Geometric Hitting Set for Line-Constrained Disks

Gang Liu(✉) and Haitao Wang

University of Utah, Salt Lake City, UT 84112, USA
{u0866264,haitao.wang}@utah.edu

Abstract. Given a set P of n weighted points and a set S of m disks in the plane, the hitting set problem is to compute a subset P' of points of P such that each disk contains at least one point of P' and the total weight of all points of P' is minimized. The problem is known to be NP-hard. In this paper, we consider a line-constrained version of the problem in which all disks are centered on a line ℓ. We present an $O((m+n)\log(m+n)+\kappa\log m)$ time algorithm for the problem, where κ is the number of pairs of disks that intersect. For the unit-disk case where all disks have the same radius, the running time can be reduced to $O((n+m)\log(m+n))$. In addition, we solve the problem in $O((m+n)\log(m+n))$ time in the L_∞ and L_1 metrics, in which a disk is a square and a diamond, respectively.

Keywords: Hitting set · Line-constrained · Disks · Coverage

1 Introduction

Let S be a set of m disks and P a set of n points in the plane such that each point of P has a weight. The *hitting set problem* is to find a subset $P_{opt} \subseteq P$ of minimum total weight so that each disk of S contains a least one point of P_{opt} (i.e., each disk is *hit* by a point of P_{opt}). The problem is NP-hard even if all disks have the same radius and all point weights are the same [8,14,17].

In this paper, we consider the *line-constrained* version of the problem in which centers of all disks of S are on a line ℓ (e.g., the x-axis). To the best of our knowledge, this line-constrained problem was not particularly studied before. We give an algorithm of $O((m+n)\log(m+n)+\kappa\log m)$ time, where κ is the number of pairs of disks that intersect. We also present an alternative algorithm of $O(nm\log(m+n))$ time. For the *unit-disk case* where all disks have the same radius, we give a better algorithm of $O((n+m)\log(m+n))$ time. We also consider the problem in L_∞ and L_1 metrics (the original problem is in the L_2 metric), where a disk becomes a square and a diamond, respectively; we solve the problem in $O((m+n)\log(m+n))$ time in both metrics. The 1D case where all disks are line segments can also be solved in $O((m+n)\log(m+n))$ time.

This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356. A full version of this paper is available at http://arxiv.org/abs/2305.09045.

P. Morin and S. Suri (Eds.): WADS 2023, LNCS 14079, pp. 574–587, 2023.
https://doi.org/10.1007/978-3-031-38906-1_38

In addition, by a reduction from the element uniqueness problem, we prove an $\Omega((m+n)\log(m+n))$ time lower bound in the algebraic decision tree model even for the 1D case (even if all segments have the same length and all points of P have the same weight). The lower bound implies that our algorithms for the unit-disk, L_∞, L_1, and 1D cases are all optimal.

Related Work. The hitting set and many of its variations are fundamental and have been studied extensively; the problem is usually hard to solve, even approximately [15]. Hitting set problems in geometric settings have also attracted much attention and most problems are NP-hard, e.g., [4,5,10,12,16], and some approximation algorithms are known [10,16].

A "dual" problem is the coverage problem. For our problem, we can define its *dual coverage problem* as follows. Given a set P^* of n weighted disks and a set S^* of m points, the problem is to find a subset $P^*_{opt} \subseteq P^*$ of minimum total weight so that each point of S^* is covered by at least one disk of P^*_{opt}. This problem is also NP-hard [11]. The line-constrained problem was studied before and polynomial time algorithms were proposed [19]. The time complexities of the algorithms of [19] match our results in this paper. Specifically, an algorithm of $O((m+n)\log(m+n)+\kappa^*\log n)$ time was given in [19] for the L_2 metric, where κ^* is the number of pairs of disks that intersect [19]; the unit-disk, L_∞, L_1, and 1D cases were all solved in $O((n+m)\log(m+n))$ time [19]. Other variations of line-constrained coverage have also been studied, e.g., [1,3,18].

Our Approach. We propose a novel and interesting method, dubbed *dual transformation*, by reducing our hitting set problem to the 1D dual coverage problem and consequently solve it by applying the 1D dual coverage algorithm of [19]. Indeed, to the best of our knowledge, we are not aware of such a dual transformation in the literature. Two issues arise for this approach: The first one is to prove a good upper bound on the number of segments in the 1D dual coverage problem and the second is to compute these segments efficiently. These difficulties are relatively easy to overcome for the 1D, unit-disk, and L_1 cases. The challenge, however, is in the L_∞ and L_2 cases. Based on many interesting observations and techniques, we prove an $O(n+m)$ upper bound and present an $O((n+m)\log(n+m))$ time algorithm to compute these segments for the L_∞ case; for the L_2 case, we prove an $O(m+\kappa)$ upper bound and derive an $O((n+m)\log(n+m)+\kappa\log m)$ time algorithm.

Outline. In Sect. 2, we define notation and some concepts. Section 3 introduces the dual transformation and solves the 1D, unit-disk, and L_1 cases. Algorithms for the L_∞ and L_2 cases are presented in Sects. 4 and 5, respectively. The lower bound proof can be bound in Sect. 6. Due to the space limit, many details and proofs are ommited but can be found in the full paper.

2 Preliminaries

We follow the notation defined in Sect. 1, e.g., P, S, P_{opt}, κ, ℓ, etc. In this section, unless otherwise stated, all statements, notation, and concepts are applicable for all three metrics, i.e., L_1, L_2, and L_∞, as well as the 1D case. Recall that we assume ℓ is the x-axis, which does not lose generality for the L_2 case but is special for the L_1 and L_∞ cases.

We assume that all points of P are above or on ℓ since if a point $p \in P$ is below ℓ, we could replace p by its symmetric point with respect to ℓ and this would not affect the solution as all disks are centered at ℓ. For ease of exposition, we make a general position assumption that no two points of P have the same x-coordinate and no point of P lies on the boundary of a disk of S (these cases can be handled by standard perturbation techniques [9]). We also assume that each disk of S is hit by at least one point of P since otherwise there would be no solution (we could check whether this is the case by slightly modifying our algorithms).

For any point p in the plane, we use $x(p)$ and $y(p)$ to refer to its x- and y-coordinates, respectively. We sort all points of P in ascending order of their x-coordinates; let $\{p_1, p_2, \cdots, p_n\}$ be the sorted list.

For any point $p \in P$, we use $w(p)$ to denote its weight. We assume that $w(p) > 0$ for each $p \in P$ since otherwise one could always include p in the solution.

We sort all disks of S by their centers from left to right; let $s_1, s_2, \cdots, s_m$ be the sorted list. For each disk $s_j \in S$, let l_j and r_j denote its leftmost and rightmost points on ℓ, respectively. Note that l_j is the leftmost point of s_j and r_j is the rightmost point of s_j. More specifically, l_j (resp., r_j) is the only leftmost (resp., rightmost) point of s_j in the 1D, L_1, and L_2 cases. For each of exposition, we make a general position assumption that no two points of $\{l_i, r_i \mid 1 \leq i \leq m\}$ are coincident.

For $1 \leq j_1 \leq j_2 \leq m$, let $S[j_1, j_2]$ denote the subset of disks $s_j \in S$ for all $j \in [j_1, j_2]$.

We often talk about the relative positions of two geometric objects O_1 and O_2 (e.g., two points, or a point and a line). We say that O_1 is to the *left* of O_2 if $x(p) \leq x(p')$ holds for any point $p \in O_1$ and any point $p' \in O_2$, and *strictly left* means $x(p) < x(p')$. Similarly, we can define *right, above, below,* etc.

Non-containment Property. We observe that to solve the problem it suffices to consider only a subset of S with certain property, called the *Non-Containment subset*, defined as follows. We say that a disk of S is *redundant* if it contains another disk of S. The Non-Containment subset, denoted by $\widehat{S}$, is defined as the subset of S excluding all redundant disks. We have the following observation, called the *Non-Containment property.*

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

Observe that it suffices to work on $\widehat{S}$ instead of S. Indeed, suppose P_{opt} is an optimal solution for $\widehat{S}$. Then, for any disk $s \in S \backslash \widehat{S}$, there must be a disk $s' \in \widehat{S}$ such that s contains s'. Hence, any point of P_{opt} hitting s' must hit s as well.

We can easily compute $\widehat{S}$ in $O(m \log m)$ time in any metric. Indeed, because all disks of S are centered at ℓ, a disk s_k contains another disk s_j if and only the segment $s_k \cap \ell$ contains the segment $s_j \cap \ell$. Hence, it suffices to identify all redundant segments from $\{s_j \cap \ell \mid s_j \in S\}$. This can be easily done in $O(m \log m)$ time, e.g., by sweeping endpoints of disks on ℓ; we omit the details.

In what follows, to simplify the notation, we assume $S = \widehat{S}$, i.e., S does not have any redundant disk. As such, S has the Non-Containment property in Observation 1. As will be seen later, the Non-Containment property is very helpful in designing algorithms.

3 Dual Transformation and the 1D, Unit-Disk, and L_1 Problems

By making use of the Non-Containment property of S, we propose a *dual transformation* that can reduce our hitting set problem on S and P to an instance of the 1D dual coverage problem. More specifically, we will construct a set S^* of points and a set P^* of weighted segments on the x-axis such that an optimal solution for the coverage problem on S^* and P^* corresponds to an optimal solution for our original hitting set problem. We refer to it as the *1D dual coverage problem*. To differentiate from the original hitting set problem on P and S, we refer to the points of S^* as *dual points* and the segments of P^* as *dual segments*.

As will be seen later, $|S^*| = m$, but $|P^*|$ varies depending on the specific problem. Specifically, $|P^*| \leq n$ for the 1D, unit-disk, and L_1 cases, $|P^*| = O(n+m)$ for the L_∞ case, and $|P^*| = O(m+\kappa)$ for the L_2 case. In what follows, we present the details of the dual transformation by defining S^* and P^*.

For each disk $s_j \in S$, we define a dual point s_j^* on the x-axis with x-coordinate equal to j. Define S^* as the set of all m points $s_1^*, s_2^*, \ldots, s_m^*$. As such, $|S^*| = m$.

We next define the set P^* of dual segments. For each point $p_i \in P$, let I_i be the set of indices of the disks of S that are hit by p_i. We partition the indices of I_i into maximal intervals of consecutive indices and let $\mathcal{I}_i$ be the set of all these intervals. By definition, for each interval $[j_1, j_2] \in \mathcal{I}_i$, p_i hits all disks s_j with $j_1 \leq j \leq j_2$ but does not hit either s_{j_1-1} or s_{j_2+1}; we define a dual segment on the x-axis whose left (resp., right) endpoint has x-coordinate equal to j_1 (resp., j_2) and whose weight is equal to $w(p_i)$ (for convenience, we sometimes also use the interval $[j_1, j_2]$ to represent the dual segment and refer to dual segments as intervals). We say that the dual segment is *defined* or *generated* by p_i. Let P_i^* be the set of dual segments defined by the intervals of $\mathcal{I}_i$. We define $P^* = \bigcup_{i=1}^{n} P_i^*$. The following observation follows the definition of dual segments.

Observation 2. *p_i hits a disk s_j if and only if a dual segment of P_i^* covers the dual point s_j^*.*

Suppose we have an optimal solution P_{opt}^* for the 1D dual coverage problem on P^* and S^*, we obtain an optimal solution P_{opt} for the original hitting set problem on P and S as follow: for each segment of P_{opt}^*, if it is from P_i^* for some i, then we include p_i into P_{opt}.

Clearly, $|S^*| = m$. We will prove later in this section that $|P_i^*| \leq 1$ for all $1 \leq i \leq n$ in the 1D problem, the unit-disk case, and the L_1 metric, and thus $|P^*| \leq n$ for all these cases. Since $|P_i^*| \leq 1$ for all $1 \leq i \leq n$, in light of Observation 2, P_{opt} constructed above is an optimal solution of the original hitting set problem. Therefore, one can solve the original hitting set problem for the above cases with the following three main steps: (1) Compute S^* and P^*; (2) apply the algorithm for the 1D dual coverage problem in [19] to compute P_{opt}^*, which takes $O((|S^*|+|P^*|)\log(|S^*|+|P^*|))$ time [19]; (3) derive P_{opt} from P_{opt}^*. For the first step, computing S^* is straightforward. For P^*, we will show later that for all above three cases (1D, unit-disk, L_1), P^* can be computed in $O((n+m)\log(n+m))$ time. As $|S^*| = m$ and $|P^*| \leq n$, the second step can be done in $O((n+m)\log(n+m))$ time [19]. As such, the hitting set problem of the above three cases can be solved in $O((n+m)\log(n+m))$ time.

For the L_∞ metric, we will prove in Sect. 4 that $|P^*| = O(n+m)$ but each P_i^* may have multiple segments. If P_i^* has multiple segments, a potential issue is the following: If two segments of P_i^* are in P_{opt}^*, then the weights of both segments will be counted in the optimal solution value (i.e., the total weight of all segments of P_{opt}^*), which corresponds to counting the weight of p_i twice in P_{opt}. To resolve the issue, we prove in Sect. 4 that even if $|P_i^*| \geq 2$, at most one dual segment of P_i^* will appear in any optimal solution P_{opt}^*. As such, P_{opt} constructed above is an optimal solution for the original hitting set problem. Besides proving the upper bound $|P^*| = O(n+m)$, another challenge of the L_∞ problem is to compute P^* efficiently, for which we propose an $O((n+m)\log(n+m))$ time algorithm. Consequently, the L_∞ hitting set problem can be solved in $O((n+m)\log(n+m))$ time.

For the L_2 metric, we will show in Sect. 5 that $|P^*| = O(m+\kappa)$. Like the L_∞ case, each P_i^* may have multiple segments but we can also prove that P_i^* can contribute at most one segment to any optimal solution P_{opt}^*. Hence, P_{opt} constructed above is an optimal solution for the original hitting set problem. We present an algorithm that can compute P^* in $O((n+m)\log(n+m)+\kappa\log m)$ time. As such, the L_2 hitting set problem can be solved in $O((m+n)\log(m+n)+\kappa\log m)$ time. Alternatively, a straightforward approach can prove $|P^*| = O(nm)$ and compute P^* in $O(nm)$ time; hence, we can also solve the problem in $O(nm\log(n+m))$ time.

In the rest of this section, following the above framework, we solve the unit-disk case. Due to the space limit, the 1D and the L_1 cases are omitted but can be found in the full paper.

3.1 The Unit-Disk Case

In the unit-disk case, all disks of S have the same radius. We follow the dual transformation and have the following lemma.

Lemma 1. *In the unit-disk case, $|P_i^*| \leq 1$ for any $1 \leq i \leq n$. In addition, P_i^* for all $1 \leq i \leq n$ can be computed in $O((n+m)\log(n+m))$ time.*

Proof. Consider a point $p_i \in P$. Observe that p_i hits a disk s_j if and only if the segment $D(p_i) \cap \ell$ covers the center of s_j, where $D(p_i)$ is the unit disk centered at p_i. By definition, the indices of the disks whose centers are covered by the segment $D(p_i) \cap \ell$ must be consecutive. Hence, $|P_i^*| \leq 1$ must hold.

To compute P_i^*, it suffices to determine the disks whose centers are covered by $D(p_i) \cap \ell$. This can be easily done in $O((n+m)\log(n+m))$ time for all $p_i \in P$ (e.g., first sort all disk centers and then do binary search on the sorted list with the two endpoints of $D(p_i) \cap \ell$ for each $p_i \in P$). $\square$

In light of Lemma 1, using the dual transformation, we have the following result.

Theorem 1. *The line-constrained unit-disk hitting set problem is solvable in* $O((n + m)\log(n + m))$ *time.*

Note that in both the 1D and the L_1 cases we can prove results similar to Lemma 1 and thus solve both cases in $O((n + m)\log(n + m))$ time. The details are omitted but can be found in the full paper.

4 The L_∞ Metric

In this section, following the dual transformation, we present an $O((m + n)\log(m + n))$ time algorithm for L_∞ case.

In the L_∞ metric, each disk is a square whose edges are axis-parallel. For a disk $s_j \in S$ and a point $p_i \in P$, we say that p is *vertically above* s_j if p_i is outside s_j and $x(l_j) \leq x(p_i) \leq x(r_j)$.

In the L_∞ metric, using the dual transformation, it is easy to come up with an example in which $|P_i^*| \geq 2$. Observe that $|P_i^*| \leq \lceil m/2 \rceil$ as the indices of S can be partitioned into at most $\lceil m/2 \rceil$ disjoint maximal intervals. Despite $|P_i^*| \geq 2$, the following critical lemma shows that each P_i^* can contribute at most one segment to any optimal solution of the 1D dual coverage problem on P^* and S^*. The proof of the lemma can be found in the full paper.

Lemma 2. *In the* L_∞ *metric, for any optimal solution* P^*_{opt} *of the 1D dual coverage problem on* P^* *and* S^**,* P^*_{opt} *contains at most one segment from* P_i^* *for any* $1 \leq i \leq n$*.*

Lemma 2 implies that an optimal solution to the 1D dual coverage problem on P^* and S^* still corresponds to an optimal solution of the original hitting set problem on P and S. As such, it remains to compute the set P^* of dual segments. In what follows, we first prove an upper bound for $|P^*|$.

4.1 Upper Bound for $|P^*|$

As $|P_i^*| \leq \lceil m/2 \rceil$, an obvious upper bound for $|P^*|$ is $O(mn)$. Below we reduce it to $O(m + n)$.

Our first observation is that if the same dual segment of P^* is defined by more than one point of P, then we only need to keep the one whose weight is minimum. In this way, all segments of P^* are distinct (i.e., P^* is not a multi-set).

We sort all points of P from top to bottom as $q_1, q_2, \ldots, q_n$. For ease of exposition, we assume that no point of P has the same y-coordinate as the upper edge of any disk of S. For each $2 \leq i \leq n$, let S_i denote the subset of disks whose upper edges are between q_{i-1} and q_i. Let S_1 denote the subset of disks whose upper edges are above q_1. For each $1 \leq i \leq n$, let $m_i = |S_i|$.

We partition the indices of disks of S_1 into a set $\mathcal{I}_1$ of maximal intervals. Clearly, $|\mathcal{I}_1| \leq m_1$. The next lemma shows that other than the dual segments corresponding to the intervals in $\mathcal{I}_1$, q_1 can generate at most two dual segments in P^*.

Lemma 3. *The number of dual segments of $P^* \setminus \mathcal{I}_1$ defined by q_1 is at most* 2.

Proof. Assume to the contrary that q_1 defines three intervals $[j_1, j_1']$, $[j_2, j_2']$, and $[j_3, j_3']$ in $P^* \setminus \mathcal{I}_1$, with $j_1' < j_2$ and $j_2' < j_3$. By definition, $\mathcal{I}_1$ must have an interval, denoted by I_k, that strictly contains $[j_k, j_k']$ (i.e., $[j_k, j_k'] \subset I_k$), for each $1 \leq k \leq 3$. Then, I_2 must contain an index j that is not in $[j_1, j_1'] \cup [j_2, j_2'] \cup [j_3, j_3']$ with $j_1' < j < j_3$ (e.g., see Fig. 1). As such, q_1 does not hit s_j. Also, since $j \in I_2$, s_j is in S_1.

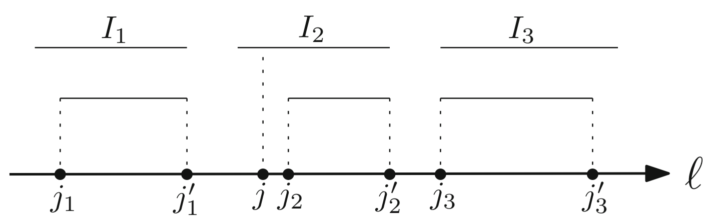

Fig. 1. Illustrating a schematic view of the intervals $[j_k, j_k']$ and I_k for $1 \leq k \leq 3$.

Since $j_1' < j < j_3$, due to the Non-Containment property of S, $x(l_j) \leq x(l_{j_3})$ and $x(r_{j_1'}) \leq x(r_j)$. As q_1 hits both $s_{j_1'}$ and s_{j_3}, we have $x(l_{j_3}) \leq x(p_1) \leq x(r_{j_1'})$. Hence, we obtain $x(l_j) \leq x(q_1) \leq x(r_j)$. Since q_1 does not hit s_j, the upper edge of s_j must be below q_1. But this implies that s_j is not in S_1, which incurs contradiction. □

Now we consider the disks of S_2 and the dual segments defined by q_2. For each disk s_j of S_2, we update the intervals of $\mathcal{I}_1$ by adding the index j, as follows. Note that by definition, intervals of $\mathcal{I}_1$ are pairwise disjoint and no interval contains j.

1. If neither $j+1$ nor $j-1$ is in any interval of $\mathcal{I}_1$, then we add $[j, j]$ as a new interval to $\mathcal{I}_1$.
2. If $j+1$ is contained in an interval $I \in \mathcal{I}_1$ while $j-1$ is not, then $j+1$ must be the left endpoint of I. In this case, we add j to I to obtain a new interval I' (which has j as its left endpoint) and add I' to $\mathcal{I}_1$; but we still keep I in $\mathcal{I}_1$.

3. Symmetrically, if $j-1$ is contained in an interval $I \in \mathcal{I}_1$ while $j+1$ is not, then we add j to I to obtain a new interval I' and add I' to $\mathcal{I}_1$; we still keep I in $\mathcal{I}_1$.
4. If both $j+1$ and $j-1$ are contained in intervals of $\mathcal{I}_1$, they must be contained in two intervals, respectively; we merge these two intervals into a new interval by padding j in between and adding the new interval to $\mathcal{I}_1$. We still keep the two original intervals in $\mathcal{I}_1$.

Let $\mathcal{I}_1'$ denote the updated set $\mathcal{I}_1$ after the above operation. Clearly, $|\mathcal{I}_1'| \leq |\mathcal{I}_1| + 1$.

We process all disks $s_j \in S_2$ as above; let $\mathcal{I}_2$ be the resulting set of intervals. It holds that $|\mathcal{I}_2| \leq |\mathcal{I}_1| + |S_2| \leq m_1 + m_2$. Also observe that for any interval I of indices of disks of $S_1 \cup S_2$ such that I is not in $\mathcal{I}_2$, $\mathcal{I}_2$ must have an interval I' such that $I \subset I'$ (i.e., $I \subseteq I'$ but $I \neq I'$). Using this property, by exactly the same analysis as Lemma 3, we can show that other than the intervals in $\mathcal{I}_2$, q_2 can generate at most two intervals in P^*. Since $\mathcal{I}_1 \subseteq \mathcal{I}_2$, combining Lemma 3, we obtain that other than the intervals of $\mathcal{I}_2$, the number of intervals of P^* generated by q_1 and q_2 is at most 4.

We process disks of S_i and point q_i in the same way as above for all $i = 3, 4, \ldots, n$. Following the same argument, we can show that for each i, we obtain an interval set $\mathcal{I}_i$ with $\mathcal{I}_{i-1} \subseteq \mathcal{I}_i$ and $|\mathcal{I}_i| \leq \sum_{k=1}^{i} m_k$, and other than the intervals of $\mathcal{I}_i$, the number of intervals of P^* generated by $\{q_1, q_2, \ldots, q_i\}$ is at most $2i$. In particular, $|\mathcal{I}_n| \leq \sum_{k=1}^{n} m_k \leq m$, and other than the intervals of $\mathcal{I}_n$, the number of intervals of P^* generated by $P = \{q_1, q_2, \ldots, q_n\}$ is at most $2n$. We thus achieve the following conclusion.

Lemma 4. *In the L_∞ metric, $|P^*| \leq 2n + m$.*

4.2 Computing P^*

Using Lemma 4, we present an algorithm that computes P^* in $O((n+m)\log(n+m))$ time.

For each segment $I \in P^*$, let $w(I)$ denote its weight. We say that a segment I of P^* is *redundant* if there is another segment I' such that $I \subset I'$ and $w(I) \geq w(I')$. Clearly, any redundant segment of P^* cannot be used in any optimal solution for the 1D dual coverage problem on S^* and P^*. A segment of P^* is *non-redundant* if it is not redundant.

In the following algorithm, we will compute a subset P_0^* of P^* such that segments of $P^* \backslash P_0^*$ are all redundant (i.e., the segments of P^* that are not computed by the algorithm are all redundant and thus are useless). We will show that each segment reported by the algorithm belongs to P^* and thus the total number of reported segments is at most $2n+m$ by Lemma 4. We will show that the algorithm spends $O(\log(n+m))$ time reporting one segment and each segment is reported only once; this guarantees the $O((n+m)\log(n+m))$ upper bound of the runtime of the algorithm.

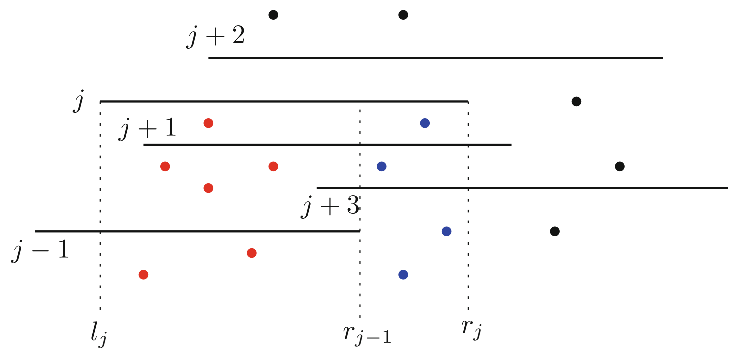

Fig. 2. Illustrating P_j^1 (the red points) and P_j^2 (the blue points). Only the upper edges of disks are shown. The numbers are the indices of disks. (Color figure online)

For each disk $s_j \in S$, we use $y(s_j)$ to denote the y-coordinate of the upper edge of s_j.

Our algorithm has m iterations. In the j-th iteration, it computes all segments in P_j^*, where P_j^* is the set of all non-redundant segments of P^* whose starting indices are j, although it is possible that some redundant segments with starting index j may also be computed. Points of P defining these segments must be inside s_j; let P_j denote the set of points of P inside s_j. We partition P_j into two subsets (e.g., see Fig. 2): P_j^1 consists of points of P_j to the left of r_{j-1} and P_j^2 consists of points of P_j to the right of r_{j-1}. We will compute dual segments of P_j^* defined by P_j^1 and P_j^2 separately; one reason for doing so is that when computing dual segments defined by a point of P_j^1, we need to additionally check whether this point also hits s_{j-1} (if yes, such a dual segment does not exist in P^* and thus will not be reported). In the following, we first describe the algorithm for P_j^1 since the algorithm for P_j^2 is basically the same but simpler. Note that our algorithm does not need to explicitly determine the points of P_j^1 or P_j^2; rather we will build some data structures that can implicitly determine them during certain queries.

If the upper edge of s_{j-1} is higher than that of s_j, then all points of P_j^1 are in s_{j-1} and thus no point of P_j^1 defines any dual segment of P^* starting from j. Indeed, assume to the contrary that a point $p_i \in P_j^1$ defines such a dual segment $[j, j']$. Then, since p_i is in s_{j-1}, $[j, j']$ cannot be a maximal interval of indices of disks hit by p_i and thus cannot be a dual segment defined by p_i. In what follows, we assume that the upper edge of s_{j-1} is lower than that of s_j. In this case, it suffices to only consider points of P_j^1 above s_{j-1} since points below the upper edge of s_{j-1} (and thus are inside s_{j-1}) cannot define any dual segments due to the same reason as above. Nevertheless, our algorithm does not need to explicitly determine these points.

We start with performing the following *rightward segment dragging query*: Drag the vertical segment $x(l_j) \times [y(s_{j-1}), y(s_j)]$ rightwards until a point $p \in P$ and return p (e.g., see Fig. 3). Such a segment dragging query can be answered in $O(\log n)$ time after $O(n \log n)$ time preprocessing on P (e.g., using Chazelle's result [6] one can build a data structure of $O(n)$ space in $O(n \log n)$ time such that

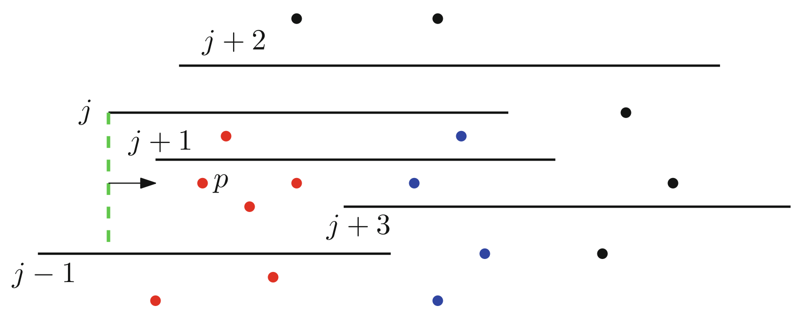

Fig. 3. Illustrating the rightward segment dragging query: the green dashed segment is the dragged segment $x(l_j) \times [y(s_{j-1}), y(s_j)]$. (Color figure online)

each query can be answered in $O(\log n)$ time; alternatively, if one is satisfied with an $O(n \log n)$ space data structure, then an easier solution is to use fractional cascading [7] and one can build a data structure in $O(n \log n)$ time and space with $O(\log n)$ query time). If the query does not return any point or if the query returns a point p with $x(p) > x(r_{j-1})$, then P_j^1 does not have any point above s_{j-1} and we are done with the algorithm for P_j^1. Otherwise, suppose the query returns a point p with $x(p) \leq x(r_{j-1})$; we proceed as follows.

We perform the following *max-range query* on p: Compute the largest index k such that all disks in $S[j,k]$ are hit by p (e.g., in Fig. 3, $k = j + 2$). We will show later in Lemma 5 that after $O(m \log m)$ time and $O(m)$ space processing, each such query can be answered in $O(\log m)$ time. Such an index k must exist as s_j is hit by p. Observe that $[j,k]$ is a dual segment in P^* defined by p. However, the weight of $[j,k]$ may not be equal to $w(p)$, because it is possible that a point with smaller weight also defines $[j,k]$. Our next step is to determine the minimum-weight point that defines $[j,k]$.

We perform a *range-minima query* on $[j,k]$: Find the lowest disk among all disks in $S[j,k]$ (e.g., in Fig. 3, s_{j+1} is the answer to the query). This can be easily done in $O(\log m)$ time with $O(m)$ space and $O(m \log m)$ time preprocessing. Indeed, we can build a binary search tree on the upper edges of all disks of S with their y-coordinates as keys and have each node storing the lowest disk among all leaves in the subtree rooted at the node. A better but more complicated solution is to build a range-minima data structure on the y-coordinates of the upper edges of all disks in $O(m)$ time and each query can be answered in $O(1)$ time [2,13]. However, the binary search tree solution is sufficient for our purpose. Let y^* be the y-coordinate of the upper edge of the disk returned by the query.

We next perform the following *downward min-weight point query* for the horizontal segment $[x(l_k), x(r_{j-1})] \times y^*$: Find the minimum weight point of P below the segment (e.g., see Fig. 4). We will show later in Lemma 6 that after $O(n \log n)$ time and space preprocessing, each query can be answered in $O(\log n)$ time. Let p' be the point returned by the query. If $p' = p$, then we report $[j,k]$ as a dual segment with weight equal to $w(p)$. Otherwise, if p' is inside s_{j-1} or s_{k+1}, then $[j,k]$ is a redundant dual segment (because a dual segment defined by p' strictly contains $[j,k]$ and $w(p') \leq w(p)$) and thus we do not need to report it. In any case, we proceed as follows.

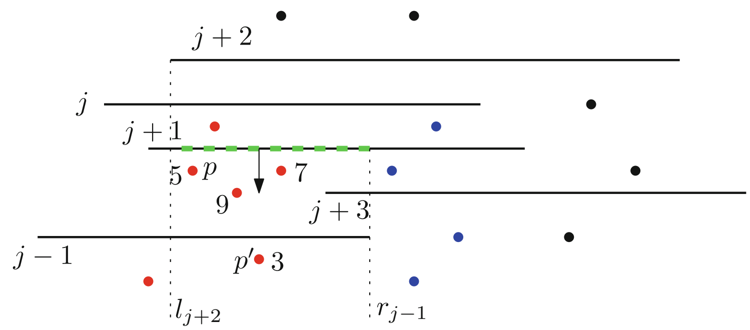

Fig. 4. Illustrating the downward min-weight point query (with $k = j + 2$): the green dashed segment is the dragged segment $[x(l_k), x(r_{j-1})] \times y^*$. The numbers besides the points are their weights. The answer to the query is p', whose weight is 3. (Color figure online)

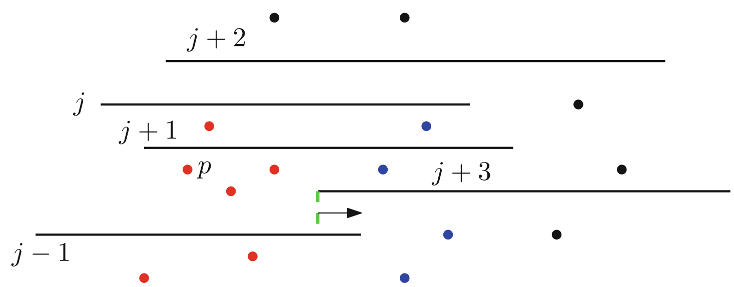

Fig. 5. Illustrating the rightwards segment dragging query: the green dashed segment is the dragged segment $x(l_{k+1}) \times [y(s_{j-1}), y']$. (Color figure online)

The above basically determines that $[j, k]$ is a dual segment in P^*. Next, we determine those dual segments $[j, k']$ with $k' > k$. If such a segment exists, the interval $[j, k']$ must contain index $k + 1$. Hence, we next consider s_{k+1}. If $y(s_{k+1}) > y(s_{j-1})$, then let $y' = \min\{y^*, y(s_{k+1})\}$; we perform a rightward segment dragging query with the vertical segment $x(l_{k+1}) \times [y(s_{j-1}), y']$ (e.g., see Fig. 5) and then repeat the above algorithm. If $y(s_{k+1}) \leq y(s_{j-1})$, then points of P_j^1 above s_{j-1} are also above s_{k+1} and thus no point of P_j^1 can generate any dual segment $[j, k']$ with $k' > k$ and thus we are done with the algorithm on P_j^1.

For time analysis, we charge the time of the above five queries to the interval $[j, k]$, which is in P^*. Note that $[j, k]$ will not be charged again in future because future queries in the j-th iteration will be charged to $[j, k']$ for some $k' > k$ and future queries in the j'-th iteration for any $j' > j$ will be charged to $[j', k'']$. As such, each dual segment of P^* is charged $O(1)$ times in the entire algorithm. As each query takes $O(\log(n + m))$ time, the total time of all queries in the entire algorithm is $O(|P^*| \log(n + m))$, which is $((n + m) \log(n + m))$ by Lemma 4.

Proofs of Lemmas 5 and 6 are in the full paper.

Lemma 5. *With $O(m \log m)$ time and $O(m)$ space preprocessing on S, each max-range query can be answered in $O(\log m)$ time.*

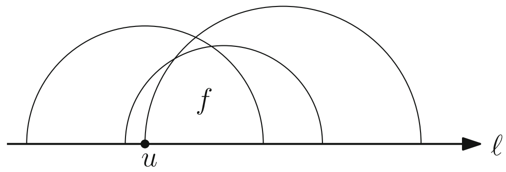

Fig. 6. Illustrating an initial face f with leftmost vertex u.

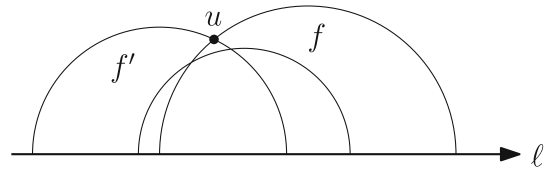

Fig. 7. A non-initial face f with leftmost vertex u and its opposite face f'.

Lemma 6. *With $O(n \log n)$ time and space preprocessing on P, each downward min-weight point query can be answered in $O(\log n)$ time.*

This finishes the description of the algorithm for P_j^1. The algorithm for P_j^2 is similar with the following minor changes. First, when doing each rightward segment dragging query, the lower endpoint of the query vertical segment is at $-\infty$ instead of $y(s_{j-1})$. Second, when the downward min-weight point query returns a point, we do not have to check whether it is in s_{j-1} anymore. The rest of the algorithm is the same. In this way, all non-redundant intervals of P^* starting at index j can be computed. As analyzed above, the runtime of the entire algorithm is bounded by $O((n+m)\log(n+m))$.

As such, using the dual transformation, we have the following result.

Theorem 2. *The line-constrained L_∞ hitting set can be solved in $O((n+m)\log(n+m))$ time.*

5 The L_2 Case – A Sketch

Due to the space limit, we only sketch our result for the L_2 case; the full details can be found in the full paper.

As in the L_∞ case, $|P_i^*| \geq 2$ is possible and $|P_i^*| \leq \lceil m/2 \rceil$. We first prove a lemma similar to Lemma 2, following a similar proof scheme. As such, it suffices to find an optimal solution to the 1D dual coverage problem on P^* and S^*.

Upper bound for $|P^*|$. We then prove the upper bound $|P^*| = O(m+\kappa)$, where κ is the number of pairs of disks of S that intersect. To this end, we consider the arrangement $\mathcal{A}$ of the boundaries of all disks of S in the half-plane above ℓ. An easy but critical observation is that points of P located in the same face of $\mathcal{A}$ define the same subset of dual segments of P^*. As such, it suffices to consider the dual segments defined by all faces of $\mathcal{A}$.

We define *initial faces* of $\mathcal{A}$. Roughly speaking, a face is an *initial face* if its leftmost vertex is on ℓ (e.g., see Fig. 6).

We define a directed graph G as follows. The faces of $\mathcal{A}$ form the node set of G. There is an edge from a node f' to another node f if the face f is a non-initial face and f' is the *opposite* face of f (i.e., the rightmost vertex of f' is the leftmost vertex of f; e.g., in Fig. 7, there is a directed edge from f' to f). Since each face of $\mathcal{A}$ has only one leftmost vertex and only one rightmost vertex, each node G

has at most one incoming edge and at most one outgoing edge. Also, each initial face does not have an incoming edge while each non-initial face must have an incoming edge. As such, G is actually composed of a set of directed paths, each of which has an initial face as the first node.

For each face $f \in \mathcal{A}$, let $P^*(f)$ denote the subset of the dual segments of P^* generated by f (i.e., generated by any point in f). Our goal is to obtain an upper bound for $|\bigcup_{f\in\mathcal{A}} P^*(f)|$, which is an upper bound for $|P^*|$ as $P^* \subseteq \bigcup_{f\in\mathcal{A}} P^*(f)$. To this end, we first show that $|P^*(f)| = 1$ for each initial face f and we then show that for any two adjacent faces f' and f in any path of G, the symmetric difference of $P^*(f)$ and $P^*(f')$ is $O(1)$. As such, we can obtain $|P^*| = O(m+\kappa)$ as $\mathcal{A}$ has $O(m+\kappa)$ faces.

Computing P^*. To compute the set P^*, following the above idea, it suffices to compute the dual segments generated by all faces of $\mathcal{A}$ (or equivalently, generated by all nodes of the graph G). The main idea is to directly compute for each path $\pi \in G$ the dual segments defined by the initial face of π and then for each non-initial face $f \in \pi$, determine $P^*(f)$ indirectly based on $P^*(f')$, where f' is the predecessor face of f in π. To this end, after constructing $\mathcal{A}$ and G and other preprocessing, we first show that $P^*(f)$ for each initial face f can be computed in $O(\log m)$ time, and we then show that $P^*(f)$ can be determined in $O(\log m)$ time based on $P^*(f')$, where f' is the predecessor face of f in π, for each path π of G. As such, P^* can be computed in $O(n\log(n+m) + (m+\kappa)\log m)$ time. Consequently, using the dual transformation, the L_2 hitting set problem on P and S can be solved in $O((n+m)\log(n+m) + \kappa\log m)$ time.

6 Lower Bound

We can prove an $\Omega((n+m)\log(n+m))$ time lower bound for the problem even for the 1D unit-disk case (i.e., all segments have the same length), by a simple reduction from the element uniqueness problem (Pedersen and Wang [19] used a similar approach to prove the same lower bound for the 1D coverage problem). Indeed, the element uniqueness problem is to decide whether a set $X = \{x_1, x_2, \ldots, x_N\}$ of N numbers are distinct. We construct an instance of the 1D unit-disk hitting set problem with a point set P and a segment set S on the x-axis ℓ as follows. For each $x_i \in X$, we create a point p_i on ℓ with x-coordinate equal to x_i and create a segment on ℓ that is the point p_i itself. Let $P = \{p_i \mid 1 \le i \le N\}$ and S the set of segments defined above (and thus all segments have the same length); then $|P| = |S| = N$. We set the weights of all points of P to 1. Observe that the elements of X are distinct if and only if the total weight of points in an optimal solution to the 1D unit disk hitting set problem on P and S is n. As the element uniqueness problem has an $\Omega(N\log N)$ time lower bound under the algebraic decision tree model, $\Omega((n+m)\log(n+m))$ is a lower bound for our 1D unit disk hitting set problem.

References

1. Alt, H., et al.: Minimum-cost coverage of point sets by disks. In: Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG), pp. 449–458 (2006)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000). https://doi.org/10.1007/10719839_9
3. Bilò, V., Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: Geometric clustering to minimize the sum of cluster sizes. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 460–471. Springer, Heidelberg (2005). https://doi.org/10.1007/11561071_42
4. Bus, N., Mustafa, N.H., Ray, S.: Practical and efficient algorithms for the geometric hitting set problem. Discrete Appl. Math. **240**, 25–32 (2018)
5. Chan, T.M., Grant, E.: Exact algorithms and APX-hardness results for geometric packing and covering problems. Comput. Geom.: Theory Appl. **47**, 112–124 (2014)
6. Chazelle, B.: An algorithm for segment-dragging and its implementation. Algorithmica **3**(1–4), 205–221 (1988)
7. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica **1**(1), 133–162 (1986)
8. Durocher, S., Fraser, R.: Duality for geometric set cover and geometric hitting set problems on pseudodisks. In: Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG) (2015)
9. Edelsbrunner, H., Mücke, E.P.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graph. **9**, 66–104 (1990)
10. Even, G., Rawitz, D., Shahar, S.: Hitting sets when the VC-dimension is small. Inf. Process. Lett. **95**, 358–362 (2005)
11. Feder, T., Greene, D.H.: Optimal algorithms for approximate clustering. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 434–444 (1988)
12. Ganjugunte, S.K.: Geometric hitting sets and their variants. Ph.D. thesis, Duke University (2011)
13. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**, 338–355 (1984)
14. Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations, pp. 85–103 (1972)
15. Moreno-Centeno, E., Karp, R.M.: The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. Oper. Res. **61**, 453–468 (2013)
16. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problems via local search. In: Proceedings of the 25th Annual Symposium on Computational Geometry (SoCG), pp. 17–22 (2009)
17. Mustafa, N.H., Ray, S.: Improved results on geometric hitting set problems. Discrete Comput. Geom. **44**, 883–895 (2010)
18. Pedersen, L., Wang, H.: On the coverage of points in the plane by disks centered at a line. In: Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG), pp. 158–164 (2018)
19. Pedersen, L., Wang, H.: Algorithms for the line-constrained disk coverage and related problems. Comput. Geom.: Theory Appl. **105–106**(101883), 1–18 (2022)