International Journal of Computational Geometry
& Applications
(2024) 1–23

DOI: 10.1142/S0218195924500018

Algorithms for Covering Barrier Points by Mobile Sensors with Line Constraint*

Princy Jain[†,§] and Haitao Wang[‡,¶]

[†]*Department of Computer Science*
Utah State University, Logan, Utah, USA

[‡]*Kahlert School of Computing*
University of Utah, Salt Lake City, Utah, USA
[§]*princy.jain@usu.edu*
[¶]*haitao.wang@utah.edu*

Received 16 September 2021
Revised 6 September 2023
Accepted 8 January 2024
Published
Communicated by T. Tokuyama

We study the problem of covering barrier points by mobile sensors. Each sensor is represented by a point in the plane with the same covering range r so that any point within distance r from the sensor can be covered by the sensor. Given a set B of m points (called "barrier points") and a set S of n points (representing the "sensors") in the plane, the problem is to move the sensors so that each barrier point is covered by at least one sensor and the maximum movement of all sensors is minimized. The problem is NP-hard. In this paper, we consider two line-constrained variations of the problem and present efficient algorithms that improve the previous work. In the first problem, all sensors are given on a line ℓ and are required to move on ℓ only while the barrier points can be anywhere in the plane. We propose an $O((n+m)\log(n+m))$ time algorithm for the problem. We also consider the weighted case where each sensor has a weight; we give an $O(m \log m + n \log^2 n)$ time algorithm for this case. In the second problem, all barrier points are on ℓ while all sensors are in the plane but are required to move onto ℓ to cover all barrier points. We also solve the weighted case in $O(m \log m + n \log^2 n)$ time.

Keywords: Barrier coverage; disk coverage; geometric coverage; barrier points; mobile sensors; line-constrained; algorithms; computational geometry.

1. Introduction

Let B be a set of m points and D be a set of n disks of the same radius r in the plane. We consider the problem of moving the disks of D to cover all points of B so that the maximum moving distance of all disks is minimized. The problem is

*A preliminary version of this paper appeared in *Proceedings of the* 33*rd Canadian Conference on Computational Geometry* (*CCCG* 2021). This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356.

NP-hard.[a] In this paper, we consider two line-constrained variations of the problem and present efficient algorithms for them.

Due to its potential applications in barrier coverage of mobile sensors in wireless sensor networks,[14,15,17] we consider the problem from the barrier coverage point of view. We call the points of B the *barrier points*. Let S be the set of centers of all disks of D, and points of S are called *sensors*. All sensors have the same *covering range* (or *sensing range*) r so that any point within distance r from a sensor s can be covered by s (i.e., s covers all points in the disk centered at s with radius r). Hence, our problem becomes the following: move sensors of S to cover all barrier points of B such that the maximum moving distance of all sensors is minimized.

We study a line-constrained variation of the problem where all sensors are given on a line ℓ and are required to move on ℓ only while the barrier points can be anywhere in the plane. We also consider its *weighted case* where each sensor s_i has a weight $w_i > 0$ and the *moving cost* of s_i is defined to be its moving distance times w_i.

To the best of our knowledge, we are not aware of any previous work on this particular problem. If all barrier points are all on ℓ, which becomes a 1D problem (our original problem can be considered as a 1.5D problem), the algorithm of Li and Wang[18] can solve the unweighted case in $O(m \log m + n \log m \log n)$ time. In this paper, we present an $O((n+m)\log(n+m))$ time for the unweighted case and an $O((n+m)\log^2(n+m))$ time algorithm for the weighted case. Hence, our algorithm for the unweighted case, albeit solving the 1.5D problem, improves the algorithm of Ref. 18 by roughly a logarithmic factor.

We also consider another problem variation in which all barrier points are on a line ℓ while sensors can be anywhere in the plane. We want to move all sensors onto ℓ to cover all barrier points so that the maximum moving cost of all sensors is minimized. Previously, Huang *et al.*[14] studied the unweighted case and gave an $O(n(m + n\log n)\log(n+m))$ time algorithm. Our techniques solve the weighted case in $O(m \log m + n \log^2 n)$ time. This improves the algorithm of Huang *et al.*[14] by almost a linear factor. Note that we do not have a faster algorithm for the unweighted case. As all barrier points are on ℓ and all sensors will finally move to ℓ, once a sensor s moves to ℓ, the portion of the covering disk of s that is relevant is an interval of ℓ. For this reason, we refer to this problem as the *mobile interval coverage* problem; for differentiation, we refer to the first problem above as the *mobile disk coverage problem*. Note that if sensors have different ranges, even the 1D problem (i.e., all sensors and barrier points are on ℓ) is NP-hard.[14]

1.1. *Related work*

Many variations of mobile sensor barrier coverage problem have been studied in the literature.

[a]This can be proved by an easy reduction from the minimum disk coverage problem;[13] e.g., see Ref. 19 for a reduction for a similar problem.

Czyzowicz *et al.*[6] studied the problem of covering a barrier segment on a line ℓ by moving a set of n sensors on ℓ (the sensors are initially given on ℓ); they gave an $O(n^2)$ time algorithm. Chen *et al.*[3] presented a more efficient $O(n \log n)$ time algorithm. Chen *et al.*[3] also studied the case where sensors may have different covering ranges and proposed an $O(n^2 \log n)$ time algorithm. For the weighted case where the sensors have weights as defined in our problems (but sensors have the same range), Lee *et al.*[16] derived an algorithm of $O(n^2 \log n \log \log n)$ time.

Li and Shen[17] studied the same problem as our interval coverage problem except that their barrier is not a set of points but a single line segment on ℓ. They proposed an $O(n^3 \log n)$ time algorithm. The algorithm was later improved to $O(n^2 \log n \log \log n)$ time by Li and Wang.[18] Li and Wang[18] also studied a more general problem setting where the barrier is a set of m disjoint line segments on ℓ (and the sensors are still in the plane and are required to move to ℓ); they gave an $O(n^2 \log n \log \log n + nm \log m)$ time algorithm. Further, for the 1D case where all sensors are initially on ℓ, the algorithm of Li and Wang[18] solves the problem in $O(m \log m + n \log n \log m)$ time. These results are for the case where sensors have the same range; if sensors have different ranges, even the 1D problem is NP-hard by a simple reduction from the Partition Problem as in Ref. 6.

The min-sum version of the line-constrained barrier coverage was also studied in the literature where sensors are given on ℓ and a barrier segment is also on ℓ, and the goal is to move sensors to cover the barrier segment such that the total sum of the moving distances of all sensors is minimized. If sensors have different ranges, the problem is NP-hard.[7] Otherwise, Czyzowicz *et al.*[7] solved the problem in $O(n^2)$ time. Later Andrews and Wang[1] proposed a faster algorithm of $O(n \log n)$ time.

A circular barrier coverage problem was also considered, where the barrier is a circle and sensors are initially located inside the circle and the goal is to move all sensors to the circle to form a regular n-gon (to form a coverage) so that the maximum moving distance of all sensors is minimized. Bhattacharya[2] first gave an algorithm of $O(n^{3.5} \log n)$ time. An improved algorithm of $O(n \log^3 n)$ time was later derived by Chen *et al.*[4]

There are also other variations of the barrier coverage problem, e.g., see Refs. 8, 9, 20 and 21.

1.2. *Our approach*

We first discuss the mobile disk coverage problem. Let λ^* denote the optimal moving cost, i.e., the maximum moving cost of all sensors in an optimal solution. In both the unweighted and weighted cases, we first consider the *decision problem*: Given any value λ, determine whether $\lambda \geq \lambda^*$.

For the unweighted case, a critical property is an *order-preserving property*: There exists an optimal solution in which the order of the sensors are consistent with their order in the input. Due to the property, we can solve the decision problem in linear time by a simple greedy algorithm (after all barrier points and all sensors

are sorted). Next, we use the decision algorithm to compute λ^*. To this end, we define $2m$ arrays of size n each and we show that λ^* must be an element of one of the arrays. To search λ^* in these arrays in an efficient way, we form these arrays implicitly. A helpful observation is that each of these arrays is sorted. Consequently, by using our decision algorithm, we apply a *sorted matrix searching* technique[10–12] (or a simpler implementation called *binary search on sorted arrays* in Ref. 5) to find λ^* in these arrays in $O((n+m)\log(n+m))$ time.

For the weighted case, unfortunately the order-preserving property does not hold anymore. In fact, the major difficulty is to find the correct order for sensors in an optimal solution. This is also the case for solving the decision problem. So we have to use a different approach to solve the decision problem. The runtime of the algorithm is $O((n+m)\log(n+m))$. To compute the optimal cost λ^*, we implicitly form $2n$ arrays of size m each such that λ^* is one of the array elements. To apply the sorted matrix searching technique, we manage to find a way to order the array elements implicitly so that the arrays are still sorted. Then, with the decision algorithm, the value λ^* can be found in $O(m\log m + n\log^2 n)$ time.

For the mobile interval coverage problem, we solve the weighted cases directly (without having a faster algorithm for the unweighted case). As above, we also solve the decision problem first, and then form sorted arrays and apply the sorted array searching technique. To solve the decision algorithm, we use an algorithm similar to the weighted case of the above mobile disk coverage problem, but with a simpler and slightly faster implementation. The runtime of our decision algorithm is $O(m + n\log n)$ after $O((n+m)\log(n+m))$ time preprocessing for sorting all sensors and barrier points. The time of the overall algorithm (for computing the optimal value λ^*) is $O(m\log m + n\log^2 n)$.

Outline. The rest of the paper is organized as follows. We define notation in Sec. 2. In Sec. 3, we present our algorithm for the unweighted case of the mobile disk coverage problem, while the weighted case is discussed in Sec. 4. The algorithm for the mobile interval coverage is described in Sec. 5. Section 6 concludes with some remarks on the 1D problem.

2. Preliminaries

For each problem we consider, we use λ^* to denote the optimal moving cost. Given any λ, the *decision problem* is to decide whether $\lambda \geq \lambda^*$, i.e., whether it is possible to move sensors to cover all barrier points so that the moving cost of each sensor is at most λ. If $\lambda \geq \lambda^*$, we say that λ is a *feasible value*. We use *feasibility test* to refer to the procedure for determining whether $\lambda \geq \lambda^*$. For differentiation, we refer to our original problem for computing λ^* as the *optimization problem*.

Without loss of generality, we assume that the line ℓ is the x-axis. Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of sensors (unless otherwise stated, the order is arbitrary).

For each s_i, we use (x_i, y_i) to denote its coordinate in the input. For differentiation, for each barrier point $b \in B$, we use (x_b, y_b) to denote its coordinate.

In each problem, we use a *configuration* to refer to a specification of where each sensor s_i is located. For example, in the input configuration, each sensor s_i is at (x_i, y_i).

For each sensor s, we use $D(s)$ to refer to its covering disk, i.e., the disk of radius r centered at s. For any disk D, we use ∂D to denote its boundary, which is a circle. The *left half-circle* of ∂D refers to the portion of ∂D to the left of the vertical line through the center of D; the *right half-circle* is defined similarly.

For the mobile disk coverage problem, for simplicity of discussion, we assume that all barrier points are above or on ℓ since if a barrier point is below ℓ, then we can use its symmetric point about ℓ to replace it and that does not affect the solution of the problem.

For any point p on ℓ, for convenience, sometimes we also use p to refer to its x-coordinate. For example, for two points p and q on ℓ, $p \leq q$ means that p is to the left of q (including the case where p and q are coincident) and $p < q$ means that p is *strictly* to the left of q.

For each problem, for ease of exposition, we assume that it is always possible to cover all barrier points by moving sensors (i.e., the covering range r is big enough). Our algorithm can actually determine whether the assumption is true or not. This implies that in the mobile disk coverage problem, for each barrier point b, $y_b \leq r$ must hold since otherwise no sensor on ℓ can cover b. Also, for each problem we assume that $\lambda^* > 0$, i.e., one has to move at least one sensor in order to form a coverage for all barrier points. Note that whether $\lambda^* = 0$ can be easily determined in $O(n + m)\log(n + m)$ time for each problem (which does not affect the time complexity of the overall algorithm asymptotically).

For a barrier point b and the covering disk $D(s)$ of a sensor s, we say that $D(s)$ is *strictly to the left* (*resp.*, *right*) of b if $D(s)$ does not cover b and the x-coordinate of s is smaller (resp., larger) than that of b.

3. The Mobile Disk Coverage Problem: The Unweighted Case

In this section, we consider the unweighted case of the mobile disk coverage problem. In this problem, all sensors of S are on the line ℓ while each barrier of B can be anywhere in the plane.

We first present an algorithm to solve the decision algorithm. Consider a value λ. If $\lambda \geq \lambda^*$, we use a *feasible solution* to refer to a configuration in which all barrier points are covered and the moving cost of each sensor is no more than λ. As all sensors have the same range, it is not difficult to see that *the order-preserving property* in the following observation holds.

Observation 1 (The order-preserving property). *If $\lambda \geq \lambda^*$, then there exists a feasible solution in which the order of sensors is the same as in the input.*

Due to the order-preserving property, we can solve the decision problem by a simple greedy algorithm in linear time (after sensors and barrier points are sorted).

Lemma 1. *After $O(n \log n + m \log m)$ time preprocessing, given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O(n+m)$ time.*

Proof. In the preprocessing, we sort all sensors of S from left to right on ℓ; let $S = \{s_1, s_2, \ldots, s_n\}$ be the sorted list. We also sort all barrier points of B by their x-coordinates from left to right; let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list. Given any λ, in what follows we describe our $O(n+m)$ time algorithm for deciding whether $\lambda \geq \lambda^*$, which is based on the greedy strategy.

We first move each sensor rightwards on ℓ by distance λ and we use C_0 to refer to the configuration, i.e., in C_0, the location of each s_i is $x_i + \lambda$. Then, during the algorithm, each sensor will not be allowed to move rightwards anymore but can move leftwards by 2λ.

Starting from $i = 1$ and $j = 1$, we process sensors s_i and barrier points b_j incrementally. We first check whether b_j is covered by s_i. If yes, we increase j by one (if $j = m$ before the increase, then all barrier points are covered and we have found a feasible solution; in this case, we can stop the algorithm and report that λ is a feasible value, i.e., $\lambda \geq \lambda^*$). Otherwise, either b_j is to the right of the covering disk $D(s_i)$ of s_i or b_j is to the left of $D(s_i)$. In the former case, we increase i by one and proceed as above (if $i = n$ before the increase, then we can stop the algorithm and report that λ is not a feasible value, i.e., $\lambda < \lambda^*$). In the latter case, we check whether it is possible to move s_i leftwards by distance at most 2λ to cover b_j. If not, then we can stop the algorithm and report that λ is not a feasible value. Otherwise, we move s_i leftwards until b_j is covered (i.e., b_j is on the left half-circle of $\partial D(s_i)$); we then increase j by one and proceed as above (if $j = m$ before the increase, then all barrier points are covered and thus we can stop the algorithm and report that λ is a feasible value). This finishes the description of the algorithm.

The correctness of the algorithm is based on the order-preserving property. It is not difficult to see that the running time of the algorithm is $O(n+m)$. □

We next tackle the optimization problem for computing λ^*, by making use of our decision algorithm in Lemma 1 as a subroutine. For this, we have the following lemma.

Lemma 2. *λ^* is equal to $x_i - \sqrt{r^2 - y_b^2} - x_b$ or $x_b - \sqrt{r^2 - y_b^2} - x_i$ for a sensor s_i and a barrier point b.*

Proof. Consider an optimal solution OPT, where λ^* is the maximum moving distance of all sensors. Then, λ^* is equal to the moving distance of a sensor s_i. For ease of discussion, we assume that the moving distances of all other sensors are strictly smaller than λ^* (otherwise, we could apply the following analysis to all these sensors). Let x_i' be the position of s_i in OPT. If $x_i' < x_i$, then s_i has been

moved leftwards. In this case, there must be a barrier point b on the left half-circle of $\partial D(s_i)$ since otherwise we could move $D(s_i)$ rightwards slightly so that $D(s_i)$ still covers the same set of barrier points as before but the moving distance of s_i is strictly smaller than λ^*, a contradiction to the definition of λ^*. Thus, we have $x'_i = \sqrt{r^2 - y_b^2} + x_b$. Hence, $\lambda^* = x_i - x'_i = x_i - \sqrt{r^2 - y_b^2} - x_b$. If $x'_i > x_i$, then by similar analysis as above, we can show that $\lambda^* = x_b - \sqrt{r^2 - y_b^2} - x_i$. □

We sort all sensors of S from left to right on ℓ; let $S = \{s_1, s_2, \ldots, s_n\}$ be the sorted list. For each barrier point $b \in B$, we define two arrays $A_b[1 \cdots n]$ and $A'_b[1 \cdots n]$ of size n each as follows: For each $i \in [1, n]$, define $A_b[i] = x_i - \sqrt{r^2 - y_b^2} - x_b$ and $A'_b[i] = x_b - \sqrt{r^2 - y_b^2} - x_i$. According to Lemma 2, λ^* is an element in one of the $2m$ arrays A_b and A'_b for all $b \in B$. We next find λ^* in these arrays. Computing these arrays explicitly will take $\Omega(nm)$ time. Below, we present a near linear time algorithm without computing these arrays explicitly. Indeed, given an index $i \in [1, n]$ and a barrier point $b \in B$, we can obtain the values $A_b[i]$ and $A'_b[i]$ in constant time.

An easy observation is that elements of the array A_b are sorted in ascending order and elements of A'_b are sorted in descending order. Therefore, we are searching λ^* in $2m$ sorted arrays of size n each. Note that λ^* is actually the smallest feasible value in these $2m$ arrays. We can use the sorted matrix searching techniques[10–12] (or a simpler implementation, called *binary search on sorted arrays*, in Ref. 5) to search sorted arrays with the following lemma.

Lemma 3 (Refs. 5 and 10–12). *Suppose we have a set of M sorted arrays of size at most N each such that each array element can be evaluated in $O(1)$ time (i.e., given the index of an array, the element of the array can be obtained in $O(1)$ time). Then, the smallest feasible value in these arrays can be computed by $O(\log(N+M))$ feasibility tests and the total time of the algorithm excluding the feasibility tests is $O(M \log N)$.*

Applying Lemma 3 and using our decision algorithm in Lemma 1, λ^* can be found in $O((n + m)\log(n + m))$ time. We summarize our result in the following theorem.

Theorem 1. *Given a set of m barrier points in the plane and a set of n sensors on a line ℓ, the problem of moving sensors on ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O((n + m)\log(n + m))$ time.*

4. The Mobile Disk Coverage Problem: The Weighted Case

In this section, we solve the weighted case of the mobile disk coverage problem. Here also, we start with the decision problem and later solve the optimization problem by applying sorted array searching techniques in Lemma 3. In the weighted case, each sensor s_i is associated with a weight $w_i > 0$.

4.1. *The decision problem*

Given any λ, the problem is to decide whether $\lambda \geq \lambda^*$. Although our algorithm is similar in spirit to those in the previous work,[3,16,18] our algorithm is for a more general problem setting in that the barrier points are in the plane while the barriers in all previous work[3,16,18] are on ℓ. In the following, we first describe our algorithm, and then prove its correctness; finally, we will discuss how to efficiently implement the algorithm in $O((n+m)\log(n+m))$ time.

4.1.1. *The algorithm description*

For each sensor s_i, define $x_i^l = x_i - \lambda/w_i$ and $x_i^r = x_i + \lambda/w_i$, i.e., x_i^l is the leftmost location on ℓ where s_i can move to and x_i^r is the rightmost location on ℓ where s_i can move to with respect to λ. We call x_i^l (resp., x_i^r) the *leftmost* (*resp.*, *rightmost*) λ*-reachable location*.

For each barrier point b, we use $c(b)$ to denote the center of the circle of radius r whose center is on ℓ and whose left half-circle contains b, i.e., $c(b) = x_b + \sqrt{r^2 - y_b^2}$. We sort all barrier points $b \in B$ in the order of the values $c(b)$. Alternatively, it is also the order of the barrier points of B encountered by sweeping a left half-circle centered at ℓ from left to right. Let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list.

Initially, we move each sensor s_i to x_i^r and thus s_i will not be allowed to move rightwards anymore but can move leftwards by $2\lambda/w_i$. Let C_0 denote the resulting configuration. If $\lambda \geq \lambda^*$, our algorithm will find a subset of sensors with their new locations such that all barrier points are covered and the maximum moving cost of each sensor is at most λ (sensors not in the subset are still in their positions of C_0).

Consider the i-th iteration of the algorithm (initially, $i = 1$). Let C_{i-1} be the configuration right before the iteration. Our algorithm maintains the following invariants.

(1) A subset of sensors $S_{i-1} = \{s_{g_1}, \ldots, s_{g_{i-1}}\}$ has been computed, where g_j is the index of the sensor s_{g_j} for each $j \in [1, i-1]$.
(2) In C_{i-1}, each sensor s_k of S_{i-1} is at a location, denoted by x'_k, which may not be equal to x_k^r, while sensors of $S \backslash S_{i-1}$ are still in their locations of C_0 (i.e., each sensor of $S \backslash S_{i-1}$ is at its rightmost λ-reachable location).
(3) An index h_{i-1} of a barrier point is maintained such that in the configuration C_{i-1}, the barrier point $b_{h_{i-1}}$ is not covered by any sensor of S_{i-1} while b_k is covered by a sensor in S_{i-1} for each $k < h_{i-1}$ (note that it is possible that b_k for some $k > h_{i-1}$ is also covered by a sensor in S_{i-1}, which cannot happen in the problem settings of the previous work;[3,16,18] this case makes our problem more challenging to solve).
(4) The locations of the sensors $s_{g_1}, s_{g_2}, \ldots, s_{g_{i-1}}$ in C_{i-1} are sorted from left to right on ℓ.
(5) The barrier point $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$ of $s_{g_{i-1}}$ if $S_{i-1} \neq \emptyset$.

Initially when $i = 1$, we have $S_0 = \emptyset$ and we set $h_0 = 1$; thus, all algorithm invariants trivially hold. The i-th iteration of the algorithm finds a sensor from $S \backslash S_{i-1}$, denoted by s_{g_i}, and move it to a new location $x'_{g_i} \in [x^l_{g_i}, x^r_{g_i}]$ to obtain a new configuration C_i with $S_i = S_{i-1} \cup \{s_{g_i}\}$. The details of the i-th iteration of the algorithm are described below.

Define S_{i1} to be the set of sensors that cover the barrier point $b_{h_{i-1}}$ in the configuration C_{i-1}. According to our algorithm invariants, $b_{h_{i-1}}$ is not covered by any sensor in S_{i-1}. Hence, $S_{i1} \subseteq S \backslash S_{i-1}$.

If $S_{i1} \neq \emptyset$, we pick an arbitrary sensor from S_{i1} as s_{g_i} and set $x'_{g_i} = x^r_{g_i}$ (i.e., the sensor does not move from its position in C_{i-1}); thus $C_i = C_{i-1}$. We set $h_i = k+1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = m+1$, all barrier points b_j for all $j \in [h_{i-1}, n]$ are covered, and thus we can stop the algorithm and report $\lambda \geq \lambda^*$.

Lemma 4. *All algorithm invariants hold.*

Proof. We go through every invariant. Invariant (1) trivially holds. Invariant (2) holds because $C_i = C_{i-1}$. Invariant (3) follows immediately from how our algorithm computes h_i. Invariant (4) holds because s_{g_i} covers $b_{h_{i-1}}$ in C_i. For Invariant (5), it suffices to show that $s_{g_{i-1}}$ is to the left of the s_{g_i} in C_i. Indeed, according to Invariant (6) in C_{i-1}, $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$. Since $b_{h_{i-1}}$ is covered by s_{g_i} in C_i, we obtain that $s_{g_{i-1}}$ must be to the left of s_{g_i} in C_i. For Invariant (6), since the sensor s_{g_i} covers $b_{h_{i-1}}$ but does not cover b_{h_i} and $h_{i-1} < h_i$, according to the definition of the indices of the barrier points, we can obtain that b_{h_i} must be strictly to the right of the covering disk $D(s_{g_i})$ of s_{g_i} (e.g., see Fig. 1). This proves Invariant (6). □

If $S_{i1} = \emptyset$, we define $S_{i2} = \{s_k \,|\, x^l_k \leq c(b_{h_{i-1}}) < x^r_k, s_k \in S \backslash S_{i-1}\}$, i.e., the set of sensors s_k that do not cover $b_{h_{i-1}}$ in C_{i-1} but can be moved leftwards to cover $b_{h_{i-1}}$; e.g., see Fig. 2. Note that each sensor of S_{i2} is currently at its rightmost λ-reachable location in C_{i-1}.

If $S_{i2} \neq \emptyset$, then among all sensors of S_{i2}, we choose the leftmost one (with respect to their positions in C_{i-1}) as s_{g_i} and add it to S_{i-1} to obtain S_i. We move

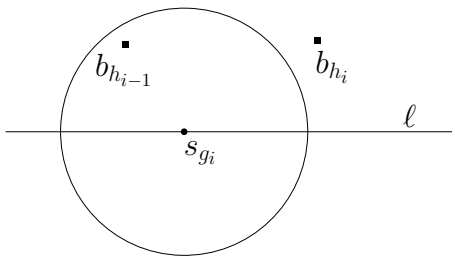

Fig. 1. Illustrating the Invariant (6) in the proof of Lemma 4: the circle is the boundary of $D(s_{g_i})$.

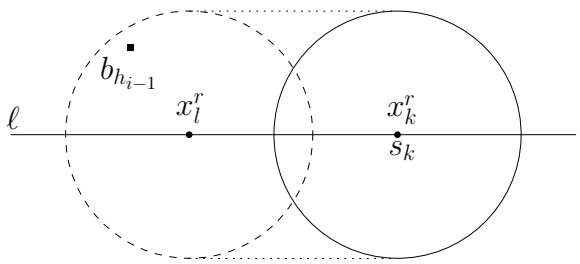

Fig. 2. Illustrating the definition of S_{i2}: The solid circle shows the position of s_k in C_{i-1}, i.e., at x_k^r, and the dashed circle shows its leftmost λ-reachable location, i.e., x_k^l.

s_{g_i} leftwards until $b_{h_{i-1}}$ is covered (i.e., it is on the left half-circle of ∂D_{g_i}); this obtains the configuration C_i. Next, we set $h_i = k + 1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = n + 1$, then all barrier points are covered and thus we can stop the algorithm and report $\lambda \geq \lambda^*$. Following the similar analysis as Lemma 4, we can show that all algorithm invariants hold.

If $S_{i2} = \emptyset$, then we terminate the algorithm and report that $\lambda < \lambda^*$.

In summary, if $S_{i1} = S_{i2} = \emptyset$, then the algorithm will terminate and report $\lambda < \lambda^*$. Otherwise, a sensor s_{g_i} is found from either S_{i1} (if it is not empty) or S_{i2} and added to S_{i-1} to obtain S_i. In either case, $h_i = k + 1$, where k is the largest index in $[h_{i-1}, n]$ such that barrier points b_j for all $j \in [h_{i-1}, k]$ are covered by sensors of S_i. If $h_i = m + 1$, then the algorithm will terminate and report $\lambda \geq \lambda^*$; otherwise, the algorithm will proceed to the next iteration $i + 1$ and all algorithm invariants hold. As there are m barrier points and a new barrier point is covered in each iteration, the algorithm has at most m iterations. On the other hand, as there are n sensors and each iteration finds a new sensor to form S_i, the algorithm has at most n iterations. Hence, the algorithm will stop in $\min\{n, m\}$ iterations.

4.1.2. *The algorithm correctness*

We now prove the correctness of the algorithm. The high-level idea of the proof is similar to the previous work,[3,16,18] although the details are quite different because in our problem barrier points are in the plane while the barriers in the previous work[3,16,18] are all on ℓ.

Suppose the algorithm reports $\lambda \geq \lambda^*$, say, in the i-th iteration of the algorithm. Then, according to our algorithm, the configuration C_i is a feasible solution. Thus, it suffices to show that if the algorithm reports $\lambda < \lambda^*$, then no feasible solution exists.

For any index $i \in [0, m]$ for the barrier points, we say that $[0, i]$ is a *prefix interval* of $[0, m]$. For convenience, depending on the context, we may also use $[0, i]$ to represent the subset of barrier points b_j for all $j \in [0, i]$ (the subset is $\emptyset$ if $i = 0$). For example, we say that the interval $[0, i]$ is *covered* by a set of sensors if all barrier

points b_j, $0 < j \leq i$, are covered by the set of sensors. We say that another prefix interval $[0, i']$ is *larger than* $[0, i]$ if $i' > i$.

Lemma 5. *Consider the configuration C_i produced in the i-th iteration of our algorithm with $i \geq 1$. Suppose S'_i is the set of sensors of S whose covering disks are strictly to the left of b_{h_i} in C_i. Then, $[0, h_i - 1]$ is the largest possible prefix interval that can be covered by sensors of S'_i with respect to λ (i.e., the moving cost of each sensor of S'_i is at most λ).*

Before proving Lemma 5, we use it to prove the correctness of our algorithm, i.e., we prove that if the algorithm reports $\lambda < \lambda^*$, then no feasible solution exists.

Suppose our algorithm reports $\lambda < \lambda^*$ in the i-th iteration. Then, according to our algorithm, $b_{h_{i-1}}$ is not covered by any sensor in C_{i-1} and $S_{i1} = S_{i2} = \emptyset$. By Lemma 5 (replacing the index i in the lemma by $i - 1$), $[0, h_{i-1} - 1]$ is the largest prefix interval that can be covered by sensors of S'_{i-1}. According to our algorithm invariants, the covering disk of each sensor of S_{i-1} is strictly to the left of $b_{h_{i-1}}$ in C_{i-1}. Hence, S_{i-1} is a subset of S'_{i-1}. Since both S_{i1} and S_{i2} are empty in C_{i-1}, no sensor in $S \backslash S'_{i-1}$ can cover the barrier point $b_{h_{i-1}}$. Therefore, it is not possible to cover all barrier points in the interval $[0, h_{i-1}]$ using the sensors of S (with respect to the maximum moving cost λ). This implies that no feasible solution exists.

4.1.3. *The proof of Lemma* 5

We now prove Lemma 5. We follow the notation in Lemma 5. Note that according to our algorithm invariants, $S_i = \{s_{g_1}, s_{g_2}, \ldots, s_{g_i}\}$ is a subset of S'_i.

We first prove the following lemma and then use the lemma to prove Lemma 5.

Lemma 6. *If C is a configuration in which a prefix interval $[0, t]$ is covered by the sensors of S'_i, then there also exists a configuration C^* in which $[0, t]$ is covered and the location of each sensor s_{g_j} of S_i in C^* is the same as its location in C_i.*

Proof. We prove the lemma by induction. We assume that the lemma statement holds for $k - 1$, $1 \leq k \leq i$, i.e., there exists a configuration C' in which the interval $[0, t]$ is covered and the location of each sensor s_{g_j} of S_i with $1 \leq j \leq k - 1$ in C^* is the same as its location in C_i (i.e., x'_{g_j}). The assumption trivially holds when $k = 1$. Below we show that the lemma statement holds for general k.

Our goal is to find a configuration C'' in which barrier points of the interval $[0, t]$ are also covered and the location of each sensor s_{g_j} of S_i with $1 \leq j \leq k$ in C'' is x'_{g_j}. We refer to such a configuration that satisfies the above condition as a *satisfying configuration.*

According to our algorithm, in the configuration C_k, s_{g_j} is at x'_{g_j} for all $1 \leq j \leq k$, and the interval $[0, h_k - 1]$ is covered by sensors of S_k. Hence, if $t \leq h_k - 1$, then we can simply let $C'' = C_k$, which is a satisfying configuration. In the following, we assume that $t \geq h_k$. Let $x_{C'}(s_{g_k})$ be the location of s_{g_k} in the configuration C'.

If $x_{C'}(s_{g_k}) = x'_{g_k}$, then let $C'' = C'$, which is a satisfying configuration. In what follows, we assume that $x_{C'}(s_{g_k}) \neq x'_{g_k}$. According to our algorithm, s_{g_k} is either from S_{k1} or from S_{k2}. We discuss the two cases below.

The case $s_{g_k} \in S_{k1}$. If s_{g_k} is from S_{k1}, then according to our algorithm, $x'_{g_k} = x^r_{g_k}$. As $x_{C'}(s_{g_k}) \neq x'_{g_k}$, it must be that $x_{C'}(s_{g_k}) < x'_{g_k}$. Let C'' be the configuration obtained from C' by moving s_{g_k} from $x_{C'}(s_{g_k})$ rightwards to x'_{g_k}. In the following, we show that C'' is a satisfying configuration.

Indeed, in light of the induction hypothesis, the location of each sensor s_{g_j} of S_i with $1 \leq j \leq k$ in C'' is x'_{g_j} (i.e., the same as its location in C_i). Thus, it suffices to show that the interval $[0, t]$ is covered by sensors of S'_i in C''. Consider any barrier point b_l with $l \in [1, t]$.

— If $l \leq h_k - 1$, then according to our algorithm, b_l is covered by a sensor s in S_k in C_i. As $S_k \subseteq S_i \subseteq S'_i$, s is in S'_i. Further, since $s \in S_k$, its location position in C'' is the same as in C_i. Therefore, b_l is covered by s in C'' and thus b_l is covered by sensors of S'_i in C'' since $s \in S'_i$.

— If $l \geq h_k$, then depending on whether b_l is covered by a sensor of S_k in C_i, there are two subcases. If b_l is covered by a sensor of S_k in C_i, then following the same analysis as above, b_l is covered by sensors of S'_i in C''. Otherwise, since the locations of the sensors of S_{k-1} in C'' are the same as in C', b_l must be covered in C' by either s_{g_k} or a sensor in $S'_i \backslash S_k$.

  We claim that b_l cannot be covered by s_{g_k} in C'. Indeed, according to our algorithm invariants, the covering disk of s_{g_k} is strictly to the left of b_{h_k} in C_k. Since $x_{C'}(s_{g_k}) < x'_{g_k}$, i.e., the location of s_{g_k} in C' is strictly to the left of its location in C_k, the covering disk $D(s_{g_k})$ is also strictly to the left of b_{h_k} in C'. Since $l \geq h_k$, by our definition of the indices of the barrier points, b_l cannot be in $D(s_{g_k})$ in C'.

  The above claim implies that b_l is covered in C' by a sensor s of $S'_i \backslash S_k$. Since the location of s in C'' is the same as its location in C', s still covers b_l in C''. Therefore, b_l is covered by sensors of S'_i in C''.

This proves that C'' is a satisfying configuration.

The case $s_{g_k} \in S_{k2}$. If s_{g_k} is from S_{k2}, then according to our algorithm, $S_{k1} = \emptyset$ and s_{g_k} is the leftmost sensor of S_{k2} in the configuration C_{k-1} and x'_{g_k} is the rightmost location for s_{g_k} to cover $b_{h_{k-1}}$ (i.e., $b_{h_{k-1}}$ is on the left half-circle of $\partial D(s_{g_k})$). If $x_{C'}(s_{g_k}) < x'_{g_k}$, then we can use the same argument as the above case to obtain a satisfying configuration. In the following, we assume that $x_{C'}(s_{g_k}) > x'_{g_k}$. This also implies that s_{g_k} does not cover $b_{h_{k-1}}$ in C'. Since $t \geq h_k > h_{k-1}$, there must be a sensor s_a that covers the barrier point $b_{h_{k-1}}$ in C'. Also, because $S_{k1} = \emptyset$ and the positions of the sensors s_{g_j} for all $1 \leq j \leq k-1$ in C' are the same as in C_{k-1}, s_a must be from S_{k2}. As s_{g_k} is the leftmost sensor of S_{k2} in C_{k-1}, it must hold that $x^r_{g_k} \leq x^r_a$.

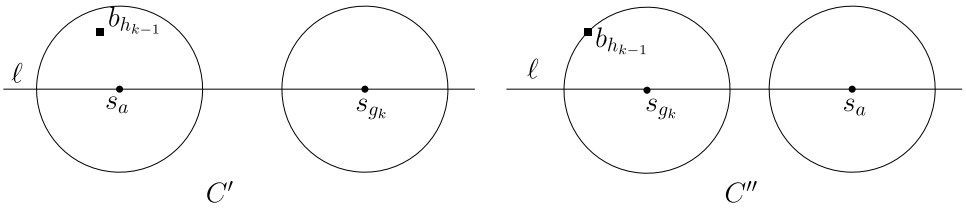

Fig. 3. Illustrating the sensors s_a and s_{g_k} in the two configurations C' and C''.

Let C'' be the configuration obtained from C' by moving s_a to $x_{C'}(s_{g_k})$ and moving s_{g_k} to x'_{g_k}, i.e., s_a moves to the position of s_{g_k} in C' and s_{g_k} moves to its position in C_k (e.g., see Fig. 3). Below we argue that C'' is a satisfying configuration. For this, we will show the following: (1) The interval $[0, t]$ is still covered by sensors of S'_i in C''; (2) the moving cost of s_a is no more than λ (note that since the position of s_{g_k} in C'' is the same as its position in C_k, we know that its moving cost in C'' is no more than λ; other sensors do not change locations from C' to C'').

We first prove the above (1). Since the locations of the senors of s_{g_j} for all $j \in [1, k]$ in C'' are the same as their locations in C_k, these sensors together cover all barrier points of the interval $[0, h_k - 1]$. Consider any other barrier point b_l with $l \in [h_k, t]$. To prove (1), it suffices to show that b_l is covered by a sensor of S'_i in C''. Recall that b_l is covered by a sensor of S'_i in C'; let s be such a sensor.

(1) If s is s_{g_j} for any $j \leq k - 1$, since s has the same location in C' and C'', s also covers b_l in C''.
(2) If s is s_a, then we claim that b_l must be covered by s_{g_k} in C''. Indeed, recall that $b_{h_{k-1}}$ is on the left half-circle of the covering disk of s_{g_k} when s_{g_k} is at x'_{g_k} in C'' (and also in C_k). Since $b_{h_{k-1}}$ is covered by s_a in C', we obtain that $x_{C'}(s_a) \leq x'_{g_k}$, where $x_{C'}(s_a)$ is the location of s_a in C' (e.g., see Fig. 4). Since s_a also covers b_l and $l \geq h_k > h_{k-1}$, if we move a disk D of radius r centered at $x_{C'}(s_a)$ rightwards until x'_{g_k}, D starts at the covering disk of s_a in C' and stops at the covering disk of s_{g_k} in C''. Hence, in the beginning of the movement of D, it covers b_l, and at the end of the movement, $b_{h_{k-1}}$ is on the left half-circle of

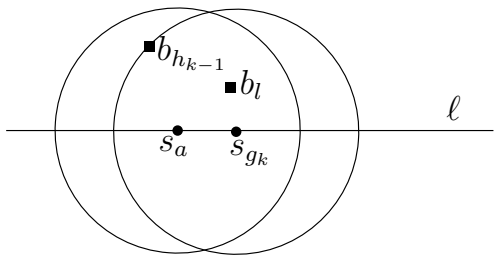

Fig. 4. Illustrating the relative positions of s_a, s_{g_k}, $b_{h_{k-1}}$, and b_l: the locations of s_a and s_{g_k} are $x_{C'}(s_a)$ and x'_{g_k}, respectively.

∂D. Since $l > h_{k-1}$, during the above movement of D, its left half-circle cannot encounter the barrier point b_l. This implies that b_l is always inside D during the movement of D. This further implies that b_l is covered by s_{g_k} in C''.

(3) If s is s_{g_k}, then since s_a moves to the position of s_{g_k} in C'', s_a also covers b_l in C''.

(4) If s is not a sensor in the above three cases, then s does not change its location from C' to C''. Hence, s still covers b_l in C''.

In summary, the barrier point b_l is still covered by sensors of S'_i in C''.

We proceed to prove the above (2), i.e., the moving cost of s_a is no more than λ in C''. Let $x_{C''}(s_a)$ denote the location of s_a in C''. It suffices to show that $x_{C''}(s_a) \in [x^l_a, x^r_a]$.

According to our definition of C'', $x_{C''}(s_a) = x'_{g_k}$. Recall that $x^r_{g_k} \leq x^r_a$. Since $x'_{g_k} \leq x^r_{g_r}$, we obtain that $x_{C''}(s_a) = x'_{g_k} \leq x^r_{g_r} \leq x^r_a$.

On the other hand, recall that $x'_{g_k} < x_{C'}(s_{g_k}) = x_{C''}(s_a)$. Also, $x_{C'}(s_a) \geq x^l_a$, where $x_{C'}(s_a)$ is the location of s_a in C'. Since $b_{h_{k-1}}$ is on the left half-circle of $\partial D(s_{g_k})$ when s_{g_k} is at x'_{g_k} and $b_{h_{k-1}}$ is covered by s_a in C' when s_a is at $x_{C'}(s_a)$, we obtain that $x_{C'}(s_a) \leq x'_{g_k}$. Therefore, we can derive $x^l_a \leq x_{C'}(s_a) \leq x'_{g_k} < x_{C'}(s_{g_k}) = x_{C''}(s_a)$.

This proves that $x_{C''}(s_a) \in [x^l_a, x^r_a]$. Hence, C'' is a satisfying configuration. □

Proving Lemma 5. In what follows, we use Lemma 6 to prove Lemma 5.

Let $[0, t]$ be the largest prefix interval of sensors that can be covered by sensors of S'_i (with respect to the maximum moving cost λ). By Lemma 6, there exists a configuration C^* in which $[0, t]$ is still covered and the location of each sensor s_{g_j} of S_i in C^* is the same as its location in C_i, i.e., x'_{g_j}.

Consider any sensor $s_k \in S'_i \backslash S_i$. According to our algorithm, s_k is at x^r_k. By the definition of S'_i, the covering disk $D(s_k)$ is strictly to the left of b_{h_i} in C_i. Hence, s_k cannot be used to cover b_{h_i} in any configuration (with respect to λ), in particular, in C^*. On the other hand, according to our algorithm, all barrier points of the interval $[0, h_i - 1]$ are covered by sensors of S_i in C_i. As the sensors of S_i have the same locations in C^* as in C_i, all barrier points of $[0, h_i - 1]$ are covered by sensors of S_i in C^*. Combining the above, we can conclude that $[0, h_i - 1]$ is the largest prefix interval that can be covered by sensors of S'_i in C^*, i.e., $t = h_i - 1$. This proves Lemma 5.

4.1.4. *The algorithm implementation*

We now provide an efficient way to implement the algorithm in $O((n + m) \log(n + m))$ time. For differentiation, we use "algorithm implementation" to refer to the algorithm we will discuss below and use "algorithm description" to refer to the algorithm we described in Sec. 4.1.1.

We sweep a point p on ℓ from left to right. The event point set is $E = \{c(b) \mid b \in B\} \cup \{x^l_i, x^r_i \mid s_i \in S\}$. To sort E, we sort $\{c(b) \mid b \in B\}$ in the preprocessing. Then,

we only need to sort x_i^r and x_i^l for all sensors $s_i \in S$ in $O(n \log n)$ time, after which, we merge this sorted list with that of $\{c(b) \mid b \in B\}$. As such, we can obtain the sorted list of the event set E in $O(m + n \log n)$ time. Using the sorted list E as a guide, we sweep p on ℓ from left to right. When p encounters a point x_k^l for some sensor s_k, we insert s_k to a balanced binary search tree T in which the sensors s_k are ordered by their values x_k^r. As will be shown later, the tree T is used to maintain the set S_{i2}. When p encounters a point x_k^r, we remove s_k from T and store s_k at a variable s^* (if s^* already stores a sensor, we simply update s^* to s_k). Our algorithm implementation maintains the following invariant: the sensor s_k stored in s^* and all sensors of T are at their positions in C_0.

Now consider the case where p encounters $c(b_j)$ for some barrier point b_j. We assume that j is equal to h_{i-1} for some i as defined in the algorithm description. The assumption is true initially when $j = 1$ and $i = 1$. This means that we are at the beginning of the i-th iteration in the algorithm description. We first need to check whether $S_{i1} = \emptyset$. To this end, we have the following Lemma 7. But before giving Lemma 7, we prove the following observation, which will be used in the proofs of Lemma 7 and other lemmas.

Observation 2. *Consider a barrier point b and two sensors s and s'. Suppose the followings hold (e.g., see Fig. 5): (1) s' is to the right of s; (2) s covers b; (3) b is to the right of the left half-circle of $\partial D(s')$. Then, s' also covers b.*

Proof. Assume to the contrary that s' does not cover b. Then, since b is to the right of the left half-circle of $\partial D(s')$, b must be strictly to the right of the right half-circle of $\partial D(s')$. Because s' is to the right of s, b must also be strictly to the right of the right half-circle of $\partial D(s)$. But this means that s does not cover b, a contradiction. □

Lemma 7. *If the sensor s_k stored in s^* covers b_j when s_k is at x_k^r, then $s_k \in S_{i1}$; otherwise (including the case where s^* does not store any sensor) $S_{i1} = \emptyset$.*

Proof. Suppose the sensor s_k stored in s^* covers b_j when s_k is at x_k^r. To prove the lemma, it suffices to show that if $S_{i1} \neq \emptyset$, then s_k must be S_{i1}. In the following, we

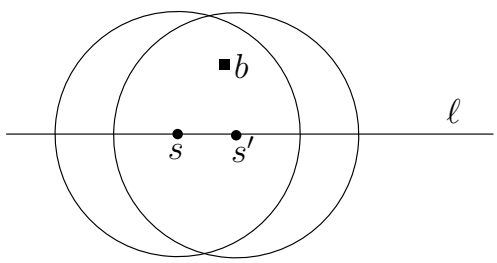

Fig. 5. Illustrating Observation 2.

assume that $S_{i1} \neq \emptyset$. Our goal is to prove that s_k is in S_{i1}. Since s_k is stored in s^*, according to our algorithm implementation invariant, s_k is at x_k^r. Hence, to prove $s_k \in S_{i1}$, by the definition of S_{i1}, it is sufficient to show that s_k covers b_j (when s_k is at x_k^r).

Let s_a be a sensor of S_{i1}. If s_a is s_k, then it is vacuously true that $s_k \in S_{i1}$. In what follows, we assume that s_a is not s_k. Because s_a is in S_{i1}, according to our algorithm description, s_a is at x_a^r and has never been moved during the algorithm, and further, s_a covers b_j. Since the sweeping point p is at $c(b_j)$, which is the rightmost position on ℓ for the center of a circle of radius r to cover b_j, p must have passed x_a^r. Therefore, according to our algorithm implementation, s_a had been stored in s^* before and later s^* got updated to s_k. This implies that s_k is to the right of s_a (and both of them are at their rightmost λ-reachable locations). Because p is now at $c(b_j)$, p has already passed x_k^r. Therefore, b_j is to the right of left half-circle of $\partial D(s_k)$. Since b_j is covered by s_a and s_k is to the right of s_a, by Observation 2, b_j must be covered by s_k. □

By Lemma 7, if s^* does not store any sensor or if the sensor stored at s^* does not cover b_j, then $S_{i1} = \emptyset$. Otherwise, the sensor stored at s^*, denoted by s_k, covers b_j and is in S_{i1}. Depending on whether $S_{i1} = \emptyset$, there are two cases to proceed.

The case $S_{i1} \neq \emptyset$. We first consider the case $S_{i1} \neq \emptyset$. In this case, according to our algorithm description, we can simply choose s_k as s_{g_i} and add it to S_{i-1} to obtain S_i. Next, we need to determine h_i, which is equal to $l+1$ with l as the largest index such that all barrier points $b_j, b_{j+1}, \ldots, b_l$ can be covered by sensors of S_i. To find l, we initialize $l = j$ and then keep sweeping p rightwards. If p encounters a point x_k^l or x_k^r, we process the event in the same way as before. If p encounters a point $c(b_{j'})$, we know that $j' = l + 1$. We need to determine whether $b_{j'}$ can be covered by sensors of S_i. For this, we have the following lemma.

Lemma 8. *$b_{j'}$ can be covered by sensors of S_i if and only if $b_{j'}$ can be covered by s_{g_i}.*

Proof. If $b_{j'}$ is covered by s_{g_i}, then it is vacuously true that $b_{j'}$ is covered by sensors of S_i because s_{g_i} is in S_i.

Now assume that $b_{j'}$ is covered by a sensor $s_{g_a} \in S_i$. We need to prove that s_{g_i} also covers $b_{j'}$. This is obviously true if $a = i$. We now assume $a \neq i$, implying that $a < i$. According our algorithm implementation, $b_{j'}$ is to the right of the left half-circle of $\partial D(s_k)$ and $s_{g_i} = s_k$. According to our algorithm invariants in the algorithm description, s_{g_a} is to the left of s_{g_i}. Since s_{g_a} covers $b_{j'}$, by Observation 2, s_{g_i} also covers $b_{j'}$. □

In light of Lemma 8, we check whether $b_{j'}$ is covered by s_{g_i}. If yes, we increment l by one and proceed as above (if $l = n$, then all barrier points are covered and we can stop the algorithm and report $\lambda \geq \lambda^*$). Otherwise, we set $h_i = j'$; in this

case, we have finished the i-th iteration of the algorithm and we then proceed to the $(i+1)$-th iteration.

The case $S_{i1} = \emptyset$. We now consider the case $S_{i1} = \emptyset$. In this case, we need to know whether $S_{i2} = \emptyset$, and if not, we need to find the leftmost sensor in S_{i2}. For this, we have the following lemma.

Lemma 9. *The sensors stored in the current tree T are exactly the sensors of S_{i2}.*

Proof. We prove the lemma by analyzing our algorithm implementation. Recall that the sweeping point p is now at $c(b_j)$ and $j = h_{i-1}$.

— Let s_a be a sensor of S_{i2}. We show that s_a is stored in T. Indeed, since s_a is in S_{i2}, by the definition of S_{i2}, we have $x_a^l \leq c(b_j) < x_a^r$. According to our algorithm implementation, when p encounters x_a^l, s_a is inserted to T and will not be removed from T until p counters x_a^r. Since p is at $c(b_j)$ right now and $c(b_j) < x_a^r$, s_a is still in T.
— Let s_a be a sensor stored in T. We show that s_a is in S_{i2}. Indeed, since s_a is in T, according to our algorithm implementation, p has already passed x_a^l but not encountered x_a^r yet. Since p is at $c(b_j)$ right now, we obtain that $x_a^l \leq c(b_j) < x_a^r$. Further, according to our algorithm implementation invariant, s_a has not been moved from its position in C_0, i.e., s_a is still at x_a^r. Therefore, s_a is in S_{i2}.

This proves the lemma. □

In light of Lemma 9, we can use T to find the leftmost sensor of T in $O(\log n)$ time; let s_k denote the sensor. We choose s_k as s_{g_i} and add it to S_{i-1} to obtain S_i. Then, we move s_k leftwards to $c(b_j)$, i.e., setting $x'_k = c(b_j)$, and remove s_k from T. We also remove both events x_k^l and x_k^r from the list E because we do not need to process these two events anymore.[b] Next, we need to determine h_i. This can be done using the same method as in the above case where $S_{i1} \neq \emptyset$ (i.e., keep sweeping p rightwards and making use of Lemma 8, which is still applicable here). After h_i is found, we finish the i-th iteration of the algorithm and begin the $(i+1)$-th iteration.

This finishes the description of the algorithm implementation. The proof of the following lemma analyzes the running time of the algorithm.

Lemma 10. *After $O(m \log m)$ time preprocessing, given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O(m + n \log n)$ time.*

Proof. First of all, it takes $O(m \log m)$ time to sort $\{c(b) \mid b \in B\}$ in the preprocessing. We next analyze the running time of our implementation for determining

[b]To implement each remove operation in constant time, we can store the list E by a doubly-linked list and associate each of the values x_a^l and x_a^r for all sensors $s_a \in S$ with a pointer pointing to its location in E.

whether $\lambda \geq \lambda^*$. In the beginning, computing the sorted list E takes $O(m+n\log n)$ time. There are $O(n+m)$ operations on E, each of which takes $O(1)$ time. The time we spent on the binary search tree T is bounded by $O(n\log n)$ as there are n sensors and each sensor can be inserted and removed from T at most once (also, there are at most n operations of "finding the leftmost sensor"). Therefore, the total time of the algorithm is $O(m+n\log n)$. □

4.2. *The optimization problem*

We now solve the optimization problem, i.e., computing λ^*, by using the algorithm of Lemma 10 as a subroutine. We begin with the following lemma.

Lemma 11. *λ^* is equal to $(x_i - \sqrt{r^2 - y_{b_j}^2} - x_{b_j})/w_i$ or $(x_{b_j} - \sqrt{r^2 - y_{b_j}^2} - x_i)/w_i$ for a sensor s_i and a barrier point b_j.*

Proof. The proof is almost the same as that of Lemma 2 except that we have to consider the weight in the last step of the proof. We briefly discuss it below.

Consider an optimal solution OPT, where λ^* is the maximum moving cost of all sensors. Then, λ^* is equal to the moving cost of some sensor s_i. Let x_i' be the x-coordinate of s_i in OPT. If $x_i' < x_i$, then s_i has been moved leftwards and there must be a barrier point b_j on the left-circle of $\partial D(s_i)$. Thus, we have $x_i' = \sqrt{r^2 - y_{b_j}^2} + x_{b_j}$. Hence, $\lambda^* = (x_i - x_i')/w_i = (x_i - \sqrt{r^2 - y_{b_j}^2} - x_{b_j})/w_i$. If $x_i' > x_i$, by similar analysis, we can show that $\lambda^* = (x_{b_j} - \sqrt{r^2 - y_{b_j}^2} - x_i)/w_i$. □

For each sensor s_i, we will define two sorted arrays $A_i[1\cdots m]$ and $B_i[1\cdots m]$ of size m each. Unlike the unweighted case where defining sorted arrays is relatively straightforward, here the definitions are quite subtle. We define the array A_i first, which consists of the values $(x_i - \sqrt{r^2 - y_{b_j}^2} - x_{b_j})/w_i$ for all $j = 1, \ldots, m$. For each $j \in [1, m]$, let $a_j = \sqrt{r^2 - y_{b_j}^2} + x_{b_j}$. We sort the values a_j for all $j = 1, \ldots, m$ in ascending order. For each $j \in [1, m]$, we let $\pi(j) = k$ if a_k ranks the j-th place in the above sorted list. Hence, $\pi(\cdot)$ is a permutation of the indices $1, 2, \ldots, m$; note that we can obtain $\pi(\cdot)$ in $O(m\log m)$ time. For each $j \in [1, m]$, we define $A_i[j] = (x_i - a_{\pi(j)})/w_i$. In light of the definition of $\pi(\cdot)$, A_i is a sorted array. Analogously, we can define a sorted array B_i for the m values $(x_{b_j} - \sqrt{r^2 - y_{b_j}^2} - x_i)/w_i$, $j = 1, \ldots, m$. Note that the permutation $\pi(\cdot)$ can be used to define A_i for all $i = 1, 2, \ldots, n$. Hence, in $O(n + m\log m)$ time, we can implicitly form $2n$ sorted arrays A_i and B_i for all $i = 1, 2, \ldots, n$, such that given any index j and any array A_i (resp., B_i), we can obtain the array element $A_i[j]$ (resp., $B_i[j]$) in $O(1)$ time. Also, Lemma 11 implies that λ^* is the smallest feasible value of all elements of these arrays. By applying Lemma 3 and using our decision algorithm in Lemma 10, λ^*

can be computed in $O(m \log m + (m + n \log n) \log(n + m))$ time, which is bounded by $O(m \log m + n \log^2 n)$.[c]

Theorem 2. *Given a set of m barrier points in the plane and a set of n weighted sensors on a line ℓ, the problem of moving sensors on ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O(m \log m + n \log^2 n)$ time.*

5. The Mobile Interval Coverage Problem

In this section, we consider the mobile interval coverage problem, where the barrier points are on the x-axis ℓ while the sensors can be anywhere in the plane. The problem is to move all sensors to ℓ to cover all barrier points so that the minimum moving cost of all sensors is minimized.

We first sort all barrier points from left to right on ℓ in $O(m \log m)$ time; let $B = \{b_1, b_2, \ldots, b_m\}$ be the sorted list. Recall that for each sensor $s_i \in S$, (x_i, y_i) is its coordinate. In the weighted case, each sensor s_i has a weight $w_i > 0$. In the following, we only give an algorithm for the weighted case because we do not have a faster algorithm for the unweighted case. Our goal is to compute the optimal moving cost λ^*. Note that since we require that all sensors finally move to ℓ, it must hold that $\lambda^* \geq \max_{1 \leq i \leq n} w_i \cdot y_i$.

We again first consider the decision problem: Given any λ, decide whether $\lambda \geq \lambda^*$. We present an algorithm of $O(m + n \log n)$ time (not including the time for sorting the barrier points) for the problem. Later we will solve the optimization problem (i.e., computing λ^*) using Lemma 3 and the decision algorithm.

5.1. *The decision problem*

Consider a value λ. We assume that $\lambda \geq \max_{1 \leq i \leq n} w_i \cdot y_i$ since otherwise it is impossible to move all sensors to ℓ (and thus we immediately report $\lambda < \lambda^*$). For each sensor s_i, define $x_i^r = x_i + \sqrt{(\lambda/w_i)^2 - y_i^2}$ and $x_i^l = x_i - \sqrt{(\lambda/w_i)^2 - y_i^2}$. We call x_i^r (resp., x_i^l) the *rightmost (resp., leftmost) λ-reachable location* of s_i.

At the outset, we move each sensor s_i to x_i^r on ℓ. Let C_0 denote the resulting configuration. The rest of the algorithm is similar to the one in Sec. 4.1. In fact, we can basically apply the same algorithm. But since the problem setting here is simpler (because all barrier points are now on ℓ), below we describe the algorithm in a simpler way (the running time is also slightly faster if m is significantly larger than n).

Consider the i-th iteration of the algorithm (initially $i = 1$). Let C_{i-1} denote the configuration right before the iteration. Our algorithm maintains the following

[c]To see this, first notice that $m \log m + (m + n \log n) \log(n + m) = O(m \log m + n \log n \log(n + m))$. Further, if $m \geq n^2$, then $m \log m + n \log n \log(n + m) = O(m \log m)$; otherwise, $\log(n + m) = \Theta(\log n)$ and thus $m \log m + n \log n \log(n + m) = O(m \log m + n \log^2 n)$.

invariants:

(1) A subset $S_{i-1} = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i-1)}\}$ of sensors has been computed.
(2) In C_{i-1}, each sensor s_k of S_{i-1} is at a location, denoted by x'_k, which may not be equal to x^r_k, while sensors of $S \backslash S_{i-1}$ are still in their locations of C_0.
(3) An index h_{i-1} of a barrier point is maintained such that in the configuration C_{i-1}, the barrier point $b_{h_{i-1}}$ is not covered by any sensor of S_{i-1} while b_k is covered by a sensor in S_{i-1} for each $k < h_{i-1}$.
(4) The locations of the sensors $s_{g_1}, s_{g_2}, \ldots, s_{g_{i-1}}$ in C_{i-1} are sorted from left to right on ℓ.
(5) The barrier point $b_{h_{i-1}}$ is strictly to the right of the covering disk $D(s_{g_{i-1}})$ of $s_{g_{i-1}}$ if $S_{i-1} \neq \emptyset$.

Initially when $i = 1$, we have $S_0 = \emptyset$ and set $h_0 = 1$; thus all algorithm invariants hold. The i-th iteration of the algorithm finds a sensor s_{g_i} from $S \backslash S_{i-1}$ and move it to a new location x'_{g_i}; we thus obtain a new configuration C_i with $S_i = S_{i-1} \cup \{s_{g_i}\}$. We briefly discuss algorithm below.

Define S_{i1} be the set of sensors that cover the barrier point $b_{h_{i-1}}$ in C_{i-1}. Again, due to our algorithm invariants, $S_{i1} \subseteq S \backslash S_{i-1}$.

If $S_{i1} \neq \emptyset$, we choose an arbitrary sensor in S_{i1} as s_{g_i} and set $x'_{g_i} = x^r_{g_i}$. Hence, $C_i = C_{i-1}$. Next, we set $h_i = k+1$, where k is the largest index such that all barrier points of $[h_{i-1}, k]$ are covered by S_i (it is easy to see that a barrier point b_l with $l \geq h_{i-1}$ is covered by S_i if and only if b_l is covered by s_{g_i}, i.e., Lemma 8 is still applicable). If $k = m$, then we stop the algorithm and report $\lambda \geq \lambda^*$.

If $S_{i1} = \emptyset$, we define S_{i2} as the set of sensors of $S \backslash S_{i-1}$ that do not cover $b_{h_{i-1}}$ in C_{i-1} but can be moved leftwards to cover $b_{h_{i-1}}$. If $S_{i2} \neq \emptyset$, we choose the leftmost sensor of S_{i2} as s_{g_i} and set $x'_{g_i} = x_b + r$ to obtain a new configuration C_i, where $b = b_{h_{i-1}}$. Next, we set h_i in the same way as above. If $S_{i2} = \emptyset$, then we terminate the algorithm and report $\lambda < \lambda^*$.

The algorithm will terminate in at most $\min\{m, n\}$ iterations. The correctness of the algorithm can be proved in a similar way as before.

To implement the algorithm, as in Sec. 4.1, we sort $\{c(b) \mid b \in B\}$ in the preprocessing, which takes $O(m \log m)$ time. Then, given any λ, we can implement the algorithm in $O(m + n \log n)$ time using essentially the same implementation as in Sec. 4.1. We briefly discuss it below.

We first compute x^r_i and x^l_i for each sensor $s_i \in S$, and sort all these $2n$ values in $O(n \log n)$ time. Since we already have the sorted list of $\{c(b) \mid b \in B\}$ in the preprocessing, by merging it with the sorted list of x^r_i and x^l_i for all sensors $s_i \in S$, we can obtain the sorted list of the event set $E = \{c(b) \mid b \in B\} \cup \{x^l_i, x^r_i \mid s_i \in S\}$ in additional $O(n+m)$ time. Using E, we run the same sweeping algorithm as before. We still use a binary search tree T to maintain the sensors of S_{i2} and use a variable s^* to store a sensor of S_{i1}. When p encounters x^l_k for a sensor s_k, we insert s_k to T. When p encounters x^r_k, we remove s_k from T and set s^* to s_k. When p encounters

a barrier point b_j, we determine the sensor s_{g_i} using the variable s^* and the tree T in the same way as before. As analyzed in the proof of Lemma 10, the total time of the algorithm is $O(m + n \log n)$.

Lemma 12. *After $O(m \log m)$ time preprocessing, given any λ, whether $\lambda \geq \lambda^*$ can be decided in $O(m + n \log n)$ time.*

5.2. *The optimization problem*

We now show how to compute λ^*. Using analysis similar to Lemmas 2 and 11, we can show that λ^* is equal to $\sqrt{(x_i - r - x_{b_j})^2 + y_i^2}/w_i$ or $\sqrt{(x_{b_j} - r - x_i)^2 + y_i^2}/w_i$. In the former case we have $x_i - r - x_{b_j} \geq 0$ while in the latter case $x_{b_j} - r - x_i \geq 0$ holds. As such, for each sensor s_i, we implicitly form two sorted arrays of at most m elements as follows. Recall that $B = \{b_1, b_2, \ldots, b_m\}$ is in ascending order by x-coordinate. Let j be smallest index such that $x_{b_j} - r - x_i \geq 0$. Then, we define an array $A_i[0 \cdots m - j]$ for the above latter case: $A_i[k] = \sqrt{(x_{b_{k+j}} - r - x_i)^2 + y_i^2}/w_i$, for each $0 \leq k \leq m - j$. It is easy to see that A_i is sorted. Similarly, we define another sorted array for the above former case. As such, λ^* must be in one of the $2n$ sorted arrays thus defined by all sensors. Then, as in Sec. 4.2, applying Lemma 3 with our decision algorithm in Lemma 12 can compute λ^* in $O(m \log m + n \log^2 n)$ time.

Theorem 3. *Given a set of m barrier points on a line ℓ and a set of n weighted sensors in the plane, the problem of moving sensors to ℓ to cover all barrier points such that the maximum moving cost of all sensors is minimized can be solved in $O(m \log m + n \log^2 n)$ time.*

6. Concluding Remarks

In this paper, we present efficient algorithms for solving line-constrained mobile sensor coverage problems. Future work includes investigating whether a logarithmic factor can be further shaved for the weighted case of the mobile disk coverage problem as well as for the mobile interval coverage problem. In particular, it would be interesting to see whether a faster algorithm exists for the unweighted case of the mobile interval coverage problem.

Note that for the 1D problem, i.e., all sensors and barrier points are given on ℓ and sensors are allowed to move on ℓ only, the algorithm can be simplified as follows. For the unweighted case, we can use the same algorithm as in Sec. 3 but the algorithm becomes simpler as $y_b = 0$ for each barrier point $b \in B$. The runtime of the algorithm is $O((n + m) \log(n + m))$. For the weighted case, we can use the same algorithm as in Sec. 5 but the algorithm becomes simpler as $y_i = 0$ for each sensor $s_i \in S$. The runtime of the algorithm is $O(m \log m + n \log^2 n)$.

ORCID

Haitao Wang https://orcid.org/0000-0001-8134-7409

References

1. A. M. Andrews and H. Wang, Minimizing the aggregate movements for interval coverage, *Algorithmica* **78** (2017) 47–85.
2. B. Bhattacharya, B. Burmester, Y. Hu, E. Kranakis, Q. Shi and A. Wiese, Optimal movement of mobile sensors for barrier coverage of a planar region, *Theoretical Computer Science* **410**(52) (2009) 5515–5528.
3. D. Z. Chen, Y. Gu, J. Li and H. Wang, Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain, *Discrete and Computational Geometry* **50** (2013) 374–408.
4. D. Z. Chen, X. Tan, H. Wang and G. Wu, Optimal point movement for covering circular regions, *Algorithmica* **72** (2015) 379–399.
5. D. Z. Chen, C. Wang and H. Wang, Representing a functional curve by curves with fewer Peaks, *Discrete and Computational Geometry* **46**(2) (2011) 334–360.
6. J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia and M. Yazdani, On minimizing the maximum sensor movement for barrier coverage of a line segment, In *Proceedings of the* 8*th International Conference on Ad-Hoc*, *Mobile and Wireless Networks* (2009) 194–212.
7. J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, J. Opatrny, L. Stacho, J. Urrutia and M. Yazdani, On minimizing the sum of sensor movements for barrier coverage of a line segment, In *Proceedings of the* 9*th International Conference on Ad-Hoc*, *Mobile and Wireless Networks* (2010) 29–42.
8. S. Dobrev, S. Durocher, M. Eftekhari, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, S. Shende and J. Urrutia, Complexity of barrier coverage with relocatable sensors in the plane, *Theoretical Computer Science* **579** (2015) 64–73.
9. H. Fan, M. Li, X. Sun, P. Wan and Y. Zhao, Barrier coverage by sensors with adjustable ranges, *ACM Transactions on Sensor Networks* **11** (2014) 14:1–14:20.
10. G. Frederickson and D. Johnson, Generalized selection and ranking: Sorted matrices, *SIAM Journal on Computing* **13**(1) (1984) 14–30.
11. G. N. Frederickson, Optimal algorithms for tree partitioning, In *Proceedings of the* 2*nd Annual ACM-SIAM Symposium of Discrete Algorithms* ($SODA$) (1991) 168–177.
12. G. N. Frederickson, Parametric search and locating supply centers in trees, In *Proceedings of the* 2*nd International 620 Workshop on Algorithms and Data Structures* ($WADS$) (1991) 299–319.
13. D. S. Hochbaum and W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, *Journal of the ACM* **32** (1985) 130–136.
14. P. Huang, W. Zhu and L. Guo, On the complexity of and algorithms for min-max target coverage on a line boundary, In *Proceedings of the 15th International Conference on Theory and Applications of Models of Computation* ($TAMC$) (2019) 313–324.
15. S. Kumar, T. H. Lai and A. Arora, Barrier coverage with wireless sensors, In *Proceedings of the* 11*th Annual International Conference on Mobile Computing and Networking* ($MobiCom$) (2005) 284–298.
16. V. C. S Lee, H. Wang and X. Zhang, Minimizing the maximum moving cost of interval coverage, *International Journal of Computational Geometry and Applications* **27** (2017) 187–205.
17. S. Li and H. Shen, Minimizing the maximum sensor movement for barrier coverage in the plane, In *Proceedings of the* 2015 *IEEE Conference on Computer Communications* ($INFOCOM$) (2015) 244–252.
18. S. Li and H. Wang, Algorithms for covering multiple barriers, *Theoretical Computer Science* **758** (2019) 61–72.

19. D. Liang, H. Shen and L. Chen, Maximum target coverage problem in mobile wireless sensor networks, *Sensors* **21** (2015) 1–13, Article No. 184.
20. M. Mehrandish, L. Narayanan and J. Opatrny, Minimizing the number of sensors moved on line barriers, In *Proceedings of IEEE Wireless Communications and Networking Conference* ($WCNC$) (2011) 653–658.
21. X. Zhang, H. Fan, V. C. S. Lee, M. Li, Y. Zhao and C. Liu, Minimizing the total cost of barrier coverage in a linear domain, *Journal of Combinatorial Optimization* **36** (2018) 434–457.