $k-1$ uncolored blocks in between, and perhaps one or two uncolored blocks on the sides.

Table 1 provides an example of a small solved puzzle. Initially, we note that the second row needs to contain 5 black squares, and has only 5 places. Therefore all squares in this row need to be colored black. This completes filling the second and fourth columns, which should contain only one black square each. Similarly, the third column should contain three black squares and has only three places. Therefore all squares of this column are colored black. This completes filling the third row. The first row should contain three black squares, with at least one uncolored square between each two. Thus there is only one way to color this row, and we are done.

	2	1	3	1	2
$1, 1, 1$	x		x		x
5	x	x	x	x	x
1			x		

Table 1: Example of a small puzzle

Assume, for example, that the given sequence for some row is $2, 1, 4$. In this case, if the length of the corresponding row is 9, then we know exactly how the squares are to be colored. If $n = 10$, then there are four possible configurations, corresponding to the possible locations of the "extra" uncolored square. However, in all these configurations, the squares in places $2, 7, 8, 9$ of the row are colored black. In order to color the remaining squares, we need to use data regarding the columns and the other rows. If $n \geq 13$, then we cannot deduce the color of any square without using data from the rest of the puzzle.

The puzzle was studied from several points of view. The decision problem of unique solvability of a puzzle is NP-complete [5]. In [1] it was shown that one can find all squares of a partially colored row that must be colored in some way (black or white), or that must be left uncolored, according to the hints regarding that row, in polynomial time. In [3] it was shown that, for a large randomly colored puzzle, the probability that even the color of a single square will be uniquely determined, by the hints relating to the corresponding row or column only, is very small. (However, when we do consider all hints simultaneously, the probability of having some squares with a uniquely determined color is bounded from 0.) It was also proved that such a puzzle usually has a huge number of solutions. In [2], the number of possible puzzles was studied, if, instead of recording the length of all blocks of black squares in all rows and columns, one records only the total number of black squares in each row and column.

When solving a paint-by-numbers puzzle, it is natural to start by looking for rows (and columns) whose hint sequences determine uniquely the color of some of the squares along the row, without resort to data

[*]Departments of Mathematics and Computer Science, Ben-Gurion University, `berend@math.bgu.ac.il`

[†]Research supported in part by the Milken Families Foundation Chair in Mathematics

[‡]Department of Computer Science, Sapir Academic College, `zucker.shira@gmail.com`

from the rest of the puzzle. Thus, it is natural to study the number of hint sequences, for rows of any length n, that provide immediate information on at least some of the squares. It follows from [4] that most hint sequences satisfy this property. Hence it will be more convenient to study the complement, namely the number of sequences that give no such information – *forceless sequences.* In [4], Mullen studied the asymptotic of this number as a function of the row length. More precisely, he first observed that the total number of hint sequences, for a length-n row, is F_{n+2}, where $(F_n)_{n=1}^{\infty}$ is the Fibonacci sequence, defined by:

$$\begin{aligned} &F_1 = F_2 = 1, \\ &F_n = F_{n-1} + F_{n-2}, \qquad n \geq 3. \end{aligned}$$

Note that there are obviously 2^n ways to color a length-n row, but we care only about hint sequences. As many distinctly colored rows give rise to the same hint sequence, the total number of hint sequences is much smaller than 2^n. Recall that F_n behaves asymptotically as $\phi^n/\sqrt{5}$, where $\phi = \left(1+\sqrt{5}\right)/2$ is the golden ratio. Denoting by G_n the number of forceless sequences for rows of length n, Mullen [4] proved

Theorem A *For appropriate constants $c_1, c_2 > 0$,*

$$G_n \geq c_1 \cdot \frac{\phi^n}{n} \tag{1}$$

and

$$G_n \leq c_2 \cdot \frac{\phi^n \ln n}{n}, \tag{2}$$

for all $n \geq 2$.

In this paper we show that the right-hand side of (1) provides the correct order of magnitude of G_n. In Section 2 we state this result formally, and in Section 3 – prove it.

2 The Main Result

Our main result, employing the notations in the preceding section, is

Theorem 1 *There exist two constants $C_1, C_2 > 0$ such that*

$$C_1 \cdot \frac{\phi^n}{n} \leq G_n \leq C_2 \cdot \frac{\phi^n}{n}, \quad n \geq 1.$$

Theorem 1 can be interpreted as giving an asymptotic estimate of the probability of a random hint sequence being forceless. In fact, as the total number of hint sequences is F_{n+2}, this probability is G_n/F_{n+2}. Since $F_{n+2} \approx \phi^{n+2}/\sqrt{5}$, the probability in question is approximately $\frac{G_n\sqrt{5}}{\phi^{n+2}}$, which by the theorem is $\Theta(\frac{1}{n})$.

In Table 2 we provide the value of G_n/F_{n+2}, normalized by multiplying it by n, for several n's up to 6000. (The computations were carried out using (3) and (4) below.) According to Theorem 1, these values are bounded. A quick glance at Table 2 may lead one to conjecture that nG_n/F_{n+2} increases to some limit as $n \to \infty$. However, we believe that the sequence oscillates between two very close values, but does not actually converge.

n	G_n	nG_n/F_{n+2}
10	33	2.2917
20	2258	2.5498
50	$1.8 \cdot 10^{9}$	2.7331
100	$2.6 \cdot 10^{19}$	2.8007
200	$1.0 \cdot 10^{40}$	2.8358
500	$2.1 \cdot 10^{102}$	2.8573
1000	$3.3 \cdot 10^{202}$	2.8646
2000	$1.6 \cdot 10^{415}$	2.8682
3000	$1.0 \cdot 10^{624}$	2.8694
4000	$7.5 \cdot 10^{832}$	2.8700
5000	$5.8 \cdot 10^{1041}$	2.8704
6000	$4.7 \cdot 10^{1250}$	2.8706

Table 2: The probability of being forceless for some small n's

3 Proof of Theorem 1

Following [4], let the *norm* of a hint sequence be the maximal number in that sequence, i.e., the maximal length of a block of squares to be colored black in a row. The *length* of a hint sequence is the fewest number of columns needed for that sequence to fit in a puzzle. (For example, the norm of the hint sequence $3, 10, 5, 10, 2$ is 10 and its length is 34.) Let $A_{m,k}$ denote the number of hint sequences of length m with norm at most k.

By [4, p.6],

$$A_{m,k} = \begin{cases} F_m, & m \leq k, \\ \sum_{j=m-k-1}^{m-2} A_{j,k}, & m > k. \end{cases} \tag{3}$$

To count all forceless sequences, we need only sum $A_{m,n-m}$ over all possible lengths [4], that is

$$G_n = \sum_{m=0}^{n-1} A_{m,n-m}. \tag{4}$$

For a fixed k, let ϕ_k be the largest root of the characteristic polynomial of $(A_{m,k})_{m=1}^{\infty}$, namely $x^{k+1} - x^{k-1} - x^{k-2} - ... - 1$. Mullen [4] noted that this is in fact the only positive root larger than 1 .

We will first establish an upper bound on $A_{m,k}$.

Proposition 2 $A_{m,k} \leq 2\phi_k^m$ *for every $m, k \geq 1$.*

Proof. For $m \leq k$, by (3) and the definition of ϕ, we know that $A_{m,k} = F_m \leq \phi^m$. Now, [4, Prop. 5.3] gives

$$\phi \leq \phi_k \left(1 + \frac{1}{\sqrt{5}\phi^{k+1} - (k+2)}\right),$$

which implies

$$\phi \le \phi_k \left(1 + \frac{1}{2\phi^{k+1}}\right), \qquad k \ge 8.$$

Thus,

$$\begin{aligned} \phi^m &\le \phi_k^m \left(1 + \tfrac{1}{2\phi^{k+1}}\right)^m \\ &\le \phi_k^m \left(1 + \tfrac{1}{2k}\right)^k \\ &\le \phi_k^m \cdot \sqrt{e} \\ &\le 2\phi_k^m, \qquad m \le k. \end{aligned} \tag{5}$$

For $m > k$ we proceed by induction. If $m = k+1$ then

$$A_{m,k} = \sum_{j=k+1-k-1}^{k+1-2} A_{j,k} = \sum_{j=0}^{k-1} A_{j,k} \le 2\sum_{j=0}^{k-1} \phi_k^j.$$

Since ϕ_k is a root of the polynomial $x^{k+1} - x^{k-1} - x^{k-2} - ... - 1$, we have $\sum_{j=0}^{k-1} \phi_k^j = \phi_k^{k+1}$ and therefore

$$A_{m,k} \le 2\phi_k^{k+1} = 2\phi_k^m,$$

as required.

Now suppose that the inequality holds for each m' such that $m' < m$. By the induction hypothesis:

$$\begin{aligned} A_{m,k} &= \textstyle\sum_{j=m-k-1}^{m-2} A_{j,k} \le \sum_{j=m-k-1}^{m-2} 2\phi_k^j \\ &= 2\phi_k^{m-k-1} \cdot \textstyle\sum_{j=0}^{k-1} \phi_k^j = 2\phi_k^{m-k-1} \cdot \phi_k^{k+1} \\ &= 2\phi_k^m. \end{aligned} \tag{6}$$

This proves the proposition. □

Lemma 3 *For each $k \ge 1$*

$$\phi_k \le \phi - \frac{1}{\sqrt{5}\phi^k}. \tag{7}$$

Proof. Multiplying the characteristic polynomial of $A_{m,k}$ (for a fixed k) by $x - 1$, we obtain the polynomial $P_k = x^{k+2} - x^{k+1} - x^k + 1$, of which ϕ_k is also a root. Recall that ϕ_k is the only real zero greater than 1 of this polynomial.

If we calculate P_k at the point $\phi - \frac{L}{\phi^k}$ instead of ϕ_k, where $L > 0$, and recall that $\phi^2 = \phi + 1$, we obtain

$$\begin{aligned} P_k\left(\phi - \tfrac{L}{\phi^k}\right) &= (\phi - \tfrac{L}{\phi^k})^{k+2} - (\phi - \tfrac{L}{\phi^k})^{k+1} \\ &\quad -(\phi - \tfrac{L}{\phi^k})^k + 1 \\ &= (\phi - \tfrac{L}{\phi^k})^k \cdot [(\phi - \tfrac{L}{\phi^k})^2 - (\phi - \tfrac{L}{\phi^k}) - 1] \\ &\quad +1 \\ &= (\phi - \tfrac{L}{\phi^k})^k \cdot [\phi^2 - \phi - 1 \\ &\qquad + \tfrac{L}{\phi^k} - \tfrac{2L}{\phi^{k-1}} + \tfrac{L^2}{\phi^{2k}}] + 1 \\ &= (1 - \tfrac{L}{\phi^{k+1}})^k \cdot L(1 - 2\phi + \tfrac{L}{\phi^k}) + 1 \\ &= 1 - L(2\phi - 1 - \tfrac{L}{\phi^k})(1 - \tfrac{L}{\phi^{k+1}})^k \\ &= 1 - L(\sqrt{5} - \tfrac{L}{\phi^k})(1 - \tfrac{L}{\phi^{k+1}})^k, \end{aligned} \tag{8}$$

which is positive for $L = \frac{1}{\sqrt{5}}$. Since ϕ_k is a root of P_k, this completes the proof of the lemma. □

3.1 Conclusion of the Proof of Theorem 1

By Proposition 2 and Lemma 3:

$$A_{m,k} \le 2\phi_k^m \le 2(\phi - \frac{1}{\sqrt{5}\phi^k})^m, \qquad m, k \ge 1.$$

In particular,

$$A_{m,n-m} \le 2(\phi - \frac{1}{\sqrt{5}\phi^{n-m}})^m = 2\phi^m(1 - \frac{1}{\sqrt{5}\phi^{n-m+1}})^m,$$

where $1 \le m \le n$.

As will become evident in the sequel, for a fixed large n, the significant terms $A_{m,n-m}$ on the right-hand side of 4, are those with m close to $n - \lg_\phi n$, while the other terms are relatively negligible. Hence we will write m in the form $m = n - \lg_\phi n + d$, where d varies between approximately $-n + \lg_\phi n$ and $\lg_\phi n$. (Note that d is non-integer.) We have

$$\begin{aligned} &\phi^m(1 - \tfrac{1}{\sqrt{5}\phi^{n-m+1}})^m \\ &\quad = \phi^{n-\lg_\phi n + d} \cdot \left(1 - \frac{1}{\sqrt{5}\phi^{\lg_\phi n - d + 1}}\right)^m \\ &\quad = \frac{\phi^n}{\phi^{\lg_\phi n}} \cdot \phi^d \cdot \left(1 - \frac{1}{\sqrt{5}n/\phi^{d-1}}\right)^m \\ &\quad = \frac{\phi^n}{n} \cdot \phi^d \cdot \left(1 - \frac{1}{\sqrt{5}n/\phi^{d-1}}\right)^{\frac{\sqrt{5}n}{\phi^{d-1}} \cdot \frac{\phi^{d-1}}{\sqrt{5}n} \cdot m}. \end{aligned} \tag{9}$$

For $d \ge 0$ (and n sufficiently large) this gives:

$$\phi^m \left(1 - \frac{1}{\sqrt{5}\phi^{n-m+1}}\right)^m \le \frac{\phi^n}{n} \cdot \frac{\phi^d}{e^{\phi^{d-1}/3}}.$$

Hence:

$$\sum_{m \ge n - \log_\phi n} A_{m,n-m} \le \frac{\phi^n}{n} \cdot 2\sum_{d=0}^{\infty} \frac{\phi^{d+1}}{e^{\phi^{d-1}/3}}.$$

As the series on the right-hand side converges, this means:

$$\sum_{m \ge n - \log_\phi n} A_{m,n-m} = O\left(\frac{\phi^n}{n}\right). \tag{10}$$

For $d < 0$ we obtain by (9)

$$\phi^m \left(1 - \frac{1}{\sqrt{5}\phi^{n-m+1}}\right) \le \frac{\phi^n}{n} \cdot \phi^d,$$

so that

$$\sum_{m < n - \log_\phi n} A_{m,n-m} \le \frac{\phi^n}{n} \cdot \sum_{d=-\infty}^{0} \phi^d = O\left(\frac{\phi^n}{n}\right). \tag{11}$$

According to (4), (10) and (11):

$$G_n = O\left(\frac{\phi^n}{n}\right).$$

Together with Theorem A, this completes the proof.

References

[1] Batenburg, K. J., and Kosters, W. A., Solving Nonograms by combining relaxations, *Pattern Recognition* 42:1672–1683, 2009.

[2] Benton, J., Snow, R., and Wallach, N., A combinatorial problem associated with nonograms, *Linear Algebra Appl.* 412:30–38, 2006.

[3] Berend, D., Pomeranz, D., Rabani, R. and Raziel, B., Nonograms: Combinatorial questions and algorithms, *Discrete Applied Mathematics* 169:30–42, 2014.

[4] Mullen, R., On determining paint-by-numbers puzzles with nonunique solutions. *J. Integer Seq.* 12, no. 6, Article 09.6.5, 19 pp, 2009.

[5] Ueda, N., and Nagao, T., NP-completeness results for Nonogram via Parsimonious Reductions, Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996.

An upper bound of 84 for Morpion Solitaire 5D

Henryk Michalewski[*] Andrzej Nagórko[†] Jakub Pawlewicz[‡]

Abstract

The Morpion Solitaire is a paper-and-pencil single-player game played on a square grid with initial position consisting of 36 dots. In each move the player puts a dot on an unused grid position and draws a line that consists of four consecutive segments passing through the dot. The goal is to find the longest possible sequence of moves. There are two main variants of the game: 5T and 5D. They have different restrictions on how the moves may intersect.

Providing lower and upper bounds in Morpion Solitaire is a significant computational and mathematical challenge that led to a discovery of a Nested Rollout Policy Adaptation algorithm [10]. Best lower bounds for Morpion Solitaire were found using this algorithm in 2011 by Rosin [10]: 178 moves for the 5T variant, 82 moves for the 5D variant.

In the Morpion 5D variant Kawamura et al. proved in [6] an upper bound of 121 moves using a geometrical argument. We improve this bound to 84. This is done in two steps. 1) We state, using linear constraints, a geometric property that defines a class of graphs that includes all Morpion positions — finding practically solvable linear constraints is a main conceptual advance of the present paper. 2) We solve the mixed integer problems and obtain an upper bound of 84 for Morpion 5D. The same method fully solves Morpion 5D with center symmetry — the optimal sequence is 68 moves long.

1 Introduction

The Morpion Solitaire is a paper-and-pencil single-player game played on a square grid with the initial configuration of 36 dots forming a cross depicted in Figure 1. In each move the player puts a dot on an unused grid position and draws a line that consists of four consecutive segments passing through the dot. The line must be horizontal, vertical or diagonal. The goal is to find the longest possible sequence of moves. There are two main variants of the game: 5T and 5D. They have different restrictions on how the moves may intersect. In the 5T variant of the game, no segment may be drawn twice, i.e. the moves have to be segment-disjoint. In the 5D variant of the game, any two moves in the same direction have to be dot-disjoint. The difference is demonstrated in Figure 1.

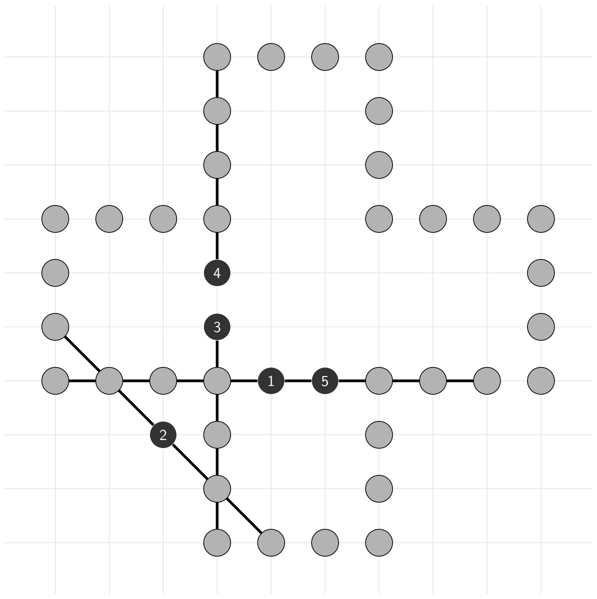


Figure 1: An example of a position of the Morpion Solitaire. The first 4 moves are legal both in Morpion 5D and Morpion 5T variants, but the 5–th move is legal only in Morpion 5T.

The problem of finding the longest sequence of moves in the Morpion Solitaire is notoriously hard for computers. For 34 years the longest known sequence in the Morpion 5T game was one of 170 moves discovered by Bruneau in 1976. In Morpion 5D a record of 68 was found by Langerman in 1999, later improved by Cazenave [2] to 78, and by Hyyrö and Poranen [5] to 79[1]. In 2010 Rosin [10] obtained the current world records of 82 moves in Morpion 5D and of 178 moves in Morpion 5T using a specialized Monte Carlo algorithm. This work was recognized as a best paper of the IJCAI conference in 2011. The webpage [1] maintained by Boyer contains an extensive and up-to-date information about records in all Morpion Solitaire variants. The Monte Carlo approach, which proved to be very effective in finding the lower bounds, is not very useful in establishing upper bounds for the Morpion Solitaire. The linear programming methods presented in this paper and in [8] seem to be the strongest tools for proving the upper bounds in Morpion 5D and Morpion 5T.

[*]University of Warsaw, H.Michalewski@mimuw.edu.pl
[†]University of Warsaw, A.Nagorko@mimuw.edu.pl
[‡]University of Warsaw, J.Pawlewicz@mimuw.edu.pl

[1]See the webpage [7] for more details.

1.1 Morpion Solitaire and linear programming

The rules of Morpion Solitaire do not limit the size of the grid on which the game is played, hence a priori it is not clear if the sequences have to be bounded. However, as a single player game, Morpion Solitaire can be encoded in a natural way as a linear optimization problem with the optimization target being the length of the sequence. On its own, the encoding is not very helpful, because the problem is too large to be practically solvable. Nevertheless, this inspires a natural approach towards construction of upper bounds. Instead of solving the Morpion Solitaire, we solve a more general game such that

- every gameplay of ordinary Morpion Solitaire is a gameplay of the generalization,
- the generalization is practically solvable and
- an upper bound proved for this more general game is still interesting for the Morpion Solitaire.

We do not expect that the more general game will be playable by humans. Here is a summary of results we managed to obtain using this approach:

Morpion 5D. A geometric analysis of Morpion 5D led in [6] to an upper bound of 121 which improved on an earlier bound of 144 found in [3] using the potential method; considering a generalization of Morpion 5D, in this paper we prove an upper bound of 84 on the length of Morpion 5D sequences.

Morpion 5T. The bound of 705 in Morpion 5T was proved in 2006 in [3] using a careful geometric analysis of moves in Morpion 5T and an argument based on the potential method. The current best bound of 485 in Morpion 5T was obtained in [8] using an appropriate generalization of Morpion 5T. In particular, in [8] it was shown that the bound of 586 can be obtained via the relaxation of the linear program encoding the rules of Morpion 5T; the `github` repository [9] contains the code used in the work [8] and in this article, as well as the LaTeX source files of the article [8]. The main difference between the approach presented here and in [8] consists in the fact, that proving the bound for Morpion 5D in this paper we use the order of moves – this information is ignored in [8]. Moreover, an isoperimetric inequality used in [8] does not provide strong enough information to obtain the 84 bound in Morpion 5D. Instead of the isoperimetric method, in this paper we use the resizing technic described in Section 4. For a more technical comparison of the method used in [8] and in this paper, see also Remark 1 in Appendix A.

1.2 Organization of the paper

In Section 2 we formulate the main results:

- in Theorem 1 we show that every Morpion 5D gameplay must be contained in a relatively small rectangular board around the initial cross in Figure 1; this already proves the bound of 85 in Morpion 5D,
- in Corollary 2 we improve this bound to 84 through an analysis of 3 special cases.

The proof of these results is divided between next four sections. In Section 2 we introduce basic notions and formulate the main theorem. In Section 3 we define a useful generalization of Morpion 5D and encode it as a linear program. In Section 4 we generate a list of rectangular boards ("boxes") such that any position of Morpion 5D is contained in one of the boxes — this is achieved using an auxiliary linear program. In Section 5 we solve the generalization of Morpion 5D on all boards found in Section 4 and conclude the proof of Theorem 1 along with Corollary 2. Moreover, we apply the same approach to the symmetric variant of Morpion 5D and solve completely this game.

2 Geometry of Morpion Solitaire positions

In this section we collect a number of observations regarding geometric properties of Morpion Solitaire positions and formulate the basic definitions as well as the main result of this paper. In order to make linear computations feasible, we will limit the size of boards to rectangular "boxes". In Section 4 we explain why this is enough to solve the general problem of bounding the sequences in Morpion 5D.

Definition 1. *A box is a rectangular subset of $\mathbb{Z}^2$ bounded by 4 edges parallel to the coordinate axes. A bounding box of a Morpion 5D position is the smallest box containing all dots of this position.*

Every Morpion 5D graph contains the initial cross, hence the smallest bounding box has dimensions 9×9. The graph depicted in Figure 2 has a bounding box with dimensions 14×13. However, the dimensions of the bounding box do not give information about the position of the initial cross inside. Since this information is important for computations, we introduce the following

Notation. The distance of a bounding box to the initial cross can be described by 4 numbers: the distances of the sides of the box to the edges of the cross. For the graph depicted in Figure 2, the distance from top edge of the cross to the top side of the bounding box is equal to 3. For the right side it is 4, for the bottom side it is 1 and for the left side it is also 1. The bounding box of the position in Figure 2 is denoted as $(3, 4, 1, 1)$.

Below we formulate the key theorem in this paper (required definitions are formulated in Section 3.1):

No	BBox	Size
1	(4, 3, 1, 1)	85
2	(4, 3, 1, 2)	85
3	(4, 3, 1, 3)	85
4	(4, 2, 1, 2)	84
5	(4, 2, 2, 2)	84
6	(5, 2, 2, 1)	84
7	(5, 2, 1, 2)	84
8	(5, 2, 2, 2)	84
9	(3, 3, 2, 1)	84
10	(3, 3, 2, 2)	84
11	(4, 3, 2, 1)	84
12	(4, 3, 3, 1)	84
13	(4, 3, 2, 2)	84
14	(4, 3, 2, 3)	84
15	(4, 3, 0, 2)	84
16	(3, 2, 1, 2)	83
17	(3, 2, 2, 2)	83
18	(5, 2, 1, 1)	83
19	(3, 3, 3, 1)	83
20	(4, 3, 3, 2)	83
21	(5, 3, 1, 1)	83
22	(5, 3, 1, 2)	83
23	(4, 3, 0, 3)	83
24	(4, 4, 1, 0)	83
25	(4, 4, 2, 0)	83
26	(4, 4, 1, 1)	83
27	(4, 4, 2, 1)	83
28	(4, 4, 3, 1)	83

Table 1: Bounding boxes mentioned in Theorem 1.1 that admit Morpion 5D graphs of sizes 85, 84 *and* 83.

Theorem 1. *1. Up to a symmetry every Morpion 5D* `position` *is contained in one of the following boxes:* $(4,3,3,3)$, $(4,4,3,2)$, $(5,3,2,3)$, $(4,3,4,2)$, $(5,3,3,2)$, $(5,4,2,1)$, $(6,2,2,1)$, $(5,4,0,2)$, $(5,1,4,0)$.

2. Every box contained in one of the nine boxes listed in 1, with the exception of boxes $(6,2,2,0)$ *and* $(6,0,2,0)$, *is a bounding box of a Morpion 5D* `graph`.

3. For each box $\mathcal{B}$ *described in 2, the size of a maximal Morpion 5D graph with the bounding box* $\mathcal{B}$ *does not exceed* 85. *Bounding boxes that admit Morpion 5D graphs of sizes greater than* 82 *are listed in Table 1.*

We will present a proof of this theorem in the next three sections with a summary of the argument in Subsection 5.1. In Section 3 we precisely define the notion of a Morpion 5D graph. Intuitively, Morpion 5D graphs are obtained from Morpion 5D positions by forgetting about the order in which the moves were played.

Corollary 2. *The longest sequence of moves in Morpion 5D does not exceed* 84.

Proof. In Table 1 there are four bounding boxes with maximal graphs of size 85. For these boxes we formulate mixed integer problems with additional constraints that force the graph to be a Morpion 5D position. These problems are much harder to solve, but with correct choice of solver optimization parameters, we are able to show that there are no solutions of these problems of size 85. Equipped with additional definitions and lemmas formulated later in this paper, in Subsection 5.1 we add some technical information to this proof. ☐

3 Morpion Solitaire via Linear Programming

Let `Cross` $\subset \mathbb{Z}^2$ be a set of 36 dots that form an initial cross of Morpion Solitaire. Let $\mathcal{D} = \{(1,0),(1,1),(0,1),(-1,1)\} \subset \mathbb{Z}^2$. We call elements of $\mathcal{D}$ *directions*. An unordered pair $\{u,v\} \subset \mathbb{Z}^2$ is a *unit edge* if $u-v \in \mathcal{D}$ or $v-u \in \mathcal{D}$. We call $G=(V,E)$ a *lattice graph* if $V \subset \mathbb{Z}^2$ and each edge of G is a unit edge. A *move* m with a starting vertex $v \in \mathbb{Z}^2$ and direction $d \in \mathcal{D}$ is the set $m = \{v + n\cdot d \colon n \in \{0,1,2,3,4\}\}$. The set of all moves is denoted by $\mathcal{M}$. We let $E(m)$ denote the set $\{\{v+n\cdot d, v+(n+1)\cdot d\} \colon n \in \{0,1,2,3\}\}$. We say that moves m_1, m_2 are *parallel* if they have the same direction d. We use the notation $m_1 \parallel m_2$, $m_i \parallel d$. We let $\mathcal{M}(G) = \{m \in \mathcal{M} \colon E(m) \subset E\}$. We call $\mathcal{M}(G)$ the set of *moves in G*.

3.1 Definition of Morpion 5D graphs

Below we define a concept of a marked Morpion 5D graph, which is a lattice graph with a special marking of vertices. Every Morpion 5D position is a marked Morpion 5D graph, but as we discuss below, there are marked Morpion 5D graphs not related to any Morpion 5D position. In the following four definitions $G=(V,E)$ is a fixed graph.

Definition 2. *We say that the set* $\mathcal{M}(G)$ *of moves in* G *is* 5D-disjoint *if for every parallel moves* $m_1, m_2 \in \mathcal{M}(G)$ *the intersection of* m_1 *with* m_2 *is empty, i.e. parallel moves are vertex disjoint.*

Definition 3. *We say that the set* $\mathcal{M}(G)$ *of moves in* G covers E *if* $\bigcup_{m\in\mathcal{M}(G)} E(m) = E$.

Definition 4. *We say that the set* $\mathcal{M}(G)$ *of moves in* G *admits a* Morpion marking *if there exists a bijective mapping* $d \colon \mathcal{M}(G) \to V \setminus$ `Cross` *such that* $d(m) \in m$.

Definition 5. *A lattice graph* G *is a* Morpion 5D graph *if* `Cross` $\subset V$, $\mathcal{M}(G)$ *covers* E *and* $\mathcal{M}(G)$ *is 5D-disjoint. If* d *is a selected Morpion marking of* G, *then we say that* G *together with* d *is a* marked Morpion 5D graph *(see Figure 2). The* size *of Morpion 5D graph is defined as the number of vertices minus* 36, *that is minus the number of vertices in the* `Cross`.

For comments and figures related to this definition see Appendix A.

3.2 Definition of mixed integer programming problem

Let $\mathcal{B}$ be a box that contains the `Cross`. We define a mixed integer programming problem whose solutions correspond to all marked Morpion 5D graphs with a bounding box $\mathcal{B}$. The correspondence is formalized in Lemmas 3 and 4.

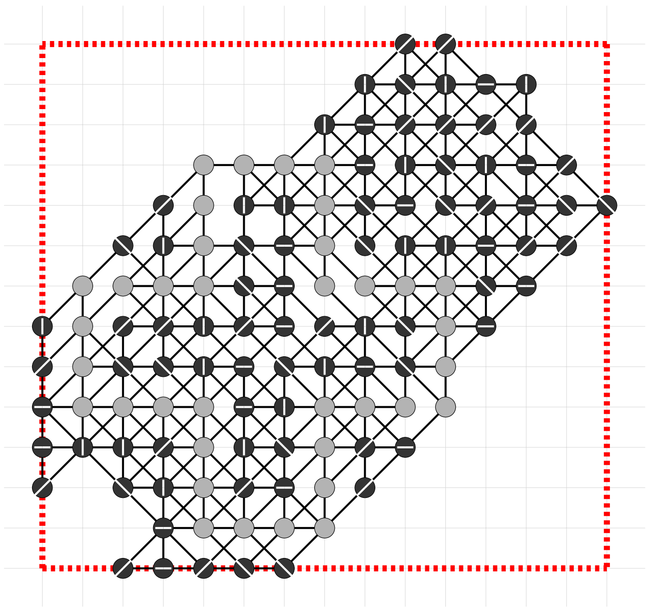


Figure 2: A marked Morpion 5D graph G_{85} of size 85 with its bounding box $(3, 4, 1, 1)$ (see Definition 5). Every vertex is labelled with one of the four directions. For every move m there exists exactly one vertex labelled with direction of m. This allows us to decode the selected marking of the graph.

Definition 6. *Let $B = (V_B, E_B)$ be a lattice graph with a vertex set $V_B = \mathcal{B}$ and E_B the set of all unit edges between vertices in $\mathcal{B}$. Let $\{\mathrm{dt}_v : v \in V_B\} \cup \{\mathrm{mv}_{m,v} : m \in \mathcal{M}(B), v \in m\}$ be a set of binary variables (i.e. variables assuming values $0, 1$).*

Below we define linear constraints L_1 - L_7. Every constraint is accompanied by an explanation that refers to a Morpion 5D graph G with marking d constructed in Lemma 3.

Constraints characterizing marked Morpion 5D graphs

L_1. For each $v \in \texttt{Cross}$, $\mathrm{dt}_v = 1$.

`Constraint L1 enforces that vertices of the Cross are vertices of` G.

L_2. For each $m \in \mathcal{M}(B)$ and each $v \in m$ such that $v \in \texttt{Cross}$, $\mathrm{mv}_{m,v} = 0$.

`Constraint L2 enforces that no vertices of the Cross are marked by` d.

L_3. For each $v \in V_B \setminus \texttt{Cross}$, $\mathrm{dt}_v = \sum_{m \in \mathcal{M}(B):\, v \in m} \mathrm{mv}_{m,v}$.

`Constraint L3 enforces that vertices of` G `that are not in the Cross are marked by exactly one move.`

L_4. For each $v \in V_B$ and each direction $d \in \mathcal{D}$,

$$\mathrm{dt}_v \geq \sum_{m \in \mathcal{M}(B):\, m \| d, v \in m} \sum_{w \in m} \mathrm{mv}_{m,w} .$$

`Constraint L4 enforces that parallel moves are vertex disjoint and that every move in` G `is placed on vertices of the graph.`

L_5. Each box $\mathcal{B}$ has four sides: the top, right, bottom and left edges. Each side we consider as a set of vertices and for each side S of $\mathcal{B}$ we require $\sum_{v \in S} \mathrm{dt}_v \geq 1$.

`Constraint L5 enforces that` $\mathcal{B}$ `is the bounding box of` G.

Additional constraint for the symmetric games

L_6. For each $m \in \mathcal{M}(B)$ and every $v \in m$, if m_s, v_s are the vertex and move symmetric to m, v with respect to the center of the `Cross`, then $\mathrm{mv}_{m,v} = \mathrm{mv}_{m_s,v_s}$. `Constraint L6 enforces center symmetry of` G.

Additional constraint enforcing that G is a Morpion 5D position In this constraint we consider continuous variables $\{\mathrm{ord}_v : v \in V_B\}$:

L_7. For every $m \in \mathcal{M}(B)$, every $w, v \in m$, $w \neq v$,

$$\mathrm{ord}_v \geq \mathrm{ord}_w + 1 - 121(1 - \mathrm{mv}_{m,v}).$$

`Constraint L7 enforces an order on the moves.`

Lemma 3. *Let $\mathcal{B}$ be a box and let dt_v and $\mathrm{mv}_{m,v}$ be a set of binary variables defined for $\mathcal{B}$. Assume that the variables satisfy conditions L_1 - L_5 of Definition 6. Let $V = \{v \in V_B : \mathrm{dt}_v = 1\}$ and*

$$E = \{e \in E_B : \exists_{m \in \mathcal{M}(B)} \exists_{v \in m}\, \mathrm{mv}_{m,v} = 1 \wedge e \in E(m)\},$$

and $G = (V, E)$. Then G is a Morpion 5D graph with a bounding box $\mathcal{B}$ and G admits a Morpion marking d.

- *If additionally the variables satisfy condition L_6, then G is center-symmetric.*
- *If additionally the variables satisfy condition L_7, then G is a Morpion 5D position, i.e. there exists an actual Morpion 5D gameplay such that G is a Morpion 5D graph corresponding to this gameplay.*

Proof of this lemma is provided in Appendix B.

Lemma 4. *Let $G = (V, E)$ be a Morpion 5D graph with marking $d \colon \mathcal{M}(G) \to V \setminus \texttt{Cross}$ and a bounding box $\mathcal{B}$. Let $\mathrm{dt}_v = \begin{cases} 1 & \text{if } v \in V \\ 0 & \text{otherwise} \end{cases}$ and*

$$\mathrm{mv}_{m,v} = \begin{cases} 1 & \text{if } m \in \mathcal{M}(G) \text{ and } v = d(m) \\ 0 & \text{otherwise} \end{cases}$$

Then the set of variables dt_v, $\mathrm{mv}_{m,v}$ satisfies conditions L_1 - L_5 of Definition 6. Moreover, if G with a marking d is symmetric, then the condition L_6 is satisfied. If G is a Morpion 5D position, then we define ord_v as the number of move when v was placed and L_7 is satisfied.

Proof of this lemma is provided in Appendix C.

4 Resizing

In this section we describe an algorithm that uses mixed integer programming problems defined in Section 3 to find a family boxes that contain all Morpion 5D *positions*. In particular, as announced in Theorem 1.1, this algorithm shows that up to a symmetry every Morpion 5D *position* is contained in one of the following boxes: $(4,3,3,3)$, $(4,4,3,2)$, $(5,3,2,3)$, $(4,3,4,2)$, $(5,3,3,2)$, $(5,4,2,1)$, $(6,2,2,1)$, $(5,4,0,2)$, $(5,1,4,0)$.

4.1 Symmetries

In order to control the number of involved cases we consider bounding boxes up to symmetries. There are 8 isometries of $\mathbb{Z}^2$ that leave `Cross` in place: 1) identity 2) horizontal, vertical and two diagonal reflections, 3) rotations by 90°, 180° and 270°, centered at the center of the `Cross`.

Lemma 5. *Up to the* 8 *isometries that leave* `Cross` *in place, every box is symmetric to a box* (a,b,c,d) *such that* a *is the maximum of* a,b,c,d, $d \le b$, *and if* $a = b$, *then* $d \le c$.

Proof. Let us recall that a,b,c,d are responsible for the top, left, bottom and right dimensions of the bounding box. We have to prove that every box (a,b,c,d) through symmetries can be turned into one of the above boxes.

First we make sure that a is the maximum of a,b,c,d via rotations. In order to satisfy the requirement $d \le b$ we apply a symmetry with respect to the vertical axis. Assume now that $a = b$. In order to satisfy the requirements $d \le c$ we apply (when needed) a diagonal reflection that interchanges a with b and c with d. □

4.2 Feasible and infeasible boxes

Let $\mathcal{B}$ be a box. By Lemma 3, solutions of mixed integer programming problem defined in Section 3 correspond to marked Morpion 5D graphs with bounding box $\mathcal{B}$. Let

$$\mathrm{Obj} = \sum_{m \in \mathcal{M}(\mathcal{B})} \sum_{v \in m} \mathrm{mv}_{m,v}$$

be an objective function for this problem. The value of Obj is equal to the number of moves in the Morpion 5D graph corresponding to the solution, i.e. it is equal to the size of this graph. Hence the maximal value of Obj is equal to the maximal size of Morpion 5D graph with bounding box $\mathcal{B}$. If the model is infeasible, then there is no Morpion 5D graph that has bounding box $\mathcal{B}$ and we call box $\mathcal{B}$ *infeasible.* Otherwise we say that $\mathcal{B}$ is *feasible.*

4.3 The resizing process

Definition 7. *A box* resized *from box* (a,b,c,d) *is one of the following boxes*

$$(a{+}1,b,c,d),(a{+}1,b{+}1,c,d),(a,b{+}1,c,d),(a,b{+}1,c{+}1,d),$$

$$(a,b,c{+}1,d),(a,b,c{+}1,d{+}1),(a,b,c,d{+}1),(a{+}1,b,c,d{+}1).$$

Lemma 6. *If* $\mathcal{B}$ *is a bounding box of a Morpion 5D position, then there exists a sequence of feasible boxes* $\mathcal{B}_0, \mathcal{B}_1, \dots, \mathcal{B}_n$ *such that* $\mathcal{B}_0 = (0,0,0,0)$, $\mathcal{B}_n = \mathcal{B}$ *and* $\mathcal{B}_{k+1}$ *is resized from* $\mathcal{B}_k$.

Even though the Lemma is obvious, its distinguishing property is that it uses the key feature of Morpion 5D positions, namely the order in which moves are played.

Proof. Let G be a Morpion 5D position and let G_k be a Morpion 5D position obtained after first k moves of the game. Let $\widetilde{\mathcal{B}}_k$ be a bounding box of G_k. Observe that G_{k+1} contains one more vertex than G_k and this vertex is at the unit distance from G_k. Therefore either $\widetilde{\mathcal{B}}_{k+1}$ is equal to $\widetilde{\mathcal{B}}_k$ or it is resized from $\widetilde{\mathcal{B}}_k$. Take a subsequence of $\widetilde{\mathcal{B}}_n$ with removed duplicates as the sequence $\mathcal{B}_0, \mathcal{B}_1, \dots, \mathcal{B}_n$. □

4.4 The algorithm for resizing

Starting from the smallest box containing the `Cross`, we recursively generate a list of of all boxes that can be resized from feasible boxes. By Lemma 6 this list contains all boxes that contain Morpion 5D positions. The code summarizing the recursive procedure is provided in Appendix D.

Lemma 7. *Every Morpion 5D position fits into one of the feasible boxes found by the resizing algorithm. The computed bound is an upper bound for Morpion 5D.*

Proof. The proof follows from Lemma 6. □

5 Upper bounds

5.1 An upper bound of 84 for Morpion 5D

In order to conclude the proof of Theorem 1 we apply the process described in Section 4 to produce all bounding boxes relevant to Morpion 5D game. We solve the linear problem L_1-L_5 specified in Definition 6 in all these boxes. Thanks to Lemmas 3 and 4, the solutions of linear programs are in one-to-one correspondence with existence of Morpion 5D graphs. This concludes the proof of Theorem 1.

Theorem 1 implies the upper bound of 85 which follows from solving the optimization problem L_1-L_5 on all boxes generated in Section 4. In order to improve the result to 84 we have to consider three bounding boxes which admit marked Morpion 5D graphs of size

85. These are specifically boxes $(4,3,1,1)$, $(4,3,1,2)$, $(4,3,1,3)$. On these three boards not only we require that a given solution is a Morpion 5D graph (conditions L_1-L_5 of Definition 6), but additionally we also require that the graph conforms to condition L_7 of Definition 6. By Lemmas 3 and 4 this means that we try to solve the full Morpion 5D game on the bounding boxes $(4,3,1,1)$, $(4,3,1,2)$, $(4,3,1,3)$. In general it does not seem to be a feasible task, but we succeeded to complete the computations with the cutoff condition of 84.9.

5.2 A solution of Morpion 5D with center symmetry

We consider Morpion 5D positions which are invariant with respect to 8 symmetries specified in Subsection 4.1. We say that a Morpion 5D position or a graph is *symmetric* if it is invariant with respect to these symmetries. In analogy to Theorem 1 we can formulate the following result.

Theorem 8. (1) *Up to symmetry every symmetric Morpion 5D* position *is contained in box* $(4,2,4,2)$ *or box* $(3,3,3,3)$. **(2)** *Every box contained in one of the boxes* $(4,2,4,2)$*,* $(3,3,3,3)$ *is a bounding box of a symmetric Morpion 5D* graph. **(3)** *For each box described in* (2)*, the size of a maximal Morpion 5D graph is:* 76 *for boxes* $(2,1,2,1),(2,2,2,2)$*;* 74 *for boxes* $(4,2,4,2)$*,* $(4,1,4,1)$*,* $(3,2,3,2)$*,* $(3,1,3,1)$*;* 72 *for boxes* $(1,1,1,1)$*,* $(1,0,1,0)$ *and* 68 *or less for other boxes.* **(4)** *For each box described in* 3*, the size of a maximal Morpion 5D position is* 68.

The proof is analogous to the proof of Theorem 1. In the proof we additionally use the condition L_6 of Definition 6. To prove Theorem 8.4, we use the condition L_7. In order to speed up computations we used an additional optimization: instead of solving the game on boxes, we solve it on slightly smaller octagons. This generates more cases, because we resize octagons in 8 and not 4 directions. However, the size of octagons is growing slower, hence the problems are easier to solve.

Corollary 9. *The longest sequence of moves leading to a symmetric Morpion 5D position is equal to* 68.

Proof. The upper bound follows from Theorem 8. A record consisting of 68 moves was found by M. Quist and is presented in [1]. □

We are not aware of any previous upper bounds for symmetric Morpion 5D or Morpion 5T.

6 Computations

The total time to solve all involved linear problems appearing in Table 1 amounted to ≈ 3000 hours on a single core of a Linux machine equipped with `Intel`® `Xeon`® `CPU 5620@2.40GHz` with 24GB of RAM. We used the `gurobi` [4] linear optimizer. The code generating the linear programs is available in the repository [9].

7 Further research

The results presented in this paper are based on a specific generalization of the Morpion 5D game. Using this generalization, the task of reducing the bound from 84 down to 82 seems to be feasible, but requires a significant computational effort[2]. It would be interesting to see another generalization which is less costly in terms of required computations, e.g. a generalization that avoids condition L_7 in favor of a different requirement.

The method presented in this paper can be used to show an upper bound in the symmetric Morpion 5T. The method used in [8] leads to the bound of 485 in an arbitrary Morpion 5T game. A record of 136 moves in symmetric Morpion 5T was found by M. Quist (see [1]). We conjecture that an upper bound of ≈ 200 moves can be found using methods presented in this paper.

References

[1] Christian Boyer. *Morpion Solitaire.* 2015. URL: `http://www.morpionsolitaire.com`.

[2] Tristan Cazenave. "Reflexive Monte-Carlo Search". In: *Proceedings of Computers Games Workshop 2007 (CGW 2007).* 2007.

[3] Erik D. Demaine et al. "Morpion Solitaire". In: *Theory Comput. Syst.* 39.3 (2006), pp. 439–453.

[4] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual.* 2015. URL: `http://www.gurobi.com`.

[5] Heikki Hyyrö and Timo Poranen. "New Heuristics for Morpion Solitaire". In: *preprint.* 2007.

[6] Akitoshi Kawamura et al. "Morpion Solitaire 5D: a new upper bound 121 on the maximum score". In: *CCCG.* 2013.

[7] Stefan Langerman. *Morpion Solitaire.* 2009. URL: `http://slef.org/jeu/`.

[8] Henryk Michalewski, Andrzej Nagórko, and Jakub Pawlewicz. "485 - A new upper bound for Morpion Solitaire". In: *CGW@IJCAI.* 2015.

[9] Henryk Michalewski, Andrzej Nagórko, and Jakub Pawlewicz. *Linear programs giving a new upper bound in Morpion 5T and 5D games.* 2015. URL: `https://github.com/anagorko/morpion-lpp`.

[10] Christopher D. Rosin. "Nested Rollout Policy Adaptation for Monte Carlo Tree Search." In: *IJCAI.* 2011, pp. 649–654.

[2]The webpage `https://github.com/anagorko/morpion-lpp/wiki/Solving-5D` shows the current status of these computations.

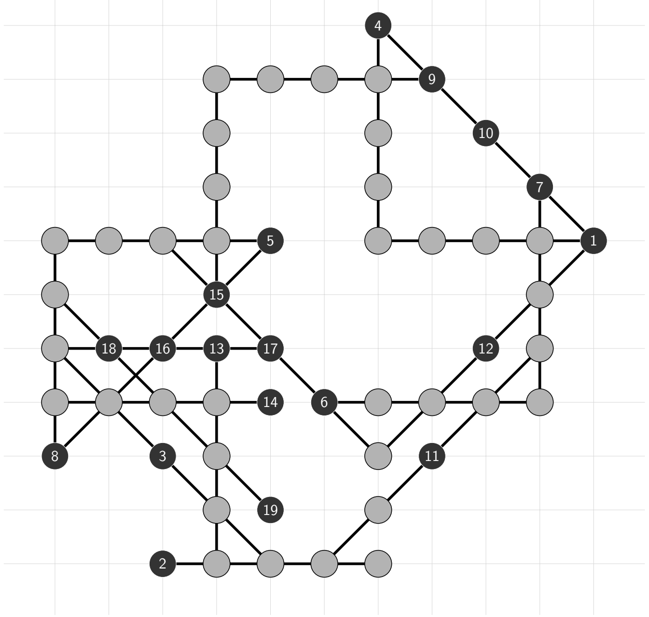


Figure 3: A Morpion 5D position.

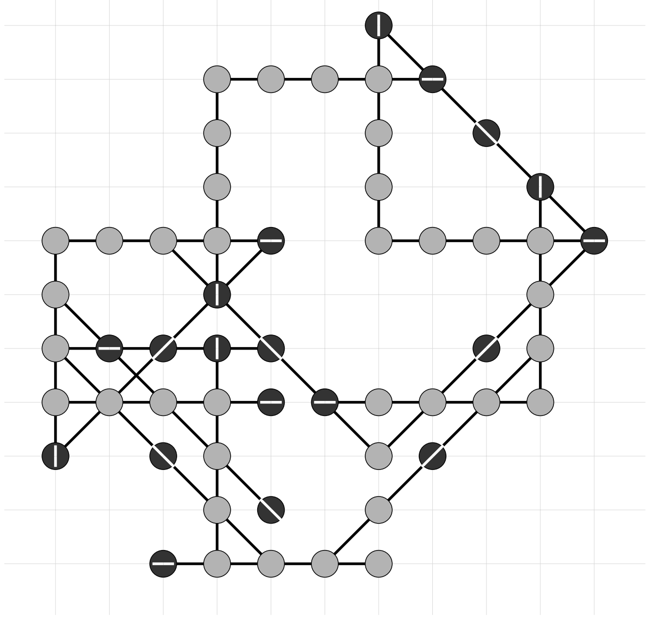


Figure 4: Corresponding unordered Morpion 5D graph.

A Appendix: comments on the definition of Morpion 5D graphs

Example 1. *This example is meant to emphasize the difference between actual Morpion 5D positions for which it is possible to identify all moves in a given gameplay and marked Morpion 5D graphs. Consider a Morpion 5D position shown in Figure 3. The diagram shows a position of a Morpion 5D gameplay. We observe that*

- *there are three different lines passing through dot with number* 1 *and it might be not clear which line corresponds to the first move, however*
- *the last move in this gameplay is unique; in Figure 3 this is move* 19 *and we can identify the whole move thanks to the fact that dot* 19 *was not used in previous moves; the rest of the moves can be decoded recursively, descending from the last move in the sequence.* [3]

For a marked Morpion 5D graphs which are not related to any Morpion 5D position, such as one in Figure 2, there is no easy way to establish order of moves.

Example 2. *Figure 2 contains an example of a marked Morpion 5D graph G_{85}. Every vertex of G_{85} that is not in* `Cross` *is labelled with one of the four directions. For every move m from $\mathcal{M}(G_{85})$ there exists exactly one vertex in m labelled with the direction of m. This allows us to decode the selected marking of G_{85}. The graph G_{85} is not a Morpion 5D position, because every Morpion 5D position admits the last move, characterized by the fact that the last vertex is of degree 1 or 2 and if it is of degree 2, then the neighbours must be located on a straight line. There is no such vertex in G_{85}.*

[3]This simple observation was generalized to a non-trivial algorithm which not only recovers the sequence of moves starting from the last one, but also recovers a correct numbering of moves, in particular identifies the last move [3].

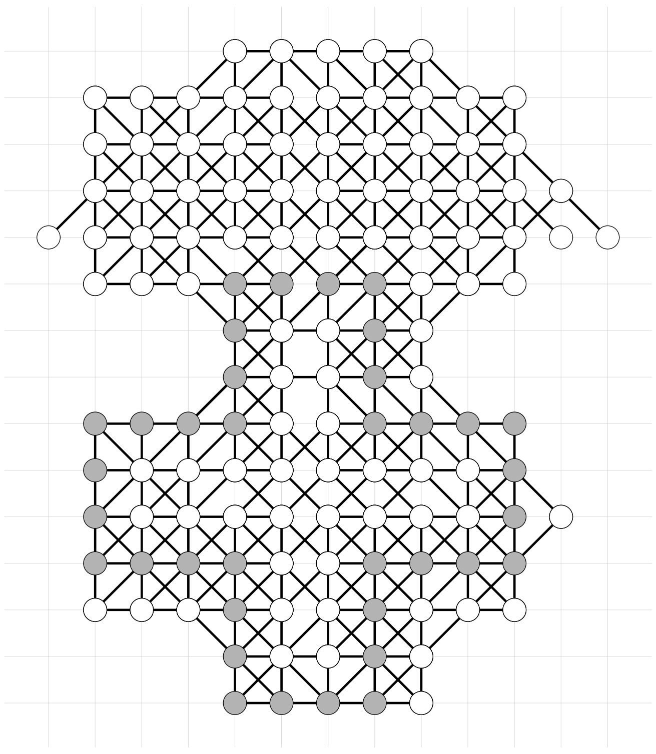


Figure 5: A Morpion 5D graph of size 94.

Example 3. *Figures 5 and 6 contain examples of Morpion 5D graphs. Neither of these graphs admit a marking. The*

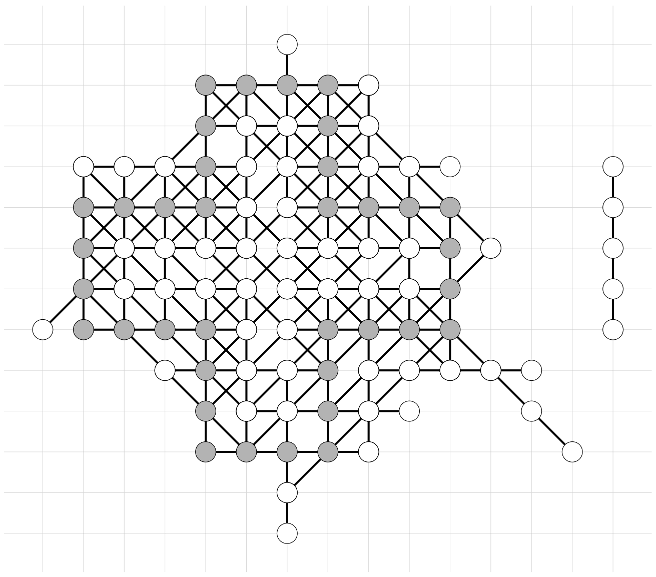


Figure 6: A nonconnected Morpion 5D graph, which may have arbitrarily large bounding box.

graph in Figure 5 has a bounding box $(5, 2, 0, 1)$. *It follows from Theorem 1 that the maximal size of a marked Morpion 5D graph with this bounding box is*[4] *at most* 82. *The size of this graph is* 94, *hence it does not admit a marking. The graph in Figure 6 has a connected component with five dots. A marking would have to be a map that assigns all of these dots to a single move. Therefore no marking exists for this graph.*

Remark 1. A Morpion 5T equivalent of the definition of an (unmarked) Morpion 5D graph was used in [8] to compute a bound for Morpion 5T. Considering this class for Morpion 5D does not give useful bounds. There are two reasons why this method does not work.

1. These graphs need not to be connected and therefore the bounding box of such a graph can be arbitrarily large, see Figure 5 (right),
2. Even if we insist on connectedness, there are examples of Morpion 5D graphs of size exceeding sizes in table 1, see Figure 5. In this specific example, the graph has 94 vertices.

B Appendix: proof of Lemma 3

Proof. Assume that the variables satisfy conditions L_1 - L_5. We will show that

> (*) if $e \in E$, then there exists a unique pair $m_e \in \mathcal{M}(G)$ and $v_e \in m_e$ such that $e \in E(m_e)$ and $\mathrm{mv}_{m_e,v_e} = 1$.

By the definition of E, if $e \in E$, then there exists $m \in \mathcal{M}(B)$ and $v \in m$ such that $e \in E(m)$ and $\mathrm{mv}_{m,v} = 1$. If there exists an another pair $m' \in \mathcal{M}(B)$ and $v' \in m'$ such that $e \in E(m')$ and $\mathrm{mv}_{m',v'} = 1$, then $m \parallel e \parallel m'$ and by L_4 for $w \in e$ we have $1 \geq \mathrm{dt}_w \geq \mathrm{mv}_{m,v} + \mathrm{mv}_{m',v'} = 2$. A contradiction that shows that m and v are unique. By the definition of E, if $m \in \mathcal{M}(B)$ and $\mathrm{mv}_{m,v} = 1$, then $E(m) \subset E$. Hence $m \in \mathcal{M}(G)$. This concludes the proof of (*).

We need to check $G = (V, E)$ is a lattice graph. By the definition of V and E, we have $V \subset \mathbb{Z}^2$ and every element of E is a unit edge, hence it remains to check that for every edge its ends are vertices in V. Let $e \in E$. By (*) there exists m_e and v_e such that $\mathrm{mv}_{m_e,v_e} = 1$ and $e \in E(m_e)$. If $v \in e$, then $v \in m_e$ and by L_4, we have $\mathrm{dt}_v \geq \mathrm{mv}_{m_e,v_e} = 1$. Hence from the definition of V we have $v \in V$. This concludes the proof that $G = (V, E)$ is a well defined lattice graph.

In order to check that G is a Morpion 5D graph we have to verify that $\mathtt{Cross} \subset V$, $\mathcal{M}(G)$ covers E and $\mathcal{M}(G)$ is 5D-disjoint.

Let us notice, that by L_1, $\mathtt{Cross} \subset V$. Moreover, from (*) for every $e \in E$ it holds that $e \in E(m_e)$. Hence the set of moves in G covers the set of edges E.

It remains to check that $\mathcal{M}(G)$ is 5D-disjoint. Assume that $e_1, e_2 \in E$ are parallel and $v \in e_1 \cap e_2$. We claim that $m_{e_1} = m_{e_2}$. By L_4 and (*), if $m_{e_1} \neq m_{e_2}$, then $1 \geq \mathrm{dt}_v \geq \mathrm{mv}_{m_{e_1},v_{e_1}} + \mathrm{mv}_{m_{e_2},v_{e_2}} = 2$, a contradiction. Assume that two moves $m_1, m_2 \in \mathcal{G}$ are parallel, have non-empty intersection and $m_1 \neq m_2$. Then $E(m_1) \cup E(m_2)$ contains at least five distinct edges $e_1, e_2, \ldots e_5$ such that e_i intersects e_{i+1}. Then for each i, $m_{e_i} = m_{e_{i+1}}$. Let $m = m_{e_i}$. We have $e_i \in E(m_{e_i}) = E(m)$, which contradicts the fact that $E(m)$ has four edges. Therefore parallel moves in $\mathcal{M}(G)$ are either equal or have empty intersection. Hence $\mathcal{M}(G)$ is 5D-disjoint. This concludes the proof that G is a Morpion 5D graph.

Now we will prove that G is a marked Morpion 5D graph. By 5D-disjointness, if $m \in \mathcal{M}(G)$ and e is an arbitrary edge in $E(m)$, then $m = m_e$, that is the choice of m_e and v_e in (*) does not depend on e. Let $d\colon \mathcal{M}(G) \to V_B$ be a map defined by the formula $d(m) = v_e$.

By (*), $\mathrm{mv}_{m_e,v_e} = 1$ and by L_2 if $v_e \in \mathtt{Cross}$, then $\mathrm{mv}_{m_e,v_e} = 0$, hence $v_e \in V_B \setminus \mathtt{Cross}$. This proves that d is into $V_B \setminus \mathtt{Cross}$.

We will show that d is one-to-one. Assume that $d(m_1) = d(m_2) = v$. If $m_1 \neq m_2$, then by L_3, $1 \geq \mathrm{dt}_v \geq \mathrm{mv}_{m_1,v} + \mathrm{mv}_{m_2,v} = 2$. Hence $m_1 = m_2$ and this concludes the proof that d is one-to-one. By L_3, the cardinality of $V_B \setminus \mathtt{Cross}$ is equal to the number of pairs m, v such that $\mathrm{mv}_{m,v} = 1$, which by (*) is the number of moves in $\mathcal{M}(G)$. Therefore the domain and range of d are of equal cardinality, so d must be a bijection. This shows that G is a marked 5D graph.

In order to check that $\mathcal{B}$ is the bounding box of G we should verify that on every side of $\mathcal{B}$ there exists a vertex of V. This is guaranteed by the condition L_5.

If values of variables mv are symmetric, then graph G is symmetric, hence L_6 enforces a center symmetry of G.

If we assume L_7, then each vertex $v \in V_G \setminus \mathtt{Cross}$ has assigned order variable ord_v. We will show a Morpion 5D gameplay which leads to the graph G.

[4]A calculation not listed in statement of Theorem 1 shows that the maximal size of a marked Morpion 5D graph with the bounding box $(5, 2, 0, 1)$ is equal to 80.

```
S  :=  $V_G \setminus$ Cross
while  $S \neq \emptyset$
 select  $v \in S$  with the minimal  $\mathrm{ord}_v$
 place  $v$
 draw  the  move  $d^{-1}(v)$
 S  :=  $S \setminus \{v\}$
```

We have to verify the following four properties of this gameplay.

(1) Every dot is placed on a place which is not occupied by another dot.

(2) Parallel moves are disjoint.

(3) The resulting Morpion 5D position is equal to G.

(4) Every move is placed only when required dots are available.

Conditions (1)-(3) follows from the previous reasoning. Let us verify the last condition. Let $v \in V_G \setminus \texttt{Cross}$ and let $m = d^{-1}(v)$. Fix $w \in m, v \neq w$. We have to check that w was placed before v. By the definition of d we have $\mathrm{mv}_{m,v} = 1$. The condition L_7 enforces that $\mathrm{ord}_v \geq \mathrm{ord}_w + 1$. Therefore, the algorithm placed w earlier in the gameplay. □

C Appendix: proof of Lemma 4

Proof. We verify conditions L_1 - L_7 of Definition 6.

L_1. By the definition of a Morpion 5D graph, $\texttt{Cross} \subset V$, hence $\mathrm{dt}_v = 1$ for $v \in \texttt{Cross}$.

L_2. Since d is into $V \setminus \texttt{Cross}$ and $\mathrm{mv}_{m,v} = 1$ implies $v = d(m)$, we have $\mathrm{mv}_{m,v} = 0$ for $v \in \texttt{Cross}$.

L_3. If $v \in V \setminus \texttt{Cross}$, then there exists exactly one $m \in \mathcal{M}(G)$ such that $d(m) = v$ as d is a bijection onto $V \setminus \texttt{Cross}$. Hence there exists exactly one move $m \in \mathcal{M}(B)$ that contains v, such that $\mathrm{mv}_{m,v} = 1$ as $\mathrm{mv}_{m,v} = 1$ iff $m \in \mathcal{M}(G)$ and $v = d(m)$. Therefore $\mathrm{dt}_v = 1 = \sum_{m \in \mathcal{M}(B):\, v \in m} \mathrm{mv}_{m,v}$. If $v \in V_B \setminus V_G$, then dt_v and all $\mathrm{mv}_{m,v}$ are equal to 0.

L_4. Let $v \in V_B$. If $v \notin V$, then there is no $m \in \mathcal{M}(G)$ such that $d(m) = v$, hence dt_v and all $\mathrm{mv}_{m,v} = 0$. If $v \in V$ and $d \in \mathcal{D}$, then $\mathrm{dt}_v = 1$ and by 5D-disjointness of G there exists at most one move $m \in \mathcal{M}(G)$ such that $m \parallel d$ and $v \in m$. For $w \in m$ we have $\mathrm{mv}_{m,w} = 1$ iff $w = d(m)$. So the sum on the right hand side of inequality in L_4 contains at most one non-zero element.

L_5. Since $\mathcal{B}$ is the bounding box of G, for each side S of $\mathcal{B}$ there exists a vertex $v \in V \cap S$. Then $\mathrm{dt}_v = 1$ and $\sum_{v \in S} \mathrm{dt}_v \geq 1$.

L_6. Let m be an arbitrary $m \in \mathcal{M}(G)$ and $v = d(m)$. Then from the assumption that G and d are symmetric follows that $v_s = d(m_s)$.

L_7. By [6] the longest sequence in Morpion 5D does not exceed 121, so the numbers in ord_v are from 0 to at most 120. If $\mathrm{mv}_{m,v} = 0$, then condition L_7 is essentially void (L_7 is triggered by $\mathrm{mv}_{m,v} = 1$). If $\mathrm{mv}_{m,v} = 1$ then in the gameplay there was a move m which placed the dot v. This implies that for every $w \in m$, $w \neq v$, w was placed before v. Hence $\mathrm{ord}_v \geq \mathrm{ord}_w + 1$ and L_7 is satisfied.

□

D Appendix: the code summarizing the resizing process

```
while unsolved:
 box = unsolved.pop()
 # we use a linear solver to establish
 # if a given box is feasible
 result = solve(box)
 solved.append(box)

 if result.bound > bound:
  bound = result.bound

 if result.feasible:
  [ a, b, c, d ] = box

 # we add eight potential new boxes to
 # the list of boxes which should be analyzed
 resized = [ [a+1, b, c, d], [a, b+1, c, d],
             [a, b, c+1, d], [a, b, c, d+1],
             [a+1,b+1, c,d], [a, b+1,c+1,d],
             [a,b, c+1,d+1], [a+1,b,c, d+1] ]

 # ... and eliminate these cases which were
 #           - already solved or
 #           - are already present on the stack.
 # this process is done up to symmetry
 for g in resized:
  if symmetryClass(g) not in solved + unsolved:
   unsolved.append(symmetryClass(g))
```

Kernelizing Buttons and Scissors

Akanksha Agrawal [*] Sudeshna Kolay[†] Saket Saurabh[†] Roohani Sharma[†]

Abstract

Buttons & Scissors (*B&S*) is a popular logic game, where we are given an n-by-m grid with some cells being sewed with colored buttons, and the objective is to decide whether we can empty the grid by certain scissor cuts. Recently, the computational complexity of this problem has attracted a lot of attention, and the problem, together with several of its restrictions, has been shown to be NP-complete. In this paper we study this problem in the realm of parameterized complexity. In particular, we show that *B&S*, when restricted to using only horizontal and vertical scissor cuts, is FPT parameterized by the number of cuts required to empty the grid (say k). Precisely, we design an algorithm that runs in time $2^{\mathcal{O}(k \log k)} + (n+m)^{\mathcal{O}(1)}$ and decides whether there is a wining sequence with at most k scissor cuts. At the heart of our algorithm is a polynomial time kernelization procedure, that, given an instance of the game on a n-by-m board, obtains an equivalent instance on a board of dimension $\mathcal{O}(k^2)$-by-$\mathcal{O}(k^2)$ that has at most $\mathcal{O}(k^3)$ buttons. When the input is restricted to n-by-1 grid, we get a kernel with board size $\mathcal{O}(k)$-by-1 and an algorithm with running time $2^{\mathcal{O}(k)} + n^{\mathcal{O}(1)}$. Finally, in the general setting, that is when diagonal scissor cuts are also allowed, we show that the problem is FPT parameterized by k and r; where r is an upper bound on the length of any diagonal cut.

1 Introduction

Buttons & Scissors (*B&S*) is a logic game which was introduced by KyWorks in 2012-2013 and is available for free as an Android and iPhone application. There are many other game applications that have been developed for *B&S*. It is a board game where we are given an n-by-m board (matrix) with some of the board cells sewed with colored buttons. The goal is to remove all the buttons on the board with scissor cuts satisfying the following properties.

- Any cut is a straight-line segment which is vertical, horizontal, or diagonal oriented at an angle of 45° or −45°.
- The endpoints of a cut are distinct buttons of the same color.
- All the buttons (including end point buttons) that lie on the line segment of the cut are of same color. Moreover, the cut removes all the buttons on its line segment.

A sequence of cuts is called a *solution* to *B&S*, if the board becomes empty when the cuts are applied in the order specified by the sequence. Recently, Gregg et al. proved that given an n-by-n board, deciding if there exists a solution for *B&S* is NP-complete [5]. Burke et al. [1] studied variants of the *B&S* problem. They showed that when the board consists of buttons of only two colors then deciding the existence of a solution is NP-complete [1]. They also showed that *B&S* is polynomial time solvable when all the buttons on the board are of the same color. The variant of the *B&S* where the frequency of each color is bounded by 4 is NP-complete, while when the frequency is bounded by 3 the problem is decidable in polynomial time [1]. In fact, NP-hardness holds even when *only horizontal and vertical cuts* are allowed, and the frequency of each colour is at most 7. The version of *B&S* where only only horizontal and vertical cuts are allowed is denoted by *B&S+*. Bruke et al [1] also introduced and studied a two player version of *Buttons & Scissors* game and proved it to be PSPACE complete.

In this paper, we study *B&S* and *B&S+* in the realm of parameterized complexity. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: our aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem, Π, is *fixed-parameter tractable (FPT)* if there is an algorithm that solves the problem in time $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the size of the input and f is an arbitrary computable function depending on the parameter k only. There is a long list of NP-hard problems that are FPT under various parameterizations.

A parameterized problem Π with parameter k is said to admit a *kernel* if there is a polynomial-time algorithm, called a *kernelization* algorithm, that reduces the input instance down to an instance whose size is bounded by $f(k)$, while preserving the answer. Here, $f(\cdot)$ is a computable function depending only on k. Such

[*]Department of Informatics, University of Bergen, Norway akanksha.agrawal@uib.no

[†]Institute of Mathematical Sciences, Chennai, India {skolay|saket|roohani}@imsc.res.in

a reduced instance from the kernelization algorithm is called an $f(k)$-*kernel* for the problem. If $f(k)$ is a polynomial we call it a *polynomial kernel.* A parameterized problem Π is in FPT if and only if it admits a kernelization algorithm [2]. For more details on parameterized complexity in general, and kernelization in particular, we refer the reader to the books of Downey and Fellows [3], Flum and Grohe [4], Niedermeier [6], and the more recent book by Cygan et al. [2].

We study the parameterized variant of the *B&S* problem. Some of the natural parameters that can be associated with an instance of *B&S* game are the number of colors of buttons in the board, the maximum frequency of a color on the board, the size of the board and the number of cuts in the solution. Observe that *B&S* is NP-hard even when the numbers of colors of buttons in the board is *two.* Therefore, an algorithm running in time $f(k)n^{\mathcal{O}(1)}$, where k is the numbers of colors of buttons in the board and n is the size of the input, would imply a polynomial time algorithm for *B&S* when the input instances have only buttons of two colors. The existence of such an algorithm implies P=NP which is highly unlikely. Hence, we believe that there is no FPT algorithm for *B&S* parameterized by the number of colors of buttons. By an analogous argument there cannot exist an FPT algorithm for *B&S* parameterized by the maximum frequency of a color in the board.

Our Results. We first define the problem *Buttons & Scissors+* (*B&S+*) and *Buttons & Scissors* (*B&S*) formally. In the problem *B&S+*, the input is an n−by−m board B and positive integers k_h, k_v and the question is whether there is a sequence of horizontal and vertical cuts, with at most k_h horizontal cuts and k_v vertical cuts, such that the resulting board in empty. In the problem *B&S*, the input is an n−by−m board B and positive integers k_h, k_v, k_d, r and the question is whether there is a sequence of cuts with at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts such that the diagonal cuts are of length at most r and the resulting board is empty. For *B&S+*, we give a kernel where the dimension of the board is $\mathcal{O}(k^2) \times \mathcal{O}(k^2)$ with $\mathcal{O}(k^3)$ non-zero entries, where $k = k_h + k_v$. Using this kernel we design an algorithm with running time $2^{\mathcal{O}(k \log k)} + (n+m)^{\mathcal{O}(1)}$. Then we study this problem restricted to n-by-1 boards. This restriction is referred as *One-dim Buttons & Scissors* (*1-Dim-B&S*). In this case notice that we *only have vertical cuts.* We give an $\mathcal{O}(k)$ kernel and a $2^{\mathcal{O}(k)} + n^{\mathcal{O}(1)}$ algorithm for *1-Dim-B&S*, where $k = k_v$. Finally for *B&S*, we show that it admits a kernel where the dimension of the board is $\mathcal{O}(r^3k^2) \times \mathcal{O}(r^3k^2)$ and the number of non-zero entries is $\mathcal{O}(r^3k^3)$. Using this kernel we give an algorithm for this problem that runs in time $2^{\mathcal{O}(k(\log r + \log k))} + (n+m)^{\mathcal{O}(1)}$. Here, $k = k_h + k_v + k_d$.

2 Preliminaries

The notations and terminologies we use are same as in [5]. We denote the set of natural numbers by $\mathbb{N}$. For $r \in \mathbb{N}$, by $[r]$ we denote the set $\{1, 2, \ldots, r\}$. For an n-by-n matrix M, by R_i we denote the 1-by-n sub-matrix of M consisting of the elements in row i of M. By $M[i,j]$ we denote the element in i^{th} row and j^{th} column of M. For $I, J \subseteq [n]$, by $M[I,J]$ we denote the sub-matrix of M restricted to the rows in I and columns in J.

A board B is an n-by-m matrix with entries in $\{0, 1, ..., c\}$, where each $i \in [c]$ is a button of colour i and 0 represents no button. A *cut*, $\mathcal{C}$, is a sequence of pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_l, j_l)$ with $l \geq 2$ satisfying one of the following directional properties.
— Vertical Cut. For all $p \in [l-1]$, $i_{p+1} = i_p + 1$ and $j_{p+1} = j_p$.
— Horizontal Cut. For all $p \in [l-1]$, $i_{p+1} = i_p$ and $j_{p+1} = j_p + 1$.
— Diagonal Cut 1. For all $p \in [l-1]$, $i_{p+1} = i_p + 1$ and $j_{p+1} = j_p + 1$.
— Diagonal Cut 2. For all $p \in [l-1]$, $i_{p+1} = i_p + 1$ and $j_{p+1} = j_p - 1$.

For convenience, we may also refer to a cut in the reverse order.

For the *Buttons & Scissors* game, a *cut*, $\mathcal{C} = (i_1, j_1), \ldots, (i_l, j_l)$ for any n-by-m board B must also satisfy the following conditions.
— For all $p \in [l]$, $i_p \in [n]$ and $j_p \in [m]$
— $B[i_1, j_1], \ldots, B[i_l, j_l]$, contains at least 2 non-zero entries and all non-zero entries correspond to buttons of the same colour.

When a cut $\mathcal{C} = (i_1, j_1), \ldots, (i_l, j_l)$ is applied to a board B, the resulting board B' is obtained by setting the entries $B[i_1, j_1]$ to zero (or empty entry) in $B'[x_i, y_i]$, for $i \in [l]$. All other entries of B' are same as B. That is, a cut removes all the buttons along its path. A cut $\mathcal{C} = (i_1, j_1), \ldots, (i_l, j_l)$ is said to *touch* the rows $\{R_{i_1}, \ldots, R_{i_l}\}$ and the columns $\{C_{j_1}, \ldots, C_{j_l}\}$. It is said to be *completely contained* in a submatrix $B[I,J]$ of B, if $\{i_1, \ldots, i_l\} \subseteq I$ and $\{j_1, \ldots, j_l\} \subseteq J$.

A *solution* for a board B_0 is a sequence $\mathcal{C}_1, \ldots, \mathcal{C}_k$ of cuts where the following property is satisfied. Let B_i denote the board obtained after application of cuts $\mathcal{C}_1, \cdots, \mathcal{C}_i$ to B. For $i \in [k]$, $\mathcal{C}_i$ is a cut in B_{i-1}. Also, B_k has no non-zero entry, i.e, B_k is an empty board. We use the term matrix entry and buttons interchangeably.

We also have two simple observations that will be useful in designing algorithms for the problems we consider in the paper.

Observation 1 *Given an $n \times m$ board with ℓ buttons, we can find if there is a sequence of cuts for the board, with at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts, in $\ell^{2(k_h+k_v+k_d)}(n+m)^{\mathcal{O}(1)}$ time.*

Proof. For any solution for the board, each cut is defined by the two buttons at the end points of the cut. This implies that we have at most $\binom{\ell}{2} \leq \ell^2$ cuts. Thus, for a solution with at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts, there are at most $l^{2(k_h+k_v+k_d)}$ possibilities for the sequence of endpoints of cuts for the solution. All such possible sequences of endpoints can be found in $\ell^{2(k_h+k_v+k_d)}(n+m)^{\mathcal{O}(1)}$ time. Further, it takes polynomial time for each sequence of endpoints, to check if the sequence defines a sequence of at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts. If such a solution exist, it exists in our enumeration and we output one which has at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts. This is a required solution for the board. Therefore, the total running time of the algorithm is $\ell^{2(k_h+k_v+k_d)}(n+m)^{\mathcal{O}(1)}$. □

Observation 2 *Suppose I is an instance of* Buttons & Scissors *game, where the board has dimensions $n \times m$. If at most k cuts are allowed and the number of buttons on B are at least $k \cdot \max\{n,m\}+1$, then I is a* NO *instance.*

Proof. A cut can set to zero at most $\max\{n,m\}$ non-zero entries to zero. Therefore, if at most k cuts are allowed, then the total number of non-zero entries that can be set to zero are at most $k \cdot \max\{n,m\}$. Thus, I is a NO instance. □

We now define some terminology regarding kernelization. A *Reduction Rule* is a polynomial time algorithm that given an instance I of some problem Π converts it to an instance I' of Π. We say that a Reduction Rule is *safe*, if I is a YES instance if and only if I' is a YES instance. We refer to I' as the resulting/reduced instance.

3 Kernel for *B&S+*

In this section we design a kernel for *B&S+*. Let (B, k_h, k_v) be an instance of *B&S+*, where B is an n-by-m matrix. We let $k = k_v + k_h$. The algorithm starts by applying the following Reduction Rules exhaustively.

Reduction Rule 1 *If B has a row (or a column), say i (or j) with all zero entries, i.e. all $j \in [m]$ (or $i \in [n]$), $B[i,j] = 0$. Let $B' = B[[n] \setminus \{i\}, [m]]$ (or $B' = B[[n], [m] \setminus \{j\}]$). Then the resulting instance is (B', k_h, k_v).*

It is easy to see that Reduction Rule 1 is safe. Reduction Rule 1 will be applied whenever possible, and with priority over other reductions rules that will be described later. In particular, we apply Reduction Rule i only if Reduction Rule i', where $i' < i$, is not applicable.

Next, we describe reduction rules to bound the number n of rows of B, when the input is a YES instance. In a symmetric way, we can bound the number m of columns of B, when the input is a YES instance. This tells us that if the input instance, after applying the reduction rules exhaustively, has a lot of rows or columns, then we can safely say NO.

Since Reduction Rule 1 do not apply, there must be at least one non-zero entry in each row B. Next, we define a notion of *row-blocks*, which is slightly more general than we need here. However, it will be used in its full generality in coming sections.

Definition 1 (Row Block) *Given a $n \times m$ matrix B, the submatrix $B[X,[m]]$ is called a* row-block *if $X = \{a, a+1, \ldots, a+l\}$ is a maximal set of consecutive rows of B such that (i) R_a has at least one non-zero entry, (ii) for each $i \in X$, either $R_i = R_a$ or all entries of R_i are equal to zero.*

Notice that two different row blocks do not intersect and thus they define a partition of the set of rows in B. The number of rows in a row-block is referred to as the *size* of the row-block. Since Reduction Rule 1 does not apply, every row is non-empty. Let $B[X,[m]]$ be a row-block of B. Then, for each $i,j \in X$, $B[i,[m]] = B[j,[m]]$. We first show that if the size of a row-block in B is too large then we can find an equivalent instance (B', k_h, k_v) such that the number of rows with non-zero entries in B' is strictly smaller than that of B (and then Reduction Rule 1 will become applicable).

Reduction Rule 2 *Let $R_i, \ldots, R_p$ be a row-block such that $p-i+1$ is at least $3(k+1)+1$. Let i_0 be such that $i+k < i_0 \leq p-(k+1)$. Let $B^{'}$ be a matrix obtained from B by setting all entries of R_{i_0} to 0. Then the resulting instance is $(B^{'}, k_h, k_v)$.*

Lemma 1 *Reduction Rule 2 is safe.*

Proof. First, as a sanity check, the definition of a row-block and the number of rows in this row-block ensures that such an i_0 exists. Now, we prove the correctness of this Reduction Rule.

First, suppose (B, k_h, k_v) is a YES instance of *B&S+*. Consider a solution $\mathcal{S}$ of B. We first construct another solution $\mathcal{S}'$ of B, using Claim 1.

Claim 1 *If $\mathcal{S}$ is a solution of the instance (B, k_h, k_v), then there exists another solution $\mathcal{S}'$ to the instance (B, k_h, k_v), such that every non-zero entry in $R_i, \ldots, R_p$ is set to zero by a vertical cut in $\mathcal{S}'$.*

Proof. [Proof Sketch] We define the cuts of $\mathcal{S}'$ based on the cuts of $\mathcal{S}$. Starting from the first cut of $\mathcal{S}$, for the j^{th} cut $\mathcal{C} \in \mathcal{S}$ we define the j^{th} $\mathcal{C}' \in \mathcal{S}'$ as follows:

1. Suppose $\mathcal{C}$ is a cut that does not touch any row of X. Then $\mathcal{C}' = \mathcal{C}$.
2. Suppose $\mathcal{C}$ touches all rows of X, then $\mathcal{C}' = \mathcal{C}$.

3. Suppose $\mathcal{C}$ is a horizontal cut completely contained in X. Then set $\mathcal{C}'$ as $\emptyset$.
4. Suppose $\mathcal{C}$ is a vertical cut that touches a column $j \in [m]$, and some rows of X. Also, suppose that there is a cut later in the sequence that touches a row of X and the column j, and sets a non-zero entry of $B[X, \{j\}]$ to zero. Then set $\mathcal{C}' = \mathcal{C}$.
5. Suppose $\mathcal{C}$ is a vertical cut on a column $j \in [m]$ such that $\mathcal{C}$ crosses row i (row p) but not row p (row i), and there is no cut later in the sequence that touches a row of X and the column j, and sets a non-zero entry of $B[X, \{j\}]$ to zero. Let $X' \subseteq X$ be the rows which are untouched by all vertical cuts till $\mathcal{C}$ that touch column j. Then $\mathcal{C}'$ is defined to be the cut with starting point at the lowest indexed row touched by $\mathcal{C}$ (highest row touched by $\mathcal{C}$) and ending at the highest indexed row r_1 of X' (lowest row r_1 of X'). Notice that the lowest indexed row (highest indexed row) where the cut starts could be above the row i. By definition of a row-block and the cut $\mathcal{C}'$, all entries in column j, between row i and r_1 (r_1 and p) are either non-zero entries of the same value or zeroes. By the definition of $\mathcal{C}$ and X', the endpoints of $\mathcal{C}'$ are non-zero entries. Therefore, $\mathcal{C}'$ is a cut of B.
6. Suppose $\mathcal{C}$ is a vertical cut on a column j such that $\mathcal{C}$ does not cross row i or row p and is completely contained inside X, and there is no cut later in the sequence that touches a row of X and the column j, and sets a non-zero entry of $B[X, \{j\}]$ to zero. Let $X' \subseteq X$ be the rows where there are non-zero entries and which are untouched by all vertical cuts till $\mathcal{C}$ that touch j. Then $\mathcal{C}'$ is defined to be the cut with starting point at the lowest indexed row of X' and ending point at the highest indexed row of X'. By definition of a row-block and the cut $\mathcal{C}$, all entries in column j, between the lowest and highest rows of X' are either non-zero entries of the same value or zeroes. By the definition of $\mathcal{C}$ and X', the endpoints of $\mathcal{C}'$ are non-zero entries. Therefore, $\mathcal{C}'$ is a cut of B.

Finally, observe that in the above sequence every cut is well-defined in the sense that either its is empty or it is *always* between two buttons of the same color and by our construction we have ensured that every button along the cut is of the same color. In the end, we go through the sequence formed above, and remove any $\emptyset$ from the sequence. This sequence is our $\mathcal{S}'$.

It remains to show that $\mathcal{S}'$ is a solution of B. By definition of $\mathcal{S}'$, the number of horizontal/vertical cuts in $\mathcal{S}'$ is at most the number of horizontal/vertical cuts of $\mathcal{S}$. We have argued that each element in $\mathcal{S}'$ is a cut of B. We show that $\mathcal{S}'$ sets to zero all non-zero entries of B. It is enough to show that all non-zero entries of $B[X, [m]]$ is set to zero by $\mathcal{S}'$. By the definition of a horizontal cut, at most k_h rows of X can be touched by the horizontal cuts of $\mathcal{S}$. Since there are at least $3(k+1)+1$ rows in X and since Reduction Rule 1 does not apply, by pigeonhole principle, there is at least one row a in X (in fact, $2k+4$ rows in X) such that no horizontal cut of $\mathcal{S}$ touches R_a. This means that each non-zero entry of R_a is set to zero by a vertical cut. Therefore, every column that has at least one non-zero entry must be touched by a vertical cut at some row of X. Cuts of type $(3-5)$ set to zero all non-zero entries of any column, at any row of X. Therefore, $\mathcal{S}'$ is a solution of B, that has the property that all non-zero entries of rows in X are set to zero only by vertical cuts. □

Next, we show that there is a solution $\mathcal{S}''$, where every row in X has its non-zero entries set to zero by vertical cuts, and where no vertical cut ends at row R_{i_0}.

Claim 2 *There exists a solution $\mathcal{S}''$ with at most k_h horizontal cuts and at most k_v vertical cuts, and such that every non-zero entry in $R_i, \ldots, R_p$ is set to zero by a vertical cut in $\mathcal{S}'$. Moreover, no non-zero entry of R_{i_0} is an end point to any cut in $\mathcal{S}''$.*

Proof. [Proof Sketch]
By choice of i_0, there are at least $k+1$ rows of X above R_{i_0} and at least $k+1$ rows below R_{i_0}. Let U_1 be the first $k+1$ rows in X. Similarly, we can define U_2 as the last $k+1$ rows in X. We define $\mathcal{S}''$ using $\mathcal{S}'$. Starting from the first cut in $\mathcal{S}'$, we consider the j^{th} cut $\mathcal{C}'$ in $\mathcal{S}'$ and try to describe the j^{th} cut $\mathcal{C}''$ in $\mathcal{S}''$ in the following way:

1. Suppose $\mathcal{C}'$ is a cut that does not touch any row of X then $\mathcal{C}'' = \mathcal{C}'$.
2. Suppose $\mathcal{C}'$ touches all rows of X, then $\mathcal{C}'' = \mathcal{C}'$.
3. Otherwise, by definition of $\mathcal{S}'$, the cut is a vertical cut touching some row of X. Suppose $\mathcal{C}'$ is vertical cut, that touches a column j and is completely contained inside X. Also, suppose there is a vertical cut, after $\mathcal{C}'$ in $\mathcal{S}'$, that touches column j and some row of X, and sets a non-zero entry in $B[X, j]$ to zero. Then, we set $\mathcal{C}''$ to be $\emptyset$.
4. Suppose the cut $\mathcal{C}'$ touches column j, crosses row i but not row p. In the sequence $\mathcal{S}'$, let $\mathcal{C}'$ be the b^{th} cut that touches column j, crosses row i but not row p. Also, suppose there is a cut later in the sequence that touches column j and some rows of X, and sets a non-zero entry in $B[X, j]$ to zero. Then $\mathcal{C}''$ is the cut, touching column j, and whose starting point is at the lowest row of $\mathcal{C}'$ while its ending point is the b^{th} row of U_1.
5. Similarly, if $\mathcal{C}'$ touches column j, crosses row p but not row i. Also, suppose there is a cut later in the sequence that touches column j and some rows of X, and sets a non-zero entry in $B[X, j]$ to zero. In the sequence $\mathcal{S}'$, let C' be the b^{th} cut that touches

column j, crosses row p but not row i. Then $\mathcal{C}''$ is the cut, touching column j, and whose ending point is at the highest row of $\mathcal{C}'$ while its starting point is the b^{th} row from bottom in U_2.

6. Suppose $\mathcal{C}'$ touches column j, crosses row i but not row p. Suppose there is no cut later in the sequence that touches rows of X and the column j, and sets a non-zero entry in $B[X, j]$ to zero. Then $\mathcal{C}''$ is the cut, touching column j, and whose starting point is at the lowest row of $\mathcal{C}'$ while its ending point is the highest row of $U_1 \cup U_2$, that is untouched by previous cuts of $\mathcal{S}'$.
7. Suppose $\mathcal{C}'$ touches column j, crosses row p but not row i. Suppose there is no cut later in the sequence that touches rows of X and the column j, and sets a non-zero entry in $B[X, j]$ to zero. Then $\mathcal{C}''$ is the cut, touching column j, and whose ending point is at the highest row of $\mathcal{C}'$ while its starting point is the lowest row of $U_1 \cup U_2$ that is untouched by previous cuts of $\mathcal{S}'$.
8. Otherwise, $\mathcal{C}'$ is vertical cut, that touches a column j and is completely contained inside X. Suppose, after $\mathcal{C}'$, there are no vertical cuts in $\mathcal{S}'$ that touches column j and some row of X, and sets a non-zero entry in $B[X, j]$ to zero. Then $\mathcal{C}''$ touches column j, has the starting point at the lowest row of U_1 and ending point at the highest row of U_2. Otherwise, if there is a vertical cut, later in the sequence that touches column j and some rows of X, we set $\mathcal{C}'' = \emptyset$.

In the end, we go through the sequence formed, and remove any $\emptyset$ from the sequence. This sequence is $\mathcal{S}''$.

We show that $\mathcal{S}''$ is a solution of B. By definition of $\mathcal{S}''$, the number of horizontal/vertical cuts in $\mathcal{S}''$ is at most the number of horizontal/vertical cuts of $\mathcal{S}'$. We have argued that each element in $\mathcal{S}'$ is a cut of B. What remains is to show that $\mathcal{S}''$ sets to zero all non-zero entries of B. It is enough to show that all non-zero entries of $B[X, [m]]$ is set to zero by $\mathcal{S}''$. The solution $\mathcal{S}'$ has the property that all non-zero entries of rows in X are set to zero only by vertical cuts. Therefore, the only cuts touching X that are possible in $\mathcal{S}'$ are as described in $(3-7)$. The cuts of $\mathcal{S}''$ described in $(3-7)$ also show how to cover, for each column, all non-zero entries of rows in X. The description of the cuts are such that the endpoints of all the cuts lie in $U_1 \cup U_2$. By definition, $i_0 \notin U_1 \cup U_2$. Therefore, $\mathcal{S}''$ is a solution of B, where all non-zero entries of X are covered by vertical cuts and there is no vertical cut with an endpoint in R_{i_0}. A formal argument can also be given using induction on the length of the sequence and will appear in the full version of the paper. ☐

$\mathcal{S}''$ is a solution for (B, k_h, k_v), such that there are no endpoints in R_{i_0}. Therefore, $\mathcal{S}''$ is also a solution for (B', k_h, k_v) and we are done with the forward direction.

The argument for the reverse direction is again similar. Because of Claim 1, we know that if (B', k_h, k_v) is a YES instance, then there is a solution $\mathcal{S}^*$ such that rows of X are set to zero only by vertical cuts. By definition of a cut, it must be true that no vertical cut has an endpoint on R_{i_0}. Consider a column j and the entry $B[i_0, j]$ which is non-zero. By definition of row-block, for any row of X, the entry in the column j is $B[i_0, j]$. In particular, $B[i_0+1, j] = B[i_0, j] = B[i_0-1, j]$. If there is a vertical cut that sets both $B'[i_0+1, j]$ and $B'[i_0-1, j]$ to zero in $\mathcal{S}^*$, then the same cut can be used to set $B[i_0, j]$ to zero in B. Suppose not. Then, it must be the case that the vertical cut that sets $B'[i_0+1, j]$ to zero, ends at $B'[i_0+1, j]$. For a solution in B, we extend this cut till $B[i_0, j]$. It is clear that this is a cut in B. In this way, all non-zero entries of R_{i_0} are set to zero.

Thus, we show that the Reduction Rule is safe. ☐

For ease of notation we refer to the reduced instance as (B, k_h, k_v) as well. Due to the exhaustive application of Reduction Rule 2, we can assume that the size of each row-block in B is upper bounded by $3(k+1)$. Next, we give a bound on the number of row-blocks in a YES instance.

Reduction Rule 3 *If the number of row-blocks in B is at least $2k+1$ then (B, k_h, k_v) is a NO instance.*

Lemma 2 *Reduction Rule 3 is safe.*

Proof. Consider a sequence $\mathcal{S}$ which is a solution for B. Let G be a path where each block B is represented by a vertex v_B, and neighbouring blocks are made adjacent. Then, the number of blocks of B is equal to the number of vertices in G. Suppose a cut of $\mathcal{S}$ sets all non-zero entries of a block B to zero, then we delete the vertex v_B and make its two neighbours in G adjacent. Similarly, if two adjacent vertices u, v in G represent the same block, then we delete one vertex u and make v adjacent to the other neighbour of u. Any horizontal or vertical cut of $\mathcal{S}$ can either set the non-zero entries of at most one block B to zero, or can make a block same as its two neighbouring blocks. Correspondingly, one of the two things can happen: (i) at most one vertex v_B of G gets deleted and of the two neighbours v_{B_1} and v_{B_2}, at most one gets deleted, or (ii) one block B becomes the same as its two neighbouring blocks B_1 and B_2, which results in the deletion of v_B and v_{B_1} and in the addition of an edge between v_{B_2} and the other neighbour of v_{B_1}. Thus because of any horizontal or vertical cut, at most two vertices could be removed from G.

Since $\mathcal{S}$ is a solution for B, in the end, the graph G should become an empty graph. As there are at most k cuts in $\mathcal{S}$, this implies that there were at most $2k$ vertices in G to start with. Thus, if at most k cuts are allowed, then the number of blocks is at most $2(k_h + k_v + rk_d)$.

□

Lemma 3 *When none of the above mentioned Reduction rules are applicable, the number of rows in the board of the reduced instance is $\mathcal{O}(k^2)$.*

Proof. Since Reduction Rule 2 does not apply, each row-block has size at most $3(k+1)$. Also, by Reduction Rule 3, the number of row-blocks is at most $2k$. Therefore, the total number of rows in B is at most $\mathcal{O}(k^2)$. □

Similarly, we can also bound the number of columns of B, using the same arguments. Using Observation 2 and 1, we get the following theorem.

Theorem 4 B&S+ *admits a kernel with $\mathcal{O}(k^2)$ rows and columns, and at most $\mathcal{O}(k^3)$ buttons. Furthermore, the problem can be solved in time $2^{\mathcal{O}(k \log k)}k^{\mathcal{O}(1)} + (n+m)^{\mathcal{O}(1)}$.*

4 Kernel for a $n-$by-1 board

In this section, we consider the *1-Dim-B&S* problem and give a linear kernel, parameterized by the number of cuts.

Let (B,k) be the input instance of *1-Dim-B&S*. In this case too, Reduction Rules 1 and 3 apply. In addition to that, in this special case, we have a more refined bound on the size of a row-block.

Reduction Rule 4 *If $B[\{i,i+1,\dots,i+p\},1]$ is a row-block such that $|p| \geq 2$, then set $B[i+p,1] = 0$. Let B' be the resulting board. Then the resulting instance is (B',k).*

Lemma 5 *Reduction Rule 4 is safe.*

Proof. [Proof Sketch] Observe that any minimum solution of B is also a solution of B'. Also any solution to B' can be extended to a solution to B by expanding a cut that touches one of the entries in $B[\{i,i+1\},1]$. □

Reduction Rule 4, together with ideas motivated from [1] about "context free language" gives us the following result.

Theorem 6 *Let (B,k) be the reduced instance with respect to Reduction Rules 1, 3 and 4. Then there are at most $4k$ rows in B. Moreover,* 1-Dim-B&S *on this reduced insance can be solved in $2^{\mathcal{O}(k)}k^{\mathcal{O}(1)}$ time.*

Proof. Since Reduction Rule 1, 3 and 4 are not applicable, any row-block of B has at most 2 rows, there are at most $2k$ row-blocks and the number of rows with zero entries is zero. Hence, the number of rows/buttons in B is at most $4k$. Using Observation 1, we obtain an algorithm for this problem that runs in $2^{\mathcal{O}(k \log k)}k^{\mathcal{O}(1)}$ time. In fact, we can obtain a $2^{\mathcal{O}(k)}k^{\mathcal{O}(1)}$ time algorithm for this problem. We know that at most k cuts are allowed. Suppose there is a solution $\mathcal{S}'$ with $k' \leq k$ cuts. By the definition of cuts, there are exactly $2k'$ cells of B which are starting and ending points for these cuts in the potential solution $\mathcal{S}'$. We guess these $2k'$ cells, and among them, we guess which cells are starting cells and which are ending cells. There are $\binom{4k}{2k'}2^{2k'} \leq 2^{6k}$ such guesses. For each guess, we execute two phases: (i) Pair up each starting cell with an ending cell and define an ordering on such pairs, (ii) verify whether the ordering ensures that each pair is a cut of the board when its turn comes.

In the first phase, we try to build a sequence $\mathcal{S}$ of pairs, where each pair consists of a starting cell and an ending cell of the guess. To start with, $\mathcal{S} = \emptyset$. First, if there is any guessed position with its entry equal to zero, then we immediately terminate the guess. Suppose the topmost guessed cell is an ending cell, then we imediately terminate this guess. Otherwise, let S be a stack. We push in the guesses starting cells from the top row downwards, till we find an ending cell. When we find an ending cell e, we pop the last starting cell s in S. Suppose the (non-zero) entries of e and s are not equal. Then we terminate this guess. Otherwise, we consider s and e as the next cut in $\mathcal{S}$. We do this till the stack S becomes empty. The number of guessed starting cells and ending cells are the same. Also, each starting cell is pushed and popped exactly once from S. Then, within k' pops and k' pushes on the stack S, S becomes empty.

In the second phase of the algorithm, we consider a pair (s,e) in the sequence $\mathcal{S}$. If all the entries in rows between s and e are either of the same non-zero value, or equal to zero, then the pair (s,e) corresponds to a cut. We set all the entries in rows between s and e to zero, and continue with the next pair in $\mathcal{S}$. If not, then we terminate this guess of starting and ending cells. If, in the end, we have set all entries in B to zero, then we output $\mathcal{S}$ as a solution, with at most k cuts, for B. We have verified that each pair $(s,e) \in \mathcal{S}$ can be thought of as a cut for the board B as well.

We argue the correctness of this algorithm. First, suppose the algorithm finds a guess that outputs a $\mathcal{S}$. By the description of the first phase of the algorithm, in the second phase, no starting or ending cell is set to zero due to any other cut of $\mathcal{S}$. Then $\mathcal{S}$ is a solution for B with at most k' cuts.

On the other hand, assume that there is a solution $\mathcal{S}$ of $k' \leq k$ cuts for the board. Let E be the set of starting and ending cells of the cuts in $\mathcal{S}$. The rows of B induce an ordering, $\mathcal{I}$, among the cells in E. Let this ordering give higher index to cells in rows with higher index. Let s and e be two consecutive cells in E under $\mathcal{I}$, such that s is a starting cell and e is an ending cell. Then it must be the case that s and e are the starting and ending cells of the same cut in $\mathcal{S}$. Otherwise, without loss of generality, let the cut C starting at s be before the cut C' ending at e in the sequence $\mathcal{S}$. Suppose e' is the ending cell of C. It must be the case that e' lies in a row below e. Then, by definition, the cut C should have set e to zero. This contradicts the definition of C', where the starting and ending cells must be non-zero at the time the cut is made. Since, s and e are consecutive in E, it is also necessary that all the non-zero entries in the rows between s and e must be set to zero by C itself, and by no other cut. Therefore, all the entries in the board B, between the rows s and e must either be the same non-zero value, or equal to zero. Now, consider the solution $\mathcal{S}$. In one of the guesses of starting and ending cells, we should have guessed the cells of E for the solution $\mathcal{S}$. The first cell in E must be a starting cell of a cut in $\mathcal{S}$. By the arguments above, the pairs created by the first phase of the algorithm are in fact the correct pairs for cuts in $\mathcal{S}$. The ordering $\mathcal{S}'$ we construct might be different from $\mathcal{S}$. Because of the pairing of the $2k'$ cells of E, there are at most k' cuts in $\mathcal{S}'$. Therefore, $\mathcal{S}'$ is a required solution for B.

This completes the proof of correctness of this algorithm. The running time of this algorithm is $2^{\mathcal{O}(k)}k^{\mathcal{O}(1)}$. □

Using similar arguments, if B is a $1-$by$-m$ matrix, we can apply similar reduction rules and upper bound m by $4k$.

5 Kernel for a $n-$by$-m$ board using diagonal cuts

In this section, we consider the *B&S* problem, where the allowed cuts can be horizontal, vertical as well as diagonal. The only constraint is that a diagonal cut can be of length at most r. This problem is NP-complete, because of the NP-completeness of *B&S+* [5].

We show that, given $k_h + k_v + k_d + r$ as a parameter, the *B&S* problem has a polynomial kernel. The strategy is the following: we first bound the size of each row-block (Reduction Rule 5 and 6) and then the number of row-blocks in a YES instance (Reduction Rule 7). Let $k = k_v + k_h + k_d$. First, we make an observation regarding diagonal cuts.

Observation 3 *A diagonal cut in B can set at most r non-zero entries to zero, and hence, can touch at most r rows and r columns.*

Reduction Rule 5 *Let (B, k_h, k_v, k_d, r) be an instance of* B&S. *Consider a maximal set $Y = \{i, i+1\ldots, p\}$ of consecutive rows of B that have only zero entries. If there are at least $r+2$ rows in this maximal set, then other than the first $r+1$ rows, we delete the rest of the rows from the board. This gives us a new board B', which is an $(n-(Y-r+1)) \times m$ matrix. The resulting instance is (B', k_h, k_v, k_d, r).*

Lemma 7 *Reduction Rule 5 is safe.*

Proof. [Proof Sketch] The key idea here is Observation 3. From this observation and the fact that Y is a set of zero rows and $|Y| \geq 2$, none of the rows in Y are touched by diagonal cuts. Also no minimum solution will have a horizontal cut touching any of the rows of Y. Thus, the rows of Y are touched only by vertical cuts (if at all touched) and any vertical cut that touches a row of Y touches all rows of Y (because end points of a cut are non-zero entries). Thus, deleting the extra (anything more than $r+2$) rows preserves the solution, as the cuts touching the removed rows can easily be shrunk for the forward direction and expanded for the reverse direction. □

We now give a bound on the size of each block in B. To make the arguments notationally clear, we say that a cut *crosses* a row, if it not only touches this row but also the row above and the row below.

Reduction Rule 6 *Let $R_i, R_{i+1}, \ldots, R_p$ be a row-block such that $p-i+1$ is at least $2(rk_d+1) + (rk_d+k_h)(r+2) + 2k_v(r+2) + 2$. Let i_0 be such that $i + rk_d + k_v(r+1) < i_0 \leq p - (rk_d+1) - k_v(r+1)$. Also, let R_{i_0} have non-zero entries. Let $B^{'}$ be a matrix obtained from B by setting all entries of R_{i_0} to 0. Then the resulting instance is $(B^{'}, k_h, k_v, k_d, r)$.*

Lemma 8 *Reduction rule 6 is safe.*

Proof. As a sanity check, let us first see that such an i_0 exists where there is some non-zero entry. By definition of i_0, there are at least $(rk_d+k_h)(r+2)+2$ rows that can be assigned to i_0. Since Reduction Rule 5 is not applicable, there must be at least one row among them that has at least one non-zero entry. Now, we prove the correctness of this Reduction Rule.

First, suppose (B, k_h, k_v, k_d, r) is a YES instance of *B&S*. Let $X = \{i, i+1, \ldots, p\}$ be the rows forming the row-block. Let $X_U = \{i, i+1, \ldots i+rk_d\}$ and $X_D = \{p-(rk_d+1)+1, \ldots, p\}$ be the first and last rk_d+1 rows of X respectively. By definition, i_0 does not belong to either X_U or X_D. Next, consider a solution $\mathcal{S}$ of B. By Observation 3 and by the budget on the number of diagonal cuts, at most rk_d rows of X_U and X_D are touched by diagonal cuts. Then, by pigeonhole principle, X_U has a row u such that no diagonal cut crosses u. Similarly, X_D has a row d such that no diagonal cut crosses d. Let $X_m = \{u+1, \ldots, d-1\}$. First, we show that there is a solution $\mathcal{S}'$, with at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts of length r, such that all non-zero entries in X_m are set to zero by vertical cuts only.

Claim 3 *There exists a solution $\mathcal{S}'$ with at most k_h horizontal cuts, k_v vertical cuts, and k_d diagonal cuts of length r, such that every non-zero entry in $R_i, \ldots, R_p$ is set to zero by a vertical cut in $\mathcal{S}'$.*

The proof of Claim 3 can be obtained from the proof of Claim 1 by replacing X by X_m. Next, we show that there is a solution $\mathcal{S}''$, where every row in X_m has its non-zero entries set to zero by vertical cuts alone, and where no vertical cut ends at row R_{i_0}.

Claim 4 *There is a solution $\mathcal{S}''$, where every row in X_m has its non-zero entries set to zero by vertical cuts alone, and where no vertical cut ends at row R_{i_0}. Moreover, there are at most k_h horizontal cuts, k_v vertical cuts and k_d diagonal cuts in $\mathcal{S}''$.*

Proof. [Proof Sketch] By choice of i_0, there are at least $k_v(r+2)$ rows of X_m above R_{i_0} and at least $k_v(r+2)$ rows below R_{i_0}. Let U_1 be the first $k_v(r+2)$ rows in X_m, and U_1' be the subset of rows of U_1 which have non-zero entries. By Pigeonhole principle, there are at least k_v rows in U_1'. Similarly, we can define U_2 as the last $k_v(r+2)$ rows in X_m, and U_2' as the subset of rows of U_2 which have non-zero entries. The number of rows in U_2' must be at least k_v. To construct $\mathcal{S}''$ using $\mathcal{S}'$, we follow exactly the construction of $\mathcal{S}''$ in the proof of Claim 2 modulo the following changes. Replace X by X_m, i by u, p by d, U_1 by U_1' and U_2 by U_2' in the proof of the Claim 2. ☐

$\mathcal{S}''$ is a solution for (B, k_h, k_v), such that there are no endpoints in R_{i_0}. Therefore, $\mathcal{S}''$ is also a solution for (B', k_h, k_v) and we are done with the forward direction.

The argument for the reverse direction is similar. We show that if (B', k_h, k_v, k_d, r) is a YES instance, then there is a solution $\mathcal{S}^*$ such that rows of X_m are set to zero only by vertical cuts. By definition of a cut, it must be true that no vertical cut has an endpoint on R_{i_0} of B'. Consider a column j and the entry $B[i_0, j]$ which is non-zero. By definition of row-block, for any row of X_m, the entry in the column j is either $B[i_0, j]$ or an empty space. Let i_u and i_d be the highest row just above and lowest row just below i_0, respectively, where $B[i_u, j] = B[i_0, j] = B[i_d, j]$. If there is a vertical cut that sets both $B'[i_u, j] = B[i_u, j]$ and $B'[i_d, j] = B[i_d, j]$ to zero in $\mathcal{S}^*$, then the same cut can be used to set $B[i_0, j]$ to zero in B. Suppose not. Then, it must be the case that the vertical cut that sets $B'[i_u, j] = B[i_u, j]$ to zero, ends at $B'[i_u, j] = B[i_u, j]$. For a solution in B, we extend this cut till $B[i_0, j]$. In B, this is a cut. In this way, all non-zero entries of R_{i_0} are set to zero. ☐

Reduction Rule 7 *If the number of row-blocks is at least $2(k_h + k_v + rk_d) + 1$, then (B, k_h, k_v, k_d, r) is a NO instance.*

The proof of safeness of Reduction Rule 7 is exactly like Lemma 2 with an addition that even though any vertical or horizontal cut can remove at most 2 vertices from G, any diagonal cut can set the non-zero entries of at most m blocks to zero and can make at most r blocks same as its two neighbouring blocks. Thus, because of any diagonal cut, at most $2r$ vertices could be removed from G.

Putting the above results together, we conclude that when none of the reduction rules are applicable, $n \leq (2(rk_d + 1) + (rk_d + k_h + 2k_v)(r+2) + 2)2(k_h + k_v + rk_d) = \mathcal{O}(r^3k^2)$. We can apply analogous reduction rules to bound the number of columns (m) by $\mathcal{O}(r^3k^2)$. Combining this with Observations 1 and 2, we complete the proof of Theorem 9.

Theorem 9 B&S *admits a kernel with $\mathcal{O}(r^3k^2)$ columns and rows and $\mathcal{O}(r^3k^3)$ buttons. Furthermore, the problem can be solved in time $2^{\mathcal{O}(k(\log r + \log k))} + (n + m)^{\mathcal{O}(1)}$.*

6 Conclusion

Here are a few open questions that we would like to address as an extension of the current work. The parameterized complexity of *Buttons & Scissors* game, parameterized by the number of cuts and the classical complexity of *Buttons & Scissors* game, where the board is an $n \times 1$ board, and the aim is to determine the minimum number of cuts required for a solution of the board.

References

[1] K. Burke, E. Demaine, R. Hearn, A. Hesterberg, M. Hoffman, H. Ito, I. Kostitsyna, M. Loffler, Y. Uno, C. Schmidt, R. Uehara, and A. Williams. Single-player and two-player buttons & scissors games. In *Proceedings of the 18th Japan Conference on Discrete and Computational Geometry and Graphs*, page 2.

[2] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms.* Springer, 2015.

[3] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer-Verlag, 1997.

[4] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[5] H. Gregg, J. Leonard, A. Santiago, and A. Williams. Buttons & scissors is np-complete. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*, 2015.

[6] R. Niedermeier. Invitation to fixed-parameter algorithms. 2006.

Minimizing Uncertainty through Sensor Placement with Angle Constraints

Ioana O. Bercea[*] Volkan Isler[†] Samir Khuller[*]

Abstract

We study the problem of sensor placement in environments in which localization is a necessity, such as ad-hoc wireless sensor networks that allow the placement of a few anchors that know their location or sensor arrays that are tracking a target. In most of these situations, the quality of localization depends on the relative angle between the target and the pair of sensors observing it. In this paper, we consider placing a small number of sensors which ensure good angular α-coverage: given α in $[0, \pi/2]$, for each target location t, there must be at least two sensors s_1 and s_2 such that the $\angle(s_1 t s_2)$ is in the interval $[\alpha, \pi - \alpha]$. One of the main difficulties encountered in such problems is that since the constraints depend on at least two sensors, building a solution must account for the inherent dependency between selected sensors, a feature that generic Set Cover techniques do not account for.

We introduce a general framework that guarantees an angular coverage that is arbitrarily close to α for any $\alpha <= \pi/3$ and apply it to a variety of problems to get bi-criteria approximations. When the angular coverage is required to be at least a constant fraction of α, we obtain results that are strictly better than what standard geometric Set Cover methods give. When the angular coverage is required to be at least $(1 - 1/\delta) \cdot \alpha$, we obtain a $\mathcal{O}(\log \delta)$- approximation for sensor placement with α-coverage on the plane. In the presence of additional distance or visibility constraints, the framework gives a $\mathcal{O}(\log \delta \cdot \log k_{\mathsf{OPT}})$-approximation, where k_{OPT} is the size of the optimal solution. We also use our framework to give a $\mathcal{O}(\log \delta)$-approximation that ensures $(1 - 1/\delta) \cdot \alpha$-coverage and covers every target within distance $3R$.

1 Introduction

Localization is an important necessity in many mobile computing applications. In ad-hoc wireless sensor networks, it centers around the ability of nodes to self localize using little to no absolute spatial information. When mobility is considered, the problem becomes that of tracking a moving target through a sensor network in which a set of sensors must combine measurements in order to detect the location of the target.

When a large number of sensors are deployed, it becomes impractical to equip all of them with the capability of localizing themselves with respect to a global system (such as through GPS). From this perspective, a commonly used technique is to employ a small number of anchors (or beacons) that know their location and are capable of transmitting it to the other nodes seeking to localize themselves [27]. Alternatively, sensors such as cameras or microphone arrays placed in the environment can collect measurements which can then be used to estimate the locations of objects of interest. In some of the most popular scenarios, each target seeking to localize itself has access to Euclidean distances and/or angular measurements (bearing) *relative* to the sensors that are in its vicinity. When exact distances or bearings from two sensors to a target are known, localization can be easily performed through the process of triangulation. In practice, however, the inherent sensor measurements are noisy and several models of uncertainty have been proposed [4].

From a geometric perspective, a common benchmark for estimating uncertainty is the Geometric Dilution of Precision (GDOP). This benchmark investigates how the relative geometry between sensors and target nodes amplifies measurement errors and affects the localization error. Savvides et al. [23, 24] observe that the error is largest when the angle θ between two sensors and the target node is either very small or close to π. The analysis of Kelly [14] further shows that, when triangulation is used, this angle contributes to the GDOP at a fundamental level. When distance measurements are used in triangulation, the GDOP is proportional to $1/|\sin\theta|$. When angular measurements are used, the GDOP is proportional to $d_1 \cdot d_2/|\sin\theta|$, where d_1 and d_2 are the distances from the sensors to the target. Intuitively, each measurement becomes a constraint that restricts the set of possible locations of the target and the quality of localization depends on both the area and the shape of all intersections. As seen in Figure 1, when the angle θ is close to 0 or π, the feasible set becomes unconstrained and the error unbounded. In particular, when the sensors and the target are collinear, localization is impossible.

Inspired by these observations, our paper focuses on the geometry of sensor deployment and asks the question of where should the sensors be placed so as to en-

[*]Department of Computer Science, University of Maryland, College Park, {ioana,samir}@cs.umd.edu

[†]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, isler@cs.umn.edu

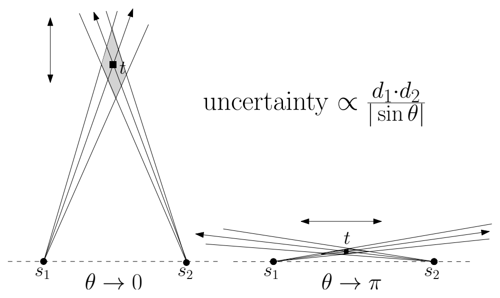


Figure 1: When bearing information is used to determine a target's location, the measurement corresponds to a cone centered at the true bearing from the sensor to the target. Given sensors s_1, s_2 and target t, let $\theta = \angle s_1 t s_2$, $d_1 = d(s_1, t)$ and $d_2 = d(s_2, t)$. When the sensors are too far from the target, the uncertainty becomes unbounded. If θ is small, the y-direction becomes unconstrained. If θ is large, the x-direction becomes unconstrained.

sure that the GDOP is below a given threshold at all target locations? To this end, we define an angular constraint which we call α-*coverage*: given a parameter $\alpha \in [0, \pi/2]$, each target at position t must be assigned two distinct sensors (at positions s_1 and s_2) such that the angle $\theta = \angle s_1 t s_2$ is in the range $[\alpha, \pi - \alpha]$ (i.e. neither too small nor too big). We then frame the problem of sensor placement as a bicriteria optimization problem: given a set of possible sensor and target locations, we wish to select the smallest number of sensors that provide α-coverage for all target locations. We address these variants from a theoretical perspective and present a general algorithmic framework that specifically addresses the angular constraint and iteratively obtains better angular guarantees at the expense of larger solution sizes.

Our model. Formally, we consider the two dimensional model in which the set of candidate sensor locations is a discrete set $X \subseteq \mathbb{R}^2$ and the set of target locations is a discrete set $T \subseteq \mathbb{R}^2$. We have chosen to discuss this discrete setting (instead of the continuous one) because we consider it to be more theoretically rich. If we could choose sensor locations anywhere in the plane, we could essentially impose a grid on the plane and use a constant number of sensors per cell to cover the targets (without violating the angular constraints). The analysis would then use k-CENTER or ART GALLERY techniques to obtain constant factor approximations.

Given a parameter $\alpha \in [0, \pi/2]$, we say that an (unordered) pair (s_1, s_2) α-covers a target t if $\theta = \angle s_1 t s_2$ is in the range $[\alpha, \pi - \alpha]$. Notice that the higher the value of α, the smaller the range of values that θ can take. Our algorithmic framework applies to several sensor coverage problems. First, we define the **Minimum Sensor Placement with α-coverage** ($\alpha-$Ang) problem, which asks for the smallest set of sensors that α-covers T. We then consider its clustering variant, in which we additionally require that the sensors be within a given range R of the target (corresponding to a finite sensing range scenario). We call this the **Minimum Sensor Placement with (α, R)-coverage** ($(\alpha, R)-$AngDist) problem and say that a pair of sensors (α, R)-covers a target if both sensors respect the constraints. Finally, we consider a version of the ART GALLERY problem, in which the target must be visible to both sensors. This problem was first introduced by Efrat, Har-Peled and Mitchell [8] which discussed the case in which the sensors (contained in a region P) are required to α- guard targets that are contained in a smaller region $Q \subseteq P$. Two sensors s_1, s_2 α-guard a target t if they both see the target and (s_1, s_2) α-covers t. Given a sensor $s \in X$ and a target $t \in T$, we say that s sees t if the segment connecting the two does not cross the boundary of P (i.e. t is within line-of-sight of s). We henceforth refer to this problem as the **Art Gallery with α-coverage** ($\alpha-$ArtAng) problem.

Related work. When it comes to sensor coverage problems, extensive work has been done although surprisingly few results discuss α-coverage. Notable exceptions are the work of Efrat, Har-Peled and Mitchell [8], Tekdas and Isler [25] and Isler, Khanna, Spletzer and Taylor [13]. As mentioned before, Efrat et al. [8] introduce the $\alpha-$ArtAng problem in which two sensors are required to α-guard a target. They present a $\mathcal{O}(\log k_{\mathsf{OPT}})$-approximation algorithm that guarantees $\alpha/2$-coverage, where k_{OPT} is the size of the optimal solution. Their main subroutine is similar to an algorithm for the ART GALLERY problem that first imposes a grid Γ on P and chooses guards located at vertices of Γ. We note however, that such a step is not necessary in our case, since X is already a discrete set. Their algorithm runs in time $\mathcal{O}(n k_{\mathsf{OPT}}^4 \log^2 n \log m)$, where n is the number of vertices of P and m is the number of points in $\Gamma \cap P$ (i.e. possible sensor locations). In contrast, we present a framework that achieves $(1 - 1/\delta) \cdot \alpha$-coverage, approximates the number of sensors by $\mathcal{O}(\log \delta \cdot \log k_{\mathsf{OPT}})$ and runs in time $\mathcal{O}(\log \delta \cdot k_{\mathsf{OPT}} \cdot mn \log m)$, for any $\delta \geq 1$ and $\alpha \leq \pi/3$. In our case and throughout the rest of the paper, n represents the number of targets.

Tekdas and Isler [25] formalize the angle constraint by requiring that the uncertainty $d_1 \cdot d_2 / |\sin\theta|$ computed by Kelly [14] be smaller than a certain threshold U. When the targets are contained in some subset of the plane and the sensors can be placed anywhere (continuous case), they present a 3-approximation with maximum uncertainty $\leq 5.5U$. We note that $(\alpha, R)-$AngDist is a generalization of the above problem in the sense in which an algorithm for $(\alpha, R)-$AngDist can be used in approximately solving the former. Finally, Isler et al [13] consider the case in which the sensor locations are already given and one must compute an assignment of sensors to targets that minimizes the total sum of errors. In addition, they require that each sensor be used in tracking only one target. The version relevant to our problem is when the error is defined as $1/\sin\theta$. In the case in which the sensors are equally spaced on a circle, they present a 1.42-approximation that also applies to minimizing the maximum error.

Our contributions. For the case of $\alpha \leq \pi/3$, we provide a general bi-criteria framework that approximates the angular coverage to arbitrary precision while guaranteeing a good approximation in the size of the solution. Specifically, for any $\delta > 1$, we propose an iterative method that guarantees $(1 - 1/\delta) \cdot \alpha$-coverage and approximates the solution size by $\mathcal{O}(\log\delta)$ for $\alpha-\mathsf{Ang}$ and $\mathcal{O}(\log\delta \log k_{\mathsf{OPT}})$ for $(\alpha, R)-\mathsf{AngDist}$ and $\alpha-\mathsf{ArtAng}$. When the polygon in $\alpha-\mathsf{ArtAng}$ is allowed to have h holes, we obtain a $\mathcal{O}(\log\delta \cdot \log k_{\mathsf{OPT}} \cdot \log h)$-approximation. It is worthwhile to note that the main technical theorem of the framework refers solely to the angle coverage constraint and as such, could be applied to a variety of other problems as long as the other constraints (such as distance or line-of-sight visibility) define a good set system (one with constant VC dimension, for example).

In addition, we present further approximations for $(\alpha, R)-\mathsf{AngDist}$. We relax the distance constraints from R to $3R$ and reduce the approximation factor of the solution size from $\mathcal{O}(\log\delta \cdot \log k_{\mathsf{OPT}})$ to $\mathcal{O}(\log\delta)$, while keeping the angular coverage at $(1-1/\delta)\cdot\alpha$. We achieve this by using a more involved technique of employing ϵ-nets that we believe may be of independent interest.

We also consider the case in which $\alpha = 0$ and construct a set of optimal size that covers the targets within distance $(1 + \sqrt{3}) \cdot R$. This particular case remains relevant since it captures the spirit of fault tolerance by requiring two distinct sensors to be assigned to a target. We achieve our result by showing a $(1 + \sqrt{3})$-approximation for the more general Euclidean Fault Tolerant k-suppliers problem which improves on the existing 3-approximation by Khuller et al [15].

Discussion of existing techniques. In a more general context, we believe that angular constraints are interesting not just because they contribute to a new geometrical direction in sensor coverage problems. From a purely theoretical perspective, they also present the challenge of approximating an optimization problem whose objective function is linear in the number of chosen sensors but whose constraints depend on pairs of sensors jointly satisfying a condition.

In such a case, an algorithm that chooses pairs which satisfy the constraints might incur an overall cost that is quadratic in the objective function. For example, one natural way in which we can consider $\alpha-\mathsf{Ang}$ is as an instance of Set Cover. For each pair of sensors (s, s'), we can define $S_{(s,s')}$ to be the set of targets $t \in T$ such that (s, s') α-covers them. Set Cover asks for the smallest number of pairs whose union covers T. The generic greedy charging scheme for Set Cover gives us a solution of size at most $k^* \cdot \log n$, where k^* is the size of the optimal Set Cover solution. Notice, however, that k^* can be much larger than k_{OPT} (the size of the optimal set of sensors for $\alpha-\mathsf{Ang}$) and this could lead to a quadratic blowup in the size of the solution. In the worst case, greedy Set Cover could pick as many as $\binom{k_{\mathsf{OPT}}}{2} \cdot \log n$ sensors in its solution. This degenerate case might happen when we end up picking distinct pairs of sensors to cover each target while the optimal solution picks a much smaller set of sensors that collectively α-cover the targets.

Using the geometry of the problem, one could slightly improve the above approximation factor from $\mathcal{O}(k_{\mathsf{OPT}} \cdot \log n)$ to $\mathcal{O}(k_{\mathsf{OPT}} \cdot \log k_{\mathsf{OPT}})$. Specifically, one can show that each $S_{(s,s')}$ is induced by the symmetric difference of two circles. As such, these objects have constant Vapnik-Chervonenkis (VC) dimension [19] and allow for a $\mathcal{O}(\log k^*)$-approximation for Set Cover [10, 3]. Unfortunately, this only gets us a $\mathcal{O}(k_{\mathsf{OPT}} \cdot \log k_{\mathsf{OPT}})$-approximation guarantee. The persistent k_{OPT} factor in the approximation comes from the fact that the Set Cover framework cannot distinguish between sensors that help cover a lot of targets (locally) and sensors that, additionally, can also help cover more targets in conjunction with other sensors. In other words, it does not make use of the **global dependency** between sensors in order to get a small solution size.

In fact, such observations are more in the spirit of Label Cover type problems in which we need to assign labels to vertices of a graph but a specific labeling is considered feasible only when it satisfies certain edge constrains. Indeed, when considered in its full generality (i.e. points lie in arbitrary space and coverage is defined arbitrarily), the problem becomes a generalization of MinRep and, as such, incurs a hardness of approximation bound of $2^{\log^{1-\epsilon} n}$, for any $0 < \epsilon < 1$ unless NP $\subseteq$ DTIME$(n^{\text{polylog}(n)})$ [17]. Such occurrence of Label Cover in a natural setting is intriguing in its own right. Furthermore, it can conceivably model other instances in which the quality of a solution depends on pairs of elements jointly satisfying a condition, such as in pairwise feature selection for Machine Learning tasks [22, 6]. We defer the rest of the discussion to the extended version of the paper [1].

2 Algorithmic Framework

Let $m = |X|$ be the number of possible sensor locations and $n = |T|$ be the number of target locations. The underlying distance function will be the Euclidean ℓ_2

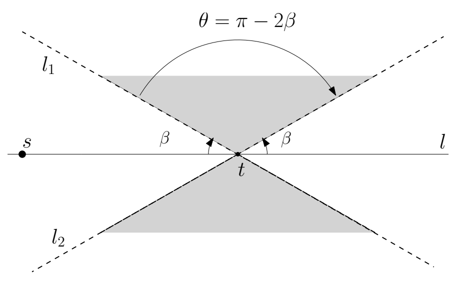


Figure 2: The set $R_t(s, \beta)$ is induced by the double-wedge generated by the lines l_1 and l_2 and has a central angle $\theta = \pi - 2\beta$.

metric. We consider (unordered) pairs of the form (s, s') where $s \neq s'$, $s \in S$, $s' \in S'$ and $S, S' \subseteq X$ are sets of sensors. We denote the set of such pairs as $S \times S'$. Formally, $S \times S' = \{(s, s') | s \neq s', s \in S, s' \in S'\}$. We say that the set of pairs $S \times S'$ α-covers t if there exists a pair $(s, s') \in S \times S', s \neq s'$, that α-covers t. When $S' = S$, we simply say that the set S α-covers t. A pair or a set of pairs α-covers a set T of targets when it α-covers each element of T. In addition, a pair or a set of pairs (α, R)-covers a set T of targets *within distance* R if, for at least one of the pairs that α-covers a target, the distance from both sensors to the target is $\leq R$.

We begin by considering a fixed set S of given sensors and asking the question of how should we pick a second set of sensors S' such that pairs of the form (s, s') with $s \in S$ and $s' \in S'$ will β-cover the set of targets (for a given β). Formally, given a target $t \in T$, a sensor $s \in X$ and angle parameter $\beta \in [0, \pi/2]$, we define the set $R_t(s, \beta)$ of feasible sensor locations that, together with s, β-cover t:

$$R_t(s, \beta) = \{s' \in X | (s, s') \ \beta\text{-covers } t\}$$

As seen in Fig. 2, this set is induced by two wedges centered around t. A *wedge* is defined as the intersection of two non-parallel half spaces in $\mathbb{R}^2$. In our case, we are interested in the two wedges defined by the lines that form angles of β with the line that passes through s and t. The union of these two wedges will be referred to as the *double-wedge* around t. The task of constructing a pair that β-covers t can now be reduced to that of finding a second sensor s' inside the double-wedge, i.e. once we find such a sensor, we are guaranteed that the pair (s, s') β-covers t.

Specifically, the set S' becomes a hitting set for the given collection of double-wedges (one for each target). Given a set system $\mathcal{F}(X, \mathcal{R})$, where X is the set of sensors and $\mathcal{R}$ is a collection of subsets (double-wedges) of X, a hitting set is a set $H \subseteq X$ that intersects every subset in $\mathcal{R}$ non-trivially. The HITTING SET problem asks for a hitting set of minimum cardinality. Our method constructs S' by approximately solving the HITTING SET problem, in which the double-wedges may be further restricted to intersect the circle of radius R centered at the targets (for $(\alpha, R)-$AngDist) or the visibility polygons of the corresponding targets (for $\alpha-$ArtAng). For this step, we employ known techniques using constant VC dimension and ϵ-net constructions.

When it comes to the analysis, the challenge is making sure that S' is not much larger than $k_{\mathsf{OPT}} = |S_{\mathsf{OPT}}|$ (the size of the optimal set of sensors) and therein lies the difficulty. We do this by showing that S_{OPT} itself is a hitting set, so the size of the optimal hitting set is smaller than k_{OPT}. Specifically, our structural theorem shows that given a set S of sensors that $(\alpha - 2\epsilon)$-covers T for some $\epsilon \in (0, \alpha/2]$, S_{OPT} must intersect the double-wedges induced by S around each target with $\beta = \alpha - \epsilon$:

Theorem 1 *Let $\epsilon > 0$ be such that $\alpha - \epsilon \leq \pi/3$ and $\epsilon \leq \alpha/2$. Given a set S that $(\alpha - 2\epsilon)$- covers T, let $T' \subseteq T$ be the set of targets that S does* ***not*** *$(\alpha - \epsilon)$-cover. Then the set of pairs $S \times S_{\mathsf{OPT}}$ $(\alpha - \epsilon)$-covers T'.*

When $\epsilon = \alpha/2$, we start with an arbitrary set S and recover the observation of Efrat et al. [8]. In order to get better than $\alpha/2$-coverage, the seed set S cannot be chosen arbitrarily. In fact, our proof crucially uses the fact that the sensors in S already $(\alpha - 2\epsilon)$-cover the targets. Given a pair (s_1, s_2) in S that $(\alpha - 2\epsilon)$-covers a target t, we show that one of the optimal sensors must be in $R_t(s_1, \alpha - \epsilon) \cup R_t(s_2, \alpha - \epsilon)$, i.e. it either makes a good pair with s_1 or with s_2. The restriction that $\alpha \leq \pi/3$ is used when $\epsilon < \alpha/2$ and guarantees that the union of the two double-wedges (for s_1 and s_2) is itself a double-wedge with a large enough central angle that it must intersect S_{OPT}.

By adding the set S' to S, we now obtain a larger set that $(\alpha - \epsilon)$-covers the targets. Iterating this procedure $\log \delta$ times, we guarantee $(1 - 1/\delta) \cdot \alpha$-coverage and approximate the size of the optimal solution by $\log \delta \cdot c$, where c is the approximation guarantee we get from solving the HITTING SET problem in each iteration (Section 2.2).

Finally, we note that while adding more sensors in order to obtain a better solution is a classical approach, the challenge is do so in a way that addresses the global dependency between sensors and does not incur a quadratic cost in each iteration. In this context, instead of looking for pairs in which both sensors are unknown, our framework looks for pairs in which one sensor is in S (i.e. already known) and the other one is in S'. This allows us to cast the task of constructing S' as a global optimization problem (HITTING SET) for which good approximations exist. Moreover, when $\alpha \leq \pi/3$, our main structural theorem allows us to bound the size of such hitting sets linearly in k_{OPT} (essentially shaving off the additional k_{OPT} factor from SET COVER).

2.1 Proof of Theorem 1

In order to prove Theorem 1, we essentially show that S_{OPT} must intersect at least one of the double-wedges generated by S around a given target t. First, notice that if a set S already $(\alpha-\epsilon)$-covers a target t, then we do not need to worry: S will continue to $(\alpha-\epsilon)-$cover t even when we add S' to S. We are therefore concerned with targets in T' that are not already $(\alpha-\epsilon)$-covered by S.

Fix such a target $t \in T'$ and let $s_1, s_2 \in S$ be any two sensors that $(\alpha-2\epsilon)$-cover t but do not $(\alpha-\epsilon)$-cover it. Our strategy will be to show that there exists $s^* \in S_{\mathsf{OPT}}$ such that either (s_1, s^*) $(\alpha-\epsilon)$-covers t or (s_2, s^*) $(\alpha-\epsilon)$-covers t, i.e. $s^* \in R_t(s_1, \alpha-\epsilon)$ or $s^* \in R_t(s_2, \alpha-\epsilon)$. The candidates will be $s_1^*, s_2^* \in S_{\mathsf{OPT}}$ where (s_1^*, s_2^*) is the optimal pair that α-covers t. We will show that s^* is either s_1^* or s_2^*. Intuitively, each of the double-wedges induced by s_1 and s_2 alone is not big enough to "capture" s_1^* or s_2^*. However, if $\angle s_1ts_2$ is in the range $[\alpha-2\epsilon, \pi-(\alpha-2\epsilon)]$, then the union D_t of these double-wedges will have a large enough central angle to guarantee that one of the optimal sensors is contained in it. In other words, at least one of the optimal sensors s_1^* or s_2^* together with either s_1 or s_2 will $(\alpha-\epsilon)$-cover t.

Let D_1 and D_2 be the double-wedges corresponding to $R_t(s_1, \alpha-\epsilon)$ and $R_t(s_2, \alpha-\epsilon)$, respectively. Notice that they have central angles $\theta_{D_1} = \theta_{D_2} = \pi - 2(\alpha-\epsilon)$. Let $\alpha' = \angle(s_1ts_2)$. We begin by first establishing that the union of these two double-wedges generated by s_1 and s_2 is a larger double-wedge.

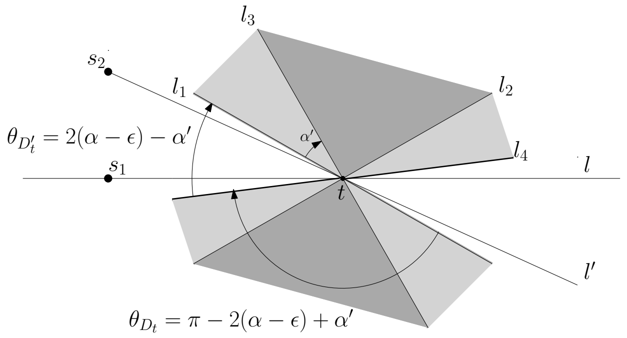
Lemma 2 *The union of the two double-wedges D_1 and D_2 is a larger double-wedge D_t centered at t with central angle $\theta_{D_t} = \pi - 2(\alpha-\epsilon) + \alpha'$.*

Proof. We refer the reader to Figure 3 for an intuitive explanation. Formally, let l be the line that passes through s_1 and t and let l_1 and l_2 the two lines that define D_1. Since $\angle(s_1ts_2) \notin [\alpha-\epsilon, \pi-(\alpha-\epsilon)]$, it follows that s_2 is not in D_1. Assume without loss of generality that s_2 is between the lines l and l_1 in the counterclockwise direction. The same proof follows for the other possible locations of s_2.

Now consider D_2 and let l_3 and l_4 be the defining lines through t, while l' is the line that passes through s_2 and t. Notice that D_1 and D_2 are identical except that D_2 is a rotated copy of the D_1. In other words, since $\angle(l, l') = \alpha'$, we also have that $\angle(l_1, l_3) = \alpha'$ and $\angle(l_2, l_4) = \alpha'$.

We will show that $(l_1, l_3) \leq \angle(l_1, l_2)$, and hence conclude that l_3 must lie between l_1 and l_2. Since $\angle(l_1, l_3) = \alpha' \leq \alpha - \epsilon$ (by choice of s_2) and $\angle(l_1, l_2) = \pi - 2(\alpha-\epsilon)$, we have that when $\alpha - \epsilon < \pi/3$:

$$\alpha - \epsilon \leq \pi - 2(\alpha-\epsilon).$$



l_3
s_2
l_1
l_2
$\theta_{D'_t} = 2(\alpha-\epsilon) - \alpha'$
α'
l_4
s_1
l
t
l'
$\theta_{D_t} = \pi - 2(\alpha-\epsilon) + \alpha'$

Figure 3: Since $\angle(s_1ts_2) = \alpha'$, D_2 is a rotation by α' of D_1 . Their union is another double-wedge D_t defined by l_1 and l_4 with central angle $\theta_{D_t} = \pi - 2(\alpha-\epsilon) + \alpha'$.

Therefore, the union of the two double-wedges D_1 and D_2 is a continuous double-wedge D_t determined by l_1 and l_4. It has central angle $\theta_{D_t} = \theta_{D_1} + \angle(l_2, l_4) = \pi - 2(\alpha-\epsilon) + \alpha'$.

□

The next step is to show that one of the two optimal sensors s_1^* and s_2^* must be in D_t. The intuition is that by making D_t have a large central angle, we ensure that the complement D'_t of D_t has such a small central angle that it would not be able to contain both s_1^* and s_2^*.

Lemma 3 *At least one of the two optimal sensors s_1^* and s_2^* assigned to t must be in D_t.*

Proof. Let D'_t be the complement of D_t. Notice that D'_t forms another double-wedge defined by l_1 and l_4 but that it does not actually contain points on these lines. Moreover, D'_t has a central angle $\theta_{D'_t} = \pi - \theta_D = 2(\alpha - \epsilon) - \alpha'$. Since s_1 and s_2 $(\alpha-2\epsilon)$-cover the target, and we are considering the case where s_2 is between l and l_1, we have that $\alpha' \geq \alpha - 2\epsilon$. Hence, we have that $\theta_{D'_t} \leq \alpha$. This implies that s_1^* and s_2^* cannot be both in the same wedge of D'_t without being exactly situated on the lines l_1 and l_4(i.e. in D_t). The other bad situation would be for them to be in different wedges of D'. But then the angle between them would be greater than θ_{D_t}. Since $\alpha' \geq \alpha - 2\epsilon$, we get that

$$\theta_{D_t} = \pi - 2(\alpha-\epsilon) + \alpha' \geq \pi - \alpha,$$

which would contradict the fact the $\angle(s_1^*ts_2^*) \in [\alpha, \pi - \alpha]$. In other words, at least one of the optimal sensors s_1^* and s_2^* must be in D_t. □

This concludes the proof of Theorem 1. At this point, it is worthwhile to notice that the requirement that $\alpha \leq \pi/3$ is relatively tight in this framework. When $\alpha - \epsilon > \pi/3$, both of the above claims fail. In particular, the central angle of D_1 and D_2 would be too small and their union would no longer correspond to a bigger double-wedge. Furthermore, it would no longer be true that such a union must intersect S_{OPT}.

2.2 Iterating to obtain $((1-1/\delta)\cdot\alpha)$-coverage

Given the technical lemmas from before that allow us to refine the angular coverage of a given seed set S, we can now develop a more general algorithm that constructs a new set that achieves $((1-1/\delta)\cdot\alpha)$-coverage for any $\delta > 1$. The idea is to iteratively apply the refinement step (by setting $S = S \cup S'$) $\log\delta$ times, first with $\epsilon = \alpha/2$, then with $\epsilon = \alpha/4$ etc. At the end of $\log\delta$ iterations, we have that the updated set S $((1-1/\delta)\cdot\alpha)$-covers T. The running time of the algorithm is $\log\delta$ times the time to find the appropriate hitting set plus the time it takes to find the starting set. This first set (denoted S_1) requires special care and depends on the problem at hand.

Notice that we require S_1 to 0-cover T but one can check that the proof of Theorem 1 follows in this case even when we do not have two distinct sensors 0-covering a target. Therefore, in the case of $\alpha-$Ang, it suffices to pick S_1 to consist of any sensor in X and get the following:

Theorem 4 *Given X, T, $\alpha \in [0,\pi/3]$ as above, we can find a set of sensors $S \subseteq X$ such that S $((1-1/\delta)\cdot\alpha)$-covers T and $|S| = \mathcal{O}(\log\delta)\cdot k_{OPT}$. The running time of the algorithm is $\mathcal{O}(\log\delta\cdot k_{OPT}\cdot m\log m)$.*

When it comes to the $(\alpha, R)-$AngDist problem, we require that the initial set S_1 has the property that each target is within distance R of at least one sensor in S_1. Without loss of generality, we can assume that $R = 1$ and then our problem becomes an instance of the Discrete Unit Disk Cover (DUDC) problem [5]. In DUDC, we are given a set $\mathcal{P}$ of n points and a set of $\mathcal{D}$ of m unit disks in the Euclidean plane. The objective is to select a set of disks $\mathcal{D}^* \subseteq \mathcal{D}$ of minimum cardinality that covers all the points. The problem is a geometric version of Set Cover and is NP-hard [9]. Nevertheless, several constant factor approximations have been developed and all could be used to compute a good approximation while balancing the trade-off between the approximation factor and the running time. For our purposes, we use the 18-approximation by Das et al [5] that has a runtime of $\mathcal{O}(n\log n + m\log m + mn)$. We note that better approximations are known, but using them in our framework could increase the total runtime. In each iteration, we increase the size of our set by $\mathcal{O}(\log k_{\mathsf{OPT}})\cdot k_{\mathsf{OPT}}$ and since $|S_1| \le 18\cdot k_{\mathsf{OPT}}$, we get the following:

Theorem 5 *Given X, T, α and R as above, we can find a set of sensors $S \subseteq X$ such that S $((1-1/\delta)\cdot\alpha)$-covers T within distance R and $|S| = \mathcal{O}(\log\delta\cdot\log k_{OPT})\cdot k_{OPT}$. The running time of the algorithm is $\mathcal{O}(\log\delta\cdot k_{OPT}\cdot mn\log m)$.*

For $\alpha-$ArtAng, we need to find a set of sensors $S_1 \subseteq X$ that guard T. To this extent, we can again employ the fact that the set of visibility polygons has finite VC-dimension [26]. Notice that finding a small S_1 that guards T corresponds to the hitting set problem for the set system made of sensors and visibility polygons of target locations. We therefore obtain a set of size $\mathcal{O}(\log k^*)\cdot k^*$, where k^* is the size of the smallest set of sensors from X that guard T. Since S_{OPT} also guards T, we have that $k^* \le k_{\mathsf{OPT}}$, so we are guaranteed to obtain a solution of size $\mathcal{O}(\log k_{\mathsf{OPT}})\cdot k_{\mathsf{OPT}}$. In each iteration, we increase the size of our set by $\mathcal{O}(\log k_{\mathsf{OPT}})\cdot k_{\mathsf{OPT}}$ and since $|S_1| = \mathcal{O}(\log k_{\mathsf{OPT}})\cdot k_{\mathsf{OPT}}$, we get the following:

Theorem 6 *Given polygons $Q \subseteq P$,X, T, and $\alpha \in [0,\pi/3]$ as above, we can find a set of sensors $S \subseteq X$ such that S $((1-1/\delta)\cdot\alpha)$-guards T and $|S| = \mathcal{O}(\log\delta\cdot\log k_{OPT})\cdot k_{OPT}$. The running time of the algorithm is $\mathcal{O}(\log\delta\cdot k_{OPT}\cdot mn\log m)$.*

We note that, in the case of $\alpha-$ArtAng, we improve on the result of Efrat et al [8], in that we approximate the α-coverage constraint to any constant factor while maintaining the same approximation factor of $\mathcal{O}(\log k_{\mathsf{OPT}})$. Moreover, our running times are comparable: $\mathcal{O}(nk^4_{\mathsf{OPT}}\log^2 n\log m)$ in [8] versus $\mathcal{O}(k_{\mathsf{OPT}}\cdot mn\log m)$ for our approximation. We note that their running time comes from the fact they do not directly use the bounded VC dimension of the set system. Instead, they use a previous algorithm designed by Efrat et al [7] for approximating the more general Art Gallery problem when the set of targets is restricted to vertices of that grid. When angle constraints are added, they adapt this algorithm to only consider vertices of the grid that also satisfy α-coverage. In our scenario in which targets have to be chosen from a discrete set, we do not need to impose a grid and can directly apply the Brönnimann and Goodrich algorithm [3]. For the case in which the sensors can be placed anywhere, their algorithm could be employed instead while maintaining the same approximation guarantees.

3 Other bi-criteria approximations for $(\alpha, R)-$AngDist

The geometric objects at the core of our method are wedges centered at targets whose central angles depend on α and ϵ. These ranges define the set of feasible locations from which we must choose a new set of sensors and are given as input to the subsequent Hitting Set problem. In the case of $(\alpha, R)-$AngDist, the distance constraints require that the sensors we pick also be within range R of the target, so our wedges become sectors through intersection with a disk of radius R centered at the target.

When it comes to solving the Hitting Set problem, we employ the instrumental results of Haussler and Welzl [10] and Brönnimann and Goodrich [3] which are based on the existence of good ϵ-nets for a given set system. We defer the intricacies of ϵ-nets to the long for of the paper[1] but mention now that, given an algorithm that computes in polynomial time an ϵ-net of size $\mathcal{O}((1/\epsilon) \cdot g(1/\epsilon))$, the algorithm of Brönnimann and Goodrich [3] returns a hitting set of size $\mathcal{O}(\tau \cdot g(\tau))$, where τ is the size of the optimal hitting set and g is a monotonically increasing sublinear function. Several good ϵ-net constructions are known when the underlying objects are geometrical. For the case of $(\alpha, R)-\mathsf{AngDist}$, we employ the canonical ϵ-net construction of Blumer et al [2] and Komlós et al [16] which exploits the fact that our ranges have constant VC dimension. This yields a $\mathcal{O}(\log k_{\mathsf{OPT}})$-approximation for Hitting Set on sectors of radius R.

In order to reduce the approximation factor, we consider relaxing the distance constraint and allowing the chosen sensors to be within distance $3R$ of the targets. In other words, we extend the radius of our sectors from R to $3R$. Inspired by the construction of Kulkarni and Govindarajan [18] for (unbounded) wedges, we then propose a deterministic rule for picking sensors and obtain a "relaxed" ϵ-net of size $\mathcal{O}(\frac{R_I}{R} \cdot \frac{1}{\epsilon})$, where R_I is the diameter of the largest enclosing ball of all possible sensor locations. The main difference between our construction and the one in [18] is the fact that the objects the latter considers are unbounded and, as such, allow for simpler grid-based constructions. In fact, this distinction is indeed the source of the additional $\frac{R_I}{R}$ factor that we incur in our bound.

To our knowledge, this is the first ϵ-net construction whose size depends linearly on $\frac{1}{\epsilon}$ and the ratio of the diameter of the input space to the size of the ranges (that is, when size can be appropriately defined). We note that the $\mathcal{O}(\frac{1}{\epsilon})$ construction of Pach and Woeginger [21] for translates of convex polygons does implicitly depend on solving the problem for points contained inside a bounded square. It is unclear, however, how to adapt their method for the case in which the ranges are sectors of similar radius but can have arbitrary central angles and orientations.

In order to overcome the dependency on $\frac{R_I}{R}$, we further employ the shifting technique of Hochbaum and Maass [11] to first partition our space into cells of bounded width and height and apply our ϵ-net construction to obtain good hitting sets in those restricted spaces. The analysis then yields an overall hitting set of size $\mathcal{O}(k_{\mathsf{OPT}})$ that achieves the desired $(\alpha - \epsilon)$-coverage and is within distance $3R$ of the targets. The complete argument is rather involved and we defer the exact details to extended version of the paper [1]. Formally, we get that:

Theorem 7 *Given X, T, $\alpha \leq \pi/3$ and R, we can find a set of sensors $S \subseteq X$ such that S $((1 - 1/\delta) \cdot \alpha)$-covers T within distance at most $3R$ and $|S| = \mathcal{O}(\log \delta) \cdot k_{\mathsf{OPT}}$. The running time of the algorithm is $\mathcal{O}(k_{\mathsf{OPT}} \cdot mn \log m \log n)$.*

Another interesting special case of $(\alpha, R)-\mathsf{AngDist}$ is the one in which $\alpha = 0$, since it requires us to place two distinct sensors within distance R of each target. A related problem is the fault tolerant k-suppliers problem as defined by Khuller et al [15] that requires us to select k suppliers such that each client has δ suppliers within an optimal distance r^* of it. While the objective in k-suppliers is to minimize the covering radius (as opposed to the number of sensors used), we will nevertheless exploit the connection and use it for $(\alpha, R)-\mathsf{AngDist}$. Under arbitrary metrics, Khuller et al. [15] develop a 3-approximation for fault tolerant k-suppliers: they select k sensors that are guaranteed to cover the clients within $3 \cdot r^*$. Karloff and then Hochbaum et al. [12] show that this factor is tight for the general k-suppliers problem (i.e. $\delta = 1$), unless P=NP. When the underlying space is $\mathbb{R}^d$ with the ℓ_2 metric, Nagarajan et al. [20] improve this factor to $(1 + \sqrt{3})$ for Euclidean k-suppliers. They crucially use the observation that three clients who are pairwise more than $\sqrt{3} \cdot r^*$ apart from each other can never be covered by the same supplier within distance r^* and reduce the problem of finding k suppliers to that of computing a minimum edge cover. A similar approach can be used in our case, except the structure of the optimal solution corresponds to a b-edge cover. In the long form of the paper [1], we present the details of the analysis and guarantee that δ suppliers are within distance $(1 + \sqrt{3})r^*$ of each client:

Theorem 8 *There exists a polynomial time $(1 + \sqrt{3})$-approximation algorithm for the Euclidean* fault tolerant k-suppliers *in any dimension for arbitrary $\delta \geq 1$.*

The k-suppliers technique minimizes the covering radius by relying on the existence of k suppliers that cover all the clients within a guess radius R. Given such a guess radius, the algorithm itself never picks more than k sensors (i.e. without explicitly knowing the value of k). The binary search technique of Hochbaum and Shmoys [12] is then used to obtain guarantees with respect to the optimal covering radius r^*. The algorithm hence starts with a guess R and returns a set of k sensors that cover everything within distance $(1 + \sqrt{3}) \cdot R$. In our case, $k = k_{\mathsf{OPT}}$, so we will always pick at most k_{OPT} sensors. Since we already know the value of R, we get the following result as well:

Theorem 9 *Given X, T, and R as above, we can find a set of sensors $S \subseteq X$ such that S 0-covers T within distance $R \cdot (1 + \sqrt{3})$ and $|S| = k_{\mathsf{OPT}}$, where k_{OPT} is the cardinality of the smallest set of sensors that 0-covers T*

within distance R. The running time of the algorithm is $\mathcal{O}(n^2 \log n(m + n \log n))$.

References

[1] I. O. Bercea, V. Isler, and S. Khuller. Minimizing uncertainty through sensor placement with angle constraints. *CoRR*, arXiv:1607.05791, 2016.

[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4), 1989.

[3] H. Brönnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.

[4] N. Cressie. *Statistics for spatial data*. John Wiley & Sons, 2015.

[5] G. K. Das, R. Fraser, A. López-Ortiz, and B. G. Nickerson. On the discrete unit disk cover problem. In *WALCOM: Algorithms and Computation*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011.

[6] S. Dreiseitl and M. Osl. Feature selection based on pairwise classification performance. In *Computer Aided Systems Theory-EUROCAST 2009*, pages 769–776. Springer, 2009.

[7] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[8] A. Efrat, S. Har-Peled, and J. Mitchell. Approximation algorithms for two optimal location problems in sensor networks. In *BroadNets'05*, pages 714–723 Vol. 1, Oct 2005.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York,USA, 1979.

[10] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In *Symposium on Computational Geometry*, pages 61–71, 1986.

[11] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, Jan. 1985.

[12] D. S. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, May 1986.

[13] V. Isler, S. Khanna, J. Spletzer, and C. Taylor. Target tracking with distributed sensors: The focus of attention problem. *Computer Vision and Image Understanding Journal*, (1-2):225–247, 2005.

[14] A. Kelly. Precision dilution in triangulation based mobile robot position estimation. In *Intelligent Autonomous Systems*, volume 8, pages 1046–1053, 2003.

[15] S. Khuller, R. Pless, and Y. J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(12):237 – 245, 2000.

[16] J. Komlós, J. Pach, and G. Woeginger. Almost tight bounds for ϵ-nets. *Discrete Comput. Geom.*, 7(2):163–173, Mar. 1992.

[17] G. Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001.

[18] J. Kulkarni and S. Govindarajan. New ϵ-net constructions. In *CCCG'10*.

[19] J. Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[20] V. Nagarajan, B. Schieber, and H. Shachnai. The Euclidean k-supplier problem. In *Integer Programming and Combinatorial Optimization*, volume 7801, pages 290–301. Springer Berlin Heidelberg, 2013.

[21] J. Pach and G. Woeginger. Some new bounds for epsilon-nets. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 10–15. ACM, 1990.

[22] E. Pekalska, A. Harol, C. Lai, and R. P. Duin. Pairwise selection of features and prototypes. In *Computer Recognition Systems*, pages 271–278. Springer, 2005.

[23] A. Savvides, W. Garber, S. Adlakha, R. Moses, and M. B. Srivastava. On the error characteristics of multihop node localization in ad-hoc sensor networks. In *Information Processing in Sensor Networks*, pages 317–332. Springer, 2003.

[24] A. Savvides, W. Garber, R. Moses, and M. Srivastava. An analysis of error inducing parameters in multihop sensor node localization. *Mobile Computing, IEEE Transactions on*, 4(6):567–577, Nov 2005.

[25] O. Tekdas and V. Isler. Sensor placement for triangulation based localization. *IEEE Tran. Automation Science and Engineering*, 7(3):681–685, 2010.

[26] P. Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104(1):1–16, 1998.

[27] F.-b. Wang, L. Shi, and F.-y. Ren. Self-localization systems and algorithms for wireless sensor networks. *Ruan Jian Xue Bao(J. Softw.)*, 16(5):857–868, 2005.

New Bounds for Facial Nonrepetitive Colouring*

Prosenjit Bose† Vida Dujmović‡ Pat Morin† Lucas Rioux-Maldague§

Abstract

We prove that the facial nonrepetitive chromatic number of any outerplanar graph is at most 11 and of any planar graph is at most 22.

1 Introduction

A sequence $S = s_1, s_2, \cdots, s_{2r}$, $r \geq 1$, is a *repetition* if $s_i = s_{r+i}$ for each $i \in \{1, \dots, r\}$. For example, 1212 is a repetition, while 1213 is not. A *block* of a sequence S is any subsequence of consecutive terms in S. A sequence is *nonrepetitive* if for every non-empty block B of S, B is not a repetition. Otherwise S is *repetitive*. For example, 1312124 is repetitive as it contains the block 1212 which is a repetition, while 123213 is nonrepetitive as it contains no such block. No sequence of length greater than three using only two symbols can be nonrepetitive. A result by Axel Thue in 1906 states that nonrepetitive sequences of infinite length can be created using three symbols [26]. Thue's work is considered to have initiated the study of the combinatorics of words [1].

A graph colouring variation on this theme was proposed by Alon et al. [2]. A nonrepetitive (vertex) colouring of a graph G is an assignment of colours to the vertices of G such that, for every path P in G, the sequence of colours of vertices in P is not a repetition. The *nonrepetitive chromatic number*, or *Thue chromatic number*, of G, denoted $\pi(G)$, is the minimum number of colours required to nonrepetitively colour G. In this setting, Thue's result states that, for all $n \geq 1$, the path P_n on vertices has $\pi(P_n) \leq 3$. Since its introduction, nonrepetitive graph colouring has received much attention [3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25].

A well-known conjecture, due to Alon et al. [2], is that there exists a constant K such that, for every planar graph G, $\pi(G) \leq K$. The current best upper bound for n-vertex planar graphs is $O(\log n)$ [10]. No planar graph with nonrepetitive chromatic number greater than 11 is known (see Appendix A in [10]).

More is known about the *facial* version of the problem for embedded planar graphs. Harant and Jendrol [18] asked if every plane graph can be coloured with a constant number of colours such that every *facial* path[1] is nonrepetitively coloured. Barát and Czap [3] answered this question in the affirmative by showing that 24 colours are sufficient. We reduce this bound to 22 by proving a bound of 11 for facial nonrepetitive colouring of outerplane graphs.

1.1 Related Work

Nonrepetitive Colouring. It is known that some families of graphs have bounded nonrepetitive chromatic number. In their original work, Alon et al. [2] showed that $\pi(G) = O(\Delta^2)$ if G has maximum degree Δ and that there are are graphs of maximum degree Δ with nonrepetitive chromatic number $\Omega(\Delta^2/\log\Delta)$. The constants in the $O(\Delta^2)$ upper bound have been steadily improved [11, 15, 16, 18].

Barát and Varjú [4] and Kündgen and Pelsmajer [21] independently showed that $\pi(G) \leq 12$ if G is outerplanar and, more generally, $\pi(G) \leq c^t$ if G has treewidth at most t. (Barát and Varjú proved the latter bound with $c = 6$ while Kündgen and Pelsmajer proved it with $c = 4$.) The upper bound of 4^t for t-trees is tight if $t = 1$ (trees), but it is not known if it is tight for other values of t. Even the upper bound of 12 for outerplanar graphs might not be tight, as no outerplanar graph with nonrepetititive chromatic number greater than 7 is known [4].

Facial Nonrepetitive Colouring. Facial nonrepetitive colouring was first considered by Havet et al [19], who studied the edge-colouring variant of the problem. In this setting, they were able to show that the edges of any plane graph can be 8-coloured so that every facial trail[2] is coloured nonrepetitively. For the list-colouring version of this problem, Przybyło [23] showed that lists with at least 12 colours are sufficient to colour the edges of any plane graph so that every facial trail is coloured nonrepetitively.

For the vertex-colouring version we study, Harant and Jendrol [18] proved that $\pi_f(G) = O(\log\Delta)$ if G is a

*This research is partially funded by NSERC and the Ontario Ministry of Research and Innovation.

†School of Computer Science, Carleton University

‡Department of Computer Science and Electrical Engineering, University of Ottawa

§Google

[1] A facial path is a path that is a contiguous subsequence of a facial walk; see Section 2 for a more rigorous definition.

[2] A *facial trail* is a contiguous subsequence of the edges traversed during the boundary walk of a face.

plane graph of maximum degree Δ and that $\pi_f(G) \leq 16$ if G is a Hamiltonian plane graph. They also conjectured that $\pi_f(G) = O(1)$ when G is any plane graph. As mentioned above, this latter conjecture was confirmed by Barát and Czap [3], who showed that, for any plane graph G, $\pi_f(G) \leq 24$. The results of Barát and Czap [3] also extend to graphs embedded in surfaces. They show that a graph embedded on a surface of genus g can be facially nonrepetitively 24^{2g}-coloured. The best lower bounds for facial nonrepetitive chromatic numbers are 5 for plane graphs and 4 for outerplane graphs [3].

2 Preliminary Results and Definitions

We assume the reader is familiar with standard graph theory terminology as used by, e.g., Bondy and Murty [6]. All graphs we consider are undirected, but not necessarily simple; they may contain loops and parallel edges. For a graph, G, we use the notations $V(G)$ and $E(G)$ to denote G's vertex and edge sets, respectively. For $S \subset V(G)$, $G[S]$ denotes the subgraph of G induced by the vertices in S and $G - S = G[V(G) \setminus S]$.

A graph is *k-connected* if it contains more than k vertices and has no vertex cut of size less than k. A *k-connected component* of a graph G is a maximal subset of vertices of G that induces a k-connected subgraph. We use *biconnected* as a synonym for 2-connected. A *bridge* in a graph G is an edge whose removal increases the number of connected components. A graph is *bridgeless* if it has no bridges.

A *plane graph* G is a fixed embedding of a graph in the plane such that its edges intersect only at their common endpoints. An *outerplane* graph G is a plane graph such that all the vertices of G are incident on the outer face of G. A *chord* in an outerplane graph is an edge that is not incident to the outer face. A *cactus graph* is an outerplane graph with no chords. An *ear* in a simple outerplane graph is an inner face that is incident to exactly one chord. An ear is *triangular* if it has exactly three vertices. The *weak dual* of a outerplane graph G is a forest whose vertices are the inner faces of G and that contains the edge fg if the face f and the face g have a chord in common. Note that the ears of G are leaves in the weak dual and that, if G is biconnected, then its weak dual is a tree.

A *walk* in a graph G is a sequence of vertices $v_0, \ldots, v_{\ell-1}$ such that, for every $i \in \{0, \ldots, \ell-2\}$ the edge $v_i v_{i+1}$ is in $E(G)$. The walk is *closed* if $v_{\ell-1}v_0$ is also in $E(G)$. A walk is a *path* if all its vertices are distinct. A *facial walk* in a plane graph G is a closed walk $v_0, \ldots, v_{\ell-1}$ such that, for every $i \in \{0, \ldots, \ell-1\}$, the edges $v_{(i-1) \bmod \ell} v_i$ and $v_i v_{(i+1) \bmod \ell}$ occur consecutively in the counterclockwise cyclic ordering of the edges incident to v_i in the embedding of G. A *facial path* is a contiguous subsequence of a facial walk that is also a path in G. A facial path is an *outer-facial path* if it appears in a facial walk of the outer face of G and it is an *inner-facial* path if it appears in a facial walk of some inner face of G.

Before proceeding with our results, we introduce a helper lemma due to Havet et al. [19] and two theorems that will be used throughout the paper. The helper lemma provides a way to interlace nonrepetitive sequences.

Lemma 1 (Havet et al. [19]). *Let $B = B_1, B_2, \ldots, B_k$ be a nonrepetitive sequence over an alphabet $\mathcal{B}$ in which each B_i has size at least 1. For each $i \in \{0, \ldots, k\}$, let A_i be a (possibly empty) nonrepetitive sequence over an alphabet $\mathcal{A}$ with $\mathcal{B} \cap \mathcal{A} = \emptyset$. Then $S = A_0, B_1, A_1, \ldots, B_k, A_k$ is a nonrepetitive sequence.*

We will require two results about the nonrepetitive chromatic number of trees and cycles:

Theorem 1 (Alon et al. [2]). *For every tree, T, $\pi(T) \leq 4$.*

Theorem 2 (Currie [9]). *For every $n > 2$, the cycle C_n on n vertices has*

$$\pi(C_n) = \begin{cases} 4 & \text{if } n \in \{5, 7, 9, 10, 14, 17\} \\ 3 & \text{otherwise.} \end{cases}$$

3 Outerplane Graphs

We begin with a simple lemma that allows us to focus, when convenient, on simple outerplane graphs (the proof is included in the full paper [7]).

Lemma 2. *Let G be a simple outerplane graph and let G' be an outerplane graph obtained by adding parallel edges and/or loops to G. Then, any facially nonrepetitive colouring of G is also a facially nonrepetitive colouring of G', so $\pi_f(G') \leq \pi_f(G)$.*

Next, we introduce a definition that is crucial to the rest of the paper. Let G be an outerplane graph. A *blocking set* of G is a set of vertices $B \subseteq V(G)$ such that for each 2-connected component H of G, $H - B$ is a tree and for each inner face F, $V(F) \setminus B \neq \emptyset$. See Figure 1 for an example of a blocking set.

The definition of a blocking set is subtle and implies some properties that we will use throughout.

Observation 1. *For any blocking set B of G, B does not include both endpoints of any chord c of G.*

Observation 2. *For any blocking set B of G and any inner face F of G, the vertices of $V(F) \cap B$ occur consecutively on the boundary of F. In other words, $F - B$ is a non-empty path.*

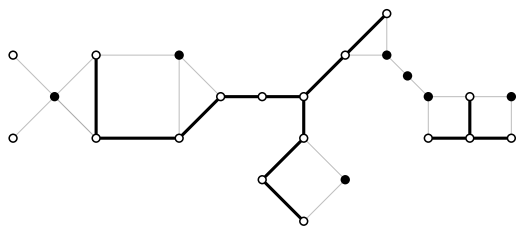


Figure 1: The blocking set, B (black vertices), of an outerplane graph, G, (with the edges of $G - B$ shown in bold).

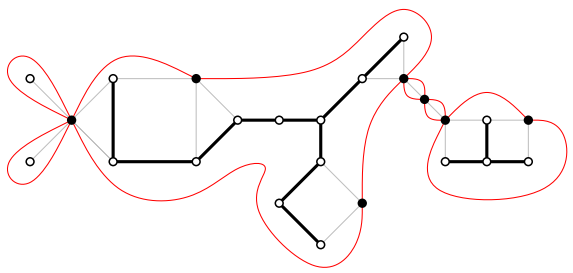


Figure 2: The blocking graph (curved edges in red) associated with a blocking set.

Observations 1 and 2 are true because, otherwise, $H - B$ would be disconnected for the 2-connected component containing c or F, respectively.

Lemma 3. *For every biconnected outerplane graph, G, and any vertex $v \in V(G)$, there exists a blocking set B of G such that $v \in B$ and, for each inner face F of G, $|B \cap V(F)| = 1$.*

Proof. The proof is by induction on the number of inner faces. If G has only one inner face, we take $B = \{v\}$ and we are done. Otherwise, select some ear, F of G whose chord is uw and such that $v \notin V(F) \setminus \{u, w\}$. Such an ear F always exists because G has at least two ears. Let $G' = G - (V(F) \setminus \{u, w\})$. The graph G' has one less inner face than G so, by induction, it has a blocking set B' that satisfies the conditions of the lemma. There are two cases to consider:

1. If one of u or w is in B' then we take $B = B'$ to obtain a blocking set that satisifes the conditions of the lemma.

2. Otherwise, let x be any vertex in $V(F) \setminus \{u, w\}$ and take $B = B' \cup \{x\}$ to obtain a blocking set that satisifes the conditions of the lemma. □

Lemma 3 allows us to prescribe that a particular vertex v be included in the blocking set, but it will also be convenient to exclude a particular vertex v by using Lemma 3 to force the inclusion of v's neighbour on the outer face (which is also on some inner face with v).

Corollary 1. *For every biconnected outerplane graph, G, and any vertex $v \in V(G)$, there exists a blocking set B of G such that $v \notin B$ and, for each inner face F of G, $|B \cap V(F)| = 1$.*

At this point we pause to sketch how Lemma 3 can already be used to give an upper-bound of 8 on the facial nonrepetitive chromatic number of biconnected outerplane graphs. For a biconnected outerplane graph, G, we take a blocking set B of G using Lemma 3. By Theorem 1, we can nonrepetitively 4-colour the tree $T = G - B$ using the colours $\{1, 2, 3, 4\}$, so what remains is to assign colours to the vertices in B. To do this, we use Theorem 2 to nonrepetitively 4-colour the cycle, C, that contains the vertices of B in the order they appear on the outer face of G using the colours $\{5, 6, 7, 8\}$. We claim that the resulting 8-colouring of G is facially nonrepetitive. No facial path on an inner face is coloured repetitively since each such facial path is also either present in the tree T or it contains exactly one vertex of B. No facial path on the outer face is coloured repetitively since it is obtained by interleaving a nonrepetitive sequence of colours in C with nonrepetitive sequences taken from T; by Lemma 1, a sequence obtained in this way is nonrepetitive.

In the full paper, we show that the preceding argument can be improved to give a bound of 7 on the facial nonrepetitive chromatic number of biconnected outerplane graphs. This is just a matter of adding vertices to the blocking set so that the cycle C does not have length in $\{5, 7, 9, 10, 14, 17\}$, so that it can be nonrepetitively 3-coloured.

Finally, we remind the reader that, although Lemma 3 and Corollary 1 provide blocking sets that include only one vertex on each inner face, not all blocking sets have this property. It is helpful to keep this in mind in the next section.

3.1 The Blocking Graph

The *blocking graph* of G for a blocking set B is the graph, denoted by $\text{block}_B(G)$, whose vertex set is B and whose edges are defined as follows: Begin with the (closed) facial walk W on the outer face of G. Remove every vertex not in B from W to obtain a cyclic sequence W' of vertices in B. For each consecutive pair of vertices uw in W' we add an edge uw to $\text{block}_B(G)$. This naturally defines the embedding of $\text{block}_B(G)$. See Figure 2 for an example. Note that $\text{block}_B(G)$ is a plane graph that is not necessarily simple; it may contain parallel edges (cycles of length two) and self-loops (cycles of length one).

The fact that a blocking set does not contain both endpoints of any chord of G (Observation 1) implies the following observation:

Observation 3. *For every outerplane graph G and any blocking set B of G, the blocking graph $\mathrm{block}_B(G)$ is a bridgeless cactus graph.*

Observation 4. *For every outerplane graph G, any blocking set B of G, and any facial path P on the outer face of G, the subsequence of P containing only the vertices of B is a (outer) facial path in $\mathrm{block}_B(G)$.*

Observation 5. *For every outerplane graph G, any blocking set B of G and any inner face F of G, $G[V(F) \cap B]$ is a non-empty path that is a facial path (on some inner face) in $\mathrm{block}_B(G)$.*

In the previous section, we sketched a proof of an upper bound of 8 on the facial chromatic number of biconnected outerplane graphs. This proof works by nonrepetitively 4-colouring the tree, $G - B$, obtained after removing the blocking set and then nonrepetitively 4-colouring a cycle, C, of vertices in the blocking set. This cycle, C, is actually the blocking graph, $\mathrm{block}_B(G)$. The following lemma shows that this strategy generalizes to the situation where we can find a facial nonrepetitive colouring of $\mathrm{block}_B(G)$ with few colours.

Lemma 4. *Let G be an outerplane graph and B be a blocking set of G. If there exists a facial nonrepetitive k-colouring of (the outer face of) $\mathrm{block}_B(G)$, then there exists a facial nonrepetitive $(4 + k)$-colouring of G.*

Proof. By Theorem 1 we can colour $G - B$ nonrepetitively using colours $\{1, 2, 3, 4\}$ and, by assumption, we can facially nonrepetitively colour $\mathrm{block}_B(G)$ with colours $\{5, \dots, k + 4\}$. These two colourings define a colouring of G that we now show is facially nonrepetitive.

Let P be a facial path in G. If P is a facial path of $\mathrm{block}_B(G)$ or a path in $G - B$ then there is nothing to prove.

Otherwise consider first the case that P is a path on an inner face F of G. There are two cases to consider:

1. The colouring of P is of the form A_0, B_1, A_1, where A_0 and A_1 are obtained from (possibly-empty) paths in $G - B$ and B_1 is obtained from a non-empty outer-facial path in $\mathrm{block}_B(G)$ (by Observation 5). Lemma 1 therefore implies that the colour sequence A_0, B_1, A_1 is nonrepetitive.

2. The colouring of P is of the form B_0, A_1, B_1, where A_1 is obtained from a non-empty path in $G-B$ and B_0 and B_1 are (possibly empty) outer-facial paths in $\mathrm{block}_B(G)$ (by Observation 5). Again, Lemma 1 implies that the resulting colour sequence is nonrepetitive.

Finally, consider the case where P is a facial path on the outer face of G. In this case, the colour sequence obtained from P is of the form $A_0, B_1, A_1, \dots, B_k, A_k$ where each A_i is obtained from a (possibly empty) path in $G - B$ and $B_1, \dots, B_k$ is obtained from a (outer) facial path in $\mathrm{block}_B(G)$ (by Observation 4). Again, Lemma 1 implies that the resulting colour sequence is nonrepetitive. □

3.2 Colouring Even Cactus Graphs

We now show how to colour the blocking graph—a cactus graph—of an outerplane graph. By Lemma 4, if we can find a facial nonrepetitive k-colouring of any cactus graph, we can get a facial nonrepetitive $k + 4$-colouring of any outerplane graph.

Recall that the best known upper bound for the facial Thue chromatic number of outerplane graphs is 12, which is the bound for the Thue chromatic number [4, 21]. Thus, to improve this bound, we need to find a facial nonrepetitive 7-colouring of the blocking graph. We have been unable to do this unless all cycles of the blocking graph are even. We will eventually address this limitation in Section 3.3 by proving the existence of a blocking set B such that $\mathrm{block}_B(G)$ has no odd cycles.

For any graph, G, a *levelling* of G is a function $\lambda \colon V(G) \to \{0, 1, 2, \dots\}$ such that for each $uv \in E(G)$, $|\lambda(u) - \lambda(v)| \le 1$. The level pattern of a path $v_1, \dots, v_k$ is the sequence $\lambda(v_1), \lambda(v_2), \dots, \lambda(v_k)$.

A *palindrome* is a sequence, $s_1, \dots, s_k$, that equal to its reversal, i.e., $s_i = s_{k-i+1}$ for all $i \in \{1, \dots, k\}$. A sequence is *palindrome-free* if it none of its substrings is a palindrome.

Lemma 5 (Kündgen and Pelsmajer [21])**.** *Let G be a graph and $\lambda \colon V(G) \to \{0, 1, 2, \dots\}$ be a levelling of G. Let $S = s_0, s_1, \dots, s_m$ be a nonrepetitive palindrome-free sequence on an alphabet $\mathcal{A}$ with $m = \max\{\lambda(v) \mid v \in V(G)\}$ and $c : V(G) \to \mathcal{A}$ be a colouring of G defined as $c(v) = s_{\lambda(v)}$. If a path $P = P_1, P_2$ with $|P_1| = |P_2|$ in G is repetitively coloured by c, then P_1 and P_2 have the same level pattern.*

Lemma 6. *For every cactus graph, G, with no odd cycles (and therefore, for every blocking graph with no odd cycles), $\pi_f(G) \le 7$.*

Proof. By Lemma 2, we may assume that G is simple. We may also assume that G is connected as this does not affect its nonrepetitive chromatic number. Also, assume that G is neither a cycle nor a tree since $\pi_f(G) \le 4 < 7$ for both these classes of graphs. If there exists a vertex v of G of degree 1, then let the *root* r of G be v. Otherwise, let r be any vertex of G of degree at least 3. Let λ be a levelling of G where $\lambda(v)$ is the distance in G from r to v. Let H be a graph that contains all vertices $v \in V(G)$ such that

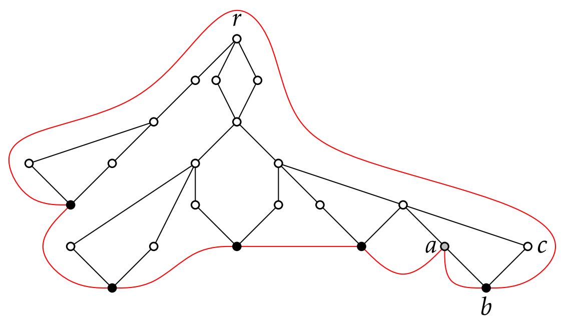


Figure 3: The graph H (black vertices and grey vertex) obtained from a cactus graph with no odd cycles. The vertex a is added in the final step so that H is not a cycle of length 5.

1. v is on a cycle C of G,
2. $\lambda(v) = \max_{u \in C} \lambda(u)$ and
3. $\deg_G(v) = 2$.

In other words, H contains the vertices of degree 2 that are on the deepest level of a cycle (see Figure 3). Notice that since every cycle of G is even, there is at most one vertex of H in each cycle of G.

If $\deg_G(r) \neq 1$, there must exist at least one face F^* of G such that exactly one vertex v of F^* has degree greater than two. From our choice of r, it follows that $\lambda(v)$ is the minimum over all vertices in $V(F^*)$. Since $|V(F^*)| \geq 4$, F^* has three consecutive degree-2 vertices a, b, and c, such that $b \in V(H)$. If $|V(H)| \in \{5, 7, 10, 14, 17\}$, we add a to $V(H)$. If $|V(H)| = 9$, we add both a and c to $V(H)$. Notice that now, either $\deg_G(r) = 1$ or $|V(H)| \notin \{5, 7, 9, 10, 14, 17\}$.

We now define the edge set $E(H)$ of H. For each $u, v \in V(H)$, we add the edge uv to $E(H)$ if there is a facial path on the outer face of G with endpoints u and v that does not contain any other vertices in $V(H)$. Note that H is either a cycle or a forest of paths. It can only be a cycle if G has no vertices of degree 1, in which case, $\deg_G(r) \neq 1$. In this case, our choice of $V(H)$ ensures that the length of this cycle is not in $\{5, 7, 9, 10, 14, 17\}$. This implies that H can be nonrepetitively coloured using the colour set $\mathcal{B} = \{1, 2, 3\}$, either by using the result of Thue [26] or Currie (Theorem 2).

To colour the remaining vertices of G, let $h = \max_{v \in V(G)} \lambda(v)$ and $S = s_0, s_1, \ldots, s_h$ be a palindrome-free nonrepetitive sequence on $\mathcal{A} = \{4, 5, 6, 7\}$. (A nonrepetitive palindrome-free sequence can be constructed from any ternary nonrepetitive sequence by adding a fourth symbol between blocks of size 2 [8].) Then, each vertex $v \in V(G) \setminus V(H)$ is assigned the colour $s_{\lambda(v)}$.

We will now show that the resulting 7-colouring of G is a facial nonrepetitive colouring. Suppose that this is not the case. Thus, there exists a path $P = P_1, P_2$ such that the colour sequence S corresponding to vertices of P is a repetition. Let us first suppose that P is on the outer face of G. We will use the following claim:

Claim 1. *Let P be a path on the outer face of G such that $V(P) \cap V(H) = \emptyset$. The level sequence L corresponding to vertices of P must be strictly decreasing, strictly increasing, or strictly decreasing then strictly increasing.*

Proof of Claim 1. Suppose that this is not the case. Then L cannot contain two consecutive elements of the form i, i as this can only correspond to an odd cycle of G, but all cycles of G are even. Thus, L must contain a block of the form $i, i+1, i$. Since P is on the outer face, we must have that the vertex v corresponding to $i+1$ is the highest numbered vertex on some cycle C and that $\deg_G(v) = 2$. But in this case, v must be in H, which is a contradiction. □

By Lemma 5, P_1 and P_2 have the same level pattern. However, if $V(P) \cap V(H) = \emptyset$ this is incompatible with Claim 1. Thus, P must contain vertices of H. Let $P_H = p_1, p_2, \ldots, p_k$ be the sequence of vertices of $V(P) \cap V(H)$ in the same order as in P. Notice that P_H is a path in H. Therefore, the colour sequence corresponding to P_H is nonrepetitive. Now, observe that the colour sequence formed by P is of the form $A_0, B_1, A_1, \ldots, B_\ell, A_\ell$ where $B_1, \ldots, B_\ell$ is a non-repetitive sequence of colours from $\mathcal{B} = \{1, 2, 3\}$ and each A_i is a non-repetitive sequence of colours from $\mathcal{A} = \{4, 5, 6, 7\}$. Therefore, by Lemma 1, P is coloured nonrepetitively.

Thus, P must be a facial path on some inner face F of G. If $V(P) \cap V(H) = \emptyset$ then, by Lemma 5, P_1 and P_2 have the same level pattern. No path on an even cycle has such a pattern using the levelling λ. Therefore, $V(P) \cap V(H) \neq \emptyset$, so P contains 1, 2, or 3 vertices of H. If P is a repetition it must contain exactly 2 vertices of H, thus P is a facial path on F^* (since every other inner face contributes at most one vertex to $V(H)$). Reusing the notation above, P cannot contain b since b has a unique colour in $V(F^*)$. Therefore, P must contain a and c and, in fact, these are the endpoints of P. The colour sequence of P must therefore be of the form xAx where $x \in \{1, 2, 3\}$ and A is a non-empty sequence over the alphabet $\{5, 6, 7, 8\}$. Such a sequence is not a repetition. □

3.3 Making an Even Blocking Graph

The proof of the following lemma requires more work than any other result in this paper and is excluded due to space limitations. It can, however, be found in the full version [7].

Lemma 7. *For every biconnected outerplane graph, G, and any vertex $v \in V(G)$:*

- *G has a blocking set B such that* $\text{block}_B(G)$ *is an even cycle and $v \in B$; and*
- *G has a blocking set B such that* $\text{block}_B(G)$ *is an even cycle and $v \notin B$.*

Lemma 8. *Every simple bridgeless outerplane graph G has a blocking set B such that all cycles in* $\text{block}_B(G)$ *are even.*

Proof. The proof is by induction on the number of 2-connected components of G. If G has no 2-connected components, then we take B to be the empty blocking set. If G has only one 2-connected component, then we apply Lemma 7. Otherwise, we select a 2-connected component, C, that shares exactly one vertex, v, with the rest of G. Let $G' = G - (V(C) \setminus \{v\})$ and apply induction on G' to obtain a blocking set, B', of G' such that $\text{block}_{B'}(G')$ has only even cycles. There are two cases to consider:

1. If B' contains v, then we apply the first part of Lemma 7 to obtain a blocking set B'' of C such that $\text{block}_{B''}(C)$ is an even cycle and $v \in B''$. We take $B = B' \cup B''$, which clearly forms a blocking set of G. The blocking graph $\text{block}_B(G)$ is simply the union of the two blocking graphs $\text{block}_{B'}(G)$ and $\text{block}_{B''}(C)$, which have only the vertex v in common. Thus, every cycle in $\text{block}_B(G)$ is also a cycle in one of these two graphs, so it has even length.

2. If B' does not contain v, then we apply the second part of Lemma 7 to obtain a blocking set B'' of C such that $\text{block}_{B''}(C)$ is an even cycle and $v \notin B''$. We take $B = B' \cup B''$.

   Refer to Figure 4. Starting at some appropriate vertex in $V(G) \setminus V(C)$ in the facial walk on the outer face of G, there is a last vertex, $x \in V(G') \cap B'$, encountered before the walk encounters the first vertex $u \in V(C) \cap B''$ and there is a last vertex $w \in V(C) \cap B''$ encountered before the walk returns to the next vertex $y \in V(G') \cap B'$. The edge xy is in $\text{block}_{B'}(G')$ and the edge vw is in $\text{block}_{B''}(C)$.

   Since every blocking graph is a bridgeless cactus graph (Observation 3), each of these edges is part of one even cycle in its respective graph. In $\text{block}_B(G)$ these two cycles are merged by removing the edges xy and vw and adding the edges xv and yw. The resulting cycle is even. Every other cycle in $\text{block}_B(G)$ is also a cycle in one of $\text{block}_{B'}(G')$ or $\text{block}_{B''}(C)$ so it has even length. □

Lemma 9. *Every simple outerplane graph G has a blocking set B such that all cycles in* $\text{block}_B(G)$ *are even.*

Proof. The proof is by induction on the number of bridges of G. If G has no bridges, then we apply Lemma 8. Otherwise, select some bridge uw of G and contract it to obtain a graph G' in which uw corresponds to a single vertex v. By induction, we obtain a blocking set B' of G' such that $\text{block}_{B'}(G')$ has only even cycles (or is empty). There are two cases to consider:

1. If $v \in B'$, then we take $B = B' \cup \{u, w\} \setminus \{v\}$. This introduces exactly one new cycle in $\text{block}_B(G)$ that is not present in $\text{block}_{B'}(G')$ and this cycle has length 2.
2. If $v \notin B'$, then we take $B = B'$, so $\text{block}_B(G) = \text{block}_{B'}(G')$. □

Finally, have all the tools to prove our main result on outerplane graphs:

Theorem 3. *For every outerplane graph, G, $\pi_f(G) \leq 11$.*

Proof. By Lemma 2, we may assume that G is simple. From Lemma 9, G has a blocking set B such that $\text{block}_B(G)$ has no odd cycles. Therefore, by Lemma 6, $\pi_f(\text{block}_B(G)) \leq 7$. Using this with Lemma 4 implies that $\pi_f(G) \leq 11$. □

4 Plane Graphs

In this section, we reuse the ideas from Barát and Czap [3] to facially nonrepetitively 22-colour every plane graph. Some modifications are needed because Barát and Czap use a nonrepetitive 12-colouring of outerplanar graphs whereas our Theorem 3 provides a *facial* nonrepetitive 11-colouring of *outerplane* graphs. Details are included in the full version [7].

Theorem 4. *Let $r = \max\{\pi_f(H) : H$ is outerplane$\}$. Then, for every plane graph G, $\pi_f(G) \leq 2r$.*

5 Concluding Remarks

We note that the proofs in this paper lead to straightforward linear-time algorithms. After the appropriate decomposition steps, there are essentially two subproblems: (1) finding an appropriate blocking set in a biconnected outerplane graph (Lemma 7) and (2) colouring cactus graphs with no odd cycles (Lemma 6). The proof of Lemma 6 is easily made into a linear-time algorithm. The proof of Lemma 7 can be implemented by a recursive ear-cutting algorithm that implements Lemma 3 followed by a traversal of the dual tree in order to find the face F_{uw} used in the proof of Lemma 7.

It seems unlikely that our upper bound of 11 for the facial nonrepetitive chromatic number of outerplane graphs (and hence the bound of 22 for plane graphs) is tight. (Recall that the best known lower bounds are 4

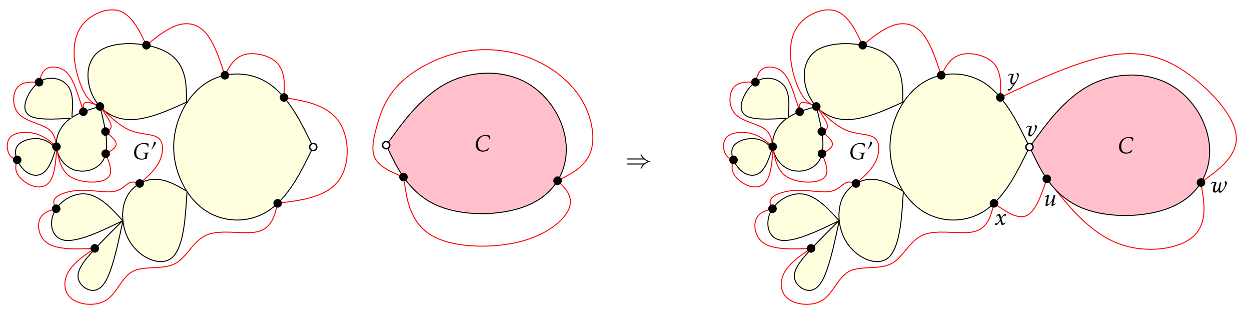


Figure 4: Case 2 in the proof of Lemma 8.

and 5, respectively [3].) Thus, an obvious direction for future work is to improve these bounds. Our proof of Lemma 4 uses a nonrepetitive 4-colouring of trees, but a facial nonrepetitive colouring would also be sufficient. This leads naturally to the following problem:

Open Problem 1. Is $\pi_f(T) \leq 3$ for every tree, T?

References

[1] Jean-Paul Allouche and Jeffrey Shallit. The ubiquitous Prouhet-Thue-Morse sequence. In *Sequences and their applications*, pages 1–16. Springer, 1999.

[2] Noga Alon, Jarosław Grytczuk, Mariusz Hałuszczak, and Oliver Riordan. Nonrepetitive colorings of graphs. *Random Structures & Algorithms*, 21(3-4):336–346, 2002.

[3] János Barát and Július Czap. Facial nonrepetitive vertex coloring of plane graphs. *Journal of Graph Theory*, 74(1):115–121, 2013.

[4] János Barát and Péter P. Varjú. On square-free vertex colorings of graphs. *Studia Scientiarum Mathematicarum Hungarica*, 44(3):411–422, 2007.

[5] János Barát and David R. Wood. Notes on nonrepetitive graph colouring. *Electronic Journal of Combinatorics*, 15(1):Research Paper 99, 13, 2008.

[6] John-Adrian Bondy and U. S. R. Murty. *Graph theory*. Graduate texts in mathematics. Springer, New York, London, 2007.

[7] Prosenjit Bose, Vida Dujmović, Pat Morin, and Lucas Rioux-Maldague. New bounds for facial nonrepetitive colouring. arxiv:1604.01282, 2016.

[8] Boštjan Brešar, Jaroslaw Grytczuk, Sandi Klavžar, Staszek Niwczyk, and Iztok Peterin. Nonrepetitive colorings of trees. *Discrete Mathematics*, 307(2):163–172, 2007.

[9] James D. Currie. There are ternary circular square-free words of length n for $n \geq 18$. *Electronic Journal Combinatorics*, 9(1):N10, 2002.

[10] Vida Dujmović, Fabrizio Frati, Gwenaël Joret, and David R. Wood. Nonrepetitive colourings of planar graphs with $O(\log n)$ colours. *The Electronic Journal of Combinatorics*, 20(1):P51, 2013.

[11] Vida Dujmović, Gwenaël Joret, Jakub Kozik, and David R. Wood. Nonrepetitive colouring via entropy compression. *Combinatorica*, (in press) 2013.

[12] Francesca Fiorenzi, Pascal Ochem, Patrice Ossona de Mendez, and Xuding Zhu. Thue choosability of trees. *Discrete Applied Mathematics*, 159(17):2045–2049, 2011.

[13] Adam Gagol, Gwenaël Joret, Jakub Kozik, and Piotr Micek. Pathwidth and nonrepetitive list coloring, 2016. arXiv:1601.01886.

[14] Daniel Gonçalves, Mickaël Montassier, and Alexandre Pinlou. Entropy compression method applied to graph colorings. In *9th International colloquium on graph theory and combinatorics*, 2014.

[15] Jarosław Grytczuk. Nonrepetitive colorings of graphs—a survey. *International Journal of Mathematics and Mathematical Sciences*, 2007 Article 74639, 2007.

[16] Jarosław Grytczuk. Nonrepetitive graph coloring. In *Graph Theory in Paris*, pages 209–218. Springer, 2007.

[17] Jarosław Grytczuk, Jakub Kozik, and Piotr Micek. New approach to nonrepetitive sequences. *Random Structures & Algorithms*, 42(2):214–225, 2013.

[18] Jochen Harant and Stanislav Jendrol. Nonrepetitive vertex colorings of graphs. *Discrete Mathematics*, 312(2):374–380, 2012.

[19] Frédéric Havet, Stanislav Jendrol, Roman Soták, and Erika Škrabul'áková. Facial non-repetitive edge-coloring of plane graphs. *Journal of Graph Theory*, 66(1):38–48, 2011.

[20] Jakub Kozik and Piotr Micek. Nonrepetitive choice number of trees. *SIAM Journal on Discrete Mathematics*, 27(1):436–446, 2013.

[21] Andre Kündgen and Michael J. Pelsmajer. Nonrepetitive colorings of graphs of bounded treewidth. *Discrete Mathematics*, 308(19):4473–4478, 2008.

[22] Andrzej Pezarski and Michał Zmarz. Non-repetitive 3-coloring of subdivided graphs. *Electronic Journal of Combinatorics*, 16(1):N15, 2009.

[23] Jakub Przybyło. On the facial Thue choice index via entropy compression. *Journal of Graph Theory*, 77(3):180–189, 2014.

[24] Jens Schreyer and Erika Škrabul'áková. On the facial Thue choice index of plane graphs. *Discrete Mathematics*, 312(10):1713–1721, 2012.

[25] Jens Schreyer and Erika Škrabul'áková. Total Thue colourings of graphs. *European Journal of Mathematics*, 1(1):186–197, 2015.

[26] Axel Thue. Über unendliche zeichenreichen. *Norske Vid Selsk. Skr. I. Mat. Nat. Kl. Christiana*, 7:1–22, 1906.

Problems on One Way Road Networks

Jammigumpula Ajaykumar* Avinandan Das† Navaneeta Saikia‡ Arindam Karmakar§

Abstract

Let $OWRN = \langle W_x, W_y \rangle$ be a One Way Road Network where W_x and W_y are the sets of directed horizontal and vertical roads respectively. $OWRN$ can be considered as a variation of directed grid graph. The intersections of the horizontal and vertical roads are the vertices of $OWRN$ and any two consecutive vertices on a road are connected by an edge. In this work, we analyze the problem of collision free traffic configuration in a $OWRN$. A traffic configuration is a two-tuple $TC = \langle OWRN, C \rangle$, where C is a set of cars travelling on a pre-defined path. We prove that finding a maximum cardinality subset $C_{sub} \subseteq C$ such that $TC = \langle OWRN, C_{sub} \rangle$ is collision-free, is NP-hard. Lastly we investigate the properties of connectedness, shortest paths in a $OWRN$.

1 Introduction

The rapid development in the existing motor vehicle technology has led to the increase in demand of automated vehicles, which are in themselves capable of various decision activities such as motion-controlling , path planning etc .This has motivated many to address a large number of algorithmic and optimisation problems. The 1939 paper by Robbins [2], which gives the idea of *orientable* graphs and the paper by Masayoshi et.al [5], are to a certain extent an inspiration to formulating our graph network. The work by Dasler and Mount [1], which basically considers motion coordination of a set of vehicles at a *traffic-crossing* (intersection), has been a huge motivation and a much closer approach to that of ours. But unlike their work, we consider a much simpler version of a grid graph and mainly concentrate on analysing essential properties , and deriving suitable algorithms and structures to have a collision-free movement of traffic in the given graph network. Further, our work also mentions the possible shortest path configurations in our defined graph. We now discuss some definitions and notations that are referred to in the rest of the paper.

1.1 Definitions and Notations

A road is a directed line, which is either parallel to X-axis (X_i) or Y-axis (Y_i) and it is uniquely defined by its direction and distance from the corresponding parallel axis. Here direction is the constraint which restricts the movement on the road.

Formally, a *road* is defined as a 2-tuple, $X_i = \langle d_i, x_i \rangle$, $Y_j = \langle d_j, y_j \rangle$, where i, j are the indices with respect to their parallel axis, d_k is the direction of the road i.e., $d_k \in \{0, 1\}$ (where 0 represents $-ve$ direction and 1 represents $+ve$ direction of the respective axis) , and x_i is the distance of the road X_i from X-axis, similarly for y_j.

We define a *One Way Road Network (OWRN)* as a network with a set of n horizontal and m vertical Roads. Formally a OWRN is a 2-tuple, $OWRN = \langle W_x, W_y \rangle$, where, $W_x = \{X_1, X_2, X_3 \ldots, X_n\}$, $W_y = \{Y_1, Y_2, Y_3 \ldots, Y_m\}$.

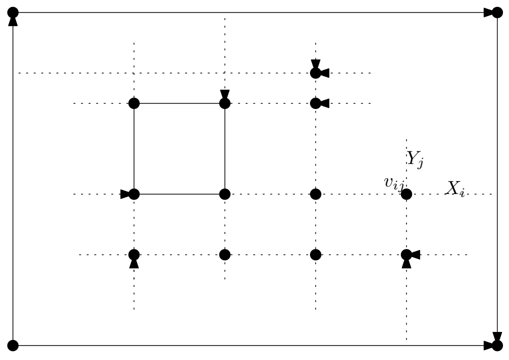


Figure 1: Graphical representation of a $OWRN$

A *junction* or *vertex* v_{ij} is defined as the intersection of X_i and Y_j. Formally, $v_{ij} \in (W_x \times W_y)$.

An *edge* is a connection between two adjacent vertices of a road and all the edges on a road are in the same direction as that of the road.

The *boundary roads* of a OWRN are the outermost roads, i.e., X_1, X_n, Y_1 and Y_m. In this paper we term each vertex on the boundary roads as *boundary vertex*, i.e., all the vertices with degrees 2 and 3.

A *vehicle* c is defined as a 3-tuple $\langle t, s, P \rangle$, where t is the starting time of the vehicle, s is the speed of the vehicle , which is constant throughout the journey and each vehicle moves non-stop from its starting vertex to destination vertex (unless a collision occurs), and P is the path to be travelled by the vehicle.

A *path* P_r of a vehicle c_r is defined as the ordered set of vertices through which it traverses the OWRN, Formally, $P = \{v_1, v_2, v_3 \ldots, v_l\}$, $\forall i$, $v_i \in (W_x \times W_y)$, and $\forall i, 0 < i < l, v_i \rightarrow v_{i+1}$.

*Tezpur University, ak.jammi@gmail.com
†Tezpur University, adas33745@gmail.com
‡Tezpur University, saikia.navaneeta@gmail.com
§Tezpur University, karmarind@gmail.com

A *traffic configuration* is defined as a collection of vehicles over a OWRN. Formally a TC is a 2-tuple, $TC = \langle OWRN, C \rangle$, where C is the set of vehicles $\{c_1, c_2, c_3 \ldots, c_k\}$.

Now, we define a *collision* as two vehicles c_i and c_j $(i \neq j)$ reaching the same vertex orthogonally at the same time. So a *collision-free* traffic configuration is a TC without any collisions.

2 Results

Before considering the traffic configuration problem, we define the connectivity of a OWRN.

2.1 Connectivity of a One Way Road Network

In this section, we consider a general OWRN of $n \times m$ roads, and show the conditions for it to be strongly-connected.

The reachability to (and from) the non-boundary vertices is evident from the following lemmas.

Lemma 1 *For every non-boundary vertex v_{ij}, $1 < i < n$, $1 < j < m$ there exists e,f such that we can always reach the boundary vertices v_{ie}, v_{fj} from v_{ij}.*

Proof. We prove this lemma by considering the two roads which intersect to form the vertex v_{ij}.

1. For the road X_i, if $d_i = 0$ then by definition we can reach v_{i1} from v_{ij}, i.e., $e = 1$. Otherwise we can reach v_{in} from v_{ij}, i.e., $e = n$.
2. For the road Y_j, if $d_j = 0$ then by definition we can reach v_{1j} from v_{ij}, i.e., $f = 1$. Otherwise we can reach v_{mj} from v_{ij}, i.e., $f = m$.

From the above conditions we can clearly see that for any non-boundary vertex v_{ij}, $\exists e, f$ such that v_{ie}, v_{ej} are reachable from v_{ij}.

□

Lemma 2 *For every non-boundary vertex v_{ij} there exists e,f such that v_{ij} is reachable from the boundary vertices v_{ie}, v_{fj}.*

Proof. The proof of this lemma is analogous to that of Lemma 1. □

Theorem 3 *A One Way Road Network is strongly-connected iff the boundary roads form a cycle.*

Proof. The proof of this theorem follows from Lemma 4 and Lemma 5. □

Lemma 4 *If all the boundary vertices of a OWRN form a cycle, then it is strongly-connected.*

Proof. Given two vertices v_{ij}, v_{kl} in a OWRN, to reach from v_{ij} to v_{kl}, we have four different possibilities

1. *Both boundary vertices*: Any boundary vertex is reachable from any other boundary vertex, since they all form a cycle. Therefore a path exists.
2. *v_{ij} non-boundary vertex, v_{kl} boundary vertex*: From *Lemma 1* we know that, from any non-boundary vertex v_{ij} we can always reach a boundary vertex, and from that vertex we can reach v_{kl} as shown in 1. Therefore a path exists.
3. *v_{ij} boundary vertex, v_{kl} non-boundary vertex*: From *Lemma 2* we know that any non-boundary vertex v_{kl} is always reachable from a boundary vertex, and which in turn is reachable from v_{ij} as shown in 1. Therefore a path exists.
4. *Both non-boundary vertices*: From 1, 2 and 3 it is implied that there exists a path in this case too.

□

Lemma 5 *If a given One Way Road Network is strongly-connected, then all the boundary vertices form a cycle.*

Proof. Let us assume on the contrary that the boundary vertices do not form a cycle in the OWRN. Then there will exist a boundary vertex of degree 2 (corner vertex) such that either both the boundary roads are incoming or outgoing.

1. *Both incoming roads*: In this case, we will not be able to reach any other vertex from that vertex.
2. *Both outgoing roads*: In this case, we will not be able to reach that vertex from any other vertex.

Therefore, the OWRN is not strongly-connected.

Hence, by contradiction, we can claim that the boundary vertices of a strongly-connected OWRN will always form a cycle. □

2.2 Traffic Configuration

We now define the traffic configuration problem in a connected OWRN.

Problem 1 *Given a traffic configuration $\langle OWRN, C \rangle$, our objective is to find a maximum cardinality subset C_{sub} , $C_{sub} \subseteq C$, such that the new traffic configuration $\langle OWRN, C_{sub} \rangle$ is **collision-free**.*

In the following sections we discuss the hardness of the above problem, and also mention some of the restricted versions of the same.

2.2.1 Hardness of Collision-Free Traffic Configuration

In this section we show that finding a solution to the traffic configuration problem is ***NP-Hard***. For this , we have the following theorem.

Theorem 6 *Given an undirected graph $G = \langle V, E \rangle$, there exists a traffic configuration $\langle OWRN, C \rangle$, computable in polynomial-time, such that the cardinality of Maximum Independent Set of G is k iff the maximum cardinality of C_{sub} is k.*

To prove this theorem, we reduce Maximum Independent Set problem to the Traffic Configuration problem, which is achieved with the help of the following lemmas and algorithms.

Lemma 7 *For any complete graph K_n, there exists a traffic configuration TC, such that every vertex in K_n has a respective car, and for every edge in K_n there is a collision between the respective vehicles.*

Proof. We prove this lemma using proof by construction. The following steps show how to construct a TC from K_n.

1. We construct a OWRN of $2n \times n$ roads, with $2n$ horizontal roads and n vertical roads in which
   (a) For the road X_i, $d_i = \begin{cases} 1 & 1 < i \leq 2n \\ 0 & i = 1 \end{cases}$
   and $x_i = \begin{cases} 0 & i = 1 \\ x_{i-1} + \delta & 1 < i \leq 2n \end{cases}$
   (b) For the road Y_j, $d_j = \begin{cases} 0 & 1 < j \leq n \\ 1 & j = 1 \end{cases}$
   and $y_j = \begin{cases} 0 & j = 1 \\ y_{j-1} + \delta & 1 < j \leq n \end{cases}$
   where δ is a numeric constant.
2. The set of vehicles C is defined as $\{c_1, c_2, c_3 \ldots, c_n\}$ and for each vehicle $c_i \in C$ we assume
   (a) The start time to be 0 and the velocity to be ω.
   (b) $P_i = \{v_{ri}, v_{(r-1)i} \ldots, v_{qi}, v_{q(i+1)} \ldots, v_{qn}\}$, where $r = n + i - 1$, $q = n - i + 1$.
   (c) $c_i = \langle 0, \omega, P_i \rangle$
3. Now we can observe that two vehicles $\{c_i, c_j\} \in C$ collide at vertex $v_{(n-i+1)(j)}$, where $i < j$.
4. We assume that each node l in K_n corresponds to a vehicle c_l , and each edge between two nodes α and γ in K_n corresponds to the collision of the respective vehicles c_α,c_γ.

$\therefore$ We obtain the corresponding $TC = \langle OWRN, C \rangle$ of K_n. □

Now to reduce any simple graph G, we first compute the corresponding TC for the complete graph $K_n(G)$. We then introduce 4 equi-spaced roads with directions (d_k 's) $\{0, 1, 0, 1\}$ between every two adjacent roads X_i ,X_{i+1} and Y_j, Y_{j+1}, respectively, in the above formed OWRN, the path of each vehicle is to be modified accordingly.

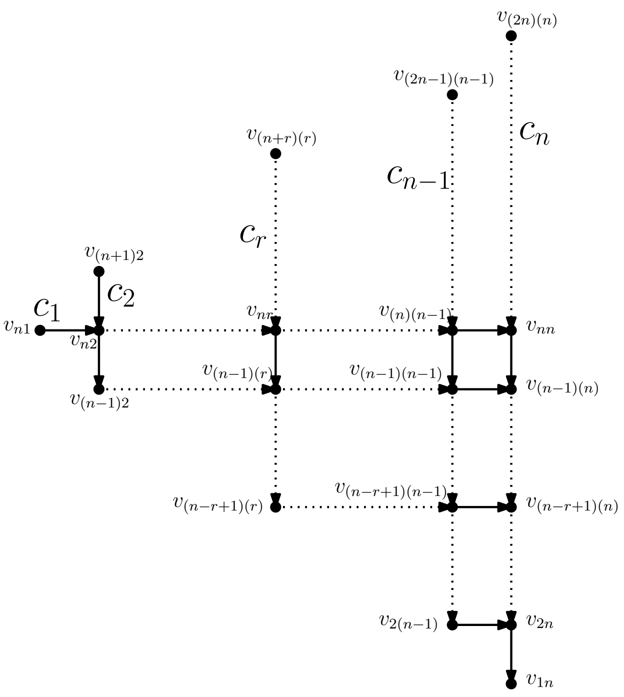


Figure 2: Paths for vehicles $\{c_1, c_2, \ldots, c_n\}$ in the TC obtained from Lemma 9

We define method DELAY($\alpha, \beta, P_i, \Delta$), where α and β are the two vertices in the path of c_i, and Δ is the total number of delays, which modify the path P_i to introduce a Δ number of small time delays in between the vertices α,β, this delay will also be propagated to all the successive vertices of β in P_i.

Algorithm 1: Delay method

```
Input: $P_r = (\ldots, \alpha \ldots, \beta \ldots)$, no.of delays $\Delta$
Output: $P_r$ after introduction of $\Delta$ delays
  procedure DELAY($\alpha, \beta, P_r, \Delta$)
      if $\Delta = 0$ then
          then return
      end
      $\gamma_1, \gamma_2$ are two successive vertices of $\alpha$ in $P_r$
      $\epsilon_1 \neq \gamma_2$, is a vertex | there is an edge $\gamma_1 \rightarrow \epsilon_1$
      $\epsilon_2$ is a vertex | $\epsilon_1 \rightarrow \epsilon_2, \epsilon_2 \rightarrow \gamma_1$
      $P_r = (\ldots, \alpha, \gamma_1, \epsilon_1, \epsilon_2, \gamma_2 \ldots, \beta \ldots)$
      DELAY($\gamma_2, \beta, P_r, \Delta - 1$)
  end procedure
```

The method COLLISIONVERTEX(c_i, c_j)($i \neq j$) will return the common vertex through which both the vehicles travel. In the base case($i = 0$) the Collision method returns the starting vertex of the vehicle c_j.

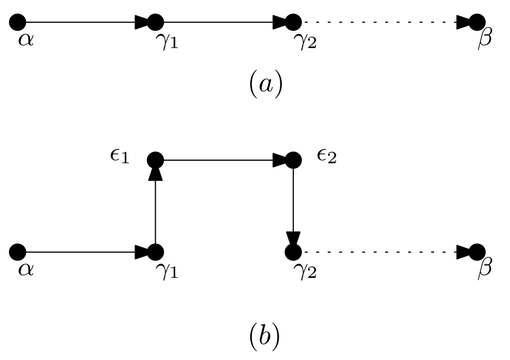


Figure 3: Path from α to β (a) Before delay introduction, (b) After delay introduction

Algorithm 2: Collision method

```
procedure COLLISIONVERTEX($c_i, c_j$)
    if $i = j$ then
        return COLLISIONVERTEX($c_i, c_{j+1}$)
    end
    else if $i = 0$ then
        return Starting Vertex of $c_j$
    end
    return Common vertex in $P_i$,$P_j$
end procedure
```

The following algorithm makes use of the above mentioned methods to construct the required TC by introducing some number of delays in the path of each vehicle.

Algorithm 3: Reduction algorithm

```
procedure REDUCE($c_i, c_j$)
    if $i > j$ or $i = 0$ then
        return 0
    end
    $\Delta$ = REDUCE($c_{i-1}, c_j$)
    REDUCE($c_i$,$c_{j-1}$)
    $\alpha$ = COLLISIONVERTEX($c_{i-1}, c_j$)
    $\beta$ = COLLISIONVERTEX($c_i, c_j$)
    if HASEDGE($i, j$) then
        DELAY($\alpha, \beta, P_j, i - \Delta$)
        return $i$
    end
    else if $i - \Delta > 1$ then
        DELAY($\alpha, \beta, P_j, i - (\Delta + 1)$)
        return $i - 1$
    end
    return $\Delta$
end procedure
```

The reduction algorithm is constructed using the following properties:

Property 1: The number of delays introduced in the path of a vehicle c_i is equal to i.

Property 2: If there is an edge between two nodes $i, j, j > i$ in G, then c_i and c_j will have collision in the TC. The number of delays introduced in the path P_j before the collision of c_i,c_j is i.

Property 3: If there is no edge between two nodes $i, j, j > i$ in G, then c_i and c_j will not have a collision in the TC. The number of delays introduced in the path P_j before the collision of c_i,c_j is $i - 1$.

The method HASEDGE(i, j) will return value true if there is an edge between i and j in the graph G, else false.

Lemma 8 *The maximum number of delays introduced between the two collision vertices α and β as defined in the reduction algorithm, will be two.*

Proof. The proof of this lemma follows from the above stated properties. The number of delays introduced in the path P_j, before collision of vehicles c_i and c_j is either i,$i-1$. The number of delays introduced in the path P_j, before collision of vehicles c_{i+1} and c_j is either $i+1$,i.

$\therefore$ the maximum number of delays that can be introduced between α and β is two. □

From the above Lemma and the reduction algorithm, we have the following Lemma

Lemma 9 *The above Reduction algorithm can be solved using Dynamic Programming approach in polynomial-time $\mathcal{O}(n^2)$, and the space complexity of both TC and OWRN created is $\mathcal{O}(n^2)$.*

Lemma 10 *If C_{sub} be any subset of C in TC such that $TC_{new} = \langle OWRN, C_{sub}\rangle$ is collision-free, then C_{sub} corresponds to Independent Set of G.*

Proof. Since TC_{new} is collision-free, so no two nodes in the graph G, which corresponds to respective cars in $Csub$, consists of an edge. Thus, we can claim that C_{sub} corresponds to an independent set in G. □

From *Lemma 10* we can say that maximum C_{sub} corresponds to *Maximum Independent Set* in G. Now, using *Lemma 9* and *Lemma 10* we can prove that the traffic configuration problem is *NP-Hard*.

2.2.2 Restricted Version

If we constrain our vehicles to move in a straight line motion, then the corresponding graph to TC will be a Bipartite Graph. And *Maximum Independent Set* of a Bipartite Graph can be computed using Konig's Theorem and Network-Flow Algorithm in polynomial-time. Hence, the restricted version of the problem is solvable in polynomial-time.

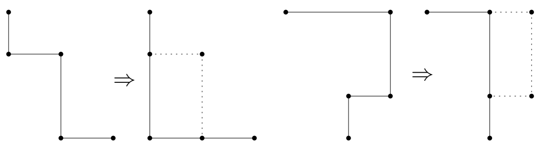
2.3 Shortest Path Properties

Suppose in a city of only One Way Road Network ,a person wants to travel from one point to another with minimum distance. Now, the objective would be to compute the shortest path to the destination in least possible time. Designing efficient algorithms to compute the shortest path in a One Way Road Network would be useful in many applications in the areas of facility location, digital micro-fluidic bio-chips,etc. The length of the shortest path between two vertices in a OWRN may not be the Manhattan distance. There may be a pair of neighbouring vertices which are the farthest pair of vertices in the OWRN metric. A *turn* in a path is defined when two consecutive pair of edges are from different roads. We have the following Lemmas for shortest path between any two vertices in a OWRN:

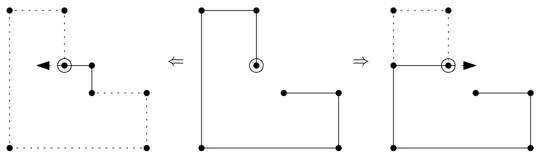
Lemma 11 *Between any pair of vertices u, v , there exists a shortest path of at most four turns.*

Proof. We can observe that a shortest path of k turns is geometrically similar to an addition of one more turn at the end of shortest path of $k-1$ turns, And the path with k turns can be reduced if it's sub-path with $k-1$ turns can be reduced.

1. The path with zero turns is a straight line and the path with one turn is an L shaped, which are trivial.
2. The path with two turns, two configurations $(c), (d)$ are possible and are valid as shown in *Figure 2.3*.
3. The path with three turns, two configurations$(e), (f)$ shown in *Figure 2.3* are valid and the other two shown below are not valid as there exist another shortest path with 1 turn.



4. The path with four turns, from the above explanation it can be easily seen that only $(g), (h)$ are valid configurations with four turns.
5. The path with five turns, in the below figure we can see that a five turn path will become either a one turn or four turn. similarly we can see that for any five turn path



there exist a shortest path with less number of turns.

Hence we can see that there always exist a shortest path with at most 4 turns. □

From *Lemma 11* we observe that there exist a shortest path between every pair of vertices in the OWRN which will be a rotationaly symmetric to one of the paths shown below:

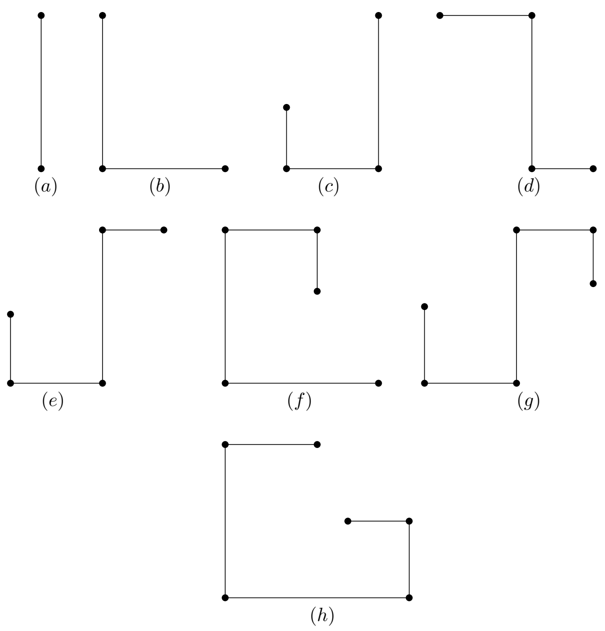


Figure 4: Shortest path configurations

Lemma 12 *The upper bound on the length of the shortest path between any pair of vertices u, v is the perimeter of the boundary of the OWRN.*

Proof. The proof of this lemma follows from *Lemma 1*, *Lemma 2* and *Lemma 11*.

□

3 Remarks

We have shown all the possible configurations of the path, that connects two vertices in a OWRN. In the future we will extend this work to compute various kind of facility location problems on a OWRN. It will be interesting to investigate the time complexity of one-centre or k-centre problems with respect to OWRN metric. Other interesting problems may be to design an efficient data structure for dynamic maintenance of shortest path in directed grid graphs.

References

[1] Philip Dasler, David M. Mount. On the Complexity of an Unregulated Traffic Crossing *arXiv:1505.00874 [cs]*, May 2015.

[2] H. E. Robbins. A Theorem on Graphs, with an Application to a Problem of Traffic Control *The American Mathematical Monthly*, 46: 281-283,1939.

[3] T. Kashiwabara and T. Fujisawa. NP-Completeness of the Problem of Finding a Minimum-Clique-Number Interval Graph Containing a Given Graph as a Subgraph. *Proceedings International Conference on Circuits and Systems*, 657-660, 1979.

[4] T. Kashiwabara and S. Masuda, K. Nakajima and T. Fujisawa. Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc-graph. *Journal of Algorithms*, 13: 161-174, 1992.

[5] Masayoshi Kakikura, Jun-ichi Takeno and Masao Mukaidono. A Tour Optimization Problem in a Road Network with One-way Paths. *The transactions of the Institute of Electrical Engineers of Japan.C*, 98: 257-264, 1978.

[6] Garey, M.R. and Johnson, D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness *San Francisco*, CA: Freeman, 1979.

[7] K. C. Tan and S. C. Lew. Fashioning one-way urban road network design as a module orientation problem. *Urban Transport X. Urban Transport and the Environment in the 21st Century*, p. 649-658, 2004-5.

[8] F. Berger and R. Klein. A travellers problem. *In Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, SoCG '10, page 176182, New York, NY, USA, 2010. ACM.

[9] K. M. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. Artif. Intell.Res.(JAIR)*, 31:591656, 2008.

[10] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760772, July 1998.

[11] R. Rajamani. Vehicle Dynamics and Control. *Springer Science and & Business Media*, December 2011.

[12] P. R. Wurman, R. Dandrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *The AI magazine*, 29(1):919, 2008.

[13] J. Yu and S. M. LaValle. Multi-agent path planning and network flow. *arXiv:1204.5717 [cs]*, April 2012.

[14] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):8091, October 1959.

[15] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254265, March 1987.

Appendix

Creating a robust implementation for segment intersection by refinement: A multistage assignment that defines away degeneracies

Jack Snoeyink*

Abstract

This note suggests a multistage programming assignment for an algorithms class to implement red/blue segment intersection and Boolean operations for polygons. It aims to give students experience with a complex implementation project in which:

1. students work with real data of relevance to them,
2. several algorithms and data structures must be combined,
3. initial definitions must be refined, forcing code refactoring, as the problem is better understood,
4. proofs and refutations are needed to certify correctness and reveal important test cases, and
5. degeneracies can be handled by careful definition, rather than as special cases.

1 Introduction

The experienced researcher or developer knows the importance of good definitions; Imre Lakatos' book, "Proofs and Refutations," [7] is an entertaining presentation of how mathematical definitions are refined by attempted proofs and refutations. The experienced developer knows that implementations can also be refined (and refactored) as their underlying definitions are refined. Students, however, often do not connect implementation with the concept of "proof." For most assigned algorithms and data structures, the definitions, proofs, and pseudocode are already in a book or on Wikipedia, so they are not seen as an integral part of the student's development and testing.

This note outlines a multi-stage implementation assignment for the segment intersection subroutine that is the core of map polygon overlay. It applies several basic algorithms (sorting and searching) and data structures (heap, queue, dynamic search tree or skip list) to produce visual analysis of real-world data of interest to the students.

Two key challenges for implementing geometric algorithms are 1) implementing exact predicates in floating point and 2) handling degeneracies, which are geometric configuration important to the algorithm that disappear with probability 1 if all inputs are infinitesimally and randomly perturbed. Degeneracies for segment intersection include triples of co-linear points or overlapping line segments. In this assignment, the first challenge is addressed by limiting the input to values for which the computations can be exact. (E.g., single precision inputs can be multiplied exactly in double precision.) Restricting the inputs increases the probability of degeneracies, but these are addressed by refining definitions to fold them into the generic cases. Students can recognize how effort spent on definition and proof avoids effort to write and test special case code.

The original algorithm comes from the Master's thesis of Andrea Mantler [8], prefigured in work with Boissonnat [3]. I have failed with having other graduate students implement this algorithm, so this was a foolhardy attempt to present the algorithm in a form that undergraduates could implement. The results were only partially successful; not everyone had complete working code at the end, but my assistant, Stephen Love, and I learned a lot about how to specify such an assignment and the importance of making a reference implementation available from the start, so I am eager to try again.

Remark: Instructor notes are included for most stages.

2 The problem

Given red and blue maps consisting of n line segments each, and having no red/red or blue/blue crossings, report all red/blue crossings in order along each segment.

We need not know exactly where an intersection point is to know that two segements interect – this is fortunate, because it takes more precision to record the coordinates of an intersection than to observe that it must exist. For example, scanning Fig. 1 left to right, we observe that red segment nr starts below blue segment vw, and either q or r witnesses that it ends above.

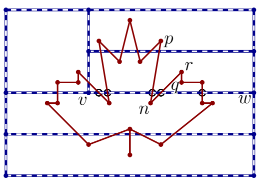


Figure 1: The 5 red/blue crossings with vw

We will be able to list the intersections with vw by just listing red segments in the order that they are crossed. The fact that there are no red/red crossings will allow you to put the reds in order in SI4–6, and

*School of Computer Science, University of North Carolina at Chapel Hill, snoeyink@cs.unc.edu; on leave at the US National Science Foundation CISE CCF, jsnoeyin@nsf.gov separately the blues. In SI7–8 you will be able to merge red and blue orders at event flags, and discover each when you discover its first witness. In the end we will be able to merge red orders and blue orders when witnesses show the order has changed.

Red/blue segment intersection is a fundamental step of map overlay in GIS and Boolean operations in 2D CAD/CAM. The classic Bentley-Ottmann line sweep for segment intersection [2] needs five times the input precision to compare event orders, so it can fail due to numerical problems even when the algorithm has been proved correct. If we had three colors, even a small triangle of red, blue, and green crossing segments could require four times the input precision to determine the order of the intersections. Nevertheless, you will be able to create, in about eight stages, a sweep algorithm for red/blue segment intersection that needs only double precision.

3 A staged solution

Each stage asks you to generate *Segment files*: text files containing red or blue line segments for testing. The first line of a file will be the number of its segments, n, followed by n lines of pairs of 20-bit integer coordinates $px\ py\ qx\ qy$, where $-2^{20} < px, py, qx, qy < 2^{20}$. Segments of the same color may touch, but not cross.

Your programs should be able to read a known *Tester file* with lines like 'R1segs.txt B1segs.txt 1000 20' that specify filenames of red and blue segment files, the number of times to repeat the algorithm after reading the files, and an optional number or filename that contains the expected result (here 20). Output the average running time (elapsed time/repetitions) and VERIFIED if results match, or the first non-matching line. Note that when timing algorithms we typically don't want to include the overhead of reading and ASCII conversion and do want to repeat the run several times since the software-accessible clocks usually have coarse granularity. You may want to be able to pass flags to tell your program to print, draw, or animate its actions, as an aid to debugging, but be sure that there is no drawing or printing when run with no flags for timing.

3.1 SI1: test if two segments cross

Read David Douglas' 1974 article [4], "It makes me so CROSS," and count the cases that complicate a test of whether two segments intersect. Let's precisely define the most generic case: line segments pq and rs *cross* if and only if they share exactly one common point that is not an endpoint of either segment.

Derive and implement an algorithm that takes 20-bit integer coordinates for the endpoints of two segments, pq and rs, and returns a Boolean indicating whether they cross. While you can express the test as a calculation on coordinates, you may prefer to use higher level abstractions of points and vectors.

Implement `BFyourname` to count crossings in the examples listed in `BFTester.txt` by brute force, applying your test to every red/blue pair. Segment and Tester files are provided. Also create two Segment files of your own with the matching tester line: `R1yourname.txt B1yourname.txt 1 #` of crossings.

Remark: With 20-bit coordinates as input, orientation determinants in IEEE 754 doubles are evaluated exactly. Parameterizing one line, plugging into the other, and solving for zero, as suggested on many web pages, requires more precision. The geometric interpretation of Euclid's GCD algorithm [9] can help create instances for which rounding error causes pairs of segments to be incorrectly classified.

3.2 SI2: Refactoring

Read chapters 1 and 3 of "Refactoring: Improving the design of existing code" [6] and refactor your code (testing at each change) so that it has classes or structs for Point and Segment and constructors (Point from coordinates; Segment from Points) and comparisons (Point LessThan: $p.<(q)$ compares x, breaks ties by y; Segment side: $st.\text{side}(p)$ tests if point p is left, right, or on the line directed from s through t.)

You may have other classes (Vector, Matrix, Line. . .), or methods (construct from string or stream, draw, calculate slope, intercept, or intersection points. . .) You should have no coordinate computations in the main code. Handle `BFTester.txt` as in SI1.

Remark: This entire assignment is really an exercise in refactoring, an important design skill in computer science.

3.3 SI3: Sorting flags

Let's define a *flag* as the use of an endpoint by a segment. E.g., A segment pq with $p.<(q)$ (comparing x, breaking ties by y) has a *start flag* touching p and a *terminal flag* touching q. A flag should know its segment. In Fig. 2, points n and q are used by two start flags each, p and r by two terminal flags. Point vk is used by two starts and one terminal.

Sort flags by these tie-breaking rules: Compare endpoint x, break ties by y; for flags using the same point, put terminal < start, then increasing slope, and finally color, so red term < blue term and blue start < red start. If colors are the same, throw an "overlapping lines of same color" error or exception.

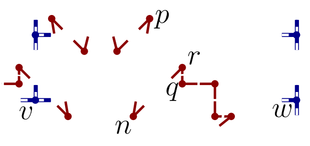


Figure 2: Example flags

Your program `SFyourname` should read SFTester.txt and produce, for each pair of red & blue inputs, their flags in sorted order, one per line, with the segment#, color R/B, and end S/T. You may write your own sort, or use a library routine.

Remark: The sorted flags become the events for a sweep algorithm in SI4+6; looking ahead can help justify the tiebreakers—they are chosen to be consistent with an infinitesimal perturbation that removes degeneracies [1].

3.4 SI4: Verify non-crossing segments

Red/blue line segment intersection is easier if there are no blue segments. Given a set of (supposedly) non-crossing red segments (ignore blues in Tester), return true if all are non-crossing, otherwise return segment numbers for a pair of segments that touch. If segments touch but do not cross, you may choose either output. SI6 will ask you to refine the handling of degenerate cases. `DSyourname` should process `DSTester.txt`.

- For efficiency, sweep a vertical line left to right, and maintain a list of the red segments in the vertical order that intersect the *sweepline*.
- The *sweep events* at which the list structure must change are start flags (where you add one segment) and term flags (where you delete one).
- The *list data structure* can start as an array or linked list, but for efficient update you'll eventually want a dynamic tree (AVL, red-black, or splay), skip list, or equivalent.
- Add red *sentinel* line segments, just above and below the bounding box of all other segments. Initialize your list with sentinels so every original point is between two line segments. (Sentinel endpoints are not sweep events.)

Thus, your list starts with the two sentinels. In Fig. 3, the first event is the start flag for af, adding af between the sentinels. The second is the start flag for ab, adding ab. Before the middle, the list has fk and $i\ell$ between the sentinels; jk starts and is added, fk terminates and is removed, jk terminates, ko starts, $i\ell$ terminates, and ℓm starts; list has ko and ℓm between sentinels.

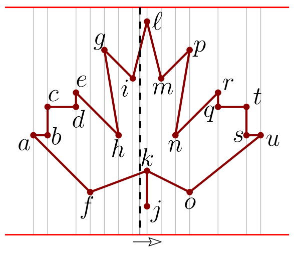


Figure 3: Sweep example

Remark: I agree with Raimund Seidel, who, in his 1993 invited CCCG talk entitled "Teaching Computational Geometry," said that any presentation of a sweep algorithm should start with invariants, which determine the events, which determine the data structures. Invariants help us write correct code and good test cases. Notice how sentinel segments avoid special handling for flags above or below all segments.

3.5 B05: Marking polygon or map regions

A variation on verifying that segments are non-crossing can allow us to perform Boolean operations (union, intersection, difference) on red and blue polygons. Suppose segments of each color can be partitioned into closed loops of segments that meet at endpoints. A point p in the plane can be declared outside or inside by an even/odd rule: it is outside if an even number of segments is crossed by a ray from p that does not pass through any segment endpoints segments, and inside if an odd number is crossed.

Add a *inside* bit to every red segment. Figure out how to modify the sweep of SI4 to set these to indicate whether the inside of the red polygon is to the left or right in the neighborhood of each red segment. If marks cannot be consistently assigned (e.g., jk in the maple leaf of Fig. 3, or shapes like θ) then return a segment number that cannot be assigned, else return true.

Another variation comes from map overlay in GIS, where regions bounded by segments may have different ids to indicate land ownership or usage. Suppose that every segment comes with integer leftId and rightId fields already set. Modify the sweep of SI4 to verify that all fields around a region are set to the same integer. Here any isolated edge has leftId=rightId, and a θ shape is fine. Notice that regions may have holes, so it is not quite enough to compare only segments that touch.

Remark: A final variation is to assign consistent leftId and rightId values during a sweep. This a good union-find exercise, since one does not know if two segments bound the same or different regions when they are first encountered. I prefer the above simpler variants, because the next stages of segment intersection are quite technical.

3.6 SI6: Breaking segments containing endpoints

This stage eliminates the degeneracy from an endpoint landing in the middle of a segment by simply breaking the segment into two; Each piece is still defined by endpoints with 20-bit integer coordinates. We cannot break at intersection points in general, because those require rational coordinate and more precision.

If we separately pre-break red segments and blue segments, each at both the red and the blue endpoints, then the only remaining degeneracies are shared endpoints, which are mostly handled by the sort order from SI3.

The task of SI5: Given non-crossing blue segments and red flags, return a list of all blue segments, broken so none contains any flag point. This is conceptually a small modification of SI4, using the sorted list of both red and blue flags for events. If, at an event, you detect that a flag point of either color lands inside a blue segment, you terminate that blue segment there and start a new blue for the remaining piece. (Red segments give flags, but are otherwise ignored.) If you detect a blue/blue crossing, report error.

It does require careful thought to realize that, thanks to our sorting tie-breakers, the degenerate cases of shared endpoints are handled by the non-crossing code of SI4. Only when a point lies inside a segment is work required.

Remark: Only those who did not complete SI4 need submit this, so it gives a breather to the TA and a chance for students to catch back up.

3.7 SI7: Compute above= & below= for each flag

Assume we are given red and blue segments with no red/red or blue/blue crossings, and no endpoints inside segments. For each flag, create four pointers (red/blue × above=/below=) that point to the segment in the list of that color that uses the flag or is above/below just before the flag event. You may assume that sentinels have been added as in SI4, so every flag has both colors above and below. These pointers will help us find intersections witnessed by the flag.

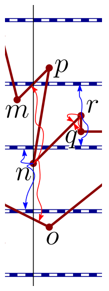


Figure 4: above=, below=

Notice that for any terminal flag, like *nr* in Fig. 7, its own segment will be both above= and below= for its color. its segment. For a start flag (and for opposite color pointers) above= or below= will always be segments already in the list. E.g., in Fig. 7 start flag of *nr* has red above= pointing to *mp* (since *np* is not yet present) and below= to *ko*. Start *np* has the same red above=, but below= is *nr*.

Your code, `AByourname` should produce a list, for each red & blue flag (in their sorted order) of the segment numbers (low to high: 0 for sentinel below all segments to $n+1$ or $m+1$ for sentinel above) for: same color below=, same color above=, opposite below=, opposite above=.

Remark: Yes, this is technical definition with odd cases. Students need not worry at this stage why it will be useful; we first need to know more about witnesses.

4 Intermezzo: define orders, avoid degeneracies

We are almost ready to create a sweep algorithm for segment intersection, but we need the concepts that capture order during a sweep. Let's make initial definitions by ignoring the degenerate cases from endpoints that share x coordinates. Labels apply to segments in Fig. 5.

1. Segment r is *comparable to* b at x iff both intersect a vertical sweepline through $(x, 0)$. E.g, b, r, s are all comparable between start s and term r.
2. The *start order* for comparable segments is whether second start is above/below first line. E.g., b starts above r, and s starts above b.
3. If r and b cross, their *witness* is the first endpoint showing change from the start order. E.g., start s witnesses $r \cap b$, and term b witnesses $b \cap s$.
4. Finally, for segments comparable at x, we say that r is *above* b iff either (r starts above b or b below r) and no event $\leq x$ witnesses their crossing, or (r starts below b or b above r) and some event $\leq x$ witnessed their crossing. Arrows indicate the ranges.

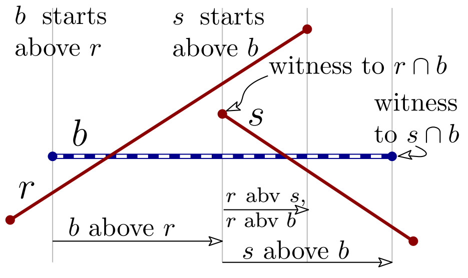


Figure 5: Definintions

For the comparable segments at an event x, *above* is a total order. (We would need to show transitivity: that if $a \succ b \succ c$ at x then $a \succ c$ at x. Can you do it? Draw the many cases and observe how you can push intersections to witnesses.)

We can restate the definitions to consistently handle the degenerate cases, assuming that segments have been broken at flag endpoints and flags are sorted.

1. Segment r is *comparable to* b at sweep event x iff both start flags and neither terminal flag occurred before x. (By the term $<$ start tiebreaker, segments terminating at p are never comparable to those starting at p.)
2. The *start order* for comparable segments is whether second start is above/below first line. (By the slope tiebreaker, and pushing r.term $<$ b.term, b.start $<$ r.start, any equal can be treated as 'above.' (If your tiebreakers are different, you may not be able to treat $=$ as 'above.' I chose my tiebreakers to avoid comparisons (and pipeline stalls) in the inner loop.)
3. If r and b cross, their *witness* is the first flag showing change from the start order. E.g., start s witnesses $r \cap b$, and term b witnesses $b \cap s$.
4. Finally, for segments comparable at a sweep event x, we say that r is *above* b iff either (r starts above b or b starts below r) and no event that witnesses their crossing is at or before x, or (r starts below b or b start above r) and the witness to their crossing was at x or before.

By careful definition of concepts and tiebreakers, we fold degeneracies into the general case handling.

5 SI8: segment intersection

The (pen)ultimate task: record, for each red and blue segment, the list of other segments crossing it in order.

We may assume that we are given: red and blue segments (no red/red or blue/blue crossings, no endpoints in segment interiors), the sorted list of flags for non-sentinel segments, and, for each flag, the four pointers to the above= and below= segments of each color.

Invariant: Maintain the list, ordered by aboveness at event x, of the segments starting but not terminating before x.

Only when we reach a flag may work need to be done to restore the invariant, so flags are the *sweep events*. The work is: to find intersections witnessed by the event's flag, to reorder by the new aboveness, and to start or terminate the flag's segment. Let's look closely at these to decide what operations a data structure will need to support.

The above= and below= pointers define intervals where the event can lie in the red and blue lists (for terminal flags, the interval of the flag color degenerates to a single point.)

If these intervals overlap, then no intersections are witnessed. If they do not, then w.l.o.g. let β denote the blue below pointer that happens to be above α, the red above pointer. Reorder all blue and red segments between β and α so all reds segments, ending with α, are above the event flag, and all blue segments, starting with β, are below. All inversions are crossings witnessed by the event point.

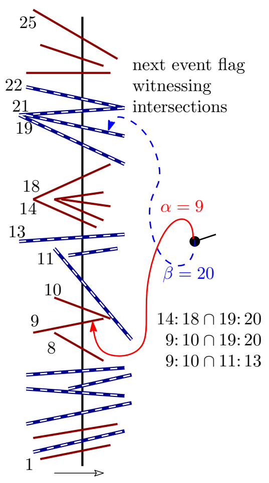


Figure 6: Non-overlapping intervals show intersections

For $2n$ segments having k intersections, we are aiming for $O(n \log n + k)$ time, so we can afford $O(\log n)$ per event plus $O(1)$ per intersection found. (It is even possible to achieve $O(n \log n)$ time by reporting intersections compactly as between dynamic bundles.)

5.1 Data structure operations

You may choose what data structure you want to use to maintain the ordered list of segments. The operations you need to support are insert and delete, deciding the order of intervals, and reordering segments and crossings. Consider the data structure options to:

1. *Decide interval order:* Given two red and blue items in the list, decide if they interleave.
    - (a) *Use ranks:* If we can obtain ranks for segments, we can compare order by comparing ranks. Time: $O(\log n)$ per event.
    - (b) *Use splay:* If we splay segments to the top of the tree, then we can compare their order at the root. Time: amortized $O(\log n)$ per event.
    - (c) *Use pred/succ:* If the intervals overlap, then our red or blue above must be the successor of our red or blue below. If they don't overlap, then we won't have this successor relationship, but know that we can go search in parallel to find which interval is after the other. Time: $O(1 + k)$ for k witnessed intersections.
    - (d) *Bundle tree:* Use a balanced tree for each same-color bundle, and keep these trees in a balanced tree. (Can do in a single splay tree with care.) Find intersections between bundles in bulk so whole algorithm is $O(\lg n)$.
2. *Rearrange interval:* red and blue are interleaved in the interval; we separate.
    - (a) List: opposite of a merge, which can be done in linear time in the number of inversions, or the number of segments involved.
    - (b) Tree or splay tree: Split the tree (esp. easy after splay) to extract the list of segments to be reordered. Build a balanced tree and reassemble.
    - (c) Bundle tree: Can report crossings in batches and reorder in $O(\log n)$ per bundle, rather than charging per segment: $O(n \log n)$ in total!

The desired output is a list of pairs of segments, one red, one blue, that intersect. These can be globally in any order, but the ones on a single segment should appear in left to right order (as usual, ties broken by y.)

5.2 Reference implementation

A MATLAB reference implementation of segment intersection[1] with prebreak[2] is available. MATLAB has good support for manipulating vectors and matrices, and poor support for dynamic or tree-like data structure. You will not want to mimic this implementation directly, since the linear lists it uses for most data structures can be replaced by dynamic search trees in other languages. However it does let you see the results that your program is expected to compute. Debug assertions and graphics code, which are commented out, can be reactivated to help keep code correct when refactoring.

Segment and Tester files are also made available, both those from the instructor and those created by fellow students.

Remark: A sequence of reference implementations is important for the rather large fraction of students who need to develop their skills for interpreting precise written specifications of a problem to be solved. A set of slides is available[3]; since generating test sets is part of the assignment, instructors may contact the author to request copies. The reference implementation for Boolean operations on polygons has not yet been student tested, so is not posted here.

[1] `http://tinyurl.com/segint6`
[2] `http://tinyurl.com/prebreak`
[3] `http://raindrops.in/view?id=54084b864251df7f2f8b4567`

5.3 BO9: Boolean operations on polygons

Stages BO5 and SI8 have set up almost everything we need to perform Boolean operations – to compute a union, intersection, or set difference of a sets of red and blue polygons. The only thing lacking is the output format when intersection points appear on the boundary of the result. We have the location of an intersection as a pair of segments (with integer endpoints). If we want to feed the output back in as input, we would have to either create a new algorithm that also accepts intersection points as input, or round rational coordinates down to integers in a way that is compatible with our application.

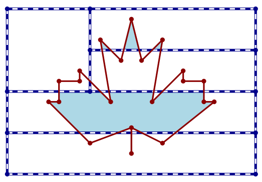


Figure 7: Intersection: leaf with two stripes

Remark: Fortune [5] identified that geometric rounding may need to be separated from the algorithm implementation. It is worth implementing the Boolean operations and drawing the results and realizing that there remain open research questions in geometry.

Acknowledgment

I am indebted to Steven Love for his hours of discussions with me and with students as my teaching assistant for this multistage project, and to Raimund Seidel for his work on perturbations to remove degeneracies and thoughts on teaching computational geometry.

I would appreciate hearing of other attempts to use multistage assignments of geometric algorithms.

This research has been supported by the US National Science Foundation through grant 1018498 and the IRD program while I serve as a program director in NSF CISE CCF Algorithmic Foundations.

References

[1] Pierre Alliez, Olivier Devillers, and Jack Snoeyink. Removing degeneracies by perturbing the problem or perturbing the world. *Reliable Computing*, 6(1):61–79, 2000.

[2] Jon L Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, 100(9):643–647, 1979.

[3] Jean-Daniel Boissonnat and Jack Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. In *Proc. 15th SoCG*, pages 370–379. ACM, 1999.

[4] David Douglas. It makes me so CROSS. In D. Marble and D. Peuquet, editors, *Introductory Readings in Geographic Information Systems*, pages 303–307. 1990. Reprinted from 1974 technical report.

[5] Steven Fortune. Robustness issues in geometric algorithms. In *Applied Computational Geometry: Towards Geometric Engineering*, number 1148 in LNCS, pages 9–14. Springer, 1996.

[6] Martin Fowler, Kent Beck, J Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the design of existing programs.* Addison-Wesley Reading, 1999.

[7] Imre Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery.* Cambridge, 1976.

[8] Andrea Mantler and Jack Snoeyink. Intersecting red and blue line segments in optimal time and precision. In *JCDCG '00: Japanese Conf. on Discrete and Computational Geometry*, volume 2098 of *LNCS*, pages 244–251. Springer, 2001.

[9] David L. Millman and Jack Snoeyink. Computing planar Voronoi diagrams in double precision: a further example of degree-driven algorithm design. In *Proc. 26th SoCG*, pages 386–392. ACM, 2010.

Index