Minimizing the Maximum Moving Cost of Interval Coverage*

Victor C. S. Lee[1], Haitao Wang[2], and Xiao Zhang[1]

[1] Department of Computer Science
City University of Hong Kong, Hong Kong
csvlee@cityu.edu.hk, xiao.zhang@my.cityu.edu.hk
[2] Department of Computer Science
Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract. In this paper, we consider an interval coverage problem. We are given n intervals of the same length on a line L and a line segment B on L. Each interval has a nonnegative weight. The goal is to move the intervals along L such that every point of B is covered by at least one interval and the maximum moving cost of all intervals is minimized, where the moving cost of each interval is its moving distance times its weight. Algorithms for the "unweighted" version of this problem have been given before. In this paper, we present a first-known algorithm for this weighted version and our algorithm runs in $O(n^2 \log n \log\log n)$ time. The problem has applications in mobile sensor barrier coverage, where B is the barrier and each interval is the covering interval of a mobile sensor.

1 Introduction

In this paper, we consider an interval coverage problem. We are given n intervals of the same lengths on a line L and a line segment B on L, where each interval has a nonnegative weight. The goal is to move the intervals along L such that every point of B is covered by at least one interval and the maximum moving cost of all intervals is minimized, where the *moving cost* of each interval is its moving distance times its weight. The problem has applications in barrier coverage of mobile sensors in wireless sensor networks. For convenience, in the following we introduce and discuss the problem from the barrier coverage point of view.

Let L be a line, say, the x-axis. Let B be a line segment on L and B is called a *barrier*. Denote by β the length of B. Without loss of generality, we assume B is the interval $[0, \beta]$ on L. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n sensors and each sensor s_i is a point on L with coordinate x_i. Each sensor s_i has a weight $w_i \geq 0$. All sensors have the same *sensing range* r. Namely, if a sensor is currently at a location x' on L, then all points of L in the interval $[x' - r, x' + r]$ are said to be *covered* by the sensor and the interval is called the *covering interval* of the sensor. The problem is to move each sensor s_i of S to a new location y_i on L such that every point of B is covered by at least one sensor of S and the value $\max_{1 \leq i \leq n} w_i \cdot |x_i - y_i|$ is minimized. For each sensor s_i, we call the value $w_i \cdot |x_i - y_i|$ the *moving cost* of s_i. We refer to the problem as the *weighted barrier coverage*, denoted by WBC. We assume $2r \cdot n \geq \beta$ since otherwise a coverage of B would not be possible.

If all sensors have the same weight, then we refer to it as the "unweighted" version. An $O(n \log n)$ time algorithm has been given for the unweighted version by Chen et al. [4]. For the weighted version, to the best of our knowledge, we are not aware of any previous work. In this paper, we present an algorithm for the weighted version and our algorithm runs in $O(n^2 \log n \log\log n)$ time.

* A preliminary version of this paper appeared in the Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC 2015). H. Wang was supported in part by NSF under Grant CCF-1317143. Part of the work by X. Zhang was carried out during his visit at Utah State University.

1.1 Related Work

Mobile Wireless Sensor Networks (MWSNs) consist of a number of mobile wireless sensor nodes, which have limited battery power and may have different energy dissipation ratio (characterized by the weights). The advantage of allowing the sensors to be mobile increases monitoring capability compared to those for which static wireless sensors are used. One of the most important applications in MWSNs is to monitor a barrier to detect intruders in an attempt to cross a specific region. Barrier coverage [11], which guarantees that every movement crossing a barrier of sensors will be detected, is known to be an appropriate model of coverage for such applications.

If the sensors of S have different sensing ranges, we call the problem the "non-uniform case" (otherwise it is the "uniform case"). Hence, our problem is the weighted uniform case. For the unweighted uniform case, Czyzowicz et al. [7] first gave an $O(n^2)$ time algorithm, and later, Chen et al. [4] solved the problem in $O(n \log n)$ time. For the unweighted non-uniform case, Chen et al. [4] presented an $O(n^2 \log n)$ time algorithm.

The *min-sum* unweighted version of the problem has also been studied, where the objective is to minimize the sum of the moving distances of all sensors. The non-uniform case of the problem is NP-hard [8]. For the uniform case, Czyzowicz et al. [8] gave an $O(n^2)$ time algorithm, and recently, Andrews and Wang [1] proposed an $O(n \log n)$ time solution. Another variation of the problem is the *min-num* version, where the goal is to move the minimum number of sensors to form a barrier coverage. Mehrandish et al. [13, 14] proved the problem is NP-hard if sensors have different ranges and gave polynomial time algorithms otherwise.

Some problems on static sensors have also been considered. For example, Bar-Noy and Baumer [2] studied a problem of maximizing the lifetime of a network with static sensors, where the goal is to schedule the active time of sensors in a network so that the lifetime is maximized. A similar problem was considered in [3]. Fan et al. [10] studied a problem that aims to set an energy for each sensor to form a coverage such that the cost of all sensors is minimized.

1.2 Our Techniques

The unweighted uniform case of WBC is much easier due to an *order preserving property* [4, 7], which says that there always exists an optimal solution in which the order of the sensors is the same as that in the input. However, the property no longer holds for the unweighted non-uniform case [4]. We can easily show that for the weighted version, the property does not hold even for the uniform case. This is one main difficulty for solving our problem WBC.

To solve the problem, we generalize the techniques in [4] for the unweighted non-uniform case. Specifically, let λ^* denote the maximum moving cost in an optimal solution of WBC. We first solve a *decision problem* in $O(n \log n)$ time to determine whether $\lambda \geq \lambda^*$ for any given value λ. If $\lambda \geq \lambda^*$, then our decision algorithm will find a "feasible solution" in which the order of the sensors will be determined. Further, with $O(n^2)$ time preprocessing, we can solve the decision problem in $O(n \log \log n)$ time for any λ (the $\log \log n$ factor is due to the van Emde Boas Tree [6]). For solving our original problem WBC (referred to as the *optimization problem* for differentiation from the decision problem), we use an approach similar in spirit to parametric search [5, 12]. Namely, our optimization algorithm tries to "parameterize" the decision algorithm. Although we do not know the value λ^*, we will simulate the behavior of the decision algorithm on $\lambda = \lambda^*$, i.e., we determine the same order of the sensors as it would be obtained by the decision algorithm on $\lambda = \lambda^*$. To this end, our algorithm maintains an interval $(\lambda_1, \lambda_2]$ that contains λ^* and each step of the algorithm

will shrink the interval by calling the decision algorithm on certain values λ. Unlike the traditional parametric search [5, 12], our approach does not involve any parallel scheme and is actually quite practical.

The rest of the paper is organized as follows. In Section 2, we introduce some notation. In Section 3, we present our algorithm for the decision problem. Section 4 solves the optimization problem. Section 5 concludes the paper.

2 Preliminaries

For ease of exposition, we make an assumption that the weight of each sensor s_i of S is positive. The case where some sensors have zero weight can also be handled by our techniques but the discussions would be more tedious. We follow some terminologies in the previous work [4].

We use a *configuration* to refer to a specification on where each sensor $s_i \in S$ is located. For example, in the *input configuration*, each s_i is at x_i. We use C_I to denote the input configuration. Note that we can determine whether $\lambda^* = 0$ by checking whether the union of the covering intervals of all sensors in C_I contains B, which can be easily done in $O(n)$ time. If $\lambda^* = 0$, we do not need to move any sensor. Henceforth, we assume $\lambda^* > 0$.

For any sensor s_i, we use $I(s_i)$ to denote its covering interval. For any subset S' of sensors, with a little abuse of notation, we use $I(S')$ to denote the union of the covering intervals of all sensors in S'. For each sensor s_i, we call the left endpoint of $I(s_i)$ the *left extension* of s_i and call the right endpoint of $I(s_i)$ the *right extension*.

With a little abuse of notation, for any point x on L, we also use x to denote its coordinate on L, and vice versa. For any point x on L, we use $p^+(x)$ to denote a point x' such that $x' > x$ and x' is infinitesimally close to x. Note that our algorithm never needs to find such a point $p^+(x)$ and we use $p^+(x)$ only for explaining our idea.

3 The Decision Problem

In this section, we consider the decision problem: given any value $\lambda > 0$, determine whether $\lambda \geq \lambda^*$. Our approach for the optimization problem in Section 4 needs an algorithm for solving the decision problem. Note that the decision problem may have independent interest because it characterizes the problem model in which each sensor has a limited energy λ and we want to know whether their energy is sufficient for them to move to form a barrier coverage.

We present an $O(n \log n)$ time algorithm that solves the decision problem. We call this algorithm the *decision algorithm*. Given any value λ, if $\lambda \geq \lambda^*$, then we say that λ is a *feasible value* and our decision algorithm will find a *feasible solution* in which B is covered and the moving cost of each sensor is at most λ.

Consider any value $\lambda > 0$. For any sensor $s_i \in S$, λ/w_i is the maximum distance that s_i is allowed to move on L. Let $a_i = x_i - r - \lambda/w_i$ and $b_i = x_i + r + \lambda/w_i$. Note that a_i is the leftmost point and b_i is the rightmost point on L that can be covered by s_i with respect to λ. We call a_i (resp., b_i) the *leftmost (resp., rightmost) λ-coverable point* of s_i. Let $x_i^l = a_i + r$ and $x_i^r = b_i - r$. Namely, x_i^l (resp., x_i^r) is the leftmost (resp., rightmost) location that s_i is allowed to move, and we call it the *leftmost (resp., rightmost) λ-reachable location* of s_i.

In Section 3.1 we describe the algorithm while deferring the implementation details to Section 3.4. In Sections 3.2 and 3.3, we prove the correctness. The high-level scheme of our decision algo-

rithm is similar to that for the unweighted non-uniform case in [4], but the low-level details are essentially different.

3.1 The Algorithm Description

In the beginning, we move each sensor s_i to its rightmost λ-reachable location x_i^r. Let C_0 denote the resulting configuration. In C_0, for each sensor s_i, it is not allowed to move rightwards but can move leftwards by distance $2\lambda/w_i$.

If $\lambda \geq \lambda^*$, our algorithm will compute a subset S^c of sensors with their new locations such that B is covered by these sensors (i.e., $B \subseteq I(S^c)$) and the moving cost of each sensor of S^c is at most λ. For each sensor $s_i \in S \setminus S^c$, it can be anywhere in $[x_i^l, x_i^r]$, but in our solution it is at x_i^r (i.e., it does not move from its location in C_0). We call S^c a *solution subset*.

Consider a general step i with $i \geq 1$. Let C_{i-1} be the configuration right before the i-th step. Our algorithm maintains the following *invariants*. (1) We have a subset of sensors $S_{i-1} = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i-1)}\}$, where for each $1 \leq j \leq i-1$, $g(j)$ is the index of the sensor $s_{g(j)}$ in S. Let $S_{i-1} = \emptyset$ for $i = 1$. (2) In C_{i-1}, for each sensor s_k of S, if s_k is in S_{i-1}, then s_k is at a new location $y_k \in [x_k^l, x_k^r]$; otherwise, it is still at x_k^r. (3) $B \cap I(S_{i-1})$, i.e., the intersection of B and the union of the covering intervals of all sensors of S_{i-1} in C_{i-1}, is an interval $[0, R_{i-1}]$ for some value $0 \leq R_{i-1} < \beta$. This means that the point $p^+(R_{i-1})$ is not covered by any sensor in S_{i-1}. Let $R_0 = 0$.

Initially when $i = 1$, we have $S_{i-1} = \emptyset$ and $R_0 = 0$, and thus all algorithm invariants hold for C_0. The i-th step of the algorithm will find a new sensor $s_{g(i)} \in S \setminus S_{i-1}$ and move it to a new location $y_{g(i)} \in [x_{g(i)}^l, x_{g(i)}^r]$ (and thus obtain a new configuration C_i). Let $R_i = y_{g(i)} + r$ and $S_i = S_{i-1} \cup \{s_{g(i)}\}$. We will show that $B \cap I(S_i) = [0, R_i]$. If $R_i \geq \beta$, then we have found a feasible solution and we can terminate the algorithm with S_i as our solution subset. Otherwise, we proceed to the next step $i+1$ and all algorithm invariants have been maintained. We give the details of the i-th step below. Note that the discussions are on the configuration C_{i-1}.

Since the interval $[0, R_{i-1}]$ is currently covered by the sensors of S_{i-1}, we need to find sensors in $S \setminus S_{i-1}$ to cover the rest of the barrier, i.e., $[R_{i-1}, \beta]$.

Define S_{i1} to be the set of sensors that cover the point $p^+(R_{i-1})$ in C_{i-1}, i.e., $S_{i1} = \{s_k \mid x_k^r - r \leq R_{i-1} < x_k^r + r\}$. According to the algorithm invariants, no sensor in S_{i-1} covers $p^+(R_{i-1})$. Thus, it holds that $S_{i1} \subseteq S \setminus S_{i-1}$.

If $S_{i1} \neq \emptyset$, then we choose an *arbitrary* sensor in S_{i1} as $s_{g(i)}$[3] (e.g., see Fig. 1) and let $y_{g(i)} = x_{g(i)}^r$. We let $R_i = y_{g(i)} + r$ and $C_i = C_{i-1}$ (i.e., C_i is the same as C_{i-1} since $s_{g(i)}$ is not moved in this case).

If $S_{i1} = \emptyset$, then define S_{i2} as the set of sensors of S whose leftmost λ-coverable points are to the left of (or at) R_{i-1} and whose left extensions are strictly to the right of R_{i-1}, i.e., $S_{i2} = \{s_k \mid a_k \leq R_{i-1} < x_k^r - r\}$. Hence, for each $s_k \in S_{i2}$, s_k currently in C_{i-1} does not cover $p^+(R_{i-1})$ but we can move s_k leftwards for a distance at most $2\lambda/w_k$ to cover it.

If $S_{i2} \neq \emptyset$, then we choose the *leftmost* sensor of S_{i2} as $s_{g(i)}$ (e.g., see Fig. 2), and let $y_{g(i)} = R_{i-1} + r$. We move $s_{g(i)}$ to $y_{g(i)}$ to obtain the configuration C_i (where the right extension of $s_{g(i-1)}$ co-locates with the left extension of $s_{g(i)}$).

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

[3] It might be more natural to pick the rightmost sensor of S_{i1} as $s_{g(i)}$. In fact, an arbitrary one is sufficient.

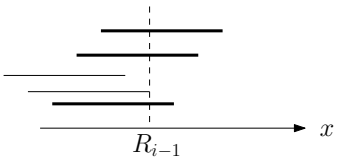

Fig. 1. Illustrating the set S_{i1}. The segments are the covering intervals of sensors. The thick segments correspond to the sensors in S_{i1}. Every sensor in S_{i1} can be $s_{g(i)}$.

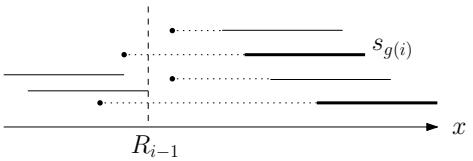

Fig. 2. Illustrating the set S_{i2}. The segments are the covering intervals of sensors. The thick segments correspond to the sensors in S_{i2}. The four black points are the leftmost λ-coverable points of the four sensors to the right of R_{i-1}. The sensor $s_{g(i)}$ is labeled.

Hence, if $S_{i1} = S_{i2} = \emptyset$, the algorithm will stop and report $\lambda < \lambda^*$. Otherwise, a sensor $s_{g(i)}$ is found from either S_{i1} or S_{i2}, and it is moved to $y_{g(i)}$. In either case, according to our discussion, $R_i = y_{g(i)} + r$ and $[0, R_i] = I(S_i)$ in C_i (recall that $S_i = S_{i-1} \cup \{s_{g(i)}\}$). If $R_i \geq \beta$, then we terminate the algorithm and report $\lambda \geq \lambda^*$ and C_i as a feasible solution; otherwise, we proceed to the next step $i + 1$ and all algorithm invariants have been maintained.

As there are n sensors in S, the algorithm will finish in at most n steps.

3.2 The Algorithm Correctness

The high-level scheme of the correctness proof is similar to that for the unweighted non-uniform case in [4], although the details are quite different.

The decision algorithm either reports $\lambda \geq \lambda^*$ or $\lambda < \lambda^*$. If the former case happens, say, in the i-th step, then according to our algorithm description, the configuration C_i is a feasible solution. Below, we show that if the algorithm reports $\lambda < \lambda^*$, then there is indeed no feasible solution for the value λ.

An interval on L is said to be *left-aligned* if its left endpoint is at 0. The correctness will be easily shown with the following lemma.

Lemma 1. *Consider any configuration C_i in our algorithm. Suppose S'_i is the set of sensors in S whose right extensions are at most R_i in the configuration C_i. Then, the interval $[0, R_i]$ is the largest possible left-aligned interval that can be covered by the sensors of S'_i in any configuration with respect to λ (i.e., the moving cost of each sensor of S'_i is at most λ).*

We defer the proof of the lemma to Section 3.3. In the following, we complete the correctness proof of our algorithm by using Lemma 1, i.e., we show that if the algorithm reports $\lambda < \lambda^*$, then there is no feasible solution for λ.

Suppose our algorithm reports $\lambda < \lambda^*$ in the i-th step. Then, according to our algorithm, $R_{i-1} < \beta$ and both S_{i1} and S_{i2} are empty in the configuration C_{i-1}. Let S'_{i-1} be the set of sensors whose right extensions are at most R_{i-1} in C_{i-1}. On the one hand, by Lemma 1 (replacing the index i in the lemma by $i - 1$), $[0, R_{i-1}]$ is the largest left-aligned interval that can be covered by the sensors in S'_{i-1}. On the other hand, since both S_{i1} and S_{i2} are empty, no sensor in $S \setminus S'_{i-1}$ can cover the point $p^+(R_{i-1})$. Due to $R_{i-1} < \beta$, we conclude that sensors of S cannot cover the interval $[0, p^+(R_{i-1})] \subseteq [0, \beta]$ with respect to the maximum moving cost λ. Thus, there is no feasible solution for λ.

This establishes the correctness of our decision algorithm.

3.3 The Proof of Lemma 1

We give the proof of Lemma 1. In the following, unless otherwise stated, in any configuration the moving cost of any sensor is no more than λ. We follow the notation defined in the statement of Lemma 1. Note that according to our algorithm, $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\} \subseteq S'_i$.

We first prove the following lemma.

Lemma 2. *If C is a configuration in which a left-aligned interval $[0, x']$ is covered by the sensors of S'_i (i.e., $[0, x'] \subseteq I(S'_i)$), then there always exists a configuration C^* in which $[0, x'] \subseteq I(S'_i)$ still holds and the location of the sensor $s_{g(j)}$ in C^* is $y_{g(j)}$ for each $1 \leq j \leq i$.*

Proof. We prove the lemma by induction. We assume that the lemma statement holds for $k-1$ with $1 \leq k \leq i$, i.e., there is a configuration C' in which $[0, x'] \subseteq I(S'_i)$ and $s_{g(j)}$ is located at $y_{g(j)}$ for each $1 \leq j \leq k-1$. Note that the assumption trivially holds for $k = 1$. Below we show that the lemma also holds for k.

Our goal is to find a configuration C'' in which $[0, x'] \subseteq I(S'_i)$ and $s_{g(j)}$ is located at $y_{g(j)}$ for each $1 \leq j \leq k$. We call such a configuration C'' a *target configuration*.

According to our decision algorithm, in the configuration C_k, $s_{g(j)}$ is at $y_{g(j)}$ for each $1 \leq j \leq k$, and $I(S_k) \cap B = [0, R_k]$. Hence, if $x' \leq R_k$, we can let $C'' = C_k$, which is a target configuration. In the following, we assume $x' > R_k$.

Let $t = g(k)$ and let y'_t be the location of s_t in C', If $y_t = y'_t$, then we let $C'' = C'$ and thus C'' is a target configuration. In the following, we assume $y_t \neq y'_t$. Recall that according to our algorithm, s_t is either from S_{k1} or from S_{k2}. We discuss the two cases below.

The Case $s_t \in S_{k1}$. If $s_t \in S_{k1}$, then $y_t = x^r_t$. Since x^r_t is the rightmost λ-reachable point of s_t, it must be that $y'_t < y_t$.

By our algorithm invariants, right before the k-th step of the algorithm, the sensors in S_{k-1} cover the interval $[0, R_{k-1}]$, and for each $1 \leq j \leq k-1$, the location of sensor $s_{g(j)}$ is $y_{g(j)}$. Hence, in the configuration C', $[0, R_{k-1}]$ is covered by the sensors in $S'_i \setminus \{s_t\}$.

Consider the configuration C'' obtained from C' by moving s_t rightwards from y'_t to y_t. Since $s_t \in S_{k1}$, s_t still covers $p^+(R_{k-1})$. Therefore, after s_t moves to y_t, s_t covers a longer sub-interval of $[R_{k-1}, \infty)$ than before, which implies that $[0, x'] \subseteq I(S'_i)$ holds in C''. Hence, C'' is a target configuration.

This proves the lemma for the case $s_t \in S_{k1}$.

The Case $s_t \in S_{k2}$. If $s_t \in S_{k2}$, then $S_{k1} = \emptyset$. In this case, according to our algorithm, s_t is the leftmost sensor of S_{k2} in the configuration C_{k-1}. Also, y_t is the rightmost location for s_t to cover the point $p^+(R_{k-1})$.

If $y'_t < y_t$, by the same argument as above, we can obtain a target configuration C'' from C' by moving s_t rightwards from y'_t to y_t. In the following, we assume $y'_t > y_t$. This also implies that s_t does not cover $p^+(R_{k-1})$ in C'.

Since $x' > R_{k-1}$, there must be a sensor $s_h \in S'_i$ that covers $p^+(R_{k-1})$ in C'. Further, since for each $1 \leq j \leq k-1$, $s_{g(j)}$ is at $y_{g(j)}$ in C', and $S_{k1} = \emptyset$, s_h must be in S_{k2}. As s_t is the leftmost sensor of S_{k2}, we have $x^r_t \leq x^r_h$.

Let C'' be the configuration obtained from C' by moving s_h to y'_t and moving s_t to y_t (e.g., see Fig. 3). In the sequel, we show that C'' is a target configuration. To this end, it is sufficient to show that (1) the interval $[0, x']$ is still covered by the sensors of S'_i in C'' and (2) the moving cost of s_h is no more than λ (i.e., $y'_t \in [x^l_h, x^r_h]$).

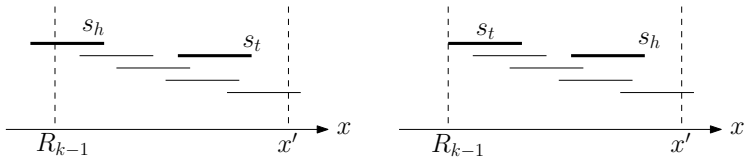

Fig. 3. Illustrating two configurations C' (left) and C'' (right) on the interval $[R_{k-1}, x']$.

- First of all, in both C' and C'', the sensors in $\{s_{g(1)}, s_{g(2)}, \ldots s_{g(k-1)}\}$ together cover $[0, R_{k-1}]$. Let $I = [R_{k-1}, x']$. Since the covering intervals of s_h and s_t have the same length and the left extension of s_t in C'' is at R_{k-1}, the sub-interval of I covered by s_h in C' is no larger than that covered by s_t in C'' (e.g., see Fig. 3). On the other hand, the sub-interval of I covered by s_t in C' is exactly the same as that covered by s_h in C'' since s_h is at the same location in C'' as s_t in C'. Since I is covered in C', it is also covered in C''. This shows that the interval $[0, x']$ is still covered by the sensors of S'_i in C''.
- To prove the above (2), let x'_h be the location of s_h in C'. Hence, $x'_h \in [x^l_h, x^r_h]$. To prove $y'_t \in [x^l_h, x^r_h]$, it suffices to show that $y'_t \in [x'_h, x^r_h]$.
  On the one hand, $y'_t < y_t = x^r_t$. Recall that $x^r_t \leq x^r_h$. Thus, $y'_t \leq x^r_h$ holds.
  On the other hand, since s_h covers $p^+(R_{k-1})$ in C' and $y'_t = R_{k-1}+r$, we have $x'_h \leq R_{k-1}+r = y'_t$.
  The above obtains that $y'_t \in [x'_h, x^r_h]$ and thus proves (2).

This proves the lemma for the case $s_t \in S_{k2}$.

The lemma thus follows. □

In the sequel, we use Lemma 2 to prove Lemma 1.

Let $[0, x']$ be the largest left-aligned interval that can be covered by the sensors of S'_i in any configuration. In light of Lemma 2, there always exists a configuration C^* in which $[0, x'] \subseteq I(S'_i)$ and the location of $s_{g(j)}$ is $y_{g(j)}$ for each $1 \leq j \leq i$. Recall that $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\}$, which is a subset of S'_i. According to our algorithm, $I(S_i) \cap B = [0, R_i]$ in C^*.

On the one hand, consider any sensor $s_k \in S'_i \setminus S_i$. According to our algorithm, its location in the configuration C_i is x^r_k, which is its rightmost λ-reachable point. By the definition of S'_i, the right extension of s_k in C_i is at most R_i, i.e., $x^r_k + r \leq R_i$. Since the location of s_k in C^* cannot be larger than its rightmost λ-reachable point, i.e., x^r_k, the right extension of s_k in C^* is at most $x^r_k + r$, which is at most R_i. Therefore, s_k cannot cover any point to the right of R_i in C^*.

On the other hand, we know that $I(S_i) \cap B = [0, R_i]$ in C^*.

Based on the above discussion, no sensor of S'_i can cover any point to the right of R_i. Further, because $[0, R_i]$ is covered by the sensors of $S_i \subseteq S'_i$ in C^* and by the definition of x', we obtain that $x' = R_i$. In other words, $[0, R_i]$ is the largest left-aligned interval that can be covered by the sensors of S'_i in any configuration with respect to λ. This proves Lemma 1.

3.4 The Algorithm Implementation

We first give an implementation for our algorithm that runs in $O(n \log n)$ time. Later we will improve the implementation with certain preprocessing.

We first move each sensor s_i to x^r_i to obtain the initial configuration C_0. Then, we sort the $2n$ extensions of all sensors in C_0 from left to right. During the algorithm, in each i-th step, we need to maintain the set S_{i1}. To this end, we sweep a point p on the line L from left to right. During

the sweeping, when p encounters the left extension of a sensor, we insert the sensor into S_{i1}, and when p encounters the right extension of a sensor, we delete it from S_{i1}. In this way, in each i-th step, when p is at R_{i-1}, the set S_{i1} is available.

If $S_{i1} \neq \emptyset$, then we arbitrarily pick a sensor in S_{i1} as $s_{g(i)}$. To store the set S_{i1}, since all sensor covering intervals have the same length, an easy observation is that the earlier a sensor is inserted into S_{i1}, the earlier it is deleted from S_{i1}. Therefore, we can simply use a first-in-first-out queue to store all sensors of S_{i1} such that each insertion and deletion can be done in constant time. To pick an arbitrary sensor in S_{i1}, we can always pick the sensor in the front of the queue, which can be done in constant time.

If $S_{i1} = \emptyset$, then we need to determine whether $S_{i2} = \emptyset$. If yes, we terminate the algorithm and report $\lambda < \lambda^*$. Otherwise, we need to find the leftmost sensor of S_{i2} as $s_{g(i)}$. We assume that the sweeping point p is now at R_{i-1}. Recall that S_{i2} consists of all sensors in C_{i-1} whose left extensions are strictly to the right of R_{i-1} and whose leftmost λ-coverable points are to the left of (or at) R_{i-1}. To maintain the set S_{i2} during the sweeping of p, we do the following.

In the beginning we sort the n leftmost λ-coverable points of all sensors of S along with the $2n$ extensions of all sensors in C_0. During the sweeping of p, if p encounters a leftmost λ-coverable point of some sensor s_k, then we insert s_k to S_{i2}. Further, if p encounters a left extension of some sensor s_k, then we delete s_k from S_{i2} (recall that this is also the moment we should insert s_k to S_{i1}).

In this way, when p is at R_{i-1}, S_{i2} is available. Since we need to find the leftmost sensor in S_{i2}, we use a balanced binary search tree T to store all sensors of S_{i2} where the "key" of each sensor s_k is the value x_k^r (which is its location in C_{i-1}). Clearly, T can support each of the following operations on S_{i2} in $O(\log n)$ time: inserting a sensor, deleting a sensor, finding the leftmost sensor.

After $s_{g(i)}$ is found, R_i can be computed immediately as discussed in the algorithm description. If $s_{g(i)}$ is from S_{i1}, then we do not need to actually move $s_{g(i)}$. We proceed to sweep p as usual. If $s_{g(i)}$ is from S_{i2}, we need to move $s_{g(i)}$ leftwards to $y_{g(i)} = R_{i-1} + r$. Since $s_{g(i)}$ is moved, we should also update the original sorted list of the $2n$ extensions of all sensors in C_0 to guide the future sweeping of p. We use the following approach to avoid the explicit update. We maintain a flag table for all sensor extensions in C_0. Initially, every table entry is *valid*. If $s_{g(i)}$ is moved, then we set the table entries of the two extensions of the sensor *invalid*, which can be done in constant time. Due to this extra table, during the sweeping of p, when p encounters a sensor extension, we first check the table to see whether the extension is still valid. If yes, then we proceed as usual; otherwise we ignore the event. This only costs an extra constant time at each event. In addition, after we proceed to sweep p from R_{i-1}, before processing the next event, we always check whether it is before p arrives at R_i (which is actually the right extension of $s_{g(i)}$). If yes, we proceed as usual; otherwise, we should process the event at R_i (i.e., determine the next sensor $s_{g(i+1)}$).

Hence, during the sweeping of p, each event can be handled in $O(\log n)$ time and each i-th step of the algorithm can be performed in $O(\log n)$ time. Since there are $O(n)$ events, the total running time of the algorithm is $O(n \log n)$.

Note that the space of the algorithm is $O(n)$. Thus we have the following result for the decision problem.

Theorem 1. *Given any value λ, we can determine whether $\lambda \geq \lambda^*$ in $O(n \log n)$ time and $O(n)$ space.*

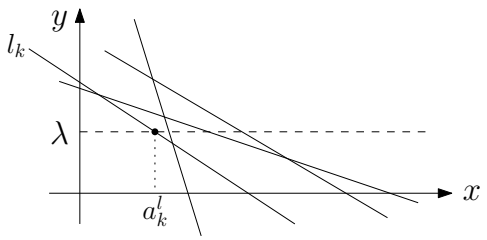

Fig. 4. Illustrating the line arrangement for sorting the leftmost λ-coverable points.

Our optimization algorithm in Section 4 will call our decision algorithm many times, for which we have the following alternative result by improving the above implementation with certain preprocessing.

Corollary 1. *With $O(n^2)$ time and $O(n^2)$ space preprocessing, we can determine whether $\lambda \geq \lambda^*$ in $O(n \log\log n)$ time for any given λ.*

Proof. Our previous $O(n \log n)$ time implementation is dominated by two parts. The first part is on sorting the $3n$ values, i.e., the leftmost λ-coverable points of all sensors and the left and right extensions of all sensors in C_0. The second part is on performing the operations on the set S_{i2}, each taking $O(\log n)$ time by using the balanced binary search tree T. The rest of the algorithm together takes $O(n)$ time. To reduce the running time (with the help of preprocessing), we need to reduce the time for the above two parts. In the following, we first discuss the second part.

Recall that the key of each sensor s_k of T is the value x_k^r because we need to find the leftmost sensor of T, which is also the sensor of T with the smallest key. Let $Q = \{x_k^r \mid 1 \leq k \leq n\}$. Assume we already have sorted all values in Q. For each sensor s_k of S, we use $rank(s_k)$ to denote the *rank* of x_k^r in the sorted Q (i.e., $rank(s_k) = t$ if x_k^r is at the t-th position from the left in the sorted list of Q). It is easy to see that the leftmost sensor of T is the sensor with the smallest rank. Therefore, we can also use the rank as the key of each sensor of T.

The advantage of using the ranks as the keys is that the rank of each sensor is an integer in $[1, n]$. Consequently, instead of using a standard balanced binary search tree, we can use an integer data structure, e.g., the van Emde Boas Tree (or vEB tree for short) [6], to maintain S_{i2}. The vEB tree can support each of the following operations on S_{i2} in $O(\log\log n)$ time [6]: inserting a sensor, deleting a sensor, and finding the sensor with the smallest rank. Using a vEB tree, all operations on S_{i2} in the algorithm can be performed in $O(n \log\log n)$ time. Note that this is based on the assumption that the sorted list of Q has been given.

In the sequel, we show that with $O(n^2)$ time and space preprocessing, given any λ, we can sort the values of Q in $O(n)$ time and also sort in $O(n)$ time the $3n$ values in the first part (i.e., the leftmost λ-coverable points and the extensions of all sensors in C_0).

Recall that the leftmost λ-coverable points are $a_k^l = x_k - r - \lambda/w_k$ for $k = 1, 2, \ldots, n$. We first show how to sort these values in $O(n)$ time with certain preprocessing.

For each $1 \leq k \leq n$, we consider a_k^l as a function of λ, which defines a line l_k in the 2D coordinate system where the x-coordinates correspond to the values of a_k^l and the y-coordinates correspond to the values of λ (e.g., see Fig. 4). For any value $\lambda > 0$, the x-coordinate of the intersection of l_k with the horizontal line $y = \lambda$ is exactly the value a_k^l at λ. Based on this geometric transformation, we sort all leftmost λ-coverable points by the following approach.

Let $\mathcal{A}$ be the line arrangement of all lines l_k for $k = 1, 2, \ldots, n$. By the above discussion, it is not difficult to see that the order of intersections of the horizontal line $y = \lambda$ with the lines of $\mathcal{A}$

from left to right corresponds exactly to the order of the leftmost λ-coverable points from left to right. Suppose $\mathcal{A}$ is already computed in the preprocessing. Given any value $\lambda > 0$, the above order can be obtained by traversing the cells of $\mathcal{A}$ along the horizontal line $y = \lambda$, which can be done in $O(n)$ time due to the zone theorem [9].

The arrangement $\mathcal{A}$ can be computed in $O(n^2)$ time and space [9]. Therefore, with $O(n^2)$ time and space preprocessing, for any λ, we can sort the leftmost λ-coverable points of all sensors of S in $O(n)$ time.

The sorting on the values in Q and the $2n$ sensor extensions in C_0 can be done similarly. Since the covering intervals of all sensors have the same length, the order of the sensors in C_0 sorted by their right extensions is the same as that by their left extensions and is also the same as that by the values of Q (because each value x_k^r of Q is the location of the sensor s_k in C_0). Therefore, once we have the sorted order by their right extensions, both the sorted order by their left extensions and the sorted order of the values of Q are available. Below, we focus on the sorting by the right extensions.

For each sensor s_k, its right extension in C_0 is actually the rightmost λ-coverable point of s_k. Hence, we can use a symmetric way to sort them as the above algorithm for sorting the leftmost λ-coverable points. With $O(n^2)$ time and space preprocessing, we can do the sorting in $O(n)$ time for any λ.

The above shows that with $O(n^2)$ time and space preprocessing, given any λ, we can obtain the following four sorted lists in $O(n)$ time: the values in Q, the leftmost λ-coverable points, the left extensions in C_0, and the right extensions in C_0. Finally, we merge the latter three sorted list into one sorted list in additional linear time, which is needed to guide the sweeping in the algorithm. The sorted lists of the values of Q is needed for the vEB tree for maintaining the set S_{i2} discussed above.

As a summary, with $O(n^2)$ time and space preprocessing, given any λ, our previous decision algorithm can be implemented in $O(n \log\log n)$ time. The lemma thus follows. □

4 The Optimization Problem

In this section, we give an $O(n^2 \log n \log\log n)$ time algorithm for solving the optimization problem. The goal is to compute λ^*, after which we can obtain an optimal solution by applying the decision algorithm on $\lambda = \lambda^*$. In the following, we focus on computing λ^*.

The high-level scheme of our algorithm is different from that for the unweighted non-uniform case in [4] because the algorithmic scheme in [4] relies on another decision algorithm that can determine whether $\lambda^* = \lambda$ for any given λ while our algorithmic scheme does not need such a decision algorithm. The details of our algorithm are even more different from those in [4].

In the following, we assume that the preprocessing in Corollary 1 has been done. Unless otherwise stated, we always use Corollary 1 to implement our decision algorithm with the understanding that if we use the implementation in Theorem 1, then we can obtain a slightly slower $O(n^2 \log^2 n)$ time algorithm but with only $O(n)$ space.

4.1 An Overview

As discussed before, a main difficulty is that we do not know the order of the sensors that cover B in an optimal solution. If we knew λ^*, then we could run our decision algorithm on $\lambda = \lambda^*$ to obtain an optimal solution in which the order of the sensors is also determined. We use an idea similar

to the parametric search [5, 12]. We "parameterize" our decision algorithm with λ as a parameter. Although we do not know the value λ^*, we execute the decision algorithm in such a way that it will determine the same solution subset of sensors $s_{g(1)}, s_{g(2)}, \ldots$ in the same order as would be obtained if we ran the decision algorithm on $\lambda = \lambda^*$. To this end, we will use our decision algorithm to prune certain λ values.

Recall that for any value λ, step i of our decision algorithm determines the sensor $s_{g(i)}$ and obtains the set $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\}$ with $I(S_i) \cap B = [0, R_i]$ in the configuration C_i. In our following algorithm, we often consider λ as a variable rather than a fixed value. Thus, we will use $S_i(\lambda)$ (resp., $R_i(\lambda)$, $s_{g(i)}(\lambda)$, $C_i(\lambda)$, $x_i^r(\lambda)$) to refer to the corresponding S_i (resp., R_i, $s_{g(i)}$, C_i, x_i^r).

Our algorithm has at most $n+1$ steps. Consider a general i-th step for $i \geq 1$. Right before the step, we have an interval $(\lambda_{i-1}^1, \lambda_{i-1}^2]$ and a sensor set $S_{i-1}(\lambda)$, such that the following algorithm invariants hold.

1. $\lambda^* \in (\lambda_{i-1}^1, \lambda_{i-1}^2]$, i.e., λ^* is either equal to λ_{i-1}^2 or in $(\lambda_{i-1}^1, \lambda_{i-1}^2)$.
2. The set $S_{i-1}(\lambda)$ is the same (with the same order) for all values $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$. If $\lambda^* \neq \lambda_{i-1}^2$, then $S_{i-1}(\lambda)$ has the same sensors as $S_{i-1}(\lambda^*)$ with the same order.
3. $R_{i-1}(\lambda)$ on $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$ is a nondecreasing linear function and it has been explicitly computed (and maintained).
4. $R_{i-1}(\lambda) < \beta$ for any $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$.

Initially when $i = 1$, we let $\lambda_0^1 = -\infty$ and $\lambda_0^2 = \infty$. Since $S_0(\lambda) = \emptyset$ and $R_0(\lambda) = 0$ for any λ, all invariants hold for $i = 1$.

The i-th step will either compute λ^*, or obtain a new interval $(\lambda_i^1, \lambda_i^2] \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2]$ and a sensor $s_{g(i)}(\lambda)$ with $S_i(\lambda) = S_{i-1}(\lambda) \cup \{s_{g(i)}(\lambda)\}$ such that all algorithm invariants hold on $S_i(\lambda)$ and $(\lambda_i^1, \lambda_i^2]$. We will show that the i-th step runs in $O(n \log n \log\log n)$ time. The details of the i-th step are given below.

4.2 A General i-th Step

We assume $\lambda^* \neq \lambda_{i-1}^2$ and thus λ^* is in $(\lambda_{i-1}^1, \lambda_{i-1}^2)$. In fact, our following algorithm does not rely on this assumption, but only uses the set $S_{i-1}(\lambda)$, the interval $(\lambda_{i-1}^1, \lambda_{i-1}^2)$, and the function $R_{i-1}(\lambda)$, which are all known. We make the assumption only for explaining the rationale of our approach.

Since $\lambda^* \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$, by our algorithm invariants, for any $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$, $S_{i-1}(\lambda)$ has the same sensors as $S_{i-1}(\lambda^*)$ with the same order. We simulate the decision algorithm on $\lambda = \lambda^*$. In order to determine the sensor $s_{g(i)}(\lambda^*)$, we first compute $S_{i1}(\lambda^*)$, i.e., the set of sensors covering the point $p^+(R_{i-1}(\lambda^*))$ in the configuration $C_{i-1}(\lambda^*)$, as follows.

Consider any sensor s_k in $S \setminus S_{i-1}(\lambda)$. Its position in the configuration $C_{i-1}(\lambda)$ is $x_k^r(\lambda) = x_k + \lambda/w_k$, which is an increasing function of λ. Thus, both the left and the right extensions of s_k in $C_{i-1}(\lambda)$ are increasing linear functions of λ. By our algorithm invariants, $R_{i-1}(\lambda)$ is a nondecreasing linear function. Suppose $f(\lambda)$ is the left or right extension of s_k in $C_{i-1}(\lambda)$. Unless the slope of $R_{i-1}(\lambda)$ is $1/w_k$, there is at most one value λ in $(\lambda_{i-1}^1, \lambda_{i-1}^2)$ such that $R_{i-1}(\lambda) = f(\lambda)$. Let S' be the set of sensors s_k of $S \setminus S_{i-1}(\lambda)$ such that $1/w_k$ is not equal to the slope of $R_{i-1}(\lambda)$. We compute the set $S_{i1}(\lambda^*)$ as follows.

Suppose we increase λ from λ_{i-1}^1 to λ_{i-1}^2. During the increasing of λ, we say that an "event" happens if $R_{i-1}(\lambda)$ is equal to the left or right extension value of a sensor $s_k \in S'$ at the current value of λ (called *event value*). It is not difficult to see that during the increasing of λ, the set

$S_{i1}(\lambda)$ is fixed between any two adjacent events. In order to compute $S_{i1}(\lambda^*)$, we first compute all event values, which can be done in linear time by using the function $R_{i-1}(\lambda)$ and all left and right extension functions of the sensors in S'. Let Λ denote the set of all event values. In addition, we add λ_{i-1}^1 and λ_{i-1}^2 to Λ. Next we sort all values in Λ. Then, by using our decision algorithm, we do binary search on the sorted list of Λ to find two adjacent values λ_1 and λ_2 in the sorted list such that $\lambda^* \in (\lambda_1, \lambda_2]$. Note that $(\lambda_1, \lambda_2] \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2]$. Since $|\Lambda| = O(n)$, the binary search calls our decision algorithm $O(\log n)$ times, which takes $O(n \log n \log\log n)$ time in total.

We make another assumption that $\lambda^* \neq \lambda_2$. Again, this assumption is only for explaining the rationale of our approach, and the following algorithm does not rely on this assumption. Under the assumption, for any λ in (λ_1, λ_2), the set $S_{i1}(\lambda)$ is exactly $S_{i1}(\lambda^*)$. Hence, we can compute $S_{i1}(\lambda^*)$ by taking any $\lambda \in (\lambda_1, \lambda_2)$ and explicitly computing $S_{i1}(\lambda)$, which can be done in $O(n)$ time.

The above has computed $S_{i1}(\lambda^*)$ in $O(n \log n \log\log n)$ time (provided that our previous two assumptions are true). According to our decision algorithm, depending on whether $S_{i1}(\lambda^*) = \emptyset$, there are two cases.

If $S_{i1}(\lambda^*) \neq \emptyset$, we take any sensor of $S_{i1}(\lambda^*)$ as $s_{g(i)}(\lambda^*)$. Further, we let $\lambda_i^1 = \lambda_1$, $\lambda_i^2 = \lambda_2$, and $S_i(\lambda) = S_{i-1}(\lambda) \cup \{s_{g(i)}(\lambda^*)\}$. We will show later that all algorithm invariants hold on $S_i(\lambda)$ and $(\lambda_i^1, \lambda_i^2]$ (further processing may be needed).

If $S_{i1}(\lambda^*) = \emptyset$, then according to our decision algorithm, we need to compute the set $S_{i2}(\lambda^*)$, i.e., the set of sensors of S whose leftmost λ-coverable points are to the left of (or at) $R_{i-1}(\lambda^*)$ and whose left extensions are strictly to the right of $R_{i-1}(\lambda^*)$. Under our previous two assumptions, we have $\lambda^* \in (\lambda_1, \lambda_2) \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$. By our algorithm invariants, $R_{i-1}(\lambda)$ is a nondecreasing linear function on $\lambda \in (\lambda_1, \lambda_2)$. For each sensor $s_k \in S$, its leftmost λ-coverable point $a_k(\lambda) = x_k - \lambda/w_k - r$ is a decreasing linear function. Therefore, the interval (λ_1, λ_2) contains at most one value λ such that $R_{i-1}(\lambda) = a_k(\lambda)$.

Suppose we increase λ from λ_1 to λ_2. During the increasing of λ, we say that an "event" happens when $R_{i-1}(\lambda)$ is equal to $a_k(\lambda)$ for some sensor $s_k \in S$ at some *event value* λ. During the increasing of λ, the set $S_{i2}(\lambda)$ is fixed between any two adjacent events. This suggests the following way to compute $S_{i2}(\lambda^*)$.

First, we compute all event values by using $R_{i-1}(\lambda)$ and the functions $a_k(\lambda)$ of all sensors s_k of S. Let Λ contain all event values. We also add λ_1 and λ_2 to Λ. We sort all values of Λ. Then, by using our decision algorithm, we do a binary search on the sorted list of Λ to find two adjacent values λ_1' and λ_2' in the sorted list such that $\lambda^* \in (\lambda_1', \lambda_2']$. Note that $(\lambda_1', \lambda_2'] \subseteq (\lambda_1, \lambda_2]$. Since $|\Lambda| = O(n)$, the above binary search calls the decision algorithm $O(\log n)$ times, which takes $O(n \log n \log\log n)$ time in total.

By our above analysis, the set $S_{i2}(\lambda)$ is fixed for all $\lambda \in (\lambda_1', \lambda_2')$. We take an arbitrary value $\lambda \in (\lambda_1', \lambda_2')$ and compute the set $S_{i2}(\lambda)$ explicitly, which can be done in $O(n)$ time.

Lemma 3. *If $S_{i2}(\lambda) = \emptyset$, then λ^* is in $\{\lambda_{i-1}^2, \lambda_2, \lambda_2'\}$.*

Proof. If $S_{i2}(\lambda) = \emptyset$, assume to the contrary that λ^* is not in $\{\lambda_{i-1}^2, \lambda_2, \lambda_2'\}$. This implies that our previous two assumptions on λ^* are true. Further, since $\lambda^* \neq \lambda_2'$, it holds that $\lambda^* \in (\lambda_1', \lambda_2')$. Consequently, according to our algorithm, $S_{i2}(\lambda^*) = S_{i2}(\lambda) = \emptyset$. This means that if we applied the decision algorithm on $\lambda = \lambda^*$, the sensor $s_{g(i)}(\lambda^*)$ would not exist (i.e., the decision algorithm would stop in the first $i-1$ steps). This implies that the optimal solution produced by the decision algorithm only uses sensors in $S_{i-1}(\lambda^*)$ to cover B, i.e., $B \subseteq I(S_{i-1}(\lambda^*))$.

On the other hand, according to our algorithm invariants, $R_{i-1}(\lambda) < \beta$ for $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$. Since $\lambda^* \in (\lambda_1', \lambda_2') \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$, $R_{i-1}(\lambda^*) < \beta$ holds. By our decision algorithm, $[0, R_{i-1}(\lambda^*)] =$

$B \cap I(S_{i-1}(\lambda^*))$ holds in the optimal solution. However, $R_{i-1}(\lambda^*) < \beta$ contradicts with the fact that $B \subseteq I(S_{i-1}(\lambda^*))$.

The lemma thus follows. □

By Lemma 3, if $S_{i2}(\lambda) = \emptyset$, then λ^* is the smallest feasible value of $\{\lambda_{i-1}^2, \lambda_2, \lambda_2'\}$, which can be found by calling our decision algorithms on the three values respectively. Otherwise, we proceed as follows.

We make the third assumption that $\lambda^* \neq \lambda_2'$. Similarly, this assumption is only for explaining our approach, and the following algorithm does not rely on this assumption. Under the assumption, $\lambda^* \in (\lambda_1', \lambda_2')$. By our above analysis, $S_{i2}(\lambda^*) = S_{i2}(\lambda)$. Next, we find the sensor $s_{g(i)}(\lambda^*)$, which is the leftmost sensor of $S_{i2}(\lambda^*)$. Although $S_{i2}(\lambda)$ is fixed for all $\lambda \in (\lambda_1', \lambda_2')$, the leftmost sensor of $S_{i2}(\lambda^*)$ may not be the same for all $\lambda \in (\lambda_1', \lambda_2')$. To find $s_{g(i)}(\lambda^*)$, we use the following approach.

According to our decision algorithm, for each sensor $s_k \in S_{i2}(\lambda)$, its location in the configuration $C_{i-1}(\lambda)$ is $x_k^r(\lambda) = x_k + \lambda/w$ for any $\lambda \in (\lambda_1', \lambda_2')$. Hence, $x_k^r(\lambda)$ is an increasing linear function of λ, which defines a line in the 2D coordinate system in which the x-coordinates correspond to the λ values and the y-coordinates correspond to x_k^r values. We consider the lower envelope $\mathcal{L}$ of the lines defined by all sensors of $S_{i2}(\lambda)$. For each point q of $\mathcal{L}$, suppose q lies on the line defined by the sensor s_k and q's x-coordinate is λ_q. Then, if $\lambda = \lambda_q$, the leftmost sensor of $S_{i2}(\lambda)$ is s_k. This means that each line segment of $\mathcal{L}$ corresponds to the same leftmost sensor of $S_{i2}(\lambda)$. Based on this observation, we proceed to compute $s_{g(i)}(\lambda^*)$ as follows.

We first compute the lower envelope $\mathcal{L}$, which can be done in $O(n \log n)$ time [9]. Then, let Λ be the set of the x-coordinates of the vertices of $\mathcal{L}$. Note that $|\Lambda| = O(n)$ since $\mathcal{L}$ is the lower envelope of at most n lines. We also add λ_1' and λ_2' to Λ. We sort all values of Λ. Then, by using our decision algorithm, we do binary search on the sorted list of Λ to find two adjacent values λ_1'' and λ_2'' such that $\lambda^* \in (\lambda_1'', \lambda_2'']$. Note that $(\lambda_1'', \lambda_2''] \subseteq (\lambda_1', \lambda_2']$. Since λ_1'' and λ_2'' are two adjacent values of the sorted Λ, by our above analysis, there is a sensor that is always the leftmost sensor of $S_{i2}(\lambda)$ for all $\lambda \in (\lambda_1'', \lambda_2'']$. To find the above leftmost sensor, we only need to take any value λ in $(\lambda_1'', \lambda_2'')$ and explicitly compute the locations of sensors in $S_{i2}(\lambda)$. The above algorithm finds $s_{g(i)}(\lambda^*)$ in $O(n \log n \log\log n)$ time, which is dominated by the binary search.

Further, we let $\lambda_i^1 = \lambda_1''$, $\lambda_i^2 = \lambda_2''$, and $S_i(\lambda) = S_{i-1}(\lambda) \cup \{s_{g(i)}(\lambda^*)\}$. We will show later that all algorithm invariants hold on $S_i(\lambda)$ and $(\lambda_i^1, \lambda_i^2]$ (further processing may be needed).

4.3 Maintaining the Algorithm Invariants

If λ^* has been computed above, then we terminate the algorithm. Otherwise, we have obtained an interval $(\lambda_i^1, \lambda_i^2] \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2]$ that contains λ^* and a sensor set $S_i(\lambda)$. Hence, the first algorithm invariant holds. In the sequel, we discuss the other three invariants.

According to our algorithm, $S_i(\lambda)$ is the same for all values λ in the open interval $(\lambda_i^1, \lambda_i^2)$. Further, assume $\lambda^* \neq \lambda_i^2$. Then, our above three assumptions (i.e., $\lambda^* \notin \{\lambda_{i-1}^2, \lambda_2, \lambda_2'\}$) are all true. By our algorithm invariants, $S_{i-1}(\lambda) = S_{i-1}(\lambda^*)$ holds for all $\lambda \in (\lambda_i^1, \lambda_i^2) \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$. Since our above algorithm for computing $s_{g(i)}(\lambda^*)$ is based on the above three assumptions and now that the assumptions are all true, the sensor $s_{g(i)}(\lambda^*)$ has been correctly computed above. Since $S_i(\lambda) = S_{i-1}(\lambda) \cup \{s_{g(i)}(\lambda^*)\}$ and $S_{i-1}(\lambda) = S_{i-1}(\lambda^*)$, $S_i(\lambda) = S_i(\lambda^*)$ holds for all $\lambda \in (\lambda_i^1, \lambda_i^2)$. This shows that the second invariant holds.

For the third invariant, recall that $R_i(\lambda)$ is equal to the right extension of $s_{g(i)}(\lambda)$ in $C_i(\lambda)$. Further, if $s_{g(i)}(\lambda) \in S_{i1}(\lambda)$, then $R_i(\lambda)$ is equal to $x_{g(i)} + r + \lambda/w_{g(i)}$, which is a nondecreasing linear

function of λ. If $s_{g(i)}(\lambda) \in S_{i2}(\lambda)$, then $R_i(\lambda) = R_{i-1}(\lambda)+2r$. Since $R_{i-1}(\lambda)$ is a nondecreasing linear function on $(\lambda_{i-1}^1, \lambda_{i-1}^2)$, $R_i(\lambda)$ is also a nondecreasing linear function on $(\lambda_i^1, \lambda_i^2) \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$. Therefore, in either case, the third algorithm invariant holds.

For the fourth invariant, if $R_i(\lambda) < \beta$ for all $\lambda \in (\lambda_i^1, \lambda_i^2)$, then the fourth invariant also holds. Otherwise, we first prove the following lemma.

Lemma 4. *If it is not true that $R_i(\lambda) < \beta$ for all $\lambda \in (\lambda_i^1, \lambda_i^2)$, then $R_i(\lambda)$ is a strictly increasing linear function on $(\lambda_i^1, \lambda_i^2)$ and there is a value $\lambda' \in (\lambda_i^1, \lambda_i^2)$ such that $R_i(\lambda') = \beta$.*

Proof. We first show that it is not possible that $R_i(\lambda) > \beta$ for all $\lambda \in (\lambda_i^1, \lambda_i^2)$.

Assume to the contrary that $R_i(\lambda) > \beta$ for all $\lambda \in (\lambda_i^1, \lambda_i^2)$. Since $\lambda^* \in (\lambda_i^1, \lambda_i^2]$, let λ' be any value in (λ_i^1, λ^*). We have $R_i(\lambda') > \beta$. But this would imply that we have found a feasible solution using only sensors in $S_i(\lambda)$, but the moving cost of each sensor in $S_i(\lambda)$ is at most $\lambda' < \lambda^*$, contradicting with that λ^* is the maximum moving cost in an optimal solution.

Hence, there must be at least a value $\lambda' \in (\lambda_i^1, \lambda_i^2)$ such that $R_i(\lambda') = \beta$. Next, we show that $R_i(\lambda)$ must be a strictly increasing linear function.

Assume to the contrary this is not true. Since $R_i(\lambda)$ is a nondecreasing linear function, $R_i(\lambda)$ must be constant on $(\lambda_i^1, \lambda_i^2)$. Thus, $R_i(\lambda) = \beta$ for all $\lambda \in (\lambda_i^1, \lambda_i^2)$. Since $\lambda^* \in (\lambda_i^1, \lambda_i^2]$, let λ' be any value in (λ_i^1, λ^*). Hence, $R_i(\lambda') = \beta$. As above, this means that λ' is a feasible value. However, $\lambda' < \lambda^*$ incurs contradiction. □

By Lemma 4, we compute the value $\lambda' \in (\lambda_i^1, \lambda_i^2)$ such that $R_i(\lambda') = \beta$. This means that B is covered by the sensors $S_i(\lambda')$ in the configuration $C_i(\lambda')$. Thus, λ' is a feasible value and $\lambda^* \in (\lambda_i^1, \lambda']$. Due to the fact that $R_i(\lambda)$ is a strictly increasing function, $R_i(\lambda) < \beta$ for all $\lambda \in (\lambda_i^1, \lambda')$. We update λ_i^2 to λ'. Now the fourth invariant also holds. Note that all the first three invariants still hold with this updated (and smaller) interval $(\lambda_i^1, \lambda_i^2)$.

This completes the i-th step. The running time is $O(n \log n \log\log n)$. If λ^* is not computed, then all algorithm invariants hold and we proceed on the next step. The next lemma shows that λ^* will eventually be computed.

Lemma 5. *The algorithm computes λ^* after at most n steps.*

Proof. According to our algorithm, λ^* will be computed in the i-th step if both $S_{i1}(\lambda)$ and $S_{i2}(\lambda)$ are empty. Assume that the algorithm has not found λ^* in the first n steps. Then, $S_n(\lambda)$ must contain all sensors of S. Therefore, in the $(n+1)$-th step, it is guaranteed that both $S_{n+1,1}(\lambda)$ and $S_{n+1,2}(\lambda)$ are empty, and thus λ^* will be computed (which is actually equal to λ_n^2 by Lemma 3). □

Since each step of the algorithm takes $O(n \log n \log\log n)$ time, λ^* can be finally computed in $O(n^2 \log n \log\log n)$ time. The space complexity of the algorithm is $O(n^2)$, which is dominated by the preprocessing of Corollary 1. If we use Theorem 1 to implement the decision algorithm, then the total time for computing λ^* is $O(n^2 \log^2 n)$ and the space complexity is $O(n)$.

Theorem 2. *The problem WBC can be solved in $O(n^2 \log n \log\log n)$ time and $O(n^2)$ space; alternatively, it can be solved in $O(n^2 \log^2 n)$ time and $O(n)$ space.*

5 Concluding Remarks

In this paper, we gave a first-known algorithm for solving the weighted (uniform) case of the barrier coverage problem. For the unweighted non-uniform case, Chen et al. [4] gave the following result for their decision problem: With $O(n \log n)$ time preprocessing, whether $\lambda \geq \lambda^*$ can be determined in $O(n)$ time for any given λ. We are not able to give such a result for our decision problem. An immediate consequence is that our algorithm for the optimization problem runs a little slower than their $O(n^2 \log n)$ time algorithm [4].

References

1. A.M. Andrews and H. Wang. Minimizing the aggregate movements for interval coverage. In *Proc. of the 14th Algorithms and Data Structures Symposium (WADS)*, pages 28–39, 2015.
2. A. Bar-Noy and B. Baumer. Average case network lifetime on an interval with adjustable sensing ranges. *Algorithmica*, 72:148–166, 2015.
3. A. Bar-Noy, D. Rawitz, and P. Terlecky. Maximizing barrier coverage lifetime with mobile sensors. In *Proc. of the 21st European Symposium on Algorithms (ESA)*, pages 97–108, 2013.
4. 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*, 50:374–408, 2013.
5. R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989.
6. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3nd edition, 2009.
7. 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 *Proc. of the 8th International Conference on Ad-Hoc, Mobile and Wireless Networks*, pages 194–212, 2009.
8. 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 *Proc. of the 9th International Conference on Ad-Hoc, Mobile and Wireless Networks*, pages 29–42, 2010.
9. M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry — Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.
10. H. Fan, M. Li, X. Sun, P. Wan, and Y. Zhao. Barrier coverage by sensors with adjustable ranges. *ACM Transactions on Sensor Networks*, 11:14, 2014.
11. S. Kumar, T.H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proc. of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 284–298, 2005.
12. N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.
13. M. Mehrandish. *On Routing, Backbone Formation and Barrier Coverage in Wireless Ad Doc and Sensor Networks*. PhD thesis, Concordia University, Montreal, Quebec, Canada, 2011.
14. M. Mehrandish, L. Narayanan, and J. Opatrny. Minimizing the number of sensors moved on line barriers. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 653–658, 2011.