Algorithms on Moving Sensors for Barrier Coverage of a Line Segment and a Simple Cycle *

Danny Z. Chen† Yan Gu‡ Haitao Wang§

Abstract

In this paper, we study the problem of moving sensors on a line to cover a barrier represented by a segment of the line such that the maximum moving distance of the sensors is minimized. Previously, it has been open whether this problem is solvable in polynomial time. We answer this open question positively by giving an $O(n^2 \log n \log \log n)$ time algorithm for $n$ sensors. Since this is the first-known polynomial time algorithm for this problem, our results and techniques may be useful for other related problems as well. For the special case where all sensors have the same sensing range, the previously best algorithm for it takes $O(n)$ time and we present an improved algorithm of $O(n \log n)$ time; further, if all sensors are initially located on the barrier, our algorithm takes only $O(n)$ time. In addition, we extend our techniques to the cycle version of the problem where the barrier is represented by a simple cycle and the sensors are only allowed to move along the cycle. If all sensors have the same sensing range, we solve the cycle version in $O(n)$ time.

1 Introduction

A Wireless Sensor Network (WSN) has a large number of sensors which monitor some surrounding environmental phenomena [1]. Intrusion detection and border surveillance constitute a major application category for WSNs. A major goal in these applications is to detect intruders as they cross the boundary of a region. For example, research efforts have been under way to extend the scalability of WSN to the monitoring of international borders [10, 11]. Unlike the traditional full coverage [12, 17, 18] which requires the full region to be covered by the sensors, the barrier coverage [2, 3, 8, 9, 11] only covers the perimeter of the region to guarantee that any intruders can be detected as they cross the region border. Since barrier coverage requires fewer sensors, it is preferable to full coverage. Because sensors have limited battery-supplied energy, we wish to minimize their movements. In this paper, we study a one dimensional barrier coverage problem where the barrier is represented as a (finite) line segment and sensors are initially located on the line containing the barrier and allowed to move on the line. As discussed in the previous work [8, 9, 14] as well as illustrated in this paper, the one dimensional barrier coverage contains many challenging algorithmic issues. Further, our solutions may also be useful for solving more general problems. For example, if the barrier is represented as a simple polygon, we can consider each of its edges separately and thus apply our algorithms on each edge. In our problem, each sensor has a sensing range (or range for short) and we wish to move the sensors to form a coverage for the barrier such that the maximum sensor movement is minimized. We present efficient algorithms for this problem, which improve the previous work and resolve an open problem. Further, we also extend our techniques to the cycle version where the barrier is represented as a simple cycle and the sensors are only allowed to move along the cycle.

*This research was supported in part by NSF under Grant CCF-0916606.
†Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. E-mail: {dchen, hwang6}@nd.edu.
‡Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: henryy321@gmail.com
§Corresponding author.
1.1 Problem Definitions 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 located on the \( x \)-axis. For each sensor \( s_i \in S \), it has a range \( r_i > 0 \) and denote by \( x_i \) the coordinate of \( s_i \) on the \( x \)-axis. We assume \( x_1 \leq x_2 \leq \cdots \leq x_n \). For each sensor \( s_i \in S \), if it is at the position \( x' \), we say the sensor \( s_i \) covers the interval \([x'-r_i, x'+r_i]\) and the interval is called the covering interval of \( s_i \). The goal of the problem is to find a set of destinations \( \{y_1, y_2, \ldots, y_n\} \) for the sensors (i.e., for each sensor \( s_i \in S \), move \( s_i \) 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 the problem the barrier coverage on a line segment and denote it by BCLS. In this paper we assume \( 2 \cdot \sum_{i=1}^{n} r_i \geq L \) since otherwise it would not be possible to form a barrier coverage.

The decision version of BCLS is defined as follows. Given a value \( \lambda \), determine whether there is a feasible solution in which the moving distance of each sensor is at most \( \lambda \). The decision version characterizes the problem model where the sensors have a limited energy and we want to know whether their energy is sufficient to form a barrier coverage.

If the ranges of all sensors are the same, we refer to it as the uniform case of BCLS; otherwise we refer it as the general case.

The BCLS problem has been studied before. The uniform case has been solved in \( O(n^2) \) time [8]. An \( O(n) \) time algorithm is also given in [8] for the decision version of the uniform case. However, it has been open whether the general case is solvable in polynomial time [8].

In this paper, we answer the open question positively by developing an \( O(n^2 \log n \log \log n) \) time algorithm for the general BCLS. We also solve its decision version in \( O(n \log n) \) time. Since this is a very basic problem about sensors and intervals and our algorithm is the first-known polynomial time solution for it, we expect our results and techniques may be useful for other related problems. Furthermore, we derive an \( O(n \log n) \) time algorithm for the uniform case, improving the \( O(n^2) \) time previous work [8], and if all sensors are initially on \( B \), our algorithm runs in \( O(n) \) time.

In addition, we also consider the simple cycle barrier coverage where the barrier is represented as a simple cycle and all \( n \) sensors are initially on the cycle and are allowed to move only along the cycle. The goal is to move all sensors to form a barrier coverage and minimize the maximum movement. To the best of our knowledge, we have not found any previous work on this problem. If all sensors have the same range, we solve the problem in \( O(n) \) time.

1.2 Related Work

Besides the results mentioned above, an \( O(n) \) time 2-approximation algorithm for the uniform BCLS was also given in [8] and a variation of the decision version of the general BCLS is shown to be NP-hard [8]. Additional results were also given in [8] for the case \( 2 \cdot \sum_{i=1}^{n} r_i < L \).

Mehrandish et al. [14] also considered the line segment barrier, but unlike our problem, their goal is to use the minimum number of sensors to form a barrier coverage. Mehrandish et al. [14] proved the problem is NP-Hard. But if all sensors have the same range, polynomial time algorithms were given [14]. The line segment barrier was also studied in [9], where the goal is to minimize the sum of the moving distances of all sensors. Similarly, the general problem is NP-hard [9], but it is solvable in polynomial time if all sensors have the same range.

Bhattacharya et al. [2] studied a two-dimensional barrier coverage problem in which the barrier is a circle and the sensors are given inside the circle and the goal is to move the sensors to the circle to form a coverage such that the maximum sensor movement is minimized. There, the ranges of the
sensors are not explicitly set but the destinations of the sensors are required to form a regular $n$-gon on the circle. Subsequent improvements have been done later in [4, 15]. In addition, Bhattacharya et al. [2] also presented some results on the corresponding min-sum version for minimizing the sum of the moving distances of all sensors; further improvement is also given in [4, 15].

There are other barrier coverage problems that have been studied as well. 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 where the sensors are deployed deterministically and the randomized version where the sensors are deployed randomly and the goal is to compute a barrier coverage with high probability. Chen [3] introduced the local barrier coverage in which individual sensors determine the barrier coverage locally.

1.3 An Overview of Our Approaches

For any problem, let $\lambda^*$ denote the maximum sensor movement in an optimal solution.

For the uniform BCLS, as discussed in [8], a good property is that there always exists an order preserving optimal solution $OPT$. Namely, the order of the sensors in $OPT$ is the same as that in the input. Based on this property, the previous $O(n^2)$ time algorithm [8] tries to cover $B$ from left to right; each step picks the next sensor and re-balances the current maximum sensor movement. In this paper, we use a completely different approach. Based on the order preserving property, we determine a set $\Lambda$ of candidate values for $\lambda^*$ such that $\lambda^* \in \Lambda$. Consequently, by using the decision algorithm we can find $\lambda^*$ in $\Lambda$. However, this approach is inefficient since $|\Lambda| = \Theta(n^2)$. To reduce the running time, our strategy is that we do not compute the set $\Lambda$ explicitly. Instead, we compute an element in $\Lambda$ whenever we need it. Our first attempt is that if we can find a sorted order for the elements in $\Lambda$ or (implicitly) sort the elements in $\Lambda$, then $\lambda^*$ can be found efficiently by binary search. However, it seems not easy to (implicitly) sort the elements of $\Lambda$. Instead, based on new observations, we manage to partition the elements in $\Lambda$ into $n$ sorted lists and each list contains $O(n)$ elements. Consequently, by using a technique called binary search on sorted arrays [5], we are able to find $\lambda^*$ in $\Lambda$ in $O(n \log n)$ time. For the special case where all sensors are initially located on $B$, a key observation we discover is that $\lambda^*$ is exactly the maximum value of the candidate set $\Lambda$. Although $\Lambda = \Theta(n^2)$, based on observations, we show that its maximum value can be computed in $O(n)$ time. Note that the above key observation does not hold if some sensors are not initially located on $B$.

For the general BCLS, as discussed in [8], the order preserving property does not hold any more. Consequently, the approach for the uniform case does not work. The difficulty for this problem is that we do not know the order of the sensors that will appear in an optimal solution. Due to this difficulty, no polynomial time algorithms have been discovered before. To resolve this issue, we first develop an algorithm for the decision version of the problem. After $O(n \log n)$ time preprocessing, our decision algorithm runs in $O(n \log \log n)$ time for any value $\lambda$. If $\lambda \geq \lambda^*$, which implies there exists a feasible solution, our decision algorithm can determine the order of sensors in a feasible solution for covering $B$. Then, to solve the general BCLS problem, we try to simulate the behavior of the decision algorithm on $\lambda = \lambda^*$. Although we do not know the value $\lambda^*$, our algorithm determines the same sensor order as would be determined 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 the parametric search [6, 13] and here we “parameterize” our decision algorithm. However, there are a few differences we should point out. First, unlike the typical parametric search [6, 13], our approach does not involve any parallel scheme and is practical. Second, typically, if a problem can be solved by parametric search, there also exist
other easier polynomial time algorithms for the problem although they might be less efficient than
the parametric search, e.g., the slope selection problem [7]. In contrast, for our general BCLS
problem, we have not found any other (even very trivial) polynomial time algorithms.

In addition, our $O(n)$ time algorithm for the simple cycle barrier coverage is a generalization of
our algorithm for the special case of the uniform BCLS where all sensors are initially located on $B$.

The rest of the paper is organized as follows. In Section 2, we introduce our algorithms for the
general BCLS. In Section 3, we present our algorithms for the uniform BCLS. Our results for the
simple cycle barrier coverage are discussed in Section 4.

For ease of exposition, we assume 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 sensors do not share a common
endpoint. Our algorithms can be easily generalized to the general situation.

2 The General Case of BCLS

In this section, we present our results for the general BCLS problem. Previously, it has been open
whether the problem can be solved in polynomial time. The difficulty is that we do not know the
order of the sensors that will appear in an optimal solution. Our main effort is to resolve this
difficulty. We derive an $O(n^2 \log n \log \log n)$ time algorithm for this problem.

We first present our algorithm for the decision version in Section 2.1, which is crucial for solving
the general BCLS in Section 2.2 (we refer to it as the optimization version of the problem).

For each sensor $s_i \in S$, we call the right (resp., left) endpoint of the covering interval of $s_i$
the right extension (resp., left extension) of $s_i$. The right or left extension 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.,
$p^-(x')$) the point to the right (resp., left) of $p(x')$ and infinitely close to $p(x')$. The concept of
$p^+(x')$ or $p^-(x')$ is only used to explain the algorithms and our algorithms never need to find such
a point explicitly. We use $\lambda^*$ to denote the maximum moving distance in an optimal solution
for the optimization version of the general BCLS problem. Note that we can easily determine whether
$\lambda^* = 0$, say, in $O(n \log n)$ time. In the following, we assume $\lambda^* > 0$.

2.1 The Decision Version of the General BCLS

Given any value $\lambda$, our goal is to determine whether there is a feasible solution in which the
maximum sensor movement is at most $\lambda$. Clearly, there is a feasible solution if and only if $\lambda \geq \lambda^*$.
We show that after $O(n \log n)$ time preprocessing, for any $\lambda$, we can determine whether $\lambda \geq \lambda^*$ in
$O(n \log n)$ time.

In the following, we first explore the properties of a feasible solution in Section 2.1.1. We describe
our algorithm in Section 2.1.2, argue its correctness in Section 2.1.3, and discuss its implementation
in Section 2.1.4. In Section 2.1.5, we show that by slightly extending the algorithm, we can also
determine whether $\lambda > \lambda^*$ with the same time performance, and this is particularly useful in our
optimization algorithm in Section 2.2.

2.1.1 Preliminaries

We refer to a configuration of the sensors as a specification on where each sensor is located. According
to this definition, the input is a configuration in which each sensor $s_i \in S$ is located at $x_i$.
For clarity in the later discussion, the displacement of each sensor in a configuration is the distance
between the position of the sensor in the configuration and its original position in the input. A
configuration is a feasible solution for the distance \( \lambda \) if the sensors form a barrier coverage and the displacement of each sensor is at most \( \lambda \). In a feasible solution, a subset \( S' \subseteq S \) is called a solution set if the sensors in \( S' \) form a barrier coverage, i.e., the union of the covering intervals of the sensors in \( S' \) contains the barrier \( B \). Of course, \( S \) itself is a solution set, and each 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 \( \lambda \), if \( \lambda \geq \lambda^* \), our decision algorithm will find a critical set and determine the order of sensors in the critical set that will appear in a feasible solution. For the purpose of showing the correctness of our algorithm, we explore some properties of a critical set.

Consider a critical set \( S^c \). For each sensor \( s \in S^c \), we call the set of points of \( B \) that are covered by \( s \) but not covered by any other sensors in \( S^c \) the unique coverage of \( s \).

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

**Proof:** Consider a sensor \( s \in S^c \). Assume to the contrary the unique coverage of \( s \) is not a continuous portion of \( B \). Then, there must exist at least one sensor \( s' \) in \( S^c \) whose covering interval is between two adjacent continuous portions of the unique coverage of \( B \). But that would mean \( s' \) is not critical since the covering interval of \( s' \) is a subset of that of \( s \). Hence, the observation follows.

For a critical set \( S^c \) in a feasible solution \( SOL \), we define the standard order of the sensors in \( S^c \) as the order of the sensors in \( SOL \) such that their unique coverages are from left to right.

**Observation 2** The standard order of the sensors of a critical set \( S^c \) in a feasible solution \( SOL \) is consist with the order of the positions of these sensors in \( SOL \) from left to right. Further, the standard order is also consist 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 \( S^c \) (their ranges are \( r_i \) and \( r_j \), respectively). Without loss of generality, assume \( s_i \) is in front of \( s_j \) in the standard order, i.e., the unique 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 is sufficient 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 unique coverage of \( s_j \). We also use \( p \) to denote its coordinate on \( x \)-axis. Thus, \( p \) is not covered by \( s_i \), implying that either \( p > y_i + r_i \) or \( p < y_i - r_i \). But the latter case cannot happen otherwise the unique 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, we obtain \( y_i + r_i < p \leq y_j + r_j \). By using a symmetric way of analysis, we can also prove \( y_i - r_i < y_j - r_j \) and we omit the details. Clearly, the two inequalities \( y_i + r_i < y_j + r_j \) and \( y_i - r_i < y_j - r_j \) infer \( y_i < y_j \) since each sensor is at the middle of its covering interval. The observation thus follows.

An interval of \( B \) is called a left-aligned interval if the left endpoint of the interval is at 0 (i.e., it is the form \([0, x']\) or \([0, x')\)). A set of sensors is said to be in attached positions in a configuration 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 their covering intervals. Two intervals of the \( x \)-axis overlap if they contain at least one common point.
2.1.2 The Algorithm Description

Initially, we move all sensors in \( S \) to the right for the distance \( \lambda \), i.e., for each \( 1 \leq i \leq n \), we move \( s_i \) to the position \( x_i' = x_i + \lambda \). Denote by \( C_0 \) the new configuration. Clearly, there is a feasible solution for \( \lambda \) if and only if we can move the sensors in \( C_0 \) to the left at most \( 2\lambda \) to form a coverage for \( B \). Thus, in the following we only need to consider moving the sensors to the left. Recall that we have assumed the extensions of any two sensors are different and thus in \( C_0 \) the extensions of any two sensors are also different.

Our algorithm is a greedy approach and it tries to find sensors to cover \( B \) from left to right. The algorithm has at most \( n \) steps. If \( \lambda \geq \lambda^* \), the algorithm will eventually find a critical set \( S^c \) of sensors along with the destinations for all these sensors; in theory, the other sensors in \( S \setminus S^c \) can be anywhere such that their displacements are at most \( \lambda \), but in the solution found by our algorithm, they are at the same positions as in \( C_0 \). Below, if a sensor is at the same position as in \( C_0 \), we say it stands still.

In the \( i \)-th step (initially, \( i = 1 \)), based on certain criterion, the algorithm finds a sensor \( s_{g(i)} \) and determines its destination \( y_{g(i)} \), where \( g(i) \) is the index of the sensor in \( S \). The destination \( y_{g(i)} \) is either \( x_{g(i)}' \) or a value at least \( x_{g(i)}' - 2\lambda \). Then, we move the sensor \( s_{g(i)} \) to \( y_{g(i)} \) to obtain a new configuration \( C_i \) (if \( y_{g(i)} = x_{g(i)}' \), we do not need to move the sensor and \( C_i \) is the same as the previous configuration \( C_{i-1} \)). Let \( R_i = y_{g(i)} + r_{g(i)} \), i.e., the right extension of \( s_{g(i)} \) in \( C_i \). 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 we terminate the algorithm. Otherwise, we proceed on the \((i+1)\)-th step. Further, it is possible such a sensor \( s_{g(i)} \) cannot be found, in which case we terminate the algorithm and report \( \lambda < \lambda^* \). In the following, we give the details, and in particular, discuss how to determine the sensor \( s_{g(i)} \) in each step.

Before discussing the first step, we give some intuition. Let \( S_t \) consists of the sensors whose right extensions are at most 0 in \( C_0 \). We claim that due to \( L > 0 \) no sensor in \( S_t \) can be in a critical set of a feasible solution if \( \lambda^* \leq \lambda \). Indeed, since sensors have been moved to their rightmost positions in \( C_0 \), if there is no sensor in \( S_t \) whose right extension is 0 in \( C_0 \), then the claim trivially follows; otherwise, suppose \( s_t \) is such a sensor. 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 \( p(0) \) since otherwise no sensor in \( S^c \) would cover \( p^+(0) \). Therefore, \( s_t \) is not critical with respect to \( S^c \), incurring contradiction. The claim thus follows. Therefore, we do not need to consider the sensors in \( S_t \) since they are not helpful for finding a feasible solution.

In the first step, we determine the sensor \( s_{g(1)} \), as follows. Define \( S_{11} = \{s_j \mid x'_j - r_j \leq 0 < x'_j + r_j\} \) (see Fig. 1). Namely, \( S_{11} \) consists of all sensors covering the point \( p(0) \) in \( C_0 \) except the sensor whose right extension is 0; but if the left extension of a sensor is 0, it is contained in \( S_{11} \). In other words, \( S_{11} \) consists of all sensors covering the point \( p^+(0) \) in \( C_0 \). If \( S_{11} \neq \emptyset \), we choose the sensor in \( S_{11} \) whose right extension is the largest (e.g., \( s_i \) in Fig. 1) as \( s_{g(1)} \) and let \( y_{g(1)} = x'_{g(1)} \). Note that since the extensions of any two sensors in \( C_0 \) are different, the sensor \( s_{g(1)} \) is unique. Otherwise, define \( S_{12} \) to be the set of sensors whose left extension is larger than 0 and at most \( 2\lambda \) (e.g., see Fig. 2). If \( S_{12} = \emptyset \), we terminate the algorithm and report \( \lambda < \lambda^* \). Otherwise, we choose the sensor in \( S_{12} \) whose right extension is the smallest as \( s_{g(1)} \) (e.g., \( s_i \) in Fig. 2) and let \( y_{g(1)} = r_{g(1)} \) (i.e., the left extension of \( s_{g(1)} \) is at 0 after it is moved to the destination).

If the algorithm is not terminated, we move \( s_{g(1)} \) to \( y_{g(1)} \) and obtain a new configuration \( C_1 \). Let \( S_1 = \{s_{g(1)}\} \). Let \( R_1 \) be the right extension of \( s_{g(1)} \) in \( C_1 \). If \( R_1 \geq L \), we have found a feasible solution \( C_1 \) with the critical set \( S_1 \) and we terminate the algorithm. Otherwise, we proceed on the
second step.

The general step is very similar as the first step. Consider the \( i \)-th step. We determine the sensor \( s_{g(i)} \) as follows. Let \( S_{i1} \) be the set of sensors covering the point \( p^{+}(R_{i-1}) \) in the configuration \( C_{i-1} \). If \( S_{i1} \neq \emptyset \), we choose the sensor in \( S_{i1} \) with the largest right extension as \( s_{g(i)} \) and let \( y_{g(i)} = x_{g(i)}^{t} \). Otherwise, let \( S_{i2} \) be the set of sensors whose left extension is 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 smallest right extension as \( s_{g(i)} \) and let \( y_{g(i)} = R_{i-1} + \tau_{g(i)} \). 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 \( S_{i} \) and we terminate the algorithm. Otherwise, we proceed on the \((i+1)\)-th step. If the sensor \( s_{g(i)} \) is from \( S_{i1} \) (resp., \( S_{i2} \)), we call it the type I (resp., type II) sensor.

Since there are totally \( n \) sensors in \( S \), the algorithm will be terminated in at most \( n \) steps. We finish the description of our algorithm.

### 2.1.3 The Correctness of the Algorithm

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

**Lemma 1** After the \( i \)-th step, suppose the algorithm produces the set \( S_{i} \) and obtain the configuration \( C_{i} \); we have the following properties for \( S_{i} \) and \( C_{i} \).

(a) \( S_{i} \) consists of type I and type II sensors.

(b) For each sensor \( s_{g(j)} \in S_{i} \) with \( 1 \leq j \leq i \), if \( s_{g(j)} \) is of type I, it stands still (i.e., 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 \( j > 1 \).

(c) The interval of \( B \) covered by the sensors in \( S_{i} \) is \([0, R_{i}]\).

(d) For each \( 1 < j \leq i \), the right extension of \( s_{g(j)} \) is larger than that of \( s_{g(j-1)} \).

(e) For each \( 1 \leq j \leq i \), \( s_{g(j)} \) is the only sensor in \( S_{i} \) that covers the point \( p^{+}(R_{j-1}) \).

**Proof:** The first three properties are trivial according to the algorithm description.

For property (d), note that the right extension of \( s_{g(j)} \) (resp., \( s_{g(j-1)} \)) is \( R_{j} \) (resp., \( R_{j-1} \)). According to our algorithm, the sensor \( s_{g(j)} \) covers the point \( p^{+}(R_{j-1}) \), implying that \( R_{j} > R_{j-1} \). Hence, property (d) follows.

For property (e), in the \( j \)-th step, the sensor \( s_{g(j)} \) is determined. On one hand, \( s_{g(j)} \) always cover \( p^{+}(R_{j-1}) \). On the other hand, consider a sensor \( s_{g(t)} \in S_{i} \). If \( t < j \), then the right extension of \( s_{g(t)} \) is at most \( R_{j-1} \) and thus \( s_{g(t)} \) cannot cover \( p^{+}(R_{j-1}) \). If \( t > j \), depending on whether \( s_{g(t)} \in S_{11} \) or \( s_{g(t)} \in S_{12} \), there are two cases. If \( s_{g(t)} \in S_{12} \), then the left extension of \( s_{g(t)} \) is \( R_{t-1} \),
which is larger than \( R_{j-1} \), and thus \( s_{g(t)} \) cannot cover \( p^+(R_{j-1}) \) in \( C_i \). Otherwise (i.e., \( s_{g(t)} \in S_{t1} \), \( s_{g(t)} \) stands still. Assume to the contrary that \( s_{g(t)} \) covers \( p^+(R_{j-1}) \) in \( C_i \). Then \( s_{g(t)} \) must have been in \( S_{j1} \) in the \( j \)-th step in the configuration \( C_{j-1} \). This implies \( S_{j1} \neq \emptyset \), \( s_{g(j)} \in S_{j1} \), and \( s_{g(j)} \) stands still. Since \( R_t \) is the right extension of \( s_{g(t)} \) and \( R_j \) is the right extension of \( s_{g(j)} \), according to \( (d) \), we have \( R_t > R_j \) due to \( t > j \). Since \( s_{g(t)} \) stands still in the \( t \)-th step, it also stands still in the \( j \)-th step (due to \( j < t \)). This implies that the right extension of \( s_{g(t)} \) is the value \( R_t \) and the right extension of \( s_{g(j)} \) is the value \( R_j \) in \( C_{j-1} \). Since \( R_t > R_j \) (i.e., the right extension of \( s_{g(j)} \) is less than that of \( s_{g(t)} \)), the algorithm cannot choose \( s_{g(j)} \) in the \( j \)-th step, incurring contradiction. Therefore, \( s_{g(t)} \) cannot cover the point \( p^+(R_{j-1}) \). Property \( (e) \) thus follows.

After the algorithm is terminated, it will either report \( \lambda \geq \lambda^* \) or \( \lambda < \lambda^* \). To argue the correctness of our algorithm, in the following, we will show that if the algorithm reports \( \lambda \geq \lambda^* \), then it indeed finds a feasible solution; otherwise, we prove that there is no feasible solution for \( \lambda \).

Suppose in the \( i \)-th step our algorithm reports \( \lambda \geq \lambda^* \). Then, according to our algorithm, it must be \( R_i \geq L \). By Lemma 1 \( (c) \) and \( (e) \), \( C_i \) is a feasible solution and \( S_i \) is a critical set. Further, by Lemma 1 \( (d) \) and Observation 2, the standard 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 it is true that there is no feasible solution for \( \lambda \). This is almost an immediate consequence of the following lemma.

**Lemma 2** Suppose \( S_i' \) is the set of sensors whose right extensions are at most \( R_i \) in the configuration \( C_i \); then the interval \([0, R_i]\) is the largest left-aligned interval that can be covered by the sensors in \( S_i' \) such that the displacement of each sensor in \( S_i' \) is at most \( \lambda \).

**Proof:** In our following discussion, when we say an interval is covered by the sensors in \( S_i' \), we implicitly mean the displacement of each sensor in \( S_i' \) is at most \( \lambda \).

We first prove the following claim: If there is a configuration \( C \) for the sensors in \( S_i' \) in which a left-aligned interval \([0, x']\) is covered by the sensors in \( S_i' \), then there always exists a configuration \( C^* \) for \( S_i' \) in which the interval \([0, x']\) is still covered by the sensors in \( S_i' \) and for each \( 1 \leq j < i \), the position of the sensor \( s_{g(j)} \) in \( C^* \) is \( y_{g(j)} \), where \( y_{g(j)} \) is the value that has been computed in our algorithm.

Similar to our discussion in Section 2.1.1, the configuration \( C \) always has a critical set for covering the interval \([0, x']\); we assume \( S_C \) is such a critical set of \( C \).

We prove the claim by induction. We first prove the base case, i.e., suppose there is a configuration \( C \) for the sensors in \( S_i' \) in which a left-aligned interval \([0, x']\) is covered by the sensors in \( S_i' \); then there is a configuration \( C'_1 \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_1 \) and the position of the sensor \( s_{g(1)} \) in \( C'_1 \) is \( y_{g(1)} \).

Let \( t = g(1) \). If the position of \( s_t \) in \( C'_1 \) is \( y_t \), then we are done with the proof and \( C'_1 \) is \( C \). Otherwise, let \( y'_t \) be the position of \( s_t \) in \( C \), and \( y'_t \neq y_t \). Depending whether \( s_t \in S_{11} \) and \( s_t \in S_{12} \), there are two cases.

- If \( s_t \in S_{11} \), then we have \( y_t = x'_t \). Since \( y_t \) is the right most place that the sensor \( s_t \) is allowed to move, we have \( y'_t < y_t \). Depending on whether \( s_t \) is in the critical set \( S_C \), there are further two subcases.

  If \( s_t \notin S_C \), then according to the definition of a critical set, the sensors in \( S_C \) always form a coverage for \([0, x']\) regardless of where \( s_t \) is. In other words, if we move \( s_t \) to \( y_t \) (other sensors stay in the same places as they were in \( C \)) to obtain a new configuration \( C'_1 \), then the sensors in \( S'_1 \) still form a coverage for \([0, x']\).

8
If \( s_t \in S_c \), since \( y_t > y'_t \), if we move \( s_t \) from \( y'_t \) to \( y_t \), \( s_t \) is moved to the right. Since \( s_t \in S_{11} \), when \( s_t \) is at \( y_t \), \( s_t \) still covers the point \( p(0) \). Thus, moving \( s_t \) from \( y'_t \) to \( y_t \) does not make \( s_t \) cover smaller sub-interval of \([0, x']\). Intuitively, moving \( s_t \) to the right does not lose anything for covering \([0, x']\). Therefore, after moving \( s_t \) to \( y_t \), we obtain a new configuration \( C'_1 \) in which the sensors in \( S'_i \) still form a coverage for \([0, x']\).

- If \( s_t \in S_{12} \), then according to our algorithm, \( s_t \) is the sensor in \( S_{12} \) with the smallest right extension in \( C_0 \). If \( s_t \notin S_C \), then by the same analysis as above, we can obtain a configuration \( C'_1 \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_i \) and the position of the sensor \( s_t \) in \( C'_1 \) is \( y_t \). Below, we discuss the case where \( s_t \in S_C \).

In \( S_C \), some sensors must cover the point \( p(0) \) in \( C \). Let \( S' \) be the set of sensors in \( S_C \) that cover \( p(0) \) in \( C \). If \( s_t \in S' \), then it is easy to know that \( y'_t < y_t \) since \( y_t \) is rightmost position for \( s_t \) to cover \( p(0) \). In this case, again, by the analysis before, we can always move \( s_t \) to the right from \( y'_t \) to \( y_t \) to obtain a configuration \( C'_1 \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_i \). Otherwise (i.e., \( s_t \notin S' \)), we show below that we can always move \( s_t \) to \( y_t \) by switching the relative positions of \( s_t \) and other sensors in \( S_C \).

An easy observation is that each sensor in \( S' \) must be in \( S_{12} \). Consider an arbitrary sensor \( s_h \in S' \). Since \( s_t \) is the sensor in \( S_{12} \) with the smallest right extension in \( C_0 \), the right extension of \( s_h \) is larger than that of \( s_t \) in \( C_0 \). Depending on whether the covering intervals of \( s_t \) and \( s_h \) overlap in \( C \), there are two subcases.

If the covering intervals of \( s_t \) and \( s_h \) overlap in \( C \), then let \([0, x'']\) be the left-aligned interval that is covered by \( s_t \) and \( s_h \) in \( C \) (see Fig. 3). If we switch their relative positions by moving \( s_t \) to \( y_t \) and moving \( y_h \) to \( x'' - r_h \) (i.e., the left extension of \( s_t \) is at 0 and the right extension of \( s_h \) is at \( x'' \)), then the two sensors still cover \([0, x'']\) (see Fig. 3), and thus the sensors in \( S'_i \) still form a coverage for \([0, x'']\). Further, after the above switch operation, the displacements of the two sensors are no more than \( \lambda \). To see this, clearly, the displacement of \( s_t \) is at most \( \lambda \). For the sensor \( s_h \), on one hand, it is easy to see that the above switch operation moves \( s_h \) to the right. On the other hand, since \( s_t \) covers \( p(x'') \) in \( C \), \( x'' \) is no larger than the right extension of \( s_t \) in \( C_0 \), which is smaller than that of \( s_h \) in \( C_0 \). Thus, \( x'' \) is less than \( x_h' + r_h \), implying that the position of \( s_h \) after the switch is still to the left of its position in \( C_0 \). Hence, after the switch operation, the displacement for \( s_h \) is no more than \( \lambda \). In summary, after the switch operation, we obtain a new configuration \( C'_1 \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_i \) and the position of the sensor \( s_t \) in \( C'_1 \) is \( y_t \).

If the covering intervals of \( s_t \) and \( s_h \) do not overlap in \( C \), then suppose the sensors in the critical set \( S_C \) between \( s_h \) and \( s_t \) are \( s_h, s_{f(1)}, s_{f(2)}, \ldots, s_{f(m)}; s_t \) in the standard order. Therefore, the covering intervals of any two consecutive sensors in the above list overlap in \( C \). Below, we show that we can also switch the relative positions between \( s_t \) and \( s_{f(m)} \) such that we can still form a coverage for \([0, x']\), and then we continue this switch procedure until \( s_t \) is switched
with \( s_h \). Note that since \( s_{11} = \emptyset \), the right extension of \( s_{f(j)} \) for each \( 1 \leq j \leq m \) is larger than that of \( s_t \) in \( C_0 \).

Let \( x''_1 \) be the maximum of 0 and the left extension of \( s_{f(m)} \) in \( C \). Let \( x''_2 \) be the minimum of \( x' \) and the right extension of \( s_t \) in \( C \) (see Fig. 4). Clearly \( x''_1 < x''_2 \). Thus, the sub-interval of \([0, x']\) covered by \( x_t \) and \( s_{f(m)} \) in \( C \) is \([x''_1, x''_2]\). We do a switch operation on \( s_t \) and \( s_{f(m)} \) by moving \( s_t \) to the left and moving \( s_{f(m)} \) to the right such that the left extension of \( s_t \) is at \( x''_1 \) and the right extension of \( s_{f(m)} \) is at \( x''_2 \) (see Fig. 4). It is easy to see that after the switch operation the sensors in \( S_C \) still form a coverage for \([0, x']\). Since the right extension of \( s_{g(m)} \) is larger than that of \( s_t \) in \( C_0 \), by similar analysis as before, we can also prove that after the switch, the displacements of \( s_t \) and \( s_{f(m)} \) are no more than \( \lambda \). Then, we continue this switch operation on \( s_t \) and \( s_{f(m-1)}, s_{f(m-2)}, \ldots \), until \( s_t \) is switched with \( s_h \), after which \( s_t \) is at \( y_t \), and we obtain a new configuration \( C'_1 \) in which the interval \([0, x']\) is still covered by the sensors in \( S_C \subseteq S'_t \) and the position of the sensor \( s_t \) in \( C'_t \) is \( y_t \).

The above proves the base case, i.e., there is always a configuration \( C'_1 \) in which the interval \([0, x']\) is covered by the sensors in \( S'_t \) and the position of the sensor \( s_{g(1)} \) in \( C'_1 \) is \( y_{g(1)} \).

We assume that the claim holds for \( k - 1 \) with \( 2 \leq k \leq i \), i.e., there is a configuration \( C'_{k-1} \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_t \) and the position of the sensor \( s_{g(j)} \) for each \( 1 \leq j \leq k - 1 \) in \( C'_{k-1} \) is \( y_{g(j)} \). In the following, we show that the claim holds for \( k \), i.e., there is a configuration \( C'_k \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_t \) and the position of the sensor \( s_{g(j)} \) for each \( 1 \leq j \leq k \) in \( C'_k \) is \( y_{g(j)} \). The proof is very similar to the base case and we briefly discuss it below.

Let \( t = g(k) \). If the position of \( s_t \) in \( C'_{k-1} \) is \( y_t \), then we are done with the proof and \( C'_k \) is \( C'_{k-1} \). Otherwise, let \( y'_t \) be the position of \( s_t \) in \( C'_{k-1} \), and \( y'_t \neq y_t \). Depending whether \( s_t \in S_{k1} \) and \( s_t \in S_{k2} \), there are two cases.

- If \( s_t \in S_{k1} \), then we have \( y_t = x'_t \). Since \( y_t \) is the right most place that the sensor \( s_t \) is allowed to move, we have \( y'_t < y_t \). Depending on whether \( s_t \) is in the critical set \( S_C \), there are further two subcases.

  If \( s_t \notin S_C \), then the sensors in \( S_C \) always form a coverage for \([0, x']\) regardless of where \( s_t \) is. In other words, if we move \( s_t \) to \( y_t \), we can obtain a new configuration \( C'_k \) in which the sensors in \( S'_t \) still form a coverage for \([0, x']\) and the position of the sensor \( s_{g(j)} \) for each \( 1 \leq j \leq k \) in \( C'_k \) is \( y_{g(j)} \).

  If \( s_t \in S_C \), since \( y_t > y'_t \), if we move \( s_t \) from \( y'_t \) to \( y_t \), \( s_t \) is moved to the right. By Lemma 2 (c), the interval \([0, R_{k-1}]\) is covered by the sensors \( \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(k-1)}\} \) in \( C'_{k-1} \) (since they are in positions \( y_{g(1)}, y_{g(2)}, \ldots, y_{g(k-1)} \), respectively). When \( s_t \) is at \( y_t \), \( s_t \) still covers the point \( p^+(R_{k-1}) \). Thus, after moving \( s_t \) to \( y_t \), we can obtain a new configuration \( C'_k \) in which the sensors in \( S'_t \) still form a coverage for \([0, x']\).
• If \( s_t \in S_{k2} \), then \( s_t \) is the sensor in \( S_{k2} \) with the smallest right extension. If \( s_t \not\in S_C \), then by the same analysis as before, we can obtain a configuration \( C'_t \) in which the interval \([0, x'_t]\) is still covered by the sensors in \( S'_t \) and the position of the sensor \( s_{g(j)} \) for each \( 1 \leq j \leq k \) in \( C'_k \) is \( y_{g(j)} \). Below, we discuss the case where \( s_t \in S_C \).

In \( S_C \), some sensors must cover the point \( p^+(R_{k-1}) \) in \( C \). Let \( S' \) be the set of sensors in \( S_C \) that cover \( p^+(R_{k-1}) \) in \( C \). If \( s_t \in S' \), then \( y'_t < y_t \) since \( y_t \) is rightmost position for \( s_t \) to cover \( p^+(R_{k-1}) \). In this case, again, by the analysis before, we can always move \( s_t \) to the right from \( y'_t \) to \( y_t \) to obtain a configuration \( C'_k \) in which the interval \([0, x']\) is still covered by the sensors in \( S'_t \). Otherwise (i.e., \( s_t \not\in S' \)), consider a sensor \( s_h \) in \( S' \). Let the sensors in \( S_C \) between \( s_h \) and \( s_t \) in the standard order be \( s_h, s_f(1), s_f(2), \ldots, s_f(m), s_t \) (this sequence may only contain \( s_h \) and \( s_t \)). Note that for each \( 1 \leq j \leq k-1 \), the sensor \( s_{g(j)} \) is not in the above sequence. Then, by a similar sequence of switch operations as we did in the base case, we can obtain a new configuration \( C'_k \) such that the sensors in \( S_C \) still form a coverage for \([0, x']\). Again, the position of the sensor \( s_{g(j)} \) for each \( 1 \leq j \leq k \) in \( C'_k \) is \( y_{g(j)} \).

The above proves that the claim holds for \( k \). Therefore, the claim is proved. The lemma can then be easily proved by using the claim, as follows.

Suppose the largest left-aligned interval that can be covered by the sensors in \( S'_t \) is \([0, x']\). Then according to the above claim, there always exists a configuration \( C^* \) for \( S'_t \) in which the interval \([0, x']\) is covered by the sensors in \( S'_t \) and for each \( 1 \leq j \leq i \), the position of the sensor \( s_{g(j)} \) in \( C^* \) is \( y_{g(j)} \). Then \( \lambda < \lambda^* \) in the \( i \)-th step. According to our 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 \( S \setminus S'_{i-1} \) is able to 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 in \( S'_{i-1} \). Therefore, the sensors in \( S \) cannot cover the interval \([0, R_{i-1}] \). Due to \( R_{i-1} < L \), we have \([0, R_{i-1}] \subseteq [0, L] \), and thus the sensors in \( S \) cannot cover the barrier \( B = [0, L] \). In other words, there is no feasible solution for the distance \( \lambda \). The correctness of our algorithm is established.

2.1.4 The Algorithm Implementation

For the implementation, in the \( i \)-th step, we need to maintain two sets of sensors, \( S_{i1} \) and \( S_{i2} \), as defined above. For this purpose, as preprocessing, we sort the 2\( n \) extensions of all sensors by \( x \)-coordinate. Also as preprocessing, we move each sensor \( s_i \in S \) to \( x'_i \) to obtain the configuration \( C_0 \). During the algorithm, we use two sweeping points \( p_1 \) and \( p_2 \) to sweep the \( x \)-axis for maintaining \( S_{i1} \) and \( S_{i2} \), respectively. Specifically, the point \( p_1 \) will follow the position \( R_0, R_1, R_2, \ldots \) and \( p_2 \) will follow the position \( R_0 + 2\lambda, R_1 + 2\lambda, R_2 + 2\lambda, \ldots \). Thus, \( p_2 \) is always \( 2\lambda \) to the right of \( p_1 \). For maintaining the set \( S_{i1} \), when the sweeping point \( p_1 \) encounters the left extension of a sensor, we insert the sensor into \( S_{i1} \); when \( p_1 \) encounters the right extension of a sensor, we delete the sensor from \( S_{i1} \). In this way, when the sweeping point \( p_1 \) is at \( R_{i-1} \), we have the set \( S_{i1} \) ready. For
maintaining the set $S_{i2}$, the situation is a little more subtle. First, whenever the sweeping point $p_2$ encounters the left extension of a sensor, we insert the sensor to $S_{i2}$. The subtle part is on the deletion operation. By the definition, if the left extension of a sensor is less than or equal to $R_{i-1}$, then it should not be in $S_{i2}$. Since eventually the first sweeping line $p_1$ is at $R_{i-1}$ in the $i$-th step, whenever a sensor is inserted into the first set $S_{i1}$, we need to delete the sensor from $S_{i2}$ (note that the sensor must be in $S_{i2}$ since $p_2$ is to the right of $p_1$). Thus, a deletion on $S_{i2}$ only happens when the same sensor is inserted into $S_{i1}$. In addition, we also need a search operation on $S_{i1}$ that is to find the sensor in $S_{i1}$ with the largest right extension and a search operation on $S_{i2}$ that is to find the sensor in $S_{i2}$ with the smallest right extension.

It is easy to see that there are $O(n)$ insertions and deletions in the entire algorithm. Further, the search operations on both $S_{i1}$ and $S_{i2}$ are dependent on the right extensions of the sensors. If we use a binary search tree to represent both sets in which the right extensions of the sensors are used as keys, then the algorithm takes $O(n \log \log n)$ time. Another way is to use integer data structures (e.g., van Emde Boas tree [16]) as follows. As preprocessing, we also sort the sensors by their right extensions, and for each sensor, we assign the integer $i$ to it as its key if the sensor is the $i$-th one in the above sorted order. Thus, all keys form an integer set $\{1, 2, \ldots, n\}$. By using the van Emde Boas tree [16], each insertion, deletion, or search operation only takes $O(\log \log n)$ time. Thus, after $O(n \log n)$ time preprocessing, the algorithm takes $O(n \log \log n)$ time for each value $\lambda$. Although the integer data structure does not change the overall running time, it makes our optimization algorithm in Section 2.2 faster.

**Theorem 1** With $O(n \log n)$ time preprocessing, for any $\lambda$, 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.

2.1.5 Another Decision Version

Our optimization algorithm in Section 2.2 also needs to determine whether $\lambda^*$ is strictly less than $\lambda$ (i.e., $\lambda^* < \lambda$) for any $\lambda$. By slightly modifying our algorithm for Theorem 1, we have the following result.

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

**Proof:** We first apply the algorithm for Theorem 1 on the value $\lambda$. If the algorithm reports $\lambda^* > \lambda$, then we know $\lambda^* < \lambda$ is false. Otherwise, we have $\lambda^* \leq \lambda$. In the following, we slightly modify the algorithm for Theorem 1 to determine whether $\lambda^* < \lambda$, i.e., $\lambda^*$ is strictly less than $\lambda$. Note that this is equivalent to determine whether $\lambda^* \leq \lambda - \epsilon$ for any arbitrarily small constant $\epsilon > 0$. Of course, we cannot enumerate all such small values $\epsilon$. Instead, we add a new mechanism to the algorithm for Theorem 1 such that the displacement of each sensor is strictly less than $\lambda$.

In the beginning of the algorithm, we move all sensors to the right for distance $\lambda$ to obtain the configuration $C_0$. But the displacement of each sensor should be strictly less than $\lambda$. To resolve this issue, later in the algorithm, if the destination of a sensor $s_i$ is set to $y_i = x'_i$, then we will adjust the destination of $s_i$ by moving it to the left a little such that displacement of $s_i$ is strictly less than $\lambda$.

Consider a general step, the $i$-th step, of the algorithm. We define the set $S_{i1}$ the same as before, i.e., it consists of all sensors covering the point $p^+(R_{i-1})$ in $C_{i-1}$. If $S_{i1} \neq \emptyset$, the algorithm is the same as before. In this case, the sensor $s_{g(i)}$ we choose in this step has displacement exactly
λ, which is “illegal” since the displacement of each sensor should be strictly less than λ. We will handle this issue later. If \( S_{11} = \emptyset \), however, the set \( S_{12} \) is defined slightly different from before. Here, since \( S_{11} = \emptyset \), we have to use the sensor to the right of \( R_{i-1} \) to cover \( p^+(R_{i-1}) \). As the displacement of each sensor should be strictly less than \( \lambda \), we do not allow any sensor to move to the left at exactly the distance \( 2\lambda \). To reflect this difference, we define \( S_{12} \) as the set of sensors each of which has its left extension larger than \( R_{i-1} \) and its right extension strictly less than \( R_{i-1} + 2\lambda \) (previously, it was “at most”). In this way, if we move a sensor in \( S_{12} \) to the left to cover \( p^+(R_{i-1}) \), the displacement of the sensor is strictly less than \( \lambda \). The rest of the algorithm is the same as before. We define the type I and type II sensors the same as before.

If the algorithm terminates without a feasible solution, then it must be \( \lambda^* > \lambda \); otherwise, the algorithm finds a “feasible” solution \( \text{SOL} \) with a critical set \( S^c = \{ s_{g(1)}, s_{g(2)}, \ldots, s_{g(m)} \} \). However, this does not necessarily mean \( \lambda^* < \lambda \) since in \( \text{SOL} \) the displacements of some sensors in \( S^c \) may be exactly \( \lambda \). Specifically, the type I sensors in \( S^c \) are in the same positions as they are in \( C_0 \) and thus their displacements are exactly \( \lambda \). In contrast, during the algorithm, the type II sensors in \( S^c \) have been moved strictly to the left with respect to their positions in \( C_0 \), and also due to our new definition of the set \( S_{12} \), the displacements of the type II sensors are strictly less than \( \lambda \). Therefore, if there is no type I sensors in \( S^c \), then the displacement of each sensor in \( S^c \) is strictly less than \( \lambda \) and thus we have \( \lambda^* < \lambda \). In the sequel, we assume \( S^c \) contains at least one type I sensor. To make sure \( \lambda^* < \lambda \), we need to find a real feasible solution in which the displacement of each sensor in \( S \) is strictly less than \( \lambda \). On the other hand, to make sure \( \lambda^* \geq \lambda \), we have to show that there is no real feasible solution. To this end, we use the following algorithmic procedure.

We try to adjust the solution \( \text{SOL} \) to obtain a real feasible solution. According to our algorithm, for each sensor \( s_i \in S^c \), if it is a type I sensor, then \( y_i = x_i' \) and thus its moving distance is exactly \( \lambda \); otherwise, its moving distance is less than \( \lambda \). The purpose of our adjustment on \( \text{SOL} \) is to move the type I sensors slightly to the left so that (1) their moving distances are strictly less than \( \lambda \), and (2) we can still form a coverage for \( B \). In some cases, we may need to use other sensors in \( S \setminus S^c \). Also, we may end up with the conclusion that no real feasible solution exists.

For example, since \( s_{g(m)} \) is the last sensor in \( S^c \), we have \( R_m \geq L \). If \( R_m > L \), then we can always adjust \( \text{SOL} \) to get a real feasible solution by shifting each sensor in \( S^c \) to the left for a very small value \( \epsilon \) such that (1) the displacement of each sensor in \( S^c \) is less than \( \lambda \), and (2) the sensors in \( S^c \) can still form a coverage on \( B \). Note that there always exists such a small value \( \epsilon \) such that the above adjustment is possible. Therefore, if \( R_m > L \), we have \( \lambda^* < \lambda \).

If \( R_m = L \), however, the above strategy does not work. There are several cases. If there exists a sensor \( s_i \in S \setminus S^c \) such that \( x_i \in (L - \lambda, L + \lambda) \), then we can also get a real feasible solution by shifting the sensors in \( S^c \) as before and using the sensor \( s_i \) to cover the rest interval of \( L \) that is not covered by the sensors in \( S^c \), and thus we also have \( \lambda^* < \lambda \). Otherwise, we claim that it must be \( \lambda^* \geq \lambda \). We prove the claim below.

Consider the rightmost Type I sensor \( s_i \) in \( S^c \). Suppose \( s_i = s_{g(j)} \), i.e., \( s_i \) is determined in the \( j \)-th step. Thus, \( s_i \) is at \( x_i' \) in \( \text{SOL} \). Let \( \epsilon \) be an arbitrarily small value (we will determine in the following how small it should be). Since we assumed that the extensions of any two sensors are different, the value \( \epsilon \) can be small enough such that if we move \( s_i \) to \( x_i' - \epsilon \) (instead of \( x_i' \)) in \( C_0 \) then the relative order of the extensions of all sensors are the same as before. Further, according to our above algorithm, the value \( \epsilon \) can also be small enough such that the behavior of the algorithm is the same as before, i.e., the algorithm finds the same set \( S^c \) with the same standard order as before. Now, at the \( j \)-th step, the new value \( R_j \), which is the right extension of \( s_i \), is \( \epsilon \) smaller than before since \( s_i \) was at \( x_i' - \epsilon \) in \( C_0 \). Since \( s_i \) is the rightmost type I sensor in \( S^c \), after the \( j \)-th
In this section, we discuss the optimization version of the general BCLS problem. We show that it is solvable in $O(n \log \log n)$ time for each value $\lambda$, after $O(n \log n)$ time preprocessing. The theorem thus follows.

Theorems 1 and 2 together lead to the following corollary.

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

### 2.2 The Optimization Version of the General BCLS

In this section, we discuss the optimization version of the general BCLS problem. We show that it is solvable in $O(n^2 \log n \log \log n)$ time and thus answer the open problem in [8].

The difficulty is that we do not know the order of the critical sensors in an optimal solution. Our strategy is to determine a critical set of sensors and their standard order in a feasible solution for the optimal value $\lambda^*$. Again, the idea is somewhat similar to the parametric search [6, 13] and here we “parameterize” our algorithm for Theorem 1. Unlike the typical parametric search [6, 13], our approach does not involve any parallel scheme and is practical. We first give an overview of our algorithm. In the following discussion, the “decision algorithm” refers to our algorithm for Theorem 1 unless otherwise stated.

Recall that given any value $\lambda$, the $i$-th step of our decision algorithm determines the sensor $s_{g(i)}$ and obtain the set $S_i = \{s_{g(1)}, s_{g(2)}, \ldots, s_{g(i)}\}$. We also call the order of the above sensors the standard order of the sensors in $S_i$. In our optimization algorithm, we usually use $\lambda$ as a variable, and thus we use $S_i(\lambda)$ (resp., $R_i(\lambda)$, $s_{g(i)}(\lambda)$, and $C_i(\lambda)$) to refer to the corresponding $S_i$ (resp., $R_i$, $s_{g(i)}$, and $C_i$) obtained when we run our decision algorithm on the specific value $\lambda$. Denote by $C_I$ the configuration of the input.

Our optimization algorithm has at most $n$ steps. Each $i$-th step receives an interval $(\lambda_{i-1}^1, \lambda_{i-1}^2)$ and the sensor set $S_{i-1}(\lambda^*)$, such that $\lambda^* \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$ and for each value $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$, we have $S_{i-1}(\lambda) = S_{i-1}(\lambda^*)$ and their standard orders are also the same. The $i$-th step will either finds the value $\lambda^*$ and then terminate the algorithm or determine the sensor $s_{g(i)}(\lambda^*)$ by using the algorithms for Theorems 1 and 2. The interval $(\lambda_{i-1}^1, \lambda_{i-1}^2)$ will shrink to a new interval $(\lambda_i^1, \lambda_i^2) \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$ and we also obtain the set $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_{g(i)}(\lambda^*)\}$. Again, the algorithm maintains the invariant that $\lambda^* \in (\lambda_i^1, \lambda_i^2)$ and for any value $\lambda \in (\lambda_i^1, \lambda_i^2)$, we have $S_i(\lambda) = S_i(\lambda^*)$ and their standard orders are also the same. Each step can be done in $O(n \log n \log \log n)$ time. The details of the algorithm are given below.

Initially, let $S_0(\lambda^*) = \emptyset$, $R_0(\lambda^*) = 0$, and $\lambda_0^1 = 0$ and $\lambda_0^2 = +\infty$.

In the first step, our goal is to choose the sensor $s_{g(1)}(\lambda^*)$, namely, the sensor determined in the first step of our decision algorithm on $\lambda = \lambda^*$, and let $S_1(\lambda^*) = \{s_{g(1)}(\lambda^*)\}$. In addition, we will also
obtain an interval \((\lambda_1, \lambda_2^2)\) such that \(\lambda^* \in (\lambda_1, \lambda_2^2)\) and for any \(\lambda \in (\lambda_1, \lambda_2^2)\), we have \(S_1(\lambda) = S_1(\lambda^*)\).

To determine \(s_{g(1)}(\lambda^*)\), as in the decision algorithm, we first determine the set \(S_{11}(\lambda^*)\). To this end, we define a value set \(A_{11}\) as follows. For each sensor \(s_i \in S\), if \(x_i + r_i \leq 0\) (resp., \(x_i - r_i \leq 0\)), then \(A_{11}\) contains the value \(|x_i + r_i|\) (resp., \(|x_i - r_i|\)). In other words, \(A_{11}\) consists of the absolute values of the sensor extensions at most 0. Further, let \(A_{11}\) contains \(\lambda_0\) and \(\lambda_0\) as well. Note that \(\lambda_0\) (resp., \(\lambda_0^2\)) is the smallest (resp., largest) value in \(A_{11}\). We sort all values in \(A_{11}\). Consider any two consecutive values \(\lambda_1\) and \(\lambda_2\) in the sorted list of \(A_{11}\). Assume \(\lambda_1 < \lambda_2\). It is easy to see that for any value \(\lambda \in (\lambda_1, \lambda_2)\), the set of sensors covering the point \(p(0)\) in the configuration \(C_0(\lambda)\) is the same. Recall that \(C_0(\lambda)\) is the configuration where each sensor \(s_i \in S\) is at \(x_i' = x_i + \lambda\). In other words, for any \(\lambda \in (\lambda_1, \lambda_2)\), the set \(S_{11}(\lambda)\) is the same. By binary search on the sorted list of \(A_{11}\) and using our decision algorithm, we determine the two consecutive values \(\lambda_1\) and \(\lambda_2\) in \(A_{11}\) such that \(\lambda_1 < \lambda^* \leq \lambda_2\) (in \(O(n \log n \log \log n)\) time). Further, we use Corollary 2 to determine whether 

\[\lambda^* = \lambda_2\]  

If \(\lambda^* = \lambda_2\), we terminate the algorithm and report \(\lambda^* = \lambda_2\). Otherwise, \(S_{11}(\lambda^*)\) is the set \(S_{11}(\lambda)\) for any \(\lambda \in (\lambda_1, \lambda_2)\). Thus, to compute \(S_{11}(\lambda^*)\), we can pick an arbitrary \(\lambda\) in \((\lambda_1, \lambda_2)\) and then find the set \(S_{11}(\lambda)\) in \(C_0(\lambda)\) in the same way as in the decision algorithm. Hence, \(S_{11}(\lambda^*)\) can be easily found, say, in \(O(n)\) time.

If \(S_{11}(\lambda^*) \neq \emptyset\), according to our decision algorithm, \(s_{g(1)}(\lambda^*)\) is the sensor in \(S_{11}(\lambda^*)\) with the largest right extension. Recall that we have assumed the extensions of any two sensors are different. An easy observation is that for any \(\lambda \in (\lambda_1, \lambda_2)\), the sensor in \(S_{11}(\lambda^*)\) with the largest right extension is the same. Thus, \(s_{g(1)}(\lambda^*)\) can be easily found, say, in \(O(n)\) time. Then, we let \(\lambda_1^2 = \lambda_1\) and \(\lambda_2^2 = \lambda_2\). The algorithm invariant holds. Further, it is easy to see that as the value \(\lambda\) increases in the interval \((\lambda_1, \lambda_2)\), the value \(R_1(\lambda)\), which is the right extension of \(s_{g(1)}(\lambda^*)\) (i.e., \(s_{g(1)}(\lambda^*)\)), increases with the same amount. In other words, the function of \(R_1(\lambda)\) in \((\lambda_1, \lambda_2)\) is a line segment of slope 1 (recall that \(s_{g(1)}(\lambda^*)\) is always in \(S_{11}(\lambda)\) for any \(\lambda \in (\lambda_1, \lambda_2)\)). If \(R_1(\lambda_2) < L\), we proceed on the next step, along with the interval \((\lambda_1^2, \lambda_2^2)\). If \(R_1(\lambda_2) = L\), since we know \(\lambda^* < \lambda_2\), we also proceed on the next step with \((\lambda_1^2, \lambda_2^2)\). If \(R_1(\lambda_2) > L\), we find the value \(\lambda'\) in \((\lambda_1, \lambda_2)\) such that \(R_1(\lambda') = L\), which can be easily done since \(R_1(\lambda)\) is a line segment of slope 1 in \((\lambda_1, \lambda_2)\). Clearly, \(\lambda^* \leq \lambda'\). We use Corollary 2 to determine whether \(\lambda^* = \lambda'\). If \(\lambda^* = \lambda'\), we terminate the algorithm and report \(\lambda^* = \lambda'\); otherwise, we have \(\lambda^* \in (\lambda_1, \lambda')\) and we update \(\lambda_2^2\) to \(\lambda'\). The algorithm invariant still holds. We proceed on the next step, along with the interval \((\lambda_1^2, \lambda_2^2)\).

If \(S_{11}(\lambda^*) = \emptyset\), according to our decision algorithm, we need to determine \(S_{12}(\lambda^*)\). For each \(\lambda \in (\lambda_1, \lambda_2)\), the set \(S_{12}(\lambda)\) consists of all sensors whose left extension are larger than 0 and at most 2\(\lambda\) in the configuration \(C_0(\lambda)\). As \(\lambda\) increases in \((\lambda_1, \lambda_2)\), the position of each sensor in \(C_0(\lambda)\) is a linear function of \(\lambda\) and the value \(2\lambda\) is a linear function of \(\lambda\). Therefore, there are \(O(n)\) \(\lambda\) values in \((\lambda_1, \lambda_2)\) each of which incurs the change of set \(S_{12}(\lambda)\) since each such value corresponds to the extension of a sensor. Further, we can determine these values in \(O(n \log n)\) time by sweeping (the fact \(S_{11}(\lambda^*) = \emptyset\) also makes this simpler). We omit the detailed discussion for this. (Actually, as \(\lambda\) increases in \((\lambda_1, \lambda_2)\), the size of the set \(S_{12}(\lambda)\) is monotone increasing.) Let \(A_{12}\) denote the set of all these \(O(n)\) \(\lambda\) values in \((\lambda_1, \lambda_2)\). Let \(A_{12}\) also contain both \(\lambda_1\) and \(\lambda_2\). Note that \(\lambda_1\) (resp., \(\lambda_2\)) is the smallest (resp., largest) value in \(A_{12}\). We sort the values in \(A_{12}\). Similarly, for any two consecutive values \(\lambda_1' < \lambda_2'\) in the sorted list of \(A_{12}\), the set \(S_{12}(\lambda)\) for any \(\lambda \in (\lambda_1', \lambda_2')\) is the same. Again, by binary search on the sorted list of \(A_{12}\) and using our decision algorithm, we determine (in \(O(n \log n \log \log n)\) time) the two consecutive values \(\lambda_1'\) and \(\lambda_2'\) in \(A_{12}\) such that \(\lambda_1' < \lambda^* \leq \lambda_2'\). Further, we use Corollary 1 to determine whether \(\lambda^* = \lambda_2'\). If \(\lambda^* = \lambda_2'\), we terminate the algorithm and report \(\lambda^* = \lambda_2'\). Otherwise, \(S_{12}(\lambda^*)\) is same as the set \(S_{12}(\lambda)\) for any \(\lambda \in (\lambda_1', \lambda_2')\), which can be easily obtained. Note that \((\lambda_1', \lambda_2') \subseteq (\lambda_1, \lambda_2)\).
After knowing $S_{12}(\lambda^*)$, we need to determine $s_{g(1)}(\lambda^*)$, which is the sensor in $S_{12}(\lambda^*)$ with the smallest right extension. As before, the sensor in $S_{12}(\lambda^*)$ with the smallest right extension is the same for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. Thus, $s_{g(1)}(\lambda^*)$ can be easily determined. Then, we let $\lambda_1^2 = \lambda_1^1$ and $\lambda_2^2 = \lambda_2^1$. The algorithm invariant holds. Further, unlike the former case, as the value $\lambda$ increases in the interval $(\lambda_1^1, \lambda_2^1)$, the right extension of $s_{g(1)}(\lambda)$ in $C_1(\lambda)$, which is $R_1(\lambda)$, does not change. In other words, the function of $R_1(\lambda)$ in $(\lambda_1, \lambda_2)$ is a horizontal line segment. Precisely, the value of $R_1(\lambda)$ is always equal to the length of the covering interval of $s_{g(1)}(\lambda^*)$. We claim that it must be $R_1(\lambda) < L$ due to $\lambda^* \in (\lambda_1^1, \lambda_2^1)$. Indeed, assume to the contrary that $R_1(\lambda) \geq L$. Then, the length of the covering interval of $s_{g(1)}(\lambda^*)$ is at least $L$. Observe that according to the definitions of $\lambda_1^1$ and $\lambda_2^1$, the set $S_{12}(\lambda_1^1)$ is the same as the set $S_{12}(\lambda)$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. This means the sensor $s_{g(1)}(\lambda^*)$ is also in $S_{12}(\lambda_1^1)$, implying that $\lambda^* \leq \lambda_1^1$. This contradicts with $\lambda^* \in (\lambda_1^1, \lambda_2^1)$. Therefore, we obtain that $R_1(\lambda) < L$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. We proceed on the next step, along with the interval $(\lambda_1^1, \lambda_2^2)$. Again, the algorithm invariant holds.

We finish the discussion for the first step. We have determined the sensor $s_{g(1)}(\lambda^*)$ and $S_1(\lambda^*) = \{s_{g(1)}(\lambda^*)\}$. We have obtained an interval $(\lambda_1^1, \lambda_2^1)$ such that $\lambda^* \in (\lambda_1^1, \lambda_2^1)$ and $S_1(\lambda) = S_1(\lambda^*)$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. It is easy to see that the first step can be implemented in $O(n \log n \log \log n)$ time.

In general, consider the $i$-th step for $i \geq 2$. The processing of it is similar to the first step. In the beginning of this step, we have the interval $(\lambda_1^{i-1}, \lambda_2^{i-1})$ and the set $S_{i-1}(\lambda^*)$ such that $\lambda^* \in (\lambda_1^{i-1}, \lambda_2^{i-1})$ and for any $\lambda \in (\lambda_1^{i-1}, \lambda_2^{i-1})$, $S_{i-1}(\lambda)$ is $S_{i-1}(\lambda^*)$ (with the same standard order). Later in Lemma 3, we will show that when $\lambda \in (\lambda_1^{i-1}, \lambda_2^{i-1})$, the function $R_{i-1}(\lambda)$ is a line segment of slope 1 or 0, and we can compute the function of $R_{i-1}(\lambda)$ explicitly in $O(n)$ time. At the moment, we assume this is true. In addition, we assume $R_{i-1}(\lambda) < L$ for any $\lambda \in (\lambda_1^{i-1}, \lambda_2^{i-1})$. This will be proved inductively at the end of the $i$-th step and the base case when $i = 1$ has been proved in the first step. In this step, 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 $\lambda \in (\lambda_1^i, \lambda_2^i)$, the set $S_i(\lambda)$ is $S_i(\lambda^*)$ (with the same standard order). The details are discussed below.

To determine the sensor $s_i(\lambda^*)$, 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 $S \setminus S_{i1}(\lambda^*)$, 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 $\lambda$ increases in $(\lambda_1^{i-1}, \lambda_2^{i-1})$, if the function of $R_{i-1}(\lambda)$ 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 of $R_{i-1}(\lambda)$ 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 $\lambda$ in $(\lambda_1^{i-1}, \lambda_2^{i-1})$ that can incur the change of the set $S_{i1}(\lambda)$ and each such value corresponds to a sensor extension, and further, these values can be easily determined in $O(n \log n)$ time by sweeping. Let $\Lambda_{i1}$ be the set of all above $\lambda$ values. Let $\Lambda_{i1}$ also contain both $\lambda_1^{i-1}$ and $\lambda_2^{i-1}$. Note that $\lambda_1^{i-1}$ (resp., $\lambda_2^{i-1}$) is the smallest (resp., largest) value in $\Lambda_{i1}$. Again, for any two consecutive values $\lambda_1 < \lambda_2$ in the sorted list, the set $S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$ is the same. Again, by binary search on the sorted list of $\Lambda_{i1}$ and using our decision algorithm, we determine the two consecutive values $\lambda_1$ and $\lambda_2$ in $\Lambda_{i1}$ such that $\lambda_1 < \lambda^* \leq \lambda_2$. Further, we use Corollary 2 to determine whether $\lambda^* = \lambda_2$. If $\lambda^* = \lambda_2$, we terminate the algorithm and report $\lambda^* = \lambda_2$. Otherwise, according to our discussion above, $S_{i1}(\lambda^*)$ is the set $S_{i1}(\lambda)$ for any $\lambda \in (\lambda_1, \lambda_2)$, which can be determined easily (say, in $O(n \log n)$ time). Note that $(\lambda_1, \lambda_2) \subseteq (\lambda_1^{i-1}, \lambda_2^{i-1})$.

If $S_{i1}(\lambda^*) \neq \emptyset$, $s_{g(i)}(\lambda^*)$ is the sensor in $S_{i1}(\lambda^*)$ with the largest right extension. Again, 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. Then, we let $\lambda_1^1 = \lambda_1$ and $\lambda_2^2 = \lambda_2$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_i(\lambda^*)\}$. The algorithm invariant holds. Further, as $\lambda$ increases in $(\lambda_1, \lambda_2)$, the right extension of $s_{g(i)}(\lambda)$, which is $R_i(\lambda)$, increases with the same amount. In other words, the function of $R_i(\lambda)$ in $(\lambda_1, \lambda_2)$ is a line segment of slope 1. If $R_i(\lambda_2) < L$, we proceed on the next step, along with the interval $(\lambda_1^1, \lambda_2^2)$. If $R_i(\lambda_2) = L$, since we know $\lambda^* < \lambda_2$, we proceed on the next step with $(\lambda_1^1, \lambda_2^2)$. If $R_i(\lambda_2) > L$, we find the value $\lambda' \in (\lambda_1, \lambda_2)$ such that $R_i(\lambda') = L$, which can be easily done since $R_i(\lambda)$ is a line segment of slope 1 in $(\lambda_1, \lambda_2)$. Clearly, $\lambda^* \leq \lambda'$. We use Corollary 2 to determine whether $\lambda^* = \lambda'$. If $\lambda^* = \lambda'$, we terminate the algorithm and report $\lambda^* = \lambda'$; otherwise, we have $\lambda^* \in (\lambda_1, \lambda')$ and we update $\lambda_2^2$ to $\lambda'$. The algorithm invariant still holds. We proceed on the next step, along with the interval $(\lambda_1^1, \lambda_2^2)$.

If $S_{i1}(\lambda^*) = \emptyset$, 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 extension 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 of $R_{i-1}(\lambda)$ in $(\lambda_1^1, \lambda_2^2)$ is a line segment of slope 1 or 0. Due to $(\lambda_1, \lambda_2) \subseteq (\lambda_1^1, \lambda_2^2)$, the function of $R_{i-1}(\lambda) + 2\lambda$ in $(\lambda_1, \lambda_2)$ is a line segment of slope 3 or 2. Again, as $\lambda$ 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 $O(n)$ values in $(\lambda_1, \lambda_2)$ each of which incurs the change of the set $S_{i2}(\lambda)$ and each such value corresponds to the extension of a sensor, and further, these values can be determined in $O(n \log n)$ time. (Actually, as $\lambda$ increases, the size of the set $S_{i2}(\lambda)$ is monotone increasing.) Let $A_{i2}$ denote the set of the above $\lambda$ values. Let $A_{i2}$ contain both $\lambda_1$ and $\lambda_2$. Note that $\lambda_1$ (resp., $\lambda_2$) is the smallest (resp., largest) value in $A_{i2}$. Again, $|A_{i2}| = O(n)$. We sort the values in $A_{i2}$. By binary search on the sorted list of $A_{i2}$ and using our decision algorithm, we determine the two consecutive values $\lambda_1'$ and $\lambda_2'$ in $A_{i2}$ such that $\lambda_1' < \lambda^* \leq \lambda_2'$. Further, we use Corollary 1 to determine whether $\lambda^* = \lambda_2'$. If $\lambda^* = \lambda_2'$, we terminate the algorithm and report $\lambda^* = \lambda_2'$. Otherwise, $S_{i2}(\lambda^*)$ is same as the set $S_{i2}(\lambda)$ for any $\lambda \in (\lambda_1', \lambda_2')$, which can be easily obtained. Note that $(\lambda_1', \lambda_2') \subseteq (\lambda_1, \lambda_2)$.

After knowing $S_{i2}(\lambda^*)$, $s_{g(i)}(\lambda^*)$ 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. Then, we let $\lambda_1^1 = \lambda_1'$ and $\lambda_2^2 = \lambda_2'$. Let $S_i(\lambda^*) = S_{i-1}(\lambda^*) \cup \{s_i(\lambda^*)\}$. The algorithm invariant holds. Further, we examine the function of $R_i(\lambda)$, i.e., the right extension of $s_{g(i)}(\lambda)$ in the configuration $C_i(\lambda)$, as $\lambda$ increases in $(\lambda_1', \lambda_2')$. Since $g_{i-1}(\lambda^*)$ and $g_{i}(\lambda^*)$ are always in attainable positions, for any $\lambda \in (\lambda_1', \lambda_2')$, we have $R_i(\lambda) = R_{i-1}(\lambda) + 2g_{i}(\lambda^*)$. Since we already explicitly know the function of $R_{i-1}(\lambda)$ for $\lambda \in (\lambda_1', \lambda_2')$, which is a line segment of slope 1 or 0, we can compute the function $R_i(\lambda)$ immediately. If $R_i(\lambda_2') < L$, then we proceed on the next step, along with the interval $(\lambda_1^1, \lambda_2^2)$. If $R_i(\lambda_2') = L$, since we already know that $\lambda^* < \lambda_2'$, we also proceed on the next step with $(\lambda_1^1, \lambda_2^2)$. If $R_i(\lambda_2') > L$, since we know explicitly the function of $R_i(\lambda)$ for $\lambda \in (\lambda_1', \lambda_2')$, which is a line segment of slope 1 or 0, we can compute the value $\lambda'' \in (\lambda_1', \lambda_2')$ such that $R_i(\lambda'') = L$. Clearly, $\lambda^* \leq \lambda''$. Then, we use Corollary 1 to test whether $\lambda^* = \lambda''$. If $\lambda^* = \lambda''$, then we terminate the algorithm and report $\lambda^* = \lambda''$. Otherwise, it must be $\lambda^* < \lambda''$ and we update $\lambda_2^2$ to $\lambda''$. Again, the algorithm invariant holds. We proceed on the next step, along with the interval $(\lambda_1^1, \lambda_2^2)$.

We finish the discussion for the general $i$-th step. Note that in any case where we proceed on the next step, for any $\lambda \in (\lambda_1^1, \lambda_2^2)$, the value $R_i(\lambda)$ is always less than $L$, which proves our previous assumption that $R_{i-1}(\lambda) < L$ for any $\lambda \in (\lambda_1^1, \lambda_2^2)$ inductively. We have determined the sensor $s_{g(i)}(\lambda^*)$ and the set $S_i(\lambda^*)$. We have obtained an interval $(\lambda_1^1, \lambda_2^2)$ such that $\lambda^* \in (\lambda_1^1, \lambda_2^2)$ and $S_i(\lambda) = S_i(\lambda^*)$ for any $\lambda \in (\lambda_1^1, \lambda_2^2)$. The running time of this step is clearly bounded by $O(n \log n \log \log n)$. It remains to prove the following lemma.
Lemma 3 Suppose the algorithm is not terminated after the i-th step with $i \geq 1$; then the function $R_i(\lambda)$ on $\lambda \in (\lambda_1^1, \lambda_2^1)$ is a line segment of slope 1 or 0. Further, the function $R_i(\lambda)$ on $\lambda \in (\lambda_1^1, \lambda_2^1)$ can be computed in $O(n)$ time.

Proof: We prove this by induction. In fact, the idea has already been given when we discuss the algorithm. The proof here is a summarization of them.

Consider the base case $i = 1$. The value $R_1(\lambda)$ is the right extension of the sensor $s_{g(1)}(\lambda^*)$, which is $s_{g(1)}(\lambda)$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. If $s_{g(1)}(\lambda^*) \in S_{11}(\lambda^*)$, then as $\lambda$ increases in $\lambda \in (\lambda_1^1, \lambda_2^1)$, since the position of the sensor $s_{g(1)}(\lambda^*)$ moves to the right at the same “speed” as $\lambda$ (because the position of $s_{g(1)}(\lambda^*)$ is at $x_{g(1)} + \lambda$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$). Therefore, the function $R_1(\lambda)$ on the interval $(\lambda_1^1, \lambda_2^1)$ is a line segment of slope 1. If $s_{g(1)}(\lambda^*) \in S_{12}(\lambda^*)$, the left extension of $s_{g(1)}(\lambda^*)$ is always at 0. Hence, the sensor $s_{g(1)}(\lambda^*)$ does not change position as $\lambda$ changes in $(\lambda_1^1, \lambda_2^1)$, implying that the function $R_1(\lambda)$ on the interval $(\lambda_1^1, \lambda_2^1)$ is a horizontal line segment.

The above proves the base case. We assume the statement in the lemma holds for $i - 1$, i.e., the function $R_{i-1}(\lambda)$ on $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$ is a line segment of slope 1 or 0. Below, we prove it also holds for $i$.

The value $R_i(\lambda)$ is the right extension of the sensor $s_{g(i)}(\lambda^*)$, which is $s_{g(i)}(\lambda)$ for any $\lambda \in (\lambda_1^1, \lambda_2^1)$. If $s_{g(i)}(\lambda^*) \in S_{i1}(\lambda^*)$, again, as $\lambda$ increases in $(\lambda_1^1, \lambda_2^1)$, the position of the sensor $s_{g(i)}(\lambda^*)$ moves to the right at the same “speed” as $\lambda$. Therefore, the function $R_i(\lambda)$ on the interval $(\lambda_1^1, \lambda_2^1)$ is a line segment of slope 1. If $s_{g(i)}(\lambda^*) \in S_{i2}(\lambda^*)$, the left extension of $s_{g(i)}(\lambda^*)$ is always at $R_{i-1}(\lambda)$, i.e., the two sensors $s_{g(i-1)}(\lambda^*)$ and $s_{g(i)}(\lambda^*)$ are at attached position. In other words, for any $\lambda \in (\lambda_1^1, \lambda_2^1)$, $R_i(\lambda) = R_{i-1}(\lambda) + 2r_{g(i)}$. Therefore, the function of $R_i(\lambda)$ is a vertical shift of that of $R_{i-1}(\lambda)$ for the distance $2r_{g(i)}$. By our hypothesis assumption that the function $R_{i-1}(\lambda)$ on $\lambda \in (\lambda_{i-1}^1, \lambda_{i-1}^2)$ is a line segment of slope 1 or 0, since $(\lambda_1^1, \lambda_2^1) \subseteq (\lambda_{i-1}^1, \lambda_{i-1}^2)$, the function $R_{i-1}(\lambda)$ on $\lambda \in (\lambda_1^1, \lambda_2^1)$ is a line segment of slope 1 or 0. Therefore, the function $R_i(\lambda)$ on $(\lambda_1^1, \lambda_2^1)$ is a line segment of slope 1 or 0.

According to the above analysis, for each $i \geq 2$, once we know the function $R_{i-1}(\lambda)$ on $(\lambda_{i-1}^1, \lambda_{i-1}^2)$, we can determine the function $R_i(\lambda)$ on $(\lambda_1^1, \lambda_2^1)$ in constant time. Initially, we can determine the function $R_1(\lambda)$ on $(\lambda_1^1, \lambda_2^1)$ in constant time. The lemma thus follows.

In at most $n$ steps, the algorithm will be terminated with the value $\lambda^*$. By applying our decision algorithm on $\lambda = \lambda^*$, we can finally find a solution for the barrier coverage in which the displacement of every sensor is at most $\lambda^*$. Since each step takes $O(n \log n \log \log n)$ time, the running time of the algorithm is $O(n^2 \log n \log \log n)$.

We make a technical remark. The typical parametric search [6, 13] usually returns with an interval containing the optimal value and then uses an additional step to find the optimal value. In contrast, our algorithm is guaranteed to find the optimal value $\lambda^*$ directly. This is due to the mechanism in our algorithm that requires $R_i(\lambda) < L$ for any $\lambda \in (\lambda_1^i, \lambda_2^i)$ after each $i$-th step if the algorithm is not terminated. The mechanism actually plays the role of the additional step used in the typical parametric search.

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

3 The Uniform Case of BCLS

In this section, we present our $O(n \log n)$ time algorithm for the uniform case. Previously, the best algorithm for it takes $O(n^2)$ time [8]. Further, for the special case where all sensors are initially on the barrier, we solve it in $O(n)$ time.

18
3.1 Preliminaries

Recall that in the input, all sensors are ordered from left to right by their index order in $S$, i.e., $x_1 \leq x_2 \leq \cdots \leq x_n$. Suppose we have a solution in which the destination for each sensor $s_i$ is $y_i$ ($1 \leq i \leq n$). We say that the solution is order preserving if $y_1 \leq y_2 \leq \cdots \leq y_n$. Since the ranges of all sensors are the same in the uniform case, for simplicity, we use $r$ to denote the sensor range. The following Lemma 4 has been proved in [8].

**Lemma 4 (Czyzowicz et al. [8])** For the uniform case, there always exists an optimal solution that is order preserving.

As discussed in [8], Lemma 4 is not applicable to the general BCLS. Consequently, the approach in this section does not work for the general BCLS.

Based on the order preserving property in Lemma 4, the previous $O(n^2)$ time algorithm [8] tries to cover $B$ from left to right; each step picks the next sensor and re-balances the current maximum sensor movement. Here, we use a completely different approach.

Denote by $\lambda^*$ the maximum sensor movement in an optimal solution. We use $OPT$ to denote an order preserving solution in which the destination for each sensor $s_i$ is $y_i$ ($1 \leq i \leq n$). For each sensor $s_i$, if $x_i > y_i$ (resp., $x_i < y_i$), then we say $s_i$ is moved to the left (resp., right) for a distance $|x_i - y_i|$. 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 the covering intervals of the sensors. A single sensor is always in attached position. The following result is almost self-evident and is given in [8].

**Lemma 5 (Czyzowicz et al. [8])** If $\lambda^* \neq 0$, then in OPT, there exists a sequence of sensors $s_i, s_{i+1}, \ldots, s_j$ with $i \leq j$ that are in attached positions, and further, one of the following three cases is true. (a) The sensor $s_j$ is moved to the left for the distance $\lambda$ and $y_i = r$ (i.e., the sensors $s_i, s_{i+1}, \ldots, s_j$ together cover exactly the interval $[0, 2r(j - i + 1)]$). (b) The sensor $s_i$ is moved to the right for the distance $\lambda$ and $y_j = L - r_j$. (c) The sensor $s_i$ is moved to the right for the distance $\lambda$ and the sensor $s_j$ is moved to the left for the distance $\lambda$; in this case, it is necessary that $i \neq j$.

Cases (a) and (b) in Lemma 5 are symmetric. In light of Lemma 5, for each pair of sensors $s_i$ and $s_j$ with $i \leq j$, we can compute three distances $\lambda_1(i, j), \lambda_2(i, j)$, and $\lambda_3(i, j)$ corresponding to the three cases in Lemma 5 as candidates for the optimal distance $\lambda^*$. Specifically, $\lambda_1(i, j) = x_j - [2r(j - i) + r]$, where the value $2r(j - i) + r$ is supposed to be the destination of the sensor $s_j$ in OPT if case (a) happens. Symmetrically, $\lambda_2(i, j) = [L - 2r(j - i) - r] - x_i$. Let $\lambda_3(i, j) = [x_j - x_i - 2r(j - i)]/2$ for $i < j$. Denote by $\Lambda$ the set of all $\lambda_1(i, j), \lambda_2(i, j)$, and $\lambda_3(i, j)$ values. Clearly, $\lambda^* \in \Lambda$ and $\Lambda = \Theta(n^2)$. By using an algorithm for the decision problem of the uniform case to search in $\Lambda$, we can find the value $\lambda^*$. Recall that the decision problem is that given any value $\lambda$, determine whether there exists a feasible solution to cover $B$ such that the moving distances of all sensors are at most $\lambda$. Thus, $\lambda^*$ is the smallest value in $\Lambda$ such that the answer for the decision problem on that value is “yes”. A simple greedy $O(n)$ time algorithm is given in [8] for the decision problem.

**Lemma 6 (Czyzowicz et al. [8])** The decision version of the uniform case is solvable in $O(n)$ time.

However, the above approach takes $\Omega(n^2)$ time due to $|\Lambda| = \Theta(n^2)$. To reduce the running time, we cannot compute the set $\Lambda$ explicitly. In general, our $O(n \log n)$ time algorithm uses the following idea. First, we do not compute all elements of $\Lambda$ explicitly. Instead, we compute an
element of \( \Lambda \) whenever we need it (we may need to do some preprocessing). Second, suppose we already (implicitly) know the sorted order of all values in \( \Lambda \); then we can use binary search and the decision algorithm for Lemma 6 to find \( \lambda^* \). However, we are not able to order the values of \( \Lambda \) in a single sorted list; instead we order them (implicitly) in \( O(n) \) sorted lists and each list has \( O(n) \) values. Consequently, by a technique called binary search in sorted arrays [5], we are able to compute \( \lambda^* \) in \( O(n \log n) \) time. The details of the algorithm are given in the next subsection.

3.2 Our Algorithm

Due to the order preserving property, it is easy to check whether \( \lambda^* = 0 \) in \( O(n) \) time. In the following, we assume \( \lambda^* \neq 0 \).

We focus on how to (implicitly) order the elements in \( \Lambda \) in \( O(n) \) sorted lists and each list contains \( O(n) \) elements. We also show that after preprocessing each element in any sorted list can be computed in constant time by specifying the index of the element. In summary, we will prove the following lemma.

**Lemma 7** In \( O(n \log n) \) time, the elements in \( \Lambda \) can be (implicitly) ordered in \( O(n) \) sorted lists such that each list contains \( O(n) \) elements and each element in any list can be computed in constant time by giving the index of the list and the index of the element in the list.

The following technique, which is called binary search on sorted arrays [5], will be used. Suppose there is a “black-box” decision procedure \( \Pi \) available such that given any value \( a \), \( \Pi \) can report whether \( a \) is a feasible value to \( \Pi \) in \( O(T) \) time, and further, if \( a \) is a feasible value, then any value larger than \( a \) is also feasible. Given a set of \( m \) arrays \( A_i \), \( 1 \leq i \leq m \), each containing \( O(n) \) elements in sorted order, the goal is to find the smallest feasible value \( \delta \) in \( A = \bigcup^n_{i=1} A_i \). Suppose each element of any array can be obtained in constant time by giving its indices. An algorithm for the following result is given in [5].

**Lemma 8** (Chen et al. [5]) The value \( \delta \) in \( A \) can be found in \( O((m + T) \log(nm)) \) time.

If we use the algorithm for Lemma 6 as the decision procedure \( \Pi \), then by Lemmas 7 and 8, we can find \( \lambda^* \) in \( \Lambda \) in \( O(n \log n) \) time. After \( \lambda^* \) is computed, we can use the algorithm for Lemma 6 to compute the destinations for all sensors in \( O(n) \) time. Hence, we have the following result.

**Theorem 4** The uniform case of the BCLS problem is solvable in \( O(n \log n) \) time.

In the rest of this subsection, we focus on proving Lemma 7.

For each \( 1 \leq t \leq 3 \), denote by \( \Lambda_t \) the set of all \( \lambda_t(i, j) \) values. Clearly, \( \Lambda = \Lambda_1 \cup \Lambda_2 \cup \Lambda_3 \). We order the three sets \( \Lambda_1, \Lambda_2, \Lambda_3 \) into sorted lists, respectively.

We discuss \( \Lambda_1 \) first. This case is trivial. It is easy to see that for each fixed value \( j \), we have \( \lambda_1(i_1, j) \leq \lambda_1(i_2, j) \) for any \( i_1 \leq i_2 \leq j \). Thus, for any value \( j \), we have a sorted list \( \lambda_1(1, j), \lambda_1(2, j), \ldots, \lambda_1(j, j) \) of \( j \) elements and each element can be computed in constant time by given the index of the element in the list. Therefore, we have \( n \) sorted list, and clearly, the set of the elements in all these lists is exactly \( \Lambda_1 \). Hence we have the following lemma.

**Lemma 9** In \( O(n \log n) \) time, the elements in \( \Lambda_1 \) can be (implicitly) ordered in \( O(n) \) sorted lists such that each list contains \( O(n) \) elements and each element in any list can be computed in constant time by giving the index of the list and the index of the element in the list.
Figure 5: Illustrating three groups of sensors: \( G_{i-1}, G_i, G_{i+1} \). The sensors with indices \( a_{i-1}, a_i, a_{i+1} \) (resp., \( b_{i-1}, b_i, b_{i+1} \)) are in \( S_a \) (resp., \( S_b \)), which are represented by thick segments.

The subset \( \Lambda_2 \) can be processed in a symmetric way as \( \Lambda_1 \) and we omit the details.

**Lemma 10** In \( O(n \log n) \) time, the elements in \( \Lambda_2 \) can be (implicitly) ordered in \( O(n) \) sorted lists such that each list contains \( O(n) \) elements and each element in any list can be computed in constant time by giving the index of the list and the index of the element in the list.

In the following, we focus on (implicitly) ordering the subset \( \Lambda_3 \) and prove the following Lemma 11, which together with Lemmas 9 and 10 proves Lemma 7.

**Lemma 11** In \( O(n \log n) \) time, the elements in \( \Lambda_3 \) can be (implicitly) ordered in \( O(n) \) sorted lists such that each list contains \( O(n) \) elements and each element in any list can be computed in constant time by giving the index of the list and the index of the element in the list.

Proving Lemma 11 is one main challenge of our algorithm. The reason is that unlike \( \Lambda_1 \) and \( \Lambda_2 \), for a fixed \( j \), for any \( 1 \leq i_1 \leq i_2 \leq j \), either \( \lambda_3(i_1, j) \leq \lambda_3(i_2, j) \) or \( \lambda_3(i_1, j) \geq \lambda_3(i_2, j) \) is possible. To prove Lemma 11, we have to find another way to order the elements in \( \Lambda_3 \).

Our approach is that we first remove some elements from \( \Lambda_3 \) that are guaranteed not to be \( \lambda^* \) (for example, a negative value cannot be \( \lambda^* \)). We begin with some intuitions. We say two intervals on the \( x \)-axis are **strictly overlapped** if they contain more than one common point. In the following discussion, the sensors are always on their input positions unless otherwise stated. We define two subsets of sensors \( S_a \) and \( S_b \) as follows. A sensor \( s_j \) is in \( S_a \) if and only if there is no sensor \( s_i \) with \( i < j \) such that their covering intervals are overlapped (e.g., see Fig. 5). A sensor \( s_i \) is in \( S_b \) if and only if there is no sensor \( s_j \) with \( i < j \) such that their covering intervals are overlapped. Let the indices of the sensors in \( S_a \) be \( a_1, a_2, \ldots, a_{n_1} \) and the indices of the sensors in \( S_b \) be \( b_1, b_2, \ldots, b_{n_2} \), from left to right. We claim that \( n_1 = n_2 \). Indeed, consider the interval graph \( G \) where the covering interval of each sensor is a vertex and two vertices are connected by an edge if their corresponding intervals are strictly overlapped. It is easy to see that in each connected component of \( G \), there is exactly one interval whose corresponding sensor is in \( S_a \) and there is exactly one interval whose corresponding sensor is in \( S_b \), and vice versa. Thus, \( n_1 = n_2 \), which is the number of connected components of \( G \). We let \( m = n_1 = n_2 \leq n \). Further, it is easy to see that the covering intervals of \( a_i \) and \( b_i \) must be in the same connected component of \( G \) and \( a_i \leq b_i \). Indeed, \( a_i \) (resp., \( b_i \)) is the leftmost (resp., rightmost) sensor in the subset of sensors whose covering intervals are in the connected component of \( G \) (e.g., see Fig. 5). Note that \( a_i = b_i \) is possible. Hence, there are \( m \) connected components of \( G \). For each \( 1 \leq i \leq m \), let \( G_i \) denote the connected component that contains the covering intervals of \( a_i \) and \( b_i \); with a little abuse of notation, we also use \( G_i \) to denote the subset of sensors whose covering intervals are in the connected component \( G_i \). Clearly, \( G_i = \{ s_j \mid a_i \leq j \leq b_i \} \) (e.g., see Fig. 5). We also call \( G_i \) a **group** of sensors. Clearly, all groups \( G_1, G_2, \ldots, G_m \) form a partition of \( S \). Notice that the sensor \( s_{a_i} \) (resp., \( s_{b_i} \)) is the leftmost (resp., rightmost) sensor of \( G_i \).

**Lemma 12** For any two sensors \( s_i \) and \( s_j \) with \( i < j \), if \( s_i \not\in S_b \) or \( s_j \not\in S_a \), then \( \lambda_3(i, j) \neq \lambda^* \).
Proof: Assume \( s_i \notin S_b \). In the following, we prove that \( \lambda_3(i, j) \) cannot be \( \lambda^* \) for any \( i < j \).

Suppose \( s_i \) is in the group \( G_k \). Then, \( i < b_k \) due to \( s_i \notin S_b \). Further, the covering intervals of \( s_i \) and \( s_{i+1} \) must be strictly overlapped since otherwise \( s_i \) would be in \( S_b \). Assume to the contrary that \( \lambda_3(i, j) = \lambda^* \), which implies the case (c) in Lemma 5 happens. Thus, in the corresponding solution, \( s_i \) is moved to the right for the distance \( \lambda_3(i, j) \) and all sensors \( s_i, s_{i+1}, \ldots, s_j \) must be in attached positions in OPT. It is easy to know that the sensor \( s_{i+1} \) must move to the right for the distance \( \lambda_3(i, j) + 2r - (x_{i+1} - x_i) \). Since the covering intervals of \( s_i \) and \( s_{i+1} \) are strictly overlapped, we have \( 2r - (x_{i+1} - x_i) > 0 \). Therefore, the moving distance of \( s_{i+1} \) must be larger than that of \( s_i \). Since the moving distance of \( s_{i+1} \) is \( \lambda_3(i, j) = \lambda^* \), we have contradiction. Therefore, \( \lambda_3(i, j) \) cannot be \( \lambda^* \).

Assume \( s_j \neq S_a \). By a symmetric analysis as above, we can prove that \( \lambda_3(i, j) \) cannot be \( \lambda^* \) for any \( i < j \). The lemma thus follows. \( \square \)

By Lemma 12, if \( \lambda^* \in \Lambda_3 \), it can only in the set \( \Lambda_3' = \{ \lambda_3(i, j) \mid i < j, i \in S_b, j \in S_a \} \), and \( |\Lambda_3'| = O(m^2) \). In the following, we will order the elements in \( \Lambda_3 \) in \( O(m) \) sorted lists and each list contains \( O(m) \) elements. One may attempt to use the following way. Clearly, for each \( 1 \leq k \leq m - 1 \), \( \Lambda_3' \) contains \( \lambda_3(s_{b_k}, s_{a_h}) \) for \( h = k + 1, k + 2, \ldots, m \), and thus we can simply form them as a list. However, the list is not sorted. Specifically, for any two indices \( h_1 \) and \( h_2 \) with \( k + 1 \leq h_1 \leq h_2 \leq m \), both \( \lambda_3(s_{b_k}, s_{a_{h_1}}) \leq \lambda_3(s_{b_k}, s_{a_{h_2}}) \) and \( \lambda_3(s_{b_k}, s_{a_{h_1}}) > \lambda_3(s_{b_k}, s_{a_{h_2}}) \) are possible. Our approach relies on more observations. Below, for simplicity of notation, we use \( \lambda_3(b_k, a_h) \) to refer to \( \lambda_3(s_{b_k}, s_{a_h}) \). We first examine the value of each \( \lambda_3(b_k, a_h) \).

By definition, we have \( \lambda_3(b_k, a_{k+1}) = (x_{a_{k+1}} - x_{b_k} - 2r) / 2 \), which is equal to the half length of the interval between the right extension of \( s_{b_k} \) and the left extension of \( s_{a_{k+1}} \); we call the above interval a gap. It should be noted that the above gap may be 0 if the two sensors \( s_{b_k} \) and \( s_{a_{k+1}} \) are in attached positions. For each \( 1 \leq k \leq m - 1 \), define \( g_k = x_{a_{k+1}} - x_{b_k} - 2r \), which is the length of the gap. Hence, we have \( \lambda_3(b_k, a_{k+1}) = g_k / 2 \). Further, for each \( 1 \leq k \leq m \), define \( l_k \) to be the sum of the lengths of the covering intervals of the sensors in the group \( G_k \), i.e., \( l_k = 2r(b_k - a_k + 1) \). Then, it is not difficult to see that \( \lambda_3(b_k, a_h) = (\sum_{t=k}^{h-1} g_t - \sum_{t=k+1}^{h-1} l_t) / 2 \), for each \( h \) with \( k + 2 \leq h \leq m \).

Lemma 13 For four indices \( k_1, k_2, h_1, h_2 \), suppose \( \max\{k_1, k_2\} < \min\{h_1, h_2\} \); then \( \lambda_3(b_{k_1}, a_{h_1}) - \lambda_3(b_{k_2}, a_{h_2}) = \lambda_3(b_{k_2}, a_{h_1}) - \lambda_3(b_{k_2}, a_{h_2}), \) and consequently, \( \lambda_3(b_{k_1}, a_{h_1}) \leq \lambda_3(b_{k_2}, a_{h_2}) \) if and only if \( \lambda_3(b_{k_2}, a_{h_1}) \leq \lambda_3(b_{k_2}, a_{h_2}) \).

Proof: For any \( 1 \leq k \leq m \), we have \( \lambda_3(b_k, a_h) = (\sum_{t=k}^{h-1} g_t - \sum_{t=k+1}^{h-1} l_t) / 2 \) for \( k + 2 \leq h \leq m \), and \( \lambda_3(b_k, a_h) = g_k / 2 \) for \( h = k + 1 \).

If \( h_1 = h_2 \), then the lemma trivially follows since \( \lambda_3(b_{k_1}, a_{h_1}) = \lambda_3(b_{k_2}, a_{h_2}) \) and \( \lambda_3(b_{k_2}, a_{h_1}) = \lambda_3(b_{k_2}, a_{h_2}) \). Below we assume \( h_1 \neq h_2 \).

By their definitions, we have \( \lambda_3(b_{k_1}, a_{h_1}) - \lambda_3(b_{k_2}, a_{h_1}) = \sum_{t=h_1}^{h_2-1} g_t + \sum_{t=h_1}^{h_2-1} l_t \). Similarly, \( \lambda_3(b_{k_2}, a_{h_1}) - \lambda_3(b_{k_2}, a_{h_2}) = \sum_{t=h_1}^{h_2-1} g_t + \sum_{t=h_1}^{h_2-1} l_t \). Hence, the lemma follows. \( \square \)

Lemma 13 implies that for any \( k_1 \) and \( k_2 \) with \( 1 \leq k_1 < k_2 < m - 1 \), the sorted order of \( \lambda_3(b_{k_1}, a_t) \) for \( t = k_2 + 1, k_2 + 2, \ldots, m - 1 \) is the same as that of the list \( \lambda_3(b_{k_2}, a_t) \) for \( t = k_2 + 1, k_2 + 2, \ldots, m - 1 \) in terms of the indices of \( a_t \). This also means that if we sort the values in the list \( \lambda_3(b_1, a_t) \) for \( t = 2, 3, \ldots, m \), then the sorted order of the list \( \lambda_3(b_k, a_t) \) for any \( 1 < k < m - 1 \) with \( t = k + 1, k + 2, \ldots, m \) is also implicitly obtained. Our “sorting” algorithm uses this idea as follows.

We first explicitly compute the values \( \lambda_3(b_1, a_t) \) for \( t = 2, 3, \ldots, m \), which can be done in \( O(m) \) time, and then sort them in \( O(m \log m) \) time. Let \( p \) be the permutation of \( 2, 3, \ldots, m \) such that the increasing sorted list of the above values are \( \lambda_3(b_1, a_{p(1)}), \lambda_3(b_1, a_{p(2)}), \ldots, \lambda_3(b_1, a_{p(m-1)}) \). Note that the permutation \( p \) is obtained immediately once we have the above sorted list. For any
1 < k < m, we say the element \( \lambda_3(b_k, a_h) \) is valid if \( k+1 \leq h \leq m \) and undefined otherwise. By Lemma 13, the valid elements in the list \( \lambda_3(b_k, a_{p(1)}), \lambda_3(b_k, a_{p(2)}), \ldots, \lambda_3(b_k, a_{p(m-1)}) \) are also sorted increasingly. Further, if we compute \( g_1, g_2, \ldots, g_{m-1} \) and \( l_1, l_2, \ldots, l_m \) as well as their prefix sums as preprocessing, then given the indices of any valid element in the above list, we can obtain its actual value in constant time. Clearly, the preprocessing takes \( O(n) \) time. Therefore, we have (implicitly) sorted the elements in \( \Lambda'_3 \) in \( O(m) \) sorted lists and each list has \( O(m) \) elements.

However, we are not done yet. Since eventually we will apply the binary search in Lemma 8 on the above sorted lists, the algorithm may choose an undefined element in the above list and apply it to the decision procedure that is the algorithm for Lemma 6. But the undefined elements do not have meaningful values. To resolve this issue, we (implicitly) assign an appropriate value to it such that the new list is still sorted, as follows. The idea is inspired by Lemma 13. We use the list \( \Lambda_3 \) as the reference list since all its elements are valid. Every other list \( \Lambda(k) \) has at least one valid element, for example, the element \( \lambda_3(b_k, a_{k+1}) \). We compute explicitly the value \( \lambda_3(b_k, a_{k+1}) \) for each \( k < m \), which can be done in \( O(m) \) time. Consider a list \( \Lambda(k) \) for \( 1 < k < m \). For any undefined element \( \lambda_3(b_k, a_{p(i)}) \in \Lambda(k) \), we assign to it the value \( \lambda_3(b_k, a_{k+1}) + \lambda_3(b_1, a_{p(i)}) - \lambda_3(b_1, a_{k+1}) \) (note that all three values have already been explicitly computed). Lemma 14 shows that the new list \( \Lambda(k) \) is still sorted increasingly after the value assignment.

**Lemma 14** For any \( 1 < k < m \), the list \( \Lambda(k) \) is still sorted increasingly after all its undefined elements are assigned values.

**Proof:** Consider any \( k \) with \( 1 < k < m \), and any two indices \( i \) and \( j \) with \( 1 \leq i < j \leq m - 1 \). It is sufficient to prove that \( \lambda_3(b_k, a_{p(i)}) \leq \lambda_3(b_k, a_{p(j)}) \).

If both values are valid, then according to our previous discussion, the inequality holds. Otherwise, we assume \( \lambda_3(b_k, a_{p(i)}) \) is undefined. After our value assignment, \( \lambda_3(b_k, a_{p(i)}) = \lambda_3(b_k, a_{k+1}) + \lambda_3(b_1, a_{p(i)}) - \lambda_3(b_1, a_{k+1}) \). Depending on whether \( \lambda_3(b_1, a_{p(j)}) \) is undefined, there are two cases.

If \( \lambda_3(b_1, a_{p(j)}) \) is undefined, we have \( \lambda_3(b_k, a_{p(j)}) = \lambda_3(b_k, a_{k+1}) + \lambda_3(b_1, a_{p(j)}) - \lambda_3(b_1, a_{k+1}) \). Hence, \( \lambda_3(b_k, a_{p(j)}) - \lambda_3(b_k, a_{p(i)}) = \lambda_3(b_1, a_{p(j)}) - \lambda_3(b_1, a_{p(i)}) \geq 0 \) due to \( j > i \).

If \( \lambda_3(b_1, a_{p(j)}) \) is valid, by Lemma 13, \( \lambda_3(b_k, a_{p(j)}) = \lambda_3(b_k, a_{k+1}) + \lambda_3(b_1, a_{p(j)}) - \lambda_3(b_1, a_{k+1}) \). Hence, \( \lambda_3(b_k, a_{p(j)}) - \lambda_3(b_k, a_{p(i)}) = \lambda_3(b_1, a_{p(j)}) - \lambda_3(b_1, a_{p(i)}) \geq 0 \).

Therefore, in both cases we have \( \lambda_3(b_k, a_{p(i)}) \leq \lambda_3(b_1, a_{p(j)}) \), which lead to the lemma.

In summary, in \( O(n \log n) \) time, we have (implicitly) ordered the elements in \( \Lambda'_3 \) in \( O(m) \) sorted lists and each list has \( O(m) \) elements such that each element in any list can be obtained in constant time. Hence, Lemma 11 is proved. We remark that assigning values to the undefined elements in \( \Lambda'_3 \) does not affect the correctness of the algorithm. Indeed, assigning values to undefined elements actually makes our candidate set \( \Lambda \) for \( \lambda^* \) larger, which obviously does not affect the algorithm correctness because the larger candidate set still contains \( \lambda^* \). It should also be pointed out that the statement of Lemma 11 (and thus Lemma 7) is not that accurate since we actually only ordered the elements in a subset \( \Lambda'_3 \) of \( \Lambda_3 \).

### 3.3 The Special Uniform Case

In this subsection, we discuss the special uniform case in which all sensors are initially located on the barrier \( B = [0, L] \), i.e., \( 0 \leq x_i \leq L \) for each \( 1 \leq i \leq n \). We give an \( O(n) \) time algorithm. Again, we assume \( \lambda^* \neq 0 \).
Clearly, Lemmas 4 and 5 still hold. Further, since all sensors are initially on $B$, in case (a) of Lemma 5, $s_i$ must be $s_1$. To see this, since $s_i$ is initially located on $B=[0,L]$, it is always the best to use $s_i$ to cover the beginning portion of $B$ due to the order preserving property. We omit the formal proof on this. Similarly, in case (b) of Lemma 5, $s_j$ must be $s_n$. We restate Lemma 5 below as a corollary for this special case.

**Corollary 2** If $\lambda^* \neq 0$, then in $OPT$, there exists a sequence of sensors $s_i, s_{i+1}, \ldots, s_j$ with $i \leq j$ that are in attached positions, and further, one of the following three cases is true. (a) The sensor $s_j$ is moved to the left for the distance $\lambda^*$, $i = 1$, and $y_1 = r$. (b) The sensor $s_i$ is moved to the right for the distance $\lambda^*$, $j = n$, and $y_n = L - r$. (c) The sensor $s_i$ is moved to the right for the distance $\lambda^*$ and the sensor $s_j$ is moved to the left for the distance $\lambda^*$; in this case, it is necessary that $i \neq j$.

For any $1 \leq i < j \leq n$, we define $\lambda_3(i, j)$ the same as before, i.e., $\lambda_3(i, j) = \lfloor x_j - x_i - 2r(j - i) \rfloor / 2$, which corresponds to case (c) of Corollary 2. For any $1 \leq j \leq n$, define $\lambda'_1(j) = x_j + r - 2rj$, which corresponds to case (a). Similarly, for any $1 \leq i \leq n$, define $\lambda'_2(i) = L - 2(rn - i) - (x_i + r)$, which corresponds to case (b). We still use $\Lambda_3$ to denote the set of all $\lambda_3(i, j)$ values. Define $\Lambda'_1 = \{\lambda'_1(j) | 1 \leq j \leq n\}$ and $\Lambda'_2 = \{\lambda'_2(i) | 1 \leq i \leq n\}$. Let $\Lambda' = \Lambda'_1 \cup \Lambda'_2 \cup \Lambda_3$. By Corollary 2, we have $\lambda^* \in \Lambda'$. The following lemma is crucial to our algorithm.

**Lemma 15** The optimal value $\lambda^*$ is the maximum value in $\Lambda'$.

**Proof**: Let $\lambda'$ be the maximum value in $\Lambda'$. It is sufficient to show $\lambda^* \leq \lambda'$ and $\lambda' \leq \lambda^*$. Due to $\lambda^* \in \Lambda'$, $\lambda'^* \in \Lambda'$ trivially holds. Below, we focus on proving $\lambda' \leq \lambda^*$.

Since $\lambda^*>0$, we have $\lambda'>0$. It is possible $\lambda' \in \Lambda'_1$, $\lambda' \in \Lambda'_2$, or $\lambda' \in \Lambda_3$. We analyze the three cases.

If $\lambda' \in \Lambda'_1$, suppose $\lambda' = \lambda'_1(j)$ for some $j$. Due to $\lambda' > 0$, we have $0 < \lambda' = \lambda'_1(j) = x_j + r - 2rj$, and thus $x_j - r > 2r(j - 1)$. Since all sensors are initially on the barrier $B$, $x_j \leq L$ holds. Therefore, even if all sensors $s_1, s_2, \ldots, s_{j-1}$ are moved such that they are in attached position to cover the sub-interval $[0, 2r(j - 1)]$ of $B$, the sub-interval $[2r(j - 1), x_j - r]$ is still not covered by any of the above sensors. In light of the order preserving property, to cover the above sub-interval, the best way is to move $s_j$ to the left such that the new position of $s_j$ is at $2r(j - 1) + r$ (i.e., the sensors $s_1, s_2, \ldots, s_j$ are in attached position), in which the moving distance of $s_j$ is exactly $\lambda'_1(j)$. Therefore, the maximum sensor movement in any optimal solution has to be at least $\lambda'_1(j)$, i.e., $\lambda' = \lambda'_1(j) \leq \lambda^*$.

If $\lambda' \in \Lambda'_2$, the analysis is symmetric to the above case and we omit the details.

If $\lambda' \in \Lambda_3$, the analysis has the similar spirit and we briefly discuss it. Suppose $\lambda' = \lambda_3(i, j) = \lfloor x_j - x_i - 2r(j - i) \rfloor / 2$ for some $i < j$. Since all sensors are initially on the barrier $B$, we have $0 \leq x_i < x_j \leq L$. Consider the sub-interval $[x_i + r, x_j - r]$ of $B$. Due to $\lambda' > 0$, we have $x_j - x_i - 2r(j - i) > 0$, which is also $(x_j - r) - (x_i + r) > 2r(j - i - 1)$. This implies that even if we move the sensors $s_{i+1}, s_{i+2}, \ldots, s_{j-1}$ such that they are in attached positions inside $[x_i + r, x_j - r]$, there are still points in $[x_i + r, x_j - r]$ that are not covered by the sensors above. In light of the order preserving property, to cover the interval $[x_i + r, x_j - r]$, we have to move both $s_i$ and $s_j$ and the best way is to move $s_i$ to the right and move $s_j$ to the left at equal distances so that all sensors $s_i, s_{i+1}, \ldots, s_j$ are in attached positions, where the moving distances of $s_i$ and $s_j$ are exactly $\lambda_3(i, j)$. Therefore, the maximum sensor movement in any optimal solution has to be at least $\lambda_3(i, j)$. In other words, $\lambda' = \lambda_3(i, j) \leq \lambda^*$.
In summary, in any case, $\lambda' \leq \lambda^*$ holds. The lemma thus follows.

Base on Lemma 15, to compute $\lambda^*$, we only need to find the maximum value in $\Lambda'$, which can be easily done in $O(n^2)$ time by computing the set $\Lambda'$ explicitly (note that $|\Lambda'| = \Theta(n^2)$). However, we show in Lemma 16 that we can find the maximum value in only $O(n)$ time without computing $\Lambda'$ explicitly.

**Lemma 16** The maximum value in $\Lambda'$ can be computed in $O(n)$ time.

**Proof:** Let $\lambda_1$, $\lambda_2$, and $\lambda_3$ be the maximum values in the three sets $\Lambda'_1$, $\Lambda'_2$, and $\Lambda'_3$, respectively. It is sufficient to compute $\lambda_1$, $\lambda_2$, and $\lambda_3$ in $O(n)$ time.

Both sets $\Lambda'_1$ and $\Lambda'_2$ can be computed explicitly in $O(n)$ time. Thus, we can compute $\lambda_1$ and $\lambda_2$ in $O(n)$ time. Below, we focus on computing $\lambda_3$.

Recall that for each value $\lambda_3(i,j) \in \Lambda'_3$ with $i < j$, we have $\lambda_3(i,j) = \frac{[x_j - x_i - 2r(j - i)]}{2}$. For each $1 \leq t \leq n - 1$, define $z_t = x_{t+1} - x_t - 2r$. Hence, $\lambda_3(i,j) = \frac{(\sum_{i=1}^{j-1} z_t)}{2}$. This implies that finding the maximum value in $\Lambda'_3$ is equivalent to finding a subsequence of $z_1, z_2, \ldots, z_{n-1}$ such that the sum of the subsequence is maximum among all subsequences. This can be done easily in $O(n)$ time. Specifically, we first compute all values $z_1, z_2, \ldots, z_{n-1}$, in $O(n)$ time. If all values are negative, then $\lambda_3$ is the maximum value divided by 2. Otherwise, we let $z'_0 = 0$, and for each $1 \leq t \leq n - 1$, let $z'_t = \max\{z'_{t-1}, 0\} + z_t$. It is not difficult to see that $\lambda_3 = \frac{1}{2} \cdot \max_{1 \leq t \leq n-1}\{z'_t\}$. Thus, $\lambda_3$ can be computed in $O(n)$ time.

The lemma thus follows.

After $\lambda^*$ is computed, we can use the linear time decision algorithm for Lemma 6 to compute the destinations for all sensors such that the maximum sensor movement is at most $\lambda^*$.

**Theorem 5** The special uniform case of the BCLS problem is solvable in $O(n)$ time.

4 The Simple Cycle Barrier Coverage

In this section, we discuss the simple cycle barrier coverage and give an $O(n)$ time algorithm for it. In this problem, the protected region $R$ is enclosed by a simple cycle $B$ that is the barrier we want to cover. The sensors in $S = \{s_1, s_2, \ldots, s_n\}$ are initially on $B$ and each sensor is only allowed to move on $B$ (e.g., not allowed to move inside or outside $R$). The sensors in $S$ have the same range $r$. Here the distance between two points on $B$ is not measured by their Euclidean distance in the plane but by their shortest distance along $B$. If a sensor is at a point $p$ on $B$, then it covers the points of $B$ whose distances to $p$ is at most $r$. Suppose all sensors in $S$ are initially ordered clockwise on $B$ by their indices. Our goal is to move the sensors along $B$ to form a coverage for $B$ such that the maximum sensor movement is minimized. To the best of our knowledge, we have not found any previous solutions for this problem.

Since now $B$ is a cycle, a sensor will be said to move clockwise or counterclockwise instead of right or left. Let $L$ be the length of $B$. Again, we assume $L \leq 2nr$ since otherwise it would not be possible to form a coverage for $B$. Since $B$ is a cycle, if $L \leq 2r$, then every sensor itself forms a coverage for $B$. Below, we assume $L > 2r$. Imagine that we pick a point $p_0$ on $B$ between $s_1$ and $s_n$ as the origin, and define the coordinate of any point $p$ on $B$ as the distance traversed if we move from $p_0$ to $p$ clockwise along $B$. Assume the coordinate of each sensor $s_i \in S$ is $x_i$. Thus, we have $0 < x_1 \leq x_2 \leq \cdots \leq x_n < L$. Further, for each $1 \leq i \leq n$, we let $s_{i+n}$ denote a duplication of the sensor $s_i$ with coordinate $x_{i+n} = x_i + L$, which refers to the position on $B$ with coordinate $x_i$. 

25
Since all sensors have the same range, it is easy to see that there always exists an order preserving optimal solution OPT where the sensors in OPT are also ordered clockwise on B by their indices. Again, let $\lambda^*$ be the optimal moving distance. Still, we can check whether $\lambda^* = 0$ in $O(n)$ time. Below, we assume $\lambda^* > 0$.

In fact, our algorithm will consider the $2n$ sensors $S' = \{s_1, s_2, \ldots, s_{2n}\}$. Specifically, the algorithm will determine a sequence of sensors $S'_{ij} = \{s_i, s_{i+1}, \ldots, s_j\} \subseteq S'$ with $i < j < i + n$ and move the sensors in $S'_{ij}$ to form a barrier coverage such that the maximum sensor movement is minimized. Clearly, for each sensor $s_k \in S$, at most one of $s_k$ and its duplication $s_{k+n}$ is in $S'_{ij}$.

The following result is a corollary of Lemma 5.

**Corollary 3** If $\lambda^* \neq 0$, then there exists an optimal solution in which a sequence of sensors $s_i, s_{i+1}, \ldots, s_j$ in $S'$ with $1 \leq i < j < i + n$ are in attached positions, and further, the sensor $s_i$ is moved clockwise for the distance $\lambda^*$ and the sensor $s_j$ is moved counterclockwise for the distance $\lambda^*$.

For each pair of $i$ and $j$ with $1 \leq i < j \leq 2n$, we define $\lambda_{ij} = [x_j - x_i - 2r(j-i)]/2$. Let $\Lambda$ be the set of all such $\lambda_{ij}$’s. By Corollary 3, we have $\lambda^* \in \Lambda$. Similar to Lemma 15, we have the following result.

**Lemma 17** The optimal value $\lambda^*$ is the maximum value in $\Lambda$.

**Proof:** The proof is very similar to that for Lemma 15 and we briefly discuss it below.

Let $\lambda'$ be the maximum value in $\Lambda$. It is sufficient to show $\lambda^* \leq \lambda'$ and $\lambda' \leq \lambda^*$. Due to $\lambda^* \in \Lambda$, $\lambda^* \leq \lambda'$ trivially holds. Below, we focus on proving $\lambda' \leq \lambda^*$. Since $\lambda^* > 0$, we have $\lambda' > 0$.

Suppose $\lambda' = \lambda(i,j) = [x_j - x_i - 2r(j-i)]/2$ for some $1 \leq i < j < i + n$. Consider the interval $[x_i + r, x_j - r]$ of $B$, i.e., the union of the points on $B$ from $x_i + r$ to $x_j - r$ clockwise. Due to $\lambda' > 0$, we have $x_j - x_i - 2r(j-i) > 0$, which is also $(x_j - r) - (x_i + r) > 2r(j-i-1)$. This implies that even if we move the sensors $s_{i+1}, s_{i+2}, \ldots, s_{j-1}$ such that they are in attached positions inside $[x_i + r, x_j - r]$, there are still points in $[x_i + r, x_j - r]$ that are not covered by sensors. In light of the order preserving property, to cover the interval $[x_i + r, x_j - r]$, the best way to move $s_i$ clockwise and move $s_j$ counterclockwise for the same distance, where the moving distances for $s_i$ and $s_j$ are exactly $\lambda(i,j)$. Therefore, the maximum sensor movement in any optimal solution has to be at least $\lambda(i,j)$. In other words, $\lambda' = \lambda(i,j) \leq \lambda^*$.

The lemma thus follows. $\square$

By the same algorithm for Lemma 16, we can find $\lambda^* \in \Lambda$ in $O(n)$ time. With the value $\lambda^*$, we can easily compute an optimal solution (i.e., compute the destinations for all sensors) in $O(n)$ time, as follows.

Suppose $\lambda^* = \lambda(i,j) \in \Lambda$ for some $1 \leq i < j < i + n$. In the case of $i > n$, we have $j > n$, and we let $i = i - n$ and $j = j - n$; thus, we still have $\lambda^* = \lambda(i,j)$ since $\lambda(i,j) = \lambda(i - n, j - n)$ when $i > n$ and $j > n$. Below, we assume $1 \leq i \leq n$. Note that it is possible that $j > n$.

First, we move $s_i$ clockwise for the distance $\lambda^*$ and move $s_j$ counterclockwise for the same distance $\lambda^*$. Next, move all sensors $s_{i+1}, s_{i+2}, \ldots, s_{j-1}$ such that the sensors $s_i, s_{i+1}, \ldots, s_j$ are in attached positions. Since $\lambda^*$ is the maximum value in $\Lambda$, the above movements of the sensors $s_{i+1}, s_{i+2}, \ldots, s_{j-1}$ are at most $\lambda^*$. Then, beginning from the sensor $s_{j+1}$, we consider the other sensors $s_{j+1}, s_{j+2}, \ldots, s_{i-1}$ clockwise on $B$, and move them to cover the portion of $B$ that are not covered by the sensors $s_i, s_{i+1}, \ldots, s_j$. To this end, we can view the above uncovered portion of $B$ as a line segment and apply the linear time greedy algorithm for Lemma 6. The overall running time is $O(n)$. 

26
**Theorem 6** The simple cycle barrier coverage problem is solvable in $O(n)$ time.

**References**


