Unweighted Geometric Hitting Set for Line-Constrained Disks and Related Problems

Gang Liu
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points and a set S of m disks in the plane, the disk hitting set problem asks for a smallest subset of P such that every disk of S contains at least one point in the subset. The problem is NP-hard. This paper considers a line-constrained version in which all disks have their centers on a line. We present an $O(m \log^2 n + (n+m)\log(n+m))$ time algorithm for the problem. This improves the previous result of $O(m^2 \log m + (n+m)\log(n+m))$ time for the weighted case of the problem where every point of P has a weight and the objective is to minimize the total weight of the hitting set. Our algorithm also solves a more general line-separable problem with a single intersection property: The points of P and the disk centers are separated by a line ℓ and the boundary of every two disks intersect at most once on the side of ℓ containing P.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases hitting set, line-constrained, line-separable, unit disks, half-planes, coverage

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.68

Related Version *Full Version*: http://arxiv.org/abs/2407.00331

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Let P be a set of n points and S a set of m disks in the plane. The *hitting set* problem is to compute a smallest subset of P such that every disk in S contains at least one point in the subset (i.e., every disk is *hit* by a point in the subset, and the subset is called a *hitting set*). The problem is NP-hard, even if all disks have the same radius [18,25]. Polynomial-time approximation algorithms are known for the problem, e.g., [3,15,16,21,24,25].

In this paper, we consider the *line-constrained* version of the problem, where centers of all disks are on a line while the points of P can be anywhere in the plane. The weighted case of the problem was studied by Liu and Wang [22], where each point of P has a weight and the objective is to minimize the total weight of the hitting set. Their algorithm runs in $O((m+n)\log(m+n) + \kappa \log m)$ time, where κ is the number of pairs of disks that intersect and $\kappa = O(m^2)$ in the worst case. They reduced the runtime to $O((m+n)\log(m+n))$ for the *unit-disk case*, where all disks have the same radius [22]. Our problem in this paper is for the unweighted case. To the best of our knowledge, we are not aware of any previous work that particularly studied the unweighted hitting set problem for line-constrained disks. We propose an algorithm of $O(m \log^2 n + (n+m)\log(n+m))$ time, which improves the weighted case algorithm of $O(m^2 \log m + (n+m)\log(n+m))$ worst-case time [22]. Perhaps theoretically more interesting is that the worst-case runtime of our algorithm is near linear.

1.1 Related work

A closely related problem is the disk coverage problem, which is to compute a smallest subset of S that together cover all the points of P. This problem is also NP-hard because it is dual to the hitting set problem in the unit-disk case (i.e., all disks have the same radius). Polynomial-time algorithms are known for certain special cases, e.g., [1,4,9,10]. In particular, the line-constrained problem (in which all disks are centered on a line) was studied by Pedersen and Wang [26]. Their algorithm runs in $O((m+n)\log(m+n)+\kappa\log m)$ time, where κ is the number of pairs of disks that intersect and $\kappa = O(m^2)$ in the worst case; they also solved the unit-disk case in $O((m+n)\log(m+n))$ time. As noted above, in the unit-disk case, the coverage and hitting set problems are dual to each other and therefore the two problems can essentially be solved by the same algorithm. However, this is not the case if the radii of the disks are different.[1]

In addition, the $O((m+n)\log(m+n)+\kappa\log m)$ time algorithm of Pedersen and Wang [26] also works for the weighted *line-separable unit-disk* version, where all disks have the same radius and the disk centers are separated from the points of P by a line.

All the above results are for the weighted case. The unweighted disk coverage case was also particularly studied before. Liu and Wang [23][2] considered the line-constrained problem and gave an $O(m\log n\log m+(n+m)\log(n+m))$ time algorithm. For the line-separable unit-disk case, Ambühl et al. [1] derived an algorithm of $O(m^2n)$ time, which was used as a subroutine in their algorithm for the general coverage problem in the plane (without any constraints). An improved $O(nm+n\log n)$ time algorithm is presented in [8]. Liu and Wang's approach [23] solves this case in $O((n+m)\log(n+m))$ time.

If disks of S are half-planes, the problem becomes the half-plane coverage problem. For the weighted case, Chan and Grant [4] proposed an algorithm for the *lower-only case* where all half-planes are lower ones; their algorithm runs in $O(n^4)$ time when $m=n$. With the observation that a half-plane may be considered as a unit disk of infinite radius, the lower-only half-plane coverage problem is essentially a special case of the line-separable unit-disk coverage problem [26]. Consequently, applying the algorithm of [26] can solve the weighted lower-only case in $O(n^2\log n)$ time (when $m=n$) and applying the algorithm of [23] can solve the unweighted lower-only case in $O(n\log n)$ time. Wang and Xue [28] derived another $O(n\log n)$ time algorithm for the unweighted lower-only case with a different approach and also proved an $\Omega(n\log n)$ lower bound under the algebraic decision tree model by a reduction from the set equality problem [2] (note that this leads to the same lower bound for the line-separable unit-disk coverage problem). For the general case where both upper and lower half-planes are present, Har-Peled and Lee [17] solved the weighted problem in $O(n^5)$ time. Pedersen and Wang [26] showed that the problem can be reduced to $O(n^2)$ instances of the lower-only case problem. Consequently, applying the algorithms of [26] and [23] can solve the weighted and unweighted cases in $O(n^4\log n)$ and $O(n^3\log n)$ time, respectively. Wang and Xue [28] gave a more efficient algorithm of $O(n^{4/3}\log^{5/3}n\log^{O(1)}\log n)$ time for the unweighted case.

[1] Note that [12] provides a method to reduce certain coverage problems to instances of the hitting set problem; however, the reduction algorithm, which takes more than $O(n^5)$ time, is not efficient.

[2] See the arXiv version of [23], which improves the result in the original conference paper. The algorithms follow the same idea, but the arXiv version provides more efficient implementations.

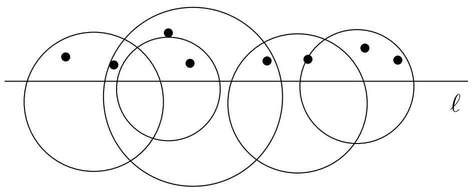

Figure 1 Illustrating the line-separable single-intersection case: Centers of all disks are below ℓ.

1.2 Our result

Instead of solving the line-constrained problem directly, we tackle a more general problem in which the points of P and the centers of the disks of S are separated by a line ℓ such that the boundaries of every two disks intersect at most once on the side of ℓ containing P (see Fig. 1). We refer to it as the *line-separable single-intersection* hitting set problem (we will explain it later why this problem is more general than the line-constrained problem). We present an algorithm of $O(m \log^2 n + (n + m) \log(n + m))$ time for the problem. To this end, we find that some points in P are "useless" and thus can be pruned from P. More importantly, the remaining points have certain properties so that the problem can be reduced to the 1D hitting set problem, which can then be easily solved. The algorithm itself is relatively simple and quite elegant. However, one challenge is to show its correctness, and specifically, to prove why the "useless" points are indeed useless. The proof is lengthy and fairly technical, which is one of our main contributions.

The line-constrained problem. To solve the line-constrained problem, where all disks of S are centered on a line ℓ, the problem can be reduced to the line-separable single-intersection case. Indeed, without loss of generality, we assume that ℓ is the x-axis. For each point p of P below ℓ, we replace p by its symmetric point with respect to ℓ. As such, we obtain a set of points that are all above ℓ. Since all disks are centered on ℓ, it is not difficult to see that an optimal solution using this new set of points corresponds to an optimal solution using P. Furthermore, since disks are centered on ℓ, although their radii may not be equal, the boundaries of any two disks intersect at most once above ℓ. Therefore, the problem becomes an instance of the line-separable single-intersection case. As such, applying the algorithm for line-separable single-intersection problem solves the line-constrained problem in $O(m \log^2 n + (n + m) \log(n + m))$ time. Therefore, in the rest of the paper, we will focus on solving the line-separable single-intersection problem.

The unit-disk case. As mentioned earlier, the unit-disk case problem where all disks have the same radius can be reduced to the coverage problem (and vice versa). More specifically, if we consider the set of unit disks centered at the points of P as a set of "dual disks" and consider the centers of the disks of S a set of "dual points", then the hitting set problem is equivalent to finding a smallest subset of dual disks whose union covers all dual points. Consequently, applying the line-separable unit-disk coverage algorithm in [23] solves the hitting set problem in $O((n+m) \log(n+m))$ time. Nevertheless, we show that our technique can directly solve the hitting set problem in this case in the same time complexity.

The half-plane hitting set problem. As in the coverage problem discussed above, if disks of S are half-planes, the problem becomes the half-plane hitting set problem. For the weighted case, the approach of Chan and Grant [4] solves the lower-only case in $O(n^4)$ time when

$m = n$. Again, with the observation that a half-plane may be viewed as a unit disk of infinite radius, the lower-only half-plane hitting set problem is a special case of the line-separable unit-disk hitting set problem. As such, applying the algorithm of [22] can solve the weighted lower-only case in $O(n^2 \log n)$ time (when $m = n$) and applying the unit-disk case algorithm discussed above can solve the unweighted lower-only case in $O(n \log n)$ time. For the general case where both upper and lower half-planes are present, Har-Peled and Lee [17] solved the weighted problem in $O(n^6)$ time. Liu and Wang [22] showed that the problem (for both the weighted and unweighted cases) can be reduced to $O(n^2)$ instances of the lower-only case problem. Consequently, applying the above algorithms for the weighted and unweighted lower-only case problems can solve the weighted and unweighted general case problems in $O(n^4 \log n)$ and $O(n^3 \log n)$ time, respectively.

Lower bound. As discussed above, the $\Omega(n \log n)$ lower bound in [28] for the lower-only half-plane coverage problem leads to the $\Omega(n \log n)$ lower bound for the line-separable unit-disk coverage problem when $m = n$. As the unit-disk hitting set problem is dual to the unit-disk coverage problem, it also has $\Omega(n \log n)$ as a lower bound. Since the unit-disk hitting set problem is a special case of the line-separable single-intersection hitting set problem, $\Omega(n \log n)$ is also a lower bound of the latter problem. Similarly, since the lower-only half-plane hitting set is dual to the lower-only half-plane coverage, $\Omega(n \log n)$ is also a lower bound of the former problem.

An algorithm in the algebraic decision tree model. In the algebraic decision tree model, where the time complexity is measured only by the number of comparisons, our method, combining with a technique recently developed by Chan and Zheng [6], shows that the line-separable single-intersection problem (and thus the line-constrained problem) can be solved using $O((n+m)\log(n+m))$ comparisons, matching the above lower bound. To ensure clarity in the following discussion, unless otherwise stated, all time complexities are based on the standard real RAM model.

Outline. The rest of the paper is organized as follows. After introducing the notation in Section 2, we describe our algorithm in Section 3. The algorithm correctness is proved in Section 4. We show how to implement the algorithm efficiently in Section 5. The algebraic decision tree algorithm and the unit-disk case algorithm are also discussed in Section 5.

2 Preliminaries

This section introduces some notation and concepts that will be used throughout the paper.

As discussed above, we focus on the line-separable single-intersection case. Let P be a set of n points and S a set of m disks in the plane such that the points of P and the centers of the disks of S are separated by a line ℓ and the boundaries of every two disks intersect at most once on the side of ℓ which contains P. Note that the points of P and the disk centers may be on ℓ. Without loss of generality, we assume that ℓ is the x-axis and the points of P are above (or on) ℓ while the disk centers are below (or on) ℓ (see Fig. 1). As such, the boundaries of every two disks intersect at most once above ℓ. Our goal is to compute a smallest subset of P such that each disk of S is hit by at least one point in the subset.

Under this setting, for each disk $s \in S$, only its portion above ℓ matters for our problem. Hence, unless otherwise stated, a disk s refers only to its portion above (and on) ℓ. As such, the boundary of s consists of an *upper arc*, i.e., the boundary arc of the original disk above ℓ, and a *lower segment*, i.e., the intersection of s with ℓ. Note that s has a single leftmost (resp., rightmost) point, which is the left (resp., right) endpoint of the lower segment of s.

If P' is a subset of P that form a hitting set for S, we call P' a *feasible solution*. If P' is a feasible solution of minimum size, then P' is an *optimal solution*.

We assume that each disk of S is hit by at least one point of P since otherwise there would be no feasible solution. Our algorithm is able to check whether the assumption is met.

We make a general position assumption that no two points of A have the same x-coordinate, where A is the union of P and the set of the leftmost and rightmost points of the upper arcs of all disks. Degenerate cases can be handled by standard perturbation techniques, e.g., [14].

For any point p in the plane, we denote its x-coordinate by $x(p)$. We sort all points in P in ascending order of their x-coordinates, resulting in a sorted list $\{p_1, p_2, \cdots, p_n\}$. We use $P[i,j]$ to denote the subset $\{p_i, p_{i+1}, \cdots, p_j\}$, for any $1 \leq i \leq j \leq n$. We sort all disks in ascending order of the x-coordinates of the leftmost points of their upper arcs; let $\{s_1, s_2, \cdots, s_m\}$ be the sorted list. We use $S[i,j]$ to denote the subset $\{s_i, s_{i+1}, \cdots, s_j\}$, for $1 \leq i \leq j \leq m$. For convenience, let $P[i,j] = \emptyset$ and $S[i,j] = \emptyset$ if $i > j$. For each disk s_i, let l_i and r_i denote the leftmost and rightmost points of its upper arc, respectively.

For any disk $s \in S$, we use $S_l(s)$ (resp., $S_r(s)$) to denote the subset of disks S whose leftmost points are to the left (resp., right) of that of s, that is, if the index of s is i, then $S_l(s) = S[1, i-1]$ and $S_r(s) = S[i+1, m]$. For any disk $s' \in S_l(s)$, we also say that s' is *to the left* of s; similarly, if $s' \in S_r(s)$, then s' is *to the right* of s. For convenience, if s' is to the left of s, we use $s' \prec s$ to denote it.

For a point $p_i \in P$ and a disk $s_k \in S$, we say that p_i is *vertically above* s_k (or s_k is *vertically below* p_i) if p_i is outside s_k and $x(l_k) < x(p_i) < x(r_k)$.

The non-containment property. If a disk s_i contains another disk s_j completely, then s_i is redundant for our problem since any point hitting s_j also hits s_i. It is easy to find those redundant disks in $O(m \log m)$ time (indeed, this is a 1D problem since s_i contains s_j if and only if the lower segment of s_i contains that of s_j). Therefore, to solve our problem, we first remove such redundant disks from S and then work on the remaining disks. For simplicity, from now on we assume that no disk of S contains another. Therefore, S has the following *non-containment* property, which is critical to our algorithm.

▶ **Observation 1.** (Non-Containment Property) *For any two disks $s_i, s_j \in S$, $x(l_i) < x(l_j)$ if and only if $x(r_i) < x(r_j)$.*

3 The algorithm description

In this section, we describe our algorithm. We follow the notation defined in Section 2.

We begin with the following definition, which is critical for our algorithm.

▶ **Definition 2.** *For each disk $s_i \in S$, among all the points of P covered by s_i, define $a(i)$ as the smallest index of these points and $b(i)$ the largest index of them.*

Since each disk s_i contains at least one point of P, both $a(i)$ and $b(i)$ are well defined.

▶ **Definition 3.** *For any point $p_k \in P$, we say that p_k is* prunable *if there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) < k < b(i)$.*

We now describe our algorithm. Although the description seems simple, establishing its correctness is by no means an easy task. We devote Section 4 to the correctness proof. The implementation of the algorithm, which is also not straightforward, is presented in Section 5.

Algorithm description. The algorithm has three main steps.

1. Compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We will show in Section 5 that this can be done in $O(m \log^2 n + (n+m)\log(n+m))$ time.
2. Find all prunable points; let Q be the set of all prunable points. We will show in Section 5 that Q can be computed in $O((n+m)\log(n+m))$ time.
   Let $P^* = P \setminus Q$. We will prove in Section 4 that P^* contains an optimal solution to the hitting problem on P and S. This means that it suffices to work on P^* and S.
3. Reduce the hitting set problem on P^* and S to a 1D hitting set problem, as follows. For each point of P^*, we project it perpendicularly onto ℓ. Let $\tilde{P}$ be the set of all projected points. For each disk $s_i \in S$, we create a segment on ℓ whose left endpoint has x-coordinate equal to $x(p_{a(i)})$ and whose right endpoint has x-coordinate equal to $x(p_{b(i)})$. Let $\tilde{S}$ be the set of all segments thus created.
   We solve the following 1D hitting set problem: Find a smallest subset of points of $\tilde{P}$ such that every segment of $\tilde{S}$ is hit by a point of the subset. This 1D problem can be easily solved in $O((|\tilde{S}| + |\tilde{P}|)\log(|\tilde{S}| + |\tilde{P}|))$ time [22],[3] which is $O((m+n)\log(m+n))$ since $|\tilde{P}| \le n$ and $|\tilde{S}| = m$.
   Suppose that $\tilde{P}_{\text{opt}}$ is any optimal solution for the 1D problem. We create a subset P^*_{opt} of P^* as follows. For each point of $\tilde{P}_{\text{opt}}$, suppose that it is the projection of a point $p_i \in P^*$; then we add p_i to P^*_{opt}. We will prove in Section 4 that P^*_{opt} is an optimal solution to the hitting set problem on P^* and S.

   We summarize the result in the following theorem.

▶ **Theorem 4.** *Given a set P of n points and a set S of m disks in the plane such that the disk centers are separated from the points of P by a line and the single-intersection condition is satisfied, the hitting set problem is solvable in $O(m \log^2 n + (n+m)\log(n+m))$ time.*

4 Algorithm correctness

In this section, we prove the correctness of our algorithm. More specifically, we will argue the correctness of the second and the third main steps of the algorithm. We start with the third main step, as it is relatively straightforward. In fact, arguing the correctness of the second main step is quite challenging and is a main contribution of our paper.

Correctness of the third main step. For each disk $s_i \in S$, let s'_i refer to the segment of $\tilde{S}$ created from s_i. For each point $p_j \in P$, let p'_j refer to the point of $\tilde{P}$ which is the projection of p_j. Lemma 5 justifies the correctness of the third main step of the algorithm.

▶ **Lemma 5.** *A point $p_j \in P^*$ hits a disk $s_i \in S$ if and only if p'_j hits s'_i.*

Proof. Suppose p_j hits s_i. Then, $p_j \in s_i$. By definition, we have $a(i) \le j \le b(i)$. Hence, $x(p_{a(i)}) \le x(p_j) \le x(p_{b(i)})$, and thus p'_j hits s'_i by the definitions of p'_j and s'_i.

On the other hand, suppose that p'_j hits s'_i. Then, according to the definitions of p'_j and s'_i, $x(p_{a(i)}) \le x(p_j) \le x(p_{b(i)})$ holds. If $j = a(i)$ or $j = b(i)$, then p_j must hit s_i following the definitions of $a(i)$ and $b(i)$. Otherwise, we have $a(i) < j < b(i)$. Observe that p_j must be inside s_i since otherwise p_j would be a prunable point and therefore could not be in P^*. As such, p_j must hit s_i. ◀

[3] The algorithm in [22], which uses dynamic programming, is for the weighted case where each point has a weight. Our problem is simpler because it is the unweighted case. We can use a simple greedy algorithm to solve it.

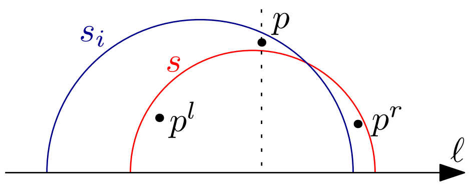

Figure 2 Illustrating the proof of Observation 6.

4.1 Correctness of the second main step

In what follows, we focus on the correctness of the second main step.

For any disk s, let $P(s)$ denote the subset of points of P inside s. For any point p, let $S(p)$ denote the subset of disks of S hit by p. For any subset $P' \subseteq P$, by slightly abusing notation, let $S(P')$ denote the subset of disks of S hit by at least one point of P', i.e., $S(P') = \bigcup_{p\in P'} S(p)$.

The following observation follows directly from the definition of prunable points.

▶ **Observation 6.** *Suppose a point p is a prunable point in P. Then, there is a disk $s \in S$ vertically below p such that the following are true.*

1. *$P(s)$ has both a point left of p and a point right of p.*
2. *For any two points $p^l, p^r \in P(s)$ with one left of p and the other right of p, we have $S(p) \subseteq S(p^l) \cup S(p^r)$.*

Proof. The first statement directly follows the definition of prunable points. For the second statement, without loss of generality, assume that p^l is left of p while p^r is right of p (see Fig. 2). Consider any disk $s_i \in S(p)$. By definition, $p \in s_i$. As $p \notin s$, $s_i \neq s$. Hence, s_i is either in $S_l(s)$ or in $S_r(s)$. If $s_i \in S_l(s)$, then due to the non-containment property, s_i must contain the area of s to the left of p and therefore must contain p^l, which implies $s_i \in S(p^l)$. Similarly, if $s_i \in S_r(s)$, then s_i must be in $S(p^r)$. ◀

The following lemma establishes the correctness of the second main step of the algorithm.

▶ **Lemma 7.** *P^* contains an optimal solution for the hitting set problem on S and P.*

Proof. Let P_{opt} be an optimal solution for S and P. Let Q be the set of all prunable points. Recall that $P^* = P \setminus Q$. If $P_{\text{opt}} \cap Q = \emptyset$, then $P_{\text{opt}} \subseteq P^*$ and thus the lemma is vacuously true. In what follows, we assume that $|P_{\text{opt}} \cap Q| \geq 1$.

Pick an arbitrary point from $P_{\text{opt}} \cap Q$, denoted by $\hat{p}_1$. Below, we give a process that can find a point p^* from P^* to replace $\hat{p}_1$ in P_{opt} such that the new set $P^1_{\text{opt}} = \{p^*\} \cup P_{\text{opt}} \setminus \{\hat{p}_1\}$ is a feasible solution, implying that P^1_{opt} is still an optimal solution since $|P^1_{\text{opt}}| = |P_{\text{opt}}|$. As $p^* \in P^*$, we have $|P^1_{\text{opt}} \cap Q| = |P_{\text{opt}} \cap Q| - 1$. Therefore, if $P^1_{\text{opt}} \cap Q$ is still nonempty, then we can repeat the process for other points in $P^1_{\text{opt}} \cap Q$ until we obtain an optimal solution P^*_{opt} with $P^*_{\text{opt}} \cap Q = \emptyset$, which will prove the lemma. The process involves induction. To help the reader understand it better, we first provide the details for the first two iterations of the process (we will introduce some notation that appears unnecessary for the first two iterations, but these will be needed for explaining the inductive hypothesis later).

The first iteration. Let $P'_{\text{opt}} = P_{\text{opt}} \setminus \{\hat{p}_1\}$. Since $\hat{p}_1 \in Q$, by Observation 6, S has a disk $\hat{s}_1$ vertically below $\hat{p}_1$ such that $P(\hat{s}_1)$ contains both a point left of $\hat{p}_1$, denoted by $\hat{p}^l_1$, and a point right of $\hat{p}_1$, denoted by $\hat{p}^r_1$. Furthermore, $S(\hat{p}_1) \subseteq S(\hat{p}^l_1) \cup S(\hat{p}^r_1)$. Since $\hat{p}_1 \notin \hat{s}_1$ and $P_{\text{opt}} = P'_{\text{opt}} \cup \{\hat{p}_1\}$ forms a hitting set of P, P'_{opt} must have a point p that hits $\hat{s}_1$. Clearly, p

is left or right of $\hat{p}_1$. Without loss of generality, we assume that p is right of $\hat{p}_1$. Since $\hat{p}_1^r$ refers to an arbitrary point to the right of $\hat{p}_1$ that hits $\hat{s}_1$ and p is also a point to right of $\hat{p}_1$ that hits $\hat{s}_1$, for notational convenience, we let $\hat{p}_1^r$ refer to p. As such, $\hat{p}_1^r$ is in P'_{opt}.

Consider the point $\hat{p}_1^l$. Since $S(\hat{p}_1) \subseteq S(\hat{p}_1^l) \cup S(\hat{p}_1^r)$ and $\hat{p}_1^r$ is in P'_{opt}, it is not difficult to see that $S(P_{\text{opt}}) \subseteq S(P'_{\text{opt}}) \cup S(\hat{p}_1^l)$ and thus $P'_{\text{opt}} \cup \{\hat{p}_1^l\}$ is a feasible solution. As such, if $\hat{p}_1^l \notin Q$, then we can use $\hat{p}_1^l$ as our target point p^* and our process (to find p^*) is performed. In what follows, we assume $\hat{p}_1^l \in Q$.

We let $\hat{p}_2 = \hat{p}_1^l$. Define $A_1 = \{\hat{p}_1^r\}$. According to the above discussion, $A_1 \subseteq P'_{\text{opt}}$, $S(\hat{p}_1) \subseteq S(A_1) \cup S(\hat{p}_2)$, $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution, $\hat{p}_1$ is vertically above $\hat{s}_1$, and $\hat{p}_2 \in \hat{s}_1$.

The second iteration. We are now entering the second iteration of our process. First, notice that $\hat{p}_2$ cannot be $\hat{p}_1$ since $\hat{p}_2 = \hat{p}_1^l$, which cannot be $\hat{p}_1$. Our goal in this iteration is to find a *candidate point* p' to replace $\hat{p}_2$ so that $P'_{\text{opt}} \cup \{p'\}$ also forms a hitting set of S. Consequently, if $p' \notin Q$, then we can use p' as our target p^*; otherwise, we need to guarantee $p' \neq \hat{p}_1$ so that our process will not enter a loop. The discussion here is more involved than in the first iteration.

Since $\hat{p}_2 \in Q$, by Observation 6, S has a disk $\hat{s}_2$ vertically below $\hat{p}_2$ such that $P(\hat{s}_2)$ contains both a point left of $\hat{p}_2$, denoted by $\hat{p}_2^l$, and a point right of $\hat{p}_2$, denoted by $\hat{p}_2^r$. Further, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_2^r)$. Depending on whether $\hat{s}_2$ is in $S(A_1)$, there are two cases.

- If $\hat{s}_2 \notin S(A_1)$, since $\hat{p}_2$ does not hit $\hat{s}_2$ and $S(\hat{p}_1) \subseteq S(A_1) \cup S(\hat{p}_2)$, we obtain $\hat{s}_2 \notin S(\hat{p}_1)$. Now we can basically repeat our argument from the first iteration. Since $\hat{p}_2$ does not hit $\hat{s}_2$ and $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution, P'_{opt} must have a point p that hits $\hat{s}_2$. Clearly, p is either left or right of $\hat{p}_2$.
  We first assume that p is right of $\hat{p}_2$. Since $\hat{p}_2^r$ refers to an arbitrary point to the right of $\hat{p}_2$ that hits $\hat{s}_2$ and p is also a point to right of $\hat{p}_2$ that hits $\hat{s}_2$, for notational convenience, we let $\hat{p}_2^r$ refer to p. As such, $\hat{p}_2^r$ is in P'_{opt}.
  We let $\hat{p}_2^l$ be our candidate point, which satisfies our need as discussed above for p'. Indeed, since $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is an optimal solution, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_2^r)$, and $\hat{p}_2^r \in P'_{\text{opt}}$, we obtain that $P'_{\text{opt}} \cup \{\hat{p}_2^l\}$ also forms a hitting set of S. Furthermore, since $\hat{p}_2^l$ hits $\hat{s}_2$ while $\hat{p}_1$ does not, we know that $\hat{p}_2^l \neq \hat{p}_1$. Therefore, if $\hat{p}_2^l \notin Q$, then we can use $\hat{p}_2^l$ as our target p^* and we are done with the process. Otherwise, we let $\hat{p}_3 = \hat{p}_2^l$ and then enter the third iteration. In this case, we let $A_2 = A_1 \cup \{\hat{p}_2^r\}$. According to our above discussion, $A_2 \subseteq P'_{\text{opt}}$, $S(\hat{p}_2) \subseteq S(A_2) \cup S(\hat{p}_3)$, $\{\hat{p}_3\} \cup P'_{\text{opt}}$ is a feasible solution, $\hat{p}_2$ is vertically above $\hat{s}_2$, and $\hat{p}_3 \in \hat{s}_2$.
  The above discussed the case where p is right of $\hat{p}_2$. If p is left of $\hat{p}_2$, then the analysis is symmetric.[4]
- If $\hat{s}_2 \in S(A_1)$, we let $\hat{p}_2^l$ be our candidate point. We show below that it satisfies our need as discussed above for p', i.e., $\{\hat{p}_2^l\} \cup P'_{\text{opt}}$ forms a hitting set of S and $\hat{p}_2^l \neq \hat{p}_1$.
  Indeed, since $A_1 = \{\hat{p}_1^r\}$ and $\hat{s}_2 \in S(A_1)$, $\hat{s}_2$ is hit by $\hat{p}_1^r$. Since $\hat{p}_1^r$ is to the right of $\hat{p}_1$, and $\hat{p}_2$, which is $\hat{p}_1^l$, is to the left of $\hat{p}_1$, we obtain that $\hat{p}_1^r$ is to the right of $\hat{p}_2$. Since $\hat{p}_2^l$ hits $\hat{s}_2$, $\hat{p}_2^l$ is to the left of $\hat{p}_2$, $\hat{p}_1^r$ hits $\hat{s}_2$, and $\hat{p}_1^r$ is to the right of $\hat{p}_2$, by Observation 6, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_1^r)$, i.e., $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(A_1)$. Since $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution and $A_1 \subseteq P'_{\text{opt}}$, it follows that $\{\hat{p}_2^l\} \cup P'_{\text{opt}}$ is also a feasible solution. On the other hand, since $\hat{p}_2^l$ is to the left of $\hat{p}_2$ while $\hat{p}_2$ (which is $\hat{p}_1^l$) is to the left of $\hat{p}_1$, we know that $\hat{p}_2^l$ is to the left of $\hat{p}_1$ and thus $\hat{p}_2^l \neq \hat{p}_1$.

[4] More specifically, if $\hat{p}_2^r \notin Q$, then we can use $\hat{p}_2^r$ as our target p^* and the process is complete. Otherwise, we let $\hat{p}_3 = \hat{p}_2^r$ and enter the third iteration; in this case, we let $A_2 = A_1 \cup \{\hat{p}_2^l\}$.

As such, if $\hat{p}_2^l \notin Q$, we can use $\hat{p}_2^l$ as our target p^* and we are done with the process. Otherwise, we let $\hat{p}_3 = \hat{p}_2^l$ and continue with the third iteration. In this case, we let $A_2 = A_1$. According to our above discussion, $A_2 \subseteq P'_{\text{opt}}$, $S(\hat{p}_2) \subseteq S(A_2) \cup S(\hat{p}_3)$, $P'_{\text{opt}} \cup \{\hat{p}_3\}$ is a feasible solution, $\hat{p}_2$ is vertically above $\hat{s}_2$, and $\hat{p}_3 \in \hat{s}_2$.

This finishes the second iteration of the process.

Inductive step. In general, suppose that we are entering the i-th iteration of the process with the point $\hat{p}_i \in Q$, $i \geq 2$. We make the following inductive hypothesis for i.

1. We have points $\hat{p}_k \in Q$ for all $k = 1, 2, \ldots, i-1$ in the previous $i-1$ iterations such that $\hat{p}_i \neq \hat{p}_k$ for any $1 \leq k \leq i-1$
2. We have subsets A_k for all $k = 1, 2, \ldots, i-1$ such that $A_1 \subseteq A_2 \subseteq \cdots \subseteq A_{i-1} \subseteq P'_{\text{opt}}$, and $S(\hat{p}_k) \subseteq S(A_k) \cup S(\hat{p}_{k+1})$ holds for each $1 \leq k \leq i-1$.
3. For any $1 \leq k \leq i$, $\{\hat{p}_k\} \cup P'_{\text{opt}}$ is a feasible solution.
4. We have disks $\hat{s}_k \in S$ for $k = 1, 2, \ldots, i-1$ such that $\hat{s}_k$ is vertically below $\hat{p}_k$ and $\hat{p}_{k+1} \in \hat{s}_k$.

Our previous discussion already established the hypothesis for $i = 2$ and $i = 3$. Next, we proceed with the i-th iteration argument for any general $i \geq 4$. Our goal is to find a candidate point $\hat{p}_{i+1}$ such that $P'_{\text{opt}} \cup \{\hat{p}_{i+1}\}$ is a feasible solution and the inductive hypothesis still holds for $i+1$.

Since $\hat{p}_i \in Q$, by Observation 6, there is a disk $\hat{s}_i$ vertically below $\hat{p}_i$ such that $P(\hat{s}_i)$ has a point left of $\hat{p}_i$, denoted by $\hat{p}_i^l$, and a point right of $\hat{p}_i$, denoted by $\hat{p}_i^r$. Furthermore, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$. Depending on whether $\hat{s}_i$ is in $S(A_{i-1})$, there are two cases.

1. If $\hat{s}_i \notin S(A_{i-1})$, then since $\hat{s}_i$ does not contain $\hat{p}_i$ and $P'_{\text{opt}} \cup \{\hat{p}_i\}$ is a feasible solution, P'_{opt} must have a point p that hits $\hat{s}_i$. Clearly, p is to the left or right of $\hat{p}_i$. Without loss of generality, we assume that p is to the right of $\hat{p}_i$. Since $\hat{p}_i^r$ refers to an arbitrary point to the right of $\hat{p}_i$ that hits $\hat{s}_i$ and p is also a point to the right of $\hat{p}_i$ that hits $\hat{p}_i$, for notational convenience, we let $\hat{p}_i^r$ refer to p. As such, $\hat{p}_i^r$ is in P'_{opt}.
   We let $\hat{p}_{i+1}$ be $\hat{p}_i^l$ and define $A_i = A_{i-1} \cup \{\hat{p}_i^r\}$. In the following, we argue that the inductive hypothesis holds.
   - First of all, by definition, $\hat{p}_i$ is vertically above $\hat{s}_i$ and $\hat{p}_{i+1} \in \hat{s}_i$. Hence, the fourth statement of the hypothesis holds.
   - Since $\{\hat{p}_i\} \cup P'_{\text{opt}}$ is a feasible solution, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$, $\hat{p}_i^r \in P'_{\text{opt}}$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain that $\{\hat{p}_{i+1}\} \cup P'_{\text{opt}}$ is a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_i = A_{i-1} \cup \{\hat{p}_i^r\}$, $A_{i-1} \subseteq P'_{\text{opt}}$ by inductive hypothesis, and $\hat{p}_i^r \in P'_{\text{opt}}$, we obtain $A_i \subseteq P'_{\text{opt}}$. Furthermore, since $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$, $\hat{p}_i^r \in A_i$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we have $S(\hat{p}_i) \subseteq S(A_i) \cup S(\hat{p}_{i+1})$. This proves the second statement of the hypothesis.
   - For any point $\hat{p}_k$ with $1 \leq k \leq i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{p}_k \neq \hat{p}_{i+1}$. To this end, since $\hat{s}_i$ is hit by $\hat{p}_{i+1}$, it suffices to show that $\hat{s}_i$ is not hit by $\hat{p}_k$. Indeed, by the inductive hypothesis, $S(\hat{p}_k) \subseteq S(A_k) \cup S(\hat{p}_{k+1})$ and $S(\hat{p}_{k+1}) \subseteq S(A_{k+1}) \cup S(\hat{p}_{k+2})$. Hence, $S(\hat{p}_k) \subseteq S(A_k) \cup S(A_{k+1}) \cup S(\hat{p}_{k+2})$. As $S(A_k) \subseteq S(A_{k+1})$, we obtain $S(\hat{p}_k) \subseteq S(A_{k+1}) \cup S(\hat{p}_{k+2})$. Following the same argument, we can derive $S(\hat{p}_k) \subseteq S(A_{i-1}) \cup S(\hat{p}_i)$. Now that $\hat{s}_i \notin S(A_{i-1})$ and $\hat{s}_i$ is not hit by $\hat{p}_i$, we obtain that $\hat{s}_i$ is not hit by $\hat{p}_k$.

2. If $\hat{s}_i \in S(A_{i-1})$, then $\hat{s}_i$ is hit by a point of A_{i-1}, say p. As $\hat{p}_i \notin \hat{s}_i$, p is left or right of $\hat{p}_i$. Without loss of generality, we assume that p is to the right of $\hat{p}_i$.
   We let $\hat{p}_{i+1}$ be $\hat{p}_i^l$ and define $A_i = A_{i-1}$. We show in the following that the inductive hypothesis holds.
   - By definition, $\hat{p}_i$ is vertically above $\hat{s}_i$ and $\hat{p}_{i+1} \in \hat{s}_i$. Hence, the fourth statement of the hypothesis holds.
   - Since $\hat{s}_i$ is hit by both p and $\hat{p}_i^l$, $\hat{p}_i^l$ is to the left of $\hat{p}_i$, and p is to the right of $\hat{p}_i$, by Observation 6, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(p)$. Further, since $\{\hat{p}_i\} \cup P'_{\text{opt}}$ is a feasible solution, $p \in A_{i-1} \subseteq P'_{\text{opt}}$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain that $\{\hat{p}_{i+1}\} \cup P'_{\text{opt}}$ is also a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_{i-1} \subseteq P'_{\text{opt}}$ by the inductive hypothesis and $A_i = A_{i-1}$, we have $A_i \subseteq P'_{\text{opt}}$. As discussed above, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(p)$. Since $p \in A_{i-1} = A_i$ and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain $S(\hat{p}_i) \subseteq S(A_i) \cup S(\hat{p}_{i+1})$. This proves the second statement of the hypothesis.
   - For any point $\hat{p}_k$ with $1 \leq k \leq i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{p}_k \neq \hat{p}_{i+1}$. Depending on whether $x(\hat{p}_i) < x(\hat{p}_k)$, there are two cases (note that $\hat{p}_i \neq \hat{p}_k$ by our hypothesis and thus $x(\hat{p}_i) \neq x(\hat{p}_k)$ due to our general position assumption).
     - If $x(\hat{p}_i) < x(\hat{p}_k)$, then since $\hat{p}_{i+1} = \hat{p}_i^l$, we have $x(\hat{p}_{i+1}) < x(\hat{p}_i) < x(\hat{p}_k)$. Hence, $\hat{p}_k \neq \hat{p}_{i+1}$.
     - If $x(\hat{p}_k) < x(\hat{p}_i)$, then we can prove $\hat{p}_k \notin \hat{s}_i$. This implies that $\hat{p}_k \neq \hat{p}_{i+1}$ as $\hat{p}_{i+1} \in \hat{s}_i$. The proof of $\hat{p}_k \notin \hat{s}_i$, which is quite technical and lengthy, is omitted due to the space limit.

     This proves the first statement of the hypothesis.

   This proves that the inductive hypothesis still holds for $i+1$.

According to the inductive hypothesis, each iteration of the process finds a new candidate point $\hat{p}_i$ such that $P'_{\text{opt}} \cup \{\hat{p}_i\}$ is a feasible solution. If $\hat{p}_i \notin Q$, then we can use $\hat{p}_i$ as our target point p^* and we are done with the process. Otherwise, we continue with the next iteration. Since each iteration finds a new candidate point (that was never used before) and $|Q|$ is finite, eventually we will find a candidate point $\hat{p}_i$ that is not in Q.

This completes the proof of the lemma. ◀

5 Algorithm implementation

In this section, we present the implementation of our algorithm. In particular, we describe how to implement the first two steps of the algorithm: (1) Compute $a(i)$ and $b(i)$ for all disks $s_i \in S$; (2) find the subset Q of all prunable points from P.

The following lemma gives the implementation for the first step of the algorithm.

▶ **Lemma 8.** *Computing $a(i)$ and $b(i)$ for all disks $s_i \in S$ can be done in $O(m \log^2 n + (m+n)\log(m+n))$ time.*

Proof. We only discuss how to compute $a(i)$ since computing $b(i)$ can be done analogously.

Recall that points of P are indexed in ascending order of their x-coordinates as $p_1, \ldots, p_n$. Let T be a complete binary search tree whose leaves from left to right correspond to points of P in their index order. Since $n = |P|$, the height of T is $O(\log n)$. For each node $v \in T$, let P_v denote the subset of points of P in the leaves of the subtree rooted at v. Our algorithm is based on the following observation: a disk $s_i \in S$ contains a point of P_v if and only if s_i contains the closest point of P_v to c_i, where c_i is the center of s_i. In light of this observation, we construct the Voronoi diagram for P_v, denoted by VD_v, and build a point location data structure on VD_v so that each point location query can be answered in $O(\log n)$ time [13,20]. For time analysis, after VD_v is computed, building the point location data structure on VD_v takes $O(|P_v|)$ time [13,20]. To compute VD_v, if we do so from scratch, then it takes $O(|P_v|\log|P_v|)$ time. However, using Kirkpatrick's algorithm [19], we can compute VD_v in $O(|P_v|)$ time by merging the Voronoi diagrams VD_u and VD_w for the two children u and w of v, since $P_v = P_u \cup P_w$. As such, if we construct the Voronoi diagrams for all nodes of T in a bottom-up manner, the total time is linear in $\sum_{v\in T}|P_v|$, which is $O(n\log n)$.

For each disk $s_i \in S$, we can compute $a(i)$ using T, as follows. Starting from the root of T, for each node v, we do the following. Let u and w be the left and right children of v, respectively. First, we determine whether s_i contains a point of P_u. To this end, using a point location query on VD_u, we find the point p of P_v closest to c_i. As discussed above, s_i contains a point of P_u if and only if $p \in s_i$. If $p \in s_i$, then we proceed with $v = u$; otherwise, we proceed with $v = w$. In this way, $a(i)$ can be computed after searching a root-to-leaf path of T, which has $O(\log n)$ nodes as the height of T is $O(\log n)$. Because we spend $O(\log n)$ time on each node, the total time to compute $a(i)$ is $O(\log^2 n)$. The time for computing $a(i)$ for all disks $s_i \in S$ is thus $O(m\log^2 n)$.

In summary, the overall time to compute $a(i)$ for all disks $s_i \in S$ is bounded by $O(m\log^2 n + (n+m)\log(n+m))$. ◀

With $a(i)$ and $b(i)$ computed in Lemma 8, Lemma 10 finds all prunable points of P. The algorithm of Lemma 10 relies on the following observation.

▶ **Observation 9.** *For any point $p_k \in P$, p_k is prunable if and only if there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) \le k \le b(i)$.*

Proof. If p_k is prunable, then by definition there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) < k < b(i)$.

On the other hand, suppose that there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) \le k \le b(i)$. By the definition of a_i, s_i contains the point $p_{a(i)}$. Since $p_k \notin s_i$, we obtain $k \ne a(i)$. By a similar argument, we have $k \ne b(i)$. As such, since $a(i) \le k \le b(i)$, we can derive $a(i) < k < b(i)$. Therefore, p_k is prunable. ◀

▶ **Lemma 10.** *All prunable points of P can be found in $O((n+m)\log(n+m))$ time.*

Proof. Recall that ℓ denotes the x-axis. We define T as the standard segment tree [11, Section 10.3] on the n points of ℓ whose x-coordinates are equal to $1, 2, \ldots, n$, respectively. The height of T is $O(\log n)$. For each disk $s_i \in S$, let I_i denote the interval $[a(i), b(i)]$ of ℓ. Following the definition of the standard segment tree [11, Section 10.3], we store I_i in $O(\log n)$ nodes of T. For each node $v \in T$, let S_v denote the subset of disks s_i of S whose interval I_i is stored at v. As such, $\sum_{v\in T}|S_v| = O(m\log n)$.

Consider a point $p_k \in P$. The tree T has a leaf corresponding to a point of ℓ whose x-coordinate is equal to k, called *leaf-k*. Let π_k denote the path of T from the root to leaf-k. Following the definition of the segment tree, we have the following observation: $\bigcup_{v\in\pi_k} S_v = \{s_i \mid s_i \in S, a(i) \le k \le b(i)\}$. By Observation 9, to determine whether p_k is prunable, it suffices to determine whether there is a node $v \in \pi_k$ such that S_v has a disk s_i that does not contain p_k. Recall that all points of P are above ℓ while the centers of all disks of S are below ℓ. Let C_v denote the common intersection of all disks of S_v in the halfplane above ℓ. Observe that S_v has a disk s_i that does not contain p_k if and only if p_k is outside C_v. Based on this observation, for each node $v \in T$, we compute C_v and store it at v. Due to the single-intersection property that the upper arcs of every two disks of S intersect at most once, C_v has $O(|S_v|)$ vertices; in addition, by adapting Graham's scan, C_v can be computed in $O(|S_v|)$ time if the centers of all the disks of S_v are sorted by x-coordinate (due to the non-containment property, this is also the order of the disks sorted by the left or right endpoints of their upper arcs). Assuming that the sorted lists of S_v as above are available for all nodes $v \in T$, the total time for constructing C_v for all nodes $v \in T$ is linear in $\sum_{v\in T} |S_v|$, which is $O(m \log n)$. We show that the sorted lists of S_v for all nodes $v \in T$ can be computed in $O(m \log m + m \log n)$ time, as follows. At the start of the algorithm, we sort all disks of S by the x-coordinates of their centers in $O(m \log m)$ time. Then, for each disk s_i of S following this sorted order, we find the nodes v of T where the interval I_i should be stored, and add s_i to S_v, which can be done in $O(\log n)$ time [11, Section 7.4]. In this way, after all disks of S are processed as above, S_v for every node $v \in T$ is automatically sorted. As such, all processing work on T together takes $O((m + n) \log(m + n))$ time.

For each point $p_k \in P$, to determine whether p_k is prunable, following the above discussion, we determine whether p_k is outside C_v for each node $v \in \pi_k$. Deciding whether p_k is outside C_v can be done in $O(\log m)$ time. Indeed, since the centers of all disks are below ℓ, the boundary of C_v consists of a segment on ℓ bounding C_v from below and an x-monotone curve bounding C_v from above. The projections of the vertices of C_v onto ℓ partition ℓ into a set $\mathcal{I}_v$ of $O(|S_v|)$ intervals. If we know the interval of $\mathcal{I}_v$ that contains $x(p_k)$, the x-coordinate of p_k, then whether p_k is outside C_v can be determined in $O(1)$ time. Clearly, we can find the interval of $\mathcal{I}_v$ that contains $x(p_k)$ in $O(\log m)$ time by binary search. In this way, whether p_k is prunable can be determined in $O(\log m \log n)$ time as π_k has $O(\log n)$ nodes. The time can be improved to $O(\log m + \log n)$ using fractional cascading [7], as follows.

We construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(m \log n)$ time [7] since the total number of such intervals is $O(m \log n)$. With the fractional cascading data structure, for each point $p_k \in P$, we only need to do binary search on the set of the intervals stored at the root of T to find the interval containing $x(p_k)$, which takes $O(\log(m \log n))$ time. Subsequently, following the path π_k in a top-down manner, the interval of $\mathcal{I}_v$ containing $x(p_k)$ for each node $v \in \pi_k$ can be determined in $O(1)$ time [7]. As such, whether p_k is prunable can be determined in $O(\log n + \log m)$ time. Hence, the total time for checking all the points $p_k \in P$ is $O(n \log(m + n))$.

In summary, the time complexity of the overall algorithm for finding all prunable disks of S is bounded by $O((n + m) \log(n + m))$. ◀

With Lemmas 8 and 10, Theorem 4 is proved.

An algebraic decision tree algorithm. In the algebraic decision tree model, where only comparisons are counted towards time complexities, the problem can be solved in $O((n + m) \log(n + m))$ time, i.e., using $O((n + m) \log(n + m))$ comparisons. To this end, observe that the entire algorithm, with the exception of Lemma 8, takes $O((n + m) \log(n + m))$ time. As such, we only need to show that Lemma 8 can be solved using $O((n + m) \log(n + m))$ comparisons. For this, notice that the factor $O(m \log^2 n)$ in the algorithm of Lemma 8 is caused by the point location queries on the Voronoi diagrams VD_v. The number of point location queries is $O(m \log n)$. The total combinatorial complexity of the Voronoi diagrams VD_v of all nodes $v \in T$ is $O(n \log n)$. To answer these point location queries, we employ a method recently introduced by Chan and Zheng [6]. In particular, by applying [6, Theorem 7.2], all point location queries can be solved using $O((n+m) \log(n+m))$ comparisons (specifically, following the notation in [6, Theorem 7.2], we have $t = O(n)$, $L = O(n \log n)$, $M = O(m \log n)$, and $N = O(n + m)$ in our problem; according to the theorem, all point location queries can be answered using $O(L + M + N \log N)$ comparisons, which is $O((n + m) \log(n + m))$).

The unit-disk case. If all disks of S have the same radius (and the points of P are separated from the centers of all disks of S by the x-axis ℓ), then as discussed in Section 1 this problem can be solved in $O((n+m)\log(n+m))$ time by reducing it to a line-separable unit-disk coverage problem and then applying the algorithm in [23]. Here, we show that our approach can provide an alternative algorithm with the same runtime.

We apply the same algorithm as above. Observe that the algorithm, except for Lemma 8, runs in $O((n+m)\log(n+m))$ time. Hence, it suffices to show that Lemma 8 can be implemented in $O((n+m)\log(n+m))$ time for the unit-disk case, which is done in the following lemma.

▶ **Lemma 11.** *If all disks of S have the same radius, then $a(i)$ and $b(i)$ for all disks $s_i \in S$ can be computed in $O((n+m)\log(n+m))$ time.*

Proof. We only discuss how to compute $a(i)$ since the algorithm for $b(i)$ is similar. We modify the algorithm in the proof of Lemma 8 and follow the notation there.

For any disk $s_i \in S$, to compute $a(i)$, recall that a key subproblem is to determine whether s_i contains a point of P_v for a node $v \in T$. To solve the subproblem, the algorithm of Lemma 8 uses Voronoi diagrams. Here, we use a different approach by exploring the property that all disks of S have the same radius, say r. For any point $p \in P$, let D_p denote the disk of radius r and centered at p. Define $\mathcal{D}_v = \{D_p \mid p \in P_v\}$. For each point $p \in P$, since p is above the axis ℓ, the portion of the boundary of D_p below ℓ is an arc on the lower half circle of the boundary of D_p, and we call it the *lower arc* of D_p. Let $\mathcal{L}_v$ denote the lower envelope of ℓ and the lower arcs of all disks of $\mathcal{D}_v$. Our method is based on the observation that s_i contains a point of P_v if and only if c_i is above $\mathcal{L}_v$, where c_i is the center of s_i.

In light of the above discussion, we construct $\mathcal{L}_v$ for every node $v \in T$. Since all disks of $\mathcal{D}_v$ have the same radius and all their centers are above ℓ, the lower arcs of every two disks of $\mathcal{D}_v$ intersect at most once. Due to this single-intersection property, $\mathcal{L}_v$ has at most $O(|P_v|)$ vertices. To see this, we can view the lower envelope of each lower arc of $\mathcal{D}_v$ and ℓ as an extended arc. Every two such extended arcs still cross each other at most once and therefore their lower envelope has $O(|P_v|)$ vertices following the standard Davenport-Schinzel sequence argument [27] (see also [5, Lemma 3] for a similar problem). Notice that $\mathcal{L}_v$ is exactly the lower envelope of these extended arcs and thus $\mathcal{L}_v$ has $O(|P_v|)$ vertices. Note also that $\mathcal{L}_v$ is x-monotone. In addition, given $\mathcal{L}_u$ and $\mathcal{L}_w$, where u and w are the two children of v, $\mathcal{L}_v$ can be computed in $O(|P_v|)$ time by a straightforward line sweeping algorithm. As such, if we compute $\mathcal{L}_v$ for all nodes $v \in T$ in a bottom-up manner, the total time is linear in $\sum_{v\in T}|P_v|$, which is $O(n\log n)$.

For each disk $s_i \in S$, we now compute $a(i)$ using T, as follows. Starting from the root of T, for each node v, we do the following. Let u and w be the left and right children of v, respectively. We first determine whether c_i is above $\mathcal{L}_u$; since $|P_u| \leq n$, this can be done in $O(\log n)$ time by binary search. More specifically, the projections of the vertices of $\mathcal{L}_u$ onto ℓ partition ℓ into a set $\mathcal{I}_u$ of $O(P_u)$ intervals. If we know the interval of $\mathcal{I}_u$ that contains $x(c_i)$, the x-coordinate of c_i, then whether c_i is above $\mathcal{L}_u$ can be determined in $O(1)$ time. Clearly, finding the interval of $\mathcal{I}_u$ containing $x(c_i)$ can be done by binary search in $O(\log n)$ time. If c_i is above $\mathcal{L}_u$, then s_i must contain a point of P_u; in this case, we proceed with $v = u$. Otherwise, we proceed with $v = w$. In this way, $a(i)$ can be computed after searching a root-to-leaf path of T, which has $O(\log n)$ nodes as the height of T is $O(\log n)$. Because we spend $O(\log n)$ time on each node, the total time for computing $a(i)$ is $O(\log^2 n)$. As in Lemma 10, the time can be improved to $O(\log n)$ using fractional cascading [7], as follows.

We construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(n\log n)$ time [7] since the total number of such intervals is $O(n\log n)$. With the fractional cascading data structure, for each disk $s_i \in S$, we only need to do binary search on the set of the intervals stored at the root of T to find the interval containing $x(c_i)$, which takes $O(\log n)$ time. After that, the interval of $\mathcal{I}_u$ containing $x(c_i)$ for each node u in the algorithm as discussed above can be determined in $O(1)$ time [7]. As such, $a(i)$ can be computed in $O(\log n)$ time. Hence, computing $a(i)$ for all disks $s_i \in S$ takes $O(m \log n)$ time.

In summary, the total time to compute $a(i)$ for all disks $s_i \in S$ is bounded by $O((n+m)\log(n+m))$ time. ◀

References

1 Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), and the 10th International Conference on Randomization and Computation (RANDOM)*, pages 3–14, 2006. doi:10.1007/11830924_3.

2 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 1983. doi:10.1145/800061.808735.

3 Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018. doi:10.1016/j.dam.2017.12.018.

4 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry: Theory and Applications*, 47:112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.

5 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

6 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. *ACM Transactions on Algorithms*, 2023. doi:10.1145/3591357.

7 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986. doi:10.1007/BF01840440.

8 Francisco Claude, Gautam K. Das, Reza Dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, and Alejandro Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discrete Mathematics, Algorithms and Applications*, 2:77–88, 2010. doi:10.1142/S1793830910000486.

9 Gruia Călinescu, Ion I. Măndoiu, Peng-Jun Wan, and Alexander Z. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mobile Networks and Applications*, 9:101–111, 2004. doi:10.1023/B:MONE.0000013622.63511.57.

10 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Homogeneous 2-hop broadcast in 2D. *Computational Geometry: Theory and Applications*, 43:182–190, 2010. doi:10.1016/j.comgeo.2009.06.005.

11 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry — Algorithms and Applications.* Springer-Verlag, Berlin, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

12 Stephane Durocher and Robert Fraser. Duality for geometric set cover and geometric hitting set problems on pseudodisks. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015. URL: https://research.cs.queensu.ca/cccg2015/CCCG15-papers/10.pdf.

13 Herbert Edelsbrunner, Leonidas J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. doi:10.1137/0215023.

14 Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990. doi:10.1145/77635.77639.

15 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the VC-dimension is small. *Information Processing Letters*, 95:358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

16 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants.* PhD thesis, Duke University, 2011. URL: https://dukespace.lib.duke.edu/server/api/core/bitstreams/0d37dabc-42a5-4bc8-b4e9-e69b263a10ca/content.

17 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3:65–85, 2012. doi:10.20382/jocg.v3i1a4.

18 Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 David G. Kirkpatrick. Efficient computation of continuous skeletons. In *20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, 1979. doi:10.1109/SFCS.1979.15.

20 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. doi:10.1137/0212002.

21 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 898–909, 2015. doi:10.1007/978-3-662-47672-7_73.

22 Gang Liu and Haitao Wang. Geometric hitting set for line-constrained disks. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 574–587, 2023. doi:10.1007/978-3-031-38906-1_38.

23 Gang Liu and Haitao Wang. On the line-separable unit-disk coverage and related problems. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC)*, pages 51:1–51:14, 2023. Full version available at arXiv:2309.03162.

24 El O. Mourad, Fohlin Helena, and Srivastav Anand. A randomised approximation algorithm for the hitting set problem. *Theoretical Computer Science*, 555:23–34, 2014. doi:10.1007/978-3-642-36065-7_11.

25 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010. doi:10.1007/s00454-010-9285-9.

26 Logan Pedersen and Haitao Wang. Algorithms for the line-constrained disk coverage and related problems. *Computational Geometry: Theory and Applications*, 105-106:101883:1–18, 2022. doi:10.1016/j.comgeo.2022.101883.

27 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1996.

28 Haitao Wang and Jie Xue. Algorithms for halfplane coverage and related problems. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 79:1–79:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.79.