Algorithms on Minimizing the Maximum Sensor Movement for Barrier Coverage of a Linear Domain[⋆]

Danny Z. Chen[1], Yan Gu[2], Jian Li[3], and Haitao Wang[1⋆⋆]

[1] Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556, USA
{dchen, hwang6}@nd.edu

[2] Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China
henryy321@gmail.com

[3] Institute for Interdisciplinary Information Sciences (IIIS)
Tsinghua University, Beijing 100084, China
lijian83@mail.tsinghua.edu.cn

Abstract. In this paper, we study the problem of moving n sensors on a line to form a barrier coverage of a specified segment of the line such that the maximum moving distance of the sensors is minimized. Previously, it was an open question whether this problem on sensors with arbitrary sensing ranges is solvable in polynomial time. We settle this open question positively by giving an $O(n^2 \log n \log\log n)$ time algorithm. Further, if all sensors have the same-size sensing range, we give an $O(n \log n)$ time algorithm, which improves the previous best $O(n^2)$ time solution.

1 Introduction

A Wireless Sensor Network (WSN) uses a large number of sensors to monitor some surrounding environmental phenomena [1]. Intrusion detection and border surveillance constitute a major application category for WSNs. A main goal of these applications is to detect intruders as they cross the boundary of a region or domain. For example, research efforts were made to extend the scalability of WSNs to the monitoring of international borders [10, 11]. Unlike the traditional *full coverage* [13, 17, 18] which requires an entire target region to be covered by the sensors, the *barrier coverage* [2, 3, 7, 8, 11] only seeks to cover the perimeter of the region to ensure that any intruders are detected as they cross the region border. Since barrier coverage requires fewer sensors, it is often preferable to

⋆ Chen and Wang's research was supported in part by NSF under Grant CCF-0916606. Li's research was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

⋆⋆ Corresponding author. This work was partially done while the author was visiting at IIIS at Tsinghua University.

full coverage. Because sensors have limited battery-supplied energy, it is desired to minimize their movements. In this paper, we study a linear barrier coverage problem where the barrier is for a (finite) line segment and the sensors are initially located on the line containing the barrier segment and allowed to move on the line. As discussed in the previous work [7, 8, 15] and shown in this paper, barrier coverage even for linear domains poses some challenging algorithmic issues. Also, our solutions may be used in solving more general problems. For example, if the barrier is sought for a simple polygon, then we may consider each of its edges separately and apply our algorithms to each edge.

In our problem, each sensor has a *sensing range* (or *range* for short) and we want to move the sensors to form a coverage for the barrier such that the maximum sensor movement is minimized.

1.1 Problem Definitions, Previous Work, and Our Results

Denote by $B = [0, L]$ the barrier that is a line segment from $x = 0$ to $x = L > 0$ on the x-axis. A set $S = \{s_1, s_2, \ldots, s_n\}$ of n mobile sensors are initially on the x-axis. Each sensor $s_i \in S$ has a range $r_i > 0$ and is located at the coordinate x_i. We assume $x_1 \leq x_2 \leq \cdots \leq x_n$. If a sensor s_i is at the position x', then we say s_i *covers* the interval $[x' - r_i, x' + r_i]$, called the *covering interval* of s_i. Our problem is to find a set of *destinations* on the x-axis, $\{y_1, y_2, \ldots, y_n\}$, for the sensors (i.e., for each $s_i \in S$, move s_i from x_i to y_i) such that each point on the barrier B is covered by at least one sensor and the maximum moving distance of the sensors (i.e., $\max_{1 \leq i \leq n}\{|x_i - y_i|\}$) is minimized. We call this problem the *barrier coverage on a line segment*, denoted by BCLS. We assume $2 \cdot \sum_{i=1}^{n} r_i \geq L$ (otherwise, a barrier coverage for B is not possible).

The *decision version* of BCLS is defined as follows. Given a value $\lambda \geq 0$, determine whether there is a *feasible solution* in which the moving distance of each sensor is at most λ. If the ranges of all sensors are the same (i.e., the r_i's are all equal), then we call it the *uniform case* of BCLS. When the sensors have arbitrary ranges, we call it the *general case*.

The BCLS problem has been studied before. The uniform case has been solved in $O(n^2)$ time [7]. An $O(n)$ time algorithm is also given in [7] for the decision version of the uniform case. However, it has been open whether the general case is solvable in polynomial time [7].

In this paper, we settle the open problem on the general BCLS by presenting an $O(n^2 \log n \log\log n)$ time algorithm. We also solve the decision version of the general BCLS in $O(n \log n)$ time. For the uniform case, we derive an $O(n \log n)$ time algorithm, improving the previous $O(n^2)$ time solution [7]; and further, if all sensors are initially on B, our algorithm runs in $O(n)$ time.

1.2 Related Work

A variation of the decision version of the general BCLS is shown to be NP-hard [7]. Additional results were also given in [7] for the case $2 \cdot \sum_{i=1}^{n} r_i < L$.

Mehrandish *et al.* [15] also considered the line segment barrier, but unlike the BCLS problem, they intended to use the minimum number of sensors to form a barrier coverage, which they proved to be NP-hard. But, if all sensors have the same range, polynomial time algorithms were possible [15]. Another study of the line segment barrier [8] aimed to minimize the sum of the moving distances of all sensors; this problem is NP-hard [8], but is solvable in polynomial time when all sensors have the same range [8]. In addition, Li *et al.* [12] considers the linear coverage problem which aims to set an energy for each sensor to form a coverage such that the cost of all sensors is minimized. There [12], the sensors are not allowed to move, and the more energy a sensor has, the larger the covering range of the sensor and the larger the cost of the sensor.

Bhattacharya *et al.* [2] studied a 2-D barrier coverage problem in which the barrier is a circle and the sensors, initially located inside the circle, are moved to the circle to form a coverage such that the maximum sensor movement is minimized; the ranges of the sensors are not explicitly specified but the destinations of the sensors are required to form a regular n-gon on the circle. Subsequent improvements of the results in [2] have been made [4, 16].

Some other barrier coverage problems have been studied. For example, Kumar *et al.* [11] proposed algorithms for determining whether a region is barrier covered after the sensors are deployed. They considered both the deterministic version (the sensors are deployed deterministically) and the randomized version (the sensors are deployed randomly), and aimed to determine a barrier coverage with high probability. Chen *et al.* [3] introduced a local barrier coverage problem in which individual sensors determine the barrier coverage locally.

2 An Overview of Our Approaches

Throughout the paper, for any problem we consider, let λ^* denote the maximum sensor movement in an optimal solution.

For the uniform BCLS, as shown in [7], a key property is that there always exists an *order preserving* optimal solution, i.e., the order of the sensors in the optimal solution is the same as that in the input. Based on this property, the previous $O(n^2)$ time algorithm [7] covers B from left to right; in each step, it picks the next sensor and re-balances the current maximum sensor movement. We take a very different approach. With the order preserving property, we determine a set Λ of candidate values for λ^* with $\lambda^* \in \Lambda$. Consequently, by using the decision algorithm, we can find λ^* in Λ. But, this approach may be inefficient since $|\Lambda| = \Theta(n^2)$. To reduce the running time, our strategy is not to compute the set Λ explicitly. Instead, we compute an element in Λ whenever we need it. A possible attempt would be to first find a sorted order for the elements of Λ or (implicitly) sort the elements of Λ, and then obtain λ^* by binary search. However, it seems not easy to (implicitly) sort the elements of Λ. Instead, based on several new observations, we manage to find a way to partition the elements of Λ into n sorted lists, each list containing $O(n)$ elements. Next, by using a technique called *binary search on sorted arrays* [5], we are able to find λ^* in Λ

in $O(n \log n)$ time. For the special case when all sensors are initially located on B, a key observation we make is that λ^* is precisely the maximum value of the candidate set Λ. Although $\Lambda = \Theta(n^2)$, based on new observations, its maximum value can be computed in $O(n)$ time. Due to the space limit, our algorithms for the uniform BCLS are omitted and can be found in the full version of this paper.

For the general BCLS, as indicated in [7], the order preserving property no longer holds. Consequently, our approach for the uniform case does not work. The main difficulty of this case is that we do not know the order of the sensors appeared in an optimal solution. Due to this difficulty, no polynomial time algorithm was known before for the general BCLS. To solve this problem, we first develop a greedy algorithm for the decision version of the general BCLS. After $O(n \log n)$ time preprocessing, our decision algorithm takes $O(n \log \log n)$ time for any value λ. If $\lambda \geq \lambda^*$, implying that there exists a feasible solution, then our decision algorithm can determine the order of sensors in a feasible solution for covering B. For the general BCLS, we seek to simulate the behavior of the decision algorithm on $\lambda = \lambda^*$. Although we do not know the value λ^*, our algorithm determines the same sensor order as it would be obtained by the decision algorithm on the value $\lambda = \lambda^*$. To this end, each step of the algorithm uses our decision algorithm as a decision procedure. The idea is somewhat similar to parametric search [6, 14], and here we "parameterize" our decision algorithm. However, unlike the typical parametric search [6, 14], our approach does not involve any parallel scheme and is practical.

For ease of exposition, we assume that initially no two sensors are located at the same position (i.e., $x_i \neq x_j$ for any $i \neq j$), and the covering intervals of any two different sensors do not share a common endpoint. Our algorithms can be easily generalized to the general situation.

In the following, we discuss the decision version of the general BCLS in Section 3. In Section 4, we present our algorithm for the general BCLS, which we refer to as the *optimization version* of the problem. Due to the space limit, some proofs are omitted and can be found in the full version of this paper.

For each sensor $s_i \in S$, we call the right (resp., left) endpoint of the covering interval of s_i the *right* (resp., *left*) *extension* of s_i. Each of the right and left extensions of s_i is an *extension* of s_i. Denote by $p(x')$ the point on the x-axis whose coordinate is x', and denote by $p^+(x')$ (resp., $p^-(x')$) a point to the right (resp., left) of $p(x')$ and infinitely close to $p(x')$. The concept of $p^+(x')$ and $p^-(x')$ is only used to explain the algorithms, and we never need to find such a point. Note that we can easily determine whether $\lambda^* = 0$, say, in $O(n \log n)$ time. Henceforth, we assume $\lambda^* > 0$.

3 The Decision Version of the General BCLS

Given any value λ, the decision version is to determine whether $\lambda^* \leq \lambda$. Below, we first explore some properties of a feasible solution for λ.

By a sensor *configuration*, we refer to a specification of where each sensor $s_i \in S$ is located. By this definition, the input is a configuration in which each sensor s_i is located at x_i. The *displacement* of a sensor in a configuration C is the distance between the position of the sensor in C and its original position in the input. A configuration C is a *feasible solution* for the distance λ if the sensors in C form a barrier coverage of B (i.e., the union of the covering intervals of the sensors in C contains B) and the displacement of each sensor is at most λ. In a feasible solution, a subset $S' \subseteq S$ is called a *solution set* if the sensors in S' form a barrier coverage; of course, S itself is also a solution set. A feasible solution may have multiple solution sets. A sensor s_i in a solution set S' is said to be *critical* with respect to S' if s_i covers a point on B that is not covered by any other sensor in S'. If every sensor in S' is critical, then S' is called a *critical set.*

Given any value λ, if $\lambda \geq \lambda^*$, then our decision algorithm will find a critical set and determine the order in which the sensors of the critical set will appear in a feasible solution for λ. Consider a critical set S^c. For each sensor $s \in S^c$, we call the set of points on B that are covered by s but not covered by any other sensor in S^c the *exclusive coverage* of s. The proof of Observation 1 is omitted.

Observation 1 *The exclusive coverage of each sensor in a critical set S^c is a continuous portion of the barrier B.*

For a critical set S^c in a feasible solution SOL, we define the *cover order* of the sensors in S^c as the order of these sensors in SOL such that their exclusive coverages are from left to right.

Observation 2 *The cover order of the sensors of a critical set S^c in a feasible solution SOL is consistent with the left-to-right order of the positions of these sensors in SOL. Further, the cover order is also consistent with the order of the* right *(resp., left) extensions of these sensors in SOL.*

Proof. Consider any two sensors s_i and s_j in S^c with ranges r_i and r_j, respectively. Without loss of generality, assume s_i is to the left of s_j in the cover order, i.e., the exclusive coverage of s_i is to the left of that of s_j in SOL. Let y_i and y_j be the positions of s_i and s_j in SOL, respectively. To prove the observation, it suffices to show $y_i < y_j$, $y_i + r_i < y_j + r_j$, and $y_i - r_i < y_j - r_j$.

Let p be a point in the exclusive coverage of s_j. We also use p to denote its coordinate on the x-axis. Then p is not covered by s_i, implying either $p > y_i + r_i$ or $p < y_i - r_i$. But, the latter case cannot hold (otherwise, the exclusive coverage of s_i would be to the right of that of s_j). Since p is covered by s_j, we have $p \leq y_j + r_j$. Therefore, $y_i + r_i < p \leq y_j + r_j$. By using a symmetric argument, we can also prove $y_i - r_i < y_j - r_j$ (we omit the details). Clearly, the two inequalities $y_i + r_i < y_j + r_j$ and $y_i - r_i < y_j - r_j$ imply $y_i < y_j$. The observation thus holds.

An interval I of B is called a *left-aligned interval* if the left endpoint of I is at 0 (i.e., I is of the form $[0, x']$ or $[0, x')$). A set of sensors is said to be in *attached positions* if the union of their covering intervals is a continuous interval of the x-axis whose length is equal to the sum of the lengths of these covering intervals. In the sequel, we describe our algorithm.

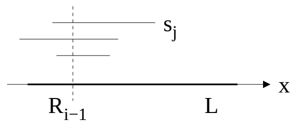

Fig. 1. The set S_{i1} consists of the three sensors whose covering intervals are shown, and $s_{g(i)}$ is s_j.

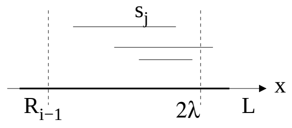

Fig. 2. The set S_{i2} consists of the three sensors whose covering intervals are shown, and $s_{g(i)}$ is s_j if $S_{i1} = \emptyset$.

3.1 The Algorithm Description

Initially, we move all sensors of S to the right by the distance λ, i.e., for each $1 \leq i \leq n$, we move s_i to the position $x'_i = x_i + \lambda$. Let C_0 denote the resulting configuration. Clearly, there is a feasible solution for λ if and only if we can move the sensors in C_0 to the left by at most 2λ to form a coverage of B. Thus, henceforth we only need to consider moving the sensors to the left. Recall that we have assumed that the extensions of any two distinct sensors are different; hence in C_0, the extensions of all sensors are also different.

Our algorithm takes a greedy approach. It seeks to find sensors to cover B from left to right, in at most n steps. If $\lambda \geq \lambda^*$, the algorithm will end up with a critical set S^c of sensors along with the destinations for all these sensors.

In step i (initially, $i = 1$), using the configuration C_{i-1} and based on certain criteria, we find a sensor $s_{g(i)}$ and determine its destination $y_{g(i)}$, where $g(i)$ is the index of the sensor in S and $y_{g(i)} \in [x'_{g(i)} - 2\lambda, x'_{g(i)}]$. We then move the sensor $s_{g(i)}$ to $y_{g(i)}$ to obtain a new configuration C_i from C_{i-1} (if $y_{g(i)} = x'_{g(i)}$, C_i is simply C_{i-1}). Let $R_i = y_{g(i)} + r_{g(i)}$ (i.e., the right extension of $s_{g(i)}$ in C_i). Assume $R_0 = 0$. Let $S_i = S_{i-1} \cup \{s_{g(i)}\}$ ($S_0 = \emptyset$ initially). We will show that the sensors in S_i together cover the left-aligned interval $[0, R_i]$. If $R_i \geq L$, we have found a feasible solution with a critical set $S^c = S_i$, and terminate the algorithm. Otherwise, we proceed to step $i + 1$. Further, it is possible that a desired sensor $s_{g(i)}$ cannot be found, in which case we terminate the algorithm and report $\lambda < \lambda^*$. Below we give the details, and in particular, discuss how to determine the sensor $s_{g(i)}$ in each step.

We first discuss a technical issue. Suppose there is a sensor s_t with its right extension at 0 in C_0. We claim s_t cannot be in a critical set of a feasible solution if $\lambda^* \leq \lambda$. Indeed, assume to the contrary that s_t is in a critical set S^c. Then $p(0)$ is the only point on B that can be covered by s_t. Since $L > 0$, there must be another sensor in S^c that also covers $p(0)$ (otherwise, no sensor in S^c would cover the point $p^+(0)$). Hence, s_t is not critical with respect to S^c, a contradiction.

Initially, we have $R_0 = 0$ and $S_0 = \emptyset$. Consider the i-th step of the algorithm with $i \geq 1$. We determine the sensor $s_{g(i)}$, as follows. Define $S_{i1} = \{s_j \mid x'_j - r_j \leq R_{i-1} < x'_j + r_j\}$ (see Fig. 1), i.e., S_{i1} is the set of sensors covering the point $p^+(R_{i-1})$ in the configuration C_{i-1}. Note that any sensor in S_{i1} covers the point $p(R_{i-1})$ in C_{i-1}. If $S_{i1} \neq \emptyset$, we choose the sensor in S_{i1} with the largest right extension as $s_{g(i)}$ and let $y_{g(i)} = x'_{g(i)}$. Otherwise, let S_{i2} be the set of sensors whose left extensions are larger than R_{i-1} and at most $R_{i-1} + 2\lambda$. If $S_{i2} = \emptyset$, we terminate the algorithm and report $\lambda < \lambda^*$. Otherwise, we choose the sensor in S_{i2} with the smallest right extension as $s_{g(i)}$ (e.g., s_j in Fig. 2), and let $y_{g(i)} = R_{i-1} + r_{g(i)}$. If the algorithm is not terminated, we move $s_{g(i)}$ to $y_{g(i)}$ and obtain a new configuration C_i. Let $S_i = S_{i-1} \cup \{s_{g(i)}\}$. Let R_i be the right extension of $s_{g(i)}$ in C_i. If $R_i \geq L$, we have found a feasible solution C_i with the critical set S_i. Otherwise, we proceed to step $i + 1$.

Since there are n sensors in S, the algorithm is terminated in at most n steps.

3.2 The Algorithm Correctness and Implementation

Based on our algorithm description, we have the following lemma.

Lemma 1. *At the end of step i, suppose the algorithm produces the set S_i and the configuration C_i; then S_i and C_i have the following properties. (a) The interval on B covered by the sensors in S_i is $[0, R_i]$. (b) For each $1 < j \leq i$, the right extension of $s_{g(j)}$ is larger than that of $s_{g(j-1)}$. (c) For each $1 \leq j \leq i$, $s_{g(j)}$ is the only sensor in S_i that covers the point $p^+(R_{j-1})$ (with $R_0 = 0$). (d) For each sensor $s_{g(j)} \in S_i$ with $1 \leq j \leq i$, it is either from S_{j1} or S_{j2}. If $s_{g(j)}$ is from S_{j1}, then its position in C_i is the same as that in C_0; otherwise, its left extension is at R_{j-1}, and $s_{g(j)}$ and $s_{g(j-1)}$ are in attached positions if $j > 1$.*

The proof of Lemma 1 is omitted. At its termination, our algorithm either reports $\lambda \geq \lambda^*$ or $\lambda < \lambda^*$. Suppose in step i, our algorithm reports $\lambda \geq \lambda^*$. Then according to the algorithm, it must be $R_i \geq L$. By Lemma 1(a) and (c), C_i is a feasible solution and S_i is a critical set. Further, by Lemma 1(b) and Observation 2, the cover order of the sensors in S_i is $s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}$.

Next, we show that if the algorithm reports $\lambda < \lambda^*$, then there is no feasible solution for λ. This is almost an immediate consequence of Lemma 2.

Lemma 2. *Suppose S'_i is the set of sensors in the configuration C_i whose right extensions are at most R_i. Then the interval $[0, R_i]$ is the largest possible left-aligned interval that can be covered by the sensors of S'_i such that the displacement of each sensor in S'_i is at most λ.*

To prove Lemma 2, the key is to prove the following. If C is a configuration for the sensors of S'_i such that a left-aligned interval $[0, x']$ is covered by the sensors of S'_i, then there always exists a configuration C^* for S'_i in which the interval $[0, x']$ is still covered by the sensors of S'_i and for each $1 \leq j \leq i$, the position of the sensor $s_{g(j)}$ in C^* is $y_{g(j)}$, where $g(j)$ and $y_{g(j)}$ are the values computed by our algorithm. We omit the proof for this.

Suppose our algorithm reports $\lambda < \lambda^*$ in step i. Then according to the algorithm, $R_{i-1} < L$ and both S_{i1} and S_{i2} are $\emptyset$. Let S'_{i-1} be the set of sensors whose right extensions are at most R_{i-1} in C_{i-1}. Since both S_{i1} and S_{i2} are $\emptyset$, no sensor in $S \setminus S'_{i-1}$ can cover any point to the left of the point $p^+(R_{i-1})$ (and including $p^+(R_{i-1})$). By Lemma 2, $[0, R_{i-1}]$ is the largest left-aligned interval that can be covered by the sensors of S'_{i-1}. Hence, the sensors in S cannot cover the interval $[0, p^+(R_{i-1})]$. Due to $R_{i-1} < L$, we have $[0, p^+(R_{i-1})] \subseteq [0, L]$; thus the sensors of S cannot cover $B = [0, L]$. In other words, there is no feasible solution for the distance λ. This establishes the correctness of our algorithm.

We briefly discuss the implementation of our algorithm. Our algorithm needs to maintain two sets of sensors, S_{i1} and S_{i2}. For this purpose, in the preprocessing, we sort the $2n$ extensions of all sensors by the x-coordinate, and move each sensor $s_i \in S$ to x'_i to produce the initial configuration C_0. During the algorithm, we sweep along the x-axis and maintain S_{i1} and S_{i2}, respectively. During the sweeping, we need to perform the sensor insertions and deletions on the two sets. In addition, we need a *search* operation on S_{i1} for finding the sensor in S_{i1} with the largest right extension, and a *search* operation on S_{i2} for finding the sensor in S_{i2} with the smallest right extension. There are $O(n)$ insertion, deletion, and search operations in the entire algorithm.

If we use a balanced binary search tree to store each of these two sets in which the right extensions of the sensors are used as keys, then the algorithm takes $O(n \log n)$ time. Another way is to use an integer data structure (e.g., van Emde Boas tree [9]), as follows. In the preprocessing, we also sort the sensors by their right extensions, and for each sensor, assign the integer k to it as its key if the sensor is the k-th one in the above sorted order. Thus, all such keys form an integer set $\{1, 2, \dots, n\}$. By using the van Emde Boas tree [9], each operation takes only $O(\log \log n)$ time. Thus, after $O(n \log n)$ time preprocessing, the algorithm takes $O(n \log \log n)$ time for each value λ. Although using the integer data structure does not change the overall running time of our decision algorithm, it helps our optimization algorithm in Section 4 to run faster.

Theorem 1. *After $O(n \log n)$ time preprocessing, for any λ, we can determine whether $\lambda^* \leq \lambda$ in $O(n \log \log n)$ time; further, if $\lambda^* \leq \lambda$, we can compute a feasible solution in $O(n \log \log n)$ time.*

Our optimization algorithm in Section 4 also needs to determine whether λ^* is strictly less than λ (i.e., $\lambda^* < \lambda$) for any λ. By modifying our algorithm for Theorem 1, we have the following Theorem 2 whose proof is omitted.

Theorem 2. *After $O(n \log n)$ time preprocessing, for any value λ, we can determine whether $\lambda^* < \lambda$ in $O(n \log \log n)$ time.*

Theorems 1 and 2 together lead to the following corollary.

Corollary 1. *After $O(n \log n)$ time preprocessing, for any value λ, we can determine whether $\lambda^* = \lambda$ in $O(n \log \log n)$ time.*

4 The Optimization Version of the General BCLS

In this section, we discuss the optimization version of the general BCLS problem. We first give an algorithm overview. In the following discussion, the "decision algorithm" refers to our algorithm for Theorem 1, unless otherwise stated.

Recall that given any value λ, step i of our decision algorithm determines the sensor $s_{g(i)}$ and obtains the set $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\}$, in this order, which we also call the *cover order* of the sensors in S_i. In our optimization algorithm, we often use λ as a variable. Thus, $S_i(\lambda)$ (resp., $R_i(\lambda)$, $s_{g(i)}(\lambda)$, and $C_i(\lambda)$) refers to the corresponding S_i (resp., R_i, $s_{g(i)}$, and C_i) obtained by running our decision algorithm on the specific value λ. Denote by C_I the configuration of the input.

Our optimization algorithm takes at most n steps. Step i receives an interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ and a sensor set $S_{i-1}(\lambda^*)$, with the *algorithm invariants* that $\lambda^* \in (\lambda^1_{i-1}, \lambda^2_{i-1})$ (although we do not know the value λ^*) and for any value $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$, we have $S_{i-1}(\lambda) = S_{i-1}(\lambda^*)$ and their cover orders are the same. Step i either finds the value λ^* or determines a sensor $s_{g(i)}(\lambda^*)$. The interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ will shrink to a new interval $(\lambda^1_i, \lambda^2_i) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$ and we also obtain the set $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. Each step can be performed in $O(n \log n \log\log n)$ time. The details of the algorithm are given below.

Initially, let $S_0(\lambda^*) = \emptyset$, $R_0(\lambda^*) = 0$, $\lambda^1_0 = 0$, and $\lambda^2_0 = +\infty$.

Consider a general step i for $i \geq 1$ and we have the interval $(\lambda^1_{i-1}, \lambda^2_{i-1})$ and the set $S_{i-1}(\lambda^*)$. While discussing the algorithm, we will also prove inductively the following lemma about the function $R_i(\lambda)$ with variable $\lambda \in (\lambda^1_i, \lambda^2_i)$.

Lemma 3. *(a) The function $R_i(\lambda)$ for $\lambda \in (\lambda^1_i, \lambda^2_i)$ is a line segment of slope* 1 *or* 0*. (b) We can compute the function $R_i(\lambda)$ for $\lambda \in (\lambda^1_i, \lambda^2_i)$ explicitly in $O(n)$ time. (c) $R_i(\lambda) < L$ for any $\lambda \in (\lambda^1_i, \lambda^2_i)$.*

In the base case for $i = 0$, the statement of Lemma 3 obviously holds. We assume the lemma statement holds for $i - 1$. We will show that after step i with $i \geq 1$, the lemma statement holds for i, and thus the lemma will be proved.

Again, in step i, we need to determine the sensor $s_{g(i)}(\lambda^*)$ and let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. We will also obtain an interval $(\lambda^1_i, \lambda^2_i)$ such that $\lambda^* \in (\lambda^1_i, \lambda^2_i) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$ and for any $\lambda \in (\lambda^1_i, \lambda^2_i)$, $S_i(\lambda) = S_i(\lambda^*)$ holds (with the same cover order). The details are given below. We assume that we already compute explicitly the function $R_{i-1}(\lambda)$ for $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$, which takes $O(n)$ time by our assumption that the statement of Lemma 3 holds for $i - 1$.

To find the sensor $s_{g(i)}(\lambda^*)$, we first determine the set $S_{i1}(\lambda^*)$. Recall that $S_{i1}(\lambda^*)$ consists of all sensors covering the point $p^+(R_{i-1}(\lambda^*))$ in the configuration $C_{i-1}(\lambda^*)$. For each sensor in $S \setminus S_{i-1}(\lambda^*)$, its position in the configuration $C_{i-1}(\lambda)$ with respect to $\lambda \in (\lambda^1_{i-1}, \lambda^2_{i-1})$ is a function of slope 1. As λ increases in $(\lambda^1_{i-1}, \lambda^2_{i-1})$, by our assumption that Lemma 3(a) holds for $i-1$, the function $R_{i-1}(\lambda)$ is a line segment of slope 1 or 0. If $R_{i-1}(\lambda)$ is of slope 1, then the relative position of $R_{i-1}(\lambda)$ in $C_{i-1}(\lambda)$ does not change and thus the set $S_{i1}(\lambda)$ does not change; if the function $R_{i-1}(\lambda)$ is of slope 0, then the relative position of $R_{i-1}(\lambda)$ in $C_{i-1}(\lambda)$ is monotonically moving to the left. Hence, there are $O(n)$ values for λ in $(\lambda^1_{i-1}, \lambda^2_{i-1})$ that can incur some changes to the set $S_{i1}(\lambda)$ and each such value corresponds to a sensor extension; further, these values can be easily determined in $O(n \log n)$ time by a simple sweeping process (we omit the discussion of it). Let Λ_{i1} be the set of all these λ values. Let Λ_{i1} also contain both λ^1_{i-1} and λ^2_{i-1}, and thus, λ^1_{i-1} and λ^2_{i-1} are the smallest and largest values in Λ_{i1}, respectively.

We sort the values in Λ_{i1}. For any two consecutive values $\lambda_1 < \lambda_2$ in the sorted Λ_{i1}, the set $S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$ is the same. By using binary search on the sorted Λ_{i1} and our decision algorithm, we determine (in $O(n \log n \log\log n)$ time) the two consecutive values λ_1 and λ_2 in Λ_{i1} such that $\lambda_1 < \lambda^* \leq \lambda_2$. Further, by Corollary 1, we determine whether $\lambda^* = \lambda_2$. If $\lambda^* = \lambda_2$, then we are done. Otherwise, based on our discussion above, $S_{i1}(\lambda^*) = S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$. Thus, to compute $S_{i1}(\lambda^*)$, we can pick an arbitrary λ in (λ_1, λ_2) and find $S_{i1}(\lambda)$ in the same way as in our decision algorithm. Hence, $S_{i1}(\lambda^*)$ can be easily found in $O(n \log n)$ time. Note that $\lambda^* \in (\lambda_1, \lambda_2) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$.

If $S_{i1}(\lambda^*) \neq \emptyset$, then $s_{g(i)}(\lambda^*)$ is the sensor in $S_{i1}(\lambda^*)$ with the largest right extension. An obvious observation is that for any $\lambda \in (\lambda_1, \lambda_2)$, the sensor in $S_{i1}(\lambda^*)$ with the largest right extension is the same, which can be easily found. We let $\lambda^1_i = \lambda_1$ and $\lambda^2_i = \lambda_2$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. The algorithm invariants hold. Further, as λ increases in (λ_1, λ_2), the right extension of $s_{g(i)}(\lambda)$, which is $R_i(\lambda)$, increases by the same amount. That is, the function $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1. Therefore, we can compute $R_i(\lambda)$ on (λ_1, λ_2) explicitly in constant time. This also shows Lemma 3(a) and (b) hold for i.

Because the function $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1, there are three cases depending on the values $R_i(\lambda)$ and L: (1) $R_i(\lambda) < L$ for any $\lambda \in (\lambda_1, \lambda_2)$, (2) $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$, and (3) there exists a unique value $\lambda' \in (\lambda_1, \lambda_2)$ such that $R_i(\lambda') = L$. For Case (1), we proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Clearly, the algorithm invariants hold and Lemma 3(c) holds for i. For Case (2), the next lemma shows that it actually cannot happen due to $\lambda^* \in (\lambda_1, \lambda_2)$.

Lemma 4. *It is not possible that $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$.*

Proof. Assume to the contrary that $R_i(\lambda) > L$ for any $\lambda \in (\lambda_1, \lambda_2)$. Since $\lambda^* \in (\lambda_1, \lambda_2)$, let λ'' be any value in (λ_1, λ^*). Due to $\lambda'' \in (\lambda_1, \lambda_2)$, we have $R_i(\lambda'') > L$. But this would implies that we have found a feasible solution where the displacement of each sensor is at most $\lambda'' \leq \lambda^*$, incurring contradiction.

For the Case (3), since $R_i(\lambda)$ on (λ_1, λ_2) is a line segment of slope 1, we can determine in constant time the unique value $\lambda' \in (\lambda_1, \lambda_2)$ such that $R_i(\lambda') = L$. Clearly, $\lambda^* \leq \lambda'$. By Corollary 1, we determine whether $\lambda^* = \lambda'$. If $\lambda^* = \lambda'$, then we are done; otherwise, we have $\lambda^* \in (\lambda_1, \lambda')$ and update λ^2_i to λ'. We proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Again, the algorithm invariants hold and Lemma 3(c) holds for i.

If $S_{i1}(\lambda^*) = \emptyset$, then we need to compute $S_{i2}(\lambda^*)$. For any $\lambda \in (\lambda_1, \lambda_2)$, the set $S_{i2}(\lambda)$ consists of all sensors whose left extensions are larger than $R_{i-1}(\lambda)$ and at most $R_{i-1}(\lambda)+2\lambda$ in the configuration $C_{i-1}(\lambda)$. Recall that the function $R_{i-1}(\lambda)$ on $(\lambda^1_{i-1}, \lambda^2_{i-1})$ is linear with slope 1 or 0. Due to $(\lambda_1, \lambda_2) \subseteq (\lambda^1_{i-1}, \lambda^2_{i-1})$, the linear function $R_{i-1}(\lambda) + 2\lambda$ on (λ_1, λ_2) is of slope 3 or 2. Again, as λ increases, the position of each sensor in $S \setminus S_{i-1}(\lambda^*)$ in $C_{i-1}(\lambda)$ is a linear function of slope 1. Therefore, there are $O(n)$ λ values in (λ_1, λ_2) each of which incurs some change to the set $S_{i2}(\lambda)$ and each such λ value corresponds to a sensor extension. Further, these values can be easily determined in $O(n \log n)$ time by a sweeping process (we omit the discussion for this). (Actually, as λ increases, the size of the set $S_{i2}(\lambda)$ is monotonically increasing.) Let Λ_{i2} denote the set of these λ values, and let Λ_{i2} contain both λ_1 and λ_2. Again, $|\Lambda_{i2}| = O(n)$. We sort the values in Λ_{i2}. Using binary search on the sorted Λ_{i2} and our decision algorithm, we determine (in $O(n \log n \log \log n)$ time) the two consecutive values λ'_1 and λ'_2 in Λ_{i2} such that $\lambda'_1 < \lambda^* \leq \lambda'_2$. Further, by Corollary 1, we determine whether $\lambda^* = \lambda'_2$. If $\lambda^* = \lambda'_2$, then we are done. Otherwise, $S_{i2}(\lambda^*) = S_{i2}(\lambda)$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, which can be easily found. Note that $\lambda^* \in (\lambda'_1, \lambda'_2) \subseteq (\lambda_1, \lambda_2)$.

After obtaining $S_{i2}(\lambda^*)$, $s_{g(i)}(\lambda^*)$ is the sensor in $S_{i2}(\lambda^*)$ with the smallest right extension. As before, the sensor in $S_{i2}(\lambda)$ with the smallest right extension is the same for any $\lambda \in (\lambda'_1, \lambda'_2)$. Thus, $s_{g(i)}(\lambda^*)$ can be easily determined. We let $\lambda^1_i = \lambda'_1$ and $\lambda^2_i = \lambda'_2$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. The algorithm invariants hold. Further, we examine the function $R_i(\lambda)$, i.e., the right extension of $s_{g(i)}(\lambda)$ in the configuration $C_i(\lambda)$, as λ increases in (λ'_1, λ'_2). Since $s_{g(i-1)}(\lambda^*)$ and $s_{g(i)}(\lambda^*)$ are always in attached positions in this case, for any $\lambda \in (\lambda'_1, \lambda'_2)$, we have $R_i(\lambda) = R_{i-1}(\lambda) + 2r_{g(i)}$. Thus, the function $R_i(\lambda)$ is a vertical shift of $R_{i-1}(\lambda)$ by the distance $2r_{g(i)}$. Because we already know explicitly the function $R_{i-1}(\lambda)$ for $\lambda \in (\lambda'_1, \lambda'_2)$, which is a line segment of slope 1 or 0, the function $R_i(\lambda)$ can be computed in constant time, which is also a line segment of slope 1 or 0. Note that this shows that Lemma 3(a) and (b) hold for i.

Similarly to the case when $S_{i1}(\lambda^*) \neq \emptyset$, since the function $R_i(\lambda)$ in (λ'_1, λ'_2) is a line segment of slope 1 or 0, there are three cases depending on the values $R_i(\lambda)$ and L: (1) $R_i(\lambda) < L$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, (2) $R_i(\lambda) > L$ for any $\lambda \in (\lambda'_1, \lambda'_2)$, and (3) there exists a unique value $\lambda'' \in (\lambda'_1, \lambda'_2)$ such that $R_i(\lambda'') = L$. For Case (1), we proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Clearly, the algorithm invariants hold and Lemma 3(c) holds for i. Similarly to Lemma 4, Case (2) cannot happen due to $\lambda^* \in (\lambda'_1, \lambda'_2)$. For the Case (3), since $R_i(\lambda)$ in (λ'_1, λ'_2) is a line segment of slope 1 or 0, we can compute in constant time the unique value $\lambda'' \in (\lambda'_1, \lambda'_2)$ such that $R_i(\lambda'') = L$. Clearly, $\lambda^* \leq \lambda''$. By Corollary 1, we determine whether $\lambda^* = \lambda''$. If $\lambda^* = \lambda''$, we are done; otherwise, we have $\lambda^* \in (\lambda'_1, \lambda'')$ and update λ^2_i to λ''. We proceed to the next step, along with the interval $(\lambda^1_i, \lambda^2_i)$. Again, the algorithm invariants and Lemma 3(c) hold for i.

This finishes the discussion of step i of our algorithm. Note that in each case where we proceed to the next step, Lemma 3 holds for i, and thus Lemma 3 has been proved. The running time of step i is clearly bounded by $O(n \log n \log \log n)$.

In at most n steps, the algorithm will stop and find the value λ^*. Then by applying our decision algorithm on $\lambda = \lambda^*$, we finally produce an optimal solution in which the displacement of every sensor is at most λ^*. Since each step takes $O(n \log n \log \log n)$ time, the total time of the algorithm is $O(n^2 \log n \log \log n)$.

Theorem 3. *The general BCLS problem is solvable in $O(n^2 \log n \log \log n)$ time.*

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. Computer Networks 38(4), 393–422 (2002)

2. Bhattacharya, B., Burmester, B., Hu, Y., Kranakis, E., Shi, Q., Wiese, A.: Optimal movement of mobile sensors for barrier coverage of a planar region. Theoretical Computer Science 410(52), 5515–5528 (2009)
3. Chen, A., Kumar, S., Lai, T.: Designing localized algorithms for barrier coverage. In: Proc. of the 13th Annual ACM International Conference on Mobile Computing and Networking. pp. 63–73 (2007)
4. Chen, D., Tan, X., Wang, H., Wu, G.: Optimal point movement for covering circular regions. arXiv:1107.1012v1 (2011)
5. Chen, D., Wang, C., Wang, H.: Representing a functional curve by curves with fewer peaks. Discrete and Computational Geometry (DCG) 46(2), 334–360 (2011)
6. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. Journal of the ACM 34(1), 200–208 (1987)
7. Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: 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. Lecture Notes in Computer Science, vol. 5793, pp. 194–212. Springer (2009)
8. Czyzowicz, J., Kranakis, E., Krizanc, D., Lambadaris, I., Narayanan, L., Opatrny, J., Stacho, L., Urrutia, J., Yazdani, M.: 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. Lecture Notes in Computer Science, vol. 6288, pp. 29–42. Springer (2010)
9. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Theory of Computing Systems 10(1), 99–127 (1977)
10. Hu, S.: 'Virtual Fence' along border to be delayed. *Washington Post* (February 28, 2008)
11. Kumar, S., Lai, T., Arora, A.: Barrier coverage with wireless sensors. Wireless Networks 13(6), 817–834 (2007)
12. Li, M., Sun, X., Zhao, Y.: Minimum-cost linear coverage by sensors with adjustable ranges. In: Proc. of the 6th International Conference on Wireless Algorithms, Systems, and Applications. pp. 25–35 (2011)
13. Li, X., Frey, H., Santoro, N., Stojmenovic, I.: Localized sensor self-deployment with coverage guarantee. ACM SIGMOBILE Mobile Computing and Communications Review 12(2), 50–52 (2008)
14. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. Journal of the ACM 30(4), 852–865 (1983)
15. Mehrandish, M., Narayanan, L., Opatrny, J.: Minimizing the number of sensors moved on line barriers. In: Proc. of IEEE Wireless Communications and Networking Conference (WCNC). pp. 653–658 (2011)
16. Tan, X., Wu, G.: New algorithms for barrier coverage with mobile sensors. In: Proc. of the 4th International Workshop on Frontiers in Algorithmics. Lecture Notes in Computer Science, vol. 6213, pp. 327–338. Springer-Verlag (2010)
17. Yang, S., Li, M., Wu, J.: Scan-based movement-assisted sensor deployment methods in wireless sensor networks. IEEE Trans. Parallel Distrib. Syst. 18(8), 1108–1121 (2007)
18. Zou, Y., Chakrabarty, K.: A distributed coverage and connectivity-centric technique for selecting active nodes in wireless sensor networks. IEEE Trans. Comput. 54(8), 978–991 (2005)