Efficient Algorithms for Selected Constrained Center Location Problems

by

Amirhossein Mozafari Khameneh

M.Sc., Sharif University of Technology, 2012
B.Sc., Sharif University of Technology, 2009

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Amirhossein Mozafari Khameneh**
Degree:	**Doctor of Philosophy**
Thesis title:	**Efficient Algorithms for Selected Constrained Center Location Problems**
Committee:	**Chair:** Mohammad Tayebi Research Associate, Computing Science

Abstract

Facility location problems are an essential family of problems in combinatorial optimization and computational geometry which has many applications in other fields of computer science like computer networks, robotics, etc. In this thesis, we study two classes of facility location problems. In the first part, we study the effect of a *beacon/repulsor* in a polygonal region and in the second part, we study the *proximity connected k-center problem (PCkCP).*

Beacons and repulsors are two types of actuators that appear in many applications such as sensor networks and robot motion planning. A beacon (resp. repulsor) in a polygonal region P is an object that can attract (resp. repel) point particles in P. We say a beacon $b \in P$ *attracts* a point $p \in P$ if the particle initially located at p finally gets to b under its attraction influence. b is called a *beacon kernel point* of P if it attracts any point in P. Similarly, for three points $r, p, t \in P$, we say a repulsor at r sends p to t if the particle initially located at p finally gets to t under the repulsion influence.

In our first result, we consider the *discrete beacon kernel problem (DBKP)* in simple polygons and provide a sub-quadratic time algorithm to solve it. In the DBKP, we are given a set of points $\mathcal{X} \subseteq P$ and the objective is specifying the beacon kernel points in $\mathcal{X}$. Also, we show how our method can be extended to the case where we replace $\mathcal{X}$ by a set of line segments inside P. In our second result, we consider the *particle transmitting problem.* In this problem, the objective is determining the points in the given polygonal domain that can be sent to a given target point by activating only one repulsor. We propose an efficient polynomial time algorithm for this problem.

Next, we define the *proximity connectedness condition (PCC)* for a set C of centers in a metric space. We say C satisfies the PCC if each pair of centers can communicate with each other via the other centers assuming any two centers sufficiently close to each other can directly communicate. In the PCkCP, we are given a set of demand points in a metric space and a positive integer k. The objective is locating k centers as close as possible to the demand points while satisfying the PCC. As our third result, we study the PCkCP when the underlying space is a path and provide a sub-quadratic time algorithm for the problem. Finally, we consider the proximity connected 2-center problem in the plane and propose an efficient polynomial time algorithm for it.

Keywords: computational geometry; combinatorial optimization; polynomial time algorithm; beacon kernel problem; k-center problem

Acknowledgements

I would like to thank Dr. Binay Bhattacharya and Dr. Thomas Shermer for supervising me to write this thesis. It was a great honour for me to have a chance to work with such knowledgeable professors. I am grateful for their time and efforts during the last several years to help me develop my ideas for solving the problems discussed in this thesis.

I would like to thank Dr. Tsunehiko Kameda for his collaboration in the fourth chapter and his constructive suggestions. I would also like to express my gratitude to Dr. Pavol Hell, Dr. Sandip Das and Dr. Igor Shinkar for their useful comments on the thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

Facility location is one of the most important class of problems in combinatorial optimization that has many applications in computer science and industry. In a facility location problem (FLP), we are given a set of demand points $\mathbb{P}$ and a set $\mathbb{L}$ of possible locations for establishing facilities in an underlying space. We also have a *validness condition* on the subsets of $\mathbb{L}$ and any subset of $\mathbb{L}$ that passes the validness condition is called a *solution* for the FLP. Let us denote the set of all solutions for the FLP by $\mathbb{S}$. The objective is to find a solution that satisfies desired properties with respect to the demand points. A Center location problem (CLP) is a FLP for which each solution $S \in \mathbb{S}$ induces a *cost* to every demand point $p \in \mathbb{P}$. We denote the cost that a solution S induces on a demand point p by $cost_S(p)$. The cost of S is defined by a non-negative real function on $\{cost_S(p) : p \in \mathbb{P}\}$ and is denoted by $cost(S)$. The objective is obtaining a solution $S \in \mathbb{S}$ such that $cost(S) = min\{cost(S') : S' \in \mathbb{S}\}$. We call such a solution, an *optimal solution* for the CLP.

Many problems in computer science can be formulated as a CLP. For example, consider the median finding problem. In this problem, we are given a set A of numbers and the objective is to find an element $m \in A$ that minimizes the difference between the number of elements greater (or equal) than m and the number of elements smaller than m. In the CLP formulation, the underlying space, demand points $\mathbb{P}$ and locations $\mathbb{L}$ are all equal to the set A. The validness condition for a subset $U \subseteq A$ is $|U| = 1$ and for a solution $U = \{m\}$ and a point $a \in A$, $cost_U(a)$ is defined -1 if $a < m$ and $+1$ if $a \geq m$. Now, the cost of U can be defined as $cost(U) = |\sum_{a \in A} cost_U(a)|$. A solution m with minimum cost is in fact the median of A that can be found in linear time [63].

Center location problems have been extensively studied within the past decades [1, 26, 61]. The k-center and the k-median problems are among the most well-known such problems. In these problems, the underlying space is either considered as $\mathbb{R}^d$ (where d is the dimension) or a metric graph (a graph for which each edge has a length and the lengths satisfy the triangle inequality) with $\mathbb{P}$ as its vertices. $\mathbb{L}$ is defined as the entire space (continuous version) or a subset of points in the space (discrete version) and $\mathbb{S}$ is the set of all k-subsets (a subset of size k) of $\mathbb{L}$. For a solution $C \in \mathbb{S}$ and $p \in \mathbb{P}$, $cost_C(p)$ is defined as $d(p, C) = min\{d(p, c) : c \in C\}$ where $d(p, c)$ is the distance between the two points c and p. The difference between these two problems is that for the k-center problem, $cost(C) = max\{cost_C(p) : p \in P\}$ while for the k-median, $cost(C) = sum\{cost_C(p) : p \in P\}$. In many real world applications we need to impose additional conditions on C to fulfill the requirements arise from real world circumstances. The *capacitated* k-center problem can be considered as an example of such problems [8, 50]. In this problem, each center c has a capacity $cap(c)$ and when we establish a center c, the number of demand points assigned to c should not exceed $cap(c)$. In the majority of these problems, the problem does not consider the structural properties of the established facilities (centers) or the underlying space. In this thesis, we study two classes of CLPs for which we need to consider the structural properties of the established facilities or the underlying space.

In Part 1, we study the behaviour of points in polygonal regions (regions that can be represented by a set of polygons) in the plane under the attraction/repulsion force of an actuator. Consider the following situation: suppose that our target area can be represented by a polygonal region and we are interested to use a speaker to invite people to hand out some flyers. Therefore, when an individual hears the invitation voice, he or she greedily moves toward the source of the sound (he/she moves in a direction that maximize the reduction of his/her distance from the source). The question here is where we should reside such that anyone in the target area can get to us. This problem is called the *beacon kernel problem*. Figure 1.1 (a) shows an example of such situation. If we reside at point b, a person at point a can finally get to b while a person at point c will get stuck at d and can never reach b (because if he/she moves in any direction from d, his/her distance from b is increased).

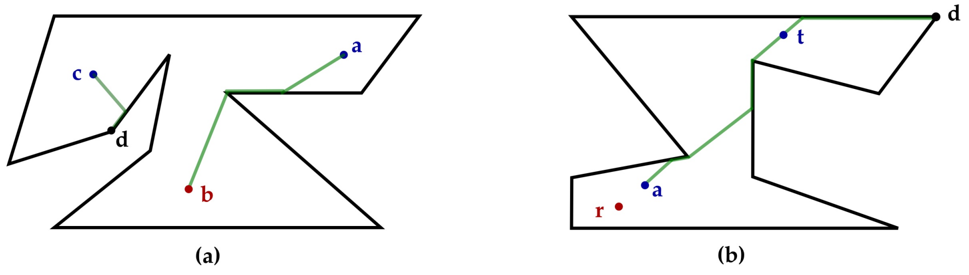


Figure 1.1: (a) Beacon b attracts a while not c. (b) Repulsor r sends a to t.

Such problems are modeled by beacon/particle terminology proposed by Biro [11, 12]. In this model, we assume that there is a particle at each point of the underlying polygonal region. Beacons are point objects (a beacon can reside in only one point) that can exert magnetic pull on the particles. For two points b and p in the region, we say b attracts p if the particle at p finally gets to b by activating a beacon at b. b is called a *beacon kernel point* if it can attract all the points in the region (for example in Figure 1.1 (a), b is not a beacon kernel point because it can not attract c). Repulsors can be considered as the reverse of beacons. When we activate a repulsor, each particle greedily gets away from it. Figure 1.1 (b) shows the traversal path of the particle at a when we activate a repulsor at r (this particle finally resides at d). If t is a point for which the particle at a gets to t by activating a repulsor at r, we say r sends a to t (see Figure 1.1 (b)).

We can formulate the problem of finding a beacon kernel point as a CLP as follows: let $\mathbb{P}$ be a set of points in the given polygonal region and consider $\mathbb{L}$ as the entire region. The validness condition of a subset $U \subseteq \mathbb{L}$ can be assumed as $|U| = 1$ (U contains only a single point). For a center $b \in \mathbb{L}$ and $p \in \mathbb{P}$, $cost_b(p) = +1$ if b can not attract p and zero otherwise. By defining $cost(b) = max\{cost_b(p) : p \in \mathbb{P}\}$, the CLP turns to finding a point b that attracts all the points in $\mathbb{P}$. So, if we assume that $\mathbb{P}$ has infinite size and uniformly distributed on the region, b has zero cost if and only if it is a beacon kernel point.

The *beacon kernel* of a polygonal region is defined as the set of its beacon kernel points. In 2013, Biro proposed an $O(n^2)$ time algorithm to compute the beacon kernel of polygonal regions and showed that this bound is tight [11]. In Chapter 2, we study a variant of this problem called the *discrete beacon kernel problem (DBKP)* and provide a sub-quadratic time algorithm for the problem when the underlying polygon is simple. In this problem, we are given a simple polygon and a set of points in it. The goal is determining beacon kernel points among the given points. In addition, we extend our algorithm to solve the *semi-discrete beacon kernel problem (SDBKP)*. In this problem, instead of points, we are given a set of line segments inside the polygon and the objective is determining the beacon kernel points on the segments. We propose sub-quadratic time algorithm for this problem and show that the SDBKP can determine the beacon kernel points on the boundary of the polygon. In Chapter 3, we consider the problem of sending a particle to a target point in polygonal regions. We call this problem the *particle transmitting problem*. In this problem, we are given a polygonal domain (polygonal region with polygonal holes in it) and a target point inside it. The problem is specifying the points in the domain that can be sent to the given target point by activating only one repulsor. We show that this problem can be solved in polynomial time and propose an efficient polynomial time algorithm for it.

In the second part of the thesis, we investigate the k-center problem when we have the *proximity connectedness condition (PCC)* on the centers. Consider the following scenario: let $\mathbb{P}$ be the locations of a set of buildings in a neighborhood and we are going establish k retail stores. Each store has a wireless device that enables it to communicate with any other store in the range δ of itself. We are interested to establish the stores such that:

1. If a customer asks for an item from a store that is unavailable, the store should be able to request that item from any of the other $k - 1$ stores.

2. The maximum distance of a building to its closest store is minimum.

We can see that in order to have the first condition, any pair of stores should be able to communicate with each other via the other stores. We call this condition, the proximity connectedness condition (PCC) with respect to the parameter δ. Generally, let C be a set of points in a metric space. The δ-distance graph of C is defined as an undirected graph for which there is a node ν_c corresponding to each point $c \in C$ and for two points $c_1, c_2 \in C$, there is an edge between ν_{c_1} and ν_{c_2} if $d(c_1, c_2) \leq \delta$ ($d(c_1, c_2)$ is the distance between c_1 and c_2). We say C satisfies the PCC with respect to δ if its δ-distance graph is connected. Figure 1.2 shows an example of a set of points satisfying PCC.

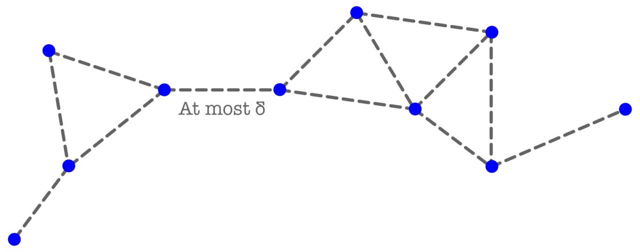


Figure 1.2: A set of points satisfying the PCC in the plane.

In the *proximity connected k-center problem (PCkCP)*, we are given a set $\mathbb{P}$ of demand points in a metric space, a positive integer k and a positive number δ. The objective is to find k centers C^* in the space such that the maximum distance of a demand point from its closest center is minimum and C^* satisfies the PCC. Figure 1.3 shows an example of the proximity connected 3-center problem and its optimal solution.

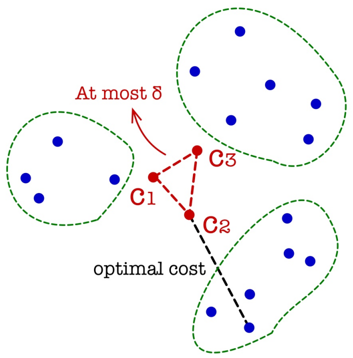


Figure 1.3: An example of the proximity connected 3-center problem in the plane. The green dashed lines classifies the points closest to each center.

Note that when δ goes to infinity, the problem turns to the (unconstrained) k-center problem. Because the k-center problem is NP-hard in metric graphs and Euclidean space [32, 53], the PCkCP is also NP-hard in these spaces. In Part 2, we study two spacial cases for the PCkCP. In Chapter 3, we study the PCkCP when the underlying space is a path and provide a sub-quadratic time algorithm for the problem and finally, in Chapter 4, we study the proximity connected 2-center problem in the plane and provide an efficient polynomial time algorithm for it which improves the running time of the previous algorithms for this problem.

Note: In order to simplify the notations we use in this thesis, we assume that the scope of each notation that we define is within its corresponding chapter and we may redefine and use them differently in other chapters.

Part I

Beacons and Repulsors in Polygonal Regions

Chapter 2

A Sub-quadratic Time Algorithm for the Discrete and Semi-Discrete Beacon Kernel Problem in Simple Polygons

In 2011, Biro et al. [12] initiated the concept of *beacon attraction trajectory* motivated by routing messages in sensor network systems. Computing the beacon kernel points of simple polygons in sub-quadratic time is a long standing open problem since it was initially proposed by Biro in 2013 [11]. In this chapter, we consider this problem by studying the *discrete beacon kernel problem (DBKP)* and the *semi-discrete beacon kernel problem (SDBKP)* for simple polygons. In the first problem, we are given a simple polygon P with n vertices and a set $\mathcal{X}$ of m points in it. We provide an $O(m \log m + nm^{\log_4 3})$ time algorithm to determine the beacon kernel points in $\mathcal{X}$. For the second problem, instead of $\mathcal{X}$, we are given a set $\mathcal{S}$ of m line segments in P. We propose an $O((n + m)^{1+\log_4 3} \log^2(n + m))$ time algorithm to determine the beacon kernel points of P that lie on the segments in $\mathcal{S}$. Finally, we show that by spending $O(n^{1+\log_4 3} \log^2 n)$ time, we can determine the beacon kernel points on the boundary of P.

2.1 Background and Previous Works

Studying the behaviour of point particles in a polygonal region under the influence of an attraction actuator called *beacon* is an active area in computational geometry due to its application in various branches in computer science such as robot motion planning and network systems [5, 42, 46]. The problem first appeared in the context of sensor network systems in the early 2000s [42]. Consider a network of sensors in a polygonal region P that gathers information and sends it to a destination point b in the region. Each sensor has a range and only can pass a message to the sensors within its range (we call these sensors *neighbor sensors*). Greedy routing protocol is widely used in such circumstances as each sensor only needs to know the location of itself, the destination point and its neighbor sensors. Specifically, each sensor passes its message to the neighbor sensor that is closest to b (if all the neighbor sensors are farther from b than itself the sensor does not pass its message). Two main problems can arise here. The first problem is determining the sensors that can successfully send their messages to the destination using the above greedy protocol. The other is determining the locations for b such that all sensors can successfully send their information to the destination. If we assume that P is uniformly filled with sensors and the range of each sensor is infinitely small, for each pair (p, b) of points in P, we can assign a path inside P that indicates the trajectory of a message that the sensor at p tries to send to b. Note that, p can successfully send its message to b if and only if this path ends up at b.

In the context of robot motion planning, the polygonal region P can be interpreted as the underlying area of interest. The location of a robot is represented by a point inside P and a beacon in P is a device that when activated at a point b, it guides any robot in the area toward itself by sending signals. When a robot receives the signals, it can detect the direction for which they are coming. Then, the robot continuously moves in a direction that satisfies the following conditions:

1. It remains inside the polygon.
2. It maximizes the reduction of its distance from the beacon (greedy moving).

The robot continues its movement until either it reaches b or gets stuck in a point in P for which moving in any direction (while remaining in P) increases it's distance from b. In the later case, the robot can never reach b. In this framework, the two main questions is as follows: first, given a beacon at b, determine the locations for which a robot can get to b by activating the beacon and second, find a location b such that by activating a beacon at b, any robot in P can get to the beacon.

In general, we use beacon/particle terminology to model such problems. In this terminology, beacons and particles are *point objects* which means at each time they can reside in only one point of P. A beacon is an attraction actuator that exerts magnetic pull on the particles. We assume that initially there is a particle at each point in P. Without any confusion, when we say *a point p in P*, based on the context, we either refer to the location p in the polygon or the particle initially lies on p. When we activate a beacon at a point $b \in P$, the particles inside P greedily move toward b with unit speed. In this model, we assume that the particles do not interact with each other. Specifically, consider the particle that initially resides at a point $p \in P$. Let p_t be the location of the particle at time t and $\vec{x}_t$ be the unit vector from p_t toward the beacon. At each time t, the particle moves along a unit vector $\vec{z}_t$ such that:

1. $\vec{z}_t$ (located at p_t) points to the inside of P (including its boundary).

2. $\vec{z}_t \cdot \vec{x}_t > 0$ ($\vec{z}_t \cdot \vec{x}_t$ is the inner product of $\vec{z}_t$ and $\vec{x}_t$).

3. $\vec{z}_t \cdot \vec{x}_t$ is maximum over all unit vectors satisfying the first two conditions.

We call above conditions the *greedy movement conditions* with respect to b. If at any time t, there is no direction satisfying the first two conditions, the particle will be stranded and can not reach b. An immediate observation here is that p can visit each point of P at most once during its traversal. This is because based on the greedy movement conditions, the distance of p from the beacon should be a decreasing function of time. For a pair (b, p) of points in P, we say *b attracts p* if by activating a beacon at b, p can finally get to b. Otherwise, we say b does not attract p. A path $\pi \subseteq P$ is called an *attraction path* of p under the influence of b if:

1. π starts at p.

2. At each point of π, the direction (unit vector) of the particle's movement satisfies the greedy movement conditions with respect to b.

Note that such an attraction path may not be unique. This situation happens when the particle at p hits a reflex vertex with incident edges e_1 and e_2 such that $d(b, sup(e_1)) = d(b, sup(e_2))$ ($sup(e)$ is the supporting line of an edge e). We denote the set of attraction paths of p under the influence of b by $\pi_b(p)$. Based on this definition, b attracts p if and only if there is a path in $\pi_b(p)$ that ends at b. Given a point b, the attraction region of b denoted by $A(b)$ is defined as the set of points in P that can get attracted by b. Conversely, the inverse attraction region of a point $p \in P$ denoted by $IA(p)$ is defined as:

$$IA(p) = \{b \in P : p \in A(b)\} \tag{2.1}$$

We can see that for any point $x \in P$, both $A(x)$ and $IA(x)$ contains the visible region of x in P (for two points $x, y \in P$, y is in the visible region of x if the connecting line segment between x and y completely lies in P). A *beacon kernel point* of P is a point $b \in P$ such that $A(b) = P$ (b attracts any point in P) and the *beacon kernel* of P denoted by $Ker(P)$ is defined as the set of all beacon kernel points in P. We call a point $d \in P$ a *dead point* with respect to b if $d \neq b$ and $\pi_b(d)$ only contains the single point d. Based on this definition, it is easy to see that for any pair (p, b) of points in P, if a path $\pi \in \pi_b(p)$ does not end at b then the endpoint of π is a dead point. Another observation here is that if d is a dead point of P with respect to a beacon b, then $d \in \partial P$ (the boundary of P) otherwise, the particle at d can move in the direction toward b. The *dead region* of a dead point $d \in P$ with respect to b is defined as,

$$\begin{aligned} DR_b(d) = \{p \in P : \exists\, \pi \in \pi_b(p),\ d \text{ is the endpoint of } \pi \\ \text{and } \ \neg\exists\, \pi \in \pi_b(p) \text{ that ends at } b\} \end{aligned}$$

We can see that a point p can be attracted by b if and only if for any dead point d with respect to b, $p \notin DR_b(d)$. For a pair (p, s) of points in P, we say p can be *routed to* s by a sequence $B = (b_1 \ldots, b_m)$ of beacons if b_1 attracts p, $b_m = s$ and for any $1 \leq i < m$, b_{i+1} attracts b_i. Let us denote the line segment between two points a and b in the plane by $seg(ab)$. If we need a direction for the segment we assume that this direction is from a to b. A *chord* C for a simple polygon P is defined as a line segment $C = seg(ab)$ such that $C \subseteq P$, $\{a, b\} \subseteq \partial P$ and $int(C) \cap \partial P = \emptyset$ ($int(C)$ is the interior of C). Note that C divides P into two simple polygons which we call them the *sub-polygons* of P induced by C. If we consider the direction of C from a to b, we denote the sub-polygon on the right side of $seg(ab)$ by $P_1(C)$ and the other one by $P_2(C)$. We say a chord C *separates* two points x and y in P if $x \in P_1(C)$ and $y \in P_2(C)$ or vise versa (note that if one of the two points lies on C, then it will be on both sub-polygons and based on our definition, C still separates them). Figure 2.1 shows an example of an attraction path, beacon kernel, a chord and its sub-polygons in a simple polygon.

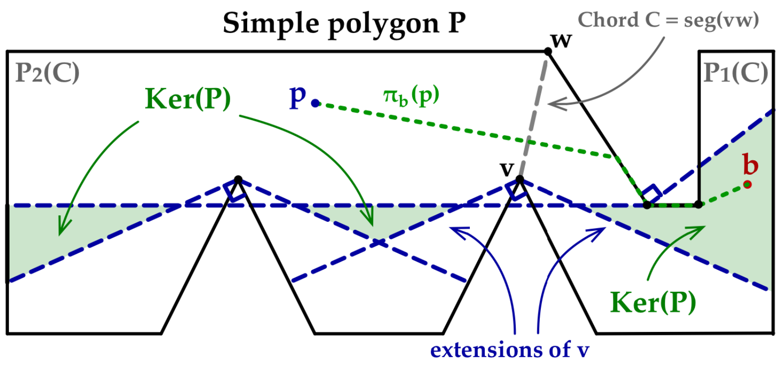


Figure 2.1: The attraction path of p under the influence of b is depicted by green dashed path. The green regions are the beacon kernel of P. As we can see, this region may not be connected. The (perpendicular) extensions of each reflex vertex is specified by blue dashed lines and finally, the chord C from v to w is determined by gray dashed segment. The sub-polygon on the right side of it is $P_1(C)$ and the one on its left is $P_2(C)$.

2.1.1 Previous Works

The concept of beacon attraction trajectory was first introduced by Biro et al. in 2011 [11, 12, 13] as a framework to address problems involving greedy routing toward a destination point. They showed that the attraction region of a point $b \in P$ can be computed in $O(n)$ (resp. $O(nh)$) time when P is a simple polygon (resp. polygonal domain) with n vertices (resp. n vertices and h holes). They also proposed an $O(n^2)$ time algorithm for computing the inverse attraction region of a given point p in polygonal domains. In addition, they proposed a naive $O(n^2)$ time algorithm for computing the beacon kernel of polygonal domains and showed that this bound is tight [11]. But, obtaining a sub-quadratic time algorithm for computing the bacon kernel of simple polygons remained open. As another study, in 2014 Biro et al. [14] showed that if P is a simple polygon (resp. polygonal domain) with n vertices (resp. n vertices and h holes) and (p, s) is a given pair of points in P, a sequence of $\lfloor \frac{n}{2} \rfloor - 1$ (resp. $\lfloor \frac{n}{2} \rfloor + h - 1$) beacons are always sufficient and sometimes necessary for routing p to s. Later, Shermer [60] improved this bound to $\lfloor \frac{n-4}{3} \rfloor$ for rectilinear simple polygons. In [47] Kouhestani et al. studied a related problem called *shortest beacon watchtower* problem. In this problem, a polygonal terrain T with respect to the x-axis is given. A watchtower W is defined as a vertical line segment with lower endpoint on T. We say W is a beacon watchtower if by activating a beacon at the upper endpoint b of W, any two points x and y in T can be routed to each other via b. Now, the question is finding a beacon watchtower with minimum possible length. They gave an $O(n \log n)$ time algorithm for solving the shortest watchtower problem where n is the number of vertices of T. In 2015, the authors presented an $O(n \log n)$ (resp. $O(n)$) time algorithm for computing the inverse attraction region of a point in monotone polygons (resp. polygonal terrains) [48]. In 2018, Kostitsyna et al. obtained an $O(n \log n)$ time algorithm for computing the inverse attraction region of a point in simple polygons and showed that this bound is optimal [49]. In 2019, Bae et al. [7] studied rectilinear polygons and showed that $\lfloor \frac{n}{6} \rfloor$ (resp. $\lfloor \frac{n+4}{8} \rfloor$) beacons are always sufficient and sometimes necessary to attract any point in simple (resp. monotone) rectilinear polygons. They also showed that the beacon kernel of rectilinear polygons can be computed in linear time. In [15] Bose and Shermer introduced the concept of *attraction-convex* polygons. A polygon P is called attraction-convex if any point $b \in P$ can attract any point $p \in P$ ($P = Ker(P)$). They provided a linear time algorithm to detect whether a simple polygon is attractive-convex. As another question in this context, it is interesting to know which points can be attracted to the beacon if instead of one static location, we allow the beacon to be at any point in a given region $R \subseteq P$. Biro [11] addressed this problem by introducing *weak attraction region* of R defined as,

$$WA(R) = \cup_{b \in R} A(b)$$

In [11], Biro gave an $O(n^4(m + n^2)^4)$ time algorithm to compute $WA(R)$ in P where P is a polygonal domain and m is the number of vertices in R.

Despite recent studies on various problems regarding the beacon base trajectory of points in polygonal regions, providing a sub-quadratic time algorithm for computing beacon kernel points of simple polygons is still an open problem. In this chapter, we consider two variants of this problem: *discrete* and *semi-discrete* beacon kernel problem and present sub-quadratic time algorithms to solve these problems.

2.2 Preliminaries

In this chapter, we assume that P is the given simple polygon with n vertices. First, we provide a proposition that helps us to simplify the concept of an attraction path between a beacon and a point.

Proposition 2.1. *For two points $b, p \in P$, if b attracts p, the path in $\pi_b(p)$ that ends at b is unique.*

Proof. Suppose we have two paths π_1 and π_2 in $\pi_b(p)$ ending at b. Let t_1 and t_2 be the first points that the paths split and intersect again respectively. Note that t_1 should be a reflex vertex such that π_1 and π_2 follow different edges at t_1. Otherwise, there is only one way to move greedily (if $t_1 \in int(P)$, p directly moves toward b and if t_1 lies on $int(e)$ for some $e \in \partial P$, p moves toward the perpendicular projection of b on $sup(e)$. Also, if t_1 is a convex vertex, one of the paths should get farther from b at t_1 which is not possible). Now, the portions of π_1 and π_2 between t_1 and t_2 create a closed non-intersecting curve which contradicts the simplicity of P. □

In order to simplify our terminology, if b attracts p, we only consider the path in $\pi_b(p)$ that ends at b as the attraction path of p (with respect to b). Otherwise, we consider an arbitrary path in $\pi_b(p)$ as the attraction path of p. Based on this assumption, henceforth we use the notation $\pi_b(p)$ only for the unique attraction path of p with respect to b. As another notation, for an edge $e \in \partial P$, we denote the perpendicular projection of b on $sup(e)$ by $h_b(e)$.

Observation 2.1. *The attraction path of p with respect to b is a polygonal chain ($p = a_1, \ldots, a_m$) and satisfies the following conditions:*

1. *For any $i > 0$, a_i lies on ∂P.*

2. *For any $1 \leq i < m$, if a_i and a_{i+1} lie on an edge e (resp. different edges) of ∂P, $seg(a_i a_{i+1})$ is a sub-segment of $seg(a_i h_b(e))$ (resp. $seg(a_i b)$).*

3. *a_m is either b or a dead point of P with respect to b.*

This is because if p lies in the interior of P, it's greedy movement is directly toward b and if it is in the interior of an edge e, it reduces its distance to b by moving toward $h_b(e)$. Note that when a particle reaches an endpoint of e, it either jumps off from ∂P or again slides on the neighbor edge of e from the endpoint. Because $\pi_b(p)$ is a polygonal chain and has a start point p, we can assign an order to its points based on their distance to p along the path. So, for two points $x_1, x_2 \in \pi_b(p)$ ($x_1 \neq x_2$), we say $x_1 < x_2$ if x_1 is closer to p than x_2 along $\pi_b(p)$.

Observation 2.2. *If x_1 and x_2 are two points in $\pi_b(p)$ such that $x_1 < x_2$ then $d(x_1, b) > d(x_2, b)$.*

The above observation is a direct consequence of the greedy movement conditions. We call any path from p to b satisfying Observation 2.2, a *distance decreasing* path from p to b. Therefore, if there is no distance decreasing path from p to b, then b can not attract p.

Observation 2.3. *A point $d \in \partial P$ is a dead point with respect to b if an only if it satisfies one of the following two conditions:*

1. *$d = h_b(e) \in int(e)$ for some edge e in ∂P.*

2. *d is a convex vertex v with incident edges e_1 and e_2 such that b lies in the cone obtained by two half-lines l_1 and l_2 perpendicular to e_1 and e_2 respectively where l_1 (resp. l_2) lies in the half-plane induced by $sup(e_1)$ (resp. $sup(e_2)$) not containing e_2 (resp. e_1) (figure 2.2 shows an example of such situation).*

Note that when we have the second condition, $h_b(e_1)$ (resp. $h_b(e_2)$) lies on the side v on $sup(e_1)$ (resp. $sup(e_2)$) that does not contain e_1 (resp. e_2). So, any direction satisfying the second condition of the greedy movement, will violate its first condition (see Figure 2.2).

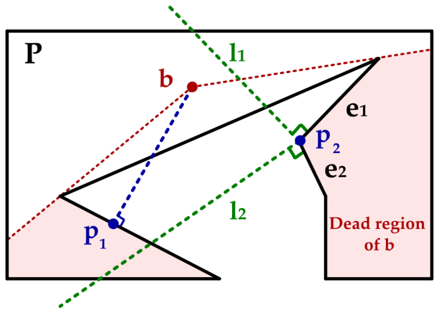


Figure 2.2: p_1 and p_2 are the two dead points of P with respect to b. p_1 lies on the interior of an edge and p_2 is a convex vertex. Also, the dead regions of p_1 and p_2 with respect to b are specified.

We call a subset $P' \subseteq P$ *convex with respect to* P if for any two points p_1 and p_2 in P', either $seg(p_1p_2) \subseteq P'$ or $seg(p_1p_2) \nsubseteq P$. In [11], Biro proved that if P is a simple polygon, for any $p \in P$, $IA(p)$ is convex with respect to P.

Corollary 2.1. *If P is a simple polygon, $Ker(P)$ is convex with respect to P.*

This is due to the fact that $Ker(P) = \cap_{p \in P} IA(p)$. Therefore, if x_1 and x_2 are two points in $Ker(P)$, they should lie on $IA(p)$ for any $p \in P$. Now, if $seg(x_1x_2) \subseteq P$, because $IA(p)$ is convex with respect to P, it should also lie in $IA(p)$ for all $p \in P$ and so $seg(x_1x_2) \subseteq Ker(P)$ which means the kernel is convex with respect to P.

Observation 2.4. *A point b is in $Ker(P)$ if it attracts all points in ∂P.*

The reason is that if b attracts all points in ∂P and $p \in int(P)$, then when we activate a beacon at b, p will either directly gets to b or it hits ∂P at some point t. Now, because b attracts t, p will eventually gets to b.

Let us denote the set of all reflex vertices of P by $\mathcal{R}$. For a reflex vertex $r \in \mathcal{R}$ and its incident edges e_1 and e_2, consider the two half-planes $H_1(r)$ and $H_2(r)$ induced by the lines perpendicular to $sup(e_1)$ and $sup(e_2)$ containing e_1 and e_2 respectively. The *dead wedge* of r is defined as the interior of the cone corresponding to $H_1(r) \cap H_2(r)$. We denote the dead wedge of r by $DW(r)$. Also, we define the *perpendicular extensions (for short extensions)* of r with respect to e_1 and e_2 as the two half-lines from r perpendicular to e_1 and e_2 respectively enclosing $DW(r)$ (see Figure 2.1). Based on Observation 2.1, if a beacon lies on $DW(r)$, it can not attract the points on at least one of the edges incident to r.

Proposition 2.2. *If d is a dead point of P with respect to b, then $\partial DR_b(d)$ (the boundary of $DR_b(d)$) is a polygonal region such that each of its edges is either an edge of ∂P or a chord from a reflex vertex.*

Proof. Suppose that $x \in \partial DR_b(d)$ in $int(P)$. Let x_0 be the point where x hits ∂P by the first time on its attraction path to d. Also, let C be the chord containing x along $sup(bx)$. Because x_0 is also the first hitting point of all points in C, each point in C is a boundary point of $DR_b(d)$ (because b sends all the points in C to x_0 and the points slightly on the left and the right sides of C get to different sides of x_0). This implies that $DR_b(d)$ is a polygonal region. If x_0 is an interior point of an edge e, either $x_0 = h_b(e)$ or $x_0 \neq h_b(e)$. For the first case, x_0 is a dead point and so all points of $int(e)$ in its neighborhood will get to x_0. This means that x can not be a boundary point of $DR_b(d)$ which is a contradiction. For the second case, all the points of $int(e)$ in the neighborhood of x_0 either get to $h_b(e)$ (if it is in e) or an endpoint of e and again x can not be a boundary point. Now, suppose that x_0 is a convex vertex with incident edges e_1 and e_2. The case x_0 is a dead point is similar but if not, both $h_b(e_1)$ and $h_b(e_2)$ should lie on the same side of x_0 in $sup(e_1)$ and $sup(e_2)$ respectively. This again implies that there is a neighborhood of x_0 such that all of its points reaches the same point which is a contradiction. Therefore, x_0 should be a reflex vertex. □

Proposition 2.3. *[11] A point $b \in P$ is a beacon kernel point if it is not contained in the dead wedge of any reflex vertex $r \in \mathcal{R}$.*

Proof. Suppose that $b \in P$ is not contained in any dead wedge of the reflex vertices $\mathcal{R}$ but it is not a beacon kernel point. So, there is a dead point $d \in P$ with respect to b. Based on the previous Proposition, $DR_b(d)$ should have a boundary on a reflex vertex r (to find such an r, we can traverse along ∂P from d until we reach a point in $A(b)$) and a chord C from it such that all the points on C hit r when we activate the beacon. Therefore, the points on one of the incident edges of r can not get attracted to b but this only happens when b is inside the dead wedge of r. □

If a point b lies in $DW(r)$, we say r eliminates b otherwise, we say b survives from r. Suppose that e_1 and e_2 are the incident edges of a reflex vertex r. Also, let C_{e_1} and C_{e_2} be the two chords from r along $sup(e_1)$ and $sub(e_2)$. These chords divides P into three sub-polygons. Let us denote the sub-polygon containing e_1 (resp. e_2) by $P(e_1)$ (resp. $P(e_2)$). Also, we call the sub-polygon containing neither of e_1 and e_2, the sub-polygon in front of r (see Figure 2.3).

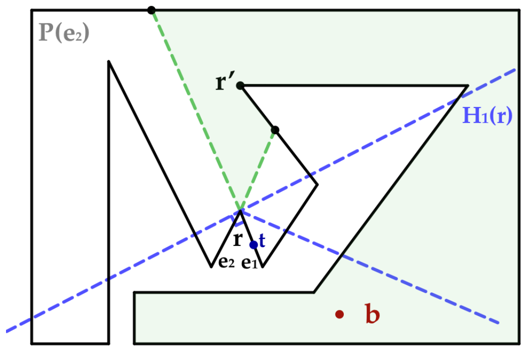


Figure 2.3: b is in the sub-polygon in front of r and lies in $DW(r)$.

Proposition 2.4. *For any reflex vertex $r \in \mathcal{R}$ with incident edges e_1 and e_2, we have:*

1. *No point in $H_1(r) \cap P(e_2)$ (similarly $H_2(r) \cap P(e_1)$), can be a beacon kernel point.*
2. *For a point b in the sub-polygon in front of r, if $b \in DW(r)$, there is a reflex vertex $r' \in \partial P$ closer to b such that $b \in DW(r')$.*

Proof. 1) If $b \in H_1(r) \cap P(e_2)$, it either lies in $DW(r)$ or $H_1(r) \setminus DW(r)$. In the former case, based on Proposition 2.3, b can not be a beacon kernel point and in the later case, any path from a point $t \in int(e_1)$ passing $sup(e_1)$ can not be distance decreasing with respect to b. Thus, based on Observation 2.2, b can not be a beacon kernel point. 2) suppose that $b \in DW(r)$. Here, any path from t to b needs to pass $sup(e_2)$ (see Figure 2.3. The case $t \in e_2$ is similar). In this situation, no path in P from t to b can be distance decreasing. Now, consider the portion of ∂P from r starting from e_1 up to the first visible point from b. For at least one reflex vertex r' in this portion and a chord C' emanating from r', the points on ∂P around r' goes into different sub-polygons with respect to C' (otherwise, b could attract t). But this case happens only if $b \in DW(r')$. □

In order to simplify our algorithm, we assume that the points are in general position by which we mean no three points are collinear and no two points have the same x-coordinate (these assumptions can be applied by slightly perturbing the points). First, we consider the discrete beacon kernel problem (DBKP) and introduce a data structure called *quaternary partition tree (QPT)*. We show how we can use this data structure to get a sub-quadratic time algorithm to solve the DBKP. Next, we show how we can use our algorithm for solving the DBKP to solve the semi-discrete beacon kernel problem (SDBKP) in a sub-quadratic time.

2.3 The Discrete Beacon Kernel Problem

Let P be the given simple polygon with n vertices and $\mathcal{X}$ be the given set on m points in P. In the DBKP, we are going to determine which points in $\mathcal{X}$ are beacon kernel points. The idea to solve the problem is applying Proposition 2.3 to the points in $\mathcal{X}$ and see which points of $\mathcal{X}$ survive from all reflex vertices in $\mathcal{R}$.

In order to solve the DBKP, we provide a data structure called quaternary partition tree (for short QPT) in the preprocessing phase. Using this data structure, given a half-plane H, we can determine the points of $\mathcal{X}$ that lie in $int(H)$ in sub-linear time. Indeed, the QPT is a quaternary tree $\mathcal{Q}$ (each of its internal nodes has four children) in addition with useful information about the points in $\mathcal{X}$. Specifically, for each node $\nu \in \mathcal{Q}$, we assign a region $R(\nu) \subseteq \mathbb{R}^2$ and a set of points $\mathcal{X}(\nu) \subseteq \mathcal{X}$ such that if $\mathcal{L}$ is the set of nodes in a level (we say two nodes in $\mathcal{Q}$ have the same level if they have equal distances to the root) of $\mathcal{Q}$, $\cup_{\nu \in \mathcal{L}} \mathcal{X}(\nu) = \mathcal{X}$ and for two nodes $\nu_1, \nu_2 \in \mathcal{L}$, $\mathcal{X}(\nu_1)$ and $\mathcal{X}(\nu_2)$ have almost the same number of points (the difference between $|\mathcal{X}(\nu_1)|$ and $|\mathcal{X}(\nu_2)|$ is at most one). We call $\mathcal{X}(\nu)$ and $R(\nu)$ the *pointset* and the *region* of ν respectively.

In order to build $\mathcal{Q}$, we first sort the points in $\mathcal{X}$ based on their x-coordinates. This step takes $O(m \log m)$ time. Next, we create a root node ν_r and assign the entire plane and $\mathcal{X}$ as its corresponding region and pointset respectively. Because we assumed that the points of $\mathcal{X}$ are in general position and sorted based on their x-coordinates, we can get a vertical line $l_1(\nu_r)$ that divides the points in $\mathcal{X}$ into two sets of almost equal sizes in a constant time. Let us denote these sets by $\mathcal{X}_1(\nu_r)$ and $\mathcal{X}_2(\nu_r)$. Also, we denote the half-plane induced by $l_1(\nu_r)$ containing $\mathcal{X}_1(\nu_r)$ (resp. $\mathcal{X}_2(\nu_r)$) by $R_1(\nu_r)$ (resp. $R_2(\nu_r)$). Let $l_2(\nu_r)$ be the ham-sandwich cut line [57] of $\mathcal{X}_1(\nu_r)$ and $\mathcal{X}_2(\nu_r)$. Therefore, $l_2(\nu_r)$ divides $\mathcal{X}_1(\nu_r)$ (resp. $\mathcal{X}_2(\nu_r)$) into two sets of almost equal sizes namely $\mathcal{X}_{11}(\nu_r)$ and $\mathcal{X}_{12}(\nu_r)$ (resp. $\mathcal{X}_{21}(\nu_r)$ and $\mathcal{X}_{22}(\nu_r)$). Similarly, we have four regions $R_{11}(\nu_r)$, $R_{12}(\nu_r)$, $R_{21}(\nu_r)$ and $R_{22}(\nu_r)$ induced by $l_1(\nu_r)$ and $l_2(\nu_r)$ such that $R_{ij}(\nu_r)$ contains $\mathcal{X}_{ij}(\nu_r)$ ($1 \leq i, j \leq 2$). We create four children nodes ν_{ij} ($1 \leq i, j \leq 2$) for ν_r and assign $\mathcal{X}_{ij}(\nu_r)$ and $R_{ij}(\nu_r)$ as their corresponding regions and pointsets respectively. We also store the pointset of each ν_{ij} in the sorted order based on their x-coordinates. We recursively continue this process until all leaf-nodes of $\mathcal{Q}$ has at most one point in their pointset. Note that for each internal node $\nu \in \mathcal{Q}$, $\cup_{1 \leq i,j \leq 2} R_{ij}(\nu)$ is the entire plane. See Figure 2.4.

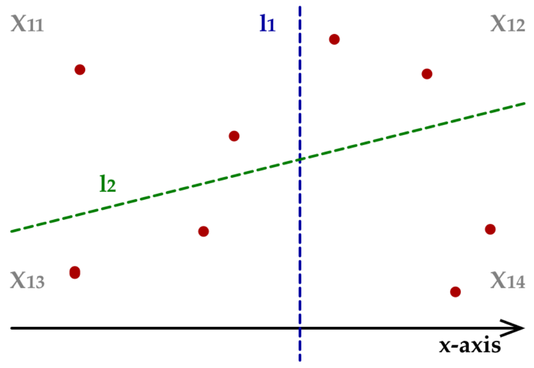


Figure 2.4: Construction of the QPT.

Proposition 2.5. *The QPT, the regions and the pointsets of its nodes can be computed in $O(m \log m)$ time.*

Proof. We show that the cost of computing the regions and pointsets of the children of the nodes in each level of the tree is $O(m)$. Let $\{\nu_1, \ldots, \nu_k\}$ be the set of nodes at a level of $\mathcal{Q}$. First note that for each $1 \leq i \neq j \leq k$, $\mathcal{X}(\nu_i) \cap \mathcal{X}(\nu_j) = \emptyset$. The first step is computing the vertical lines $l_1(\nu_i)$ $(1 \leq i \leq k)$. Because each vertical line can be computed in a constant time, the total time complexity of finding $l_1(\nu_i)$ lines is $O(k)$. On the other hand, the time complexity of computing the ham-sandwich cut line of $\mathcal{X}_1(\nu_i)$ and $\mathcal{X}_2(\nu_i)$ is $O(|\mathcal{X}(\nu_i)|)$ [54] and because $\sum_{i=1}^{k} |\mathcal{X}(\nu_i)| = O(m)$, the total cost of computing $\{l_2(\nu_i) : 1 \leq i \leq k\}$ is $O(m)$. Next, the intersection of $l_1(\nu_i)$ and $l_2(\nu_i)$, the regions and pointsets of the children of ν_i $(1 \leq i \leq k)$ can be computed in $O(|\mathcal{X}(\nu_i)|)$ time (by checking each point of it against $l_1(\nu_i)$ and $l_2(\nu_i)$). This implies that the total time complexity for obtaining the regions and pointsets of the children of all nodes $\{\nu_i : 1 \leq i \leq k\}$ is $O(m)$. Because we already have the pointset of each $\mathcal{X}(\nu_i)$ $(1 \leq i \leq k)$ in sorted order, we can traverse the points in $\mathcal{X}(\nu_i)$ according to the order and store each point in its corresponding child of ν_i to have the pointsets of the children of ν_i in order. This traversal takes $O(|\mathcal{X}(\nu_i)|)$ time and so, storing the pointsets of all children of the nodes in $\{\nu_i : 1 \leq i \leq k\}$ takes $O(m)$ time. Because the height of $\mathcal{Q}$ is $O(\log m)$, the total time complexity of building $\mathcal{Q}$, the pointsets and regions of all nodes in $\mathcal{Q}$ is $O(m \log m)$. $\square$

Observation 2.5. *For any internal node $\nu \in \mathcal{Q}$, and any half-plane H, one of the $R_{ij}(\nu)$ regions $(1 \leq i, j \leq 2)$ lies completely either inside or outside of $int(H)$.*

Given a half-plane H, we say an internal node $\nu \in \mathcal{Q}$ is *survived* from (resp. *eliminated* by) H if $R(\nu)$ completely lies outside (resp. inside) $int(H)$ (we assume that the degenerate case when ∂H passes the intersection point of $l_1(\nu)$ and $l_2(\nu)$ or ∂H is parallel to $l_1(\nu)$ or $l_2(\nu)$ is not happening). Similarly, we say a leaf-node ν is survived from (resp. eliminated by) H if $\mathcal{X}(\nu)$ completely lies outside (resp. inside) $int(H)$ (if $\mathcal{X}(\nu)$ is empty, we count it as survived). If ν is survived or eliminated by H, we say H *interacts* with ν otherwise, we say H does not interact with ν. Note that checking whether ν interacts with H can be done in a constant time. Because each single point of $\mathcal{X}$ can be either inside or outside of $int(H)$, for each root-leaf path in $\mathcal{T}$, one of the nodes on the path should interact with H. In Figure 2.5, $R(\nu_{13})$ is eliminated by H.

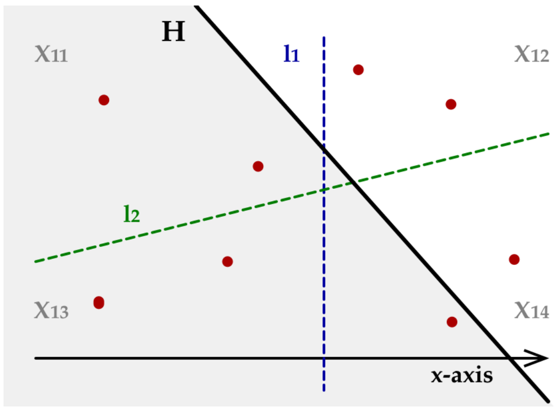


Figure 2.5: For any half-plane H, one of the R_{ij} regions should interact with H.

Indeed, we are interested to find the lowest level node (the closest node to the root) in each root-leaf path of $\mathcal{Q}$ that interacts with H. Suppose that $x \in \mathcal{X}$ and ν_x be the leaf-node of $\mathcal{Q}$ containing x. Also, let π_x is the root-leaf path from ν_r (the root of $\mathcal{Q}$) to ν_x. If an internal node $\nu \in \pi_x$ is eliminated (resp. survived) by H then x is also eliminated (resp. survived) by H. So, we start from the children of ν_r and for each child, if it interacts with H, we mark it accordingly. Otherwise, we recursively look at its children and continue this process until in each root-leaf path, we find a node that interacts with H. Procedure EXPLORE(ν_r,H) in Algorithm 1 demonstrates this process and computes the lowest level node on each root-leaf path in $\mathcal{Q}$ that interacts with H.

Algorithm 1 EXPLORE(ν,H)

```
1: for $i, j \in \{1, 2\}$ do
2:     if $\nu_{ij}$ is eliminated by $H$ then
3:         Mark $\nu_{ij}$ as eliminated.
4:     else if $\nu_{ij}$ is survived from $H$ then
5:         Do nothing.
6:     else
7:         EXPLORE($\nu_{ij}$,$H$) if $\nu_{ij}$ is an internal node of $\mathcal{Q}$.
8:     end if
9: end for
```

Proposition 2.6. *Given a half-plane H, we can identify the lowest level nodes of each root-leaf path that is eliminated by H in $O(m^{\log_4 3})$ time.*

Proof. Because for each internal node $\nu \in \mathcal{Q}$, we need to check three children, the number of nodes for which we need to check at level i is 3^i and because we have $O(\log_4 m)$ levels, the total complexity of running EXPLORE(ν_r,H) is $4 \times 3^{\log_4 m} = O(m^{\log_4 3})$. □

Solving the DBKP: We first build the QPT in $O(m \log m)$ time. Next, we consider each reflex vertex $r \in \mathcal{R}$ and its two corresponding half-planes $H_1(r)$ and $H_2(r)$. We know a point $x \in \mathcal{X}$ is eliminated by r if and only if $x \in H_1(r) \cap H_2(r) = DW(r)$. Therefore, we first consider the QPT against $H_1(r)$ and mark each node with pointset in $int(H_1(r))$, *semi-eliminated* by r (see Figure 2.6). Next, we consider each subtree of $\mathcal{Q}$ rooted at a semi-eliminated node (with respect to r) against $H_2(r)$. If any node of these subtrees lies in $int(H_2(r))$, we mark it *eliminated* (independent of r).

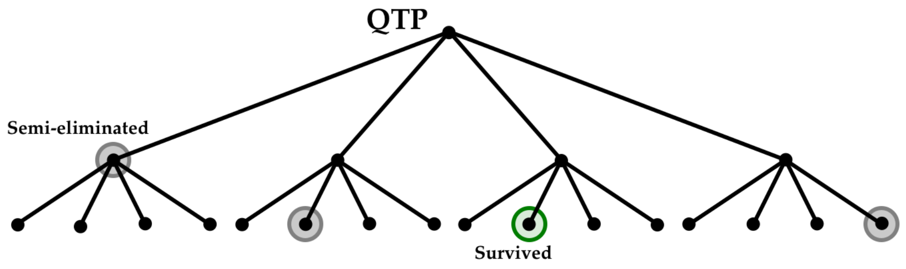


Figure 2.6: Marking the nodes of the QPT.

Corollary 2.2. *Given a reflex vertex $r \in \mathcal{R}$, we can identify the lowest level nodes of each root-leaf path that is eliminated by r in $O(m^{\log_4 3})$ time.*

The reason is that at the i^{th}-level of Q, we have 3^{i-1} subtrees to explore each with pointset of size $m/4^i$ and cost $(m/4^i)^{\log_4 3}$ (based on Proposition 2.6). Now, summing up over the $O(\log_4 m)$ levels of Q gives us the desired $O(m^{\log_4 3})$ time complexity.

Because we have $O(n)$ reflex vertices, we consider Q against each reflex vertex in $\mathcal{R}$ and mark its nodes accordingly. Because of Corollary 2.2, this step takes $O(nm^{\log_4 3})$ time. Finally, for each node $x \in \mathcal{X}$ with corresponding node $\nu_x \in Q$, we examine the root-leaf path from ν_r to ν_x in Q. If any node is marked eliminated, we report x as an eliminated point otherwise, we report it as a beacon kernel point. We have m points in $\mathcal{X}$ and for each, we need to check a path with length $O(\log m)$. Therefore, the reporting step takes $O(m \log m)$ time and we would have the following theorem:

Theorem 2.1. *The discrete beacon kernel problem can be solved in $O(m \log m + nm^{\log_4 3})$ time.*

2.4 The Semi-Discrete Beacon Kernel Problem

Suppose that $\mathcal{S}$ is the given set of line segments such that each $S \in \mathcal{S}$ completely lies in P. In the SDBKP, the objective is determining the beacon kernel points on each segment in $\mathcal{S}$. In order to solve the SDBKP, we first introduce a data structure for P called *split decomposition tree (SDT)* which enables us to eliminate the points in P by half-planes instead of dead wedges. The reason is that when an interior point of a segment S is eliminated by a dead wedge, that dead wedge might not eliminate an endpoint of S. We show that the SDT can be built in $O(n \log n)$ time which leads to a sub-quadratic time algorithm for the SDBKP.

2.4.1 The Split Decomposition Tree of P

We start by computing a triangulation Δ of P and its dual graph $\mathbb{T}_\Delta$. The triangulation of P can be done in linear time using Chazelle's polygon triangulation algorithm [18]. $\mathbb{T}_\Delta$ is a graph for which there is a node corresponding to each triangle in Δ and two nodes in $\mathbb{T}_\Delta$ are connected by an edge if their corresponding triangles share an edge. Note that because P is simple, $\mathbb{T}_\Delta$ is a tree (this is because we can always embed $\mathbb{T}_\Delta$ in P such that each node of $\mathbb{T}_\Delta$ lies in its corresponding triangle in Δ). We can see that each subtree $T \subseteq \mathbb{T}_\Delta$ corresponds to a connected region in P which is obtained by the union of the triangles corresponding to the nodes in T. We denote this region by $P(T)$ (see Figure 2.7).

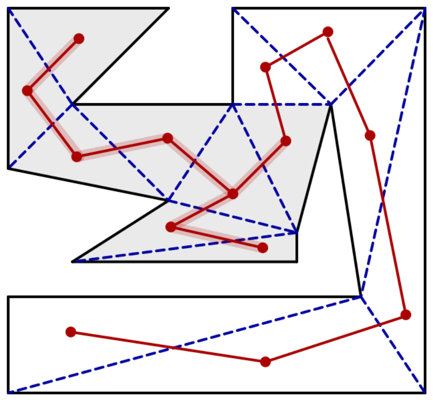


Figure 2.7: An example of $\mathbb{T}_\Delta$, a subtree of it and its corresponding region.

The *centroid* of $\mathbb{T}_\Delta$ is defined as a node for which by removing it (and its incident edges) from $\mathbb{T}_\Delta$, the maximum size of each connected component is minimum. Therefore, the size of each connected component of the remaining graph is at most $|\mathbb{T}_\Delta|/2$. Note that because the degree of each node of $\mathbb{T}_\Delta$ is at most three, by removing a node, we would have at most three connected components. In order to avoid confusion in our algorithm, we always assume that the trees are rooted and if there are multiple choices for selecting a centroid, we choose the one closest to the root. Based on this assumption, the centroid of $\mathbb{T}_\Delta$ (and each of its subtrees) is unique and can be computed in linear time [51]. Suppose that $|\mathbb{T}_\Delta| > 2$ ($|\mathbb{T}_\Delta|$ is the number of nodes in $\mathbb{T}_\Delta$) and c is the centroid of $\mathbb{T}_\Delta$. Also, suppose that c has degree three (resp. two) and $\{T_1, T_2, T_3\}$ (resp. $\{T_1, T_2\}$) are the subtrees of $\mathbb{T}_\Delta$ emanating from removing c from $\mathbb{T}_\Delta$ such that T_1 has the greatest size. We say two subtrees $\{S_1, S_2\}$ is obtained from *splitting* $\mathbb{T}_\Delta$ over c if $S_1 = T_1$ and S_2 is the joint of T_2 and T_3 by c (resp. $S_2 = T_2$). Based on the definition of centroid, we have $|S_1| \geq |\mathbb{T}_\Delta|/3$ and $|S_2| \leq 2|\mathbb{T}_\Delta|/3$.

Next, we build a data structure called *split decomposition tree* for $\mathbb{T}_\Delta$ denoted by $SDT(\mathbb{T}_\Delta)$ which is a binary tree such that each of its nodes corresponds to a subtree of $\mathbb{T}_\Delta$. We build $SDT(\mathbb{T}_\Delta)$ recursively by splitting the subtrees of $\mathbb{T}_\Delta$ over their centroids starting from $\mathbb{T}_\Delta$. In order to build $SDT(\mathbb{T}_\Delta)$, we first create a root node ω_r and assign $\mathbb{T}_\Delta$ to it. We also store the centroid of $\mathbb{T}_\Delta$ in ω_r. Next, we use the recursive procedure $BSDT(\omega_r)$ in Algorithm 2 to complete the construction of $SDT(\mathbb{T}_\Delta)$.

Algorithm 2 BSDT(ω) // Building SDT

```
1: Let $T_\omega$ be the subtree of $\mathbb{T}_\Delta$ assigned to $\omega$.
2: if $|T_\omega| > 2$ then
3:     Compute the centroid $c$ of $T_\omega$ and store it in $\omega$.
4:     Let $S_1$ and $S_2$ be the subtrees obtained by splitting $T_\omega$ over $c$.
5:     Create two children $\omega_1$ and $\omega_2$ for $\omega$.
6:     Assign $S_1$ and $S_2$ to $\omega_1$ and $\omega_2$ respectively.
7:     BSDT($\omega_1$)
8:     BSDT($\omega_2$)
9: end if
```

Theorem 2.2. *$SDT(\mathbb{T}_\Delta)$ can be constructed in $O(n \log n)$ time.*

This is because the height of $SDT(\mathbb{T}_\Delta)$ is $O(\log n)$ and the subtrees in each level of $SDT(\mathbb{T}_\Delta)$ are disjoint. Consider an internal node $\omega \in SDT(\mathbb{T}_\Delta)$ and let S and c be the subtree and its centroid stored in ω. Also, let S_1 and S_2 be the subtrees obtained by splitting S over c and e be the connecting edge of S_1 and S_2. Note that c corresponds to a triangle in Δ and e corresponds to a chord C separating $P(S_1)$ and $P(S_2)$. We call C the *chord corresponding to* ω. Let us define $\mathcal{C}$ as the set of all chords corresponding to the internal nodes in $SDT(\mathbb{T}_\Delta)$. Because $SDT(\mathbb{T}_\Delta)$ has linear number of nodes, the number of chords in $\mathcal{C}$ is also linear. Let $\Gamma = \{\omega_1, \ldots, \omega_k\}$ be the set of nodes in a level of $SDT(\mathbb{T}_\Delta)$ (they all have the same distance from the root). If we denote the subtree of $\mathbb{T}_\Delta$ stored in ω by T_ω, we would have the following observation:

Observation 2.6. *For any two nodes $\omega_i, \omega_j \in \Gamma$, $int(P(T_{\omega_1})) \cap int(P(T_{\omega_2})) = \emptyset$. Also,*

$$P = \cup_{\omega \in \Gamma} P(T_\omega) \tag{2.2}$$

If ω is a leaf-node of $\mathbb{T}_\Delta$, we call $P(T_\omega)$, the *leaf-node region* of P corresponding to ω. If a point p lies in $P(T_\omega)$ for some $\omega \in SDT(\mathbb{T}_\Delta)$, we say ω *contains* p and p *belongs* to ω. According to the above observation, if v is a vertex of ∂P and $L(v)$ is the set of leaf-nodes of $SDT(\mathbb{T}_\Delta)$ that contain p, we have

$$\sum_{v \text{ is a vertex of } \partial P} L(v) = O(n) \tag{2.3}$$

This is because we have $O(n)$ leaf-node regions in P and each leaf-node region can contain a constant number of vertices.

Corollary 2.3. *For each point $p \in P$ and any vertex r (except possibly a constant number of vertices), a chord in $\mathcal{C}$ separates p from r.*

The reason is that when a point $p \in P$ belongs to a leaf-node ω of $SDT(\mathbb{T}_\Delta)$, the chords induced by the nodes in the root-leaf path in $SDT(\mathbb{T}_\Delta)$ to ω separate (based on our definition, if p lies on a chord, the chord separates it from all points in the polygon) p from any vertex in ∂P except possibly a constant number of vertices (the vertices lie in $P(T_\omega)$).

Let us consider $\mathcal{X}$ as the set of all endpoints of the segments in $\mathcal{S}$. Thus, $\mathcal{X}$ has $2m$ points. Again, we assume that points are in general position which means no tree points (segment endpoints or polygon vertices) are collinear. Having $SDT(\mathbb{T}_\Delta)$, for each node $\omega \in SDT(\mathbb{T}_\Delta)$ with assigned subtree T_ω, we denote the points of $\mathcal{X}$ lying in $P(T_\omega)$ by $\mathcal{X}(\omega)$ and store them in ω. In order to identify the set of points lying in each triangle, we use Kirkpatrick's data structure [45] on Δ which by spending $O(n \log n)$ time for preprocessing, it enables us to determine the triangle(s) containing a query point in $O(\log n + q)$ time where q is the number of triangles containing the point. Because $\mathcal{X}$ has $O(m)$ points, this step takes in $O(n + m \log n)$ time (the additional linear factor is because a vertex may lie on multiple triangles). Note that if ω has two children ω_1 and ω_2, we would have

$$\mathcal{X}(\omega) = \mathcal{X}(\omega_1) \cup \mathcal{X}(\omega_2) \tag{2.4}$$

Also, the total number of points stored in the nodes of each level of $SDT(\mathbb{T}_\Delta)$ is $O(n+m)$. Because the height of $SDT(\mathbb{T}_\Delta)$ is $O(\log n)$, in $O((n+m)\log n)$ time we can obtain and store $\mathcal{X}(\omega)$ for each node $\omega \in SDT(\mathbb{T}_\Delta)$. In order to solve the SDBKP, we need to address the following two sub-problems:

1. Given a chord C, provide an algorithm to determine which points on $\mathcal{X} \cap P_2(C)$ (resp. $\mathcal{X} \cap P_1(C)$) survive from the reflex vertices in $P_1(C)$ (resp. $P_2(C)$). We call this problem the *chord elimination problem (for short CEP).*
2. How we can apply the algorithm for solving CEP to the chords in $\mathcal{C}$ in order to solve the SDBKP in sub-quadratic time.

In the next sections, we see how we can answer to these sub-problems.

2.4.2 The Chord Elimination Problem (CEP)

Let $C = seg(v_1v_2)$ be the given chord connecting two vertices in P. For the sake of simplicity, we assume that C is horizontal having $P_1(C)$ in its below and v_1 is on the left side of v_2. C divides P into two sub-polygons $P_1(C)$ and $P_2(C)$. Also, let $\mathcal{Y}$ be the given set of M points in $P_2(C)$ and $\mathcal{R}_1$ be the set of reflex vertices in $P_1(C)$. We first compute the intersection points of $sup(C)$ with $P_2(C)$ and add all the chords induced by $sup(C)$ on $P_2(C)$. Next, we update the triangulation of $P_2(C)$ such that these chords are edges of the triangulation. This modification can be done in $O(n')$ time [62] where n' is the number of vertices in $P_2(C)$. Let $T_2(C)$ be the subtree of $\mathbb{T}_\Delta$ covering $P_2(C)$ rooted at node τ corresponding to the triangle incident to C. Considering the embedding of $T_2(C)$ in $P_2(C)$, we divide $T_2(C)$ into three parts: $T_2^{up}(C)$ is the nodes of $T_2(C)$ reachable from τ without crossing $sup(C)$. Similarly, $T_2^{left}(C)$ (resp. $T_2^{right}(C)$) is the set of nodes $\omega \in T_2(C)$ such that the path π_ω from τ to ω goes (by the first time) below $sup(C)$ by crossing $half\text{-}line(v_2, v_1)$ (resp. $half\text{-}line(v_1, v_2)$) where $half\text{-}line(v_2, v_1)$ is the half-line from v_2 passing v_1. Let us denote the portion of $P_2(C)$ corresponding to $T_2^{up}(C)$, $T_2^{left}(C)$ and $T_2^{right}(C)$ by $P_2^{up}(C)$, $P_2^{left}(C)$ and $P_2^{right}(C)$ respectively. So, in $O(n' + M)$ time, we can identify the points of $\mathcal{Y}$ in each of these regions.

Next, suppose that we are given a reflex vertex $r \in P_1(C)$ with incident edges e_1 and e_2. In the following, we define the *eliminating half-plane* generated by r for each of $P_2^{up}(C)$, $P_2^{left}(C)$ and $P_2^{right}(C)$ such that no beacon point can lie in the half-plane. Based on Proposition 2.4, if C completely lies in $P(e_1)$ (resp. P(e_2)), $H_2(r)$ (resp. $H_1(r)$) is the eliminating half-plane of r on $P_2(C)$. Similarly, if C completely lies in the sub-polygon in front of r, r does not need to generate any eliminating half-plane on $P_2(C)$. Otherwise, at least one of c_{e_1} or c_{e_2} intersects C. If v_1 (similarly v_2) lies in $P(e_1)$ (resp. $P(e_2)$), $H_2(r)$ (resp. $H_1(r)$) would be the eliminating half-plane on $P_2^{left}(C)$. Also, if the extension of e_2 (resp. e_1) intersects $P_2^{up}(C)$, we assign its corresponding half-plane as the eliminating half-plane of r on $P_2^{up}(C)$. Note that if the both extensions of e_1 and e_2 intersect C, then C can not intersect both C_{e_1} and C_{e_2}. So in each situation, we assign at most one eliminating half-plane to each of $P_2^{up}(C)$, $P_2^{left}(C)$ and $P_2^{right}(C)$ which can be obtained in $O(\log n')$ time (see Figure 2.8 as an example).

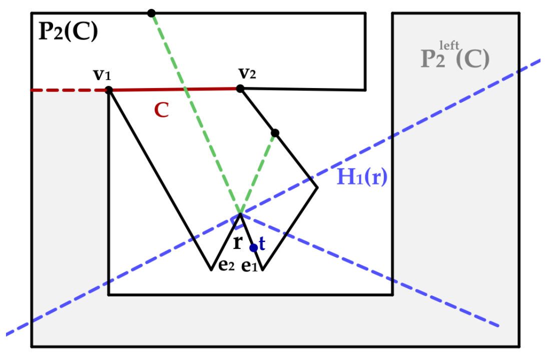


Figure 2.8: $H_1(r)$ is the eliminating half-plane for $P_2^{left}(C)$ because any beacon in this half-plane can not attract t.

We say a point $y \in \mathcal{Y}$ is eliminated by a reflex vertex $r \in \mathcal{R}_1$ if it lies in the eliminating half-plane generated by r in its region. In the CEP, the objective is determining the point of $\mathcal{Y}$ survived from any reflex vertex r in $\mathcal{R}_1$. In order to solve this problem, we first build a QPT on the points of $P_2^{up}(C)$, $P_2^{left}(C)$ and $P_2^{right}(C)$ and consider them against the eliminating half-planes generated by the reflex vertices in $\mathcal{R}_1$. We also create a list $LS(\nu)$ of half-planes for each node ν in the QPTs and when a half-plane eliminates ν, we store it in $LS(\nu)$. Assuming n'' is the number of vertice in $\mathcal{R}_1$, we have:

Proposition 2.7. *The CEP can be solved in $O((n' + M)\log(n' + M) + n''(n' + M)^{log_4 3})$ time.*

The proof is similar to the proof of Theorem 2.1.

2.4.3 Solving the Semi-Discrete Beacon Kernel Problem

Let ω be an internal node in $SDT(\mathbb{T}_\Delta)$ with two children ω_1 and ω_2 and induced chord c_ω. Let us denote the procedure of determining the points in $\mathcal{X}(\omega_1)$ (resp. $\mathcal{X}(\omega_2)$) surviving from the reflex vertices in $P(T_{\omega_2})$ (resp. $P(T_{\omega_1})$) by CEP($\mathcal{X}(\omega_1)$, $P(T_{\omega_2})$) (resp. CEP($\mathcal{X}(\omega_2)$, $P(T_{\omega_1})$)). Procedure REFINE(ω_r) specifies the points in $\mathcal{X}$ which are eliminated by the reflex vertices in $\mathcal{R}$.

Algorithm 3 REFINE(ω)

```
 1: if $\omega$ is a leaf-node of $SDT(\mathbb{T}_\Delta)$ then
 2:     for each point $x \in \mathcal{X}(\omega)$ do
 3:         for each reflex vertex $v$ in $P(T_\omega)$ do
 4:             Eliminate the portion of the segment with endpoint $x$ lying in $DW(v)$.
 5:         end for
 6:     end for
 7: else
 8:     Let $\omega_1$ and $\omega_2$ be the children of $\omega$.
 9:     CEP($\mathcal{X}(\omega_1)$,$P(T_{\omega_2})$).
10:     CEP($\mathcal{X}(\omega_2)$,$P(T_{\omega_1})$).
11:     REFINE($\omega_1$).
12:     REFINE($\omega_2$).
13: end if
```

After running REFINE(ω_r), the points of $\mathcal{X}$ that are not marked eliminated are the beacon kernel points in $\mathcal{X}$. This is because if $x \in \mathcal{X}$ and $v \in \mathcal{R}$, if they are in the same region corresponding to a leaf-node of $SDT(\mathbb{T}_\Delta)$, we directly check whether v eliminates x. Otherwise, consider the lowest level node $\omega \in SDT(\mathbb{T}_\Delta)$ with children ω_1 and ω_2 for which $x \in P(T_{\omega_1})$ and $v \in P(T_{\omega_2})$. Now, CEP($\mathcal{X}(\omega_1)$,$P(T_{\omega_2})$) eliminates x if it is eliminated by v.

Complexity analysis: In order to analyse the time complexity of running REFINE(ω_r), first note that because we assumed that the points are in general position, for each leaf-node $\omega \in SDT(\mathbb{T}_\Delta)$, $P(T_\omega)$ contains only a constant number of vertices in ∂P (because $P(T_\omega)$ is a triangle or two triangles sharing an edge). We have $O(m)$ points in $\mathcal{X}$ and the number of edges in the entire triangulation Δ is linear which implies that the first part of the algorithm (lines 2 to 8) takes $O(n+m)$ time.

Next, note that $SDT(\mathbb{T}_\Delta)$ has $O(\log n)$ levels and for two nodes ω' and ω'' in a same level of $SDT(\mathbb{T}_\Delta)$, $P(T_{\omega'})$ and $P(T_{\omega'})$ have disjoint interiors. This implies that for each level i of $SDT(\mathbb{T}_\Delta)$ with the set of nodes $\mathcal{L}_i$, we have:

$$\sum_{\nu \in \mathcal{L}_i} |\mathcal{X}(\nu)| = O(m+n) \tag{2.5}$$

Similarly, $\sum_{\omega \in \mathcal{L}_i} |P(T_\omega)| = O(n)$ where $|P(T_\omega)|$ is the number of vertices in $\partial P(T_\omega)$. Therefore, the time complexity of running the two CEP procedures (lines 11 and 12 of the algorithm) for all nodes $\omega \in \mathcal{L}_i$ is:

$$O\big((n+m)\log(n+m) + n(n+m)^{\log_4 3}\big) \tag{2.6}$$

Because $SDT(\mathbb{T}_\Delta)$ has $O(\log n)$ levels, the total time complexity of running REFINE(ω_r) is

$$O\Big(\big((n+m)\log(n+m) + n(n+m)^{\log_4 3}\big)\log n\Big) \tag{2.7}$$

where ω_r is the root of $SDT(\mathbb{T}_\Delta)$.

2.4.4 Computing the Beacon Kernel Points on the Segments

First, note that because the beacon kernel of P is convex with respect to P [11], for each segment $S \in \mathcal{S}$, $S \cap Ker(P)$ is a connected region. We call this region, the *kernel segment* of S and denote it by $KS(S)$. In the SDBKP, the objective is determining the kernel segments of the segments in $\mathcal{S}$.

Observation 2.7. *If $x \in int(S)$ gets eliminated by a reflex vertex r, then r eliminates a or b (or both).*

This is because $S \subseteq P$ and the boundary of the eliminating half-plane of r on x can intersect S in at most one point. Suppose that $\mathcal{H}_a$ (resp. $\mathcal{H}_b$) is the set of eliminating half-planes of the reflex vertices in $\mathcal{R}$ on a (resp. b) that eliminate a (resp. b) after running REFINE(ω_r). We denote the union of the half-planes in $\mathcal{H}_a$ and $\mathcal{H}_b$ by $\mathcal{U}_a$ and $\mathcal{U}_b$ respectively. Suppose a' (resp. b') is the intersection point of $half\text{-}line(a,b)$ (resp. $half\text{-}line(b,a)$) with $\partial\mathcal{U}_a$ (resp. $\partial\mathcal{U}_b$). If such an intersection does not exist, we consider an imaginary intersection at infinity. According to Observation 2.7 we have:

$$KS(S) = S \setminus \big(seg(aa') \cup seg(bb')\big) \tag{2.8}$$

In order to obtain a' and b', first let us denote the union of the half-planes in $LS(\nu)$ by $\mathcal{W}(\nu)$ where ν is a node of a QPT. Note that $\mathcal{W}(\nu)$ may not be convex.

Proposition 2.8. *If a point p lies in the intersection of the half-planes in $LS(\nu)$, then any half-line from p can intersect $\mathcal{W}(\nu)$ in at most one point.*

Proof: Suppose not and there is a half-line ℓ from p that intersects $\partial\mathcal{W}(\nu)$ in at least two points. Because ℓ intersects the boundary of each half-plane on the union in at most one point, there is two different half-planes H_1 and H_2 such that the intersection points of ℓ with ∂H_1 and ∂H_2 lie on $\partial\mathcal{W}(\nu)$. Because $p \in H_1 \cap H_2$, ℓ should first intersect $\partial(H_1 \cap H_2)$. But then it can not intersect ∂H_1 and ∂H_2 in two different points. $\square$

Based on the above proposition, we can have an angular order (clockwise or counter-clockwise) on the vertices in $\partial\mathcal{W}(\nu)$ from any point in the intersection of the half-planes in $LS(\nu)$. Our next step is computing $\mathcal{W}(\nu)$ for all nodes ν in the QPTs. Let $\mathcal{Q}$ be a QPT covering a subset of points of size n_1 for which we are going to consider it against a set of n_2 half-planes $\mathcal{H}$. For a node $\nu \in \mathcal{Q}$, the half-lines in $LS(\nu)$ should have non-empty intersection because they all covers the pointset of ν. Also, at the i^{th}-level of Q, each half-plane $H \in \mathcal{H}$ can appear at most 3^i times in the lists. For each node ν, we use the divide-and-conquer approach (we can use this approach because of Proposition 2.8) to compute $\mathcal{W}(\nu)$ and an angular order on its vertices (from a points in the intersection of the half-planes in $LS(\nu)$). Now, the total time complexity of computing $\mathcal{W}(\nu)$ for all nodes ν in the i^{th}-level of $\mathcal{Q}$ is $O(n_2 3^i \log(n_2 3^i))$. Summing over all $i \in \{1, \ldots, \log_4 n_1\}$, the total time complexity is:

$$O\big(n_2(n_1^{\log_4 3} \log n_2 + n_1^{\log_4 3} \log n_1)\big) \tag{2.9}$$

Here, $SDT(\mathbb{T}_\Delta)$ is a balanced binary tree and if $\{\omega_1, \ldots, \omega_k\}$ is the set of nodes at a level of $SDT(\mathbb{T}_\Delta)$, we have:

$$\sum_{j=1}^{k} |\mathcal{X}(\omega_j)| = O(n+m) \;\text{ and }\; \sum_{j=1}^{k} |P(T_{\omega_j})| = O(n) \tag{2.10}$$

Therefore, the total time complexity of computing $\mathcal{W}(\nu)$ and the angular order of its vertices for all nodes ν in all QPTs is $O((n{+}m)^{1+\log_4 3} \log^2(n{+}m))$. Here, we have enough information for computing the kernel segment of each segment $S = seg(ab) \in \mathcal{S}$. In order to compute the kernel segment, we consider $half\text{-}line(a,b)$ (similarly $half\text{-}line(b,a)$) and each root-leaf path in $SDT(\mathbb{T}_\Delta)$ that ends at a leaf whose region contains a. Let λ be one of these leaf nodes in $SDT(\mathbb{T}_\Delta)$ and μ_λ its corresponding root-leaf path. For each node $\omega \in \mu_\lambda$, we have $a \in \mathcal{X}(\omega)$. Let ν_a be the leaf-node containing a in its QTP for $\mathcal{X}(\omega)$ and π_a its corresponding root-leaf path. For each node $\nu \in \pi_a$, because $a \in \cap_{H \in LS(\nu)} H$, we can do binary search to find the intersection point of $half\text{-}line(a,b)$ and $\partial\mathcal{W}(\nu)$ and keep the furthest such intersection on $half\text{-}line(a,b)$. Let a_f (similarly b_f) be the farthest (from a) intersection point of $half\text{-}line(a,b)$ with $\partial\mathcal{W}(\nu)$ where ν is a node of a QTP that contains a in its pointset. Based on Equation 2.8, we have:

$$KS(S) = S \setminus seg(aa_f) \cup seg(bb_f)$$

Procedure COMPUTE-KS(S) in Algorithm 4 demonstrates this process.

Algorithm 4 COMPUTE-KS(S)

```
1: Let $S = seg(ab)$.
2: Let $L_a$ (resp. $L_b$) be the set of leaf-nodes in $SDT(\mathbb{T}_\Delta)$ whose regions contains $a$ (resp. $b$).
3: for each $\lambda \in L_a$ do (and similarly for $L_b$)
4:     Let $\mu_\lambda$ be the root-leaf path to $\lambda$ in $SDT(\mathcal{T}_\Delta)$.
5:     for each node $\omega \in \mu_\lambda$ do
6:         Let $\mathcal{Q}_\omega$ be the QPT for $\mathcal{X}(\omega)$ containing $a$.
7:         Let $\nu_a$ be the leaf-node in $\mathcal{Q}_\omega$ corresponding to the point $a$.
8:         Let $\pi_a$ be the root-leaf path to $\nu_a$ in $\mathcal{Q}_\omega$.
9:         for each node $\nu \in \pi_a$ do
10:            Find the intersection point of $half\text{-}line(a, b)$ with $\partial\mathcal{W}(\nu)$.
11:            Keep track of the farthest intersection point $a_f$ on $half\text{-}line(a, b)$ from $a$.
12:        end for
13:    end for
14: end for
15: Set $KS(S) = S \setminus \{seg(aa_f) \cup seg(bb_f)\}$.
```

Note that in Algorithm 4, $KS(S)$ might be the empty set. In order to analyse COMPUTE-KS(S), first note that

$$\sum_{seg(ab)\in\mathcal{S}} |L_a| = O(n + m)$$

This is because of our general assumption that no three points are collinear and the fact that Δ has linear number of triangles and each triangles has three vertices. So, if a is an endpoint of a segment that is not a vertex of ∂P, it can lie in at most two leaf-nodes of $SDT(\mathbb{T}_\Delta)$. Now, for each leaf-node $\lambda \in L_a$, we have $O(\log n)$ nodes on μ_λ and for each node of μ_λ, we have $O(\log m)$ nodes on π_a. Finally, for each node ν of π_a, we need to do a binary search on the vertices of the $\partial\mathcal{W}(\nu)$. This step is also costs $O(\log n)$ time. So, the total time complexity of COMPUTE-KS(S) is $O(\log^3(n + m))$. By running this procedure on each segment in $\mathcal{S}$, we can find the kernel segments of the segments in $\mathcal{S}$ in $O(m \log^3(n + m))$ time. Therefore, we can have the following theorem:

Theorem 2.3. *The semi-discrete beacon kernel problem can be solved in*

$$O((n + m)^{1+\log_4 3} \log^2(n + m))$$

time.

We can immediately see that if we consider $\mathcal{S}$ as the edges of ∂P, by solving the SDBKP on $\mathcal{S}$, we can find the beacon kernel points of P on its boundary.

Corollary 2.4. *In $O(n^{1+\log_4 3} \log^2 n)$ time, we can compute the beacon kernel points of P on its boundary.*

Chapter 3

Transmitting Particles in Polygonal Domains by Repulsion

In this chapter[1], we introduce the problem of transmitting particles to a target point by the effect of a *repulsion actuator (repulsor)*. In this problem, we are given a polygonal domain (a polygon with polygonal holes inside it) P and a target point t inside it. We also assume that initially there is a particle at each point of P. The question is which particles can get to the target point t by activating only one repulsor in P. We present an efficient polynomial time algorithm to solve this problem.

[1]A preliminary version of this chapter appeared in the proceedings of the *International Conference on Combinatorial Optimization and Applications, COCOA 2018. [55]*

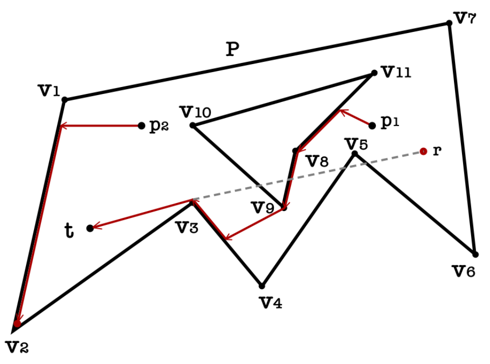


Figure 3.1: The behavior of two particles p_1 and p_2 when we activate a repulsor at r in a polygonal domain P. In this example p_1 can get to the target point t while p_2 resides at the vertex v_2.

3.1 Background and Previous Works

In contrast to the problems raising from interaction of objects by attraction, problems raising from interaction of objects under the repulsion force have been rarely investigated. In [16], Bose and Shermer studied the effect of putting a repulsor in a convex polygon full of point particles. They gave an $O(n^2)$ time algorithm to compute all locations for putting a repulsor such that all particles gather into a single point. They also gave a linear time algorithm to determine whether such a location exists for a given convex polygon. As another work, in 2020, van Goethem et al. [64] introduced the *repulsion region* of a point s in polygonal domains. The repulsion region of s is defined as the set of points p in the region for which there exists a repulsor r such that the particle placed at s will eventually reach p under the influence of a repulsor at r. They showed that if P is a simple polygon with n vertices, the repulsion region of a point inside it can be computed in $O(n^2 \log n)$ time. In this chapter, we consider another problem regarding the behaviour of point particles under the repulsion force called the *particle transmitting problem.* In this problem, we have a polygonal domain P and a target point t in its interior. When we activate a repulsor at a point r inside P, all particles move away from it until either stop at a corner of P or they hit the target point t. Specifically, each particle traverses a path such that at each time it goes in a direction that takes itself farthest from r while it remains inside the polygon. Figure 3.1 shows the behaviour of two particles when we activate a repulsor at the point r in P.

A natural question here asks which particles can get to t by activating only one repulsor in P. We say a point $x \in P$ is *valid* if there exists a point $r_x \in P$ such that by putting a repulsor at r_x, the particle at x gets to the target point t. According to this definition the problem turns to computing all valid points in P with respect to a given target point t. In Section 3.2, we give some basic definitions and essential properties. In Section 3.3, we present a polynomial time algorithm for the problem. Finally, in Section 3.4 we discuss the time complexity of the algorithm.

3.2 Preliminaries and Definitions

Let x, y and r be three points inside the given polygonal domain P. We say r can send x to y if by activating a repulsor at r the particle at x gets to y. Note that the path that the particle traverses to take itself farthest from r at each time, may not be unique: when the particle hits ∂P (boundary of P), it might get farther from r by moving clockwise or counter-clockwise around the component (a connected part of ∂P) that it has hit. So, r can send x to y if there exist such a path that ends up at y. Henceforth, instead of saying the particle at x traverses a path we simply say x traverses a path. Also, we say r is a *repulsion point* for x if r sends x to the target point t. We denote the set of all repulsion points for x by $R(x)$. By this notation, the set $\mathcal{A}$ of all valid points of P can be written as follows:

$$\mathcal{A} = \{x \in P \mid R(x) \neq \emptyset\} \tag{3.1}$$

Let $V = \{v_1, \ldots, v_n\}$ be the set of vertices in ∂P. In order to compute $\mathcal{A}$, we first compute the set of all points in P that can be sent to t by a direct path (a line segment) and denote it by A'. For example, in Figure 3.1, the vertex v_3 can get to t by a direct path. Then, we compute subsets $A_1, \ldots, A_n$ of $\mathcal{A}$ in which, A_i is the set of all points $x \in P$ for which there exist a point $r_x \in P$ such that r_x can send x to t by a path having $v_i t$ as its last segment. For example, in Figure 3.1, p_1 belongs A_3 because v_3 is the last bend point of the path that p_1 traverses to move away from the repulsor in r before getting to t. It is clear that:

$$\mathcal{A} = A' \cup A_1 \cup \cdots \cup A_n \tag{3.2}$$

An immediate observation is that if v_i is not visible from t then $A_i = \emptyset$. For a boundary points x and y we say a path p from x to y has j *jumps* if $p \setminus \partial P$ has j components.

3.3 The Algorithm

Let $L = \{l_1, \ldots, l_n\}$ where l_i ($1 \leq i \leq n$) is the half-line that starts from t and passes through v_i. We assume that the vertices are in general position and so, there is no line that passes t and two other vertices (we can have this condition by slightly perturbing the vertices). In order to compute A', we first partition the plane into a set of cones having t as their common vertex that obtained by radially adjacent half-lines of L. Figure 3.2 shows an example of such cones:

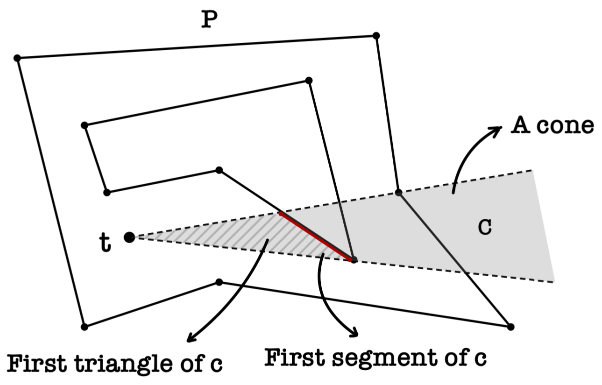


Figure 3.2: Partitioning the plane by the set of cones according to L

Let $\mathcal{C}$ be the set of all these cones. Each $C \in \mathcal{C}$ consists of a tip at the point t, two half-lines as its boundaries and a set of line segments each of which has an endpoint on each of the boundary half-lines of C. Within a cone, these segments are internally disjoint so we can have an order on them according to their closeness to t. Similarly, they partition the cone into a sequence of regions starting with a triangle having t as its vertex and the first segment as its base. We call this triangle, *the first triangle* of the cone. It can be easily seen that A' is the union of all first triangles of the cones in $\mathcal{C}$ that have more than one segment. So, in order to compute A', we only need to compute the subdivision of the plane induced by ∂P and the half-lines in L and consider the cones with more than one segment. This step can be done using the plane-sweep algorithm in $O(n^2)$ time [24, 63].

It remains to compute those valid points that follow a path that bends in order to get t. Let $x \in A_i$ and so there must be a point $r_x \in R(x)$ such that v_i is the last bend point of a path that r_x sends x to t along. Note that t lies on the interior of P and hence, r_x should lie on l_i otherwise, x can never reach t after leaving v_i. Also, note that we can compute each A_i individually and then consider their union to specify $\mathcal{A} \setminus A'$. Henceforth, we fix the index i and assume that v_i is visible from t (otherwise $A_i = \emptyset$) and discuss how to compute A_i.

Let $\widehat{l_i} = l_i \cap P$. To compute A_i, we first consider a sequence of functions $T^0, \ldots, T^n$ on V such that for a vertex $v \in V$, $T^j(v)$ $(1 \leq j \leq n)$ is defined as the set of all points r on $\widehat{l_i}$ that can send v to v_i (and therefore from v_i to t) by at most j jumps.

Proposition 3.1. *For a boundary point x and a point $r \in P$, by activating a repulsor at r, x can have at most n jumps before it stops.*

This is because if x jumps from an edge of ∂P, it can never back to that edge again (because it always moves in a direction that gets father from the repulsor). According to the above proposition, $T^n(v)$ returns all of the points on $\widehat{l_i}$ that can send v to v_i. In order to build this sequence of functions, we use a procedure called *expand*. This procedure gets

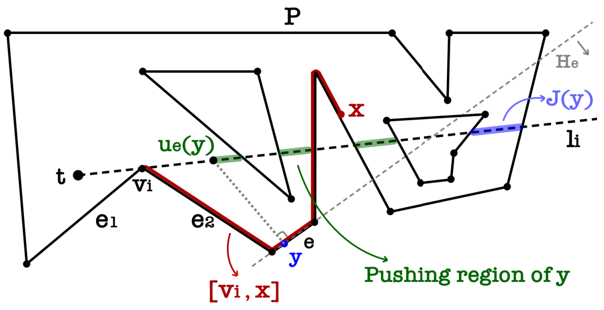


Figure 3.3: $[v_i, x]$, $u_e(y)$ and the pushing region of y.

T^{j-1} as its input and builds T^j. Running this procedure n times starting from T^0 gives us T^n. In the next sections, we discuss how to compute T_0 and the procedure *expand*.

3.3.1 Computing $\mathbf{T^0}$

Lets e_1 and e_2 be the two incident edges of v_i. We can consider that e_1 and e_2 lie on a same side of l_i otherwise it is impossible for v_i to be the last bend point for a vertex $v \neq v_i$ when we have a repulsion on l_i. Thus, if e_1 and e_2 are on different sides of l_i, we have $A_i = \{v_i\}$. Let e_1 be the closer edge to t (closer in the sense that if we consider a half-line from t that passes both e_1 and e_2, its intersection point with e_1 is closer to t than its intersection point with e_2). For a point x on the component of v_i in ∂P (∂P may not be connected because P is a polygonal domain), we denote the part of the component between v_i and x starting from e_2 by $[v_i, x]$. We chose this direction because it is impossible for a repulsion point on $\widehat{l_i}$ to send a point to t along e_1. Figure 3.3 shows an example of $[v_i, x]$. Let e be an edge of ∂P, then the interior of P should lie on one side of e. We call this side as the *P-side* of e. Also, the supporting line of e divides the plane into two half-planes. Denote the half-plane not in the P-side of e by H_e. For a point $x \in \partial P$, we define $J(x) \subseteq \widehat{l_i}$ as the set of points $r \in \widehat{l_i}$ such that when we activate a repulsor at r, x immediately jumps off from the boundary. Figure 3.3 shows an example of $J(y)$ for a point y on the interior of an edge of ∂P. The following proposition shows the connection between $J(x)$ and the edge(s) containing it:

Proposition 3.2. *For any point $x \in \partial P$, we have:*

1. *If x lies on the interior of an edge e we have $J(x) = H_e \cap \widehat{l_i}$.*
2. *If x is a reflex vertex with incident edges e' and e'', we have $J(x) = (H_{e'} \cup H_{e''}) \cap \widehat{l_i}$.*
3. *If x is a convex vertex with incident edges e' and e'', we have $J(x) = H_{e'} \cap H_{e''} \cap \widehat{l_i}$.*

Proposition 3.2 says that if x is a reflex vertex, putting a repulsor at a point in $\widehat{l_i}$ makes x jump if and only if the repulsor makes it jump from one of the supporting lines of e' and e''. Similarly, if x is a convex vertex, x jumps if and only if a repulsor make it jump from both the supporting lines of e' and e''. Figure 3.4 shows an example:

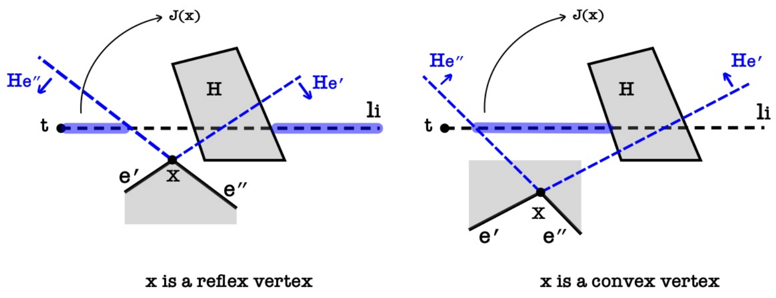


Figure 3.4: When we activate a repulsor on $J(x)$, x jumps immediately into the interior of P.

Let us denote the supporting line of l_i by $sup(l_i)$. For a point x on an edge e of ∂P, consider the half-line from x perpendicular to e toward the P-side of e. If this half-line intersects $sup(l_i)$, we denote this intersection point by $u_e(x)$ otherwise $u_e(x)$ is undefined. Note that if e does not intersect $sup(l_i)$, then u_e is defined for either all or none of the points of e. In the first case, we simply say that u_e is defined and in the second case we say u_e is undefined. For a point x on the component of v_i, we say a point $r \in \widehat{l_i}$ pushes x into $[v_i, x]$ if by activating a repulsor at r, x moves along the component toward the inside of $[v_i, x]$ (x does not jump off from the boundary and enters to the interior of $[v_i, x]$). We define the *pushing region* of x as the set of points in $\widehat{l_i}$ that push x into $[v_i, x]$ and denote it by $Push(x)$. For v_i, we define $Push(v_i) = \widehat{l_i} \setminus tv_i$. Figure 3.3 shows the pushing region of the point y.

Let $(f_0, f_1, f_2, \dots)$ be the sequence of intersection points of the component of v_i and $sup(l_i)$ when we traverse it from v_i starting along e_2 (so, $f_0 = v_i$). These f_i points break each edge of the component of v_i passing through $sup(l_i)$ into two parts. We consider each of these parts as a separate edge. By this modification, we have the sequence $(e_2, e_3, \dots, e_1)$ of edges of the component of v_i and the order on this sequence is the order as we traverse the component starting from e_2. Also, each edge lies in one side of $sup(l_i)$. We assume $e(f_k)$ as the incident edge of f_k on $[v_i, f_k]$ (consider $e(f_0)$ as e_1). So, $(e_2, e_3, \dots, e(f_1))$ is the sequence of edges in $[v_i, f_1]$. Orient P so that $\overrightarrow{v_i t}$ is directing leftward. For any edge $e_k \in (e_2, \dots, e(f_1))$, we denote its right vertex by $a(e_k)$ and its left vertex by $b(e_k)$. Then we have:

Proposition 3.3. *u_{e_k} is defined if and only if traversing the boundary starting from any point in e_k in the direction $\overrightarrow{a(e_k)b(e_k)}$ goes to v_i via e_2.*

Proof. We proceed by induction on k. Trivially the proposition is true for e_2 (this is because e_2 is the farther incident edge of v_i to t). Suppose that the above statements is true up to the edge e_{k-1}. Now, for the connection of e_{k-1} and e_k four cases may occur: $a(e_k) = a(e_{k-1})$, $a(e_k) = b(e_{k-1})$, $b(e_k) = a(e_{k-1})$ and $b(e_k) = b(e_{k-1})$. Also, each of $u_{e_{k-1}}$ and u_e may be defined or undefined which gives us 16 cases. But, because e_{k-1} and e_k are neighbour edges, it is impossible that $a(e_k) = a(e_{k-1})$ or $b(e_k) = b(e_{k-1})$ and both $u_{e_{k-1}}$ and u_e are defined or undefined. Similarly, it is impossible that $a(e_k) = b(e_{k-1})$ or $b(e_k) = a(e_{k-1})$ and one of e_k or e_{k-1} is defined and another isn't. So, eight cases left. Note that in these eight cases, four of them are exactly the mirror of others which exchanges left and right vertices. So, four cases left which is shown in Figure 3.5:

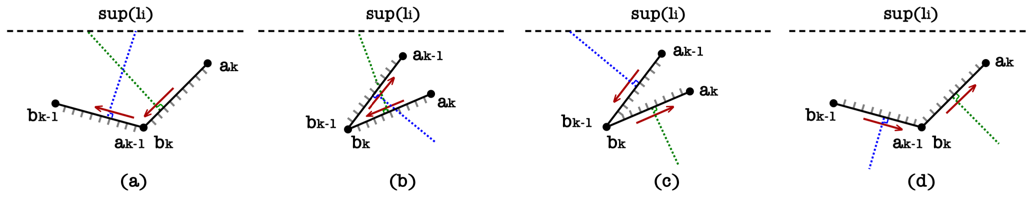


Figure 3.5: Consistency of directions to v_i in e_{k-1} and e_k according to the Proposition 3.3.

In (a) of Figure 3.5, both $u_{e_{k-1}}$ and u_{e_k} are defined and in (d) they are undefined. Also, in (b) and (c) u_{e_k} (resp. $u_{e_{k-1}}$) is defined (resp. undefined). As we can see, for all of these cases, the direction that the proposition gives for e_k is consistent with the direction the proposition gives for e_{k-1} and so this direction should go toward v_i along e_2 which proves the proposition. $\square$

Corollary 3.1. *We can extend the above proposition for each part $[f_m, f_{m+1}]$ of the component of v_i as follows:*

1. *If m is even, for $e_k \in (e(f_m)+1, \ldots, e(f_{m+1}))$, u_{e_k} is defined if and only if the direction $\overrightarrow{a(e_k)b(e_k)}$ goes to v_i via e_2.*

2. *If m is odd, for $e_k \in (e(f_m)+1, \ldots, e(f_{m+1}))$, u_{e_k} is defined if and only if the direction $\overrightarrow{b(e_k)a(e_k)}$ goes to v_i via e_2.*

The proof of the above corollary is obtained from the proof of Proposition 3.3 by simply replacing e_2 by $e(f_m)+1$. For a given point $z \in l_i$, we introduce notations $\dot{z}$ and $\ddot{z}$ as follows: z divides l_i into two parts each in one side of z. We denote the intersection of tz with P by $\dot{z}$, and the intersection of the side that does not contain t with P by $\ddot{z}$.

Corollary 3.2. *Let x be a point of an edge $e \in [f_m, f_{m+1}]$ (if x is a vertex, consider the edge in $[v_i, x]$) such that u_e is defined. Then :*

1. If m is even:

(a) If $u_e(x) \in sup(l_i) \setminus l_i$ then $Push(x) = \widehat{l_i} \setminus J(x)$.

(b) If $u_e(x) \in l_i$, we have $Push(x) = \ddot{u_e(x)} \setminus J(x)$.

2. If m is odd:

(a) If $u_e(x) \in sup(l_i) \setminus l_i$ then $Push(x) = \emptyset$.

(b) If $u_e(x) \in l_i$, we have $Push(x) = \dot{u_e(x)} \setminus J(x)$.

Proof. Suppose that r is a point of $\widehat{l_i}$. Note that if $r \in J(x)$, then it can not push x into the interior of $[v_i, x]$ so suppose that $r \notin J(x)$. According to Corollary 3.1, if e lies on the same side of e_2 with respect to $sup(l_i)$, in order that x goes to the interior of $[v_i, x]$, r should push x to the left. This can happen if and only if r lies on the right side of $u_e(x)$. Similarly, if e lies on the opposite side of e_2 with respect to $sup(l_i)$, r should push x to the right side to sent it into the interior of $[v_i, x]$ and this happens only if r lies on the left side $u_e(x)$. Note that if m is even (resp. odd), e must lie on the same side (resp. opposite side) of e_2 with respect to $sup(l_i)$ which proves the corollary. □

If an edge e is not defined, its pushing region (all of its points) is determined by the intersection point of $sup(e)$ and $sup(l_i)$ (the points on the P-side of $sup(e)$ on $\hat{l_i}$ always pushes any point $x \in e$ into $[v_i, x]$). For a vertex v on the component of v_i, we define $T^0(v)$ as the intersection of all pushing regions of points in $[v_i, v]$.

Observation 3.1. *$T^0(v)$ is exactly the set of repulsion points of v that sends the particle at v along $[v_i, v]$ to the vertex v_i and then, make it jump from v_i to t.*

Note that $T^0(v)$ is a set of intervals because it is an intersection of regions each of which consists of a set of disjoint intervals on l_i. Therefore, in order to obtain $T^0(v)$ for a given vertex v, we need to have the pushing region of infinitely many points but if we consider the inclusion relation on these pushing regions as a partial order on them, it is enough to only consider the minimal pushing regions.

Observation 3.2. *For a given vertex v, the minimal pushing regions of points in $[v_i, v]$ are among the following candidate regions:*

$$Candidate\ regions\ for\ v\ = \{Push(v') \mid v'\ is\ a\ vertex\ in\ [v_i, v]\} \tag{3.3}$$

To see why, first note that for any edge e, $J(x)$ is the same for all $x \in e$. So, for any point x on the interior of e, by slightly moving x on e we can get a pushing region not greater than the pushing region of x. Thus, in order to compute $T^0(v)$ for a given vertex v, we first compute these candidate regions according to Corollary 3.2. Then, we intersect them to obtain $T^0(v)$. Because $T^0(v)$ is a set of disjoint intervals, we can represent it with a sequence of points with even length according to their closeness to t. In this representation, the first and the second elements of the sequence represent the first interval and so on. Also, because all points are on l_i, we can represent each point by its distance to t.

3.3.2 The Expand Procedure

To explain the *expand* procedure, we assume that we have computed $T^{j-1}(v)$ for all vertices in V and discuss how to compute $T^j(v)$ for a given vertex $v \in V$. Because each point can have at most n jumps when we put a repulsor in P, we can say $v \in A_i$ if and only if $T^n(v) \neq \emptyset$. For a fixed vertex $v \in V$, we have:

$$T^j(v) = \Big(T^j(v) \cap J(v)\Big) \cup \Big(T^j(v) \cap \overline{J(v)}\Big) \tag{3.4}$$

where $\overline{J(v)}$ is the complement of $J(v)$ with respect to $\widehat{l_i}$. In the above equation, let us call the first intersection by $N_1(v)$ and the second intersection by $N_2(v)$. We first compute $N_1(v)$ for all vertices and then compute $N_2(v)$ for each vertex in V using our information about $N_1(v)$ for the vertices of P. By computing the union of $N_1(v)$ and $N_2(v)$ for each vertex, we can obtain $T^j(v)$.

Computing $\mathbf{N_1(v)}$.

In order to compute $N_1(v)$, we need a map of $J(v)$ denoted by M_v^1. For each vertex $v' \in V$, there is a corresponding region in M_v^1 denoted by $M_v^1(v')$ such that for all $z \in M_v^1(v')$, z makes v jump off from the boundary and then sends it to v' as its first visiting vertex (v' is the first vertex that v reaches after jumping). Note that some regions of M_v^1 might be empty. To construct this partition, we build the *visible triangle decomposition* of P with respect to v denoted by $VTD(v)$. This decomposition partitions the region of P visible from v by the set of line segments from v passing all vertices visible from v. Figure 3.6 shows an example of $VTD(v)$.

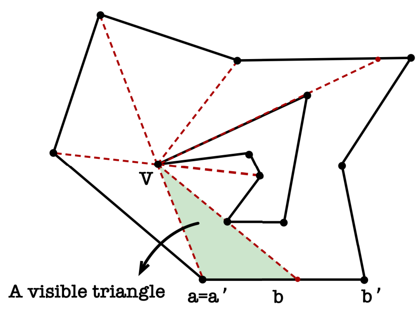


Figure 3.6: Visible triangle decomposition of P according to v.

Let vab be a triangle in $VTD(v)$ with base edge $e = ab$ and sides va and vb. Note that a and b may not be vertices of P. Also, let c_{vab} be the opposite cone of the triangle (the cone with vertex v and half-line sides along va and vb from v in the directions of $\overrightarrow{av}$ and $\overrightarrow{bv}$ respectively). Also, let $\widetilde{vab}$ be the intersection of c_{vab} with $\widehat{l_i}$. $\widetilde{vab}$ becomes empty if there is no such intersection. In this case, no point on $\widehat{l_i}$ that can make v jump into the triangle and thus, we don't consider this triangle in computation of $N_1(v)$. So, we assume that $\widetilde{vab} \neq \emptyset$. The property of $\widetilde{vab}$ is that any point in this region makes v jump into triangle vab. If we denote the vertices of P next to a and b by a' and b' respectively, we can find the partition of $\widetilde{vab}$ into subsets $\widetilde{vab}_a$ and $\widetilde{vab}_b$ such that the points in $\widetilde{vab}_a$ send v to a' and the points in $\widetilde{vab}_b$ sends v to b' (it is possible that one of these part becomes empty). Figure 3.7 shows this configuration:

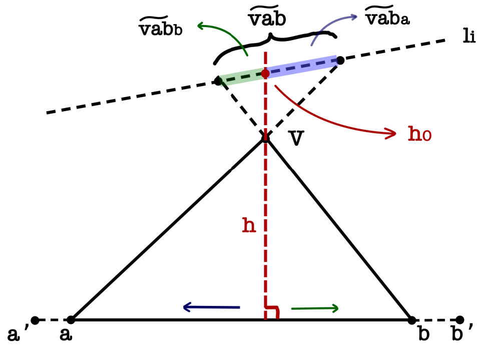


Figure 3.7: Obtaining $\widetilde{vab}_a$ and $\widetilde{vab}_b$.

In order to compute $\widetilde{vab}_a$ and $\widetilde{vab}_b$, consider the line h perpendicular to the supporting line of e passing through v. The intersection point h_0 of h and $sup(l_i)$, divides $sup(l_i)$ into two parts. The intersection of these parts with $\widetilde{vab}$ becomes $\widetilde{vab}_a$ and $\widetilde{vab}_b$. In fact, points in the b-side (resp. a-side) of h in $\widetilde{vab}$, sends v to a' (resp. b'). We apply the above method to all triangles in $VTD(v)$ and put all regions on $J(v)$ that send v to v' in $M_v^1(v')$. Having the map M_v^1, we have:

$$N_1(v) = \bigcup_{v' \in V} \Big(M_v^1(v') \cap T^{j-1}(v') \Big) \tag{3.5}$$

This is because if for a vertex v', a point r is in $M_v^1(v') \cap T^{j-1}(v')$, r sends v to v' and because r is also in $T^{j-1}(v')$, r can send it from v' to v_i using at most $j-1$ jumps. So, in total r can send v to v_i by at most j jumps.

Computing $\mathbf{N_2(v)}$.

In order to obtain $N_2(v)$, again we need a map on $\overline{J(v)}$ denoted by M_v^2. In this map, for each vertex $v' \in V$, there is a corresponding region in M_v^2 denoted by $M_v^2(v')$ which is the subset of $\overline{J(v)}$ such that each $z \in M_v^2(v')$ sends v to v' without jumping (v' is not necessarily the first vertex that v reaches). According to this definition, the regions of M_v^2 may overlap each other and some regions may become a subset of another. Instead of directly computing M_v^2, we compute two maps M_v^{21} and M_v^{22} separately such that each $z \in M_v^{21}(v')$ (resp. $z \in M_v^{22}(v')$) sends v to v' without jumping on the clockwise (resp. counter-clockwise) path on the component of v. It is clear that:

$$M_v^2(v') = M_v^{21}(v') \cup M_v^{22}(v') \tag{3.6}$$

We describe how to compute M_v^{21} and computing M_v^{22} is similar. We assume that $M_v^{21}(v) = \overline{J(v)}$. It is trivial that if v' is not in the component of v we have $M_v^{21}(v') = \emptyset$. Let $e = ab$ be an edge of the component of v. The perpendicular line on the supporting line of e passing from a divides the plane into two half-planes. Denote the half-plane doesn't include b by H_e^a. Note that any $r \in H_e^a \cap \overline{J(a)}$ sends a to b without jumping. So, we have:

Proposition 3.4. *If $M_v^{21}(v') \neq \emptyset$ and v'' is the neighbour of v' not in the clockwise path vv' on the component. Then we have:*

$$M_v^{21}(v'') = M_v^{21}(v') \cap H_{v'v''}^{v'} \cap \overline{J(v')} \tag{3.7}$$

According to the above proposition, we can start from v and traverse the component of v clockwise and build the regions of M_v^{21} (note that there must be a vertex v' on the component with $M_v^{21}(v') = \emptyset$). After computing M_v^2, we can construct $N_2(v)$ as follows:

$$N_2(v) = \bigcup_{v' \in V} \Big(M_v^2(v') \cap N_1(v') \Big) \tag{3.8}$$

Note that if $r \in M_v^2(v') \cap N_1(v')$ for a vertex $v' \in V$, r can send v to v' without jumping and because $r \in N_1(v')$, r can send it from v' to v_i using at most j jumps. This means that r can send v to v_i with at most j jumps.

3.3.3 Building $\mathbf{A_i}$

After computing $N_2(v)$, we have $T^j(v) = N_1(v) \cup N_2(v)$ and we go for the next iteration until computing $T^n(v)$ for all vertices $v \in V$. We include all vertices with $T^n(v) \neq \emptyset$ in A_i. Now, a point $x \in P$ is in A_i if there exist $r \in \widehat{l_i}$ that sends x to a vertex v as its first visiting vertex and $r \in T^n(v)$. To obtain all points in A_i, we consider each pair (v, e) individually where v is a vertex in V and e is an incident edge of v and compute a set $A_i^{(v,e)}$ which is the subset of A_i that can be sent to v_i by reaching v as their first vertex via e. So, we have:

$$A_i = \bigcup_{\textit{All pairs } (v,e)} A_i^{(v,e)} \tag{3.9}$$

Here, suppose that a pair (v, e) is given and we discuss how to compute $A_i^{(v,e)}$. Let $I_1, \ldots, I_q$ be the set of intervals of $T^n(v)$. We denote by $A_i^v(k)$ as the set of all points of P that can be sent to v as their first visiting vertex via e by some point in I_k $(1 \leq k \leq q)$. So,

$$A_i^{(v,e)} = \bigcup_{k \in \{1,\ldots,q\}} A_i^{(v,e)}(k) \tag{3.10}$$

Again we just need to compute each $A_i^{(v,e)}(k)$ independently. Let $I_k = [r_1^k, r_2^k]$ where r_1^k and r_2^k are two endpoints of I_k. For two points $r \in [r_1^k, r_2^k]$ and y on e, the segment ry might have some intersections with ∂P and so, these intersection points divide ry into a set of segments. We call the segment incident to e as the first segment of ry and denote it by $FS(ry)$. Note that it is impossible for the points on $ry \setminus FS(ry)$ to reach v as their first visiting vertex. Also, let $p(r)$ be the intersection point of $\mathit{sup}(e)$ and the line perpendicular to $\mathit{sup}(e)$ passing from r. Now, $FS(ry) \subseteq A_i^{(v,e)}(k)$ if and only if $y \in e \cap vp(r)$. So, for a given point $r \in I_k$, we can compute the set of all such $FS(ry)$ as follows: we consider the set of lines passing through r and every vertex inside the triangle obtained by r and the endpoints of $e \cap vp(r)$. These lines and ∂P partition the triangle. The union of parts incident with e are exactly the set of all $FS(ry)$ with $y \in e \cap vp(r)$. Figure 3.8 shows such configuration:

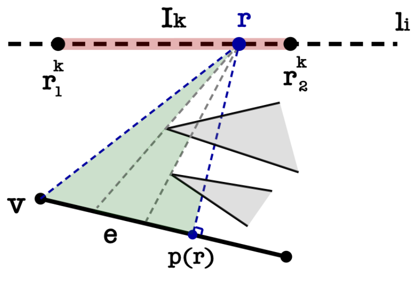


Figure 3.8: An example of $\mathit{Region}(r)$.

Lets denote this union by $\mathit{Region}(r)$. So, we have:

$$A_i^{(v,e)}(k) = \bigcup_{r \in I_k} \mathit{Region}(r) \tag{3.11}$$

In order to compute the union of infinitely many regions, let $(\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{d_k})$ be the sequence of points on I_k such that $\alpha_0 = r_1^k, \alpha_{d_k} = r_2^k$ and for each $0 < w < d_k$, $\alpha_w p(\alpha_w)$ or $\alpha_w v$ passes a vertex of ∂P as we traverse I_k from r_1^k to r_2^k. Now, as r moves from α_w to α_{w+1}, the segments of the boundary of $\mathit{Region}(r)$ changes uniformly. So, to see that which points are covered by $\mathit{Region}(r)$ when r ranges in $[\alpha_w, \alpha_{w+1}]$, it is enough to check these segments at $r = \alpha_w$ and $r = \alpha_{w+1}$. So, $A_i^{(v,e)}(k)$ is computed by considering all intervals $[\alpha_w, \alpha_{w+1}]$.

3.4 Complexity of the Algorithm

The first part of the algorithm is computing A'. It costs $O(n^2)$ to obtain the subdivision (induced by ∂P and the half-lines in L) and build the cones. Because the total number of segments in each cone is linear, we can check in linear time if a cone has more than one segment and store its first triangle. Since there are a linear number of cones, computing A' costs $O(n^2)$. We can also compute the f_ms sequence for each vertex v_i using this subdivision.

Computing A_i takes three independent steps: computing T^0, computing T^n using the *expand* procedure and building A_i having T^n. In computing T^0, first we use linear time (using the map we obtained to build cones) to check which vertices are visible from t and find the neighbor edges of the vertices that lie on the same side of the line connecting them to t. Computing the $J(v)$ for a vertex $v \in V$ takes linear time. Also, the intersection and union operations can be done linearly. In order to compute $T^0(v)$ for a given vertex v, we should compute $\dot{u}_e(v') \setminus J(v')$ or $\ddot{u}_e(v') \setminus J(v')$ (e is incident to v') for $O(n)$ vertices which again costs $O(n^2)$. So, building T^0 costs $O(n^3)$.

In the procedure *expand*, we need to compute $T^j(v)$ for all $v \in V$ having T^{j-1}. For a given $v \in V$, the maps M_v^1 and M_v^2 are independent of j and so, we can build them once and use them whenever they are needed in the *expand* procedure. In order to compute these maps, we spend $O(n \log n)$ time to build $VTD(v)$. Next, we have $O(n)$ triangles and it take constant time for each triangle to obtain $\widetilde{vab}_a$ and $\widetilde{vab}_b$. So, building M_v^1 costs $O(n \log n)$. Computing each of M_v^{21} and M_v^{22} costs $O(n^2)$ because we need to traverse the component of v and in each step, we should compute an intersection. So, M_v^2 can be computed in $O(n^2)$ and thus, building these maps for all vertices costs $O(n^3)$.

Note that each $T^0(v)$ has at most $O(n)$ endpoints and thus we have at most $O(n^2)$ endpoints in the intervals of $T^0(v)$ for all $v \in V$. On the other hand, the regions of both M_v^1 and M_v^2 have at most $O(n)$ endpoints and so, we have at most $O(n^2)$ endpoints for all maps. Now, because we don't introduce any new endpoint in the expand procedure, $T^n(v)$

should have at most $O(n^2)$ endpoints. For a fixed vertex v, $T^{j-1}(v) \subseteq T^j(v)$. So, in the expand procedure, we can compute the equations (4) and (7) for $T^{j-1}(v') \setminus T^{j-2}(v')$ instead of $T^{j-1}(v')$ and add the results to the $N_1(v)$ and $N_2(v)$ in the previous iteration to obtain new $N_1(v)$ and $N_2(v)$. So, by this modification in obtaining $N_1(v)$ and $N_2(v)$, computing $T^1(v), \ldots, T^n(v)$ costs $O(n^2)$ and because we have n vertices, computing T^n takes $O(n^3)$.

In order to build A_i, for each pair (v, e), we have at most $O(n)$ α_w points in total (for all I_k). We need to spend $O(n \log n)$ time to have these points sorted on each I_k. Now, for each interval $[\alpha_w, \alpha_{w+1}]$, in a constant time we can obtain which points are covered by $Region(r)$ for some r in this interval. Because we have at most $O(n)$ pairs (v, e), Building A_i having T^n costs $O(n^2 \log n)$. So, in total building A_i costs $O(n^3 + n^3 + n^2 \log n) = O(n^3)$. Finally, because i varies between 1 and n, the total complexity of the algorithm is $O(n^4)$.

Part II

The Proximity Connected k-center problem

Chapter 4

A Sub-quadratic Time Algorithm for the Proximity Connected k-Center Problem on Paths

In this chapter[1], we study the proximity connected k-center (PCkCP) on paths. In this problem, we are given a set of demand points in a path and a parameter $\delta > 0$. We are going to locate k center points on the path (centers can lie in the interior of the edges) such that the maximum distance of a demand point to its nearest center is minimized while the centers satisfy the proximity connectedness condition (PCC) with respect to δ. We present a sub-quadratic time algorithm for the problem for both unweighted and weighted demand points cases.

[1]A preliminary version of this chapter appeared in the proceedings of the *34rd Canadian Conference on Computational Geometry, CCCG 2022. [10]*

4.1 Background and Previous Works

The k-center problem is one of the most important facility location problems which has been extensively studied in the past [28, 32, 40, 41, 66]. In the *weighted* k-center problem, we are given a set of n demand points $\mathbb{P} = \{v_1 \ldots, v_n\}$ in a metric space such that each demand point $v_i \in \mathbb{P}$ has a non-negative weight w_i. The objective is to find a k-center (a set of k points in the space) C such that $cost(C)$ defined as $\max_{v_i \in \mathbb{P}}\{w_i d(v_i, C)\}$ is minimized, where $d(v_i, C) = min_{c \in C} d(v_i, c)$. We call this minimum cost the *optimal cost* for the problem. If we have unit weights on all demand points, the problem is called *unweighted*. We recall that a k-center C satisfies the proximity connectedness condition (PCC) with respect to a parameter $\delta > 0$ if its δ-distance graph is connected. In the proximity connected k-center problem (PCkCP), in addition to $\mathbb{P}$, we are also given a parameter $\delta > 0$ and we are going to find a k-center with minimum cost that satisfies the PCC.

In practice, if we consider the centers as facility locations, the parameter δ can represent the range for which, each facility can directly communicate with any other facility within the range δ of itself. For example, suppose that we need to locate k communication/control equipment to observe n sensors while the equipment need to send/receive messages between themselves (directly or via other equipment). Also, each equipment can safely send/receive data with any other equipment within the range δ of itself. The problem of locating the equipment as close as possible to the sensors can be modeled as PCkCP.

If we consider δ sufficiently large, the problem reduces to the k-center problem, the PCkCP becomes NP-hard in metric graphs [32, 53]. In [32], Kariv and Hakimi showed that the k-center problem can be solved in polynomial time when the underlying space is a tree and gave an $O(n^2 \log n)$ time algorithm for the problem. In 1991, Frederickson [28, 29] showed that the unweighted k-center problem can be solved in linear time in trees. Finally, in 2018, Wang and Zhang [66] provided an $O(n \log n)$ time algorithm for the k-center problem in trees. The PCC condition first appeared in the context of wireless networks in 1992 [36]. Later, Huang and Tsai studied the 2-center problem in the plane, considering the proximity condition between the centers [37, 38]. As another work, in 2022, Bhattacharya et al. [9] presented an $O(n^2 \log n)$ time algorithm (next chapter) to solve the proximity connected 2-center problem in the plane improving the previous algorithm for the problem with $O(n^5)$ time complexity [36]. Although there are some related works in the context of theory of wireless sensor networks [4, 58], the k-center problem has not been studied when we have the proximity condition between the centers. In this chapter, we address this problem by providing a sub-quadratic time algorithm for the k-center problem on paths having the PCC.

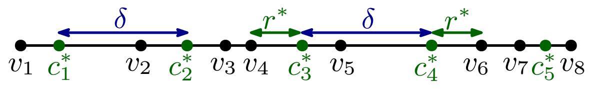


Figure 4.1: An example of the PCkCP on a path with 8 vertices.

4.2 PCkCP for Unweighted Paths

Let $P = (v_1, \ldots, v_n)$ be the given unweighted path (consisting of both the vertices and the edges between them) such that the vertices lie on the x-axis from left to right and v_1 lies on the origin. Without loss of generality, we assume that n is a power of 2. Also, we use the notation v_i $(1 \leq i \leq n)$ for both the vertex itself and the x-coordinate of the vertex. Thus, we have an order on the vertices based on their x-coordinates. Also, if $v_i < v_j$, we denote the interval between v_i and v_j on the x-axis by $[v_i, v_j]$. In this section, we are going to find a k-center C^* for P such that C^* satisfies PCC and,

$$\begin{aligned} cost(C^*) = min\{cost(C) :& \\ & C \text{ is a } k\text{-center for } P \text{ and satisfies PCC}\} \end{aligned}$$

We call C^* an *optimal solution* and its cost the *optimal cost*. We denote the optimal cost by r^* and the centers in C^* by $(c_1^*, \ldots, c_k^*)$ from left to right on the x-axis. Figure 4.1 shows an example of the PCkCP on a path and its corresponding optimal solution.

The idea of obtaining an optimal solution for the problem is first computing r^* and then, using it to build an optimal solution. In order to do that, we first design a feasibility test for the problem which gets a value $r \geq 0$ and determines whether it is feasible $(r \geq r^*)$ or infeasible $(r < r^*)$. Procedure UPATH-FT in Algorithm 5, presents such a feasibility test for the unweighted PCkCP on paths. Note that if $r \geq r^*$, UPATH-FT(P,r) also gives us a k-center with a cost at most r. Using the feasibility test, we can check whether $r^* = 0$. In this case the trivial solution is putting a center at each vertex. So henceforth, we assume that $r^* > 0$.

Algorithm 5 UPATH-FT(P, r)

```
1: Set $Counter = 1$ and $V = (v_2, \dots, v_n)$.
2: Put a center at $x_c = r$.
3: while there is an element in $V$ do
4:     Eliminate all vertices $v \in V$ with $d(x_c, v) \leq r$.
5:     Put a center at $x_c = min\{x_c + \delta, V[1] + r\}$.
6:     $Counter = Counter + 1$.
7:     if $Counter > k$ then
8:         return infeasible.
9:     end if
10: end while
11: return feasible.
```

Note that in Algorithm 5, the vertices in V are eliminated in order and so the time complexity of UPATH-FT is $O(n+k)$. It is important to mention that we might have more than one optimal solution for a given problem instance but, having r^* (which is unique), the algorithm UPATH-FT gives us a unique optimal solution. In order to avoid confusion, henceforth we exclusively use the notation C^* for this optimal solution. We say that a vertex v is *covered* by a center $c_i^* \in C^*$ if $d(v, c_i^*) = d(v, C^*)$. Also, $d(v, c_i^*)$ is called the cost that c_i^* induces on v. We say that a sequence of t points $(c_1, \dots, c_t)$ (the order is left to right on the x-axis) is a *t-train* if $\forall\ 1 \leq i < t,\ d(c_i, c_{i+1}) = \delta$.

Proposition 4.1. *There exists a pair of vertices (v_i, v_j) such that the subset $C' \subseteq C^*$ of centers in $[v_i, v_j]$ is a t-train (for some t) and $d(v_i, C') = d(v_j, C') = r^*$.*

The reason for the above proposition is that if such a pair does not exist, for any vertex v with $d(v, C^*) = r^*$, we can move the covering center of v (and possibly other centers to ensure the PCC) toward v to get a solution with a cost smaller than r^*, which contradicts the optimality of r^*. We call any pair (v_i, v_j) satisfying the condition of Proposition 4.1, a *determining pair* for the problem.

Proposition 4.2. *If $d(v_1, v_n) \geq k\delta$, then (v_1, v_n) is a determining pair for the problem.*

Proof. For any vertex v in $[c_1^*, c_k^*]$, $d(v, C^*)$ is at most $\delta/2$ because of the PCC. So, if $d(v_1, v_n) \geq k\delta$, the cost of C^* should be greater than or equal to $\delta/2$ which means that (v_1, v_n) is a determining pair. □

Based on the above proposition, if $d(v_1, v_n) \geq k\delta$, we have $d(c_1^*, c_k^*) = (k-1)\delta$ and $d(v_1, c_1^*) = d(c_k^*, v_n)$. Therefore, $r^* = (d(v_1, v_n) - (k-1)\delta)/2$. Now, UPATH-FT$(P, r^*)$ will give us C^*. Henceforth in this section, we assume that $d(v_1, v_n) < k\delta$ and so $0 < r^* < \delta/2$ (because of the PCC). In order to find r^*, we build a set of candidate values $\mathcal{C}$ and iteratively use the feasibility test to discard its values until r^* becomes clear. Consider a pair

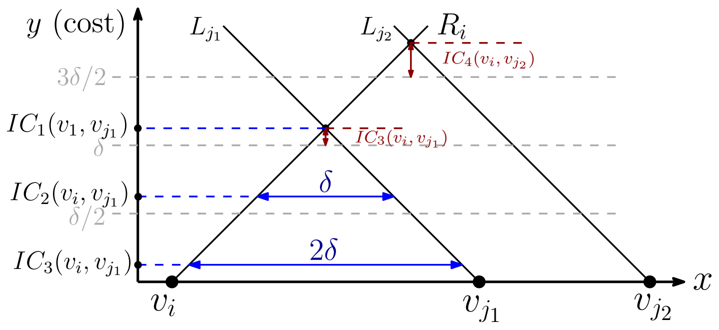


Figure 4.2: The geometric view of the candidate values generates by (v_i, v_{j_1}) and the effective candidate value generated by (v_i, v_{j_2}).

of vertices (v_i, v_j) and a t-train T such that $d(v_i, v_j) > (t-1)\delta$. We say that T is *fitted* in $[v_i, v_j]$ if $d(v_i, T) = d(v_j, T)$. Note that if T is fitted in $[v_i, v_j]$, the *induced cost* of T on v_i and v_j is $(d(v_i, v_j) - (t-1)\delta)/2$ and is denoted by $IC_t(v_i, v_j)$. If $d(v_i, v_j) \leq (t-1)\delta$, we say that (v_i, v_j) does not accept a t-train. Note that any pair of vertices accepts 1-train which is indeed the mid-point of the connecting segment of v_i and v_j. Based on Proposition 4.1, the set of candidate values $\mathcal{C}$ can be considered as follows:

$$\mathcal{C} = \{IC_t(v_i, v_j) \ : \ (v_i, v_j) \text{ accepts a } t\text{-}train\} \tag{4.1}$$

Because each pair of vertices can *generate* up to $O(k)$ candidate values, the size of $\mathcal{C}$ is $O(n^2 k)$. A naive algorithm to find r^* is computing the entire $\mathcal{C}$, then sort it and perform binary search using the feasibility test to find r^*. It is easy to see that the time complexity of this approach is $O(n^2 k \log(n+k))$. In the rest, we show that how we can reduce this bound and get a sub-quadratic time algorithm but before, it is useful to discuss about the geometric interpretation of the candidate values.

Geometric View: Let L_i and R_i be two half-lines from v_i with angles $\pi/4$ and $3\pi/4$ with the positive direction of the x-axis respectively. Note that the y-coordinate of the intersection of a vertical line at point x with $L_i \cup R_i$ is the cost that a center at x will induce on v_i (this is because we assumed that the vertices are unweighted). Based on this observation, for a pair (v_i, v_j), $IC_1(v_i, v_j)$ is the y-coordinate of the intersection point of R_i and L_j. Furthermore, if (v_i, v_j) accepts a t-train, $IC_t(v_i, v_j)$ would be the y-coordinate of the horizontal segment with length $(t-1)\delta$ with sides on R_i and L_j (see Figure 4.2). Based on this geometric view, the following observation can be concluded:

Observation 4.1. *If (v_i, v_j) accepts a t-train $(t > 1)$ then $IC_t(v_i, v_j) = IC_{t-1}(v_i, v_j) - \delta/2$.*

Consider a pair (v_i, v_j) and the non-zero candidate value $IC_{k'}(v_i, v_j)$ such that either $k' = k$ or (v_i, v_j) does not accept a $(k'+1)$-train (equivalently, k'-train is the longest train that can be fitted in (v_i, v_j)). According to Observation 4.1, $IC_{k'}(v_i, v_j)$ is the only candidate value that (v_i, v_j) can generate in $(0, \delta/2)$. If (v_i, v_j) generates a candidate value in $(0, \delta/2)$, we call this candidate value an *effective candidate value*. Because $r^* \in (0, \delta/2)$, we only need to search the effective candidates generated by the pairs in P in order to find r^*. Let us gather all the effective candidates into an $n \times n$ matrix M such that $M[i,j]$ is the effective candidate value generated by (v_i, v_j) if $i < j$ and zero otherwise. We can see that M is not a sorted matrix because for a fixed i, by increasing j, the number of centers in the train that induces $M[i,j]$ might change. Indeed, this is the main obstacle to get a linear time algorithm like [28, 29] for the unweighted PCkCP. Specifically, the k-center problem is equivalent to the PCkCP when $\delta = \infty$. In this case, all the effective candidates are generated by 1-trains. The key point here is that the effective cost generated by a 1-train on a pair (v_i, v_i) is an increasing function of $d(v_i, v_j)$. This monotonicity makes the matrix M sorted which plays a pivotal role in obtaining a linear time algorithm.

In order to search M in a sub-quadratic time, we define an auxiliary matrix $\bar{M}$ such that applying the feasibility test on its elements enables us to discard the elements of M in an efficient way. We define $\bar{M}$ as an $n \times n$ matrix such that:

$$\bar{M}[i,j] = max\{M[i,j'] : i < j' \leq j\} \tag{4.2}$$

Note that $\bar{M}$ is a row sorted (increasing) matrix but may not be sorted column-wise. We define the remainder function $rem_\delta(x)$ as follows:

$$rem_\delta(x) = x - \left\lfloor \frac{x}{\delta} \right\rfloor \times \delta \tag{4.3}$$

Observation 4.2. *If $i < j$, then we would have $M[i,j] = rem_\delta(d(v_i, v_j))/2$.*

This is from the fact that the size of the portion of $[v_i, v_j]$ not covered by the longest train in the interval is $rem_\delta(d(v_i, v_j))$.

Proposition 4.3. *If $M[i,j] = r^*$ then for all $i < j' < j$, $M[i,j'] \leq r^*$.*

Proof: We proceed by contradiction. Suppose that $M[i,j] = r^*$ and $\exists j' : i < j' < j$ such that $M[i,j'] > r^*$. Let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train in $[v_i, v_j]$ that induces r^* on v_i and v_j. Also, let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ that induces the cost $M[i,j']$. Note that $|C| < |C'|$, otherwise because $v'_j < v_j$, $M[i,j']$ could not be greater than $M[i,j]$. Also, $c^*_{h_1} < c_1$ because we assumed $M[i,j'] > r^*$. Now, if $c^*_{h_1+q} < v_{j'}$, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$, which contradicts the way we chose C. So, let us assume that $c^*_{h_1+q} > v_{j'}$ (see Figure 4.3).

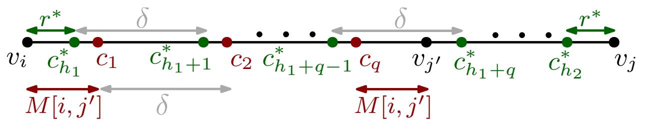


Figure 4.3: Proof of Proposition 4.3.

Here, $c^*_{h_1+q}$ is the center that covers $v_{j'}$ in C^*. If $d(v_{j'}, c^*_{h_1+q}) = r^*$, $d(v_i, v_{j'})$ would be a multiple of δ and so $M[i,j'] = 0$ which is against our assumption that $M[i,j'] > r^*$. Thus, we have $d(v_{j'}, c^*_{h_1+q}) < r^*$ but in this case we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction. □

Example: In Figure 4.4, the fitted 4-train $(c^*_1, \ldots, c^*_4)$ between v_1 and v_j induces the optimal cost r^* for the problem. In order to have $M[i,j'] > r^*$ for some $1 < j' < j$, $v_{j'}$ should lie on a *forbidden region*, which are the set of points with distances greater than r^* to their closest center (these regions are specified in red in Figure 4.4).

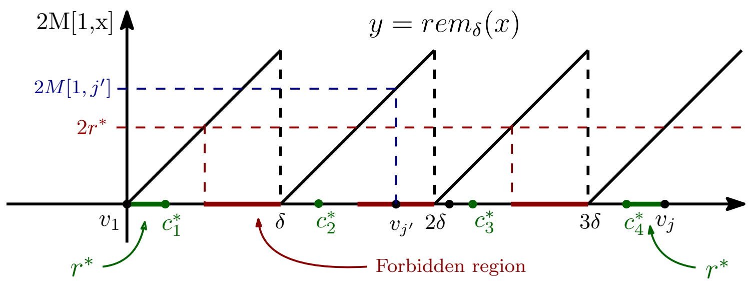


Figure 4.4: An example for Proposition 4.3.

Observation 4.3. *By applying the feasibility test on $\bar{M}[i,j]$, one of the following cases will happen:*

1. *$\bar{M}[i,j]$ is feasible. In this case, we can discard all $M[i,j']$ with $j' > j$ (based on Proposition 4.3).*

2. *$\bar{M}[i,j]$ is infeasible. In this case, we can discard all $M[i,j']$ with $j' \leq j$ (based on the definition of $\bar{M}$).*

Note that in the part 1 of the above observation, when $\bar{M}[i,j]$ is feasible, then either $\bar{M}[i,j] > r^*$ or $\bar{M}[i,j] = r^*$. For the former case, if $M[i,j'] = r^*$ for some $j' > j$, it contradicts Proposition 4.3 and for the later case we still have r^* in our undiscarded values. According to the above observation, we can find r^* by iteratively applying the feasibility test on the elements of $\bar{M}$ and discard the elements of M until r^* becomes clear. Algorithm DISC-ROUND(M) shows how we can discard $1/4^{th}$ of the undiscarded elements in M at each iteration. We can see that at the beginning of each iteration the undiscarded elements of each row make a connected region. We call this region the *undiscarded region*. Because $\bar{M}$ is row sorted, if d_1 and d_2 are the first and the last indices of the undiscarded region of an i^{th}-row in M, if we know whether $\bar{M}[i, d_1 + \lfloor (d_1 + d_2)/2 \rfloor]$ is feasible, we can discard half of the elements in the region. Note that in Algorithm 6, the variables d_1, d_2 and w_i

Algorithm 6 DISC-ROUND(M)

```
1: for $i$ from 1 to $n$ do
2:     Set $d_1$, $d_2$ and $n_i$ as the first index, the last index and the number of elements in
       the undiscarded region of the $i^{th}$-row of $M$ respectively.
3:     Set $m_i$ as $\bar{M}[i, d_1 + \lfloor (d_1 + d_2)/2 \rfloor]$.
4: end for
5: Compute the weighted median $m$ of $\{m_i : 1 \leq i \leq n\}$ where $m_i$ has weight $n_i$.
6: Run UPATH-FT($P$,$m$).
7: if $m$ is feasible then
8:     For each $i$ with $m_i \geq m$, discard $M[i, j'] : j' > m_i$.
9: else
10:    For each $i$ with $m_i \leq m$, discard $M[i, j'] : j' \leq m_i$.
11: end if
```

can be updated after the discarding phase of the previous iteration (so we don't need to search the entire matrix to compute them at the beginning of the current iteration). Also, we compute the weighted median of the mid-indexes of the undiscarded region of the rows because at the beginning of an iteration, the undiscarded region of the rows in M may not have the same size. We can see that in each iteration, we need to compute the median of $O(n)$ values in $\bar{M}$. The bottleneck of the time complexity of DISC-ROUND is the cost of obtaining an element of $\bar{M}$. Specifically, if the time complexity of computing an element of $\bar{M}$ is $O(g(n))$, then the total time complexity of DISC-ROUND would be $O(ng(n) + k)$ and so the overall time complexity of our algorithm for the unweighted PCkCP on paths is $O((ng(n) + k) \log n)$ (because we have $O(\log n)$ iterations). In the next subsection, we discuss how we can compute an element of $\bar{M}$ efficiently.

4.2.1 Computing an Element of $\bar{\mathbf{M}}$

In this subsection, we provide a preprocessing phase that enables us to compute $\bar{M}[i, j]$ in sub-linear time. Let $\mathcal{M}_{i,j} = \{M[i, i+1], \ldots, M[i, j]\}$ and so, $\bar{M}[i, j] = max\ \mathcal{M}_{i,j}$. We first build a balanced binary tree $\mathcal{T}$ on top of the vertices in P (we assumed that n is a power of 2). Thus, each leave of $\mathcal{T}$ corresponds to a single vertex. For a node $\nu \in \mathcal{T}$, *span*(ν) is defined as the set of vertices that have ν as a common ancestor. Note that the root of $\mathcal{T}$ spans the entire P. Also, we denote the first and the last indexes of the vertices in *span*(ν) by *left*(ν) and *right*(ν) respectively. In each node $\nu \in \mathcal{T}$, we store the sequence $\sigma(\nu)$ obtained from sorting $\{2M[v_1, v] : v \in span(\nu)\}$ increasingly. It is easy to see that the time complexity of building $\mathcal{T}$ and the sequences in its nodes is $O(n \log n)$ (see Figure 4.5).

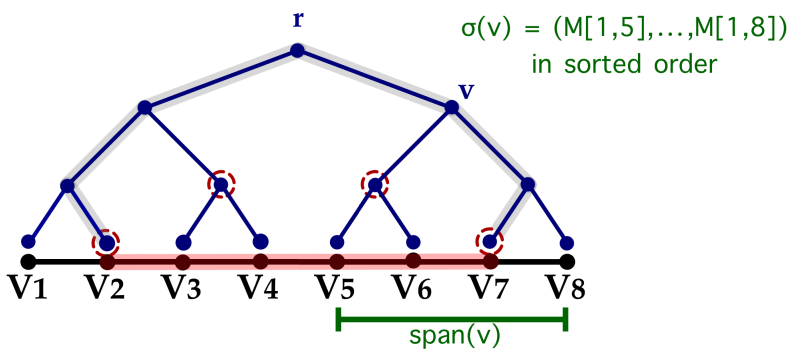


Figure 4.5: Construction of $\mathcal{T}$ on top of P.

Observation 4.4. *For any two numbers a and b, we have:*

$$rem_\delta(a+b) = rem_\delta(rem_\delta(a) + rem_\delta(b)) \tag{4.4}$$

Based on the above observation and Observation 4.2, for any $j' \geq i$ we can write $M[i,j']$ as:

$$\begin{aligned}
&M[i,j'] = rem_\delta(d(v_i, v_{j'}))/2 = \\
&rem_\delta(d(v_1, v_{j'}) - d(v_1, v_i))/2 = \\
&rem_\delta(rem_\delta(d(v_1, v_{j'})) - rem_\delta(d(v_1, v_i)))/2 = \\
&rem_\delta(2M[v_1, v_{j'}] - 2M[v_1, v_i])/2
\end{aligned}$$

Now, for each vertex ν with $\sigma(\nu) = (s_1, \ldots, s_t)$ and $i \leq \mathit{left}(\nu)$, we define $\sigma_i(\nu)$ as:

$$\sigma_i(\nu) = (rem_\delta(s_1 - 2M[v_1, v_i]), \ldots, rem_\delta(s_t - 2M[v_1, v_i])) \tag{4.5}$$

Let $\mu_i(\nu)$ be the maximum of $\sigma_i(\nu)$. Based on the above argument, we can see

$$max\{M[\mathit{left}(\nu), \mathit{left}(\nu)+1], \ldots, M[\mathit{left}(\nu), \mathit{right}(\nu)]\}$$

is indeed $\mu_i(\nu)/2$. An important observation here is that because the elements of M are at most $\delta/2$, $\sigma_i(\nu)$ is a concatenation of two sorted sequences namely $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (note that one of these sequences might be empty). So, in order to find $\mu_i(\nu)$, we need to compare the last elements of $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (if they exist) and pick the greater value. Specifically, if $s_{j'} - 2M[v_1, v_i]$ is negative (resp. positive) for some $s_{j'} \in \sigma(\nu)$, $rem_\delta(s_{j'} - 2M[v_1, v_i])$ belongs to $\sigma_i^1(\nu)$ (resp. $\sigma_i^2(\nu)$). Thus, we can do binary search to obtain the index of the last element of $\sigma_i^1(\nu)$ and so $\mu_i(\nu)$ in $O(\log|\mathit{span}(\nu)|)$ time.

We can use the above data structure to find $\bar{M}[i,j]$ as follows: we first obtain two paths π_i and π_j and their split vertex ν_{split} from the root of $\mathcal{T}$ to v_i and v_j respectively. Let $\mathcal{V}_{i,j}$ be the set of right (resp. left) children of π_i (resp. π_j) from ν_{split} to its leaf (including v_j). Now, $\mathcal{M}_{i,j} = 1/2 \cup_{\nu \in \mathcal{V}_{i,j}} \sigma_i(\nu)$ where the multiplication is done element-wise. Therefore,

$$\bar{M}[i,j] = max\ \mathcal{M}_{i,j} = max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\} \tag{4.6}$$

because $|\mathcal{V}_{i,j}| = O(\log n)$ and computing each $\mu_i(\nu)$ in (4.6) also costs $O(\log n)$, the total time complexity of computing $\bar{M}[i,j]$ is $O(\log^2 n)$ which leads to an overall $O((n \log^2 n + k) \log n)$ time complexity for the PCkCP in unweighted paths.

Further improvements: First, we observe that if for two nodes $\nu, \nu' \in \mathcal{T}$, ν' is a parent of ν then $\sigma(\nu)$ is a sub-sequence of $\sigma(\nu')$. This property enables us to use a technique called *fractional cascading* [24] to avoid doing binary search on each of the nodes in $\mathcal{V}_{i,j}$ to find their maximum. Specifically, we equip each element s of $\sigma(\nu')$ with a pointer that points to the smallest element in ν larger than or equal to s. This structure can be constructed in $O(n \log n)$ time [24]. So, in order to obtain all $\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$, we only perform one binary search on $\sigma_i(root(\mathcal{T}))$ with cost $O(\log n)$ and follow the pointers along the paths to obtain each $\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ in a constant time. So, the total time complexity of computing $\bar{M}[i,j]$ is $O(\log n)$ and so, the total running time is $O((n \log n + k) \log n)$.

As another improvement, note that we only need to do binary search on $\sigma_i(root(\mathcal{T}))$ once for each row i in the entire algorithm. Also, by spending $O(n \log n)$ time, for each root-leaf path π_i and each $\nu' \in \pi_i$, we can store $max\{\mu_i(\nu) : \nu$ is right child of a node in $\pi_i[\nu', v_i]\}$ in ν' ($\pi_i[\nu', v_i]$ is the portion of π_i from ν' to v_i) by walking along π_i twice (once for computing μ_is and once for storing the maxes). So, having ν_{split}, we only need to take care about computing $max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ and hanging from $\pi_j\}$. To address this problem, consider a fixed i^{th}-row. Based on Algorithm 6, at each iteration r, the undiscarded region of the i^{th}-row corresponds to $span(\nu^r)$ for some $\nu^r \in \mathcal{T}$. Let ν_m^r be the left child of ν^r (if we are not at the last iteration) with $m^r = right(\nu_m^r)$. We can see that m^r is the median of the undiscarded region. Now, ν_m^{r+1} is either the left child of ν_m^r or the left child of the right neighbor of ν_m^r. Let r_0 be the last iteration for which $\nu_m^{r_0}$ is on π_i. For iterations $r \leq r_0$, we only need to consider the maximum of the values in $\sigma_i(\nu')$ where ν' the first right child on π_i after ν_m^r. Also, for iterations $r > r_0$, we only need to have the set of maximum values in the left hanging nodes of $\pi_{m^r}[\nu_{split}, \nu_m^r]$ and ν_m^r itself. Now, it is easy to see that as r increases to $r+1$, these set of values can be updated in a constant time. Thus, we can conclude that computing $\bar{M}[i, m^r]$ for all iterations r only takes $O(\log n)$ time and because we have linear number of rows, we would have the following theorem:

Theorem 4.1. *The unweighted PCkCP can be solved in $O((n+k)\log n)$ time.*

4.3 PCkCP for Weighted Paths

Let $P = (v_1, \ldots, v_n)$ be the given weighted path such that w_i is the weight of v_i. For a point x on P, we define $wd(v_i, x) = w_i d(v_i, x)$. Again each pair of vertices (v_i, v_j) generates $O(k)$ candidate values which corresponds to the trains that can be fitted in $[v_i, v_j]$. Here, because the weights of v_i and v_j might be different, a train may not be required to have the same distance from v_i and v_j in order to induce the same cost on them. Again, we denote the cost that a fitted t-train in $[v_i, v_j]$ induces on v_i and v_j by $IC_t(v_i, v_j)$. Suppose that $d(v_i, v_j) > t\delta$ for some $t > 1$. We define the *width* of (v_i, v_j) as $IC_{t-1}(v_i, v_j) - IC_t(v_i, v_j)$ and denote it by $W(v_i, v_j)$. Note that this value is independent of t and only depends on w_i and w_j and so, we can compute it in a constant time (in the unweighted case, the width of all pairs in P are $\delta/2$). Because here the widths of the pairs in P might not be equal, we first need to find an interval I^* such that each pair of vertices can generate at most one cost in I^*. But before going into that, we need to update our feasibility test to support weighted vertices. Algorithm 7 presents the feasibility test procedure WPATH-FT(P,r) which gets a weighted path P and a test value r and determines whether $r \geq r^*$ or $r < r^*$.

Algorithm 7 WPATH-FT(P, r)

```
1:  Set $Counter = 1$
2:  for j=1 to n do
3:      Let $I_j = [\alpha_j, \beta_j]$ be the interval on the $x$-axis for which $wd(\alpha_j, v_j) = wd(v_j, \beta_j)$=r.
4:  end for
5:  Set $NextInterval = (-\infty, +\infty)$.
6:  Set $j = 1$.
7:  if $NextInterval \cap I_j = \emptyset$ then
8:      Put a center $c$ at the rightmost point of $NextInterval$.
9:      Counter = Counter+1
10:     if $Counter > k$ then
11:         return Infeasible.
12:     end if
13:     $NextInterval = (c, c + \delta]$.
14: else
15:     $NextInterval = NextInterval \cap I_j$.
16:     $j = j + 1$.
17: end if
18: goto Line 7 if $j \leq n$.
19: return feasible.
```

Note that in the while loop of Algorithm 7, each I_j $(1 \leq j \leq n)$ can be computed in constant time and we visit each vertex once. Therefore, the running time of the above feasibility test is $O(n+k)$. The geometric view for the weighted case is similar to the unweighted case but here, for each vertex v_i, the magnitude of the slopes of R_i and L_i is w_i. For each pair (v_i, v_j), the y-coordinate of the intersection point of R_i and L_j is the cost that a fitted 1-train (single point) in $[v_i, v_j]$ induces on v_i and v_j which is denoted by $IC_1(v_i, v_j)$. Similarly, if $d(v_i, v_j) > (t-1)\delta$, $IC_t(v_i, v_j)$ would be the y-coordinate of the horizontal segment with length $(t-1)\delta$ and endpoints on R_i and L_j (see Figure 4.6).

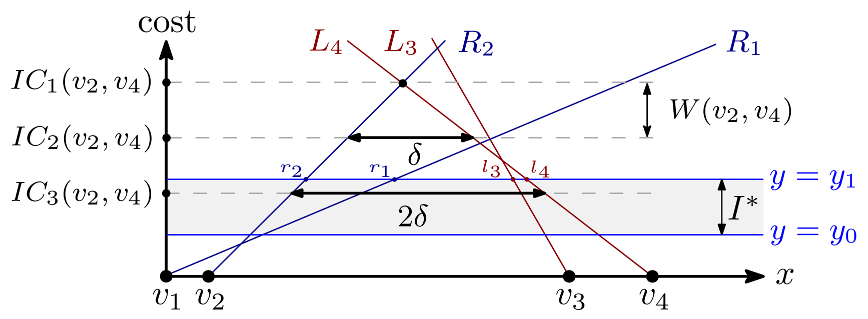


Figure 4.6: A weighted path (v_1, v_2, v_3, v_4), the width of (v_2, v_4) and three costs generated by the pair. Note that only one of them lies inside I^*.

4.3.1 Matrix Search for Weighted Paths

First, we need to build an interval $I_1 = [a, b]$ such that $r^* \in I_1$ and it's interior does not contain any $IC_1(v_i, v_j)$ for any $i < j$ (note that $IC_1(v_i, v_j)$ is indeed the y-coordinate of the intersection point of R_i and L_j). If we use Lemma 2.5 [66] on all R_i and L_j $(1 \le i, j \le n)$, we can get I_1 in $O((n+k)\log n)$ time. Let us define W^* as follows:

$$W^* = min\{W(v_i, v_j) : i < j \text{ and } IC_1(v_i, v_j) \ge b\} \tag{4.7}$$

We can see that $r^* \in I_2 := [b - kW^*, b] \cap I_1$. This is because r^* can't be smaller than the cost that a fitted k-train induces on the generating pair of W^*.

Proposition 4.4. *W^* can be computed in $O(n \log n)$ time.*

Proof. We first compute the intersection points of all L_i and R_j for $1 \le i, j \le n$ with the horizontal line $y = b$. Then, we sort these intersections on the line from left to right in $O(n \log n)$ time. So, each of these intersections corresponds to a line with a positive or a negative slope. We traverse these intersections from left to right and store the minimum positive slope and the minimum width we have seen in variables *min_slope* and *min_width* respectively. Finally, we set *min_width* as W^*. Specifically, when we visit an intersection point, if it came from a line with a positive slope, we update *min_slope* if necessary and if it came from a line with a negative slope, we compute the width it creates with the line that generated *min_slope* and update *min_width* if necessary. □

We can see that the length of I_2 is at most kW^*. This implies that by applying the feasibility test $O(\log k)$ times at the costs $b - iW^*$ $(0 \leq i \leq k)$ we get an interval $I_3 \subseteq I_2$ with length at most W^* containing r^*. Because W^* is the minimum width, each pair (v_i, v_j) with $IC_1(v_i, v_j) \geq b$ can generate at most one candidate value in I_3.

Consider the set of half-lines $\{R_1, \ldots, R_{n-1}\}$ (all with positive slopes) and their upper-envelope polygonal chain as a function $f_{UE}(x)$. We can see that f_{UE} is a piece-wise linear and an increasing function. Also, $f_{UE}(x)$ is the cost of covering all the vertices on the left side of x if we put a center at x. We can compute f_{UE} in linear time as follows: suppose that we have already computed the upper-envelope of $\{R_1, \ldots, R_{j-1}\}$ consisting of it's lines and break points. Now, when we add R_j and update our envelope, if R_j is below the last break point, we consider R_j and the last line of the envelope for a possible new break point. Otherwise, we find the first break point below the line (be checking the break points one by one from the last) and consider the line next to it (on its left) for a break point. Note that when we check a break point and it turns out it is below R_j, the line next to it (on its right) can never be a part of the envelope. Because we have linear number of lines, the time complexity of computing f_{UE} is linear.

Let $(x_1, \ldots, x_s)$ be the x-coordinates of the break points of f_{UE} where s is the number of break points. Then, we can use our feasibility test to do binary search on $\{f_{UE}(x_i) : 1 \leq i \leq s\}$ to find an interval $[x_q, x_{q+1}]$ such that $r^* \in [f_{UE}(x_q), f_{UE}(x_{q+1})]$. Let R_q (generated by v_q) be the line corresponding to the portion of f_{UE} in $[x_q, x_{q+1}]$. Then we have the following observation:

Observation 4.5. *If c_1^* induces r^*, then v_q is the first vertex of a determining pair.*

Based on the above observation, we can consider all pairs $\{(v_q, v_{q+1}), \ldots, (v_q, v_n)\}$, obtain the candidate value that each generates, sort them and do binary search (using our feasibility test) to get an interval $I^{(1)}$. Now, c_1^* can't generate any candidate value in the interior of $I^{(1)}$. Similarly, we can do the above process on $\{L_2, \ldots, L_n\}$ to get an interval $I^{(2)}$ such that c_k^* can't generate any candidate value in the interior of $I^{(2)}$. Let $I^* = I_3 \cap I^{(1)} \cap I^{(2)}$. So, it is only left to resolve the candidates in the interior of I^*.

Observation 4.6. *If (v_i, v_j) is a determining pair and a train $(c_{h_1}^*, \ldots, c_{h_2}^*)$ in $[v_i, v_j]$ induces r^* on the interior of I^*, then*

1. $0 < d(v_i, c_{h_1}^*), d(v_j, c_{h_2}^*) < \delta/2$.
2. *$(c_{h_1}^*, \ldots, c_{h_2}^*)$ is the longest train that can be fitted in $[v_i, v_j]$.*

The first part of the above observation comes from the fact that if r^* lies on the interior of I^*, then $h_1 \neq 1$ and $h_2 \neq k$. So, if for example $d(v_i, c_{h_1}^*) \geq \delta/2$ then because of the PCC,

$c^*_{h_1-1}$ can cover v_i in the optimal solution. For the second part, note that if we are able to fit a longer train in $[v_i, v_j]$ then either $d(v_i, c^*_{h_1})$ or $d(v_j, c^*_{h_2})$ is greater than $\delta/2$ which contradicts the first part.

Based on Observation 4.6, for any pair of vertices (v_i, v_j), we define our matrix M for the weighted case such that $M[i, j]$ is the cost r induced by the longest train $(c_1, \ldots, c_q)$ that can be fitted in $[v_i, v_j]$ if $r \in I^*$ and $0 < d(v_i, c_1), d(v_j, c_q) < \delta/2$. If we didn't have either of these two conditions, we assign $M[i, j] = 0$. It is clear that r^* is an element of M. Similar to the unweighted case, we define $\bar{M}[i, j]$ as $\max\{M[i, i+1], \ldots, M[i, j]\}$. Again, we can see that $\bar{M}$ is a row sorted matrix but may not be sorted column-wise. Next, we show Proposition 4.3 is still valid for our new definition of M and $\bar{M}$ in the weighted case.

Proposition 4.5. *If $M[i, j] = r^*$, then for all $i < j' < j$, $M[i, j'] \le r^*$.*

Proof. We proceed by contradiction. Suppose that (v_i, v_j) induces r^* and $\exists i < j' < j$ such that $M[i, j'] > r^*$. Let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ and induces the cost $M[i, j']$ on v_i and $v_{j'}$. Also, let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train that induces r^* in $[v_i, v_j]$. Now, $c^*_{h_1+q-1} < c_q$ (because we assumed that $M[i, j'] > r^*$) and $|C| < |C'|$ (because $v_j > v_{j'}$). We consider two cases:

case 1: $c^*_{h_1+q} \le v_{j'}$: In this case, we could fit a $(q+1)$-train namely $C'' = (c'_1, \ldots, c'_{q+1})$ in $[v_i, v'_j]$ which contradicts the fact that C was the longest train in $[v_i, v_{j'}]$.

case 2: $c^*_{h_1+q} > v_{j'}$: In this case, $v_{j'}$ should be covered from its right in C^* (because $c^*_{h_1+q-1} < c_q$ and we assumed $M[i, j'] > r^*$). Also, the cost of covering $v_{j'}$ in C^* is no more than r^*. So, if $w_i \le w_{j'}$, $d(v_i, c^*_{h_1}) \ge d(v_{j'}, c^*_{h_1+q})$ and thus, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction.

Now, assume that $w_i > w_{j'}$. Let t_1 and t_2 be the points on the right side of $v_{j'}$ such that $wd(v_{j'}, t_1) = r^*$ and $wd(v_{j'}, t_2) = M[i, j']$. Note that $t_2 > t_1$ and t_2 is the mirror image of c_q with respect to $v_{j'}$. Now, $d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (because $w_i > w_{j'}$ and the cost that v_i induces on $c^*_{h_1}$ and c_1 are r^* and $M[i, j']$ respectively). Also, because $M[i, j'] \ne 0$, $d(c_q, v_{j'}) < \delta/2$ (based on the definition of M), $c_q + \delta > v_{j'} + \delta/2$ which implies that $[t_1, t_2] \subseteq [c^*_{h_1+q}, c_q + \delta]$. This contradicts the fact that $d(c^*_{h_1+q}, c_q + \delta) = d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (see Figure 4.7). □

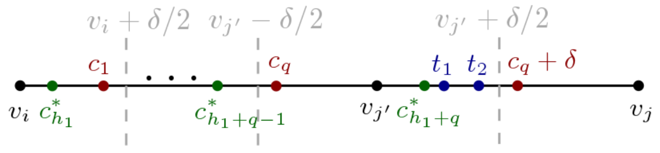


Figure 4.7: Proof of Proposition 4.5

The above proposition implies that Observation 4.3 is valid for M and $\bar{M}$ in the weighted case and so we can use Algorithm 6 to find r^* and get C^*. Based on Algorithm 6, the time complexity of finding r^* is $O((ng(n) + k) \log n)$ where $g(n)$ is the time complexity for computing an element of $\bar{M}$. In the last section of this chapter, we show how we can compute an element of $\bar{M}$ in $O(\log^3 n)$ time by spending $O(n \log^3 n)$ time for preprocessing. This gives us the following theorem:

Theorem 4.2. *The PCkCP can be solved in* $O((n \log^3 n + k) \log n)$ *time.*

4.4 Computing an Element of $\bar{\mathrm{M}}$ for Weighted Paths

In this section, we build a data structure such that for any query pair (i, j) $(i < j)$, it enables us to compute $\bar{M}[i, j] = max\{M[i, j'] : i < j' \leq j\}$ in a sub-linear time. Suppose that $I^* = [y_0, y_1]$. We denote the x-coordinates of the intersection points of L_i and R_i $(1 < i < n)$ with line $y = y_1$ by l_i and r_i, respectively (see Figure 4.6). Note that if for a pair $(v_i, v_{j'})$, $l_{j'} < r_i$, it can not generate any candidate value in I^* (because of the way we built I^*) and so, $M[i, j'] = 0$. Thus, we only consider the pairs $(v_i, v_{j'})$ for which $l_{j'} \geq r_i$. Let us define the complement function with respect to δ as:

$$comp_\delta(x) = \left\lceil \frac{x}{\delta} \right\rceil \times \delta - x \tag{4.8}$$

We also denote $comp_\delta(l_i)$ and $comp_\delta(r_i)$ by $\hat{l}_i$ and $\hat{r}_i$ respectively where $1 < i < n$. For any pair (i, j') with $j' > i$ and $l_{j'} > r_i$, let $E_{ij'} = comp_\delta(l_{j'} - r_i) = rem_\delta(\hat{l}_{j'} - \hat{r}_i)$ and $D_{ij'} = E_{ij'}/(w_i^{-1} + w_{j'}^{-1})$ (w_i and $w_{j'}$ are the magnitudes of the slopes of R_i and $L_{j'}$ respectively). Based on the geometric view, it is easy to see that $M[i, j'] = y_1 - D_{ij'}$ if $D_{ij'} \leq |I^*|$ and zero otherwise. So, the problem of finding $\bar{M}[i, j]$ is equivalent to find $D_{min} = min\{D_{ij'} : i < j' \leq j\}$. It is convenient to visualize this set as follows: for each $i < j' \leq j$, we consider $e_{ij'}$ as the point located at $(E_{ij'}, 0)$ on the x-axis. There are two half-lines corresponding to $e_{ij'}$:

1. $L^+_{ij'}$ attached to e_{ij} with slope w_i.

2. $L^-_{ij'}$ from the origin with slope $-w_{j'}$.

Figure 4.8 depicts an example of such half-lines. We can see that the distance between the intersection point of L^+_{ij} and L^-_{ij} from the x-axis is indeed D_{ij}. We call this distance the *D-coordinate* of the intersection (when a point moves downward, its D-coordinate increases). So, each value $E_{ij'}$ generates exactly one intersection D-coordinate call it the *D-value* of $E_{ij'}$. Like the unweighted case, we build a balanced binary tree $\mathcal{T}$ on top of the vertices and in each node $\nu \in \mathcal{T}$ we store $\{\hat{l}_h : v_h \in span(\nu)\}$ as an increasingly sorted sequence $\sigma(\nu)$.

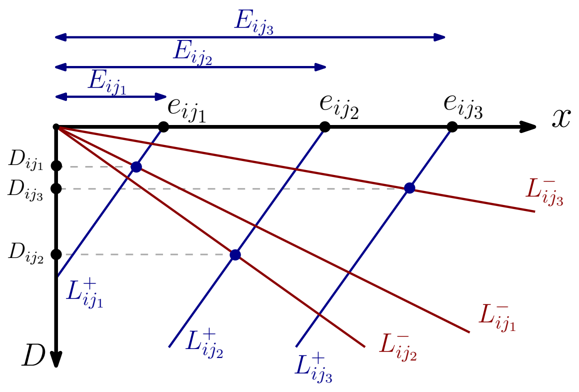


Figure 4.8: Three points e_{ij_1}, e_{ij_2} and e_{ij_3} located at distances E_{ij_1}, E_{ij_2} and E_{ij_3} respectively and their generating points. In this example, (v_i, v_{j_1}) generates the maximum of $\{M[i, j_1], M[i, j_2], M[i, j_3]\}$.

So, if we preprocess each $\nu \in \mathcal{T}$ such that for a given vertex v_i, we can quickly compute

$$\mu_i(\nu) = min\{D - value \text{ of } E_{ih} : v_h \in span(\nu)\} \tag{4.9}$$

we can decompose the set $\{v_j : i < j' \leq j\}$ into $\cup_{\nu \in \mathcal{V}_{i,j}} span(\nu)$ (as we did in Section 2.1) and set $D_{min} = min\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$

Let $\nu \in \mathcal{T}$ be a fixed node. In the rest, we show how we can preprocess ν such that given a query vertex v_i, we can efficiently compute $\mu_i(\nu)$. First, note that the set of half-lines $\{L^-_{ih} : v_h \in span(\nu)\}$ is independent of i. Also, for each i, $\{E_{ih} : v_h \in span(\nu)\}$ is the union of two sorted sequences $\sigma^1_i(\nu)$ and $\sigma^2_i(\nu)$, where $\sigma^1_i(\nu)$ (resp. $\sigma^2_i(\nu)$) is obtained by a shift (adding a constant value) of the elements in $\sigma(\nu)$ smaller than (resp. greater than or equal to) $\hat{r}_i$. Therefore, if $\mu^1_i(\nu)$ (resp. $\mu^2_i(\nu)$) is the minimum D-value generated by $\sigma^1_i(\nu)$ (resp. $\sigma^2_i(\nu)$), we have:

$$\mu_i(\nu) = min\{\mu^1_i(\nu), \mu^2_i(\nu)\} \tag{4.10}$$

Consider the lines $L^+_{ij'} : y = w_i(x - E_{ij'})$ and $L^-_{ij'} : y = -w_{j'}x$, where $E_{ij'}$ is a variable (see Figure 4.8). When $E_{ij'}$ increases, the D-value of $E_{ij'}$ (the intersection of $L^+_{ij'}$ and $L^-_{ij'}$) increases linearly. Specifically, if we set $E_{ij'}(t) = \hat{l}_{j'} + t$, the D-value generated by $E_{ij'}$ is the following linear function denoted by $\mathfrak{L}_{ij'}$:

$$\mathfrak{L}_{ij'}(t) = \frac{w_i w_{j'}}{w_i + w_{j'}}(\hat{l}_{j'} + t) \tag{4.11}$$

Let us define:

$$f(t) = min\{\mathfrak{L}_{ij'}(t) : v_{j'} \in span(\nu)\} \tag{4.12}$$

We can see that $f(t)$ is the lower-envelope of a set of lines which can be computed in $O(|\nu| \log |\nu|)$ time ($|\nu|$ is the number of vertices in $span(\nu)$) using the divide-and-conquer algorithm (we use the order in $\sigma(\nu)$ for breaking the lines). Because we need to work with the sub-sequences of $\sigma(\nu)$, we store the entire *recursion tree* [63] (with the solutions of its sub-problems) of the divide-and-conquer algorithm and denote this tree by $\mathcal{R}_i$. So, each node ω of $\mathcal{R}_i$ contains a set of lines each corresponds to a vertex $\nu_{j'} \in span(\nu)$ and their lower-envelope. We denote the indices of these vertices by $\mathcal{J}(\omega)$. Note that the indices stored in each node of $\mathcal{R}_i$ is independent of i (although the lines and the resulting lover-envelope depends on i). Also, we denote the lower-envelope stored in $\omega \in \mathcal{R}_i$ by $\mathcal{LE}_i^{\omega}$ and the indices corresponds to the lines appeared in $\mathcal{LE}_i^{\omega}$ from left to right by $\mathcal{E}_i^{\omega}$.

Based on the above discussion, one way to preprocess ν is that for each $1 < i < n$, we compute and store $\mathcal{R}_i$. Now, when we are given a vertex v_i, we first use binary search to find the last element s_{t_0} in $\sigma(\nu) = (s_1, \dots, s_t)$ smaller than $\hat{r}_i$. So, we have two sub-sequences $\hat{\sigma}_i^1(\nu) = (s_1, \dots, s_{t_0})$ and $\hat{\sigma}_i^2(\nu) = (s_{t_0+1}, \dots, s_t)$. Note that each $\mathcal{R}_i$ is a binary tree on top of $span(\nu)$. So, we can decompose $\hat{\sigma}_i^1(\nu)$ (similarly $\hat{\sigma}_i^2(\nu)$) into the union

$$\hat{\sigma}_i^1(\nu) = \cup \hat{l}_j : j \in \mathcal{J}(\omega) \text{ and } \omega \in \mathcal{W}^1 \tag{4.13}$$

for a set of nodes $\mathcal{W}^1 \subseteq \mathcal{R}_i$. Indeed, the nodes in $\mathcal{W}^1$ can be specified by considering two root-leaf paths in $\mathcal{R}_i$ in $O(\log|\sigma(\nu)|)$ time. Now, $\sigma_i^1(\nu)$ is a shifted sequence of $\hat{\sigma}_i^1(\nu)$. Let us denote the amount of this shift by x_0. In order to find $\mu_i^1(\nu)$, for each node $\omega \in \mathcal{W}^1$ we look at the value of its lower-envelope at x_0. The minimum of these values is indeed $\mu_i^1(\nu)$. Similarly, we can obtain $\mu_i^2(\nu)$ and set the minimum of these two values $\mu_i(\nu)$. Because the height of $\mathcal{T}$ is $O(\log n)$, the total time complexity of computing $\bar{M}[i,j]$ is $O(\log^2 n)$.

The problem here is that if we build $\mathcal{R}_i$ for all $1 < i < n$ and all $\nu \in \mathcal{T}$, the time complexity of the preprocessing phase is $O(n^2 \log^2 n)$. In order to make the preprocessing cost sub-quadratic, we use the following sub-sequence property for each node ω (independent of i) in the recursion trees:

Proposition 4.6 (The sub-sequence property)**.** *For any $\omega \in \mathcal{R}_i$ and two indices j_1 and j_2 such that $w_{j_1} < w_{j_2}$, $\mathcal{E}_{j_2}^{\omega}$ is a sub-sequence of $\mathcal{E}_{j_1}^{\omega}$ where w_{j_1} and w_{j_2} are the weights of v_{j_1} and v_{j_2} respectively.*

We prove the above proposition at the end of this section and here, we discuss how we use it to reduce the time complexity of the preprocessing phase and adjust the query time accordingly. First, we sort all the weights increasingly into a sequence $(w_{i_1}, \dots, w_{i_n})$. Next, we build a balanced binary tree $\mathbb{T}_W$ on top of this sequence as follows: the root r_W of $\mathbb{T}_W$ corresponds to $w_{i_{n/2}}$. Its left child corresponds to $w_{i_{n/4}}$ and its right child corresponds to $w_{i_{3n/4}}$. We recursively continue building the nodes of the tree until each weight in the sequence has a node in $\mathbb{T}_W$. In the root of $\mathbb{T}_W$, we build and store $\mathcal{R}_{i_{n/2}}$. Let $X_1 = \mathcal{E}_{i_{n/2}}$ (the result of building $\mathcal{R}_{i_{n/2}}$) and $X_2 = \{j : v_j \in span(\nu)\} \setminus X_1$. In the right child of r_W, we build and store $\mathcal{R}_{i_{3n/4}}(X_1)$ and in the left child of r_W, we build and store $\mathcal{R}_{i_{n/4}}(X_2)$ that is defined as follows: $\mathcal{R}_{i_{3n/4}}(X_1)$ (similarly $\mathcal{R}_{i_{n/4}}(X_2)$) is the tree identical to $\mathcal{R}_i$ (all the recursion trees when we process $\nu \in \mathcal{T}$ are identical but the information we store in each node is different) and a line in $\{\mathfrak{L}_{i_{3n/4}j'} : j' \in X_1\}$ is in $\omega \in \mathcal{R}_{i_{3n/4}}(X_1)$ if and only if it is in $\omega \in \mathcal{R}_{i_{n/2}}$. We can easily see that the height of $\mathcal{R}_{i_{3n/4}}(X_1)$ is $O(\log |\nu|)$ and computes the lower-envelope of the lines in X_1 denoted by $\mathcal{LE}_{i_{3n/4}}(X_1)$ in $O(|X_1| \log |\nu|)$ time. Similarly, $\mathcal{R}_{i_{n/4}}(X_2)$ and $\mathcal{LE}_{i_{n/4}}(X_2)$ can be computed in $O(|X_2| \log |\nu|)$ time. We recursively build and store the recursion trees and lower-envelopes of the nodes in $\mathbb{T}_W$ based on the above partitioning of the lines (the lines that appear in the envelope and the lines that do not appear in the envelope) in each node.

We can see that in each level of $\mathbb{T}_W$, the set of lines in the recursion trees are disjoint and their sum is $|\nu|$. Because the height of $\mathbb{T}_W$ is $O(\log n)$, the total time complexity for building the recursion trees for all nodes in $\mathbb{T}_W$ is $O(|\nu| \log |\nu| \log n)$. Because the vertices in the span of the nodes in each level of $\mathcal{T}$ are disjoint, the total time complexity for building the recursion trees for all nodes in $\mathcal{T}$ becomes $O(n \log^3 n)$.

Here, we discuss given a query vertex v_i, how we can compute $\mu_i^1(v)$ in sub-quadratic time. First, we compute $\hat{\sigma}_i^1(\nu)$ and decompose it into its corresponding set of nodes $\mathcal{W}^1$ as in Equation 4.13. Next, we find a path τ_i from r_W to w_i in $\mathbb{T}_W$. Let t_L and t_R be the left and right child of r_W respectively. We recall that we have stored $\mathcal{R}_{i_{n/4}}(X_2)$ in t_L and $\mathcal{R}_{i_{3n/4}}(X_1)$ in t_R. If τ_i goes to t_R, we do nothing in r_W and go to t_R (because $\mathcal{E}_i^\omega$ is a sub-sequence of $\mathcal{E}_{i_{n/2}}^\omega$ according to Proposition 4.6). Otherwise, we process r_W as follows: for each $\omega \in \mathcal{W}^1$, consider an intersection point $\mathfrak{L}_{i_{n/2}j'_1}$ and $\mathfrak{L}_{i_{n/2}j'_2}$ in $\mathcal{LE}_{i_{n/2}}^\omega$. Because $\mathcal{E}_{i_{n/2}}^\omega$ is a sub-sequence of $\mathcal{E}_i^\omega$ (Proposition 4.6), by examining the x-coordinate of the intersection point of $\mathfrak{L}_{ij'_1}$ and $\mathfrak{L}_{ij'_2}$ against x_0, either $\{\mathfrak{L}_{ij'} : j \in \mathcal{E}_{i_{n/2}}^\omega \text{ and } j' \leq j_1\}$ or $\{\mathfrak{L}_{ij'} : j \in \mathcal{E}_{i_{n/2}}^\omega \text{ and } j' \geq j_2\}$ can not be the line in $\mathcal{LE}_i^\omega$ at point x_0. So, by doing binary search on the intersection points of $\mathcal{LE}_{i_{n/2}}^\omega$, we can get the line that might be the line in $\mathcal{LE}_i^\omega$ at x_0. We store this line after processing r_W.

We continue processing the nodes of τ_i based on when each internal node in the path has a right or a left child and keep $O(\log |\nu|)$ (the size of $\mathcal{W}^1$) lines after processing each internal node in τ_i. Finally, we pick the minimum value of these lines at x_0 and compare it with the values of the lower-envelope stored in $\{\mathcal{LE}_i^\omega : \omega \in \mathcal{W}^1\}$ at x_0 and pick the minimum one which is $\mu_i^1(\nu)$.

According to the above discussion, given a query vertex v_i, first, we need to decompose the vertices $\{v_{j'} : i < j' \leq j\}$ into the union the spans of the vertices in $\mathcal{V}_{i,j} \subseteq \mathcal{T}$. $\mathcal{V}_{i,j}$ has $O(\log n)$ vertices and can be computed in $O(\log n)$ time. Next, for each $\nu \in \mathcal{V}_{i,j}$, we need to compute and store its $\mathcal{W}^1$ which costs $O(\log |\nu|)$. Also, for each $\omega \in \mathcal{W}^1$, we have $\mathbb{T}_W$ which has $O(\log n)$ height and in each node of it we do binary search on the vertices of its lower-envelope. So, the query time complexity is $O(\log^4 n)$. If we look closely at this process, an improvement can be applied as follows: for a fixed ν, consider three nodes ω_p and its children ω_L and ω_R in its recursion trees. Now, $\mathbb{T}_W$ is the same for all these nodes. Thus, for a node $\tau \in \mathbb{T}_W$ with corresponding weight w_t, $\mathcal{LE}_t^{\omega_p}$ is obtained by merging $\mathcal{LE}_t^{\omega_L}$

and $\mathcal{LE}_t^{\omega_R}$. Note that because all the lines in the envelopes has positive slopes, $\mathcal{LE}_t^{\omega_L}$ and $\mathcal{LE}_t^{\omega_R}$ are increasing functions and intersect each other in at most one point. So, if we do binary search on the vertices of $\mathcal{LE}_t^{\omega_L}$ and $\mathcal{LE}_t^{\omega_R}$, when processing ω_p in τ, we only need to test the intersection of $\mathcal{LE}_t^{\omega_L}$ and $\mathcal{LE}_t^{\omega_R}$ against x_0. This removes an $O(\log n)$ factor from the query time complexity. So, we can have the following theorem:

Theorem 4.3. *By spending $O(n \log^3 n)$ time for preprocessing, we can build a data structure that enables us to compute $\bar{M}[i, j]$ for any given pair (i, j) in $O(\log^3 n)$ time.*

4.4.1 Proving the Sub-sequence Property

Let S be a sequence of points on the x-axis with x-coordinates $(\hat{x}_1, \ldots, \hat{x}_g)$ sorted increasingly such that $\forall j : 0 < \hat{x}_j < \delta$ and $0 < \beta < \pi/2$ be a fixed given angle. Also, let $L_S^- = (\ell_1^-, \ldots, \ell_g^-)$ be the sequence of half-lines from the origin with negative slopes such that ℓ_j^- corresponds to $\hat{x}_j$ $(1 \leq j \leq g)$. So, there is a correspondence between the points in S and the half-lines in L_S^- (see Figure 4.8). Similarly, let $L_S^+ = (\ell_1^+, \ldots, \ell_g^+)$ be the sequence of half-lines where ℓ_j^+ starts from $\hat{x}_j$ and makes angle β with the negative direction of the x-axis. Again, there is a correspondence between the half-lines in L_S^+ and the points in S. Let us denote the intersection point of ℓ_j^+ and ℓ_j^- by ρ_j^β. We call $\hat{x}_j$, the *generating point* of ρ_j^β. Finally, we consider $I_S^\beta = \{\rho_1^\beta, \ldots, \rho_g^\beta\}$ and denote the point in I_S^β with minimum D-coordinate by m_S^β. In Figure 4.8, $S = (e_{ij_1}, e_{ij_2}, e_{ij_3})$ and m_S^β is generated by e_{ij_1}.

Consider the points in S as a set of objects initially located at $(\hat{x}_1, \ldots, \hat{x}_g)$ such that each object can move forward and backward on the x-axis. Without any confusion we refer both the locations and moving objects by *points* based on the context. We can see that if we slide the points in S to the left (resp. right), the points in I_S^β moves downward (resp. upward). So, if we consider the x-axis as the *time-axis*, we can talk about the location of S at time t by which we mean the set of points obtained by shifting S such that the x-coordinate of $\hat{x}_1$ becomes t. Let us denote the set of points in S, the intersection points in I_S^β and the point in I_S^β with minimum D-coordinate at time t by $S(t)$, $I_S^\beta(t)$ and $m_S^\beta(t)$ respectively. When t varies from zero to δ, the generating point of $m_S^\beta(t)$ might change. We call the times for which the generating point of $m_S^\beta(t)$ changes the *event times* and denote the sequence of generating points of $m_S^\beta(t)$ (in order of time) by $\epsilon_\beta(S)$. We can see that if we consider the D-coordinate of $\rho_j^\beta(t)$ $1 \leq j \leq g$ as a function of t, this function is linear (as in Equation 4.11) and so computing $\epsilon_\beta(S)$ is equivalent to computing the lower-envelope of a set of lines which can be done in $O(g \log g)$ time using the divide-and-conquer schema [63] (as we discussed in the previous subsection).

Let $(\ell_{i_1}^-, \ldots, \ell_{i_g}^-)$ be L_S^- sorted decreasingly according to their slopes magnitudes. Also, let $\Pi_S^\beta = (p_1^\beta, \ldots, p_g^\beta)$ be the sequence of points generated by S such that p_j lies on $\ell_{i_j}^-$ (and so, $\Pi_S^\beta(t) = (p_1^\beta(t), \ldots, p_g^\beta(t))$ is the location of these points at time t with respect to β). Let us denote the D-coordinate of $p_j^\beta(t)$ by $D(p_j^\beta(t))$. Then,

Observation 4.7. *If for two indices* $1 \leq j_1 < j_2 \leq g$*,* $D(p^{\beta}_{j_1}(0)) > D(p^{\beta}_{j_2}(0))$ *then* $\hat{x}_{i_{j_1}}$ *can never appear on* $\epsilon_{\beta}(S)$*.*

Therefore, if we repeatedly apply Observation 4.7 to remove the points in S not appearing in $\epsilon_{\beta}(S)$, we would end up having a sequence $S^{(1)} \subseteq S$ such that $\epsilon_{\beta}(S) = \epsilon_{\beta}(S^{(1)})$. We call this process *the first round of pruning* of the points in S. Note that the D-coordinates of the points in $\Pi^{\beta}_{S^{(1)}}(0)$ are sorted increasingly. Suppose that for two indices $j_1 < j_2$, $D(p^{\beta}_{j_1}(0)) < D(p^{\beta}_{j_2}(0))$, we define $t^{\beta}_{j_1 j_2}$ as the time for which,

$$D(p^{\beta}_{j_1}(t^{\beta}_{j_1 j_2})) = D(p^{\beta}_{j_2}(t^{\beta}_{j_1 j_2})) \tag{4.14}$$

Note that because $\ell^{-}_{i_{j_1}}$ has a greater slope magnitude than $\ell^{-}_{i_{j_2}}$, $t^{\beta}_{j_1 j_2}$ exists.

Observation 4.8. *If for three indices* $j_1 < j_2 < j_3$*,*

$$D(p^{\beta}_{j_1}(0)) < D(p^{\beta}_{j_2}(0)) < D(p^{\beta}_{j_2}(0)) \tag{4.15}$$

and $D(p^{\beta}_{j_1}(t^{\beta}_{j_2 j_3})) < D(p^{\beta}_{j_2}(t^{\beta}_{j_2 j_3}))$ *then* $\hat{x}_{i_{j_2}}$ *can never appear in* $\epsilon_{\beta}(S)$*.*

Again if we repeatedly apply Observation 4.8 to the points in $S^{(1)}$ until no such triple found, we get a sequence $S^{(2)} \subseteq S^{(1)}$ such that $\epsilon_{\beta}(S) = \epsilon_{\beta}(S^{(2)})$. Indeed, any point in $S^{(2)}$ should appear in $\epsilon_{\beta}(S)$ and the order is according to their corresponding points in Π_S. We call this process the *second round of pruning* of S. Based on the above discussion, obtaining $\epsilon_{\beta}(S)$ is equivalent to perform two rounds of pruning.

The sub-sequence property. *For any two angles* $0 < \beta_1 < \beta_2 < \pi/2$*,* $\epsilon_{\beta_2}(S)$ *is a sub-sequence of* $\epsilon_{\beta_1}(S)$*.*

Proof. We show that if a point $\hat{x}_{j_{j'}} \in S$ is eliminated with respect to β_1, then it has to be eliminated with respect to β_2. We have two cases:

Case 1: $\hat{x}_{i_{j'}}$ is eliminated in round one with respect to β_1. In this case, there is an index $j_1 > j'$ such that $D(p^{\beta_1}_{j_1}(0)) < D(p^{\beta_1}_{j'}(0))$. Now, because the magnitude of the slope of $\ell^{-}_{i_{j'}}$ is bigger than $\ell^{-}_{i_{j_1}}$, $D(p^{\beta_2}_{j_1}(0)) < D(p^{\beta_2}_{j'}(0))$ and so, $\hat{x}_{i_{j'}}$ should be eliminated in the round one pruning with respect to β_2 (see Figure 4.9 as an example of such situation).

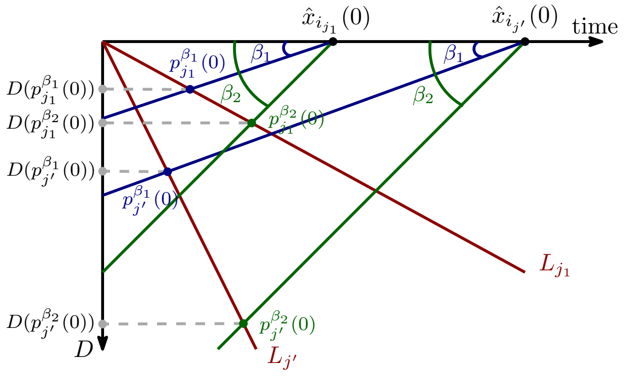


Figure 4.9: Proof of case 1.

Case 2: $\hat{x}_{i_{j'}}$ is eliminated in round two with respect to β_1. So, there are two indices $j_2 < j' < j_1$ such that:

$$D(p^{\beta_1}_{j_2}(0)) < D(p^{\beta_1}_{j'}(0)) < D(p^{\beta_1}_{j_1}(0)) \tag{4.16}$$

And,

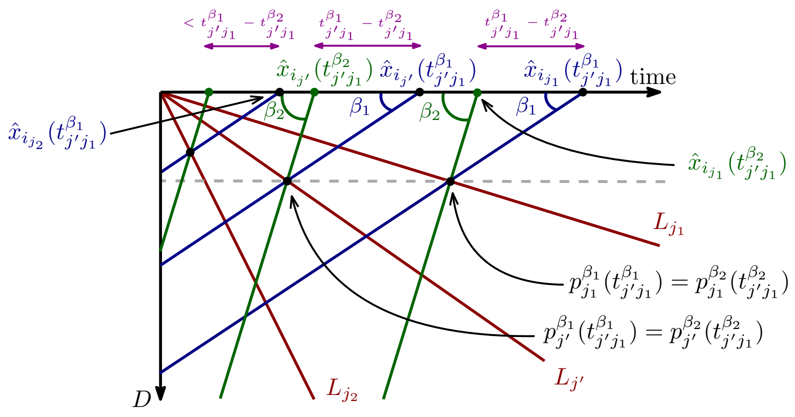
$$D(p^{\beta_1}_{j_2}(t^{\beta_1}_{j'j_1})) < D(p^{\beta_1}_{j'}(t^{\beta_1}_{j'j_1})) = D(p^{\beta_1}_{j_1}(t^{\beta_1}_{j'j_1})) \tag{4.17}$$

If $\hat{x}_{i_{j'}}$ is eliminated in round one with respect to β_2 we are done. Otherwise, we have:

$$D(p^{\beta_2}_{j'}(0)) < D(p^{\beta_2}_{j_1}(0)) \tag{4.18}$$

It is easy to see $t^{\beta_2}_{j'j_1} < t^{\beta_1}_{j'j_1}$ and,

$$D(p^{\beta_2}_{j'}(t^{\beta_2}_{j'j_1})) = D(p^{\beta_2}_{j_1}(t^{\beta_2}_{j'j_1})) = D(p^{\beta_1}_{j'}(t^{\beta_1}_{j'j_1})) \tag{4.19}$$



$< t^{\beta_1}_{j'j_1} - t^{\beta_2}_{j'j_1}$
$t^{\beta_1}_{j'j_1} - t^{\beta_2}_{j'j_1}$
$t^{\beta_1}_{j'j_1} - t^{\beta_2}_{j'j_1}$
$\hat{x}_{i_{j'}}(t^{\beta_2}_{j'j_1})$
$\hat{x}_{i_{j'}}(t^{\beta_1}_{j'j_1})$
$\hat{x}_{i_{j_1}}(t^{\beta_1}_{j'j_1})$
time
$\hat{x}_{i_{j_2}}(t^{\beta_1}_{j'j_1})$
β_2
β_1
β_2
β_1
$\hat{x}_{i_{j_1}}(t^{\beta_2}_{j'j_1})$
L_{j_1}
$p^{\beta_1}_{j_1}(t^{\beta_1}_{j'j_1}) = p^{\beta_2}_{j_1}(t^{\beta_2}_{j'j_1})$
$p^{\beta_1}_{j'}(t^{\beta_1}_{j'j_1}) = p^{\beta_2}_{j'}(t^{\beta_2}_{j'j_1})$
$L_{j'}$
L_{j_2}
D

Figure 4.10: Proof of case 2.

This implies that $D(p_{j_2}^{\beta_2}(t_{j'j_1}^{\beta_2})) < D(p_{j_2}^{\beta_1}(t_{j'j_1}^{\beta_1}))$ which means that $\hat{x}_{i_{j'}}$ should be eliminated with respect to β_2 (see Figure 4.10). $\square$

Chapter 5

An Efficient Algorithm for the Proximity Connected 2-Center Problem

Given a set P of n points in the plane, the k-center problem is to find k congruent disks of minimum possible radius such that their union covers all the points in P. The 2-center problem is a special case of the k-center problem that has been extensively studied in the recent past [22, 59, 65]. In this chapter[1] , we consider a generalized version of the 2-center problem called *proximity connected* 2-center problem (PCTCP). In this problem, we are also given a parameter $\delta \geq 0$ and we have the additional constraint that the distance between the centers of the disks should be at most δ. Note that when $\delta = 0$, the PCTCP is reduced to the 1-center (minimum enclosing disk) problem and when δ tends to infinity, it is reduced to the 2-center problem. The PCTCP first appeared in the context of wireless networks in 1992 [36], but obtaining a nontrivial deterministic algorithm for the problem remained open. In this chapter, we resolve this open problem by providing a deterministic $O(n^2 \log n)$ time algorithm for the problem.

[1]A preliminary version of this chapter appeared in the proceedings of the *33rd International Workshop on Combinatorial Algorithms, IWOCA 2022. [9]*

5.1 Background and Previous Works

The k-center problem in the plane is a fundamental facility-location problem in which we are given a set of n demand points $\mathbb{P}$ and we are going to find a set S of k center points such that $cost(S)$ defined as $\max_{p \in \mathbb{P}} \min_{s \in S} d(p, s)$ is minimized ($d(p, s)$ is the Euclidean distance between p and s). The k-center problem is known to be NP-hard [3]. However, there is a simple greedy 2-approximation algorithm for the problem which can not be improved unless $P = NP$ [3]. So, the studies on the problem went in the direction of obtaining polynomial time algorithms where k is not considered as a part of the problem input. As an example, in 2002, Agarwal and Procopiuc [1] gave a $n^{O(\sqrt{k})}$ time algorithm to solve the k-center problem. Solving the problem for specific values of k like $k = 1$ and $k = 2$ received attention due to the geometric properties that can be applied to solve these problems efficiently. The 1-center problem is indeed equivalent to the problem of covering P with a disk with minimum area. This problem is also called the *minimum enclosing disk (MED)* problem. In 1983, Megiddo [51] used the prune and search technique to give an optimal linear time algorithm to solve the MED problem.

For $k = 2$, Drenzer [25] gave the first nontrivial algorithm for the problem with $O(n^3)$ time complexity. Later in 1994, Agarwal and Sharir [2] improved the time complexity for the problem to $O(n^2 \log^3 n)$. In 1996, Eppstein [27] gave a randomized algorithm for the problem with $O(n \log^2 n)$ expected running time. In 1997, Katz and Sharir [43] proposed the novel expander-based parametric search technique and showed that applying it to the 2-center problem using the $O(n^2)$ time feasibility test of Hershberger [33], gives an $O(n^2 \log^3 n)$ time algorithm for the problem. Later in the year, Sharir [59] designed an $O(n \log^3 n)$ time algorithm for the decision version of the 2-center problem using the breakthrough observation of breaking the problem into three separate cases (far distant, distant and nearby cases). Next, he parallelized the decision algorithm and put it into the Megiddo's parametric search schema [52] to obtain an $O(n \log^9 n)$ time algorithm. Soon, it turned out that solving the problem in the nearby case is the bottleneck to reduce the time complexity. Later, Sharir's running time was improved by Chan [17] and Wang [65] to $O(n \log^2 n \log^2 \log n)$ and $O(n \log^2 n)$ respectively. Very recently, Choi and Ahn [22] (independently Cho and Oh [21]) obtained an $O(n \log n)$ time algorithm for the nearby case which led to an optimal $O(n \log n)$ time algorithm for the 2-center problem.

Note that in the PCkCP in the plane, when δ tends to zero (resp. infinity), the problem reduces to the 1-center (resp. k-center) problem. Also, when δ tends to zero and k tends to infinity the problem becomes the Euclidean Steiner tree problem (connecting the points of P by lines of minimum total length in such a way that any two points can be connected by the lines). This is because in this configuration, the centers should be placed along the lines of the minimum Steiner tree in order to minimize the cost. The Euclidean Steiner tree problem is also NP-hard but it has a PTAS approximation algorithm [6].

In practice, the parameter δ usually specifies the range for which one center can communicate with other centers. So, when S satisfies the PCC, any pair of centers can communicate with each other via the other centers. The proximity connected 2-center (PCTC) problem first emerged in the works of Huang [36] in 1992 while he was studying packet radio networks. In the network terminology, the PCTCP is the problem of locating two wireless devices as close as possible to the demand points P such that they can send/receive messages between each other. He originally gave an $O(n^5)$ time algorithm for the 2-center problem having proximity constraints between their centers. Later in 2003, Huang *et al.* [38] studied a very close problem to the PCTCP called α*-connected* 2-center problem. In this problem, instead of δ, a parameter $0 \leq \alpha \leq 1$ is given and the distance between the center of the disks should be at most $2(1-\alpha)r$ where r is the radius of the disks. They gave an $O(n^2 \log^2 n)$ time algorithm for the decision version (given an r whether it is possible to cover the points with two disks of radius r satisfying the desired conditions) of the problem. Note that this problem is a special case of the PCTC decision problem where $\delta = 2(1-\alpha)r$. Later in 2006, they gave a randomized algorithm with the same $O(n^2 \log^2 n)$ expected running time to solve the α*-connected* 2-center problem [37]. In this chapter, we consider the PCTCP and propose a deterministic $O(n^2 \log n)$ time algorithm for it.

Here, we need to mention that although we use Sharir's observation [59] of breaking the problem into three different cases (far distant, distant and nearby), the reason we can't get a sub-quadratic algorithm like [59, 17, 65, 22] is that the PCTCP is structurally different from the 2-center problem. In the 2-center problem, the optimal cost is determined by at most three points of P [59] while in the PCTCP the cost may need to be determined by more than three points (because of the PCC). This means that our search space has a higher dimension than the search space of the 2-center problem. Also, all the sub-quadratic algorithms for the 2-center problem use Megiddo's [52] or Cole's [23] parametric search schema to reduce the time complexity which makes the resulting algorithm impractical [2] while our algorithm exploits the geometric properties of the problem which make it straightforward to be implemented using standard data structures in computational geometry.

A *solution* for a given PCTCP instance is defined as a pair of disks whose centers satisfy the PCC and their union covers P. We call a disk with the larger (or equal) radius the *determining disk* of the solution and its radius the *cost* of the solution. An *optimal solution* is a solution with minimum cost among the set of all solutions for the problem. Note that there might be an infinite number of optimal solutions with different pairs of radii because we have freedom on the smaller disk. So, we try to find an optimal solution such that the radius of its smaller disk is minimum among all optimal solutions. We call such a solution a *best optimal solution (BOS)* for the problem. Therefore, if the problem has more than one BOS, they would have the same pair of radii. We can also compare two solutions S_1 and S_2 as follows: we say that S_1 is a better solution than S_2 if $cost(S_1) < cost(S_2)$ and if $cost(S_1) = cost(S_2)$, the radius of the non-determining disk of S_1 is smaller than the radius of the non-determining disk of S_2. In this chapter, our algorithm not only gives us an optimal solution but it computes a BOS for the problem.

5.2 Preliminaries and Definitions

Let (P, δ) be the given PCTCP instance where P is a set of n demand points in the plane and δ is a given non-negative number. We assume that the points are in general position, by which we mean no four points of P lie on a circle. Let (P_1, P_2) be a partition of P obtained by dividing the plane by a line or two half-lines from a point (henceforth, when we use the term *partition of the plane*, we mean a partition that satisfies this condition). We say that a pair of disks (D_1, D_2) with centers (c_1, c_2) respectively is a *solution for the partition* if D_1 covers P_1, D_2 covers P_2 and $d(c_1, c_2) \leq \delta$. *Optimal* and *best optimal solutions (BOSs)* for the partition are defined similarly. Let (D_1^*, D_2^*) be a BOS for the partition with centers (c_1^*, c_2^*) respectively. We say that a point $p \in P_1$ is a *dominating point* of D_1^* if (D_1^*, D_2^*) is not a BOS for the partition $(P_1 \setminus p, P_2)$. The dominating points of D_2^* are defined similarly. Note that the dominating points of D_1^* and D_2^* are on their boundaries. By assuming that the points are in general position, if D_1^* (resp. D_2^*) is the MED of P_1 (resp. P_2), its dominating points are either three points on the boundary such that their induced triangle contains c_1^* (resp. c_2^*) or two points on the boundary such that their connecting segment passes through c_1^* (resp. c_2^*). In order to simplify the presentation of our algorithm, in the latter case, we consider one of the dominating points as two infinitely close points and so, if D_1^* or D_2^* is the MED of their corresponding points, we assume that it has exactly three dominating points. Similarly, if D_1^* (resp. D_2^*) is not the MED of P_1 (resp. P_2), in the case that it only has one dominating point, we can consider it as two infinitely close points. But, if it has three points on its boundary such that their induced triangle does not contain c_1^*, we might have no dominating point for D_1^*. We can assume that such a situation never happens by slightly perturbing the points. So, henceforth, if D_1^* (resp. D_2^*) is not a MED, we assume that it has exactly two dominating points.

We call the problem of computing a BOS for a given partition (P_1, P_2) the *restricted PCTCP*. In the next section, we show that how we can solve the restricted PCTC using the *intersection hulls* and the *farthest-point Voronoi diagram*s of P_1 and P_2 (the intersection hull of a set of points with respect to some radius r is defined as the intersection of all disks of radius r around the points of the set). Before explaining our algorithm, we review farthest-point Voronoi diagrams, intersection hulls and their properties.

5.3 A Review on Farthest-point Voronoi Diagrams and their Properties.

For a given set of points $A = \{a_i : 1 \leq i \leq m\}$, the farthest-point Voronoi diagram of A denoted by $\mathcal{F}(A)$ is the partition of the plane into a set of disjoint-interior *cells* $\{C(a_i) : a_i \in A\}$ such that $C(a_i)$ is the set of points in the plane for which no point of A is farther from them than a_i. We say a_i is the farthest point of $C(a_i)$ and call it the *site* of the cell $C(a_i)$. For each point $x \in C(a_i)$, the *weight* of x is defined as its distance to a_i and we denote it by $w(x)$. Note that $disk(x, w(x))$ (the disk with center x and radius $w(x)$) covers all the points in A. It is easy to see that $\partial\mathcal{F}(A)$ (the set of boundaries between cells of $\mathcal{F}(A)$) consists of a set of line segments, half-lines, or it is just a line which we call the *edges* of $\partial\mathcal{F}(A)$. For each $e \in \partial\mathcal{F}(A)$, there exists a unique pair (a_i, a_j) of points in A such that for any point x on the interior of e, we have $d(x, a_i) = d(x, a_j)$ and no point of A is farther than these points from x. We call a_i and a_j the *generators* of e. Note that e lies on the perpendicular bisector of the $seg(a_i a_j)$ (the line segment connecting a_i and a_j). If v is an endpoint of e (in this case we call v a vertex), v has three points in A all farthest from v (and no more because of our assumption that no four points are on a circle). We also call these points the generators of v. Given a start and an endpoint on $\partial\mathcal{F}(A)$, its corresponding path on $\partial\mathcal{F}(A)$ is the portion of $\partial\mathcal{F}(A)$ between two points directed from the start point toward the endpoint. Note that such a path is unique otherwise a cell of the $\mathcal{F}(A)$ would be bounded which is not possible [63].

Observation 5.1. *The center of the minimum enclosing disk of A is the minimum weight point on $\partial\mathcal{F}(A)$.*

We call the center of the minimum enclosing disk of A the *root* of $\partial\mathcal{F}(A)$ and it is unique.

Proposition 5.1. *Let $p = [r, b]$ be a path on $\partial\mathcal{F}(A)$ where r is its root. Then, the weight of the points on p change monotonically increasing from r to b.*

Proof. First, note that the root is unique. So, for each edge e in the path with generators a_i and a_j, the midpoint of $seg(a_i a_j)$ can not lie on the interior of e. On the other hand, e is a subset of the perpendicular bisector of $seg(a_i a_j)$ and the weight of each point on e is its distance to a_i (which is the same as its distance to a_j). So, the weight on e should change monotonically as we move from one of its endpoints to another. Now, suppose that the proposition is not true. Then, there must be a first edge ht (direction is along p) on the path such that as we move from h to t, the weight decreases. This means that the vertex h should have local maximum weight on the path which is contradiction because if we slightly move from h, the distance with one of its generators is increased. $\square$

The intersection hull of A at radius r is defined as $\cap_{a\in A} disk(a,r)$. Let us denote the intersection hull of A at radius r by $H_A(r)$. We can easily see that $H_A(r)$ is composed of a set of circle arcs with radius r with endpoints at the edges of $\partial\mathcal{F}(A)$. So, if we start from the leftmost endpoint of $H(r)$ and traverse its arcs clockwise, we obtain a unique sequence of arcs. We refer to this sequence as the *arc-sequence* of $H_A(r)$ and denote it by $Seq(H(r))$. Let x be a vertex of $H_A(r)$. Suppose that x lies on an edge e of $\mathcal{F}(A)$. We call the half-line from x along e that does not intersect the interior of $H_A(r)$ the *arm* of $H_A(r)$ from x. See Firgure 5.1.

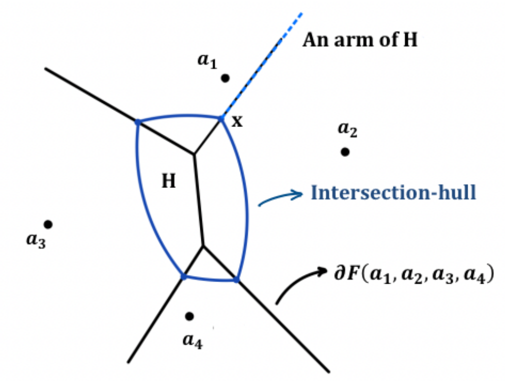


Figure 5.1: The farthest-point Voronoi diagram of a set of points and its intersection hull at some radius r.

5.4 Computing a BOS for a Partition

Let (P_1, P_2) be a given partition. First, we compute the minimum enclosing disks D_1^{**} and D_2^{**} for P_1 and P_2 respectively. This can be done in linear time due to Megiddo's algorithm [51]. Also let c_1^{**} and c_2^{**} be the centers of D_1^{**} and D_2^{**} respectively. Now, if $d(c_1^{**}, c_2^{**})$ (the distance between c_1^{**} and c_2^{**}) is at most D, then we are done and (D_1^{**}, D_2^{**}) is a BOS for the partition. Otherwise, we have the following proposition:

Proposition 5.2. *If $d(c_1^{**}, c_2^{**}) > \delta$ then for any BOS (D_1^*, D_2^*) for the partition, we have $d(c_1^*, c_2^*) = \delta$.*

Proof. We proceed by contradiction. Suppose that for an optimal solution (D_1^*, D_2^*), $d(c_1^*, c_2^*) < \delta$. So, at least one of the centers for example c_1^* is different from c_1^{**} and lies inside the region between the two perpendicular lines from c_1^{**} and c_2^{**} on $line(c_1^{**}, c_2^{**})$ (the line passing c_1^{**} and c_2^{**}). This is because if both c_1^* and c_2^* are outside this region, the distance between them can't be less than δ. If c_1^* is not on $\partial\mathcal{F}(P_1)$, then we can slightly move c_1^* toward its farthest point, reducing the radius of D_1^* while not violating the PCC which contradicts best optimality. If c_1^* is on $\partial\mathcal{F}(P_1)$, by Proposition 5.1, any point on the interior of the path from c_1^{**} to c_1^* on $\partial\mathcal{F}(P_1)$, covers P_1 with radius smaller than $r(D_1^*)$. Since $d(c_1^*, c_2^*) < \delta$, any point on this path sufficiently close to c_1^* will not violate the PCC. This contradicts the fact that (D_1^*, D_2^*) is a BOS. □

Let $H_1(r)$ and $H_2(r)$ be the intersection hulls of P_1 and P_2 with radius r. Note that the smallest radii for which the intersection hulls of P_1 and P_2 are nonempty are $r(D_1^{**})$ and $r(D_2^{**})$ respectively. Let us denote them by r_1^0 and r_2^0 (in fact $H_1(r_1^0) = c_1^{**}$ and $H_2(r_2^0) = c_2^{**}$). If $r \geq r_1^0$, for any point $q \in H_1(r)$, $disk(q, r)$ (the disk with center q and radius r) covers P_1 (we have a similar statement for P_2 and r_2^0). Based on this property, the problem turns to find a best optimal pair of radii (r_1^*, r_2^*) (the bigger radius is minimum and the smaller radius is minimum among all such pairs) such that the distance between $H_1(r_1^*)$ and $H_2(r_2^*)$ is exactly δ. Also, we call the maximum radius of optimal pair(s) the *optimal cost* and denote it by r^*. The idea to find a best optimal pair is first try to find the optimal cost r^* and then, fix one of the intersection hulls at radius r^* and find minimum possible radius for the other hull. So here, we focus on finding the optimal cost.

In order to find the optimal cost, we impose the constraint that the disks are congruent (radii of both hulls is equal). Imposing this constraint makes the problem easier while it does not change the optimal cost. In order to solve the problem for congruent disks, we can grow the intersection hulls of the points at each part of the partition to see when the distance between them becomes δ. We first build $\mathcal{F}(P_1)$ and $\mathcal{F}(P_2)$ which can be done in $O(n \log n)$ time. In order to prevent structural changes when we grow the intersection hulls, we apply a binary search (repeatedly find the median and discard half of the values) on the set of weights of the farthest-point Voronoi diagrams of both sides to obtain an interval $I^* = (i_0, i_1)$ such that $r^* \in I^*$ and for each vertex v of the diagrams, $w(v) \notin I^*$ (the weight of a point x in a cell of farthest-point Voronoi diagram denoted by $w(x)$ is the distance between x and the site of the cell containing it). At each step of the binary search, when we test a weight w, we use the algorithm of [20] to compute the distance between the two intersection hulls at radius w to see whether their distance is smaller, equal or greater than δ. Note that because the intersection hulls are convex, this step can be done in $O(\log n)$ time according to [20] (we don't need to explicitely build the intersection hulls because their vertices are along the edges of the farthest-point Voronoi diagrams).

Observation 5.2. *For any index i and any $r \in I^*$, the endpoints of the i^{th}-element of $Seq(H_1(r))$ and $Seq(H_1(i_0))$ (resp. $Seq(H_2(r))$ and $Seq(H_2(i_0))$) lie on same arms of $H_1(i_0)$ (resp. $H_2(i_0)$).*

In other words, when r varies from i_0 to i_1, no arc in intersection hulls will be emerged or vanished. Let us denote the arms of $H_1(i_0)$ by A_1 and call the partition induced by $H_1(i_0) \cup A_1$ the A_1-*partition* of the plane. Now, we discuss how to find the optimal cost for the partition. Suppose that we have not found r^* during the binary search (otherwise we are

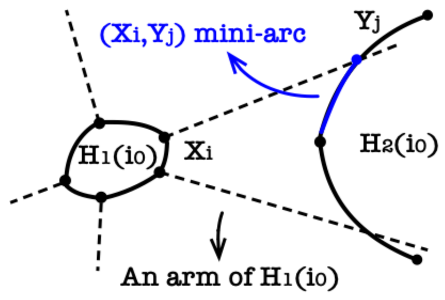


Figure 5.2: A mini-arc for two intersection hulls

done). So, $d(H_1(i_0), H_2(i_0)) > \delta$ and $d(H_1(i_1), H_2(i_1)) < \delta$. Let $Seq(H_1(i_0)) = (X_1, \ldots, X_u)$ and $Seq(H_2(i_0)) = (Y_1, \ldots, Y_{u'})$. We also label each region of the A_1-partition bounded by two neighbour arms by the name of the arc it contains. Each arm in A_1 can intersect $\partial H_2(i_0)$ in at most two points. Consider an arm in A_1 with endpoint a and an intersection point x with $\partial H_2(i_0)$. We call this intersection point a *first intersection point* if ax does not intersect the interior of $H_2(i_0)$.

Let B be the set of all first intersection points of the arms in A_1 and $\partial H_2(i_0)$. Note that $H_2(i_0)$ is convex and the arms around $H_1(i_0)$ diverges from each other. Also, we already have the order of the arms around $H_1(i_0)$ induced by $\mathcal{F}(P_1)$. In order to compute B, consider the counter-clockwise order on the arms of A_1 starting from the arm with the lowest slope (can be negative) and compute their first intersection points with $H_2(i_0)$ in order. An important point here is that if a vertex of $H_2(i_0)$ lies on the right side of an arm $\vec{a}$ (the direction is from its endpoint), it will be on the right side of any arm after $\vec{a}$. This property implies that the cost of computing B is linear (see Figure 5.2).

Now, consider the partition induced by B and the vertices of $\partial H_2(i_0)$ on $\partial H_2(i_0)$. We call each region of this partition a *mini-arc* on $\partial H_2(i_0)$ (see Figure 5.2). We assign a label (X_i, Y_j) to each mini-arc of $\partial H_2(i_0)$ where X_i is the label of the mini-arc in the A_1-partition and Y_j is the label the arc of $H_2(i_0)$ containing the mini-arc. Note that the number of such (X_i, Y_j) labels are linear (because we have a linear number of mini-arcs).

Proposition 5.3. *The labels of the two arcs containing two closest points between $H_1(r^*)$ and $H_2(r^*)$ corresponds to the label of one of the mini-arcs.*

Proof. Let p_1 and p_2 be two points on $H_1(r^*)$ and $H_2(r^*)$ respectively with distance δ. First, we observe that the perpendicular lines on $line(p_1, p_2)$ from p_1 and p_2 should not intersect the interior of $H_1(r^*)$ and $H_2(r^*)$ (otherwise it contradicts the optimallity of r^*). Suppose that p_1 and p_2 are lie on two arcs X_i and Y_j respectively (we consider the names of the arcs in $H_1(r^*)$ and $H_2(r^*)$ the same as the label of their corresponding regions in

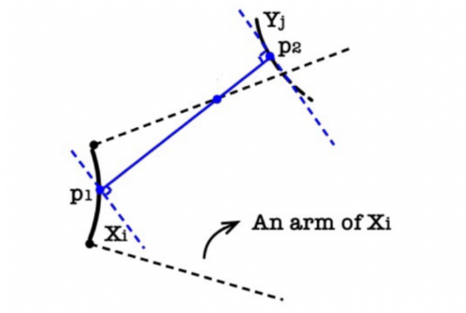


Figure 5.3: A connecting line segment of two points from two arcs that do not make a mini-arc intersects an arm.

$H_1(i_0)$ and $H_2(i_0)$ respectively). If (X_i, Y_j) is not a label of a mini-arc, p_1p_2 should intersect an arm of X_i. But in this situation, the perpendicular line on $line(p_1, p_2)$ from p_1 should intersect the interior of $H_1(r^*)$ (because of convexity of $H_1(r^*)$) which is contradiction. See Figure 5.3. $\square$

According to Proposition 5.3, we can compute r^* as follows: we consider each label (X_i, Y_j) of the mini-arcs and compute the value $r_{i,j}$ for which the distance between the two arcs X_i and Y_j becomes δ as they propagate between their bounding arms. Note that computing each $r_{i,j}$ can be done in a constant time. Now, r^* is the minimum value among all $r_{i,j}s$.

The next step is obtaining a BOS for the partition after computing r^*. As we said earlier, in order to do this, we first assume that the determining disk covers P_1 and obtain the minimum possible radius r'_1 for H_2 which makes the distance between $H_1(r^*)$ and $H_2(r'_1)$ exactly δ. This can be done in a similar way to how we obtained r^*. Then we obtain r'_2 similarly by assuming that the determining disk covers P_2. By comparing the results, we pick the one with smaller non-determining disk which is in fact a BOS for the partition. So, the total time we need to compute a BOS is $O(n \log n)$.

5.5 Obtaining a BOS for the PCTCP

We denote the optimal cost for the PCTCP by r^* and a BOS for the problem by (D_1^*, D_2^*) with centers (c_1^*, c_2^*) respectively. We can assume that c_1^* and c_2^* lie on the x-axis and c_1^* is on the left side of c_2^*. In [59], Sharir broke the 2-center decision problem (given a parameter r determine whether it is possible to cover the points with two disks of radius r) into three cases -far distant, distant and nearby- with respect to the given parameter r. He showed that providing separate algorithms for these cases will reduce the overall time complexity to solve the decision problem. Although our problem is an optimization problem and the parameter r^* is unknown, we will show that breaking the PCTCP into the same cases will simplify our algorithm and reduce the overall time complexity. So, our algorithm separately considers each of the following three assumptions about (D_1^*, D_2^*).

1. Nearby: $d(c_1^*, c_2^*) \leq r^*$.
2. Distant: $r^* < d(c_1^*, c_2^*) \leq 3r^*$.
3. Far distant: $d(c_1^*, c_2^*) > 3r^*$.

Denote the smallest cost we can get having the nearby, distant and far distant assumptions by r^{NA}, r^{DA} and r^{FA} respectively. We also use the same notation for a BOS and their corresponding centers having each assumption. So, we can obtain (D_1^*, D_2^*) by comparing (D_1^{NA}, D_2^{NA}), (D_1^{DA}, D_2^{DA}) and (D_1^{FA}, D_2^{FA}) (note that these solutions may not exist or satisfy their corresponding case conditions. For example, $d(c_1^{NA}, c_2^{NA})$ might be greater than r^{NA} but if (D_1^*, D_2^*) satisfies the nearby case, then $r^{NA} = r^*$ and (D_1^{NA}, D_2^{NA}) would be a BOS for the problem and we have $d(c_1^{NA}, c_2^{NA}) \leq r^{NA} = r^*$). Henceforth, while studing each of the cases, when we say BOS, we mean a best solution we can get having the corresponding case assumption. Given two points x and y in the plane, we denote the line passing from x and y by $line(x, y)$. The direction of this line is considered from x to y. Also, we denote the half-line from x passing y by $half\text{-}line(x, y)$ and the line segment with endpoints x and y by $seg(xy)$.

5.6 Computing a BOS in the Nearby Case

First, we can see that if $(D_1^*, D_2^*) \leq r^*$, then there is an optimal partition R^* (may not be unique) such that (D_1^*, D_2^*) is a BOS of R^*. In fact, such a partition can be obtained by considering a point in $D_1^* \cap D_2^*$ and two half-lines from it passing the intersection points of ∂D_1^* (boundary of D_1^*) and ∂D_2^*. In this section, when we say the dominating points of (D_1^*, D_2^*), we mean its dominating points with respect to R^*. Without loss of generality, we can assume that D_2^* is the determining disk. We first compute the *convex-hull*(P) and scale the problem such that it fits in a unit square (multiple both x and y coordinates of the points by the greatest constant such that the convex hull remains inside the square). This step can be done in $O(n \log n)$ time. Note that the scaling will not change the solutions.

Proposition 5.4. *If $(D_1^*, D_2^*) \leq r^*$, then the area of $D_1^* \cap D_2^*$ must be greater than a constant factor of the area of D_2^* (the determining disk).*

Proof. We proceed by contradiction. Suppose that such a factor does not exist. This means that we can build a problem instance such that it has a BOS (D_1^*, D_2^*) in which the radius of the non-determinig disk (D_1^*) becomes infinitely small (because of the nearby assumption and scaling). So, D_1^* should have at least one dominating point that is not covered by D_2^*. Because the radius of D_1^* is infinitely small, δ should tend to $radius(D_2^*)$

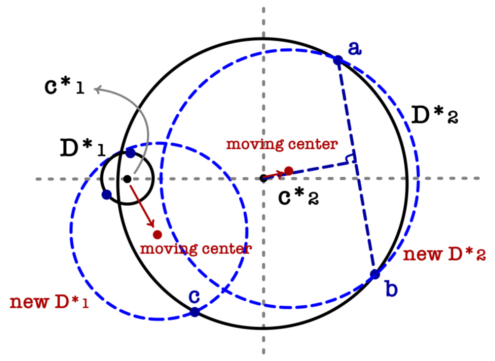


Figure 5.4: Enlarging the non-determining disk D_1^* to cover one of the dominating points of D_2^* and get a better solution.

(which tends to the radius of the MED of P). Now, D_2^* should have at least one dominating point (point c in Figure 5.4) with the x-coordinate less than or equal to c_2^* (otherwise, we can move both c_1^* and c_2^* to the right and reduce the radius of D_2^* which determines the cost). In this configuration, we can enlarge D_1^* by moving c_1^* toward this dominating point of D_2^* while satisfying the PCC in order to cover it and release it from D_2^* (D_1^* does not lose any of its own points and its radius still remains less than the radius of D_2^*). Now, we can reduce the radius of D_2^* which contradicts the optimallity of (D_1^*, D_2^*) (see Figure 5.4).□

Proposition 5.5. D_1^* *(similarly* D_2^**) should have a pair of dominating points such that:*

1. *They lie on different sides of* $line(c_1^*, c_2^*)$.
2. *Their connecting segment does not intersect* $seg(c_1^*c_2^*)$.

Proof. First, we prove that D_1^* should have a pair of dominating points each on the different sides of $line(c_1^*, c_2^*)$. Note that if D_1^* has three dominating points on one side of $line(c_1^*, c_2^*)$, their induced triangle can't cover c_1^* and based on our general assumption such situation can't happen. Now, suppose that D_1^* has two dominating points d_1 and d_2 both on a same side of $line(c_1^*, c_2^*)$. Because D_1^* is not MED (it has only two dominating points), the distance between c_1^* and c_2^* is exactly δ (otherwise, move c_1^* toward the dominating points to get better solution). Because d_1 and d_2 are on a same side of $line(c_1^*, c_2^*)$, the region $R := disk(d_1, r^*) \cap disk(d_2, r^*) \cap disk(c_2^*, \delta)$ is not empty. So, if we slightly move c_1^* into R, we would get a better solution which is contradiction (See Figure 5.5).

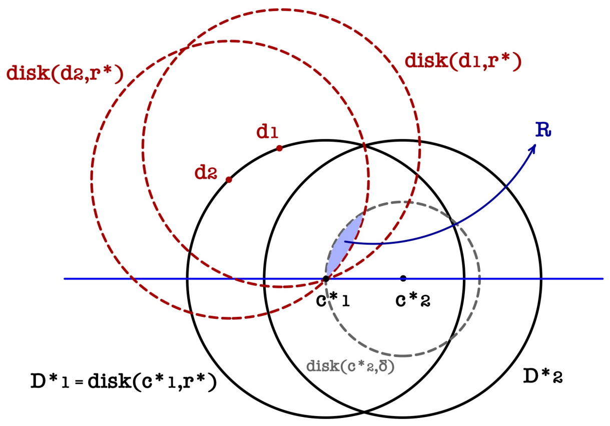


Figure 5.5: If d_1 and d_2 are on a same side of $line(c_+^{i,j}, c_-^{i,j})$, $(D_-^{i,j}, D_+^{i,j})$ can't be best optimal

For the second statement, again if D_1^* has three dominating points such that no pair of them intersect $seg(c_1^* c_2^*)$, their induced triangle can not contain c_1^* which is contradiction. Now, suppose that D_1^* has two dominating points d_1 and d_2 such that their connecting segment intersect $seg(c_1^* c_2^*)$. Because $seg(d_1 d_2)$ has non-empty intersection with the interior of $disk(c_2^*, \delta)$ and this disk is tangent to c_1^*, we can slightly move c_1^* toward the mid-point of $seg(d_1 d_2)$ while we are still inside $disk(c_2^*, \delta)$ to get a better solution. □

Considering the four dominating points in the above proposition, we can say that $D_1^* \cap D_2^*$ should cover at least a constant factor of the area of *convex-hull*(P). Furthermore, $D_1^* \cap D_2^* \cap$ *convex-hull*(P) is convex because it is the intersection of convex objects. So, we can build a constant size set of points $\mathcal{M}$ uniformly distributed on *convex-hull*(P) such that (assuming $d(c_1^*, c_2^*) \leq r^*$) for at least one point $\hat{m} \in \mathcal{M}$, $\hat{m} \in D_1^* \cap D_2^* \cap$ *convex-hull*(P). Because $\hat{m}$ is unknown, for each $m \in \mathcal{M}$, we build a BOS (D_1^m, D_2^m) assuming $m \in D_1^* \cap D_2^*$ and finally pick a best solution in $\{(D_1^m, D_2^m) \ : \ m \in \mathcal{M}\}$ and set it as (D_1^{NA}, D_2^{NA}). Based on this idea, we present our algorithm to find (D_1^m, D_2^m) for a given point $m \in$ *convex-hull*(P).

Let $\mathcal{X}$ be a set of 360 directed lines (each line has a positive direction) passing through m such that the angle between each directed line and its neighbour lines is $1°$. Now, there is a directed line in $\mathcal{X}$ such that its angel with $line(c_1^*, c_2^*)$ is at most $1°$ and c_1^* lies on the negative side of c_2^* on the line (note that D_2^* is the determining disk according to our assumption). We call this directed line the *correct directed line* which is unknown. So, we assume each line $l \in \mathcal{X}$ as the correct directed line and compute a BOS $(D_1^{m,l}, D_2^{m,l})$ having this assumption and finally pick the best one as (D_1^m, D_2^m).

So, assume that a directed line $l \in \mathcal{X}$ called the m-line is given. Here we explain how to compute $(D_1^{m,l}, D_2^{m,l})$. The m-line divides the points of P into two disjoint sets one on the right side and the other on the left side of the m-line. We sort these sets according to the polar angles of their points (from m) with respect to the positive direction of the m-line. These angles should lie between $-180°$ and $180°$ and we sort them by increasing magnitude (see Figure 5.6 for an illustration). Based on these orders, we denote the two sequences of points on the left and right side of the m-line by $(p_1, \ldots, p_{n'})$ and $(q_1, \ldots, q_{n''})$ respectively. We call a point *p-type* (resp. *q-type*) if it is in the first (resp. second) sequence. We also call a half-line from m that separates $\{p_1, \ldots, p_i\}$ from $\{p_{i+1}, \ldots, p_{n'}\}$ an *i^{th}-separator* of the p-type points. A j^{th}-separator of q-type points is defined similarly (we assume that the 0^{th} and n'^{th} (resp. n''^{th}) separators have the entire p-type (resp. q-type) points in one side). The i^{th} and j^{th} separators of the p-type and q-type points partition the plane into two parts. We call this partition the *(i, j)-partition* of the plane. One part of this partition contains the positive direction of the m-line which we call it the *positive side* of the partition and we call the other part the *negative side* of the partition.

Observation 5.3. *If $d(c_1^*, c_2^*) \leq r^*$, $m = \hat{m}$ and the m-line is correct, then an (i, j)-partition can be considered as R^* and (D_1^*, D_2^*) is its BOS.*

Note that in the above observation, the two separators from m passing the intersection points of D_1^* and D_2^* give us the desired (i, j)-partition. We denote the set of points in the positive and negative sides of the partition by $P_+^{i,j}$ and $P_-^{i,j}$ respectively. Based on our algorithm for restricted PCTCP, a BOS for an (i, j)-partition can be computed in $O(n \log n)$ time. Let $(D_-^{i,j}, D_+^{i,j})$ (with centers $(c_-^{i,j}, c_+^{i,j})$ respectively) be the output of this algorithm for the (i, j)-partition (see Figure 5.6 for an example). We refer to the first (resp. second) disk the *negative disk* (resp. *positive disk*) of the partition. A naive approach to obtain (D_1^*, D_2^*) is to apply our restricted PCTCP algorithm to each of the (i, j)-partitions and pick the best one. This will give us an $O(n^3 \log n)$ time complexity as there are quadratic partitions. In the following we show how we can get $(D_1^{m,l}, D_2^{m,l})$ by evaluating a sub-quadratic number of partitions. The idea is first computing $r^{m,l}$ which is the best cost we can get assuming m and l are correct. Then, we use it to compute $(D_1^{m,l}, D_2^{m,l})$.

5.6.1 Computing $r^{m,l}$

Let us define M^+ as a $(n' + 1) \times (n'' + 1)$ matrix whose $[i, j]$-element ($0 \leq i \leq n'$ and $0 \leq j \leq n''$) is $radius(D_+^{i,j})$. We call $M^+[i, j]$ *non-critical* if $D_+^{i,j}$ is the MED of $P_+^{i,j}$. Otherwise, we call it *critical.* We call $M^+[i, j]$ a *valid* element if $M^+[i, j] \geq radius(D_-^{i,j})$ and we call it *non-valid* otherwise. Because we assumed that l is correct, we can assume that positive disks determine $r^{m,l}$. This means that $r^{m,l}$ is indeed the minimum valid element of M^+.

Proposition 5.6. *For any $0 \leq i \leq n'$ and $0 \leq j \leq n''$, we have:*

1. *If $M^+[i, j]$ is non-critical, then $M^+[i', j'] \geq M^+[i, j]$ for all $i' \geq i$ and $j' \geq j$.*

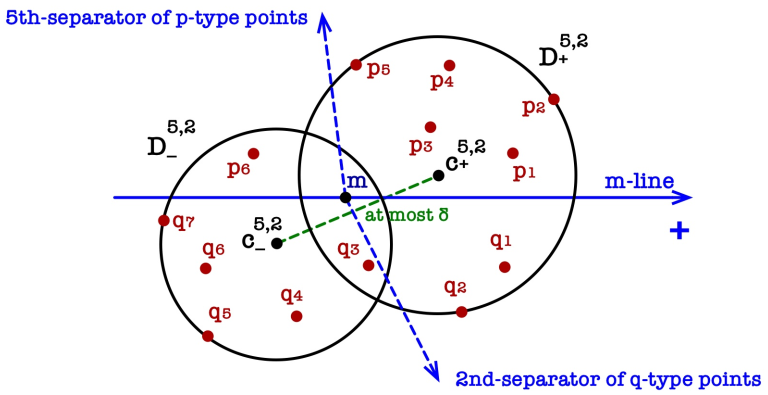


Figure 5.6: An example (i, j)-partition of a set of points and a BOS for the partition.

2. *If $M^+[i,j] > radius(D_-^{i,j})$, $M^+[i,j]$ is non-critical.*

3. *If $M^+[i,j]$ is valid and critical, then $M^+[i,j] = radius(D_-^{i,j})$ and $d(c_-^{i,j}, c_+^{i,j}) = \delta$.*

Proof. 1,2) We prove the first statement and the second statement is similar. Because $M^+[i,j]$ is non-critical, its dominating points make a triangle for $c_+^{i,j}$ (their induced triangle covers $c_+^{i,j}$). On the other hand, $P_+^{i,j} \subset P_+^{i',j'}$. So, $D_+^{i',j'}$ contains the dominating points of $D_+^{i,j}$ and its radius is greater or equal than $D_+^{i,j}$ which gives the Proposition. 3) Suppose $M^+[i,j]$ is critical and $M^+[i,j] > radius(D_-^{i,j})$. Then, we can slightly move both centers toward the dominating points of $D_+^{i,j}$ and get a solution with reduced cost which contradicts best optimality of $(D_+^{i,j}, D_-^{i,j})$ 4) If $M^+[i,j] > radius(D_-^{i,j})$, based on case 3, $M^+[i,j]$ is non-critical while we assumed it is critical. 5) If $d(c_-^{i,j}, c_+^{i,j}) < \delta$, we can slightly move $c_+^{i,j}$ toward the dominating points of $D_+^{i,j}$ without violating the PCC and reduce the radius of $D_+^{i,j}$ which again contradicts best optimality. □

We search M^+ to find $r^{m,l}$ as follows: we maintain a set of *candidate values*. During the search, when we evaluate an element $M^+[i,j]$(computing $(D_-^{i,j}, D_+^{i,j})$ and its dominating points), if $M^+[i,j]$ is valid, we add it to the candidate values and finally we set $r^{m,l}$ as the minimum candidate value.

In order to search M^+, we maintain two variables I and J where I stores the index of the current row that we are searching and J stores the column index for which we can discard any column with index greater than that. Initially, we set $I = 0$, $J = n''$ (n'' is the number of columns of M^+). We search the I^{th}-row by evaluating its elements backward starting from its J^{th}-element (if $J = -1$, the matrix search is done) toward its first element. Because we are looking for a minimum valid element of the matrix, we can use Proposition 5.6 to improve our search as follows: during the traversal of the row, if $M^+[I,j]$ is valid and non-critical, we set $J = j - 1$ (because $D_+^{I,j}$ is the MED of $P_+^{I,j}$, when we add more points to the positive side we can't get a smaller positive disk). We finish traversing the row and increase I by one if either the row is exhausted or we reach an index j such that $M^+[I,j]$ becomes non-valid. Note that in this case, $D_-^{I,j}$ is the MED of $P_-^{I,j}$ (similar to Proposition 5.6 part 3). Here we might have a valid element on some index $j' < j$ but, the cost of this solution can not be less than $radius(D_-^{I,j})$ (we add points to the negative side as we move left wise on a row). In order to make sure that we will count such costs in our algorithm, we can add $radius(D_-^{I,j})$ to the candidate set of the directed line in $\mathcal{X}$ with the opposite direction of the current m-line.

We continue this procedure until no element is left. Note that when none of $D_-^{i,j}$ and $D_+^{i,j}$ are a MED, we can't discard any element from the matrix because it is possible that when we move a point from one side to the other, the radii of both disks become greater or smaller while they remain equal (this situation can happen because of the PCC). So the number of evaluations in the above schema might be still quadratic. Next, we explain how to fix this problem.

Proposition 5.7. *If $M^+[i,j]$ is valid-critical and q_j is not a dominating point of $D_+^{i,j}$, then $M^+[i,j-1] \geq M^+[i,j]$.*

Proof. Because $D_+^{i,j}$ is not a MED, its center can't get closer to its farthest points in $P_+^{i,j}$ (dominating points of $D_+^{i,j}$) namely d_1 and d_2 because of the PCC. Now, by adding q_j to $P_-^{i,j}$, $D_-^{i,j-1}$ needs to cover more points. If its radius gets bigger, the proposition follows. Otherwise, according to the fact that q_j is not a dominating point of $D_+^{i,j}$, it is not possible to put $c_-^{i,j-1}$ on a place such that allow $c_+^{i,j-1}$ to get closer to d_1 and d_2 due to best optimality of $(D_-^{i,j}, D_+^{i,j})$. □

Note that in this proposition, if q_j does not become a dominating point of $D_-^{i,j-1}$, then $M^+[i,j-1] = M^+[i,j]$. A similar statement is also correct for two consecutive valid-critical elements in a column. Based on the above proposition, we can improve our matrix search as follows: while traversing a row(left wise), when we hit a valid-critical element $M^+[i,j]$, if both dominating points of $D_+^{i,j}$ are p-type, we discard the rest of the row (because by traversing a row, only q-type points will move to the other part of the partition) and continue the search on the next row. Similarly, if both dominating points of $D_-^{i,j}$ are q-type, we can discard the rest of its column. Otherwise, we jump to the first (largest index) element of the row for which a q-type dominating point of $D_+^{i,j}$ moves to the negative side and discard all the elements in between (because of Proposition 5.7). Similarly, we discard the portion of the rest of the column of $M^+[i,j]$ with row index smaller than the index of the p-type dominating point(s) of $D_-^{i,j}$(applying the column version of Proposition 5.7).

When we evaluate a valid-critical element $M^+[i,j]$, if we didn't discard the entire rest of its row or column, we *mark* the portion of its column that is not discarded after the evaluation of $M^+[i,j]$. Now, when we traverse the rows, we ignore and jump discarded and marked elements. Specially, if after evaluating an element $M^+[i,j]$, the largest index of the q-type dominating point of $D_+^{i,j}$ is j' and $M^+[i,j']$ is marked, we continue searching from the first(biggest index) unmarked or undiscarded element of the i^{th}-row after $M^+[i,j']$. Applying this marking schema in the matrix search will guarantee that the number of evaluations is linear. The problem of our matrix search with marking schema is that we may mark the minimum valid element of M^+ and so get an incorrect $r^{m,l}$. In the rest, we will show how to overcome this problem.

We call the above matrix search *initial search* of M^+ from *top-right*. Another way of searching M^+ is starting the search from $M^+[n',0]$ (the first element of the last row). But this time, instead of traversing the rows from right to left, we traverse the columns from bottom to top. The way we search the matrix is exactly symmetrical to the top-right search but here we mark sub-rows instead of sub-columns. We call this matrix search the initial search of M^+ from bottom-left. After performing two initial searches on M^+ one from the top-right and one from the bottom-left, still there might be some elements that are marked in both initial searches. We call these elements as *doubly-marked* elements. The next theorem enables us to search the doubly-marked elements in an efficient way which leads us to find $r^{m,l}$. Let us denote the doubly-marked elements of M^+ by $Doubly\text{-}Marked(M^+)$.

Theorem 5.1. *By evaluating a doubly-marked element $M^+[i,j]$, we can discard one of the following sub-rows or sub-columns of M^+:*

1. *Elements above $[i,j]$ ($M^+[i',j]$ with $i' \leq i$).*
2. *Elements below $[i,j]$ ($M^+[i',j]$ with $i' \geq i$).*
3. *Elements in front of $[i,j]$ ($M^+[i,j']$ where $j' \geq j$).*

Suppose that $M^+[\hat{i},\bar{j}]$ is a given doubly-marked element which is marked when we evaluate $M^+[\bar{i},\bar{j}]$ and $M^+[\hat{i},\hat{j}]$ in the initial top-right and bottom-left search respectively. When we evaluate $M^+[\hat{i},\bar{j}]$, we get $D_+^{\hat{i},\bar{j}}$ and $D_-^{\hat{i},\bar{j}}$ and their dominating points. For the sake of simplicity, let us denote the first disk by D'_+ and the second disk by D'_-. If D'_- is MED, then either $radius(D'_-) \geq radius(D'_+)$ or $radius(D'_-) < radius(D'_+)$. In the former, case 1 in Theorem 5.1 happens and in the latter, D'_+ is MED (otherwise we can reduce its cost and the solution can't be optimal) and so cases 2 and 3 of the theorem happen. We have a similar argument when D'_+ is a MED. So, the only left case is when none of the disks is MED. Note that in this case each of D'_+ and D'_- has exactly two dominating points. Let h_1, h_2 be the dominating points of D'_+ and h'_1, h'_2 be the dominating points of D'_-. If both h'_1 and h'_2 are p-type, case 3 happens (when we traverse the $\hat{i}^{th}$-row from left to right, we only add q-type points to the positive side). Also, if they are both q-type, case 2 happens. The bottleneck of proving Theorem 5.1 is when h'_1 and h'_2 have different types. In order to prove Theorem 5.1 in this special case, we use two key properties. First $M^+[\hat{i},\bar{j}]$ should be doubly-marked and second, m should be inside the convex hull of the points. We leave this proof to the end of this chapter and in the rest, we focus on how to use Theorem 5.1 to search *Doubly-Marked*(M^+) efficiently.

5.6.2 Searching the Doubly-Marked Elements

For simplicity, we assume that $n' = 2^g - 1$ for some integer value $g > 1$ (so the number of rows is a power of 2). We define the k^{th}-division of M^+ as the sub-matrix consisting of the rows from $n'/2^k$ to $n'/2^{k-1} - 1$ (we search the first row independently by evaluating all of its doubly-marked elements). We search the divisions of M^+ in order from its first division. Let us denote the k^{th}-division sub-matrix by DIV_k. Here, we explain how to search DIV_k. Let I and J be the row and column indices (with respect to DIV_k) of the element that we are processing at each time. Initially, we have $I = J = 1$ (the first row and column of DIV_k). We evaluate the non-discarded elements of the I^{th}-row from left to right starting from the column index J. If the result of an evaluation is case 1 or 2 in Theorem 5.1, we discard the corresponding portion of M^+ (in all divisions) and increase J by one. But if case 3 happens, we go to the next row and increase I (we always move rightwise). After we proceed with all divisions, some elements might left unevaluated and undiscarded in each division due to the occurrence of case 1. We recursively perform the entire above process on these unevaluated elements in each division until all elements are either discarded or evaluated. So, if only doubly-marked elements remained in M^+ (we have discarded all other elements in the two initial searches), then the procedure SEARCH-DM(M^+) in Algorithm 8 will give us a minimum valid element of M^+.

Algorithm 8 SEARCH-DM(M)

```
1: Let $M$ be a $n \times m$ matrix.
2: Split $M$ into $\log n$ divisions $\{DIV_1, \ldots, DIV_{\log n}\}$ .
3: for $k = 1, \ldots, \log n$ do // We search the divisions in order.
4:     Set $I = J = 1$.
5:     repeat
6:         Evaluate $DIV_k[I, J]$ and discard the portion of $M$ according to Theorem 5.1.
7:         if (case 1 or 2 happens) and $J < m$ then $J = J + 1$.
8:         else
9:             $I = I + 1$.
10:        end if
11:    until $I > n/2^k$ // number of rows in $DIV_k$.
12:    SEARCH-DM($DIV_k$) if $DIV_k$ has unevaluated/undiscarded element.
13: end for
14: Evaluate all non-discarded elements of the first row of $M$. // Until the case 3 happens.
```

Theorem 5.2. *SEARCH-DM(M^+) evaluates $O(n \log n)$ elements of M^+.*

Proof. First, if only cases 2 and 3 happen in the algorithm, then we don't need the recursion part and so the total number of evaluations becomes $O(n \log n)$ (in each iteration of searching DIV_k either I or J would be increased). Now, suppose that any of the cases 1, 2, or 3 can happen. Note that the number of case 3s in all divisions of a same recursion level (the original $\log n$ divisions has recursion level zero and the level of the divisions in the recursion part of the algorithm is defined based on their appearance in the recursion tree) is at most n because two divisions of a same level has disjoint rows. Because we have $O(\log n)$ levels, the total number of case 3 evaluations is $O(n \log n)$. Now, if after the evaluation of some $DIV_k[i, j]$, case 1 happens, we can't discard any new element from DIV_k but all the elements above $DIV_k[i, j]$ in M should be discarded. This means that while searching each of the divisions $DIV_{k+1}, \ldots, DIV_{\log n}$ and the first row, we don't need to evaluate the j^{th}-column. On the other hand, DIV_k has $\log(n/2^k) = \log n - k$ divisions and a row. Each of these divisions can have at most one cases 1 or 2 in the j^{th}-column. So, we can have a correspondence between the extra cases 1 and 2 evaluations in searching the divisions and the first row of DIV_k (not its recursion part) and the matrix elements that we didn't evaluate in $DIV_{k+1}, \ldots, DIV_{\log n}$. So, the total number of evaluations would remain $O(n \log n)$.□

Note that in a constant time, we can check whether an element is discarded or not. Because in each recursion level, the divisions are disjoint, at each level we check each element of M at most once and because we have $O(\log n)$ levels, the total cost of matrix element checking is $O(n^2 \log n)$. On the other hand, our algorithm to solve the restricted PCTCP costs $O(n \log n)$, if we directly use it to evaluate matrix elements, the total time complexity of SEARCH-DM(M^+) becomes $O(n^2 \log^2 n)$. As we mentioned in Section 2, the bottleneck of solving the restricted PCTCP is computing the farthest-point Voronoi diagram of each part of the partition which costs $O(n \log n)$. So, if we can reduce this cost by performing a preprocessing step, we can reduce the overall time complexity of SEARCH-DM(M^+). In order to speed up matrix element evaluation, we use the following lemma from [30]:

Lemma: [30] *If X and Y are arbitrary sets of points in the plane, then $\mathcal{F}(X \cup Y)$ can be constructed from $\mathcal{F}(X)$ and $\mathcal{F}(Y)$ in $O(|X| + |Y|)$ time ($\mathcal{F}(X)$ represents the farthest-point Voronoi diagram of X).*

The preprocessing step: Let (X_i^+, X_i^-) (resp. (Y_j^+, Y_j^-)) be the partition of the p-type (resp. q-type) points induced by the i^{th}-separator (resp. j^{th}-separator). In the preprocessing phase, we compute the farthest-point Voronoi diagram of all X_i^+, X_i^-, Y_j^+ and Y_j^- for $0 \leq i \leq n'$ and $0 \leq j \leq n''$. This step can be done in $O(n^2)$ time using the above lemma because as i or j increases or decreases by one, a point from one side would be added to the other side.

Now, we can reduce the cost of matrix element evaluation as follows: In order to evaluate $M^+[i,j]$, we construct $\mathcal{F}(P_+^{i,j})$ (resp. $\mathcal{F}(P_-^{i,j})$) in $O(n)$ time by applying the lemma to $\mathcal{F}(X_i^+)$ and $\mathcal{F}(Y_j^+)$ (resp. $\mathcal{F}(X_i^-)$ and $\mathcal{F}(Y_j^-)$). So, the total complexity of matrix evaluation is $O(n)$. This reduces the time complexity of SEARCH-DM(M^+) and so the cost of finding $r^{m,l}$ to $O(n^2 \log n)$.

5.6.3 Obtaining $(D_-^{m,l}, D_+^{m,l})$ having $r^{m,l}$

Note that we already have an initial solution that is optimal and its cost is $r^{m,l}$ (from our search for $r^{m,l}$). But, there might be another optimal solution with the same cost and a smaller non-determining disk that we discarded during the search. If this initial solution is not best optimal, then the non-determining disk of a BOS must be strictly smaller than its determining disk. So, we can assume that the positive disk of the BOS is the MED of the points in the positive side. Consider a matrix $\bar{M}$ for which its $(i,j)^{th}$-element is the radius of the MED of the points on the positive side of the (i,j)-partition. We search $\bar{M}$ from the last element of its first row and traverse the rows backwards (similar to the initial top-right search). After evaluating an element (i,j) of the matrix (which can be done in linear time according to [51]), if it is bigger than $r^{m,l}$, we discard all elements (i,j') of the matrix with $j' \geq j$ because they are all greater than $r^{m,l}$ and if it is less than $r^{m,l}$, we discard the elements with $i' \leq i$ because they are all less than $r^{m,l}$. But, when it is exactly $r^{m,l}$, we compute its non-determining disk using the restricted PCTCP algorithm (costs $O(n \log n)$) and store its radius. Here, we can also discard all elements (i',j) of the matrix with $i' \leq i$. This is because as we advance more left into the row, we would have more points on the negative side and so if there is any optimal solution on the left of (i,j) in the row, its non-determining disk should cover more points and thus can't give us a better solution. So, by each evaluation, we discard a row or a column of the matrix which means that the total number of evaluations is linear. Therefore, the total complexity of finding a BOS given $r^{m,l}$ is $O(n^2 \log n)$. Combining it with the complexity of computing $r^{m,l}$ gives us the total time complexity $O(n^2 \log n)$ to obtain (D_1^{NA}, D_2^{NA}).

5.7 Computing a BOS in the Far Distant Case

For the far distant case, we assume that $d(c_1^*, c_2^*) > 3r^*$. In this situation, the approach of Sharir's far distant case [59] for the decision 2-center problem still works as follows: set an arbitrary point in the plane as the origin and build 360 directed lines $\mathcal{X}$ passing from the origin such that the degree between each line and its neighbours is 1°. Then, for one unknown correct line $\vec{x}_c \in \mathcal{X}$, the angle between $line(c_1^*, c_2^*)$ and $\vec{x}_c$ is at most 1°. Suppose that we set $\vec{x}_c$ is the x-axis and sort the x-coordinates of the points in P as a sequence $(x_1, \ldots, x_n)$ (see Figure 5.7). Now, if we consider the set of lines $\mathcal{L}_F^{\vec{x}_c} = \{x_i \perp x_{i+1} : 1 \leq i < n\}$ ($x_i \perp x_{i+1}$ is the vertical line on $\vec{x}_c$ at the mid-point of $[x_i, x_{i+1}]$), at least one $l \in \mathcal{L}_F^{\vec{x}_c}$ will separate

D_1^* from D_2^*. Because $\vec{x}_c$ is unknown, we build $\mathcal{L}_F^{\vec{x}}$ for all $\vec{x} \in \mathcal{X}$ and set $\mathcal{L}_F = \bigcup_{\vec{x}\in\mathcal{X}} \mathcal{L}_F^{\vec{x}}$. Note that the number of lines in $\mathcal{L}_F$ is linear. Here, each line $l \in \mathcal{L}_F$ induces a partition on P. We apply our algorithm for the restricted PCTCP to each of such partitions and set the best one as (D_1^{FA}, D_2^{FA}). So, the time complexity of the far distant case is $O(n^2 \log n)$.

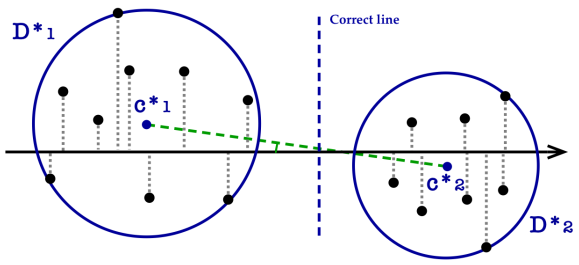


Figure 5.7: Finding a separating line in the far distant case

5.8 Computing a BOS for the Distant Case.

In the distant case, we assume that $r^* < d(c_1^*, c_2^*) \leq 3r^*$. The idea is to first compute r^{DA} by imposing the condition that the disks are congruent and then using r^{DA} to build (D_1^{DA}, D_2^{DA}). So, let $(\hat{D}_1, \hat{D}_2)$ with centers $(\hat{c}_1, \hat{c}_2)$ be a BOS having the distant assumption such that the radii of the disks are equal (disks are congruent). So, the cost of this solution is r^{DA}. Here, the objective is to compute $(\hat{D}_1, \hat{D}_2)$. We first apply the algorithm of [43] to obtain an optimal solution for the 2-center problem on P with minimum distance between their centers. In order to have this additional proximity property in the solution, we replace Hershberger's feasibility test [33] in [43] with Sharir's unparallelized feasibility test in [59] (note that we can't use Sharir's algorithm in [59] because it uses simulating parallel feasibility test which makes the algorithm impractical). If the distance between the centers of this solution is equal or less than δ, we set r^{DA} as the cost of this solution and try to build (D_1^{DA}, D_2^{DA}) (will be discussed later in the chapter) based on this cost (if we couldn't build a solution with this cost satisfying the distant assumption, we would know that a BOS with this cost exists in the far distant or the nearby cases and thus, we won't miss the BOS for the problem). Otherwise, we can assume that $d(\hat{c}_1, \hat{c}_2) = \delta$.

Similar to the Section 3, we build a set of constant size directed lines $\mathcal{X}$ such that for at least one directed line $\vec{x} \in \mathcal{X}$, $\hat{c}_1$ is on the negative side of $\hat{c}_2$ and the angle between $line(\hat{c}_1, \hat{c}_2)$ and $\vec{x}$ is at most $1°$. Without loss of generality we can assume that $\vec{x}$ is horizontal and its positive direction is right wise. Let v_1 be the leftmost point of $\hat{D}_2$ not in the interior of $\hat{D}_1$ and v_2 be the rightmost point of $\hat{D}_1$ not in the interior of $\hat{D}_2$. If v_1 lies to the right of v_2, we already catch the BOS in the far distant case and we are done. So, assume that v_1 is on the left side of v_2. The objective here is to find a vertical line l such that it separates $\hat{c}_1$ from v_1. First, note that the difference between the x-coordinates of v_1 and v_2 is at least

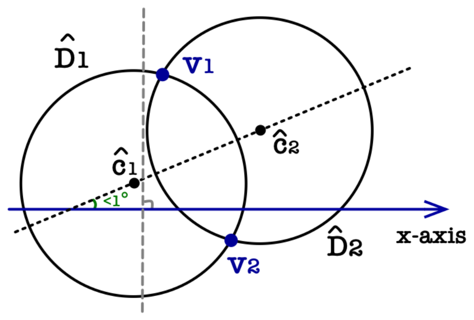


Figure 5.8: Separating $\hat{c}_1$ and v_1 by a vertical line

$0.9r^*/2$ and $r^* > \Delta x/4$ where Δx is the difference between the x-coordinates of the leftmost and rightmost points. Having Δx, one of the vertical lines at the distance $k\Delta x/9$ (for some $1 \leq k \leq 9$) from the leftmost point will separate $\hat{c}_1$ from v_1. So, we can build a set $\mathcal{L}_D$ of lines such that for at least one line $l \in \mathcal{L}_D$, there exist an x-axis in $\mathcal{X}$ such that with respect to that, l is vertical and separates $\hat{c}_1$ and v_1.

Note that all points of P on the left-side of l are covered by $\hat{D}_1$ and so, we can assume that $\hat{c}_1$ is on the boundary of the intersection hull of these points at radius r^*. According to the assumption $d(\hat{c}_1, \hat{c}_2) = \delta$, we can say that at radius r^*, the distance between the intersection hull of the points covered by $\hat{D}_1$ and the intersection hull of the points not covered by $\hat{D}_1$ is exactly δ (the distance between two intersection hulls is the minimum possible distance between their points). In the rest of this section, we discuss how we can use this property to compute r^{DA} and $(\hat{D}_1, \hat{D}_2)$.

Suppose that l is a given vertical line in $\mathcal{L}_D$. Denote the set of points in P on the left side (resp. right side) of l by P^- (resp. P^+). Also, denote the intersection hull of P^- with respect to a radius $r > 0$ by $H^-(r)$. Define r_l as the minimum radius for which there exist a point $x \in \partial H^-(r_l)$ such that the distance between x and the intersection hull of the points in P not covered by $Disk(x, r_l)$ (the disk with center x and radius r_l) is exactly δ. So, in order to find r^{DA} it is enough to compute r_l for all $l \in \mathcal{L}_D$ and set $r^{DA} = \min\{r_l : l \in \mathcal{L}_D\}$.

In order to find r_l, we need a feasibility test to answer the following question: given an r, determine whether r is greater, equal or smaller than r_l. Consider the set of circles $\mathcal{A}(r) = \{circle(p, r) : p \in P^+\}$ ($circle(p, r)$ is the circle with center p and radius r) and compute the intersection points of each circle of $\mathcal{A}(r)$ with $\partial H^-(r)$. These intersection points and the vertices of $\partial H^-(r)$ induce a partition on $\partial H^-(r)$. We denote this partition by $\pi(r)$ which can be considered as an alternating sequence of arc interiors and endpoints. We assume that the order is clockwise starting from its leftmost endpoint. We call each of these arc interiors and endpoints a *field* of $\pi(r)$ (so, if $\pi(r)$ has k arcs, it would have $2k$ fields). See Figure 5.9. We observe that for each field f of $\pi(r)$, $disk(x, r)$ covers a same set of points for any $x \in f$. For simplicity, we call the set of points in P covered by $disk(x, r)$,

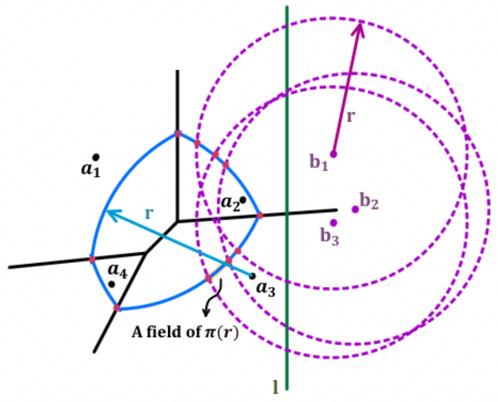


Figure 5.9: The circles of $\mathcal{A}(r)$ and its induced partition $\pi(r)$ on $\partial H^{-}(r)$

the points covered by f at radius r. So, $\pi(r)$ is a sequence of fields each covers a specific set of points. For each field $f \in \pi(r)$, denote the intersection hull of the points not covered by f with respect to r by $H_f^+(r)$. Note that the difference between the set of points covered by two neighbour fields in $\pi(r)$ is at most two(based on our assumption that the points are in general position). Also, we can compute the sequence of disks that enter or leave $H_f^+(r)$ when f varies in $\pi(r)$ in the clockwise order. Having this sequence allows us to use the data structure of [35] to compute $H_{f'}^+(r)$ having $H_f^+(r)$ in $O(\log n)$ amortized time where f' is a neighbour field of f in $\pi(r)$. So, to do the feasibility test, we first compute $H_{f_0}^+(r)$ where f_0 is the first field of $\pi(r)$ and then traverse $\pi(r)$ in the clockwise order and at each field f, update the intersection hull of the points not covered by f and compute $d(f, H_f^+(r))$. Because both f and $H_f^+(r)$ are convex, we can compute the distance between them in $O(\log n)$ time [20]. During the traversal, we stop and return *greater* as soon as for a field f, $d(f, H_f^+(r)) < \delta$. If we reach the end of the traversal, return *equal* if we've seen a field f for which $d(f, H_f^+(r)) = \delta$ and we couldn't find any other field f' with $d(f', H_{f'}^+(r)) < \delta$. Otherwise, we return *smaller*. So, the feasibility test can be done in $O(n \log n)$ time. The Procedure RL-FTEST(r) in Algorithm 9 represent the pseudocode of our feasibility test.

Algorithm 9 RL-FTEST(r)

```
1: Compute $H^-(r)$ and the sequence $\pi(r) = (f_0, \dots, f_m)$ on it.
2: Let $f = f_0$.
3: Compute $H_f^+(r)$ and the set of points covered by $f$ at radius $r$.
4: Let $d_{min} = d(f, H_f^+(r))$.
5: Traverse $\pi(r)$ and at each field store the point(s) (at most two) that leave/enter the
   coverance.
6: for $f = f_1, \dots, f_m$ do
7:     Compute $H_f^+(r)$ by updating from $H_{f-1}^+(r)$.
8:     $d_{min} = min\{d(f, H_f^+(r)), d_{min}\}$.
9:     if $d_{min} < \delta$ then
10:        Return Greater.
11:    end if
12: end for
13: if $d_{min} = \delta$ then Return Equal
14: else
15:     Return Smaller
16: end if
```

Now, we discuss our algorithm to compute r_l. We first build $\mathcal{F}(P^-)$ (farthest-point Voronoi diagram of P^-) and do a binary search on the weights (see Appendix A) of the vertices of $\mathcal{F}(P^-)$ using the above feasibility test to obtain an interval $I^* = (i_0, i_1)$ such that $r_l \in I^*$ and for each vertex v of the diagram, I^* does not contain the weight of v. Because we use $O(\log n)$ feasibility tests, the cost of obtaining I^* is $O(n \log^2 n)$. As soon as we found I^*, we can build $H^-(i_0)$ and the set of its arms A^-.

The idea here is simulating the propagation of $\pi(r)$ and the circles in $\mathcal{A}(r)$ when r varies from i_0 to i_1. To do this, we assume that at time $t \in I^*$, the radius of π and the circles in $\mathcal{A}$ is t (if a field is an endpoint, its radius is the radius of an arc containing it). The minimum time t such that $d(f, H_f^+(t)) \leq \delta$ for a field f in $\pi(t)$ is actually our r_l. Consider $t_1, t_2 \in I^*$ and let $\pi(t_1) = (f_1, \dots, f_j)$ and $\pi(t_2) = (f'_1, \dots, f'_{j'})$. We say that $\pi(t_1)$ and $\pi(t_2)$ have a same structure if $j = j'$ and for each $1 \leq i \leq j$, f_i and f'_i cover exactly a same set of points. In this case, we consider f_i and f'_i as a same field in different times. Note that if $\pi(t_1)$ and $\pi(t_2)$ have different structures, for any $t \geq t_2$, $\pi(t_1)$ has a different structure from $\pi(t)$. So, we can consider a sequence of times $T = (t_0 = i_0, t_1, \dots, t_k = i_1)$ for some integer k such that π has a same structure between any two consecutive times in the sequence. We call this sequence the *event-sequence* and each time in this sequence an *event-time*. We can see that a time t is an event-time if one of the following events happens at t:

1. Circle in $\mathcal{A}(t)$ collides with $H^-(t)$.

2. Two endpoints of $\pi(t-\epsilon)$ collide at $\pi(t)$ where $\pi(t-\epsilon)$ has the structure exactly before the event for a sufficiently small $\epsilon > 0$.

At a first type event, new fields emerge and at a second type event a field disappears (new fields may appear). Note that the number of such events is $O(n^2)$ and we can obtain T by considering the intersection of each pair of circles (or disks) on π and finally sort the times. So, we can compute T (increasingly sorted) in $O(n^2 \log n)$ time. Here, we are going to find an interval $T^* = (t', t'') \subseteq I^*$ such that $r_l \in T^*$ and it contains no event-time. To do this, we apply our feasibility test $O(\log n)$ times to perform a binary search on T to get T^*. So, computing T^* again costs $O(n \log^2 n)$ time.

Next, for each field $f \in \pi(t \in T^*)$, we store the set of points not covered by it denoted by f^+ and compute the farthest-point Voronoi diagram $\mathcal{F}(f^+)$ of these points. In order to find r_l, we need to compute t_f^δ for each field $f \in \pi(t \in T^*)$ where t_f^δ is the earliest time for which $d(f \in \pi(t_f^\delta), H_f^+(t_f^\delta)) = \delta$. Finally, we have $r_l = min\{t_f^\delta : f \in \pi(t \in T^*)\}$.

Here, we discuss how to compute t_f^δ for a field $f \in \pi(t \in T^*)$. First, we build $\mathcal{F}(f^+)$ in $O(n \log n)$ time. Note that having $\mathcal{F}(f^+)$, for a given $t \in T^*$, we can compute $H_f^+(t)$ with ordered arcs in linear time. Next, we apply binary search on the weights(times) of the vertices of $\mathcal{F}(f^+)$ using the algorithm of [20](which costs $O(\log n)$ for computing the distance between two convex objects) as the feasibility test to check whether the distance between f and H_f^+ is smaller, equal or larger than δ. Let $T_f^* \subseteq T^*$ be the resulting interval. So, when t varies in T_f^*, no arc appears or vanishes in $H_f^+(t)$. So, T_f^* can be computed in $O(n \log n)$ time. Having T_f^*, because we have no structural change in H_f^+, we can compute t_f^δ in linear time (by considering the arms of H_f^+). So, the total cost of computing t_f^δ is $O(n \log n)$ and thus, r_l can be computed in $O(n^2 \log n)$ time. The procedure FIND-RL(l) in Algorithm 10 will return r_l given a line l.

Algorithm 10 FIND-RL(l)

```
1:  Build $\mathcal{F}(P^-)$ and its weights.
2:  Perform a binary search on the weights using RL-FTEST to get an interval $I^* = [i_0, i_1]$.
3:  Compute the event-times sequence $T$ when $\pi(t)$ and $\mathcal{A}(t)$ expands as $t \in I^*$.
4:  Perform a binary search on $T$ using RL-FTEST to get an interval $T^*$.
5:  Let $f_1, \ldots, f'_m$ be the fields of $\pi(t)$ when $t \in T^*$. // The fields are same as $t$ varies in
    $T^*$.
6:  for $i$ from 1 to $m'$ do
7:      Let $f^+$ be the set of points not covered by $f_i$.
8:      Compute $\mathcal{F}(f^+)$ and perform a binary search on its weights with the following test:
9:      Test($w$):
10:         Compute $H^+_{f_i}(w)$ and let $d = d(f_i, H^+_{f_i}(w))$.
11:         if $d > \delta$ then Return *Smaller*.
12:         else if $d = \delta$ then Return *Equal* else Return *Greater*.
13:     Let $T^*_{f_i}$ be the final interval. // When $t$ varies in $T^*_{f_i}$ the structure of $H^+_{f_i}$ doesn't
        change.
14:     Compute the time $t^\delta_i$ for which $d(f_i, H^+_{f_i}) = \delta$.
15: end for
16: Return $r_l$ as $min\{t^\delta_i \ : \ 1 \le i \le m'\}$.
```

Having r^{DA}, we can obtain a BOS for it as follows: for each $l \in \mathcal{L}_2$, we consider two assumptions: first, the determining disk is on the left side of l and second it is on the right side of l. For the first case, we consider $H^-(r^{DA})$ and circles in $\mathcal{A}$ as fixed objects (not propagating) and for each field f, we propagate H^+_f up to radius R^{DA} based on the method we explained above and compute the minimum radius(time) for which $d(H^-(r^{DA}), H^+_f) = \delta$. Let r'_l be the minimum such radius and infinite if it doesn't exist. For the second case, we propagate both circles in $\mathcal{A}$ and fields of $H^-(i_0)$ to obtain event-times but we build all H^+_fs according to the fixed radius r^{DA} and follow the above algorithm to obtain a radius r''_l. Comparing r'_l, r''_l and their solutions, we obtain a BOS assuming l is a correct line. Comparing the results for all $l \in \mathcal{L}_2$ will give us a BOS (D^{DA}_1, D^{DA}_2) having the distant assumption with total time complexity of $O(n^2 \log n)$.

5.9 Proof of Theorem 5.1

Lets x and y be two points in the plane. We recall that the directed line passing from x and y directed from x to y is denoted by $line(x, y)$. Also, we denote the half-line from x passing y by $half\text{-}line(x, y)$. In this section, when we say first, second, third and fourth quarter of a point t with respect to some directed line l we mean the first, second, third and fourth quarter of the plane when we consider t as the origin and the directed line parallel to l passing t as the x-axis. We prove the theorem by providing several propositions. For simplicity, when we assign a letter to a geometric object inside a proof, the scope of that notation is only inside that proof and we may assign that letter to another object later. In addition, when we state a proposition or observation, we mean if the proposition or observation is false, the statement is either impossible or the theorem follows. So, we can assume that after a proposition or observation, its statement is always true. Also, when we say one disk is smaller than another disk, we mean smaller or equal. Let c be the center of some disk D. We say that a set of three points T make a triangle for c if their induced triangle covers c and any disk that covers T has a radius greater than $radius(D)$.

Observation 5.4. *Let d_1 and d_2 be the dominating points of $D_+^{i,j}$. Then for any point $t \notin D_+^{i,j}$ inside the cone induced by two half-lines from $c_+^{i,j}$ along $line(d_1, c_+^{i,j})$ and $line(d_2, c_+^{i,j})$ containing $c_-^{i,j}$, the points d_1d_2t makes a triangle for $c_+^{i,j}$.*

A similar statement is also true for $D_-^{i,j}$ and their dominating points. Figure 5.10 shows an example for Observation 5.4.

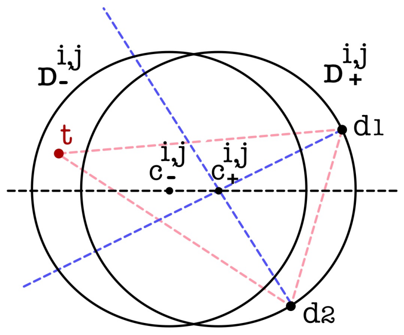


Figure 5.10: d_1, d_2 and x make a triangle for $c_+^{i,j}$

Let $M^+[\hat{i}, \bar{j}]$ be a given doubly-marked element. Also, let $M^+[\bar{i}, \bar{j}]$ and $M^+[\hat{i}, \hat{j}]$ be the elements for which we marked $M^+[\hat{i}, \bar{j}]$ when we evaluated them in the top-right and bottom-left initial searches respectively. For simplicity of notation, henceforth we denote $D_+^{\hat{i},\bar{j}}$, $D_-^{\hat{i},\bar{j}}$, $c_+^{\hat{i},\bar{j}}$ and $c_-^{\hat{i},\bar{j}}$ by D'_+, D'_-, c'_+ and c'_- respectively. Similarly, we denote $D_+^{\bar{i},\bar{j}}$, $D_-^{\bar{i},\bar{j}}$, $c_+^{\bar{i},\bar{j}}$, $c_-^{\bar{i},\bar{j}}$ by $\bar{D}_+$, $\bar{D}_-$, $\bar{c}_+$, $\bar{c}_-$ and $D_+^{\hat{i},\hat{j}}$, $D_-^{\hat{i},\hat{j}}$, $c_+^{\hat{i},\hat{j}}$, $c_-^{\hat{i},\hat{j}}$ by $\hat{D}_+$, $\hat{D}_-$, $\hat{c}_+$, $\hat{c}_-$ respectively. Note that based on our assumptions all of these disks has exactly two dominating points.

We denote the dominating points of $\bar{D}_+$ by a and b, $\bar{D}_-$ by x and y, $\hat{D}_+$ by c and d, $\hat{D}_-$ by w and u, D'_+ by h_1 and h_2 and finally D'_- by h'_1 and h'_2. Let a and c be the two dominating points of $\bar{D}_+$ and $\hat{D}_+$ who lie on the opposite side of x and w with respect to $line(\bar{c}_-, \bar{c}_+)$ and $line(\hat{c}_-, \hat{c}_+)$ respectively. Note that h'_1 and h'_2 should be in both $P_-^{\bar{i},\bar{j}}$ and $P_-^{\hat{i},\hat{j}}$ (because if they are in the positive side, they can't be dominating points of D'_-). a, b, c and d are in $P_+^{\hat{i},\bar{j}}$ and so covered by D'_+ (because we only add points to the positive side when we walk on M^+ from left to right or top to bottom). Also, suppose that x and w are the dominating points of $\bar{D}_-$ and $\hat{D}_-$ respectively who are moved to the positive side in the $(\hat{\imath}, \bar{\jmath})$-partition. So, we can assume that y and u are not in $P_+^{\hat{\imath},\bar{\jmath}}$. This is because if for example $y \in P_+^{\hat{\imath},\bar{\jmath}}$, D'_+ should cover a, b, x and y which are the all dominating points in the pair $(\bar{D}_-, \bar{D}_+)$. This means that the radius of D'_+ and any positive disk of (i', j')-partition with $i' \geq \hat{\imath}$ and $j' \geq \bar{\jmath}$ is greater than the radius of $\bar{D}_-$ and so, we can discard them and the theorem follows (cases 2 and 3 in the theorem). Note that x is p-type and w is q-type (because x (resp. w) is moved to the positive side when we walk on a column (resp. row) of M^+) also y, u should be covered by D'_-. Furthermore, $x \in P_+^{\hat{\imath},\hat{\jmath}}$ because x is p-type and if $x \in P_-^{\hat{\imath},\hat{\jmath}}$, we can not bring it into the positive side in the $(\hat{\imath}, \bar{\jmath})$-partition by walking on the $\hat{\imath}^{th}$-row. Similarly, $w \in P_+^{\bar{\imath},\bar{\jmath}}$. We assume that u is covered by $\bar{D}_-$ because if $u \in P_+^{\bar{\imath},\bar{\jmath}}$, then u should also be in $P_+^{\hat{\imath},\bar{\jmath}}$ which means D'_+ would cover c, d, w, u and the theorem follows. Similarly, y should be covered by $\hat{D}_-$. Note that the intersection of the disks are non-empty because we are in the nearby case. So, we can consider a point inside the intersection of the disks and have an angular clockwise and counter-clockwise order for all the dominating and intersection points of the disks. We consider two cases for $M^+[\hat{\imath}, \bar{\jmath}]$ and prove the theorem for each case separately.

Case 1: $M^+[\hat{\imath}, \bar{\jmath}] > max\{M^+[\bar{\imath}, \bar{\jmath}], M^+[\hat{\imath}, \hat{\jmath}]\}$

According to Proposition 5.5 part 2, both h'_1 and h'_2 can't be on D'_+. Let h'_1 be the one outside D'_+ (the case h'_2 is outside D'_+ is similar). We show that $h'_1 h_1 h_2$ makes a triangle for c'_+. Suppose not. Based on Observation 5.4, for one of h_1 or h_2 namely h_2, h'_1 should be on the opposite side of h_2 with respect to $line(c'_-, c'_+)$ (see Figure 5.11). Also, h'_1 and c'_- should lie on opposite sides of $line(h_2, c'_+)$. Note that h_2 should lie outside of D'_- otherwise, we can't place h'_1 having these conditions.

Without loss of generality, we assume that h_1 is p-type (the case h_1 is q-type is similar). Because h_1 is p-type, $h_1 \in \hat{D}_+$ (if h_1 was q-type, we would have $h_1 \in \bar{D}_+$). This is because when we traverse on a row, we only move q-type points to the positive side. On the other hand, both h'_1 and h'_2 should be covered by $\hat{D}_-$. Based on this situation, we have two sub-cases:

sub-case 1: $h_2 \in \hat{D}_+$**.** In this sub-case, because $\{h'_1, h'_2\} \in \hat{D}_-$ and $\{h_1, h_2\} \in \hat{D}_+$, $M^+[\hat{\imath}, \bar{\jmath}]$ can't be greater than $M^+[\hat{\imath}, \hat{\jmath}]$ which is contradiction.

sub-case 2: $h_2 \in \hat{D}_-$**.** Let t be the intersection point of $half\text{-}line(h_2, c'_+)$ and D'_+ (see Figure 5.11). Note that $\hat{D}_-$ covers h'_1 and it can not cover t (because it is smaller). So, $\hat{c}_-$ should lie on the side of $line(h_2, c'_+)$ that has h'_1. Also, $\hat{D}_-$ covers all points of $P_-^{\hat{\imath},\bar{\jmath}}$. Now, h'_1 should be on the third quarter of c'_+ with respect to $line(h_2, c'_+)$. This is because h'_1 is outside D'_+ (the way we chose h'_1) and h_2 is on the opposite side of $line(c'_-, c'_+)$ with respect to h'_1. Also, h'_1 and h'_2 are on opposite sides of $line(h_2, c'_+)$ (because of Proposition 5.5). Now, for any point z inside $\hat{D}_-$, if $h'_1 z$ intersects the $half\text{-}line(c'_-, c'_+)$, it can't be h'_2 (again Proposition 5.5 second part) and if it doesn't, $|zc'_-| < |h'_1 c'_-| = M^+[\hat{i}, \bar{j}]$ which again means that z can't be h'_2. So, we don't have any place for h'_2 which is contradiction (see Figure 5.11).

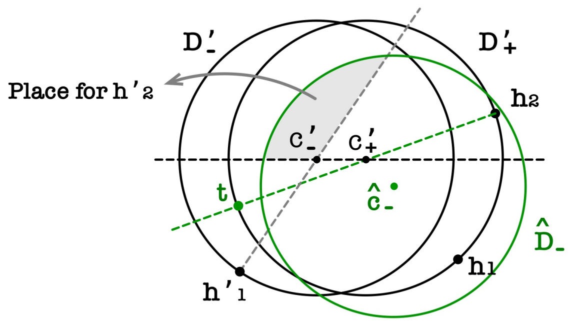


Figure 5.11: Proof of Sub-case 2 in Case 1.

Case 2: $M^+[\hat{i}, \bar{j}] \leq max\{M^+[\bar{i}, \bar{j}], M^+[\hat{i}, \hat{j}]\}$

In this section, we assume that $M^+[\hat{i}, \bar{j}] \leq M^+[\bar{i}, \bar{j}]$ and the case $M^+[\hat{i}, \bar{j}] \leq M^+[\hat{i}, \hat{j}]$ is similar. Henceforth, we consider $line(\bar{c}_-, \bar{c}_+)$ as the x-axis unless we say otherwise. We consider two sub-cases based on the position of x with respect to $\bar{D}_+$. For simplicity, when we provide a proposition within each case or sub-case, we include the assumptions of the case or sub-case in the proposition.

Sub-case 1: $\mathbf{x} \notin \bar{\mathbf{D}}_+$.

In this sub-case, we assume that x is below $line(\bar{c}_-, \bar{c}_+)$ and the case x is above the line is similar.

Proposition 5.8. *c'_+ should be on the lower-right of $\bar{c}_+$.*

Proof. D'_+ should cover a, b and x. Based on Proposition 5.5, a and b should be on different sides of $line(\bar{c}_-, \bar{c}_+)$ and a should be on the first quarter with respect to $\bar{c}_+$ and thus, outside of $\bar{D}_-$ (otherwise, because of Proposition 5.5 part 2, b should be on the right side of $\bar{c}_+$. Now, $x \notin \bar{D}_+$ and D'_+ should contain the triangle $\triangle abx$ which contradicts D'_+ is smaller than $\bar{D}_+$). If c'_+ is on the lower-left of $\bar{c}_+$, D'_+ can't cover a while it is smaller than $\bar{D}_+$. If c'_+ is on the top-left of $\bar{c}_+$, it should be above $line(a, \bar{c}_+)$ and b should be below this line on $\partial\bar{D}_+$. This implies that D'_+ can't cover a, b and x while being smaller than $\bar{D}_+$ (see Figure 5.12 for such situation). If c'_+ is on top-right $\bar{c}_+$, its distance from x would be greater than the radius of $\bar{D}_+$ which is again not possible. $\square$

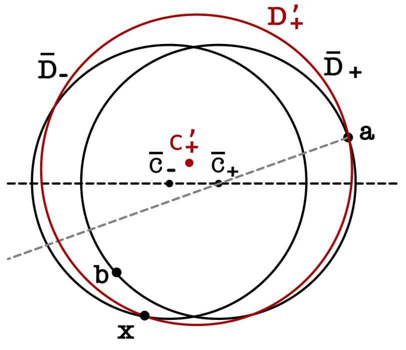


Figure 5.12: Proof of Proposition 5.8.

An immediate corollary of the above proposition is that c'_- can't be on the lower-left of c'_+ otherwise, given the fact that y is above $line(\bar{c}_-, \bar{c}_+)$ and $y \notin D'_+$ (otherwise D'_+ would have a, b, x, y which are all dominating points of $\bar{D}_-$ and $\bar{D}_+$ and so can't be the smaller disk) D'_- can't cover y.

Observation 5.5. *x and $\bar{c}_-$ should lie on different sides of $line(a, \bar{c}_+)$.*

The reason for the above observation is that because $a \notin \bar{D}_-$ and x is below $line(\bar{c}_+, \bar{c}_-)$, if x lies on the right side of $line(a, \bar{c}_+)$, axb would be a triangle for $\bar{c}_+$ which contradicts the assumption that D'_+ is smaller than $\bar{D}_+$. Also, xa can't intersect $half\text{-}line(\bar{c}_+, \bar{c}_-)$ otherwise, abx is a triangle for $\bar{c}_+$ and again contradicts that $\bar{D}_+$ is the smaller disk. Henceforth, we denote the upper and lower intersection points of $\partial\bar{D}_-$ and $\partial\bar{D}_+$ by I_1 and I_2 respectively. Also, let o be the mid-point of $\bar{c}_-\bar{c}_+$ and I_3 be the intersection of $half\text{-}line(\bar{c}_-, \bar{c}_+)$ and $\partial\bar{D}_+$.

Proposition 5.9. *c'_- can't be on the right side of c'_+.*

Proof. Suppose not and c'_- is on the right side of c'_+. First note that c'_- can't be on the lower-right of c'_+ otherwise $D'_+ \cup D'_-$ can't cover both y and x while they are smaller (because of Proposition 5.5 part 2 between x and y). So, suppose that c'_- lies on the top-right of c'_+. First note that h'_2 can't be above $line(\bar{c}_+, \bar{c}_-)$. To see why, let ϵ be the difference of the x-coordinates of c'_+ and c'_-. Also let h be minimum difference of the y-coordinates of c'_- and c'_+ in order to have h'_2 above $line(\bar{c}_+, \bar{c}_-)$. In order to keep the PCC, we need to have $h < \sqrt{\delta^2 - \epsilon^2}$ but $h \geq \sqrt{r^2 - (r - \delta - \epsilon)^2}$ (just assume that c'_+ lies on $line(\bar{c}_+, \bar{c}_-)$ and use the fact that D'_+ should cover I_3 to get the bound) which is not possible. On the other hand, h'_1 should come after y in the counter-clockwise order because D'_- needs to cover y. So, by adding h'_1 to the positive side, any disk covering x, a, h'_1 should also cover h'_2 which means its radius is bigger than $M^+[\hat{i}, \bar{j}]$ and so we can discard based on Theorem 5.1 (see Figure 5.13). $\square$

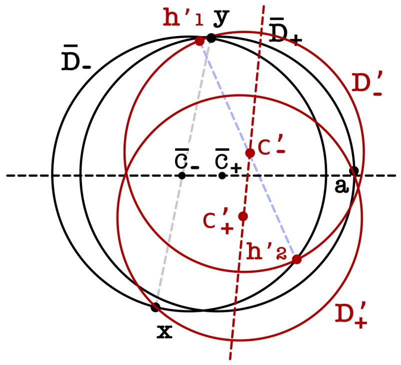


Figure 5.13: Proof of Proposition 5.9. The positions of h'_1 and h'_2 with respect to c'_- (note that we relaxed the condition that y should be covered by D'_- to make the figure clear)

We know that $y \notin D'_+$. This is because if $y \in D'_+$, D'_+ would have a, b, x, y which are the all dominating points of the solution of $M^+[\bar{i}, \bar{j}]$ and so can't be the smaller disk. On the other hand, both h'_1 and h'_2 should be in $\bar{D}_-$. Let h'_1 be the first dominating point after y in the counter-clockwise order. We recall that h'_1 should be outside D'_+ (otherwise, it contradicts Proposition 5.5). From Proposition 5.9, we know that c'_- is on the left side of c'_+. Also, note that c'_- can't be below c'_+ otherwise D'_- can't cover y (consider Proposition 5.5 between x and y). So, the intersection point of $half\text{-}line(c'_-, c'_+)$ and $\partial D'_+$ should lie on the forth quarter with respect to o. This implies that one dominating point of D'_+ namely h_1 lies after a in the angular counter-clockwise order and the other h_2 before x.

We consider two cases. First, assume that h'_1 is on the right side of $line(c'_+, c'_-)$. In this configuration, because c'_+ is on the lower-right of $\bar{c}_+$ and Proposition 5.5 for x and y, after adding h'_1 to the positive side, we always have $\triangle h_1 h_2 h'_1$ around c'_+ and so we can discard (see Figure 5.14 (a)). Now, assume that h'_1 is on the left side of $line(c'_+, c'_-)$. In this case, because a is above $\bar{c}_+$ and h_1 is after a in the order, the intersection of $half\text{-}line(h_1, c'_+)$ and D'_- should be below c'_+. Now, if h'_1 is above c'_+, we again have triangle $\triangle h'_1 h_1 h_2$ for c'_+ and thus we can discard. Otherwise, $half\text{-}line(h'_1, c'_-)$ should intersect $\partial D'_-$ inside $\bar{D}_-$ (otherwise there is no place for h'_2). Now, let t be the point on $\partial D'_-$ with same y-coordinate as c'_- on the left side of it (see Figure 5.14 (b)). Because h'_1 is below c'_-, t should lie inside D'_- but in order to have this condition $d(c'_+, c'_-)$ should be greater than δ which is not possible.

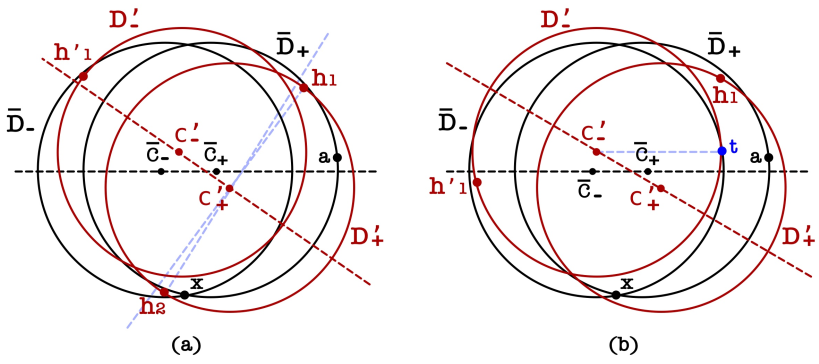


Figure 5.14: (a) h'_1 is on the right side of $line(c'_+, c'_-)$. (b) when t is inside $\bar{D}_-$, $d(c'_-, c'_+)$ is greater than δ.

Sub-case 2: $\mathbf{x \in \bar{D}_+}$.

In this sub-case, because $x \in \bar{D}_+$, y is outside of $\bar{D}_+$ and left side of $\bar{c}_-$ and indeed on $\partial convex\text{-}hull(P)$. Similar to the previous sub-section, let h'_1 be the first dominating point of D'_- that appear after y in the counter-clockwise order which is outside D'_+.

Observation 5.6. *c'_+ lies on the right side of $\bar{c}_+$.*

This is because if c'_+ is on the left side of $\bar{c}_+$, there would be no place for a and b such that ab does not intersect the $half\text{-}line(\bar{c}_+, \bar{c}_-)$ while keeping D'_+ the smaller disk. Similar to Proposition 5.9 we can assume that c'_- is on the left side of c'_+ otherwise h'_2 is covered by any disk covering a, b and x. Now, let z_1 and z_2 be the two intersection points of $\partial D'_-$ and $\partial D'_+$ where z_1 appears first in the counter-clockwise order from y. Also, let $R(z_1)$ and $R(z_2)$ be the portions of $\partial D'_-$ between two perpendicular lines from c'_- and c'_+ on $line(c'_-, c'_+)$ around z_1 and z_2 respectively (see Figure 5.15).

Proposition 5.10. *h'_1 does not intersect $R(z_1)$ and $R(z_2)$.*

Proof. We first show that h'_1 does not intersect $R(z_2)$. We proceed by contradiction and suppose that $h'_1 \in R(z_2)$. Let t_1 be the intersection point of $\partial\bar{D}_-$ and $\bar{D}'_-$ which comes first after y. In this situation, t_1 should also be in $R(z_2)$. Also, let t_2 be the first intersection point of the half-line passing from t_1 parallel to $line(c'_-, c'_+)$ and $\partial D'_+$ (see Fig 5.15 (a)). Now, t_2 is outside $\bar{D}_-$ because $t_2 \in R(z_1)$ and the positive slope of $line(c'_-, c'_+)$ (in order to have have t_1 inside $R(z_2)$). On the other hand, D'_+ has both t_2 and x. If t_2x intersect the $half\text{-}line(\bar{c}_-, \bar{c}_+)$, $|t_2x|$ should be greater than $M^+[\bar{i}, \bar{j}]$ which is contradiction. Otherwise, D'_+ should have a, b, x, t_2 which again make it bigger than $M^+[\bar{i}, \bar{j}]$ (consider a pair of disks with centers $\bar{c}_+$ and $\bar{c}_-$ and dominating points $\{a, b\}$ and $\{x, t_2\}$ respectively).

Now, we prove that h'_1 can't intersect $R(z_1)$. Let q be the last point of $R(z_2)$ in the counter-clockwise order (see Figure 5.15 (b)).

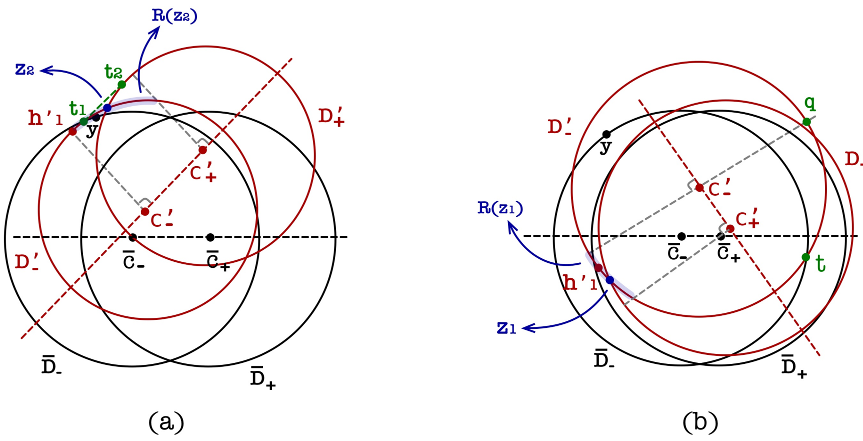


Figure 5.15: An example of configuration of points for Proposition 5.10. Note that in this figure, we relaxed the condition that x should be covered by D'_+ in order to illustrate situations where h'_1 lies inside $R(z_1)$ and $R(z_2)$.

If h'_1 lies on $R(z_1)$, h'_2 needs to lie between q and $\bar{D}_-$ on $\partial D'_-$ in order to not intersect $half\text{-}line(c'_-, c'_+)$. But, q is outside $\bar{D}_-$ because the right intersection point t of $\bar{D}_-$ and D'_- is below c'_- (this is because the PCC. Specifically, if ζ is the difference between the radii of D'_- and $\bar{D}_-$, c'_+ should lie at least ζ to the right of $\bar{c}_+$ to cover a and b. This implies that c'_- should also lie at least ζ to the right of c'_- to keep the PCC which make t below c'_-) there is no place for h'_2 inside $\bar{D}_-$ which is contradiction. □

Consider an (i, j)-partition. Let us call the (convex) cone obtained by m as its vertex and the separator half-lines from m as its sides the (i, j)-*cone*. If the positive direction of the m-line is in the cone, we say the cone is positive otherwise we say it is negative. We say two points z_1, z_2 in $P^{i,j}_-$ (resp. $P^{i,j}_+$) make a *cut* for $z_3 \in P^{i,j}_+$ (resp. $z_3 \in P^{i,j}_-$) in a positive (resp. negative) (i, j)-cone, if z_1z_2 intersects both the sides of the cone and does not separate z_3 from m in the cone. See Figure 5.16 for an example of a cone and a cut for it.

Observation 5.7. *If two point z_1 and z_2 in $P^{i,j}_+$ (resp. $P^{i,j}_-$) make a cut for a point z_3 in $P^{i,j}_-$ (resp. $P^{i,j}_+$) in a negative (resp. positive) (i, j)-cone, then if z_3 is not covered by convex-hull$(P^{i,j}_+)$ (resp. convex-hull$(P^{i,j}_-)$), then m can not be covered by convex-hull(P).*

The reason of the above observation is that if z_3 is not covered by *convex-hull*$(P^{i,j}_-)$, there is a line that separates this convex hull and m. Now, by adding the points inside the cone to the convex hull, we just move this separating line closer to m but this line can never

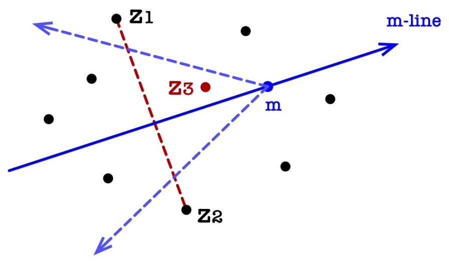


Figure 5.16: A negative cone. z_1, z_2 make a cut for z_3.

reach m.

In order to discard a sub-row or a sub-column of $M^+[\hat{i}, \bar{j}]$ according to Theorem 5.1, we need to consider different configurations of the points in the $(\hat{i}, \bar{j})$-partition. Because x can be above or below $line(\bar{c}_-, \bar{c}_+)$, in order to cover these cases, we can assume that x is below $line(\bar{c}_-, \bar{c}_+)$ but the m-line can take the both possible directions. Let h_1 be the dominating point of D'_+ on the right side of $line(c'_+, c'_-)$ and h_2 be the other one. We proceed the following cases based on the position of the m-line with respect to y and h'_1:

1) Both $\mathbf{h'_1}$ and y are on the left side of the m-line: Based on Proposition 5.10, h'_1 is on the boundary of *convex-hull*$(D'_- \cup D'_+)$ and because $x \in \bar{D}_+$, y is also on the boundary of *convex-hull*$(\bar{D}_- \cup \bar{D}_+)$. Now, because all points in P are covered by the two convex hulls, both y and h'_1 are on the boundary of *convex-hull*(P). So, if y and h'_1 are on the left side of the m-line, because $m \in$ *convex-hull*(P) and h'_1, h'_2 has different types (and so the m-line should pass between h'_1 and h'_2), when we add h'_1 to the positive side, we first need to add y to the positive side and then h'_1 (see Figure 5.17). Which means that after adding h'_1, the positive side has a, b, x, y which are the all dominating points of $\bar{D}_-$ and $\bar{D}_+$. This implies that any covering disk of them should have a radius greater than $M^+[\bar{i}, \bar{j}]$ and so, we can discard the rest of the row or column of $M^+[\hat{i}, \bar{j}]$ based on the type of h'_1.

2) $\mathbf{h'_1}$ and y are on the left and right sides of the m-line respectively: Based on Proposition 5.10 both h'_1 and y are on the convex hull of the points. Now, one of a or b should also be on *convex-hull*(P) which means it is not possible to add both of them to the positive side before either adding y or h'_1 to the positive side which is not possible.

3) $\mathbf{h'_1}$ is on the right side of the m-line: we consider two cases: first, suppose that h'_1 lies on the right side of $line(c'_+, c'_-)$. Now, if h_2 also lies on the right side of $line(h'_1, c'_+)$, then adding h'_1 makes a triangle $\triangle h'_1 h_1 h_2$ and we are done. Otherwise, h_2 should be on ∂*convex-hull*$(D'_- \cup D'_+)$ (because of Proposition 5.10) and so the points h'_1, y and h_2 is on

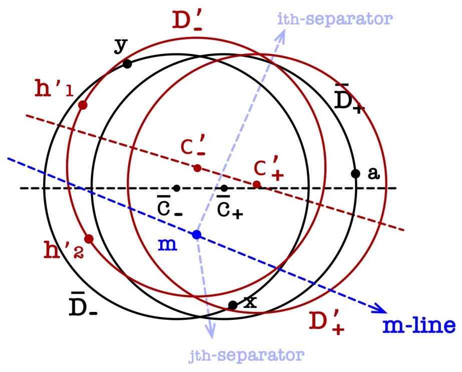
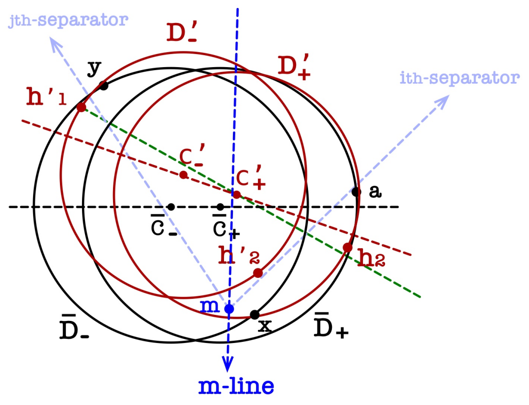
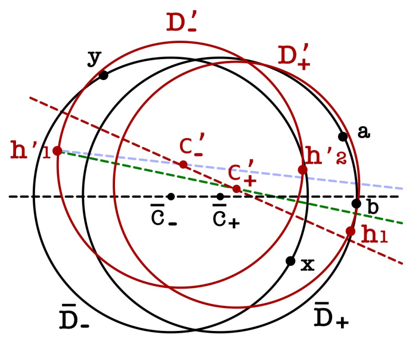


Figure 5.17: In order to add h_1' to the positive side, we first need to add y to the positive side.

∂*convex-hull*(P). Now, if h_2 and h_2' has different types, h_2 and h_1' should have a same type (because h_1' and h_2' had different types based on our assumption) and so on a same side of the m-line. In this case, because y lies on the right side of $line(h_2, h_1')$ and both h_1' and h_2 lie on the right side of the m-line, and also because h_2' should lie on the left side of the m-line, y should also lie on the right side of the m-line. In this situation in order to add h_2 to the positive side, we need to add h_1' and y to the positive side first which is not possible. This argument implies that h_2' and h_2 have a same type. Because h_2 needs to be added to the positive side before h_2', m should be on the left side of $line(h_2, h_2')$ but in this situation, after adding h_1' to the positive side, we would have negative cone $h_1'mh_2$ and $h_1'h_2$ makes a cut for h_2' (h_2' lies on the right side of $line(h_1', c_-')$ to satisfy Proposition 5.5) which implies that any disk covering h_1' and h_2 should also cover h_2' and we can discard (see Figure 5.18 (a)).



(a)



(b)

Figure 5.18: h'_1 is on the right side of the m-line. (a) h'_1 is on the right side of $line(c'_+, c'_-)$. (b) h'_1 is on the left side of $line(c'_+, c'_-)$.

Now, suppose that h'_1 is on the left side of $line(c'_+, c'_-)$. Again, if h_1 is on the left side of $line(h'_1, c'_+)$, adding h'_1 makes triangle $\triangle h'_1 h_1 h_2$ and we can discard. So, we assume that h_1 is on the right side of $line(h'_1, c'_+)$. Note that here h'_2 can't lie above c'_- because of PCC (similar to the argument of point t in the proof of Proposition 5.10) which implies c'_- is above c'_+ (otherwise, there is no place for h'_2). Now, if $a \in \bar{D}_-$, by adding h'_1 to the positive side, any disk smaller than $\bar{D}_+$ covering h_1, a and h'_1 should also cover h'_2. This is because h'_2 is on the left side of $line(h'_1, c'_-)$ and the portion of the disk in the first quarter of o is outside of $\bar{D}_+$ (in order to cover a) and so we can discard. Let us assume that $a \notin \bar{D}_-$ and so it lies on ∂*convex-hull*(P). On the other hand, because h_1 is inside $\bar{D}_+$, b should also lie on *convex-hull*(P) (see Figure 5.18 (b)). Now, if b is on the left side of $line(y, \bar{c}_-)$, x and y make triangle for $\bar{c}_-$ with both a, b (because $x \in \bar{D}_+$) and so, we could discard in the initial search. But if b lies on the right side of $line(y, \bar{c}_-)$, because h'_1 is after y in the counter-clockwise order, it is not possible to cover three points h'_1, a, b with a radius smaller than $M^+[\hat{i}, \bar{j}]$ and so again we can discard.

Chapter 6

Conclusion and Future Works

In this thesis, we studied several center location problems for which we have extra conditions on the location of the centers. In the first part, we studied two problems: finding beacon kernel points in simple polygons and the particle transmitting problem in polygonal domains. In both of these problems, we need to consider the structures of the underlying space (the given polygonal region) for establishing a beacon/repulsor. We provided the first sub-quadratic time algorithm for the discrete beacon kernel problem and used this algorithm to obtain the beacon kernel points on the boundary of simple polygons. Also, we showed that the particle transmitting problem can be solved in polynomial time. Regarding the beacon kernel problem, a natural generalization of the discrete beacon kernel problem is when we replace the given set of point $\mathcal{X}$ by a given set of arbitrary line segments or half-lines $\mathcal{S}$ (they may not completely lie inside the given polygon). In this generalization, the objective is computing the *kernel segments* of each segment or half-line $S \in \mathcal{S}$ that is the maximal segments in $S \cap Ker(P)$. Here, we show if we can solve this generalization in sub-quadratic time, we also can solve the beacon kernel problem in sub-quadratic time. First, note that each edge of $\partial Ker(P)$ is a subset of either an edge of ∂P or an extension of a reflex vertex in $\mathcal{R}$ (see Proposition 2.3). Therefore, if we consider $\mathcal{S}_1$ as the set of extensions of $\mathcal{R}$ and compute the set $\mathcal{S}_1'$ of its kernel segments, each edge of $\partial Ker(P)$ would be a segment in $\mathcal{S}'$. A key point here is that two segments $s_1, s_2 \in \mathcal{S}_1$ can not intersect themselves on their interior. This is because for s_1 (resp. s_2), the portion of the segment in one side of the intersection lies in the dead wedge of the vertex that generated s_2 (resp. s_1). So, the intersection points of the segments in $\mathcal{S}_1'$ are those vertices of $\partial Ker(P)$ not incident to an edge in ∂P. In addition, consider $\mathcal{S}_2$ as the set of edges in ∂P and compute its kernel segments $\mathcal{S}_2'$ (note that each edge has at most one kernel segment). In [11], Biro proved that the complexity of the beacon kernel of a simple polygon is linear. So, by computing the intersection points of the segments in $\mathcal{S}_1' \cup \mathcal{S}_2'$ (by using the sweep-plane algorithm [63]) we can get the vertices of $\partial Ker(P)$ and thus $Ker(P)$ itself.

In the transmitting particle problem, we were interested in transmitting particles to the target point by using only one repulsor. Similar to the concept of routing by a sequence of beacons, we can define routing by a sequence of repulsors as follows: for a pair of points (p, t) in a polygonal region, we say p can be routed by sequence $(r_1, r_2, \ldots, r_k)$ of k repulsors to t via targets $(p = t_1, \ldots, t_{k+1} = t)$ if for each $1 \leq i \leq k$, r_i sends t_i to t_{i+1}. Thus, a generalization of the particle transmitting problem is determining the points in the given polygonal domain that can be send to a target point by a sequence of k repulsors.

In the second part, we first studied the PCkCP on paths in both weighted and unweighted case. A direct generalization of this problem is considering the PCkCP on trees. Its is straightforward to see that the greedy approach for the feasibility test is still works for trees. The potential strategy to get a sub-quadratic time algorithm for the PCkCP on trees (in both weighted and unweighted case) is first considering the problem for *stems* [66]. Each stem is a path called the backbone of the stem and a set of edges hanging from it. Next follow the approach of [28] to get the stem decomposition of the given tree and recursively process the stems from the leafs of the tree and replace each processed stem by an edge to form new stems to get a sub-quadratic time algorithm to solve the PCkCP on trees.

Lastly, we studied the proximity constrained 2-center as an special case of the PCkCP in the plane. One direct generalization is considering the weighted case problem of PCTCP as we defined the weighted distance in Chapter 4. In this problem, each of the demand points have a positive weight and the problem is computing two centers satisfying PCC that minimize the maximum of the weighted distance of a demand point to its closest center. If we consider the general PCkCP in the plane, we see that if δ tends to zero, the problem turns to the minimum enclosing disk problem which can be solved in linear time [51]. If δ tends to infinity, the problem turns to the unconstrained k-center problem which we know it is NP-hard and can not be approximated by a ratio better than two. Also, if both k goes to infinity and δ goes to zero, the problem turns to the Euclidean Steiner tree problem which has a PTAS approximation algorithm. Therefore, an interesting future work is determining the hardness of approximation algorithm for the PCkCP in the plane when k and δ are not such extreme values.

Bibliography

[1] Agarwal PK, Procopiuc CM. Exact and approximation algorithms for clustering. *Algorithmica.* 2002 Jun 1;33(2):201-26

[2] Agarwal PK, Sharir M. Planar geometric location problems. *Algorithmica.* 1994 Feb;11(2):185-95.

[3] Agarwal PK, Sharir M. Efficient algorithms for geometric optimization. *ACM Computing Surveys (CSUR).* 1998 Dec 1;30(4):412-58.

[4] Ahlberg M, Vlassov V, Yasui T. Router placement in wireless sensor networks. In *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems* 2006 Oct 9 (pp. 538-541). IEEE.

[5] Al-Karaki JN, Kamal AE. Routing techniques in wireless sensor networks: a survey. *IEEE wireless communications.* 2004 Dec 20;11(6):6-28.

[6] Arora S. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings 38th Annual Symposium on Foundations of Computer Science* 1997 Oct 20 (pp. 554-563). IEEE.

[7] Bae SW, Shin CS, Vigneron A. Tight bounds for beacon-based coverage in simple rectilinear polygons. *Computational Geometry.* 2019 Jul 1;80:40-52.

[8] Barilan J, Kortsarz G, Peleg D. How to allocate network centers. *Journal of Algorithms.* 1993 Nov 1;15(3):385-415.

[9] Bhattacharya B, Mozafari A, Shermer TC. An efficient algorithm for the proximity connected two center problem. In *International Workshop on Combinatorial Algorithms* 2022 (pp. 199-213). Springer, Cham.

[10] Bhattacharya B, Kameda T, Mozafari A. A Sub-quadratic Time Algorithm for the Proximity Connected k-center Problem on Paths via Modular Arithmetic. *34th Canadian Conference on Computational Geometry,* 2022

[11] Biro M. Beacon-based routing and guarding (Doctoral dissertation, State University of New York at Stony Brook).

[12] Biro M, Gao J, Iwerks J, Kostitsyna I, Mitchell JS. Beacon-based routing and coverage. In *21st Fall Workshop on Computational Geometry (FWCG 2011)* 2011 Nov 4.

[13] Biro M, Iwerks J, Kostitsyna I, Mitchell JS. Beacon-based algorithms for geometric routing. In *Workshop on Algorithms and Data Structures* 2013 Aug 12 (pp. 158-169). Springer, Berlin, Heidelberg.

[14] Biro M, Gao J, Iwerks J, Kostitsyna I, Mitchell JS. Combinatorics of beacon-based routing and coverage. *25th Canadian Conference on Computational Geometry*, 2013

[15] Bose P, Shermer TC. Attraction-convexity and normal visibility. *Computational Geometry.* 2021 Jun 1;96:101748.

[16] Bose P, Shermer TC. Gathering by repulsion. *Computational Geometry.* 2020 Oct 1;90:101627.

[17] Chan TM. More planar two-center algorithms. *Computational Geometry.* 1999 Sep 1;13(3):189-98.

[18] Chazelle B. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry.* 1991 Sep;6(3):485-524.

[19] Chazelle B, Edelsbrunner H, Grigni M, Guibas L, Hershberger J, Sharir M, Snoeyink J. Ray shooting in polygons using geodesic triangulations. *Algorithmica.* 1994 Jul;12(1):54-68.

[20] Chin F. Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE Transactions on Computers.* 1983 Dec 1(12):1203-7.

[21] Cho K, Oh E. Optimal algorithm for the planar two-center problem. arXiv preprint arXiv:2007.08784. 2020 Jul 17

[22] Choi J, Ahn HK. Efficient planar two-center algorithms. *Computational Geometry.* 2021 Apr 2:101768.

[23] Cole R. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM (JACM).* 1987 Jan 1;34(1):200-8.

[24] Berg MD, Kreveld MV, Overmars M, Schwarzkopf O. Computational geometry. In *Computational geometry* 1997 (pp. 1-17). Springer, Berlin, Heidelberg.

[25] Drezner Z. The planar two-center and two-median problems. *Transportation Science.* 1984 Nov;18(4):351-61.

[26] Drezner Z, Hamacher HW, editors. *Facility location: applications and theory.* Springer Science & Business Media; 2004 May 3.

[27] Eppstein D. Faster construction of planar two-centers. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 131–138, 1997.

[28] Frederickson GN. Parametric search and locating supply centers in trees. In *Workshop on Algorithms and Data Structures* 1991 Aug 14 (pp. 299-319). Springer, Berlin, Heidelberg.

[29] Frederickson GN. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms* 1991 Mar 1 (pp. 168-177).

[30] Gowda I, Kirkpatrick D, Lee D, Naamad A. Dynamic voronoi diagrams. *IEEE Transactions on Information Theory.* 1983 Sep;29(5):724-31.

[31] Gudmundsson J, Haverkort H, Park SM, Shin CS, Wolff A. Facility location and the geometric minimum-diameter spanning tree. *Computational Geometry.* 2004 Jan 1;27(1):87-106.

[32] Hakimi SL. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations research.* 1965 Jun;13(3):462-75.

[33] Hershberger J. A faster algorithm for the two-center decision problem. *Information processing letters.* 1993 Aug 9;47(1):23-9.

[34] Hershberger J, Suri S. Efficient computation of Euclidean shortest paths in the plane. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science* 1993 Nov 3 (pp. 508-517). IEEE.

[35] Hershberger J, Suri S. Off-line maintenance of planar configurations. *Journal of algorithms.* 1996 Nov 1;21(3):453-75.

[36] Huang CH. Some problems on radius-weighted model of packet radio network, Doctoral dissertation, Ph. D. Dissertation, Dept. of Comput. Sci., Tsing Hua Univ., Hsinchu, Taiwan, 1992.

[37] Huang PH, Tsai YT, Tang CY. A near-quadratic algorithm for the alpha-connected two-center problem. *Journal of information science and engineering.* 2006 Nov 1;22(6):1317.

[38] Huang PH, Te Tsai Y, Tang CY. A fast algorithm for the alpha-connected two-center decision problem. *Information Processing Letters.* 2003 Feb 28;85(4):205-10.

[39] Hwang RZ, Lee RC, Chang RC. The slab dividing approach to solve the Euclidean P-Center problem. *Algorithmica.* 1993 Jan 1;9(1):1-22.

[40] Jeger M, Kariv O. Algorithms for finding *p*-centers on a weighted tree (for relatively small P). *Networks.* 1985 Sep;15(3):381-9.

[41] Kariv O, Hakimi SL. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics.* 1979 Dec;37(3):513-38.

[42] Karp B, Kung HT. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking* 2000 Aug 1 (pp. 243-254).

[43] Katz MJ, Sharir M. An expander-based approach to geometric optimization. *SIAM Journal on Computing.* 1997 Oct;26(5):1384-408.

[44] Khuller S, Sussmann YJ. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics.* 2000;13(3):403-18.

[45] Kirkpatrick D. Optimal search in planar subdivisions. *SIAM Journal on Computing.* 1983 Feb;12(1):28-35.

[46] Kim YD, Yang YM, Kang WS, Kim DK. On the design of beacon based wireless sensor network for agricultural emergency monitoring systems. *Computer standards and interfaces.* 2014 Feb 1;36(2):288-99.

[47] Kouhestani B, Rappaport D, Salomaa K. Routing in a polygonal terrain with the shortest beacon watchtower. *Computational Geometry.* 2018 Mar 1;68:34-47.

[48] Kouhestani B, Rappaport D, Salomaa K. On the Inverse Beacon Attraction Region of a Point. *27th Canadian Conference on Computational Geometry*, 2015.

[49] Kostitsyna I, Kouhestani B, Langerman S, Rappaport D. An optimal algorithm to compute the inverse beacon attraction region. arXiv preprint arXiv:1803.05946. 2018 Mar 15.

[50] Lim A, Rodrigues B, Wang F, Xu Z. k-Center problems with minimum coverage. *Theoretical Computer Science.* 2005 Feb 28;332(1-3):1-7.

[51] Megiddo N. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM journal on computing.* 1983 Nov;12(4):759-76.

[52] Megiddo N. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM).* 1983 Oct 1;30(4):852-65.

[53] Megiddo N, Supowit KJ. On the complexity of some common geometric location problems. *SIAM journal on computing.* 1984 Feb;13(1):182-96.

[54] Megiddo N. Partitioning with two lines in the plane. *Journal of Algorithms.* 1985 Sep 1;6(3):430-3.

[55] Mozafari A, Shermer TC. Transmitting particles in a polygonal domain by repulsion. In *International Conference on Combinatorial Optimization and Applications 2018* Dec 15 (pp. 495-508). Springer, Cham.

[56] Overmars MH, Van Leeuwen J. Maintenance of configurations in the plane. *Journal of computer and System Sciences.* 1981 Oct 1;23(2):166-204.

[57] Pach J, editor. *New trends in discrete and computational geometry.* Springer Science & Business Media; 2012 Dec 6.

[58] Patel M, Chandrasekaran R, Venkatesan S. Energy efficient sensor, relay and base station placements for coverage, connectivity and routing. In *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference*, 2005. 2005 Apr 7 (pp. 581-586). IEEE.

[59] Sharir M. A near-linear algorithm for the planar 2-center problem. *Discrete and Computational Geometry.* 1997 Sep 1;18(2):125-34.

[60] Shermer T. A combinatorial bound for beacon-based routing in orthogonal polygons. *Journal of Computational Geometry.* 2022 Apr 20;13(1):13-51.

[61] Shmoys DB, Tardos É, Aardal K. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* 1997 May 4 (pp. 265-274).

[62] Sojka E. A simple and efficient algorithm for sorting the intersection points between a Jordan curve and a line. In *Fifth international conference in Central Europe in computer graphics and visualisation* 97 (Plzeň, February 10-14, 1997) 1997 (pp. 524-533).

[63] Toth, Csaba D., Joseph O'Rourke, and Jacob E. Goodman. Handbook of discrete and computational geometry. Chapman and Hall/CRC, 2017.

[64] van Goethem A, Kostitsyna I, Verbeek K, Wulms J. Repulsion region in a simple polygon. In *Abstr. 36th European Workshop on Computational Geometry (EuroCG)* (Vol. 73, pp. 1-73).

[65] Wang H. On the Planar Two-Center problem and intersection hulls. arXiv preprint arXiv:2002.07945. 2020 Feb 19.

[66] Wang H, Zhang J. An $O(n \log n)$-time algorithm for the k-Center problem in trees. *SIAM Journal on Computing.* 2021;50(2):602-35.