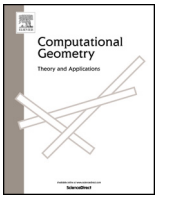
Computational Geometry: Theory and Applications 123 (2024) 102122

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

journal homepage: www.elsevier.com/locate/comgeo



# On the line-separable unit-disk coverage and related problems ☆

Gang Liu *, Haitao Wang

*Kahlert School of Computing, University of Utah, Salt Lake City, UT 84112, USA*

ARTICLE INFO

*Article history:*
Received 13 December 2023
Received in revised form 29 June 2024
Accepted 8 July 2024
Available online 22 July 2024

*Keywords:*
Disk coverage
Line-separable
Unit-disk
Line-constrained
Half-planes

ABSTRACT

Given a set $P$ of $n$ points and a set $S$ of $m$ disks in the plane, the disk coverage problem asks for a smallest subset of disks that together cover all points of $P$. The problem is NP-hard. In this paper, we consider a line-separable unit-disk version of the problem where all disks have the same radius and their centers are separated from the points of $P$ by a line $\ell$. We present an $O((n+m)\log(n+m))$ time algorithm for the problem. This improves the previously best result of $O(nm+n\log n)$ time. Our techniques also solve the line-constrained version of the problem, where centers of all disks of $S$ are located on a line $\ell$ while points of $P$ can be anywhere in the plane. Our algorithm runs in $O((n+m)\log(m+n)+m\log m\log n)$ time, which improves the previously best result of $O(nm\log(m+n))$ time. In addition, our results lead to an algorithm of $O(n^3\log n)$ time for a half-plane coverage problem (given $n$ half-planes and $n$ points, find a smallest subset of half-planes covering all points); this improves the previously best algorithm of $O(n^4\log n)$ time. Further, if all half-planes are lower ones, our algorithm runs in $O(n\log n)$ time while the previously best algorithm takes $O(n^2\log n)$ time.

© 2024 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

## 1. Introduction

Given a set $P$ of $n$ points and a set $S$ of $m$ disks in the plane, the *disk coverage* problem asks for a smallest subset of disks such that every point of $P$ is covered by at least one disk in the subset. The problem is NP-hard, even if all disks have the same radius [20,27]. Polynomial time approximation algorithms have been proposed for the problem and many of its variants, e.g., [1,7,9,10,21,25].

Polynomial time exact algorithms are known for certain special cases. If all points of $P$ are inside a strip bounded by two parallel lines and the centers of all disks lie outside the strip, then the problem is solvable in polynomial time [4]. If all disks of $S$ contain the same point, polynomial time algorithms also exist [15,16]; in particular, applying the result in [9] (i.e., Corollary 1.7) yields an $O(mn^2(m+n))$ time algorithm. In order to devise an efficient approximation algorithm for the general coverage problem (without any constraints), the *line-separable* version was considered in the literature [4,8,14], where disk centers are separated from the points by a given line $\ell$. A polynomial time 4-approximation algorithm is given in

☆ A preliminary version of this paper appeared in *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC 2023)*. This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356.

* Corresponding author.
*E-mail addresses:* u0866264@utah.edu (G. Liu), haitao.wang@utah.edu (H. Wang).

https://doi.org/10.1016/j.comgeo.2024.102122
0925-7721/© 2024 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

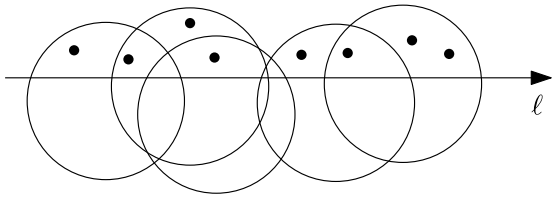



**Fig. 1.** Illustrating the line-separable unit-disk case.

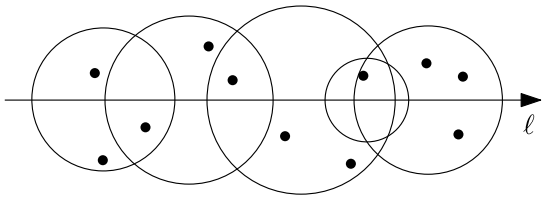



**Fig. 2.** Illustrating the line-constrained case (all disks are centred on $\ell$).

[8]. Ambühl et al. [4] derived an exact algorithm of $O(m^2n)$ time. An improved $O(nm+n\log n)$ time algorithm is presented in [14] and another algorithm in [28] runs in $O(n\log n+m^2\log n)$ in the worst case.

The *line-constrained* version of the disk coverage problem has also been studied, where disk centers are on the $x$-axis while points of $P$ can be anywhere in the plane. Pedersen and Wang [28] considered the weighted case in which each disk has a weight and the objective is to minimize the total weight of the disks in the subset that cover all points. Their algorithm runs in $O((m+n)\log(m+n)+\kappa\log m)$ time, where $\kappa$ 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, as well as the $L_\infty$ and $L_1$ cases, where the disks are squares and diamonds, respectively [28]. The 1D problem where disks become segments on a line and points are on the same line is also solvable in $O((m+n)\log(m+n))$ [28]. Other types of line-constrained coverage problems have also been studied in the literature, e.g., [3,5,6,24].

A related problem is when disks of $S$ are half-planes. For the weighted case, Chan and Grant [9] 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 techniques of [28] solve the problem in $O(n^2\log n)$ time. For the general case where both upper and lower half-planes are present, Har-Peled and Lee [22] solved the problem in $O(n^5)$ time. Pedersen and Wang [28] showed that the problem can be reduced to $O(n^2)$ instances of the lower-only case problem and thus can be solved in $O(n^4\log n)$ time. To the best of our knowledge, we are not aware of any previous work particularly on the unweighted half-plane coverage problem.

### 1.1. Our result

We assume that $\ell$ is the $x$-axis and all disk centers are below or on $\ell$ while all points of $P$ are above or on $\ell$. We consider the line-separable version of the disk coverage problem with the following *single-intersection condition*: For any two disks, their boundaries intersect at most once in the half-plane above $\ell$. Note that this condition is satisfied in both the unit-disk case (see Fig. 1) and the line-constrained case (see Fig. 2; more to explain below). Hence, an algorithm for this line-separable single-intersection case works for both the unit-disk case and the line-constrained case. Note that all problems considered in this paper are unweighted case in the $L_2$ metric.

For the above line-separable single-intersection problem, we give an algorithm of $O((n+m)\log(m+n)+m\log m\log n)$ time in Section 3. Based on observations, we find that some disks are "useless" and thus can be pruned from $S$. After pruning those useless disks, the remaining disks have certain properties so that we can reduce the problem to the 1D problem, which can then be easily solved. The overall algorithm is fairly simple conceptually. One challenge, however, is to show the correctness, that is, to prove why those "useless" disks are indeed useless. The proof is rather lengthy and technical. The bottleneck of the algorithm is to find those useless disks.

**The line-constrained problem.** Observe that the line-constrained problem where all disks of $S$ are centered on a line $\ell$ while points of $P$ can be anywhere in the plane is also a special case of the line-separable single-intersection problem. Indeed, for each point $p$ of $P$ below $\ell$, we could replace $p$ by its symmetric point with respect to $\ell$; in this way, we can obtain a set of points that are all above $\ell$. It is not difficult to see that an optimal solution using this new set of points is also an optimal solution for $P$. Furthermore, since disks are centered on $\ell$, although their radii may not be equal, the boundaries of any two disks intersect at most once above $\ell$. Hence, the problem is an instance of the line-separable single-intersection case. As such, applying our algorithm in Section 3 solves the line-constrained problem in $O((n+m)\log(m+n)+m\log m\log n)$ time; this improves the previous algorithm in [28], which runs in $O(n\log n+m^2\log m)$ time in the worst case.

**The unit-disk case.** To solve the line-separable unit-disk case, the algorithm in Section 3 still works. However, by making use of the property that all disks have the same radius, we further improve the runtime to $O((m+n)\log(m+n))$ in Section 4. This improves the $O(nm+n\log n)$ time algorithm in [14] and the $O(n\log n+m^2\log n)$ time algorithm in [28]. The main idea of the improvement (over the algorithm in Section 3) is to explore the property that all disks have the same radius.

**The half-plane coverage problem.** As in [28], our techniques also solve the half-plane coverage problem. Specifically, for the lower-only case, let $\ell$ be a horizontal line that is below all points of $P$. If we consider each half-plane as a unit disk of infinite radius with center below $\ell$, then the problem becomes an instance of the line-separable unit-disk coverage problem. Therefore, applying our result leads to an algorithm of $O((m+n)\log(m+n))$ time. When $m=n$, this is $O(n\log n)$ time, improving the previous algorithm of $O(n^2\log n)$ time [28]. For the general case where both the upper and lower half-plane are present, using the method in [28] that reduces the problem to $O(n^2)$ instances of the lower-only case, the problem is now solvable in $O(n^2(m+n))\log(m+n))$ time. When $m=n$, this is $O(n^3\log n)$ time, improving the previous algorithm of $O(n^4\log n)$ time [28].

**An algorithm in the algebraic decision tree model.** In the algebraic decision tree model, where only comparisons are counted towards the time complexity, combining with a technique recently developed by Chan and Zheng [12], our method 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. The details are presented at the end of Section 3. In the following discussion, unless otherwise stated, all time complexities are measured in the standard real RAM model.

**Remark.** The results improve our original results in the conference version of this paper [26]. The high-level algorithm framework (and its correctness proof) is the same as before, but this version provides more efficient algorithm implementations.

## 2. Preliminaries

In this section, we introduce some concepts and notations that we will use in the rest of the paper.

We follow the notation defined in Section 1, e.g., $P$, $S$, $m$, $n$, $\ell$. Without loss of generality, we assume that $\ell$ is the $x$-axis and points of $P$ are all above or on $\ell$ while centers of disks of $S$ are all below or on $\ell$. Under this setting, for each disk $s\in S$, only its portion above $\ell$ matters for our coverage problem. Hence, unless otherwise stated, a disk $s$ only refers to its portion above $\ell$. As such, the boundary of $s$ consists of an *upper arc*, i.e., the boundary arc of the original disk above $\ell$, and a *lower segment*, i.e., the intersection of $s$ with $\ell$. Notice that $s$ has a single leftmost (resp., rightmost) point, which is the left (resp., right) endpoint of the lower segment of $s$.

We assume that single-intersection condition holds, that is, for any two disks, their boundaries intersect at most once in the half-plane above $\ell$.

We assume that each point of $P$ is covered by at least one disk 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 point of $P$ lies on the boundary of a disk and 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 all disks. Degenerated cases can be easily handled by standard perturbation techniques, e.g., [19].

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

For any disk $s$, we use $S_l(s)$ (resp., $S_r(s)$) to denote the set of disks $S$ whose leftmost points are left (resp., right) of that of $s$. As such, if the index of $s$ is $i$, then $S_l(s)=S[1,i-1]$ and $S_r(s)=S[i+1,m]$. If disk $s'\in S_l(s)$, then 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 $s$.

If $S'$ is a subset of $S$ that forms a coverage of $P$, then we call $S'$ a *feasible solution*. If $S'$ is a feasible solution of minimum size, then $S'$ is an *optimal solution*.

**The non-containment property.** Suppose a disk $s_i$ contains another disk $s_j$. Then $s_j$ is redundant for our problem, since any point covered by $s_j$ is also covered by $s_i$. These redundant disks can be easily identified and removed from $S$ mentioned in Section 1 and 2 in $O(m\log m)$ time (indeed, this is a 1D problem by observing that $s_i$ contains $s_j$ if and only if the lower segment of $s_i$ contains that of $s_j$). Hence, to solve our problem, we first remove such redundant disks and 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 our algorithm relies on.

**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 line-separable single-intersection case

In this section, we present our algorithm for the disk coverage problem in the line-separable single-intersection case. We follow the notation defined in Section 2.

For each disk $s_i \in S$, we define two indices $a(i)$ and $b(i)$ of points of $P$ (where $p_{a(i)}$ and $p_{b(i)}$ are not contained in $s_i$), which are critical to our algorithm.

**Definition 1.**

- Among all points of $P$ covered by the union of the disks of $S[1, i-1]$ but not covered by $s_i$, define $a(i)$ to be the largest index of these points; if no such point exists, then let $a(i) = 0$.
- Among all points of $P$ covered by the union of the disks of $S[i+1, m]$ but not covered by $s_i$, define $b(i)$ to be the smallest index of these points; if no such point exists, then let $b(i) = n+1$.

We say that a disk $s_i \in S$ is *prunable* if $a(i) \geq b(i)$.

We now describe our algorithm. Although the algorithm description looks simple, it is quite challenging to prove the correctness; we devote Section 3.1 to it. The implementation of the algorithm, which is also not trivial, is presented in Section 3.2.

**Algorithm description.** The algorithm has three main steps.

1. We first find all prunable disks of $S$. Let $S^*$ denote the subset of disks of $S$ that are not prunable. We will prove in Section 3.1 that $S^*$ contains an optimal solution for the coverage problem on $P$ and $S$. This means that it suffices to work on $S^*$ and $P$.
2. We then compute $a(i)$ and $b(i)$ for all disks $s_i \in S^*$. We will show in Section 3.2 that this step together with the above first step for computing $S^*$ can be done in $O((n+m)\log(n+m) + m\log m \log n)$ time.
3. We reduce the disk coverage problem on $S^*$ and $P$ to a 1D coverage problem as follows. For each point of $P$, we project it vertically onto $\ell$. Let $P'$ be the set of all projected points. For each disk $s_i \in S^*$, we create a line segment on $\ell$ whose left endpoint has $x$-coordinate equal to $x(p_{a(i)+1})$ and whose right endpoint has $x$-coordinate equal to $x(p_{b(i)-1})$ (if $a(i)+1 = b(i)$, then let the $x$-coordinate of the right endpoint be $x(p_{a(i)+1})$). Let $S'$ be the set of all the segments thus created.
   We solve the following 1D coverage problem: Find a minimum subset of the segments of $S'$ that together cover all points of $P'$. This problem can be easily solved in $O((|S'|+|P'|)\log(|S'|+|P'|))$ time [28],[1] which is $O((m+n)\log(m+n))$ since $|P'| = n$ and $|S'| \leq m$.
   Suppose $S'_1$ is any optimal solution to the above 1D coverage problem. We create a subset $S_1$ of $S^*$ as follows. For each segment of $S'_1$, suppose it is created from a disk $s_i \in S^*$; then we add $s_i$ to $S_1$. We will prove in Section 3.1 that $S_1$ is an optimal solution to the coverage problem for $S^*$ and $P$.

We summarize the result in the following theorem.

**Theorem 1.** *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 points of $P$ by a line, and the single-intersection condition is satisfied, the disk coverage problem for $P$ and $S$ is solvable in $O((n+m)\log(m+n) + m\log m\log n)$ time.*

**The unit-disk case.** In Section 4, we will reduce the time to $O((n+m)\log(n+m))$ for the unit-disk case. The algorithm follows the above except that we implement the first two steps in a more efficient way (i.e., in $O((n+m)\log(n+m))$ time) by utilizing the property that all disks have the same radius.

### 3.1. Algorithm correctness

We now prove the correctness of our algorithm. Lemma 2 justifies the correctness of the first main step. To prove Lemma 2, whose proof is lengthy and technical, we first prove the following Lemma 1 (which will also be useful in our algorithm implementation in Section 3.2 for finding all prunable disks). Recall the definition of $S_l(s)$ and $S_r(s)$ in Section 2.

**Lemma 1.** *A disk $s$ is prunable if and only if there exists a point in $P$ that is outside $s$ but is covered by both a disk in $S_l(s)$ and a disk in $S_r(s)$.*

**Proof.** Let $i$ be the index of $s$, i.e., $s = s_i$. Our goal is to show that $s_i$ is prunable if and only if there exists a point in $P$ that is outside $s_i$ but is covered by both a disk in $S[1, i-1]$ and a disk in $S[i+1, m]$.

[1] The algorithm in [28], which uses dynamic programming, is for the weighted case where each segment has a weight. Our problem is simpler since it is an unweighted case. We can use a simple greedy algorithm to solve it.

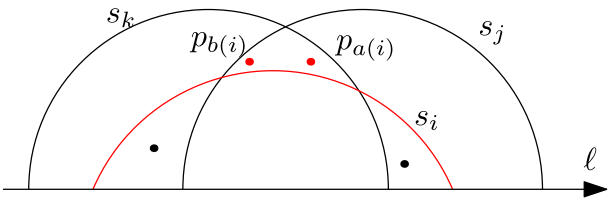



**Fig. 3.** Illustrating the case where $p_{b(i)} \in s_k$.

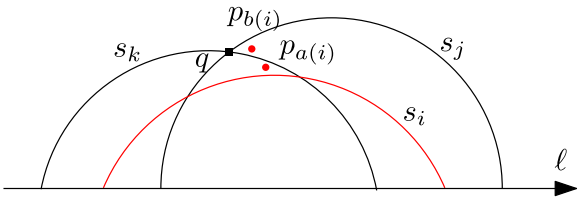



**Fig. 4.** Illustrating the case where $p_{b(i)} \notin s_k$.

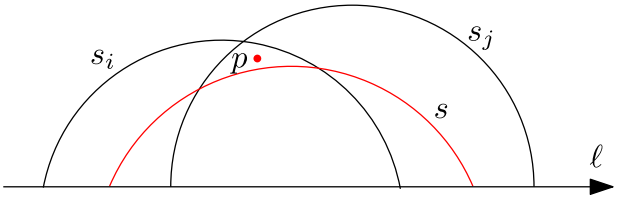



**Fig. 5.** Illustrating Observation 2.

**The "if" direction.** Suppose $P$ has a point $p_t$ that is outside $s_i$ but is covered by a disk $s_k$ and a disk $s_j$ with $k < i < j$. Our goal is to prove that $a(i) \geq b(i)$, meaning that $s_i$ is prunable by definition. Since $s_k$ covers $p_t$ and $k < i$, by definition we have $a(i) \geq t$. On the other hand, since $s_j$ covers $p_t$ and $i < j$, by definition we have $b(i) \leq t$. As such, we obtain $a(i) \geq b(i)$.

**The "only if" direction.** Suppose $s_i$ is a prunable disk. Our goal is to show that there exists a point $p^* \in P$ that is outside $s_i$ but covered by both a disk in $S[1, i-1]$ and a disk in $S[i+1, m]$. Since $s_i$ is a prunable disk, we have $a(i) \geq b(i)$, and further, there are a disk $s_k$ with $k < i$ that covers $p_{a(i)}$ and a disk $s_j$ with $j > i$ that covers $p_{b(i)}$. Depending on whether $s_k$ covers $p_{b(i)}$, there are two cases.

1. If $s_k$ covers $p_{b(i)}$ (see Fig. 3), then $p_{b(i)}$ is covered by both $s_k$ and $s_j$. Since $k < i$ and $j > i$, we can use $p_{b(i)}$ as our target point $p^*$.
2. If $s_k$ does not cover $p_{b(i)}$ (see Fig. 4), then since $s_k$ covers $p_{a(i)}$, we have $a(i) \neq b(i)$ and thus $a(i) > b(i)$. Since $k < j$, $s_k$ covers $p_{a(i)}$, and $s_j$ covers $p_{b(i)}$, due to the non-containment property of $S$, we have $x(l_k) < x(l_j) < x(p_{b(i)}) < x(p_{a(i)}) < x(r_k) < x(r_j)$, implying that the upper arcs of $s_k$ and $s_j$ must intersect, say, at point $q$ (see Fig. 4). Since $p_{b(i)}$ is inside $s_j$ but outside $s_k$, $x(q) < x(p_{b(i)})$ must hold. Hence, the region of $s_k$ to the right of $q$ must be inside $s_j$. Since $x(q) < x(p_{b(i)}) < x(p_{a(i)})$ and $p_{a(i)}$ is in $s_k$, $p_{a(i)}$ must be in $s_j$ as well. Therefore, $p_{a(i)}$ is in both $s_k$ and $s_j$. As such, we can use $p_{a(i)}$ as our target point $p^*$.

The lemma thus follows. □

The following observation, which follows immediately from the non-containment property of $S$, is needed in the proof of Lemma 2.

**Observation 2.** *For any disk $s$ and a point $p$ outside $s$, if $p$ is covered by both a disk $s_i \in S_l(s)$ and a disk $s_j$ in $S_r(s)$, then $s \subseteq s_i \cup s_j$ (see Fig. 5).*

**Lemma 2.** *$S^*$ contains an optimal solution to the coverage problem on $S$ and $P$.*

**Proof.** Let $S_{\text{opt}}$ be an optimal solution. Let $Q$ be the set of all prunable disks, i.e., $Q = S \setminus S^*$. If $S_{\text{opt}} \cap Q = \emptyset$, then $S_{\text{opt}} \subseteq S^*$ and thus the lemma trivially follows. In what follows, we assume that $|S_{\text{opt}} \cap Q| \geq 1$.

Pick an arbitrary disk from $S_{\text{opt}} \cap Q$, denoted $\hat{s}_1$. Below, we give a process that can find a disk $s^*$ from $S^*$ to replace $\hat{s}_1$ in $S_{\text{opt}}$ such that the new set $S^1_{\text{opt}} = \{s^*\} \cup S_{\text{opt}} \setminus \{\hat{s}_1\}$ still forms a coverage of $P$ (i.e., $S^1_{\text{opt}}$ is a feasible solution), implying

that $S^1_{\text{opt}}$ is also an optimal solution since $|S^1_{\text{opt}}| = |S_{\text{opt}}|$. As $s^* \in S^*$, we have $|S^1_{\text{opt}} \cap Q| = |S_{\text{opt}} \cap Q| - 1$. If $S^1_{\text{opt}} \cap Q$ is still nonempty, then we can repeat the above process for other points in $S^1_{\text{opt}} \cap Q$ until we obtain an optimal solution $S^*_{\text{opt}}$ with $S^*_{\text{opt}} \cap Q = \emptyset$, which will prove the lemma.

We now give a process to find a target disk $s^*$. The process involves induction. To help the reader understand it better, we first provide details of 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 when we describe the induction).

**The first iteration.** Let $S'_{\text{opt}} = S_{\text{opt}} \setminus \{\hat{s}_1\}$. Since $\hat{s}_1 \in Q$, by Lemma 1, $P$ has a point $\hat{p}_1$ outside $\hat{s}_1$ but is covered by a disk $\hat{s}^l_1 \in S_l(\hat{s}_1)$ and a disk $\hat{s}^r_1 \in S_r(\hat{s}_1)$. By Observation 2, $\hat{s}_1 \subseteq \hat{s}^l_1 \cup \hat{s}^r_1$. Since $\hat{p}_1$ is outside $\hat{s}_1$ and $S_{\text{opt}} = S'_{\text{opt}} \cup \{\hat{s}_1\}$ forms a coverage of $P$, $S'_{\text{opt}}$ must have a disk $s$ that covers $\hat{p}_1$. Clearly, $s$ is either in $S_l(\hat{s}_1)$ or in $S_r(\hat{s}_1)$. Without loss of generality, we assume that $s \in S_r(\hat{s}_1)$. Since $\hat{s}^r_1$ refers to an arbitrary disk of $S_r(\hat{s}_1)$ that covers $\hat{p}_1$ and $s$ is also a disk of $S_r(\hat{s}_1)$ that covers $\hat{p}_1$, for notational convenience, we let $\hat{s}^r_1$ refer to $s$. As such, $\hat{s}^r_1$ is in $S'_{\text{opt}}$.

Consider the disk $\hat{s}^l_1$. Since $\hat{s}_1 \subseteq \hat{s}^l_1 \cup \hat{s}^r_1$ and $\hat{s}^r_1$ is in $S'_{\text{opt}}$, it is not difficult to see that the area covered by the union of the disks of $S_{\text{opt}}$ is contained in the area covered by the union of the disks of $S'_{\text{opt}} \cup \{\hat{s}^l_1\}$ and thus $S'_{\text{opt}} \cup \{\hat{s}^l_1\}$ is a feasible solution. As such, if $\hat{s}^l_1 \notin Q$, then we can use $\hat{s}^l_1$ as our target disk $s^*$ and our process (for finding $s^*$) is done. In what follows, we assume $\hat{s}^l_1 \in Q$.

For any subset $S'$ of $S$, we define $\mathcal{R}(S')$ as the area covered by the union of the disks of $S'$, i.e., $\mathcal{R}(S') = \bigcup_{s \in S'} s$.

We let $\hat{s}_2 = \hat{s}^l_1$. Define $A_1 = \{\hat{s}^r_1\}$. According to the above discussion, we have $A_1 \subseteq S'_{\text{opt}}$, $\hat{s}_1 \subseteq \mathcal{R}(A_1) \cup \hat{s}_2$, and $S'_{\text{opt}} \cup \{\hat{s}_2\}$ is a feasible solution.

**The second iteration.** We are now entering the second iteration of our process. First notice that $\hat{s}_2$ cannot be $\hat{s}_1$ since $\hat{s}_2 = \hat{s}^l_1$, which cannot be $\hat{s}_1$. Our goal in this iteration is to find a *candidate disk* $s'$ to replace $\hat{s}_2$ so that $S'_{\text{opt}} \cup \{s'\}$ also forms a coverage of $P$. Consequently, if $s' \notin Q$, then we can use $s'$ as our target $s^*$; otherwise, we need to guarantee $s' \neq \hat{s}_1$ so that our process does not enter a dead loop. The discussion here is more involved than in the first iteration.

Since $\hat{s}_2 \in Q$, by Lemma 1, $P$ has a point $\hat{p}_2$ outside $\hat{s}_2$ but is covered by a disk $\hat{s}^l_2 \in S_l(\hat{s}_2)$ and a disk $\hat{s}^r_2 \in S_r(\hat{s}_2)$. By Observation 2, $\hat{s}_2 \subseteq \hat{s}^l_2 \cup \hat{s}^r_2$. Depending on whether $\hat{p}_2$ is in $\mathcal{R}(A_1)$, there are two cases.

- If $\hat{p}_2 \notin \mathcal{R}(A_1)$, then since $\hat{p}_2 \notin \hat{s}_2$ and $\hat{s}_1 \subseteq \mathcal{R}(A_1) \cup \hat{s}_2$, we obtain $\hat{p}_2 \notin \hat{s}_1$. We can now basically repeat our argument in the first iteration. Since $\hat{p}_2$ is outside $\hat{s}_2$ and $S'_{\text{opt}} \cup \{\hat{s}_2\}$ is a feasible solution, $S'_{\text{opt}}$ must have a disk $s$ that covers $\hat{p}_2$. Clearly, $s$ is either in $S_l(\hat{s}_2)$ or in $S_r(\hat{s}_2)$. Without loss of generality, we assume that $s \in S_r(\hat{s}_2)$. Since $\hat{s}^r_2$ refers to an arbitrary disk of $S_r(\hat{s}_2)$ that covers $\hat{p}_2$ and $s$ is also a disk of $S_r(\hat{s}_2)$ that covers $\hat{p}_2$, for notational convenience, we let $\hat{s}^r_2$ refer to $s$. As such, $\hat{s}^r_2$ is in $S'_{\text{opt}}$.
  We let $\hat{s}^l_2$ be our candidate disk, which satisfies our need as discussed above for $s'$. Indeed, since $S'_{\text{opt}} \cup \{\hat{s}_2\}$ is an optimal solution, $\hat{s}_2 \subseteq \hat{s}^l_2 \cup \hat{s}^r_2$, and $\hat{s}^r_2 \in S'_{\text{opt}}$, we obtain that $S'_{\text{opt}} \cup \{\hat{s}^l_2\}$ also forms a coverage of $P$. Furthermore, since $\hat{s}^l_2$ contains $\hat{p}_2$ while $\hat{s}_1$ does not, we know that $\hat{s}^l_2 \neq \hat{s}_1$. Therefore, if $\hat{s}^l_2 \notin Q$, then we can use $\hat{s}^l_2$ as our target $s^*$ and we are done with the process. Otherwise, we let $\hat{s}_3 = \hat{s}^l_2$ and then enter the third iteration argument. In this case, we let $A_2 = A_1 \cup \{\hat{s}^r_2\}$. According to our above discussion, we have $A_2 \subseteq S'_{\text{opt}}$, $\hat{s}_2 \subseteq \mathcal{R}(A_2) \cup \hat{s}_3$, and $\{\hat{s}_3\} \cup S'_{\text{opt}}$ is a feasible solution.
- If $\hat{p}_2 \in \mathcal{R}(A_1)$, then we let $\hat{s}^l_2$ be our candidate disk. We show in the following that it satisfies our need as discussed above for $s'$, i.e., $\{\hat{s}^l_2\} \cup S'_{\text{opt}}$ forms a coverage of $P$ and $\hat{s}^l_2 \neq \hat{s}_1$.
  Indeed, since $A_1 = \{\hat{s}^r_1\}$ and $\hat{p}_2 \in \mathcal{R}(A_1)$, $\hat{p}_2$ is inside $\hat{s}^r_1$. Since $\hat{s}^r_1$ is to the right of $\hat{s}_1$, and $\hat{s}_2$, which is $\hat{s}^l_1$, is to the left of $\hat{s}_1$, we obtain that $\hat{s}^r_1$ is to the right of $\hat{s}_2$, i.e., $\hat{s}^r_1 \in S_r(\hat{s}_2)$. Since $\hat{s}^l_2$ contains $\hat{p}_2$, $\hat{s}^l_2 \in S_l(\hat{s}_2)$, $\hat{s}^r_1$ contains $\hat{p}_2$, and $\hat{s}^r_1 \in S_r(\hat{s}_2)$, by Observation 2, we obtain that $\hat{s}_2 \subseteq \hat{s}^l_2 \cup \hat{s}^r_1$, i.e., $\hat{s}_2 \subseteq \hat{s}^l_2 \cup \mathcal{R}(A_1)$. Since $S'_{\text{opt}} \cup \{\hat{s}_2\}$ is a feasible solution and $A_1 \subseteq S'_{\text{opt}}$, it follows that $\{\hat{s}^l_2\} \cup S'_{\text{opt}}$ is also a feasible solution. On the other hand, since $\hat{s}^l_2$ is in $S_l(\hat{s}_2)$ while $\hat{s}_2$ (which is $\hat{s}^l_1$) is in $S_l(\hat{s}_1)$, we know that $\hat{s}^l_2$ is in $S_l(\hat{s}_1)$ and thus $\hat{s}^l_2 \neq \hat{s}_1$.
  As such, if $\hat{s}^l_2 \notin Q$, we can use $\hat{s}^l_2$ as our target $s^*$ and we are done with the process. Otherwise, we let $\hat{s}_3 = \hat{s}^l_2$ and continue on the third iteration. In this case, we let $A_2 = A_1$. According to our above discussion, we have $A_2 \subseteq S'_{\text{opt}}$, $\hat{s}_2 \subseteq \mathcal{R}(A_2) \cup \{\hat{s}_3\}$, and $\{\hat{s}_3\} \cup S'_{\text{opt}}$ is a feasible solution.

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 disk $\hat{s}_i \in Q$, $i \geq 2$. We make the following inductive hypothesis for $i$.

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

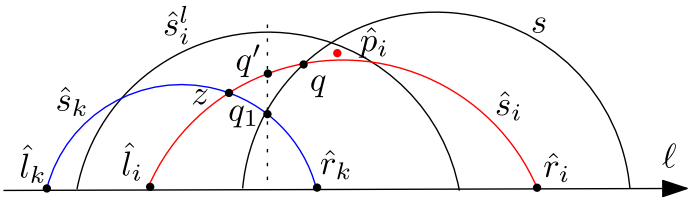



**Fig. 6.** Illustrating the proof of Lemma 2. Note that $\hat{s}_i^l$ is $\hat{s}_{i+1}$.

Our above discussion showed that the hypothesis holds for $i = 2$ and $i = 3$. Next we discuss the $i$-th iteration for any general $i$. Our goal is to find a candidate disk $\hat{s}_{i+1}$ so that $S'_{\text{opt}} \cup \{\hat{s}_{i+1}\}$ is a feasible solution and the inductive hypothesis holds for $i+1$.

Since $\hat{s}_i \in Q$, by Lemma 1, $P$ has a point $\hat{p}_i$ outside $\hat{s}_i$ but is covered by a disk $\hat{s}_i^l \in S_l(\hat{s}_i)$ and a disk $\hat{s}_i^r \in S_r(\hat{s}_i)$. By Observation 2, $\hat{s}_i \subseteq \hat{s}_i^l \cup \hat{s}_i^r$. Depending on whether $\hat{p}_i$ is in $\mathcal{R}(A_{i-1})$, there are two cases.

1. If $\hat{p}_i \notin \mathcal{R}(A_{i-1})$, then since $\hat{p}_i$ is outside $\hat{s}_i$ and $S'_{\text{opt}} \cup \{\hat{s}_i\}$ is a feasible solution, $S'_{\text{opt}}$ must have a disk $s$ that covers $\hat{p}_i$. Clearly, $s$ is either in $S_l(\hat{s}_i)$ or in $S_r(\hat{s}_i)$. Without loss of generality, we assume that $s \in S_r(\hat{s}_i)$. Since $\hat{s}_i^r$ refers to an arbitrary disk of $S_r(\hat{s}_i)$ that covers $\hat{p}_i$ and $s$ is also a disk of $S_r(\hat{s}_i)$ that covers $\hat{p}_i$, for notational convenience, we let $\hat{s}_i^r$ refer to $s$. As such, $\hat{s}_i^r$ is in $S'_{\text{opt}}$.
   We let $\hat{s}_{i+1}$ be $\hat{s}_i^l$ and define $A_i = A_{i-1} \cup \{\hat{s}_i^r\}$. We argue in the following that the inductive hypothesis holds.
   - Indeed, since $\{\hat{s}_i\} \cup S'_{\text{opt}}$ is a feasible solution, $\hat{s}_i \subseteq \hat{s}_i^l \cup \hat{s}_i^r$, $\hat{s}_i^r \in S'_{\text{opt}}$, and $\hat{s}_{i+1} = \hat{s}_i^l$, we obtain that $\{\hat{s}_{i+1}\} \cup S'_{\text{opt}}$ is a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_i = A_{i-1} \cup \{\hat{s}_i^r\}$, $A_{i-1} \subseteq S'_{\text{opt}}$ by the inductive hypothesis and $\hat{s}_i^r \in S'_{\text{opt}}$, we obtain $A_i \subseteq S'_{\text{opt}}$. Also, since $\hat{s}_i \subseteq \hat{s}_i^l \cup \hat{s}_i^r$, $\hat{s}_i^r \in A_i$, and $\hat{s}_{i+1} = \hat{s}_i^l$, we have $\hat{s}_i \subseteq \mathcal{R}(A_i) \cup \hat{s}_{i+1}$. This proves the second statement of the hypothesis.
   - For any disk $\hat{s}_k$ with $1 \le k \le i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{s}_k \neq \hat{s}_{i+1}$. To this end, since $\hat{p}_i \in \hat{s}_{i+1}$, it suffices to show that $\hat{p}_i \notin \hat{s}_k$. Indeed, by the inductive hypothesis, $\hat{s}_k \subseteq \mathcal{R}(A_k) \cup \hat{s}_{k+1}$ and $\hat{s}_{k+1} \subseteq \mathcal{R}(A_{k+1}) \cup \hat{s}_{k+2}$. Hence, $\hat{s}_k \subseteq \mathcal{R}(A_k) \cup \mathcal{R}(A_{k+1}) \cup \hat{s}_{k+2}$. As $\mathcal{R}(A_k) \subseteq \mathcal{R}(A_{k+1})$, we obtain $\hat{s}_k \subseteq \mathcal{R}(A_{k+1}) \cup \hat{s}_{k+2}$. Following the same argument, we can derive $\hat{s}_k \subseteq \mathcal{R}(A_{i-1}) \cup \hat{s}_i$. Now that $\hat{p}_i \notin \mathcal{R}(A_{i-1})$ and $\hat{p}_i \notin \hat{s}_i$, we obtain $\hat{p}_i \notin \hat{s}_k$.
2. If $\hat{p}_i \in \mathcal{R}(A_{i-1})$, then $\hat{p}_i$ is covered by a disk of $A_{i-1}$, say $s$. As $\hat{p}_i \notin \hat{s}_i$, $s \neq \hat{s}_i$ and thus $s$ is in $S_l(\hat{s}_i)$ or $S_r(\hat{s}_i)$. Without loss of generality, we assume that $s \in S_r(\hat{s}_i)$.
   We let $\hat{s}_{i+1}$ be $\hat{s}_i^l$ and define $A_i = A_{i-1}$. We show in the following that the inductive hypothesis holds.
   - Since $\hat{p}_i$ is in both $s$ and $\hat{s}_i^l$, $\hat{s}_i^l \in S_l(\hat{s}_i)$, and $s \in S_r(\hat{s}_i)$, by Observation 2, $\hat{s}_i \subseteq \hat{s}_i^l \cup s$. Furthermore, since $\{\hat{s}_i\} \cup S'_{\text{opt}}$ is a feasible solution, $s \in A_{i-1} \subseteq S'_{\text{opt}}$, and $\hat{s}_{i+1} = \hat{s}_i^l$, we obtain that $\{\hat{s}_{i+1}\} \cup S'_{\text{opt}}$ is also a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_{i-1} \subseteq S'_{\text{opt}}$ by inductive hypothesis and $A_i = A_{i-1}$, we have $A_i \subseteq S'_{\text{opt}}$. As discussed above, $\hat{s}_i \subseteq \hat{s}_i^l \cup s$. Since $s \in A_{i-1} = A_i$ and $\hat{s}_{i+1} = \hat{s}_i^l$, we obtain $\hat{s}_i \subseteq \mathcal{R}(A_i) \cup \hat{s}_{i+1}$. This proves the second statement of the hypothesis.
   - For any disk $\hat{s}_k$ with $1 \le k \le i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{s}_k \neq \hat{s}_{i+1}$. By hypothesis, we know that $\hat{s}_k \neq \hat{s}_i$, implying that $\hat{s}_k \in S_l(\hat{s}_i)$ or $\hat{s}_k \in S_r(\hat{s}_i)$. If $\hat{s}_k \in S_r(\hat{s}_i)$, since $\hat{s}_{i+1} = \hat{s}_i^l \in S_l(\hat{s}_i)$, it is obviously true that $\hat{s}_k \neq \hat{s}_{i+1}$. In the following, we assume that $\hat{s}_k \in S_l(\hat{s}_i)$ and we will prove that $\hat{s}_k$ does not contain $\hat{p}_i$, which implies that $\hat{s}_k \neq \hat{s}_{i+1}$ as $\hat{p}_i \in \hat{s}_{i+1}$.
     First of all, since $\hat{p}_i$ is covered by both $s \in S_r(\hat{s}_i)$ and $\hat{s}_{i+1} \in S_l(\hat{s}_i)$, it must hold that $x(\hat{l}_i) < x(\hat{p}_i) < x(\hat{r}_i)$, where $\hat{l}_i$ and $\hat{r}_i$ are the left and right endpoints of the lower segment of $\hat{s}_i$ (i.e., the segment $\hat{s}_i \cap \ell$), respectively (see Fig. 6). Hence, since $s \in S_r(\hat{s}_i)$ and $\hat{p}_i \in s$, the upper arcs of $s$ and $\hat{s}_i$ must cross each other, say, at a point $q$. As $\hat{p}_i$ is in $s$ but not in $\hat{s}_i$, we have $x(q) < x(\hat{p}_i)$.
     Recall that $\{\hat{s}_i\} \cup S'_{\text{opt}}$ is a feasible solution. Also, $S'_{\text{opt}}$ cannot be a feasible solution since that would contradict the fact that $S_{\text{opt}}$ is an optimal solution as $|S_{\text{opt}}| = |S'_{\text{opt}}| + 1$. This implies that $\hat{s}_i$ is not contained in $\mathcal{R}(S'_{\text{opt}})$. Since all disk centers are below $\ell$, at least one point, say $q'$, on the upper arc of $\hat{s}_i$ is not in $\mathcal{R}(S'_{\text{opt}})$. As $s \in S'_{\text{opt}}$, $q'$ is not in $s$. Therefore, $x(q') < x(q)$ must hold. As $x(q) < x(\hat{p}_i)$, we have $x(q') < x(\hat{p}_i)$ (see Fig. 6).
     Recall that our goal is to prove that $\hat{p}_i \notin \hat{s}_k$. Let $\hat{l}_k$ and $\hat{r}_k$ be the left and right endpoints of the lower segment of $\hat{s}_k$, respectively. If $x(\hat{r}_k) \le x(q')$, then since $x(q') < x(\hat{p}_i)$, it is obviously true that $\hat{s}_k$ does not contain $\hat{p}_i$. We therefore assume that $x(\hat{r}_k) > x(q')$ (see Fig. 6). Since $\hat{s}_k \in S_l(\hat{s}_i)$, we have $x(\hat{l}_k) < x(\hat{l}_i)$. As $x(\hat{l}_i) < x(q')$, we obtain $x(\hat{l}_k) < x(q')$. Since $x(\hat{l}_k) < x(q') < x(\hat{r}_k)$, the vertical line through $q'$ must intersect the upper arc of $\hat{s}_k$ at a point, say, $q_1$ (see Fig. 6). We claim $y(q_1) \le y(q')$. Indeed, since $q'$ is not inside $\mathcal{R}(S'_{\text{opt}})$, $q'$ is on the upper envelope of all disks of $S'_{\text{opt}} \cup \{\hat{s}_i\}$, denoted by $\mathcal{U}$. Recall that we have proved above (in the first case where $\hat{p}_i \in \mathcal{R}(A_{i-1})$) that $s_k \subseteq \mathcal{R}(S'_{\text{opt}}) \cup \hat{s}_i$. Therefore, the upper arc of $\hat{s}_k$ cannot be higher than $\mathcal{U}$. As $q' \in \mathcal{U}$, it follows that $y(q_1) \le y(q')$.

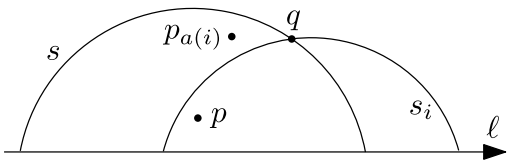



**Fig. 7.** Illustrating the proof of Lemma 4.

Since $x(\hat{l}_k) < x(\hat{l}_i) < x(q') < x(\hat{r}_k)$, due to the non-containment property, the upper arcs of $\hat{s}_k$ and $\hat{s}_i$ must cross each other at a single point, say $z$. Because $y(q_1) \leq y(q')$, we have $x(z) \leq x(q_1)$. As such, the region of $\hat{s}_k$ to the right of $q_1$ must be inside $\hat{s}_i$ (see Fig. 6). Recall that $x(q_1) = x(q') < x(\hat{p}_i)$. As $\hat{p}_i \notin \hat{s}_i$, $\hat{p}_i$ cannot be in $\hat{s}_k$.

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

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

This completes the proof of the lemma. □

It remains to prove the correctness of the third main step of our algorithm. For each disk $s_i \in S^*$, $a(i) < b(i)$ by definition; define $P(s_i) = \{p_j \mid a(i) < j < b(i)\}$.

**Lemma 3.** *All points of $P(s_i)$ are inside $s_i$.*

**Proof.** Assume to the contrary that a point $p_k \in P(s_i)$ is not in $s_i$. By definition, $a(i) < k < b(i)$. Recall that each point of $P$ is covered by a disk of $S$. Let $s$ be a disk of $S$ that covers $p_k$. Since $s \neq s_i$, $s$ is either in $S_l(s_i)$ or in $S_r(s_i)$. In the former case, by the definition of $a(i)$, $a(i) \geq k$, but this contradicts $a(i) < k$. In the latter case, by the definition of $b(i)$, $b(i) \leq k$; but this contradicts $k < b(i)$. □

The following lemma justifies the correctness of the third main step of our algorithm.

**Lemma 4.** *Suppose $S_{opt}$ is an optimal solution to the coverage problem on $S^*$ and $P$, and $s_i$ is a disk in $S_{opt}$. Then, any point of $P \setminus P(s_i)$ must be covered by a disk of $S_{opt} \setminus \{s_i\}$.*

**Proof.** Let $p$ be a point in $P \setminus P(s_i)$. If $p \notin s_i$, then since the disks of $S_{\text{opt}}$ form a coverage of $P$, there must be a disk of $S_{\text{opt}} \setminus \{s_i\}$ that covers $p$. In the following, we assume that $p \in s_i$. Since $p \notin P(s_i)$, by Lemma 3, either $x(p) \leq x(p_{a(i)})$ or $x(p) \geq x(p_{b(i)})$. In the following we only discuss the former case as the latter case is symmetric.

Since $S_{\text{opt}}$ is an optimal solution, $S_{\text{opt}}$ must have a disk $s$ that covers $p_{a(i)}$ (see Fig. 7). By definition, $p_{a(i)}$ is not in $s_i$. Hence, $s \neq s_i$ and thus $s$ is either in $S_l(s_i)$ or in $S_r(s_i)$. We claim that $s$ must be in $S_l(s_i)$. Indeed, assume to the contrary that $s \in S_r(s_i)$. Then, by the definition of $b(i)$, $b(i) \leq a(i)$ must hold, which contradicts $a(i) < b(i)$. Since $s \in S_l(s_i)$, we next prove that $s$ must cover $p$, which will prove the lemma.

Indeed, since $x(p) \leq x(p_{a(i)})$, $p$ is inside $s_i$, and $s \in S_l(s_i)$, due to the non-containment property of $S$, the upper arcs of $s_i$ and $s$ must intersect at a single point, say, $q$ (see Fig. 7). Furthermore, since $p_{a(i)}$ is in $s$ but not in $s_i$, $x(p_{a(i)}) \leq x(q)$ must hold. This implies that the region of $s_i$ left of $p_{a(i)}$ is inside $s$. As $p$ is in $s_i$ and $x(p) \leq x(p_{a(i)})$, $p$ must be inside $s$. □

In light of the preceding two lemmas, when considering the coverage of $s_i$, it suffices to consider only the points in $P(i)$. This establishes the correctness of the third main step of our algorithm.

### 3.2. Algorithm implementation

In this section, we describe the implementation of our algorithm. Recall that points of $P$ are indexed in ascending order of their $x$-coordinates as $p_1, p_2, \ldots, p_n$, and disks of $S$ are indexed in ascending order of their leftmost points as $s_1, s_2, \ldots, s_m$ (which is also the order of their rightmost points due to the non-containment property). Also recall that all points of $P$ are above the $x$-axis $\ell$ while the centers of all disks of $S$ are below $\ell$.

To implement the algorithm, based on our algorithm description, it remains to perform the following two tasks: (1) Compute the subset $S^* \subseteq S$ of disks that are not prunable; (2) for each disk $s_i \in S^*$, compute $a(i)$ and $b(i)$.

To achieve the above (1), we resort to Lemma 1. To this end, for each point $p \in P$, we define $\sigma_1(p)$ as the smallest index of the disk in $S$ that covers $p$, and $\sigma_2(p)$ as the largest index of the disk in $S$ that covers $p$. Note that both $\sigma_1(p)$ and $\sigma_2(p)$ are well defined since every point in $P$ is covered by at least one disk. We first compute $\sigma_1(p)$ and $\sigma_2(p)$ in the following lemma.

**Lemma 5.** *Computing $\sigma_1(p)$ and $\sigma_2(p)$ for all points $p \in P$ can be done in $O((n+m)\log m)$ time.*

**Proof.** We only discuss how to compute $\sigma_1(p)$ since the algorithm for $\sigma_2(p)$ is similar.

Let $T$ be a complete binary search tree whose leaves from left to right correspond to disks in their index order. Since $m = |S|$, the height of $T$ is $O(\log m)$. For each node $v \in T$, let $S_v$ denote the subset of disks of $S$ in the leaves of the subtree rooted at $v$. We use $\xi_v$ to denote the upper envelope of the $x$-axis $\ell$ and the upper arcs of all disks of $S_v$. Since all points of $P$ are above $\ell$, our algorithm is based on the observation that a point $p \in P$ is inside a disk of $S_v$ if and only if $p$ is below $\xi_v$. We construct $\xi_v$ for every node $v \in T$. Due to the single-intersection condition that the upper arcs of every two disks of $S$ intersect at most once, $\xi_v$ has at most $O(|S_v|)$ vertices. To see this, we can view the upper envelope of each upper arc of $S_v$ and $\ell$ as an extended arc. Every two such extended arcs still cross each other at most once and therefore their upper envelope has $O(|S_v|)$ vertices following the standard Davenport-Schinzel sequence argument [30] (see also [11, Lemma 3] for a similar problem). Notice that $\xi_v$ is exactly the upper envelope of these extended arcs and thus $\xi_v$ has $O(|S_v|)$ vertices. Note also that $\xi_v$ is $x$-monotone. In addition, given $\xi_u$ and $\xi_w$, where $u$ and $w$ are the two children of $v$, $\xi_v$ can be computed in $O(|S_v|)$ time by a straightforward line sweeping algorithm. As such, computing $\xi_v$ for all nodes $v \in T$ can be accomplished in time linear in $\sum_{v\in T}|S_v|$, which is $O(m\log m)$.

Consider a point $p \in P$. We compute $\sigma_1(p)$ 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 $p$ is below $\xi_u$; since $|S_v| \le m$, this can be done in $O(\log m)$ time by binary search on the sorted list of the vertices of $\xi_u$ by their $x$-coordinates. If $p$ is below $\xi_u$, then $p$ must be inside a disk of $S_u$; in this case, we proceed with $v = u$. Otherwise, we proceed with $v = w$. In this way, $\sigma_1(p)$ can be computed after searching a root-to-leaf path of $T$, which has $O(\log m)$ nodes as the height of $T$ is $O(\log m)$. Because we spend $O(\log m)$ time on each node, the total time for computing $\sigma_1(p)$ is $O(\log^2 m)$. The time can be improved to $O(\log m)$ using fractional cascading [13], as follows.

The $x$-coordinates of all vertices of the upper envelope $\xi_v$ of each node $v \in T$ partition the $x$-axis into a set $\mathcal{I}_v$ of intervals. To determine whether $p$ is below $\xi_v$, it suffices to find the interval of $\mathcal{I}_v$ that contains $x(p)$, the $x$-coordinate of $p$ (after the interval is known, whether $p$ is below $\xi_v$ can be determined in $O(1)$ time). We construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(m\log m)$ time [13] since the total number of such intervals is $O(m\log m)$.

With the fractional cascading data structure, we only need to do binary search on the set of the intervals stored at the root to determine the interval containing $x(p)$, which takes $O(\log m)$ time. After that, for each node $u$ during the algorithm described above, the interval of $\mathcal{I}_u$ containing $x(p)$ can be determined in $O(1)$ time [13]. As such, computing $\sigma_1(p)$ takes $O(\log m)$ time.

In summary, computing $\sigma_1(p)$ for all points $p \in P$ takes $O((n+m)\log m)$ time in total. □

We now describe our algorithm to compute $S^*$, or alternatively, find all prunable disks of $S$. By Lemma 1, we have the following observation.

**Observation 3.** *A disk $s_i \in S$ is prunable if and only if there exists a point $p \in P$ such that $p \notin s_i$ and $\sigma_1(p) \le i \le \sigma_2(p)$.*

**Proof.** Suppose that $s_i$ is prunable. Then, by Lemma 1, there exists a point $p \in P$ such that $p \notin s_i$ and $p$ is covered by both a disk in $S_l(s_i)$ and a disk in $S_r(s_i)$. By definition, we have $\sigma_1(p) < i < \sigma_2(p)$.

On the other hand, suppose that there exists a point $p \in P$ such that $p \notin s_i$ and $\sigma_1(p) \le i \le \sigma_2(p)$. Then, since $p \notin s_i$ and $\sigma_1(p) \le i$, we obtain $\sigma_1(p) \ne i$ and thus $\sigma_1(p) < i$. As such, $p$ is covered by a disk in $S_l(s_i)$. By a similar argument, $p$ is also covered by a disk in $S_r(s_i)$. By Lemma 1, $s_i$ is prunable. □

With Observation 3 at hand, the following lemma computes $S^*$.

**Lemma 6.** *All prunable disks of $S$ can be found in $O(n\log n + n\log m + m\log m\log n)$ time.*

**Proof.** Let $T$ be the standard segment tree [17, Section 10.3] on the indices $1, 2, \ldots, m$ by considering each index $i$ a point on the $x$-axis $\ell$ whose $x$-coordinate is $i$. The height of $T$ is $O(\log m)$. For each point $p \in P$, let $I_p$ denote the interval $[\sigma_1(p), \sigma_2(p)]$ of $\ell$. Following the definition of the standard segment tree [17, Section 10.3], we store $I_p$ in $O(\log m)$ nodes of $T$. For each node $v \in T$, let $P_v$ denote the subset of points $p$ of $P$ whose interval $I_p$ is stored at $v$. As such, $\sum_{v\in T}|P_v| = O(n\log m)$.

Consider a disk $s_i \in S$. $T$ has a leaf corresponding to the index $i$, called *leaf-$i$*. Let $\pi_i$ denote the path of $T$ from the root to leaf-$i$. Following the definition of the segment tree, we have the following observation: $\bigcup_{v\in\pi_i} P_v = \{p \mid p \in P, \sigma_1(p) \le i \le \sigma_2(p)\}$. By Observation 3, to determine whether $s_i$ is prunable, it suffices to determine whether there is a node $v \in \pi_i$ such that $P_v$ has a point $p \notin s_i$. Based on this observation, we perform the following processing on $T$. For each node $v \in T$, we construct a farthest Voronoi diagram for $P_v$, denoted by $FD_v$, and then build a point location data structure on $FD_v$ so that each point location query can be answered in $O(\log n)$ time [18,23]. Computing $FD_v$ can be done in $O(|P_v|\log|P_v|)$ time [29] and constructing the point location data structure takes $O(|P_v|)$ time [18,23]. Since $\sum_{v\in T}|P_v| = O(n\log m)$, the

total time for constructing the farthest Voronoi diagrams for all nodes of $T$ is $O(n \log m \log n)$ and the total time for building all point location data structures is $O(n \log m)$.

For each disk $s_i \in S$, for each node $v \in \pi_i$, using a point location query on $FD_v$, we find the farthest point $p$ of $P_v$ from the center of $s_i$ in $O(\log n)$ time. If $p \notin s_i$, then we assert that $s_i$ is prunable and halt the algorithm for $s_i$; otherwise, all points of $P_v$ are inside $s_i$ (and thus no point of $P_v$ can cause $s_i$ to be prunable) and we continue on other nodes of $\pi_i$. Since $\pi_i$ has $O(\log m)$ nodes and each point location query takes $O(\log n)$ time, it takes $O(\log m \log n)$ time to determine whether $s_i$ is prunable. Therefore, the total time for doing this for all disks $s_i \in S$ is $O(m \log m \log n)$.

As such, we can find all prunable disks in a total of $O(m \log m \log n + n \log m \log n + n \log m)$ time. Note that the factor $O(n \log n \log m)$ is due to the construction of all the farthest Voronoi diagrams $FD_v$ for all nodes $v \in T$. We can further reduce the time to $O(n \log n + n \log m)$, as follows.

First, $FD_v$ is determined only by the points of $P_v$ that are vertices of the convex hull $H_v$ of $P_v$ [29]. Furthermore, once $H_v$ is available, $FD_v$ can be computed in $O(|H_v|)$ time [2]. On the other hand, $H_v$ can be obtained in $O(|P_v|)$ time if the points of $P_v$ are sorted by their $x$-coordinates. To have sorted lists for $P_v$ for all nodes $v \in T$, we do the following. At the start of the algorithm, we sort all points of $P$ by their $x$-coordinates in $O(n \log n)$ time. Then, for each point $p$ of $P$ following this sorted order, we find the nodes $v$ of $T$ where the interval $I_p$ should be stored and add $p$ to $P_v$, which takes $O(\log m)$ time [17, Section 7.4]. In this way, after all points of $P$ are processed as above, $P_v$ for every node $v \in T$ is automatically sorted. As such, the total time for constructing all farthest Voronoi diagrams $FD_v$ of all nodes $v \in T$ is $O(n \log n + n \log m)$. Therefore, the total time of the overall algorithm is $O(n \log n + n \log m + m \log m \log n)$. □

The remaining task is to compute $a(i)$ and $b(i)$ for all disks $s_i \in S^*$. In what follows, we only discuss how to compute $a(i)$ since the algorithm for $b(i)$ is analogous. Our algorithm relies on the following observation, whose proof is based on the fact that none of the disks of $S^*$ is prunable.

**Observation 4.** *For each disk $s_i \in S^*$, $a(i)$ is the largest index among the points $p \in P$ with $\sigma_2(p) < i$.*

**Proof.** First, consider a point $p_j \in P$ with $\sigma_2(p_j) < i$. Since $\sigma_2(p_j) < i$, we know that $p_j \notin s_i$ and $p_j$ is covered by a disk in $S[1, i-1]$. By the definition of $a(i)$, we have $j \le a(i)$.

On the other hand, let $j = a(i)$. By definition, $p_j \notin s_i$ and $p_j$ is covered by a disk in $S[1, i-1]$. By the definition of $\sigma_1(p_j)$, $\sigma_1(p_j) < i$ holds. We claim $\sigma_2(p_j) < i$. Indeed, assume to the contrary that $\sigma_2(p_j) \ge i$. Then, we have $\sigma_1(p_j) < i \le \sigma_2(p_j)$. Since $p_j \notin s_i$, by Observation 3, $s_i$ is prunable, which contradicts the fact that none of the disks of $S^*$ is prunable. As such, we obtain $\sigma_2(p_j) < i$.

The above discussions combined lead to the observation. □

In light of Observation 4, we have the following lemma.

**Lemma 7.** *$a(i)$ for all disks $s_i \in S^*$ can be computed in $O(n \log n + m)$ time.*

**Proof.** Recall that $p_1, p_2, \ldots, p_n$ are points of $P$ sorted in ascending order by $x$-coordinate. For each point $p \in P$, let $j(p)$ denote its index, referred to as the *$x$-sorted index*. We sort all the points $p$ of $P$ in ascending order by their values $\sigma_2(p)$ as $p^1, p^2, \ldots, p^n$. We process these points in this order. Each point $p^k$ is processed as follows. Our algorithm maintains $j^{k-1}$, the largest $x$-sorted index among all points $p^1, p^2, \ldots, p^{k-1}$. We first set $j^k$ to be the larger one of $j^{k-1}$ and $j(p^k)$. Then, for each integer $i$ with $\sigma_2(p^k) < i \le \sigma_2(p^{k+1})$ (if $k = n$, then we consider each $i$ with $\sigma_2(p^k) < i \le m$), if $s_i \in S^*$, then we set $a(i) = j^k$, whose correctness follows from Observation 4. After all points of $P$ are processed as above, $a(i)$ for all disks $s_i \in S^*$ are computed. The total time is $O(n \log n + m)$. □

Combining Lemmas 5, 6, and 7, the total runtime of the algorithm is $O((n+m)\log(n+m) + m \log m \log n)$. This proves Theorem 1.

**An algebraic decision tree algorithm.** In the algebraic decision tree model, where the time complexity is measured only by the number of comparisons, we can solve the problem in $O((n+m)\log(n+m))$ time, i.e., using $O((n+m)\log(n+m))$ comparisons. For this, observe that the overall algorithm excluding Lemma 6 runs in $O((n+m)\log(n+m))$ time. Hence, it suffices to show that Lemma 6 can be solved using $O((n+m)\log(n+m))$ comparisons. To this end, notice that the factor $O(m \log m \log n)$ in the algorithm of Lemma 6 is due to the point location queries on the farthest Voronoi diagrams $FD_v$. There are a total of $O(m \log m)$ queries. The total combinatorial complexity of the diagrams $FD_v$ of all nodes $v \in T$ is $O(n \log m)$. To solve these point location queries, we can use a technique recently developed by Chan and Zheng [12]. Specifically, we can simply apply [12, Theorem 7.2] to solve all our point location queries using $O((n+m)\log(n+m))$ comparisons (indeed, following the notation in [12, Theorem 7.2], we have $t = O(m)$, $L = O(n \log m)$, $M = O(m \log m)$, 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 $O((n+m)\log(n+m))$).

## 4. The unit-disk case

In this section, we show that the runtime of the algorithm can be reduced to $O((n+m)\log(n+m))$ for the unit-disk case. For this, observe that except for Lemma 6, the algorithm in Section 3 runs in $O((n+m)\log(n+m))$ time. Hence, it suffices to show that Lemma 6 can be implemented in $O((n+m)\log(n+m))$ time for the unit-disk case. To this end, we have the following lemma.

**Lemma 8.** *For the unit-disk case, all prunable disks of $S$ can be found in $O((n+m)\log(n+m))$ time.*

**Proof.** We follow the notation in Lemma 6. The algorithmic scheme is similar to Lemma 6. The major change is that, instead of constructing the farthest Voronoi diagrams for the nodes of $T$, here we perform different processing by exploring the property that all disks of $S$ have the same radius.

Consider a disk $s_i \in S$. To determine whether $s_i$ is prunable, recall that it suffices to decide for each node $v \in \pi_i$ whether $P_v$ has a point outside $s_i$. We explore the property that all disks of $S$ have the same radius (a disk of that radius is called a *unit disk*). For each point $p \in P$, let $D_p$ denote a unit disk centered at $p$. Define $\mathcal{D}_v = \{D_p \mid p \in P_v\}$. Let $c_i$ denote the center of $s_i$. Observe that $P_v$ has a point outside $s_i$ if and only if $c_i$ is outside a disk of $\mathcal{D}_v$. Recall that all points of $P$ are above the $x$-axis $\ell$ while the centers of all disks of $S$ are below $\ell$. Let $C_v$ denote the common intersection of all disks of $\mathcal{D}_v$ below $\ell$. Observe that $c_i$ is outside a disk of $\mathcal{D}_v$ if and only if $c_i$ is outside $C_v$.

We perform the following processing on $T$. For each node $v \in T$, we construct $C_v$. Since all disks of $\mathcal{D}_v$ have the same radius and their centers are all above $\ell$, the boundaries of every two disks intersect at most once in the halfplane below $\ell$. Using this property, $C_v$ can be computed in linear time by adapting Graham's scan after $P_v$ is sorted by $x$-coordinate (see [11, Lemma 3] for a similar problem). Computing the sorted lists for $P_v$ for all nodes $v \in T$ can be done in $O(n\log n + n\log m)$ time, as described in the proof of Lemma 6. As such, constructing $C_v$ takes additional $O(|P_v|)$ time and the total time for constructing $C_v$ of all nodes $v \in T$ is $O(n\log m)$ since $\sum_{v\in T}|P_v| = O(n\log m)$.

For each disk $s_i \in S$, our task is to determine whether $c_i \in C_v$ for each node $v \in \pi_i$. Note that the boundary of $C_v$ consists of a line segment on $\ell$ bounding $C_v$ from above and an $x$-monotone curve bounding $C_v$ from below. The projections of the vertices of $C_v$ onto the $x$-axis $\ell$ partition $\ell$ into a set $\mathcal{I}_v$ of $O(|P_v|)$ intervals. To determine whether $c_i \in C_v$, it suffices to find the interval of $I_v$ containing $x(c_i)$, the $x$-coordinate of $c_i$, after which whether $c_i \in C_v$ can be decided in $O(1)$ time. Finding the interval of $I_v$ containing $x(c_i)$ can be done in $O(\log n)$ time by binary search. If we do this for all nodes $v \in \pi_i$, the total time to determine whether $s_i$ is prunable would be $O(\log m \log n)$. We can improve the runtime to $O(\log m + \log n)$ using fractional cascading [13] in a way similar to the proof of Lemma 5. More specifically, we construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(n\log m)$ time since the total number of such intervals is linear in $\sum_{v\in T}|P_v|$, which is $O(n\log m)$. With the fractional cascading data structure, 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 $O(\log(n\log m))$ time. After that, following the path $\pi_i$ in a top-down manner, the interval of $\mathcal{I}_v$ containing $x(c_i)$ for each node $v \in \pi_i$ can be determined in $O(1)$ time [13]. As such, whether the disk $s_i$ is prunable can be determined in $O(\log n + \log m)$ time.

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

Combining Lemmas 5, 7, and 8 leads to the following result for the unit-disk case.

**Theorem 2.** *Given a set $P$ of $n$ points and a set $S$ of $m$ unit disks in the plane such that the centers of the disks are separated from the points of $P$ by a line, the disk coverage problem for $P$ and $S$ is solvable in $O((n+m)\log(n+m))$ time.*

### Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

### Data availability

No data was used for the research described in the article.

### Acknowledgement

The authors wish to express their gratitude to two anonymous reviewers whose comments helped to improve the presentation of this paper.

### References

[1] Pankaj K. Agarwal, Jiangwei Pan, Near-linear algorithms for geometric hitting sets and set covers, Discrete Comput. Geom. 63 (2020) 460–482, https://doi.org/10.1007/s00454-019-00099-6.

[2] Alok Aggarwal, Leonidas J. Guibas, James B. Saxe, Peter W. Shor, A linear-time algorithm for computing the Voronoi diagram of a convex polygon, Discrete Comput. Geom. 4 (1989) 591–604, https://doi.org/10.1007/BF02187749.
[3] Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S.B. Mitchell, Kim Whittlesey, Minimum-cost coverage of point sets by disks, in: Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG), 2006, pp. 449–458.
[4] Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, 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), 2006, pp. 3–14.
[5] Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, Geometric clustering to minimize the sum of cluster sizes, in: Proceedings of the 13th European Symposium on Algorithms (ESA), 2005, pp. 460–471.
[6] Ahmad Biniaz, Prosenjit Bose, Paz Carmi, Anil Maheshwari, J. Ian Munro, Michiel Smid, Faster algorithms for some optimization problems on collinear points, in: Proceedings of the 34th International Symposium on Computational Geometry (SoCG), 2018, pp. 8:1–8:14.
[7] Norbert Bus, Nabil H. Mustafa, Saurabh Ray, Practical and efficient algorithms for the geometric hitting set problem, Discrete Appl. Math. 240 (2018) 25–32, https://doi.org/10.1016/j.dam.2017.12.018.
[8] Paz Carmi, Matthew J. Katz, Nissan Lev-Tov, Covering points by unit disks of fixed location, in: Proceedings of the International Symposium on Algorithms and Computation (ISAAC), 2007, pp. 644–655.
[9] Timothy M. Chan, Elyot Grant, Exact algorithms and APX-hardness results for geometric packing and covering problems, Comput. Geom. Theory Appl. 47 (2014) 112–124, https://doi.org/10.1016/j.comgeo.2012.04.001.
[10] Timothy M. Chan, Qizheng He, Faster approximation algorithms for geometric set cover, in: Proceedings of 36th International Symposium on Computational Geometry (SoCG), 2020, pp. 27:1–27:14.
[11] Timothy M. Chan, 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), 2016, pp. 24:1–24:13.
[12] Timothy M. Chan, Da Wei Zheng, Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees, ACM Trans. Algorithms (2023), https://doi.org/10.1145/3591357.
[13] Bernard Chazelle, Leonidas J. Guibas, Fractional cascading: I. A data structuring technique, Algorithmica 1 (1986) 133–162, https://doi.org/10.1007/BF01840440.
[14] Francisco Claude, Gautam K. Das, Reza Dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, Alejandro Salinger, An improved line-separable algorithm for discrete unit disk cover, Discrete Math. Algorithms Appl. 2 (2010) 77–88, https://doi.org/10.1142/S1793830910000486.
[15] Gruia Călinescu, Ion I. Măndoiu, Peng-Jun Wan, Alexander Z. Zelikovsky, Selecting forwarding neighbors in wireless ad hoc networks, Mob. Netw. Appl. 9 (2004) 101–111, https://doi.org/10.1023/B:MONE.0000013622.63511.57.
[16] Gautam K. Das, Sandip Das, Subhas C. Nandy, Homogeneous 2-hop broadcast in 2D, Comput. Geom. Theory Appl. 43 (2010) 182–190, https://doi.org/10.1016/j.comgeo.2009.06.005.
[17] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry — Algorithms and Applications, 3rd edition, Springer-Verlag, Berlin, 2008.
[18] Herbert Edelsbrunner, Leonidas J. Guibas, J. Stolfi, Optimal point location in a monotone subdivision, SIAM J. Comput. 15 (2) (1986) 317–340, https://doi.org/10.1137/0215023.
[19] Herbert Edelsbrunner, Ernst P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Trans. Graph. 9 (1990) 66–104, https://doi.org/10.1145/77635.77639.
[20] Tomás Feder, Daniel H. Greene, Optimal algorithms for approximate clustering, in: Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), 1988, pp. 434–444.
[21] Shashidhara K. Ganjugunte, Geometric hitting sets and their variants, PhD thesis, Duke University, 2011, https://dukespace.lib.duke.edu/server/api/core/bitstreams/0d37dabc-42a5-4bc8-b4e9-e69b263a10ca/content.
[22] Sariel Har-Peled, Mira Lee, Weighted geometric set cover problems revisited, J. Comput. Geom. 3 (2012) 65–85, https://doi.org/10.20382/jocg.v3i1a4.
[23] David G. Kirkpatrick, Optimal search in planar subdivisions, SIAM J. Comput. 12 (1) (1983) 28–35, https://doi.org/10.1137/0212002.
[24] Lev-Tov Nissan, David Peleg, Polynomial time approximation schemes for base station coverage with minimum total radii, Comput. Netw. 47 (2005) 489–501, https://doi.org/10.1016/j.comnet.2004.08.012.
[25] Jian Li, Yifei Jin, A PTAS for the weighted unit disk cover problem, in: Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP), 2015, pp. 898–909.
[26] Gang Liu, Haitao Wang, On the line-separable unit-disk coverage and related problems, in: Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC), 2023, pp. 51:1–51:14.
[27] Nabil H. Mustafa, Saurabh Ray, Improved results on geometric hitting set problems, Discrete Comput. Geom. 44 (2010) 883–895, https://doi.org/10.1007/s00454-010-9285-9.
[28] Logan Pedersen, Haitao Wang, Algorithms for the line-constrained disk coverage and related problems, Comput. Geom. Theory Appl. 105–106 (2022) 101883, https://doi.org/10.1016/j.comgeo.2022.101883, 1–18.
[29] Michael I. Shamos, Dan Hoey, Closest-point problems, in: Proceedings of the 16th Annual Symposium on Foundations of Computer Science (FOCS), 1975, pp. 151–162.
[30] Micha Sharir, Pankaj K. Agarwal, Davenport-Schinzel Sequences and Their Geometric Applications, Cambridge University Press, New York, 1996, p. 372.