On the Line-Separable Unit-Disk Coverage 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 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 ℓ. We present an $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ 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 ℓ while points of P can be anywhere in the plane. Our algorithm runs in $O(m\sqrt{n} + (n+m)\log(n+m))$ time, which improves the previously best result of $O(nm\log(m+n))$ time. In addition, our results lead to an algorithm of $n^{10/3}2^{O(\log^* n)}$ 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 $n^{4/3}2^{O(\log^* n)}$ time while the previously best algorithm takes $O(n^2\log n)$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases disk coverage, line-separable, unit-disk, line-constrained, half-planes

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.51

Related Version *Full Version*: `http://arxiv.org/abs/2309.03162`

Funding This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356.

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 [15, 20]. Polynomial time approximation algorithms have been proposed for the problem and many of its variants, e.g., [1, 6, 8, 9, 16, 19].

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 [3]. If all disks of S contain the same point, polynomial time algorithms also exist [12, 13]; in particular, applying the result in [8] (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 [3, 7, 11], where disk centers are separated from the points by a given line ℓ. A polynomial time 4-approximation algorithm is given in [7]. Ambühl et al. [3] derived an exact algorithm of $O(m^2n)$ time. An improved $O(nm + n\log n)$ time algorithm is presented in [11] and another algorithm in [21] runs in $O(n\log n + m^2\log n)$ in the worst case.

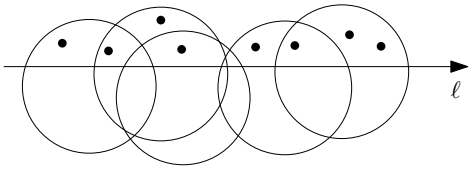

Figure 1 Illustrating the line-separable unit-disk case.

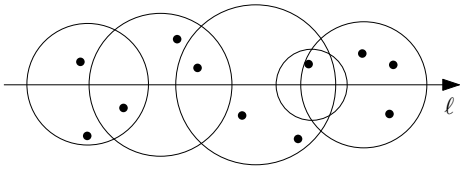

Figure 2 Illustrating the line-constrained case (all disks are centred on ℓ).

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 [21] 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 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, as well as the L_∞ and L_1 cases, where the disks are squares and diamonds, respectively [21]. The 1D problem where disks become segments on a line and points are on the same line is also solvable in $O((m+n)\log(m+n))$ [21]. Other types of line-constrained coverage problems have also been studied in the literature, e.g., [2, 4, 5, 18].

A related problem is when disks of S are half-planes. For the weighted case, Chan and Grant [8] 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 [21] 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 [17] solved the problem in $O(n^5)$ time. Pedersen and Wang [21] 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 ℓ is the x-axis and all disk centers are below or on ℓ while all points of P are above or on ℓ. 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 ℓ. 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 $O(m\sqrt{n}+(n+m)\log(n+m))$ 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 property 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, namely, 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, for which we utilize the cuttings [10].

The line-constrained problem. Observe that the line-constrained problem where all disks of S are centered on a line ℓ 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 ℓ, we could replace p by its symmetric point with respect to ℓ; in this way, we can obtain a set of points that are all above ℓ. It is not difficult to see that an optimal solution using this new set of points is also an optimal solution for P. Further, since disks are centered on ℓ, although their radii may not be equal, boundaries of any two disks intersect at most once above ℓ. 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 $O(m\sqrt{n}+(n+m)\log(n+m))$ time; this improves the previous algorithm in [21], 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 $m^{2/3}n^{2/3}2^{O(\log^*(m+n))}+O((m+n)\log(m+n))$ in Section 4. This improves the $O(nm+n\log n)$ time algorithm in [11] as well as the $O(n\log n+m^2\log n)$ time one in [21]. The main idea of the improvement (over the algorithm in Section 3) is to explore the duality of certain subproblems in the algorithm (i.e., consider the corresponding problems on the centers of all unit disks of S and the unit disks centered at the points of P). We derive new algorithms for these dual subproblems and then combine them with the algorithms in Section 3 using recursion (the number of recursions is $O(\log^*(n+m))$ and this is why there is a factor $2^{O(\log^*(m+n))}$ in the time complexity).

The half-plane coverage problem. As in [21], our techniques also solve the half-plane coverage problem. Specifically, for the lower-only case, let ℓ 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 ℓ, then the problem becomes an instance of the line-separable unit-disk coverage problem. Therefore, applying our result leads to an $m^{2/3}n^{2/3}2^{O(\log^*(m+n))}+O((m+n)\log(m+n))$ time algorithm. When $m=n$, this is $n^{4/3}2^{O(\log^* n)}$ time, improving the previous algorithm of $O(n^2\log n)$ time [21]. For the general case where both the upper and lower half-plane are present, using the method in [21] that reduces the problem to $O(n^2)$ instances of the lower-only case, the problem is now solvable in $m^{2/3}n^{8/3}2^{\log^*(m+n)}+O(n^2(m+n)\log(m+n))$ time. When $m=n$, this is $n^{10/3}2^{O(\log^* n)}$ time, improving the previous algorithm of $O(n^4\log n)$ time [21].

2 Preliminaries

This section introduces 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, ℓ. Without loss of generality, we assume that ℓ is the x-axis and points of P are all above or on ℓ while centers of disks of S are all below or on ℓ. Under this setting, for each disk $s\in S$, only its portion above ℓ matters for our coverage problem. Hence, unless otherwise stated, a disk s only refers to its portion above 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. 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 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 techniques of perturbation, e.g., [14].

For any point p in the plane, we denote its x- and y-coordinates by $x(p)$ and $y(p)$, respectively. We sort all the 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 to the 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.

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

If S' is a subset of S that form 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. Those redundant disks can be easily identified and removed from S 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, for solving 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 $x(r_i) < x(r_j)$.*

Cuttings. One algorithmic tool we use is the cuttings [10]. Let H denote the set of the upper arcs of all disks of S. Note that $|H| = m$.

For a parameter r with $1 \leq r \leq m$, a $(1/r)$-*cutting* Ξ of size $O(r^2)$ for H is a collection of $O(r^2)$ constant-complexity cells whose union covers the plane such that for any cell σ, $|H_\sigma| \leq m/r$ holds, where H_σ is the subset of arcs of H that intersect the interior of σ (H_σ is often called the *conflict list* in the literature). In our algorithm descriptions, we often use S_σ, defined as the subset of disks whose upper arcs are in H_σ.

Our algorithm actually uses *hierarchical cuttings* [10]. A cutting Ξ' *c-refines* a cutting Ξ if each cell of Ξ' is contained in a single cell of Ξ and every cell of Ξ contains at most c cells of Ξ'. Let Ξ_0 denote the cutting whose single cell is the entire plane. We define cuttings $\{\Xi_0, \Xi_1, ..., \Xi_k\}$, in which each Ξ_i, $1 \leq i \leq k$, is a $(1/\rho^i)$-cutting of size $O(\rho^{2i})$ that c-refines Ξ_{i-1}, for two constants ρ and c. By setting $k = \lceil \log_\rho r \rceil$, the last cutting Ξ_k is a $(1/r)$-cutting. The sequence $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ of cuttings is called a *hierarchical* $(1/r)$-*cutting* of H. For a cell σ' of Ξ_{i-1}, $1 \leq i \leq k$, that fully contains cell σ of Ξ_i, we say that σ' is the *parent* of σ and σ is a *child* of σ'. Thus the hierarchical $(1/r)$-cutting can be viewed as a tree structure with Ξ_0 as the root. We often use Ξ to denote the set of all cells in all cuttings Ξ_i, $0 \leq i \leq k$.

A hierarchical $(1/r)$-cutting of H can be computed in $O(mr)$ time, e.g., by the algorithm in [22], which adapts Chazelle's algorithm [10] for hyperplanes. The algorithm also produces the conflict lists H_σ (and thus S_σ) for all cells $\sigma \in \Xi$, implying that the total size of these

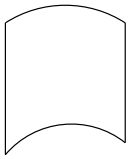

Figure 3 Illustrating a pseudo-trapezoid.

conflict lists is bounded by $O(mr)$. In particular, each cell of the cutting produced by the algorithm of [22] is a (possibly unbounded) *pseudo-trapezoid* that typically has two vertical line segments as left and right sides, a sub-arc of an arc of H as a top side (resp., bottom side) (see Fig. 3).

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 2.**

- *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 now describe our algorithm. Although the algorithm description looks simple, it is quite challenging to prove the correctness. Due to the space limit, the correctness proof is in the full version of the paper. The algorithm implementation, which is also not trivial, is presented in Section 3.1.

Algorithm description. The algorithm has three main steps.

1. We first compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We will show in Section 3.1 that this can be done in $O(m\sqrt{n} + (n+m)\log(n+m))$ time using cuttings.
2. For each disk s_i, if $a(i) \geq b(i)$, we say that s_i a *prunable disk*. Let S^* denote the subset of disks of S that are not prunable. We prove in the full version of this paper 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.
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 ℓ. Let P' be the set of all projected points. For each disk $s_i \in S^*$, we create a line segment on ℓ 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 segments thus created.

We solve the following 1D coverage problem: Find a minimum subset of segments of S' that together cover all points of P'. This problem can be easily solved in $O((|S'| + |P'|)\log(|S'| + |P'|))$ time [21],[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 prove in the full version of the paper that S_1 is an optimal solution to the coverage problem for S^* and P.

We summarize the result in the following theorem.

▶ **Theorem 3.** *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 $O(m\sqrt{n} + (n+m)\log(n+m))$ time.*

The unit-disk case. In Section 4, we will reduce the time to $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ for the unit-disk case. The algorithm is exactly the same as above, except that we compute $a(i)$'s and $b(i)$'s in a more efficient way by utilizing the property that all disks have the same radius.

3.1 Algorithm implementation

In this section, we show that the first main step of the algorithm can be implemented in $O(m\sqrt{n} + (n+m)\log(n+m))$ time. The goal is to compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We only discuss how to compute $a(i)$ since computing $b(i)$ can be done analogously. To this end, we start with the following definition.

▶ **Definition 4.** *For each point $p \in P$, define $\gamma(p)$ as the smallest index k such that the disk s_k covers p.*

One reason we introduce $\gamma(p)$ is due to the following observation.

▶ **Observation 5.** *For any disk $s_i \in S$ and any point $p \in P$ that is outside s_i, there is a disk in $S_l(s_i)$ covering p if and only if $\gamma(p) < i$.*

Our algorithm for computing $a(i)$ relies on $\gamma(p)$ for all $p \in P$. Therefore, we first present an algorithm in the following lemma to compute $\gamma(p)$.

▶ **Lemma 6.** *There is an algorithm that can compute $\gamma(p)$ for all $p \in P$ in $O(m\sqrt{n} + (m+n)\log(m+n))$ time.*

Proof. Let H be the set of the upper arcs of all disks. As discussed in Section 2, we compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10,22], for a parameter $r \in [1, m]$ to be determined later. We follow the notation about cutting as in Section 2, e.g., Ξ, H_σ, S_σ, etc. Recall that Ξ denotes the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$. As discussed in Section 2, the cutting algorithm [10, 22] also computes the conflict lists H_σ (and thus S_σ) for all cells $\sigma \in \Xi$. Also, $\sum_{\sigma\in\Xi} |S_\sigma| = O(mr)$.

[1] The algorithm in [21], 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.

For each i with $1 \leq i \leq k$, for each cell $\sigma \in \Xi_i$, let $S(\sigma)$ be the set of disks that contain σ but do not contain σ', where σ' is the parent cell of σ (which is in Ξ_{i-1}). Note that Ξ_0 consists of a single cell σ^* that is the entire plane and thus we simply let $S(\sigma^*) = \emptyset$ as no disk contains the entire plane.

We can compute $S(\sigma)$ of all cells $\sigma \in \Xi$ in $O(mr)$ time as follows. For each i with $1 \leq i \leq k$, for each cell $\sigma' \in \Xi_{i-1}$, recall that $S_{\sigma'}$ is available from the cutting algorithm. For each disk s of $S_{\sigma'}$, for each child cell σ of σ', we check whether s contains σ; if yes, we add s to $S(\sigma)$. As such, since the total size of S_σ of all cells σ of Ξ is $O(mr)$ and each cell has $O(1)$ children, the total time for computing $S(\sigma)$ for all cells $\sigma \in \Xi$ is $O(mr)$.

For each cell σ, by slightly abusing the notation, we define $\gamma(\sigma)$ as the smallest index of the disks in $S(\sigma)$. After $S(\sigma)$'s are computed, the indices $\gamma(\sigma)$ for all cells $\sigma \in \Xi$ can be computed in additional $O(mr)$ time.

Next, we run the following *point location step* for each point $p \in P$ to compute $\gamma(p)$. Initially, we set $\gamma(p) = m + 1$. Starting from the only cell of Ξ_0, we locate the cell σ_i that contains p in each cutting Ξ_i. This takes $O(\log r)$ time as each cell contains $O(1)$ children and $k = O(\log r)$. For each such cell σ_i, we update $\gamma(p) = \min\{\gamma(p), \gamma(\sigma_i)\}$. As such, the point location step on p takes $O(\log r)$ time. The total time for all points of P is $O(n \log r)$.

In addition, we do the following processing for the cell σ_k of the last cutting Ξ_k that contains each $p \in P$. For each disk $s_j \in S_{\sigma_k}$, we check whether s_j contains p. If yes, we update $\gamma(p) = \min\{\gamma(p), j\}$. After that, $\gamma(p)$ is correctly computed. As $|S_{\sigma_k}| \leq m/r$, this additional step for each point p takes $O(m/r)$ time. Therefore, the total time of this step for all points of P is $O(nm/r)$.

In summary, computing $\gamma(p)$ for all $p \in P$ takes $O(mr + n \log r + nm/r)$ time. Setting $r = \min\{\sqrt{n}, m\}$ leads to the lemma. ◀

The following lemma finally computes $a(i)$.

▶ **Lemma 7.** *Computing $a(i)$ for all disks $s_i \in S$ can be done in $O(m\sqrt{n}+(m+n)\log(m+n))$ time.*

Proof. We first compute $\gamma(p)$ for all $p \in P$ by Lemma 6.

Let H be the set of the upper arcs of all disks. As discussed in Section 2, we compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10, 22], for a parameter $r \in [1, m]$ to be determined later. We follow the notation about cutting as in Section 2, e.g., Ξ, H_σ, S_σ, etc. Recall that Ξ denotes the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$.

For each cell $\sigma \in \Xi$, let $P(\sigma)$ denote the set of points of P inside σ, i.e., $P(\sigma) = P \cap \sigma$. We can compute $P(\sigma)$ for all cells $\sigma \in \Xi$ in $O(n \log r)$ time by the point location step as discussed in Lemma 6. Note that the total size of $P(\sigma)$ for all cells $\sigma \in \Xi$ is also $O(n \log r)$. In addition, if we invoke the point location step for points of P following their index order, then points in each $P(\sigma)$ can be sorted in their index order and the time is still $O(n \log r)$.

We need to perform a *pruning procedure* for $P(\sigma)$ of each cell $\sigma \in \Xi$. Before we describe it, we first explain the motivation. Our algorithm for computing $a(i)$ needs to solve the following subproblem. Given a disk s_i and a cell $\sigma \in \Xi$ such that σ does not intersect s_i, the problem is to compute $a_\sigma(i)$, which is defined as the largest index k of a point p_k of $P(\sigma)$ with $\gamma(p_k) < i$ (if no such k exists, then $a_\sigma(i) = 0$). In light of Observation 5, $a_\sigma(i)$ is the largest index k of a point p_k of $P(\sigma)$ such that $S_l(s_i)$ has a disk covering p_k. To solve the subproblem, consider two points p_k and p_j in $P(\sigma)$ with $k < j$. A *key observation* is that if $\gamma(p_k) \geq \gamma(p_j)$, then $a_\sigma(i) \neq k$ holds for any such disk s_i with $s_i \cap \sigma = \emptyset$, and thus p_k can simply be ignored. Indeed, assume to the contrary that $a_\sigma(i) = k$. Then, we have

$\gamma(p_k) < i$. Hence, $\gamma(p_j) < i$. By definition, we can obtain $a_\sigma(i) \geq j > k$, which contradicts with $a_\sigma(i) = k$. In light of the key observation, to facilitate computing $a_\sigma(i)$ for all such disks s_i, we first perform the following pruning procedure.

The algorithm maintains a stack A of points of $P(\sigma)$. Initially, $A = \emptyset$. We process the points of $P(\sigma)$ in their index order (recall that they are already sorted). Suppose we are processing a point $p \in P(\sigma)$. Let p' be the point at the top of the stack. If $A = \emptyset$ or if $\gamma(p') < \gamma(p)$, then we push p onto A. Otherwise, we pop p' out of A (we say that p' is pruned) and repeat the above. After all points of $P(\sigma)$ are processed, let $P'(\sigma)$ denote the set of points in the stack. Due to the pruning, points of $P'(\sigma)$ are sorted by both their indices and their $\gamma(\cdot)$ values. Clearly, the pruning procedure runs in $O(|P(\sigma)|)$ time.

We use $P'(\sigma)$ in the following way. Recall that we wish to compute $a_\sigma(i)$. Let k be the largest index of $p_k \in P'(\sigma)$ such that $\gamma(p_k) < i$. Then, the above key observation and our pruning procedure guarantee that $a_\sigma(i) = k$. Hence, we could compute $a_\sigma(i)$ by a binary search on $P'(\sigma)$ using i, the index of the disk. However, doing binary search for each disk would make the total runtime of the algorithm have one more logarithmic factor. To improve it, we use the following strategy. For each cell σ, suppose $S'(\sigma)$ is a set of disks s_i (with $s_i \cap \sigma = \emptyset$) that need to compute $a_\sigma(i)$ with respect to σ (the exact definition of $S'(\sigma)$ will be given later). Then, we search $P'(\sigma)$ with disks of $S'(\sigma)$ altogether, by using a procedure similar to that for merging two sorted lists in merge-sort. In this way, the total time is linear in $|P'(\sigma)| + |S'(\sigma)|$ (in contrast, the time would be $O(|S'(\sigma)| \cdot \log |P'(\sigma)|)$ if we do binary search for each disk of $S'(\sigma)$).

We are now ready to describe our overall algorithm for computing $a(i)$. The above has computed $P(\sigma)$ for all cells $\sigma \in \Xi$, whose total size is $O(n \log r)$. We run the pruning procedure on $P(\sigma)$ for every cell $\sigma \in \Xi$ to compute $P'(\sigma)$; this takes $O(n \log r)$ time in total as $\sum_{\sigma \in \Xi} |P(\sigma)| = O(n \log r)$.

For each cell $\sigma \in \Xi$, we define $S'(\sigma)$ as the subset of disks that do not intersect σ but whose upper arcs intersect the parent cell of σ. We can compute $S'(\sigma)$ for all cells $\sigma \in \Xi$ in $O(mr)$ time as follows. Initially we set $S'(\sigma) = \emptyset$. Then, for each $0 \leq i \leq k-1$, for each cell $\sigma' \in \Xi_i$, for each disk $s \in S_{\sigma'}$, for each child σ of σ', if s does not intersect σ, then we add s to $S'(\sigma)$. As each cell σ' has $O(1)$ children and $\sum_{\sigma \in \Xi} |S_\sigma| = O(mr)$, it takes $O(mr)$ time to compute $S'(\sigma)$ for all cells $\sigma \in \Xi$. This also implies $\sum_{\sigma \in \Xi} |S'(\sigma)| = O(mr)$.

Now that we have $P'(\sigma)$ and $S'(\sigma)$ available for all cells $\sigma \in \Xi$, we compute $a(i)$ for all disks s_i as follows. Initially, we set $a(i) = 0$. Then, for each cell σ, we perform a search with $P'(\sigma)$ and $S'(\sigma)$ to compute $a_\sigma(i)$ for all disks $s_i \in S'(\sigma)$ using the procedure discussed above, which takes $O(|P'(\sigma)| + |S'(\sigma)|)$ time. Then, for each disk $s_i \in S'(\sigma)$, we update $a(i) = \max\{a(i), a_\sigma(i)\}$. Since $\sum_{\sigma \in \Xi} |P'(\sigma)| = O(n \log r)$ and $\sum_{\sigma \in \Xi} |S(\sigma)| = O(mr)$, processing all cells σ of Ξ as above takes $O(mr + n \log r)$ time in total.

Finally, for each cell σ of the last cutting Ξ_k, we perform the following additional processing: For each disk $s_i \in S_\sigma$, for each point $p_j \in P(\sigma)$, if p_j is outside s_i and $\gamma(p_j) < i$, then we update $a(i) = \max\{a(i), j\}$. After that, the values $a(i)$ for all disks $s_i \in S$ are correctly computed. Since $|S_\sigma| \leq m/r$ for each cell $\sigma \in \Xi_k$, we spend $O(m/r)$ time on each point $p \in P(\sigma)$. As $\sum_{\sigma \in \Xi_k} |P(\sigma)| = n$, the total time of the additional processing as above for all cells $\sigma \in \Xi_k$ is $O(nm/r)$.

In summary, we can compute $a(i)$ for all disks $s_i \in S$ in $O(mr + n \log r + nm/r)$ time in total. Setting $r = \min\{\sqrt{n}, m\}$ leads to the lemma. ◀

4 The unit-disk case

In this section, we consider the unit-disk case where all disks of S have the same radius. As remarked right after Theorem 3, our algorithm is the same as described in Section 3, except that we are able to compute $a(i)$'s and $b(i)$'s more efficiently in $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n+m)\log(n+m))$ time, by exploring the property that all disks have the same radius. In the following, we only discuss how to compute $a(i)$'s because the case for $b(i)$'s is similar.

For each point $p_i \in P$, define $\tilde{p}_i$ as the unit disk centered at p_i, and we call $\tilde{p}_i$ the *dual disk* of p_i. For each disk s_i, let $\tilde{s}_i$ denote the center of s_i, and we call $\tilde{s}_i$ the *dual point* of s_i. Let $\tilde{P}$ denote the set of all dual disks and $\tilde{S}$ the set of all dual points. Since all disks of S have the same radius, we have the following easy observation.

▶ **Observation 8.** *A disk $s_i \in S$ contains a point $p_j \in P$ if and only if the dual point $\tilde{s}_i$ is contained in the dual disk $\tilde{p}_j$.*

Our new algorithm for computing $a(i)$'s for the unit-disk case relies on exploring the "duality" in Observation 8. Recall in Section 3.1 that the algorithm for computing $a(i)$'s involves two steps: (1) compute $\gamma(p)$'s for all points $p \in P$ (i.e., Lemma 6); (2) compute $a(i)$'s for all $s_i \in P$ (i.e., Lemma 7). We have new algorithms for both steps in the following two subsections, respectively.

4.1 Computing $\gamma(p)$'s

We first introduce the following definition $\tilde{\gamma}(\cdot)$, which is "dual" to $\gamma(\cdot)$.

▶ **Definition 9.** *For each dual disk $\tilde{p} \in \tilde{P}$, define $\tilde{\gamma}(\tilde{p})$ as the smallest index k such that $\tilde{p}$ contains the dual point $\tilde{s}_k$.*

The following observation follows immediately from Observation 8.

▶ **Observation 10.** *For each point $p_i \in P$, $\gamma(p_i) = \tilde{\gamma}(\tilde{p}_i)$.*

Observation 10 implies that computing $\gamma(p_i)$ for all points $p_i \in S$ is equivalent to computing $\tilde{\gamma}(\tilde{p}_i)$ for all dual disks $\tilde{p}_i \in \tilde{P}$. To compute them, we will present two recursive algorithms and then combine them to obtain our final algorithm. The first algorithm computes $\gamma(p_i)$'s using P and S while the second one computes $\tilde{\gamma}(\tilde{p}_i)$ using $\tilde{P}$ and $\tilde{S}$. The combined algorithm will run the two algorithms alternatively using recursion.

The first algorithm. This algorithm follows the same framework as that for Lemma 6, but when processing the cells σ in the last cutting Ξ_k, instead of brute-force, we form subproblems and solve them recursively. We follow the notation from Lemma 6.

Let H be the set of the upper arcs of all disks of S. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for H in $O(mr)$ time [10, 22], for a parameter $r \in [1, m]$ to be determined later. Let Ξ denote the set of all cells of all cuttings σ_i, $i = 0, 1, \ldots, k$. As in Lemma 6, we compute $S(\sigma)$ and $\gamma(\sigma)$ for all cells $\sigma \in \Xi$, which takes $O(mr)$ time.

Next, we run the point location step for each point $p \in P$ as in Lemma 6. Initially, we set $\gamma(p) = 0$. Starting from Ξ_0, we locate the cell σ_i that contains p in each cutting Ξ_i. For each σ_i, we update $\gamma(p) = \min\{\gamma(p), \gamma(\sigma_i)\}$. This point location step can also compute $P(\sigma)$ for all cells $\sigma \in \Xi$, where $P(\sigma)$ denotes the set of points of P in σ. The total time for the point locations for all points $p \in P$ is $O(n \log r)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, if $|P(\sigma)| > n/r^2$, we partition $P(\sigma)$ into subsets of sizes between $n/(2r^2)$ and n/r^2, called *standard subsets* (if $|P(\sigma)| \leq n/r^2$, then $P(\sigma)$ itself is a standard subset). As Ξ_k has $O(r^2)$ cells and $\sum_{\sigma\in\Xi_k} |P(\sigma)| = n$, the total number of standard subsets for all cells $\sigma \in \Xi_k$ is $O(r^2)$. Recall that S_σ is the subset of disks whose upper arcs intersect σ. For each standard subset $P_1(\sigma)$ of $P(\sigma)$, we form a subproblem on $(P_1(\sigma), S_\sigma)$: Compute $\gamma_\sigma(p)$ for all points $p \in P_1(\sigma)$ with respect to disks of S_σ, where $\gamma_\sigma(p)$ is defined to be the smallest index of the disks of S_σ covering p. After the subproblem is solved, we update $\gamma(p) = \min\{\gamma(p), \gamma_\sigma(p)\}$ for each point $p \in P_1(\sigma)$. This will compute $\gamma(p)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|P_1(\sigma)| \leq n/r^2$ and $|S_\sigma| \leq m/r$.

If we use $T(n, m)$ to denote the runtime of the entire algorithm on the original problem (P, S), then we obtain the following recurrence relation:

$$T(n, m) = O(mr + n \log r) + O(r^2) \cdot T(n/r^2, m/r). \tag{1}$$

The second algorithm. The second algorithm computes $\tilde{\gamma}(\tilde{p})$ for all dual disks $\tilde{p} \in \tilde{P}$.

Recall that all dual disks have their centers above ℓ. Therefore, each dual disk has a "lower arc" below ℓ. Let $\tilde{H}$ denote the set of the lower arcs of all dual disks. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for $\tilde{H}$ in $O(nr)$ time [10, 22], for a parameter $r \in [1, n]$ to be determined later. We use Ξ to denote the set of all cells of all cuttings Ξ_i, $i = 0, 1, \ldots, k$. For each cell $\sigma \in \Xi$, let $\tilde{P}_\sigma$ denote the set of dual disks whose lower arcs intersect σ.

For each cell $\sigma \in \Xi$, let $\tilde{S}(\sigma)$ denote the subset of dual points of $\tilde{S}$ inside σ. For each cell $\sigma \in \Xi$, by slightly abusing the notation, let $\tilde{\gamma}(\sigma)$ denote the minimum index of all points of $\tilde{S}(\sigma)$. We can compute $\tilde{S}(\sigma)$ as well as $\tilde{\gamma}(\sigma)$ for all cells $\sigma \in \Xi$ in $O(m \log r)$ time by point locations as in the first algorithm.

We now compute $\tilde{\gamma}(\tilde{p})$'s. Initially, we set each $\tilde{\gamma}(\tilde{p}) = m + 1$. For each $1 \leq i \leq k$, for each cell $\sigma' \in \Xi_{i-1}$, for each dual disk $\tilde{p} \in \tilde{P}_{\sigma'}$, for each child $\sigma \in \Xi_i$ of σ', if $\tilde{p}$ contains σ, then we update $\tilde{\gamma}(\tilde{p}) = \min\{\tilde{\gamma}(\tilde{p}), \tilde{\gamma}(\sigma)\}$. Since $\sum_{\sigma\in\Xi} |\tilde{P}_\sigma| = O(nr)$ and each cell has $O(1)$ children, the total time of this procedure is $O(nr)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, as in the first algorithm, if $|\tilde{S}(\sigma)| > m/r^2$, we partition $\tilde{S}(\sigma)$ into *standard subsets* of sizes between $m/(2r^2)$ and m/r^2. The total number of standard subsets is $O(r^2)$. For each standard subset $\tilde{S}_1(\sigma)$ of $\tilde{S}(\sigma)$, we form a subproblem on $(\tilde{P}_\sigma, \tilde{S}_1(\sigma))$: Compute $\tilde{\gamma}_\sigma(\tilde{p})$ for all dual disks $\tilde{p} \in \tilde{P}(\sigma)$ with respect to the dual points of $\tilde{S}_1(\sigma)$, where $\tilde{\gamma}_\sigma(\tilde{p})$ is the smallest index of the dual points of $\tilde{S}_1(\sigma)$ contained in $\tilde{p}$. After the subproblem is solved, we update $\tilde{\gamma}(\tilde{p}) = \min\{\tilde{\gamma}(\tilde{p}), \tilde{\gamma}_\sigma(\tilde{p})\}$ for each $\tilde{p} \in \tilde{P}(\sigma)$. This will compute $\tilde{\gamma}(\tilde{p})$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|\tilde{S}_1(\sigma)| \leq m/r^2$ and $|\tilde{P}_\sigma| \leq n/r$.

Recall that $T(n, m)$ refers to our problem for computing $\gamma(p)$'s on (P, S), which is equivalent to computing $\tilde{\gamma}(\tilde{p})$'s on $(\tilde{P}, \tilde{S})$ by Observation 8. Hence, we can also obtain the following recurrence relation using the second algorithm:

$$T(n, m) = O(nr + m \log r) + O(r^2) \cdot T(n/r, m/r^2). \tag{2}$$

Combining the two algorithms. We now combine the two algorithms to compute $\gamma(p)$'s for all $p \in P$.

We first discuss the *symmetric case* where $m = n$ (if $m \neq n$, it is the *asymmetric case*). If we apply (1) and then (2) using the same r, we can obtain the following recurrence

$$T(n, n) = O(nr \log r) + O(r^4) \cdot T(n/r^3, n/r^3).$$

Setting $r = n^{1/3}/\log n$ leads to the following

$$T(n,n) = O(n^{4/3}) + O((n/\log^3 n)^{4/3}) \cdot T(\log^3 n, \log^3 n).$$

The recurrence solves to $T(n,n) = n^{4/3}2^{O(\log^* n)}$.

We next tackle the asymmetric case, by using the above symmetric case result. Depending on whether $m \geq n$, there are two cases.

1. If $m \geq n$, depending on whether $m < n^2$, there are two subcases.
   a. If $m < n^2$, then set $r = m/n$ so that $n/r = m/r^2$. Applying (2) with $r = m/n$ and solving each subproblem $T(n/r, m/r^2)$ using the symmetric case result give us $T(n,m) = m^{2/3}n^{2/3}2^{O(\log^* m)}$.
   b. If $m \geq n^2$, then applying (2) with $r = n$ gives us $T(n,m) = O(n^2 + m\log n) + O(n^2) \cdot T(1, m/n^2)$. Clearly, we have $T(1, m/n^2) = O(m/n^2)$. Hence, we obtain $T(m,n) = O(m\log n)$ since $m \geq n^2$.

   Hence in the case where $m \geq n$ we have $T(n,m) = O(m\log n) + m^{2/3}n^{2/3}2^{O(\log^* m)}$.
2. If $m < n$, the analysis is similar (using (1) instead) and we can obtain $T(n,m) = O(n\log m) + m^{2/3}n^{2/3}2^{O(\log^* n)}$.

In summary, computing $\gamma(p)$'s for all points $p \in P$ can be done in $O((n+m)\log(m+n)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.

4.2 Computing $a(i)$'s

With $\gamma(p)$'s computed above, we describe our algorithm for computing $a(i)$'s for all disks $s_i \in S$. As in Section 4.1, we first introduce the following definition, which is "dual" to $a(i)$.

▶ **Definition 11.** *For each dual point $\tilde{s}_i \in \tilde{S}$, define $\tilde{a}(i)$ as the largest index k of the dual disk $\tilde{p}_k \in \tilde{P}$ such that $\tilde{p}_k$ contains a dual point $\tilde{s}_j$ with $j < i$ but does not contain $\tilde{s}_i$.*

Based on Observation 8, we have the following lemma.

▶ **Lemma 12.** *For each $1 \leq i \leq m$, $a(i) = \tilde{a}(i)$.*

Proof. Consider the point $p_k \in P$ with $k = a(i)$. By definition, p_k is outside s_i and S has a disk s_j that covers p_k with $j < i$. Then, by Observation 8, the dual disk $\tilde{p}_k$ contains the dual point $\tilde{s}_j$ but does not contain the dual point $\tilde{s}_i$. By definition, it must hold that $\tilde{a}(i) \geq k = a(i)$.

Analogously, we can prove that $a(i) \geq \tilde{a}(i)$. ◀

Lemma 12 implies that computing $a(i)$ for all disks $s_i \in S$ is equivalent to computing $\tilde{a}(i)$ for all dual points $\tilde{s}_i \in \tilde{S}$. To compute them, as in Section 4.1, we will also present two recursive algorithms and then combine them. The first algorithm computes $a(i)$'s using P and S while the second one computes $\tilde{a}(i)$'s using $\tilde{P}$ and $\tilde{S}$. The combined algorithm will run the two algorithms alternatively using recursion. In what follows, we assume that $\gamma(p)$ for all points $p \in P$ and $\tilde{\gamma}(\tilde{p})$ for all $\tilde{p} \in \tilde{P}$ have been computed.

The first algorithm. The first algorithm follows the framework of Lemma 7 but uses recursion when we process the last cutting Ξ_k. Here we only discuss how to perform additional preprocessing for the cells of the last cutting Ξ_k; the rest of the algorithm is the same as before, which takes $O(mr + n\log r)$ time in total. We follow the notation in the proof of Lemma 7.

For each cell $\sigma \in \Xi_k$, if $|P(\sigma)| > n/r^2$, we partition $P(\sigma)$ into *standard subsets* of sizes between $n/(2r^2)$ and n/r^2. Recall that S_σ is the subset of disks of S whose upper arcs intersect σ. For each standard subset $P_1(\sigma)$ of $P(\sigma)$, we form a subproblem on $(P_1(\sigma), S_\sigma)$: Compute $a_\sigma(i)$ for all disks $s_i \in S(\sigma)$ with respect to points of $P_1(\sigma)$, where $a_\sigma(i)$ is the largest index k of a point $p_k \in P_1(\sigma)$ that is outside s_i but is covered by a disk s_j with $j < i$. After the subproblem is solved, we update $a(i) = \max\{a(i), a_\sigma(i)\}$ for each disk $s_i \in S(\sigma)$. This will compute $a(i)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|P_1(\sigma)| \leq n/r^2$ and $|S_\sigma| \leq m/r$.

If we use $T(n, m)$ to denote the runtime of the entire algorithm on the original problem (P, S), then we obtain the following recurrence relation:

$$T(n, m) = O(mr + n \log r) + O(r^2) \cdot T(n/r^2, m/r). \tag{3}$$

The second algorithm. The second algorithm computes $\tilde{a}(i)$ for all dual points $\tilde{s}_i \in S$.

Recall that all dual disks have their centers above ℓ. Therefore, each dual disk has a "lower arc" below ℓ. Let $\tilde{H}$ denote the set of lower arcs of all dual disks. We compute a hierarchical $(1/r)$-cutting $\Xi_0, \ldots, \Xi_k$ for $\tilde{H}$ in $O(nr)$ time [10, 22], for a parameter $r \in [1, n]$ to be determined later. We use Ξ to denote the set of cells of all cuttings Ξ_i, $i = 0, 1, \ldots, k$. For each cell $\sigma \in \Xi$, let $\tilde{P}_\sigma$ denote the set of dual disks whose lower arcs intersect σ.

For each cell $\sigma \in \Xi$, let $\tilde{S}(\sigma)$ be the set of dual points of $\tilde{S}$ inside σ. We can compute $\tilde{S}(\sigma)$ for all cells of Ξ in $O(m \log r)$ time using point locations as discussed before. Also, $\sum_{\sigma \in \Xi} |\tilde{S}(\sigma)| = O(m \log r)$. In addition, points in each $\tilde{S}(\sigma)$ can be sorted in their index order if we invoke the point location step on dual points of $\tilde{S}$ in their index order; the total time is still $O(m \log r)$.

For each cell $\sigma \in \Xi$, we define $\tilde{P}(\sigma)$ in the same way as $S'(\sigma)$ in the proof of Lemma 7. Specifically, $\tilde{P}(\sigma)$ is the subset of dual disks of $\tilde{P}$ that do not intersect σ but whose lower arcs intersect the parent cell of σ. As in Lemma 7, $\tilde{P}(\sigma)$ for all $\sigma \in \Xi$ can be computed in $O(nr)$ time and $\sum_{\sigma \in \Xi} |\tilde{P}(\sigma)| = O(nr)$.

Now consider the following problem on $\tilde{S}(\sigma)$ and $\tilde{P}(\sigma)$. For each dual point $\tilde{s}_i \in \tilde{S}(\sigma)$, we want to compute $\tilde{a}_\sigma(i)$, which is the largest index k of a dual disk $\tilde{p}_k \in \tilde{P}(\sigma)$ that contains a dual point $\tilde{s}_j$ with $j < i$. After solving the problem, we update $\tilde{a}(i) = \max\{\tilde{a}(i), \tilde{a}_\sigma(i)\}$. To solve the problem, first notice that $\tilde{p}_k$ contains a dual point $\tilde{s}_j$ with $j < i$ if and only if $\tilde{\gamma}(\tilde{p}_k) < i$. Then, consider two dual disks $\tilde{p}_k$ and $\tilde{p}_j$ in $\tilde{P}(\sigma)$ with $k < j$. A *key observation* is that if $\tilde{\gamma}(k) \geq \tilde{\gamma}(j)$, then $\tilde{a}_\sigma(i) \geq j$ holds for any dual point $\tilde{s}_i \in \tilde{S}(\sigma)$ (and thus $\tilde{p}_k$ can be ignored; this echoes the key observation in Lemma 7).

Using the key observation, as in the proof of Lemma 7, we run a pruning procedure on $\tilde{P}(\sigma)$ to obtain a subset $\tilde{P}'(\sigma)$ of dual disks that are sorted both by their indices and their $\tilde{\gamma}(\cdot)$ values. The pruning procedure takes $O(|\tilde{P}(\sigma)|)$ time if dual disks of $\tilde{P}(\sigma)$ are already sorted by their indices. We can produce the sorted lists of $\tilde{P}(\sigma)$ for all cells $\sigma \in \Xi$ in $O(nr)$ time as follows. First, for each dual disk $\tilde{p}_i$, we create a list L_i that contains all cells $\sigma \in \Xi$ such that $\tilde{p}_i$ is in $\tilde{P}_\sigma$. This can be done in $O(nr)$ time by traversing the conflict lists of all cells. Second, we process the lists $L_1, L_2, \ldots, L_n$ in this order. For each list L_i, for each cell $\sigma \in L_i$, we add $\tilde{p}_i$ to the rear of a list $L(\sigma)$ for σ (initially, $L(\sigma) = \emptyset$). Once all lists $L_1, L_2, \ldots, L_n$ are processed as above, $L(\sigma)$ contains the sorted list of the dual disks of $\tilde{P}_\sigma$ by their indices. The total time of this sorting algorithm is linear in $\sum_{\sigma \in \Xi} |\tilde{P}_\sigma|$, which is $O(nr)$.

After the pruning procedure, we proceed with $\tilde{P}(\sigma)$ as follows. Suppose k is the largest index of any dual disk $\tilde{p}_k \in \tilde{P}(\sigma)$ such that $\tilde{\gamma}(\tilde{p}_k) < i$; then we have $\tilde{a}_\sigma(\tilde{s}_i) = k$. As such, we can scan the two lists $\tilde{P}(\sigma)$ and $\tilde{S}(\sigma)$ simultaneously (recall that dual points of $\tilde{S}(\sigma)$ are

also sorted by their indices), which can compute $\tilde{a}_\sigma(\tilde{s}_i)$'s for all dual points $\tilde{s}_i \in \tilde{S}(\sigma)$ in $O(|\tilde{P}(\sigma)| + |\tilde{S}(\sigma)|)$ time. As $\sum_{\sigma\in\Xi} |\tilde{P}(\sigma)| = O(nr)$ and $\sum_{\sigma\in\Xi} |\tilde{S}(\sigma)| = O(m \log r)$, the total time for doing this for all cells $\sigma \in \Xi$ is $O(m \log r + nr)$.

Finally, we do the following additional processing for the last cutting Ξ_k. For each cell $\sigma \in \Xi_k$, if $|\tilde{S}(\sigma)| > m/r^2$, we partition $\tilde{S}(\sigma)$ into *standard subsets* of sizes between $m/(2r^2)$ and m/r^2. Recall that $\tilde{P}_\sigma$ is the subset of dual disks whose lower arcs intersect σ. For each standard subset $\tilde{S}_1(\sigma)$ of $\tilde{S}(\sigma)$, we form a subproblem on $(\tilde{P}_\sigma, \tilde{S}_1(\sigma))$: Compute $\tilde{a}_\sigma(i)$ for all dual points $\tilde{s}_i \in \tilde{S}_1(\sigma)$ with respect to dual disks of $\tilde{P}_\sigma$, where $\tilde{a}_\sigma(i)$ is the largest index k of a dual disk $\tilde{p}_k \in \tilde{P}_\sigma$ that contains a dual point $\tilde{s}_j$ with $j < i$ but does not contain $\tilde{s}_i$. After the subproblem is solved, we update $\tilde{a}(i) = \max\{\tilde{a}(i), \tilde{a}_\sigma(i)\}$ for each dual point $\tilde{s}_i \in \tilde{S}_1(\sigma)$. This will compute $\tilde{a}(i)$ correctly. Note that there are $O(r^2)$ subproblems in total and in each subproblem $|\tilde{P}_\sigma| \le n/r$ and $|\tilde{S}_1(\sigma)| \le m/r^2$.

Recall that $T(n, m)$ refers to our problem for computing $a(i)$'s on (P, S), which is equivalent to computing $\tilde{a}(i)$'s on $(\tilde{P}, \tilde{S})$ by Lemma 12. Hence, we can obtain the following recurrence relation using the second algorithm:

$$T(n, m) = O(nr + m \log r) + O(r^2) \cdot T(n/r, m/r^2). \tag{4}$$

Combining the two algorithms. Following exactly the same approach and the same analysis as in Section 4.1 and using (3) and (4), we can obtain a combined algorithm that can compute $a(i)$ for all disks $s_i \in S$ in $O((n + m) \log(n + m)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.

We summarize our result in the following theorem.

▶ **Theorem 13.** *Given a set P of n points and a set S of m unit disks in the plane such that the disk centers are separated from points of P by a line, the disk coverage problem for P and S is solvable in $O((n + m) \log(n + m)) + m^{2/3}n^{2/3}2^{O(\log^*(n+m))}$ time.*

References

1. Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discrete and Computational Geometry*, 63:460–482, 2020.
2. Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG)*, pages 449–458, 2006.
3. 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.
4. Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 460–471, 2005.
5. Ahmad Biniaz, Prosenjit Bose, Paz Carmi, Anil Maheshwari, J.Ian Munro, and Michiel Smid. Faster algorithms for some optimization problems on collinear points. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 8:1–8:14, 2018.
6. 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.
7. Paz Carmi, Matthew J. Katz, and Nissan Lev-Tov. Covering points by unit disks of fixed location. In *Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*, pages 644–655, 2007.

8 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.

9 Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *Proceedings of 36th International Symposium on Computational Geometry (SoCG)*, pages 27:1–27:14, 2020.

10 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

11 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.

12 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.

13 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Homogeneous 2-hop broadcast in 2D. *Computational Geometry: Theory and Applications*, 43:182–190, 2010.

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.

15 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444, 1988.

16 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011.

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

18 Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47:489–501, 2005.

19 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.

20 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010.

21 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.

22 Haitao Wang. Unit-disk range searching and applications. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 32:1–32:17, 2022.