

Two Optimization Problems for Unit Disks

Sergio Cabello^{*1} and Lazar Milinković²

¹ FMF, University of Ljubljana, and Institute of Mathematics, Physics and Mechanics, Slovenia

² FMF and FRI, University of Ljubljana, Slovenia

Abstract

To be done

1 Introduction

In this paper we consider two geometric optimization problems in the plane where unit disks play a prominent role. For both problems we discuss efficient algorithms to solve them, provide an implementation of these algorithms, and present experimental results on the implementation.

The first problem we consider is computing a shortest-path tree in the (unweighted) intersection graph of unit disks. The input to the problem is a set \mathcal{D} of n unit-radius disks, each disk represented by its center. The corresponding (unit-disk) intersection graph has a vertex for each disk, and an edge connecting two disks D and D' of \mathcal{D} whenever D and D' intersect. An alternative, more convenient point of view, is to take as vertex set the set of centers of the disks, denoted by P , and connecting two points p and q of P whenever the Euclidean distance $|pq|$ is at most 2. Given a root $r \in P$, the task is to compute a shortest-path tree from r in this graph.

The second problem we consider is the so-called minimum-separation problem. The input is a set \mathcal{D} of n unit-radius disks in the plane and two points s and t not covered by any disks of \mathcal{D} . We say that \mathcal{D} *separates* s and t if each path in the plane from s to t intersects some disk of \mathcal{D} . The task is to find the minimum cardinality subset of \mathcal{D} that separates s and t . Formally, we want to solve

$$\begin{array}{ll} \min & |\mathcal{D}'| \\ \text{s.t.} & \mathcal{D}' \subset \mathcal{D} \\ & \mathcal{D}' \text{ separates } z \text{ and } z'. \end{array}$$

Unit disks are the most standard model used for wireless sensors [] and networks []. They provide an appropriate trade off between simplicity and accuracy. While other models [] are more accurate, working with them is much more challenging.

Some motivating references

Our contribution. There are two algorithms to compute shortest-path trees in unit disk graphs $O(n \log n)$ worst-case time [2, 4]. Here we report on an implementation of a modification of the algorithm in [2], and we provide two other obvious alternative to compare the results. The algorithm of [?] is substantially harder to implement and it is clear that it has worse constants hidden in the O -notation.

The minimum-separation problem is studied by Cabello and Giannopoulos [1]. No obvious that this problem can be solved exactly in polynomial time.

* Supported by the Slovenian Research Agency, program P1-0297 and project L7-5459.



licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Assumptions. Henceforth P will be the set of centers of \mathcal{D} . All the computation will be around P . In particular, we assume that P is known. (For the shortest path problem, one could possibly consider weaker models based on adjacencies.)

Let $G(P)$ be the graph with vertex set P and an edge between two points $p, q \in P$ are connected whenever their distance $|pq|$ is at most 1. We drop the dependency on P and just write G instead of $G(P)$.

For simplicity of exposition we assume that $G(P)$ is connected.

Organization of the paper. In Section 2 we discuss the theoretical algorithms for both problems and their guarantees. In Section 3 we discuss the implementations and in Section 4 we present our experimental results.

2 Description of algorithms

2.1 Shortest-path tree in unit-disk graphs

We describe here the algorithm of Cabello and Jeřčič [2] to compute a shortest path tree in $G(P)$ from a given source point $s \in P$. The pseudocode of the algorithm is in Figure 1.

As it is usually done for shortest path algorithms we use tables $\text{dist}[\cdot]$ and $\pi[\cdot]$ indexed by the points of P to record, for each point $p \in P$, the distance $d(s, p)$ and the ancestor of p in a shortest (s, p) -path. We start by computing the Delaunay triangulation $DT(P)$ of P . We then proceed in rounds for increasing values of i , where at round i we find the set W_i of points at distance exactly i in $G(P)$ from the source s . We start with $W_0 = \{s\}$. At round i , we use $DT(P)$ to grow a neighbourhood around the points of W_{i-1} that contains W_i . More precisely, we consider the points adjacent to W_{i-1} in $DT(P)$ as candidate points for W_i . For each candidate point that is found to lie in W_i , we also take its adjacent vertices in $DT(P)$ as new candidates to be included in W_i . For checking whether a candidate point p lies in W_i we use a data structure to find the nearest neighbour of p in W_{i-1} , denoted by $\text{NN}(W_{i-1}, p)$. Such data structure is just a point location data structure in the Voronoi diagram of W_{i-1} . Similarly, the shortest path tree is constructed by connecting each point of W_i to its nearest neighbour in W_{i-1} . See Figure 1 for the eventual algorithm UNWEIGHTEDSHORTESTPATH.

We would like to emphasize a careful point that we employ to achieve the running time $\mathcal{O}(n \log n)$. For any point p , let $D(p, 1)$ denote the disk of radius 1 centered at p . In lines 16 and 17 of the algorithm, we check whether p is at distance at most 1 from *some* point in W_{i-1} , namely its nearest neighbour in W_{i-1} . Checking whether p is at distance at most 1 from $\pi(q)$ (or q when $q \in W_{i-1}$) would lead to a potentially larger running time. Thus, we do not grow each disk $D(w, 1)$ independently for each $w \in W_{i-1}$, but we grow the whole region $\bigcup_{w \in W_{i-1}} D(w, 1)$ at once. Growing each disk $D(w, 1)$ separately would force us to check the same edge qp of $DT(P)$ several times, once for each $w \in W_{i-1}$ such that $q \in D(w, 1)$.

► **Lemma 2.1.** *The algorithm UNWEIGHTEDSHORTESTPATH(P, S) takes $\mathcal{O}(n \log n)$ time, where n is the size of P .*

► **Theorem 2.2.** *Let P be a set of n points in the plane and let s be a point from P . In time $\mathcal{O}(n \log n)$ we can compute a shortest path tree from s in the unweighted graph $G(P)$.*

Cabello and Jeřčič [2] show that the algorithm takes $\mathcal{O}(n \log n)$ time in the worst case. In general, if we use a data structure for nearest neighbors with construction time $T_c(m)$ and query time $T_q(m)$, then the algorithm takes $\mathcal{O}(n \log n + T_c(n) + nT_q(n))$ time.

Standard data structures have construction time $T_c(m) = \mathcal{O}(m \log m)$ and query time $T_q(m) = \mathcal{O}(\log m)$, and thus the algorithm takes $\mathcal{O}(n \log n)$ time.

```

UNWEIGHTEDSHORTESTPATH( $P, s$ )
1  build the Delaunay triangulation  $DT(P)$ 
2  for  $p \in P$ 
3       $dist[p] = \infty$ 
4       $\pi[p] = \text{NIL}$ 
5   $dist[s] = 0$ 
6   $W_0 = \{s\}$ 
7   $i = 1$ 
8  while  $W_{i-1} \neq \emptyset$ 
9      build data structure for nearest neighbour queries in  $W_{i-1}$ 
10      $Q = W_{i-1}$  // candidate points
11      $W_i = \emptyset$ 
12     while  $Q \neq \emptyset$ 
13          $q$  an arbitrary point of  $Q$ 
14         remove  $q$  from  $Q$ 
15         for  $qp$  edge in  $DT(P)$ 
16              $w = \text{NN}(W_{i-1}, p)$ 
17             if  $dist[p] = \infty$  and  $\|p - w\| \leq 1$ 
18                  $dist[p] = i$ 
19                  $\pi[p] = w$ 
20                 add  $p$  to  $Q$ 
21                 add  $p$  to  $W_i$ 
22      $i = i + 1$ 
23 return  $dist[\cdot]$  and  $\pi[\cdot]$ 

```

■ **Figure 1** Algorithm to compute a shortest path tree in the unweighted case.

76 It is obvious that, when computing the shortest path tree from several sources, we only
77 need to compute the Delaunay triangulation once.

78 Let us mention two obvious alternatives that we use in our comparison. Build the graph
79 and run BFS. Build a grid like structure and use it each time to test candidate edges.

80 2.2 Minimum separation with unit-disk

81 Generic algorithm

82 Cabello and Giannopoulos [1] present an algorithm for the minimum separation problem
83 that in the worst-case runs in cubic-time. The algorithm has one feature that is both an
84 advantage and a disadvantage: it works for any reasonable shapes, like segments or ellipses,
85 and not just unit disks. This means that it is very generic, which is good, but it cannot
86 exploit any properties of disks or unit disks.

87 The algorithm can be slightly simplified for disks, as we explain below. Let us first
88 introduce some notation. Each walk W in the graph $G(P)$ defines a planar curve in the
89 obvious way: we connect the points of P with segments in the order given by W . We will
90 relax the notation slightly and denote also by W the curve itself. For any spanning tree T of
91 $G(P)$ and any edge $e \in E(G(P)) \setminus E(T)$, let $cycle(T, e)$ be the unique cycle in $T + e$. Finally,

4 Two Optimization Problems for Unit Disks

for any walk in $G(P)$, let $\text{cr}_2(st, W)$ be the modulo 2 value of the number of crossings between the segment st and (the curve defined by) W .

The following property is implicit in [1] and explicit in [3]:

Let T be any spanning tree of $G(P)$. The set of unit disks with centers in P separate s and t if and only if there exists some edge $e \in E(G(P)) \setminus E(T)$ such that $\text{cr}_2(st, \text{cycle}(T, e)) = 1$.

A consequence of this is that a minimum separation amounts to find a shortest cycle W in $G(P)$ such that W crosses st an odd number of times. Moreover, one can show that we can restrict our search to the family of generating cycles. Indeed, for each root r let us fix a shortest-path tree T_r from r in $G(P)$. Then, we can restrict our search to

$$\{\text{cycle}(T_r, e) \mid r \in P, e \in E(G(P)) \setminus E(T_r)\}.$$

This follows from the co-called 3-path condition; see [1] for the details.

The values $\text{cr}_2(st, \text{cycle}(T_r, e))$ can be computed in amortized constant time with a bit of bookkeeping. Consider a fixed r . For each point $p \in P$ we store $N[p]$ as the parity of the number of crossings of the path in T_r from r to p . When p is not the root, the value $N[p]$ can be computed from the value of its parent $\pi[p]$ in T_r using that $N[p] = N[\pi[p]] + \text{cr}_2(st, p\pi[p])$. For each edge $pq \in E(G)$ we then have $\text{cr}_2(st, \text{cycle}(T_r, pq)) = N[p] + N[q] + \text{cr}_2(st, pq)$ because the crossings in the common part, from r to the lowest common ancestor of p and q , cancel out modulo 2. The final resulting algorithm, denoted as **GENERICMINIMUMSEPARATION**, is given in Figure 2.

```

GENERICMINIMUMSEPARATION( $P, s, t$ )
1   $best = \infty$  // length of the best separation so far
2  for  $s \in P$ 
3      ( $dist[\ ], \pi[\ ]$ ) = shortest path tree from  $r$  in  $G(P)$ 
      // Compute  $N[\ ]$ 
4       $N[r] = 0$ 
5      for  $p \in P \setminus \{r\}$  in non-decreasing values of  $dist[p]$ 
6           $q = \pi[p]$ 
7           $N[p] = N[q] + \text{cr}_2(st, pq) \pmod{2}$ 
8      for  $pq \in E(G(P))$ 
9          if  $N[p] + N[q] + \text{cr}_2(pq, st) \pmod{2} = 1$ 
10              $best = \min\{best, dist[p] + dist[q] + 1\}$ 
11  return  $best$ 

```

■ **Figure 2** Adaptation of the generic algorithm to compute the minimum separation for unit disks.

Questions

for the algorithm. Is this the algorithm that is implemented? Does it check whether $pq \in E(T_r)$? Do we compute $N(p)$ with the tree

Without loss of generality, in our theoretical discussion we will assume that $z = (0, 0)$ and $z' = (0, s)$. Therefore, zz' is a segment, henceforth denoted by σ , contained in the y -axis.

110 3 Implementation

111 3.1 Shortest-path tree in unit-disk graphs

112 3.2 Minimum separation with unit-disk

113 4 Experimental results

114 4.1 Shortest-path tree in unit-disk graphs

115 4.2 Minimum separation with unit-disk

116 5 Conclusions

117 6 Building block

118 The following ideas are probably folklore. See [?].

► **Lemma 6.1.** *Let P be a set of n weighted points in the plane. In $O(n \log n)$ time we can construct a data structure that, for a query point q , it finds in time $O(\log^2 n)$ a point in*

$$\arg \min \{w_p \mid p \in P, |pq| \leq 1\}.$$

119 **Proof.** We sort the points of P by non-decreasing weight, build a balanced binary search
120 tree \mathcal{T} with n leaves, and place the points of P in the leaves of \mathcal{T} such that the order arising
121 from the search tree and the sorting of P match.

122 For every node ν of \mathcal{T} :

- 123 ■ Let $P(\nu)$ denote the set of points stored at the subtree rooted at ν . We refer to $P(\nu)$ as
124 a *canonical subset*.
- 125 ■ Let $U(\nu)$ be the union of unit disks centered at the points of $P(\nu)$.
- 126 ■ We build a data structure to decide whether a given query point is contained in $U(\nu)$. A
127 way to do this is to construct a point-location data structure for the Voronoi diagram
128 of $P(\nu)$. Given a query point q , we find a closest neighbor from $P(\nu)$ by locating a cell
129 c of the Voronoi diagram that contains q . We then check whether q is at most at unit
130 distance from the site defining c . If the points of $P(\nu)$ are sorted (lexicographically), then
131 the Voronoi diagram of $P(\nu)$ can be build in linear time ?? The preprocessing time for
132 node ν is $O(|P(\nu)|)$ and a query can be answered in $O(\log |P(\nu)|) = O(\log n)$ time.

133 This finishes the description of the construction of the data structure. Let us analyze the
134 building time. We sort the points of P lexicographically at the root and for each node ν we
135 get the points $P(\nu)$ sorted from its parent. Since the canonical subsets at each level of the
136 tree are disjoint, at each level of the tree we spend $O(n)$ time. Since \mathcal{T} is balanced, there are
137 $O(\log n)$ levels and we spend in total $O(n \log n)$ time to build the data structure.

Given a point q we first check whether q lies in $U(r)$ for the root r of \mathcal{T} . If q is not in $U(r)$, then no point of P is close enough to q . Otherwise we set $\nu = r$ and proceed with a top-to-down search in the tree \mathcal{T} until we reach a leaf. When ν is not a leaf, let ν_ℓ and ν_r be the left and right children of ν , respectively. If q is in $U(\nu_\ell)$, we then proceed with the search in the subtree at ν_ℓ by setting $\nu = \nu_\ell$, otherwise we proceed the search in the right subtree setting $\nu = \nu_r$. At each point of the search, we maintain the following invariant:

$$P(\nu) \cap \arg \min \{w_p \mid p \in P, |pq| \leq 1\} \neq \emptyset.$$

6 Two Optimization Problems for Unit Disks

139 The search path in \mathcal{T} has $O(\log n)$ nodes and, for each node in the path, spend $O(\log n)$
 140 time to decide whether q lies in $U(v)$. The total query time is thus $O(\log^2 n)$. ◀

Consider the following problem for two weighted point sets A and B .

$$\begin{aligned} \Phi(A, B) &:= \min w_a + w_b \\ \text{s.t. } &a \in A, b \in B \\ &|ab| \leq 1. \end{aligned}$$

141 ► **Lemma 6.2.** *Let A and B be sets of at most n weighted points in the plane. In $O(n \log^2 n)$
 142 time we can compute $\Phi(A, B)$.*

Proof. We build the data structure of the previous Lemma for B . For each $a \in A$, we query the data structure to obtain

$$b^*(a) \in \arg \min \{w_b \mid b \in B, |ab| \leq 1\}.$$

143 Then we find the $a \in A$ minimizing the sum $w_a + w_{b^*(a)}$.

144 We need $O(n \log n)$ time to construct the data structure and $O(\log^2 n)$ time per query.
 145 The result follows. ◀

Let σ be a segment in the plane. Without loss of generality, we will assume that s lies on the y -axis with endpoints $(0, 0)$ and $(0, s)$. Let A be a set of points with negative x -coordinates and let B be a set of points with positive x -coordinates. Each point p of $A \cup B$ has a weight w_p . We are interested in minimizing $w_a + w_b$ over all pairs $(a, b) \in A \times B$ whose segment ab have length at most 1 and intersects s . Thus, we want to find

$$\begin{aligned} \Phi_\sigma(A, B) &:= \min w_a + w_b \\ \text{s.t. } &a \in A, b \in B \\ &|ab| \leq 1 \\ &ab \text{ intersects } \sigma. \end{aligned}$$

For every point $a \in A$ we define the sets

$$\begin{aligned} B(a) &= \{b \in B \mid ab \text{ intersects } \sigma\}, \\ B_{\leq 1}(a) &= \{b \in B \mid ab \text{ intersects } \sigma \text{ and } |ab| \leq 1\} = \{b \in B(a) \mid |ab| \leq 1\} \end{aligned}$$

and the optimization problem

$$\Phi_\sigma(a, B) = w_a + \min \{w_b \mid b \in B_{\leq 1}(a)\}.$$

We thus have

$$\Phi_\sigma(A, B) = \min_{a \in A} \Phi_\sigma(a, B).$$

146 We first provide a data structure to obtain the sets $B(\cdot)$ compactly.

147 ► **Lemma 6.3.** *There is a family $\{B_1, \dots, B_t\}$ of subsets of B and a data structure $\mathcal{D}(B)$
 148 with the following properties*

- 149 ■ $\sum_{i=1}^t |B_i| = O(n \log n)$;
- 150 ■ $\mathcal{D}(B)$ has size $O(n \log n)$ and can be constructed in $O(n \log n)$ time;

- 151 ■ for each point a with negative x -coordinate, there is a subset of indices $I(a) \subset \{1, \dots, t\}$
 152 such that $|I(a)| = O(\log^2 n)$ and $B(a)$ is the disjoint union of $\{B_i\}_{i \in I(a)}$;
 153 ■ for each query point a with $a_x < 0$, the data structure $\mathcal{D}(B)$ returns $I(a)$ in $O(\log^2 n)$
 154 time.

155 **Proof.** It is convenient to use point-line duality. We use the precise duality described
 156 in BKOS Chapter XXX: the non-vertical line $\ell \equiv y = mx + c$ is mapped to the point
 157 $\ell^* = (m, -c)$.

Let \mathbb{L} be the set of non-vertical lines. Let σ^* be the set of points dual to non-vertical lines that intersect s . Thus

$$\sigma^* = \{\ell^* \mid \ell \in \mathbb{L}, \ell \cap \sigma \neq \emptyset\}$$

In the dual space, the set σ^* is the horizontal slab

$$\sigma^* = \{(m, -c) \in \mathbb{R}^2 \mid 0 \leq c \leq s\}.$$

For every point $b \in B$, let L_b^* be the set of points dual to the lines through b that intersect σ :

$$L_b^* = \{\ell^* \mid \ell \in \mathbb{L}, b \in \ell, \text{ and } \sigma \cap \ell \neq \emptyset\}.$$

158 In the dual space, L_b^* is a segment with endpoints $(\varphi_1(b), 0)$ and $(\varphi_2(b), s)$, for some values
 159 $\varphi_1(b)$ and $\varphi_2(b)$ that are easily computable. In particular, L_b^* is contained in the slab σ^*
 160 and has the endpoints on different boundaries of σ^* . Finally, define the mapping point
 161 $\varphi(b) = (\varphi_1(b), \varphi_2(b))$. Thus, φ maps points to the right of the y -axis to points in the plane.
 162 Namely, $\varphi_1(b)$ is the slope of the line through b and $(0, 0)$ while $\varphi_2(b)$ is the slope of the line
 163 through b and $(0, s)$.

Let a be any point to the left of the y -axis and let $b \in B$. The segment ab intersects σ if and only if L_a^* intersects L_b^* . Namely, an intersection of L_a^* and L_b^* is dual to the line through a and b . The segments L_a^* and L_b^* intersect if and only if the order of their endpoints on boundaries of σ^* are reversed. Thus we have the following property:

$$ab \cap \sigma \neq \emptyset \iff (\varphi_1(a) - \varphi_1(b)) \cdot (\varphi_2(a) - \varphi_2(b)) < 0.$$

164 Given a point a , the set of points $b \in B$ with the property that ab intersects σ corresponds
 165 to the points b with $\varphi(b)$ in 2 quadrants with apex $\varphi(a)$.

We can use a 2-dimensional range tree to store the point set $\varphi(B)$, where each point $b \in B$ is identified with its image $\varphi(b)$. For any query $a \in A$, the points $b \in B$ such that ab intersects σ are obtained by querying the 2-dimensional range tree for the points of $\varphi(B)$ contained in the two quadrants

$$\{(x, y) \mid \varphi_1(a) > x \text{ and } \varphi_2(a) < y\} \cup \{(x, y) \mid \varphi_1(a) < x \text{ and } \varphi_2(a) > y\}.$$

166 The details are standard in computational geometry; see Chapter XXX of BKOS. ◀

167 ▶ **Lemma 6.4.** *We can compute $\Phi_\sigma(A, B)$ in $O(n \log^4 n)$ time.*

Proof. We construct the sets $\{B_1, \dots, B_t\}$ and the data structure $\mathcal{D}(B)$ described in Lemma 6.3. For each B_j , where $j = 1, \dots, t$, we build the data structure of Lemma 6.1. Since $\sum_{i=1}^t |B_i| = O(n \log n)$, we need

$$\sum_{i=1}^t O(|B_i| \log |B_i|) = O(n \log^2 n)$$

time for this.

Consider a point $a \in A$. We query $\mathcal{D}(B)$ and obtain the set $I(a)$ of indices such that

$$B(a) = \bigcup_{i \in I(a)} B_i.$$

Now, for each $i \in I(a)$, we query with a the data structure associated to B_i to obtain

$$b_i^*(a) \in \arg \min \{w_b \mid b \in B_i, |ab| \leq 1\}.$$

We then have that

$$\min \{w_b \mid b \in B_{\leq 1}(a)\} = \min \{w_{b_i^*(a)} \mid i \in I(a)\},$$

so we can obtain $\Phi_\sigma(a, B)$ from the points $b_i^*(a)$, $i \in I(a)$.

For a point a , we need $O(\log^2 n)$ time to obtain $I(a)$, and then we need $O(\log^2 n)$ time for each index i in $I(a)$. This means that we spend time $O(\log^4 n)$ for each point $a \in A$. ◀

7 Crossing a fixed segment

We will drop in the notation the dependency on P and just write G instead of $G(P)$. We will regard G as an unweighted graph and denote by $d_G(r, p)$ the shortest-path distance in G between r and p .

For each point $r \in P$, let T_r be a shortest-path tree from r in G . If there are several possible candidates, we fix T_r . We use $T_r[p]$ for the path in T_r from r to p . Let E_r be the set of edges of G not appearing in T_r . Thus $E_r = E(G) \setminus E(T_r)$.

For each r and p from P , let $\gamma(r, p)$ be the polygonal path defined by $T_r[p]$. For each $r \in P$ and each edge $pq \in E_r$, let $\gamma(r, pq)$ be the polygonal closed path obtained by concatenating $\gamma(r, p)$, pq , and the reverse of $\gamma(r, q)$. We define $\text{len}(r, pq)$ as the number of edges in $\gamma(r, pq)$. Thus we have $\text{len}(r, pq) = d_G(r, p) + d_G(r, q) + 1$.

Let σ be a segment. For simplicity we assume that σ does not contain any point of P . For each polygonal path γ , let $X(\gamma, \sigma)$ be the modulo two value of the number of crossings between γ and σ . For each $r, p \in P$ we define $N(r, p) = X(\gamma(r, p), \sigma)$, and for each $r \in P$ and $pq \in E_r$ we define $N(r, pq) = X(\gamma(r, pq), \sigma)$.

$$\begin{aligned} \Psi_\sigma(P) &:= \min \text{len}(r, pq) \\ \text{s.t. } &r \in P \\ &pq \in E_r \\ &N(r, pq) = 1. \end{aligned}$$

The results of Cabello and Giannopoulos [?] imply the following result. (Should we provide a self-contained proof adapted to this scenario?)

► **Theorem 7.1.** *Let (r^*, p^*q^*) be an optimal solution to $\Psi_\sigma(P)$. Then the points in $T_{r^*}[p] \cup T_{r^*}[q]$ are an optimal solution to the separation problem.*

To compute $\Psi_\sigma(P)$ we will iterate over the points r , so we define

$$\begin{aligned} \Psi_\sigma(r, P) &:= \min d_G(r, p) + d_G(r, q) \\ \text{s.t. } &pq \in E_r \\ &N(r, pq) = 1. \end{aligned}$$

We readily have

$$\Psi_\sigma(P) = 1 + \min_{r \in P} \Psi_\sigma(r, P)$$

Let us fix some $r \in P$ and discuss how to solve $\Psi_\sigma(r, P)$. We compute a shortest path tree T_r from r . Together with each point $p \in P$ we store $d_G(r, p)$.

We compute the values $N(r, p)$, $p \in P$, with a simple bottom-up traversal of T_r . We start setting $N_r[r] := 0$. For each point $p \in P$ whose parent in T_r is p' , we obtain $N_r[p]$ using that $N_r[p] = N_r[p'] + |\sigma \cap pp'| \pmod{2}$. This is, if pp' crosses σ , then $N_r[p] = N_r[p'] + 1 \pmod{2}$, otherwise $N_r[p] = N_r[p']$.

Assume that σ is a vertical segment with endpoints $(0, 0)$ and $(0, s)$.

We split the search for the optimal edge pq into cases:

- pq has both endpoints on the same side of the y -axis and connects points with $N(r, p) \neq N(r, q)$, or
- pq has endpoints in both sides of the y -axis, $N(r, p) \neq N(r, q)$, and pq does not intersect σ , or
- pq has endpoints in both sides of the y -axis, $N(r, p) \neq N(r, q)$, and pq intersects σ .

► **Lemma 7.2.** *We can compute $\Psi_\sigma(r, P)$ in $O(n \log^4 n)$ time.*

Proof. For $i \in \{0, 1\}$, let L_i be the subset of points of P to the left of the y -axis with $N(r, p) = i \pmod{2}$. For $i \in \{0, 1\}$, let R_i be the subset of points of P to the right of the y -axis and $N(r, p) = i \pmod{2}$. Clearly

$$P = L_0 \cup L_1 \cup R_0 \cup R_1.$$

Let σ_+ and σ_- be the rays contained in the y -axis after deleting σ . Using $d_G(r, p)$ as the weight w_p of point $p \in P$, we then have

$$\begin{aligned} \Psi_\sigma(r, P) = \min\{ & \Phi(L_0, L_1), \Phi(R_0, R_1), \Phi_{\sigma_+}(L_0, R_1), \Phi_{\sigma_+}(L_1, R_0), \\ & \Phi_{\sigma_-}(L_0, R_1), \Phi_{\sigma_-}(L_1, R_0), \Phi_\sigma(L_0, R_0), \Phi_\sigma(L_1, R_1)\}. \end{aligned}$$

Each of these instances can be solved using Lemmas 6.2 and 6.4 in $O(n \log^4 n)$ time. ◀

► **Theorem 7.3.** *We can solve the minimum separation problem using $O(n^2 \log^4 n)$ time.*

Proof. For each $r \in P$ we compute the shortest path tree T_r from r in T , the distances $d_G(r, p)$, $p \in P$, and the parities $N(r, p)$. We can then compute $\Psi_\sigma(r, P)$ using Lemma 7.2.

Repeat for each $r \in P$. Time bound.

Correctness. ◀

References

- 1 S. Cabello and P. Giannopoulos. The complexity of separating points in the plane. *Algorithmica*, 74(2):643–663, 2016.
- 2 S. Cabello and M. Jeřič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015.
- 3 S. Cabello and M. Kerber. Semi-dynamic connectivity in the plane. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2015.
- 4 A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.