

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 disks of the same size, 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 length $|pq|$ is at most the diameter of a disk. 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 *minimum-separation problem*. The input is a set \mathcal{D} of n unit 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 curve 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

$$\begin{aligned} \min \quad & |\mathcal{D}'| \\ \text{s.t.} \quad & \mathcal{D}' \subset \mathcal{D} \text{ and } \mathcal{D}' \text{ separates } s \text{ and } t. \end{aligned}$$

Unit disks are the most standard model used for wireless sensor networks; see for example [6, 8, 16]. Often the model is referred as UDG. This model provides an appropriate trade off between simplicity and accuracy. Other models are more accurate, as for example discussed in [10, 13], but obtaining efficient algorithm for them is much more difficult.

While unit disks give a simple model, exploiting the geometric features of the model is often challenging. Shortest paths in unit disk graphs are essential for routing and are a basic subroutine for several other more complex tasks. A somehow unexpected application of shortest paths in unit-disk graphs to boundary recognition is given in [15]. The minimum-separation problem and variants thereof have been considered in [2, 7, 14]. The problem is dual to the barrier-resilience problem considered in [1, 9, 11, 12]. It is not obvious that the minimum-separation problem can be solved optimally in polynomial time, and the known algorithm for this uses as a subroutine shortest paths in unit disk graphs. Thus, both problems considered in this paper are related and it is worth to consider them together.

* 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

Our contribution. We are aware of two algorithms to compute shortest-path trees in unit disk graphs in $O(n \log n)$ worst-case time: one by Cabello and Jeřič [3] and one Efrat, Itai and Katz [5]. Here we report on an implementation of a modification of the algorithm in [3], and compare it against two obvious alternatives. The only complex ingredient in the algorithm is computing the Delaunay triangulation, but efficient libraries are available for this. The algorithm of [5] would be substantially harder to implement and it has worse constants hidden in the O -notation.

As mentioned before, it is not obvious that the minimum-separation problem can be solved in polynomial time. A 2-approximation algorithm is given by Gibson, Kanade, and Varadarajan [7]. Cabello and Giannopoulos [2] provide an exact algorithm that takes $O(n^3)$ worst-case time and works for arbitrary shapes, not just disks. In this paper we improve this last algorithm to near-quadratic time for the special case of unit disks. The basic principle of the algorithm is the same, but several additional tools from Computational Geometry have to be employed to reduce the worst-case running time. We implement a variant of the new, near-quadratic-time algorithm and report on the experiments.

Assumptions. We will assume that *unit disk* means that it has radius $1/2$. Up to scaling the input data, this choice is irrelevant. However, it is convenient for the exposition because then the disks intersect whenever the distance between their centers is 1. The implementation and the experiments also make this assumption.

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

We will work with the graph $G_{\leq 1}(P)$ with vertex set P and an edge between two points $p, q \in P$ whenever their Euclidean distance $|pq|$ is at most 1. In the notation we remove the dependency on P and just use G instead of $G_{\leq 1}(P)$. For simplicity of the theoretical exposition we will sometimes assume that G is connected. It is trivial to adapt to the general case, for example treating each connected component separately. The implementation does not make this assumption.

In the minimum-separation problem we will use s and t for the points to separate, we will assume that the segment st is vertical, t is below s , and t is at the origin. The implementation also makes this assumption. A simple rigid transformation can be applied to the input to get to this setting.

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 Jeřič [3] to compute a shortest path tree in G from a given root point $r \in P$. As it is usually done for shortest path algorithms we use tables $dist[\cdot]$ and $\pi[\cdot]$ indexed by the points of P to record, for each point $p \in P$, the distance $d_G(s, p)$ and the ancestor of p in a shortest (s, p) -path.

The pseudocode of the algorithm, which we call UNWEIGHTEDSHORTESTPATH, is in Figure 1. We explain the intuition, taken almost verbatim from [3]. 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 from the source r .

79 We start with $W_0 = \{r\}$. At round i , we use $DT(P)$ to grow a neighbourhood around the
 80 points of W_{i-1} that contains W_i . More precisely, we consider the points adjacent to W_{i-1}
 81 in $DT(P)$ as candidate points for W_i . For each candidate point that is found to lie in W_i ,
 82 we also take its adjacent vertices in $DT(P)$ as new candidates to be included in W_i . For
 83 checking whether a candidate point p lies in W_i we use a data structure to find a nearest
 84 neighbour of p in W_{i-1} . If the distance from p to its nearest neighbour w in W_{i-1} is smaller
 85 than 1, then the shortest path tree is extended by connecting p to w .

```

UNWEIGHTEDSHORTESTPATH( $P, r$ )
1  build the Delaunay triangulation  $DT(P)$ 
2  for  $p \in P$ 
3       $dist[p] = \infty$ 
4       $\pi[p] = \text{NIL}$ 
5   $dist[r] = 0$ 
6   $W_0 = \{r\}$ 
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              $w =$  nearest neighbour of  $p$  in  $W_{i-1}$ 
17             if  $dist[p] = \infty$  and  $|pw| \leq 1$ 
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 from [3] to compute a shortest path tree in the unweighted case.

86 Cabello and Jeřić [3] show that the algorithm correctly computes the shortest-path tree
 87 from r . If for nearest neighbors we use a data structure that, for n points, has construction
 88 time $T_c(n)$ and query time $T_q(n)$, and the Delaunay triangulation is computed in $T_{DT}(n)$ time,
 89 then the algorithm takes $O(T_{DT}(n) + T_c(n) + nT_q(n))$ time. Standard tools in Computational
 90 Geometry imply that $T_{DT}(n) = O(n \log n)$, $T_c(n) = O(nq \log n)$ and $T_q(n) = O(\log n)$. This
 91 leads to the following.

92 ► **Theorem 2.1** (Cabello and Jeřić [3]). *Let P be a set of n points in the plane and let r
 93 be a point from P . In time $O(n \log n)$ we can compute a shortest path tree from s in the
 94 unweighted graph $G_{\leq 1}(P)$.*

95 It is clear that, when computing the shortest path tree from several sources, we only need
 96 to compute the Delaunay triangulation once.

2.2 Minimum separation with unit-disk

Cabello and Giannopoulos [2] present an algorithm for the minimum separation problem that in the worst-case runs in cubic-time. The algorithm has one feature that is both an advantage and a disadvantage: it works for any reasonable shapes, like segments or ellipses, and not just unit disks. This means that it is very generic, which is good, but it cannot exploit any properties of unit disks.

In this section we are going to describe an algorithm to solve the minimum separation problem *for unit disks* in roughly quadratic time. The improvement is based on 3 ingredients. The first ingredient is a reinterpretation of the algorithm of [2] for disks. In the original algorithm, we had to select a point inside each shape. For disks there is a natural, obvious choice, the center of the disk. This allows for a simpler description and interpretation of the algorithm. We provide the description in Section 2.2.1

The second ingredient is the efficient algorithm for shortest-path trees for the graph G . The third ingredient is a compact treatment of the edges of G using a few tools from Computational Geometry, namely range trees, point-line duality, and nearest-neighbour searches. This is explained in Section 2.2.2.

2.2.1 Generic algorithm specialized for unit disks

Let us first introduce some notation. Each walk W in the graph $G = G_{\leq 1}(P)$ defines a planar curve in the obvious way: we connect the points of P with segments in the order given by W . We will relax the notation slightly and denote also by W the curve itself. For any spanning tree T of G and any edge $e \in E(G) \setminus E(T)$, let $\text{cycle}(T, e)$ be the unique cycle in $T + e$. Finally, 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 [2] and explicit in [4]:

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

A consequence of this is that finding a minimum separation amounts to finding a shortest cycle G that crosses the segment st an odd number of times. Moreover, one can show that we can restrict our search to a very concrete family cycles, as follows. For each root r let us fix a shortest-path tree T_r from r in G . When there are many, the choice of T_r is irrelevant. Then, we can restrict our search to

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

APPENDIX

This follows from the co-called 3-path condition; see [2] for the ideas, and appendix ?? for a self-contained proof.

The values $\text{cr}_2(st, \text{cycle}(T_r, e))$ can be computed in constant amortized time per edge with some easy bookkeeping. Consider a fixed tree T_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 $N[p] = N[\pi[p]] + \text{cr}_2(st, p\pi[p])$. In the algorithm we have written it this way (lines 4–6), but one can of course compute the values at the time of computing the shortest path tree T_r .

We then have for each T_r

$$\begin{aligned} \forall pq \in E(G) \setminus E(T_r) : \quad & \text{cr}_2(st, \text{cycle}(T_r, pq)) = N[p] + N[q] + \text{cr}_2(st, pq) \pmod{2} \\ \forall pq \in E(T_r) : \quad & 0 = N[p] + N[q] + \text{cr}_2(st, pq) \pmod{2} \end{aligned}$$

131 because crossings that are counted twice cancel out modulo 2. (In particular, the path in T_r
 132 from r to the lowest common ancestor of p and q is counted twice.) This implies that we can
 133 just check for *all* edges pq of G whether the sum $N[p] + N[q] + \text{cr}_2(st, pq)$ is 0 modulo 2. The
 134 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  $r \in P$ 
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           $N[p] = N[\pi[p]] + \text{cr}_2(st, p\pi[p]) \pmod{2}$ 
7      for  $pq \in E(G(P))$ 
8          if  $N[p] + N[q] + \text{cr}_2(pq, st) \pmod{2} = 1$ 
9               $best = \min\{best, dist[p] + dist[q] + 1$ 
10 return  $best$ 

```

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

135 Let us look into the time complexity of the algorithm. For each point $r \in P$ we have to
 136 compute a shortest-path tree in G . This can be done in $O(n \log n)$ in our case, as discussed
 137 in Section 2.1. Then, for each edge pq of G some constant amount of work is done. Thus for
 138 each point r we spend $O(n \log n + |E(G)|)$. This is cubic in the worst-case. We could get an
 139 improved running time if we can treat all the edges of G compactly. This is what we explain
 140 next.

141 2.2.2 Compact treatment of edges

142 Without loss of generality, in our theoretical discussion we will assume that $z = (0, 0)$ and
 143 $z' = (0, s)$. Therefore, zz' is a segment, henceforth denoted by σ , contained in the y -axis.

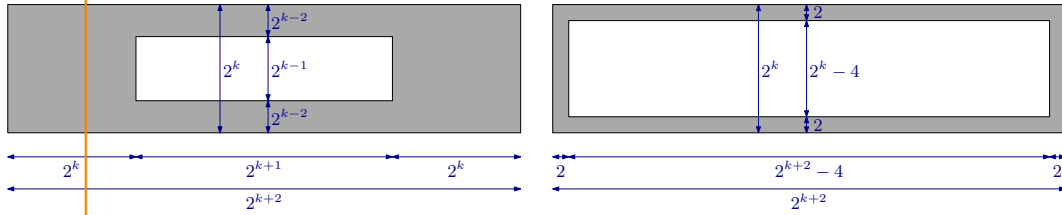
144 3 Implementation

145 3.1 Shortest-path tree in unit-disk graphs

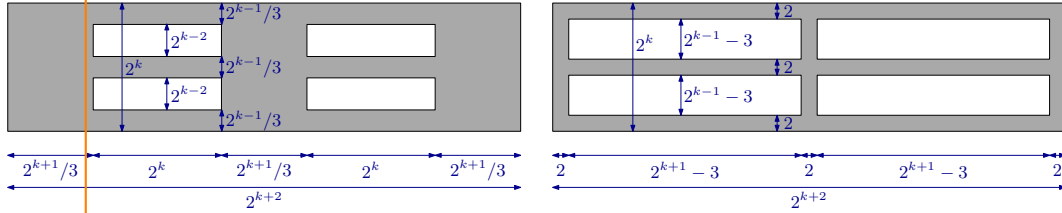
146 **Alternative constructions.** Let us mention two obvious alternatives that we use in our
 147 comparison.

148 The first alternative is to build the graph $G = G_{\leq 1}(P)$ explicitly. Thus, for each pair
 149 of points p, q we check whether their distance is at most once and add an edge to a graph
 150 data structure. We can then use breadth-first-search (BFS) from the given root r . The
 151 preprocessing is quadratic, and the time spent to compute a shortest-path tree depends on
 152 the density of the graph G .

153 The second option we consider is to use a unit-length grid. Two points (x, y) and (x', y')
 154 are in the same grid cell if and only if $(\lfloor x \rfloor, \lfloor y \rfloor) = (\lfloor x' \rfloor, \lfloor y' \rfloor)$. We store all the points of
 155 a grid cell c in a list $\ell(c)$. The non-empty lists $\ell(c)$ are stored in a dictionary, where the
 156 bottom-left corner of the cell is used as key. We can then run some sort of BFS using this



■ **Figure 3** Data generation with a small hole (left) and a large hole (right).



■ **Figure 4** Data generation with four small holes (left) and four large holes (right).

157 data. When processing a point p in a cell c , we have to treat all the points in c and its
 158 adjacent cells as candidate points. The preprocessing is linear, and the time spent to compute
 159 a shortest-path tree depends on the density of the graph G . The number of candidates that
 are checked is proportional to the number of edges of the graph.

Are perhaps
 points deleted
 from the list
 once they are
 assigned a
 level?

3.2 Minimum separation with unit-disk

4 Experimental results

Data was generated uniformly at random in polygons...

4.1 Shortest-path tree in unit-disk graphs

4.2 Minimum separation with unit-disk

5 Conclusions

References

- 1 S. Bereg and D. G. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In *Proc. 5th ALGOSENSORS*, volume 5804 of *LNCS*, pages 29–40. Springer, 2009.
- 2 S. Cabello and P. Giannopoulos. The complexity of separating points in the plane. *Algorithmica*, 74(2):643–663, 2016.
- 3 S. Cabello and M. Jeřičič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015.
- 4 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.
- 5 A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.
- 6 J. Gao and L. Guibas. Geometric algorithms for sensor networks. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 370(1958):27–51, 2011.

- 182 **7** M. Gibson, G. Kanade, and K. Varadarajan. On isolating points using disks. In *Proc. 19th*
183 *ESA*, volume 6942 of *LNCS*, pages 61–69. Springer, 2011.
- 184 **8** M. L. Huson and A. Sen. Broadcast scheduling algorithms for radio networks. In *IEEE*
185 *MILCOM '95*, volume 2, pages 647–651 vol.2, 1995.
- 186 **9** S. Kloder and S. Hutchinson. Barrier coverage for variable bounded-range line-of-sight
187 guards. In *Proc. ICRA*, pages 391–396. IEEE, 2007.
- 188 **10** F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-hoc networks beyond unit disk graphs.
189 In *Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing*, DIALM-
190 POMC '03, pages 69–78, 2003.
- 191 **11** S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proc. 11th*
192 *MobiCom*, pages 284–298. ACM, 2005.
- 193 **12** S. Kumar, T.-H. Lai, and A. Arora. Barrier coverage with wireless sensors. *Wireless*
194 *Networks*, 13(6):817–834, 2007.
- 195 **13** Z. Lotker and D. Peleg. Structure and algorithms in the sinr wireless model. *SIGACT*
196 *News*, 41(2):74–84, 2010.
- 197 **14** R. Penninger and I. Vigan. Point set isolation using unit disks is NP-complete. *CoRR*,
198 abs/1303.2779, 2013.
- 199 **15** Y. Wang, J. Gao, and J. S. Mitchell. Boundary recognition in sensor networks by topological
200 methods. In *Proceedings of the 12th Annual International Conference on Mobile Computing*
201 *and Networking*, MobiCom '06, pages 122–133, 2006.
- 202 **16** F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*.
203 Elsevier/Morgan-Kaufmann, 2004.

204 **A** Building block

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

► **Lemma 1.1.** *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*

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

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

209 For every node ν of \mathcal{T} :

210 ■ Let $P(\nu)$ denote the set of points stored at the subtree rooted at ν . We refer to $P(\nu)$ as
211 a *canonical subset*.

212 ■ Let $U(\nu)$ be the union of unit disks centered at the points of $P(\nu)$.

213 ■ We build a data structure to decide whether a given query point is contained in $U(\nu)$. A
214 way to do this is to construct a point-location data structure for the Voronoi diagram
215 of $P(\nu)$. Given a query point q , we find a closest neighbor from $P(\nu)$ by locating a cell
216 c of the Voronoi diagram that contains q . We then check whether q is at most at unit
217 distance from the site defining c . If the points of $P(\nu)$ are sorted (lexicographically), then
218 the Voronoi diagram of $P(\nu)$ can be build in linear time ?? The preprocessing time for
219 node ν is $O(|P(\nu)|)$ and a query can be answered in $O(\log |P(\nu)|) = O(\log n)$ time.

220 This finishes the description of the construction of the data structure. Let us analyze the
221 building time. We sort the points of P lexicographically at the root and for each node ν we
222 get the points $P(\nu)$ sorted from its parent. Since the canonical subsets at each level of the
223 tree are disjoint, at each level of the tree we spend $O(n)$ time. Since \mathcal{T} is balanced, there are
224 $O(\log n)$ 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:

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

225 .

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

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

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

228 ► **Lemma 1.2.** *Let A and B be sets of at most n weighted points in the plane. In $O(n \log^2 n)$
229 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

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

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

231 We need $O(n \log n)$ time to construct the data structure and $O(\log^2 n)$ time per query.

232 The result follows. \blacktriangleleft

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

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

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

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

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

234 **► Lemma 1.3.** *There is a family $\{B_1, \dots, B_t\}$ of subsets of B and a data structure $\mathcal{D}(B)$*
 235 *with the following properties*

- 236 $\blacksquare \sum_{i=1}^t |B_i| = O(n \log n);$
- 237 $\blacksquare \mathcal{D}(B)$ *has size $O(n \log n)$ and can be constructed in $O(n \log n)$ time;*
- 238 \blacksquare *for each point a with negative x -coordinate, there is a subset of indices $I(a) \subset \{1, \dots, t\}$*
 239 *such that $|I(a)| = O(\log^2 n)$ and $B(a)$ is the disjoint union of $\{B_i\}_{i \in I(a)}$;*
- 240 \blacksquare *for each query point a with $a_x < 0$, the data structure $\mathcal{D}(B)$ returns $I(a)$ in $O(\log^2 n)$*
 241 *time.*

242 **Proof.** It is convenient to use point-line duality. We use the precise duality described
 243 in BKOS Chapter XXX: the non-vertical line $\ell \equiv y = mx + c$ is mapped to the point
 244 $\ell^* = (m, -c)$.

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 :

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

245 In the dual space, L_b^* is a segment with endpoints $(\varphi_1(b), 0)$ and $(\varphi_2(b), s)$, for some values
 246 $\varphi_1(b)$ and $\varphi_2(b)$ that are easily computable. In particular, L_b^* is contained in the slab σ^*
 247 and has the endpoints on different boundaries of σ^* . Finally, define the mapping point
 248 $\varphi(b) = (\varphi_1(b), \varphi_2(b))$. Thus, φ maps points to the right of the y -axis to points in the plane.
 249 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
 250 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:

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

251 Given a point a , the set of points $b \in B$ with the property that ab intersects σ corresponds
 252 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

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

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

254 ▶ **Lemma 1.4.** *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 1.3. For each B_j , where $j = 1, \dots, t$, we build the data structure of Lemma 1.1. Since $\sum_{i=1}^t |B_i| = O(n \log n)$, we need

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

255 time for this.

Consider a point $a \in A$. We query $\mathcal{D}(B)$ and obtain the 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

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

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

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

B 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 $N(r, pq) = X(\gamma(r, pq), \sigma)$.

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

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

► **Theorem 2.1.** *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

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

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

- 288 ■ pq has endpoints in both sides of the y -axis, $N(r, p) \neq N(r, p)$, and pq does not intersect
 289 σ , or
 290 ■ pq has endpoints in both sides of the y -axis, $N(r, p) \neq N(r, p)$, and pq intersects σ .

291 ► **Lemma 2.2.** *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

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

292 Each of these instances can be solved using Lemmas 1.2 and 1.4 in $O(n \log^4 n)$ time. ◀

293 ► **Theorem 2.3.** *We can solve the minimum separation problem using $O(n^2 \log^4 n)$ time.*

294 **Proof.** For each $r \in P$ we compute the shortest path tree T_r from r in T , the distances
 295 $d_G(r, p)$, $p \in P$, and the parities $N(r, p)$. We can then compute $\Psi_\sigma(r, P)$ using Lemma 2.2.

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

297 Correctness.

298 ◀