Algorithmica (2015) 72:379–399
DOI 10.1007/s00453-013-9857-1

Optimal Point Movement for Covering Circular Regions

Danny Z. Chen · Xuehou Tan · Haitao Wang · Gangshan Wu

Received: 15 February 2013 / Accepted: 12 December 2013 / Published online: 18 December 2013

Abstract Given n points in a circular region C in the plane, we study the problems of moving the n points to the boundary of G to form a regular n-gon such that the maximum (min-max) or the sum (min-sum) of the Euclidean distances traveled by the points is minimized. These problems have applications, e.g., in mobile sensor barrier coverage of wireless sensor networks. The min-max problem further has two versions: the decision version and the optimization version. For the min-max problem, we present an $O(n \log^2 n)$ time algorithm for the decision version and an $O(n \log^3 n)$ time algorithm for the optimization version. The previously best algorithms for the two problem versions take $O(n^{3.5})$ time and $O(n^{3.5} \log n)$ time, respectively. For the min-sum problem we show that a special case with all points initially lying on the boundary of the circular region can be solved in $O(n^2)$ time, improving a previous $O(n^4)$ time solution. For the general min-sum problem, we present a 3-approximation $O(n^2)$ time algorithm. In addition, a by-product of our techniques is an algorithm for dynamically maintaining the maximum matching of a circular convex bipartite graph;

Chen's research was supported in part by NSF under Grants CCF-0916606 and CCF-1217906.

D.Z. Chen
Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA
e-mail: dchen@cse.nd.edu

X. Tan
Tokai University, 4-1-1 Kitakaname, Hiratsuka 259-1292, Japan
e-mail: tan@wing.ncc.u-tokai.ac.jp

H. Wang (✉)
Department of Computer Science, Utah State University, Logan, UT 84322, USA
e-mail: haitao.wang@usu.edu

G. Wu
State Key Lab for Novel Software Technology, Nanjing University, Nanjing 210093, China
e-mail: gswu@graphics.nju.edu.cn

our algorithm can handle each vertex insertion or deletion on the graph in $O(\log^2 n)$ time. This result may be interesting in its own right.

Keywords Computational geometry · Algorithms and data structures · Circular region coverage · Barrier coverage · Mobile sensors · Dynamic maximum matching · Circular convex bipartite graph

1 Introduction

Given n points in a circular region C in the plane, we study the problems of moving the n points to its boundary to form a regular n-gon such that the maximum (min-max) or the sum (min-sum) of the Euclidean distances traveled by the points is minimized. The problems have applications, e.g., in mobile sensor barrier coverage of wireless sensor networks. In this paper we present new algorithms that significantly improve the previous work on these problems.

1.1 Problem Definitions

Let $|ab|$ denote the Euclidean length of the line segment with two endpoints a and b in the plane. Let C be a circular region in the plane. Given a set of n points $S = \{A_0, A_1, \ldots, A_{n-1}\}$ in C (i.e., in its interior or on its boundary), we wish to move all sensors to n points $A'_0, A'_1, \ldots, A'_{n-1}$ on the boundary of C that form a regular n-gon. In other words, $A'_0, A'_1, \ldots, A'_{n-1}$ are the final positions of the points in S after their movement. The *min-max* problem aims to minimize the maximum Euclidean distance traveled by all points, i.e., $\max_{0\le i\le n-1}\{|A_i A'_i|\}$. The *min-sum* problem aims to minimize the sum of the Euclidean distances traveled by all points, i.e., $\sum_{i=0}^{n-1} |A_i A'_i|$.

Further, given a value $\lambda \ge 0$, the *decision version* of the min-max problem is to determine whether it is possible to move all points in S to the boundary of C to form a regular n-gon such that the distance traveled by each point is no more than λ. Indeed, let λ_C be the maximum distance traveled by the points in an optimal solution for the min-max problem. Then, the answer to the feasibility problem is "yes" if and only if $\lambda_C \le \lambda$. For discrimination, we refer to the original min-max problem as the *optimization version* of the min-max problem.

For the min-sum problem, if the points in S are initially located on the boundary of C, then this case is referred to as the *boundary case* of the min-sum problem.

1.2 Applications in Wireless Sensor Networks

A Wireless Sensor Network (WSN) is composed of a large number of sensors which monitor some surrounding environmental phenomenon. Usually, the sensors are densely deployed either inside the target phenomenon or are very close to it [1]. Each sensor is equipped with a sensing device with limited battery-supplied energy. The sensors process data obtained and forward the data to a base station. A typical type of WSN applications is concerned with security and safety systems, such as detecting intruders (or movement thereof) around infrastructure facilities and regions.

Particularly, it is often used to monitor a protected area so as to detect intruders as they penetrate the area or as they cross the area border. For example, research efforts have been under way to extend the scalability of wireless sensor networks to the monitoring of international borders [15, 17].

The study of *barrier coverage* using mobile sensors was originated in [7, 17] and later in [2]. Different from the traditional concept of *full coverage*, it seeks to cover the deployment region by guaranteeing that there is no path through the region that can be traversed undetectedly by an intruder, i.e., all possible crossing paths through the region are covered by the sensors [2, 7, 17]. Hence, an interesting problem is to reposition the sensors quickly so as to repair the existing security hole and thereby detect intruders [2]. Since barrier coverage requires fewer sensors for detecting intruders, it gives a good approximation of full area coverage. The planar region on which the sensors move is sometimes represented by a circle. Since sensors have limited battery-supplied energy, we wish to minimize their movement. Thus, if each sensor is represented as a point, the problem is exactly our optimal point movement min-max (the optimization version) or min-sum problem. Further, if each sensor has energy λ and we want to determine whether this level of energy is sufficient to form a barrier coverage, then the problem becomes the decision version of the min-max problem.

1.3 Previous Work and Our Results

For the min-max problem, Bhattacharya et al. [2] proposed an $O(n^{3.5})$ time algorithm for the decision version and an $O(n^{3.5}\log n)$ time algorithm for the optimization version, where the decision algorithm is based on some observations and brute force and the optimization algorithm is based on parametric search approach [9, 21]. Recently, it was claimed in [23] that these two problem versions were solvable in $O(n^{2.5})$ time and $O(n^{2.5}\log n)$ time, respectively. However, the announced algorithms in [23] contain errors. In this paper, we solve the decision version in $O(n\log^2 n)$ time and the optimization version in $O(n\log^3 n)$ time, which significantly improve the previous results.

In fact, the optimization version is equivalent to finding a regular n-gon on the boundary of C such that the bottleneck matching distance (i.e., the maximum matching distance) between the points in S and the vertices of the n-gon is minimized. The bottleneck matching problems have been studied, e.g., [6, 12, 13]. Another related work given by Bremner et al. [3] concerns two sets of points on a cycle (neither set of points have to form a regular n-gon) and one wants to rotate one set of points to minimize the matching distance between the two sets.

A by-product of our techniques that is interesting in its own right is an algorithm for dynamically maintaining the maximum matchings of *circular convex bipartite graphs*. Our algorithm handles each (online) vertex insertion or deletion on an n-vertex circular convex bipartite graph in $O(\log^2 n)$ time. This matches the performance of the best known dynamic matching algorithm for *convex bipartite graphs* [4]. Note that convex bipartite graphs are a subclass of circular convex bipartite graphs [18]. To our best knowledge, no dynamic matching algorithm for circular convex bipartite graphs was known before. In fact, our approach can be viewed as a

combination of the data structure in [4] and the linear time algorithm in [18] for computing a maximum matching in a circular convex bipartite graph. Since dynamically maintaining the maximum matching of a graph is a basic problem, our result may find other applications.

For the min-sum problem, an $O(n^2)$ time approximation algorithm with approximation ratio $1+\pi$ was given in [2]. A PTAS, which has a substantially larger polynomial time bound, was also given in [2]. In this paper, we present an $O(n^2)$ time approximation algorithm with approximation ratio 3, which improves the $(1+\pi)$-approximation result in [2]. However, whether the general min-sum problem is NP-hard is still left open.

For the boundary case of the min-sum problem, an $O(n^4)$ time (exact) algorithm was given in [23]. We show that the time bound of that algorithm can be reduced to $O(n^2)$.

The rest of this paper is organized as follows. Our algorithm for the decision version of the min-max problem is given in Sect. 2, and our algorithm for the optimization version is presented in Sect. 3. The min-sum problem is discussed in Sect. 4. Finally, Sect. 5 concludes.

To distinguish from a normal point in the plane, in the following paper we refer to each point $A_i \in S$ as a sensor.

2 The Decision Version of the Min-Max Problem

For simplicity, we assume the radius of the circle C is 1. Denote by ∂C the boundary of C. Let λ_C be the maximum distance traveled by the sensors in S in an optimal solution for the min-max problem, i.e., $\lambda_C = \min\{\max_{0\le i\le n-1}\{|A_i A'_i|\}\}$ among all possible points $A'_0, A'_1, \ldots, A'_{n-1}$ on ∂C that form a regular n-gon. Since the sensors are all in C, $\lambda_C \le 2$. In this section, we consider the decision version of the min-max problem on C: Given a value λ, determine whether $\lambda_C \le \lambda$. We present an $O(n\log^2 n)$ time algorithm for this problem.

In the following, we introduce some terminology in Sect. 2.1. The main idea of our algorithm is as follows. First, we model the problem as finding the maximum matchings in a sequence of $O(n)$ circular convex bipartite graphs in Sect. 2.2, which is further modeled in Sect. 2.3 as dynamically maintaining the maximum matching of a circular convex bipartite graph under a sequence of $O(n)$ vertex insertion and deletion operations. Second, we develop an approach for solving the latter problem in Sect. 2.3. Specifically, we show that the maximum matching of a circular convex bipartite graph of $O(n)$ vertices can be dynamically maintained in $O(\log^2 n)$ time (in the worst case) for each vertex insertion or deletion. Note that this result is of independent interest.

2.1 Preliminaries

We first discuss some concepts. A *bipartite* graph $G = (V_1, V_2, E)$ with $|V_1| = O(n)$ and $|V_2| = O(n)$ is *convex* on the vertex set V_2 if there is a linear ordering on V_2, say, $V_2 = \{v_0, v_1, \ldots, v_{n-1}\}$, such that if any two edges $(v, v_j) \in E$ and $(v, v_k) \in E$ with

$v_j, v_k \in V_2$, $v \in V_1$, and $j < k$, then $(v, v_l) \in E$ for all $j \le l \le k$. In other words, for any vertex $v \in V_1$, the subset of vertices in V_2 connected to v forms an interval on the linear ordering of V_2. For any $v \in V_1$, suppose the subset of vertices in V_2 connected to v is $\{v_j, v_{j+1}, \ldots, v_k\}$; then we denote $begin(v, G) = j$ and $end(v, G) = k$. Although E may have $O(n^2)$ edges, it can be represented implicitly by specifying $begin(v, G)$ and $end(v, G)$ for each $v \in V_1$. A *vertex insertion* on G is to insert a vertex v into V_1 with an edge interval $[begin(v, G), end(v, G)]$ and implicitly connect v to every $v_i \in V_2$ with $begin(v, G) \le i \le end(v, G)$. Similarly, a *vertex deletion* on G is to delete a vertex v from V_1 as well as all its adjacent edges.

A bipartite graph $G = (V_1, V_2, E)$ is *circular convex* on the vertex set V_2 if there is a circular ordering on V_2 such that for each vertex $v \in V_1$, the subset of vertices in V_2 connected to v forms a circular-arc interval on that ordering. Precisely, suppose such a *clockwise* circular ordering of V_2 is $v_0, v_1, \ldots, v_{n-1}$. If there exist two edges $(v, v_j) \in E$ and $(v, v_k) \in E$ with $v_j, v_k \in V_2$, $v \in V_1$, and $j < k$, then either $(v, v_l) \in E$ for all $j \le l \le k$, or $(v, v_l) \in E$ for all $k \le l \le n-1$ and $(v, v_l) \in E$ for all $0 \le l \le j$. For each $v \in V_1$, suppose the vertices of V_2 connected to v are from v_j to v_k clockwise on the ordering, then $begin(v, G)$ and $end(v, G)$ are defined to be j and k, respectively. Vertex insertions and deletions on G are defined similarly.

A maximum matching in a convex bipartite graph can be found in $O(n)$ time [14, 19, 22]. The same time bound holds for a circular convex bipartite graph [18]. Brodal et al. [4] designed a data structure for dynamically maintaining the maximum matchings of a convex bipartite graph that can support each vertex insertion or deletion in $O(\log^2 n)$ amortized time. For circular convex bipartite graphs, however, to our best knowledge, we are not aware of any previous work on dynamically maintaining their maximum matchings.

In the following, we first present the problem modeling and then give our algorithm for dynamically maintaining the maximum matching of a circular convex bipartite graph.

2.2 The Problem Modeling

Recall that in the decision version of the min-max problem, our goal is to determine whether $\lambda_C \le \lambda$. Let P be an arbitrary regular n-gon with its vertices $P_0, P_1, \ldots, P_{n-1}$ ordered clockwise on ∂C. We first consider the following subproblem: Determine whether we can move all sensors to the vertices of P such that the maximum distance traveled by the sensors is at most λ. Let G_P be the bipartite graph between the sensors $A_0, \ldots, A_{n-1}$ and the vertices of P, such that a sensor A_i is connected to a vertex P_j in G_P if and only if $|A_i P_j| \le \lambda$. The next lemma is immediate.

Lemma 1 *The bipartite graph G_P is circular convex.*

Proof This simply follows from the fact that the boundary of any circle of radius λ can intersect ∂C at most twice. □

To solve the above sub-problem, it suffices to compute a maximum matching M in the circular convex bipartite graph G_P (by using the algorithm in [18]). If M is a

perfect matching, then the answer to the sub-problem is "yes"; otherwise, the answer is "no". Thus, the sub-problem can be solved in $O(n)$ time (note that the graph G_P can be implicitly represented using $O(n)$ space, after $O(n \log n)$ time preprocessing). If the answer to the sub-problem is "yes", then we say that P is *feasible* with respect to the value λ.

If P is feasible, then clearly $\lambda_C \leq \lambda$. If P is not feasible, however, $\lambda_C > \lambda$ does not necessarily hold, because P may not be positioned "right" (i.e., P may not be the regular n-gon in an optimal solution of the optimization version of the min-max problem). To further decide whether $\lambda_C \leq \lambda$, our strategy is to rotate P clockwise on ∂C by an arc distance at most $2\pi/n$. Since the perimeter of C is 2π, the arc distance between any two neighboring vertices of P is $2\pi/n$. A simple yet critical observation is that $\lambda_C \leq \lambda$ if and only if during the rotation of P, there is a moment (called a *feasible moment*) at which P becomes feasible with respect to λ. Thus, our task is to determine whether a feasible moment exists during the rotation of P.

Consider the graph G_P. For each sensor A_i, denote by $E(A_i) = \{P_j, P_{j+1}, \ldots, P_k\}$ the subset of vertices of P connected to A_i in G_P, where the indices of the vertices of P are taken as modulo by n. We assume that $E(A_i)$ does not contain all vertices of P (otherwise, it is trivial). Since the arc distance from P_{j-1} to P_j is $2\pi/n$, during the (clockwise) rotation of P, there must be a moment after which P_{j-1} becomes connected to A_i, and we say that P_{j-1} is *added* to $E(A_i)$; similarly, there must be a moment after which P_k becomes disconnected to A_i, and we say that P_k is *removed* from $E(A_i)$. Note that these are the moments when the edges of A_i (and thus the graph G_P) are changed due to the rotation of P. Also, note that during the rotation, all vertices in $E(A_i) \setminus \{P_k\}$ remain connected to A_i and all vertices in $P \setminus \{E(A_i) \cup \{P_{j-1}\}\}$ remain disconnected to A_i. Hence throughout this rotation, there are totally n additions and n removals on the graph G_P. If we sort all these additions and removals based on the time moments when they occur, then we obtain a sequence of $2n$ circular convex bipartite graphs, and determining whether there exists a feasible moment is equivalent to determining whether there is a graph in this sequence that has a perfect matching. With the $O(n)$ time maximum matching algorithm for circular convex bipartite graphs of n vertices in [18], a straightforward solution for determining whether there is a feasible moment would take $O(n^2)$ time.

To obtain a faster algorithm, we further model the problem as follows. Consider the addition of P_{j-1} to $E(A_i)$. This can be done by first deleting the vertex of G_P corresponding to A_i and then inserting a new vertex corresponding to A_i with its edges connecting to the vertices in $\{P_{j-1}\} \cup E(A_i)$. The removal of P_k from $E(A_i)$ can be handled similarly. Thus, each addition or removal on $E(A_i)$ can be transformed to one vertex deletion and one vertex insertion on G_P. If we sort all vertex updates (i.e., insertions and deletions) by the time moments when they occur, then the problem of determining whether there is a feasible moment is transformed to determining whether there exists a perfect matching in a sequence of vertex updates on the graph G_P. In other words, we need to dynamically maintain the maximum matching in a circular convex bipartite graph to support a sequence of $2n$ vertex insertions and $2n$ vertex deletions. This problem is handled in the next subsection.

2.3 Dynamic Maximum Matching in a Circular Convex Bipartite Graph

In this subsection, we consider the problem of dynamically maintaining the maximum matching in a circular convex bipartite graph to support vertex insertions and deletions. We treat all vertex updates in an online fashion.

Let $G = (V_1, V_2, E)$ with $|V_1| = O(n)$ and $|V_2| = O(n)$ be a circular convex bipartite graph on the vertex set V_2, i.e., the vertices of V_2 connected to each vertex in V_1 form a circular-arc interval on the sequence of the vertex indices of V_2. Suppose $V_2 = \{v_0, v_1, \ldots, v_{n-1}\}$ is ordered clockwise. Recall that a vertex insertion on G is to insert a vertex v into V_1 with an edge interval $[begin(v, G), end(v, G)]$ such that v is (implicitly) connected to all vertices of V_2 from $begin(v, G)$ clockwise to $end(v, G)$. A vertex deletion is to delete a vertex v from V_1 and all its adjacent edges (implicitly). Our task is to design an algorithm for maintaining the maximum matching of G to support such update operations (i.e., vertex insertions and deletions) efficiently. Below, we present an algorithm with an $O(\log^2 n)$ time per update operation.

Our approach can be viewed as a combination of the data structure in [4] for dynamically maintaining the maximum matching in a convex bipartite graph and the linear time algorithm in [18] for computing a maximum matching in a circular convex bipartite graph. We refer to them as the BGHK data structure [4] and the LB algorithm [18], respectively. We first briefly describe the BGHK data structure and the LB algorithm.

The BGHK data structure [4] is a binary tree T, and each node of T maintains a balanced binary tree. This data structure can be constructed in $O(n \log^2 n)$ time and can support each vertex insertion or deletion in $O(\log^2 n)$ amortized time. Consider a vertex insertion, i.e., inserting a vertex v into V_1. Let M' (resp., M) be the maximum matching in the graph before (resp., after) the insertion. Let $|M|$ denote the number of matched pairs in M. After the data structure is updated (in $O(\log^2 n)$ amortized time), the value $|M|$ can be reported in $O(1)$ time and M can be reported in $O(|M|)$ time. The data structure can also determine in $O(1)$ time whether v is matched in M. Further, if another vertex $v' \in V_1$ was matched in M' but is not matched in M, then it is easy to see that v must be matched in M. Intuitively, the reason that v' is not matched in M is to "make room" for v to be matched (refer to [4] for more details). When this case occurs, we say that v *replaces* v' and v' is called the *replacement*, and the data structure is able to report the replacement in $O(1)$ time. Note that as shown in [4], although an update on the graph can cause dramatic changes on the maximum matching, the sets of the matched vertices in V_1 (and V_2) can change by at most one vertex. Thus, there is at most one such replacement v'. Similarly, consider deleting a vertex v from V_1. After the data structure is updated, the value $|M|$ can be reported in $O(1)$ time and M can be reported in $O(|M|)$ time. The data structure can also find out whether v was matched in M' in $O(1)$ time (refer to [4] for the details). If a vertex $v' \in V_1$ was not matched in M' but is matched in M, then v must be matched in M'. When this case occurs, we say v' is the *supplement*, which can be determined in $O(1)$ time.

The LB algorithm [18] finds a maximum matching in a circular convex bipartite graph $G = (V_1, V_2, E)$ by reducing the problem to two sub-problems of computing the maximum matchings in two convex bipartite graphs G_1 and G_2. Some details

are summarized below. For any vertex $v \in V_1$, if $begin(v, G) \leq end(v, G)$, then v is called a *non-boundary* vertex. Otherwise, v is a *boundary* vertex; the edges connecting v to $v_{begin(v,G)}, v_{begin(v,G)+1}, \ldots, v_{n-1}$ in V_2 are called *lower edges*, and the other edges connecting v are *upper edges*. Based on the graph G, a convex bipartite graph $G_1 = (V_1, V_2, E_1)$ is defined as follows. Both its vertex sets are the same as those in G. For each vertex $v \in V_1$ in G, $begin(v, G_1) = begin(v, G)$; if v is a non-boundary vertex, then $end(v, G_1) = end(v, G)$, and otherwise $end(v, G_1) = n - 1 + end(v, G)$ (note that this value of $end(v, G_1)$ is used only for comparison in the algorithm although there are not so many vertices in V_2). The LB algorithm has two main steps. The first step is to compute a maximum matching in G_1, which can be done in $O(n)$ time [14, 19, 22]. Let $M(G_1)$ be the maximum matching of G_1. Next, another convex bipartite graph $G_2 = (V_1, V_2, E_2)$ is defined based on $M(G_1)$ and G, as follows. Both its vertex sets are the same as those in G. For each non-boundary vertex $v \in V_1$ in G, $begin(v, G_2) = begin(v, G)$ and $end(v, G_2) = end(v, G)$. For each boundary vertex $v \in V_1$ in G, there are two cases: If v is matched in $M(G_1)$, then $begin(v, G_2) = begin(v, G)$ and $end(v, G_2) = n - 1$; otherwise, $begin(v, G_2) = 0$ and $end(v, G_2) = end(v, G)$. The second step of the LB algorithm is to compute a maximum matching in G_2 (in $O(n)$ time), denoted by $M(G_2)$. It was shown in [18] that $M(G_2)$ is also a maximum matching of the original graph G. Note that although the maximum matching in G_1 may not be unique, the LB algorithm works correctly regardless of which maximum matching of G_1 is computed in the first step.

We now discuss our algorithm for dynamically maintaining a maximum matching in the circular convex bipartite graph G. As for preprocessing, we first run the LB algorithm on G, after which both the convex bipartite graphs G_1 and G_2 of G are available. We then build two BGHK data structures for G_1 and G_2, denoted by $T(G_1)$ and $T(G_2)$, respectively, for maintaining their maximum matchings. This completes the preprocessing, which takes $O(n \log^2 n)$ time. In the following, we discuss how to perform vertex insertions and deletions.

Consider a vertex insertion, i.e., inserting a vertex v into V_1 with the edge interval $[begin(v, G), end(v, G)]$. To perform this insertion, intuitively, we need to update the two BGHK data structures $T(G_1)$ and $T(G_2)$ in a way that mimics some behavior of the LB algorithm. Specifically, we first insert v into the graph G_1 by updating $T(G_1)$. Based on the results on G_1 (e.g., whether there is a replacement) and the behavior of the LB algorithm, we modify G_2 by updating $T(G_2)$ accordingly. In this way, the maximum matching maintained by $T(G_2)$ is the maximum matching of G after the insertion. The details are given below.

Let G'_1 and G'_2 be the two graphs that would be produced by running the LB algorithm on G with the new vertex v (and its adjacent edges). Let $M(G_1)$, $M(G_2)$, $M(G'_1)$, and $M(G'_2)$ be the maximum matchings of G_1, G_2, G'_1, and G'_2, respectively. Depending on whether v is a boundary vertex, there are two main cases.

- If v is a non-boundary vertex (i.e., $begin(v, G) \leq end(v, G)$), then G'_1 can be obtained by inserting v into G_1. Hence we insert v into $T(G_1)$. Depending on whether there is a replacement, there are two cases.
  - If no replacement, then G'_2 can be obtained by inserting v into G_2. Thus, we simply insert v into $T(G_2)$ and we are done.

– Otherwise, let v' be the replacement. So v' was matched in $M(G_1)$ but is not matched in $M(G'_1)$. Depending on whether v' is a boundary vertex, there are two subcases.
  - If v' is a non-boundary vertex, then again, G'_2 can be obtained by inserting v into G_2. We thus insert v into $T(G_2)$ and we are done.
  - If v' is a boundary vertex, then since v' was matched in $M(G_1)$, according to the LB algorithm, v' with the edge interval $[begin(v', G), n-1]$ is in G_2. After the insertion of v into G_1, v' is not matched in $M(G'_1)$. Thus, according to the LB algorithm, G'_2 can be obtained by deleting v' (with the edge interval $[begin(v', G), n-1]$) from G_2, inserting v' with the edge interval $[0, end(v', G)]$ into G_2, and finally inserting v into G_2.

    In summary, for this subcase, we delete v' (with the edge interval $[begin(v', G), n-1]$) from $T(G_2)$ and insert v' with the edge interval $[0, end(v', G)]$ into $T(G_2)$. Finally, we insert v into $T(G_2)$, and we are done.

- If v is a boundary vertex (i.e., $begin(v, G) > end(v, G)$), then according to the LB algorithm, G'_1 can be obtained by inserting v with the edge interval $[begin(v, G), n-1+end(v, G)]$ into G_1. Thus we insert v with the edge interval $[begin(v, G), n-1+end(v, G)]$ into $T(G_1)$. Depending on whether there is a replacement, there are two cases.
  – If no replacement, then depending on whether v is matched in $M(G'_1)$, there are two subcases.
    - If v is matched, then according to the LB algorithm, G'_2 can be obtained by inserting v with the edge interval $[begin(v, G), n-1]$ into G_2. Thus, we insert v with the edge interval $[begin(v, G), n-1]$ into $T(G_2)$, and we are done.
    - If v is not matched, then according to the LB algorithm, G'_2 can be obtained by inserting v with the edge interval $[0, end(v, G)]$ into G_2. We thus insert v with the edge interval $[0, end(v, G)]$ into $T(G_2)$, and we are done.
  – Otherwise, there is a replacement v'. So v' was matched in $M(G_1)$ but is not matched in $M(G'_1)$, and v is matched in $M(G'_1)$. Depending on whether v' is a boundary vertex, there are two subcases.
    - If v' is a non-boundary vertex, then since v is matched in $M(G'_1)$, G'_2 can be obtained by inserting v with the edge interval $[begin(v, G), n-1]$ into G_2. We thus insert v with the edge interval $[begin(v, G), n-1]$ into $T(G_2)$.
    - If v' is a boundary vertex, then according to the LB algorithm, G'_2 is the graph obtained by deleting v' (with the edge interval $[begin(v', G), n-1]$) from G_2, inserting v' with the edge interval $[0, end(v', G)]$ into G_2, and finally inserting v with the edge interval $[begin(v, G), n-1]$ into G_2.

      Thus, we delete v' (with the edge interval $[begin(v', G), n-1]$) from $T(G_2)$, and insert v' with the edge interval $[0, end(v', G)]$ into $T(G_2)$. Finally, we insert v with the edge interval $[begin(v, G), n-1]$ into $T(G_2)$.

This completes the description of our procedure for handling a vertex insertion.

Next, consider a vertex deletion, i.e., deleting a vertex v from V_1 of G. Our procedure for this operation proceeds in a manner symmetric to the insertion procedure, and we briefly discuss it below. Define the two graphs G'_1 and G'_2 similarly as above.

- If v is a non-boundary vertex, then we delete v from $T(G_1)$. If no supplement, then we delete v from $T(G_2)$ and we are done. Otherwise, let v' be the supplement. So

v' was not matched in $M(G_1)$ but is matched in $M(G'_1)$. Depending on whether v' is a boundary vertex, there are two cases.
- If v' is a non-boundary vertex, then we delete v from $T(G_2)$ and we are done.
- If v' is a boundary vertex, then we delete v' with the edge interval $[0, end(v', G)]$ from $T(G_2)$ and insert v' with the edge interval $[begin(v', G), n-1]$ into $T(G_2)$. Finally, delete v from $T(G_2)$, and we are done.

- If v is a boundary vertex, then we delete v (with the edge interval $[begin(v, G), n-1+end(v, G)]$) from $T(G_1)$. Depending on whether there is a supplement, there are two cases.
  - If no supplement, then depending on whether v was matched in $M(G_1)$, there are two subcases. If v was matched, then we delete v (with the edge interval $[begin(v, G), n-1]$) from $T(G_2)$; otherwise, we delete v (with the edge interval $[0, end(v, G)]$) from $T(G_2)$.
  - Otherwise, let v' be the supplement. So v' was not matched in $M(G_1)$ but is matched in $M(G'_1)$, and v was matched in $M(G_1)$. Since v was matched in $M(G_1)$, according to the LB algorithm, G_2 contains v with the edge interval $[begin(v, G), n-1]$. If v' is a non-boundary vertex, then we delete v (with the edge interval $[begin(v, G), n-1]$) from $T(G_2)$ and we are done. Otherwise, since v' was not matched in $M(G_1)$, according to the LB algorithm, G_2 contains v' with the edge interval $[0, end(v', G)]$; since v' is matched in $M(G'_1)$, according to the LB algorithm, G'_2 should contain v' with the edge interval $[begin(v', G), n-1]$. Therefore, we delete v' (with the edge interval $[0, end(v', G)]$) from $T(G_2)$, insert v' with the edge interval $[begin(v', G), n-1]$ into $T(G_2)$, and finally delete v (with the edge interval $[begin(v, G), n-1]$) from $T(G_2)$.

This completes the description of our vertex deletion procedure.

As shown in Sect. 2.2, the decision version of the min-max problem can be transformed to the problem of dynamically maintaining the maximum matching in a circular convex bipartite graph subject to a sequence of vertex insertions and deletions. Hence, the correctness of our algorithm for the decision version hinges on the correctness of our dynamic maximum matching algorithm for circular convex bipartite graphs. Yet, the correctness of our (online) dynamic maximum matching algorithm for circular convex bipartite graphs can be seen quite easily. This is because our procedures for performing vertex insertions and deletions are both based on the fact that they simply mimic the behavior of the LB algorithm (while implementing their processing by the means of the BGHK data structures).

For the running time of our algorithm, each update operation involves at most two vertex insertions and two vertex deletions on $T(G_1)$ and $T(G_2)$, each of which takes $O(\log^2 n)$ amortized time [4]; thus, it takes $O(\log^2 n)$ amortized time in total. Actually, the BGHK data structure in [4] supports vertex insertions and deletions not only on V_1 but also on V_2. Inserting vertices on V_2 may make the tree unbalanced, and that is why its running time is amortized. However, if vertices are inserted only on V_1, then the tree will never become unbalanced and thus each update takes $O(\log^2 n)$ time in the worst case. In our problem formulation, the vertex updates indeed are only on V_1. Denote by $M(G)$ the maximum matching in G. We then have the following result.

Theorem 1 *A data structure on a circular convex bipartite graph $G = (V_1, V_2, E)$ can be built in $O(n \log^2 n)$ time for maintaining its maximum matching $M(G)$ so that each online vertex insertion or deletion on V_1 can be done in $O(\log^2 n)$ time in the worst case. After each update operation, $|M(G)|$ can be reported in $O(1)$ time and $M(G)$ can be reported in $O(|M(G)|)$ time.*

Since the decision version of the min-max problem has been reduced to dynamically maintaining the maximum matching in a circular convex bipartite graph under a sequence of $2n$ vertex insertions and $2n$ vertex decisions, we solve the dynamic maximum problem as follows. After each update operation, we check whether $|M(G)| = n$, and if this is true, then we report $\lambda_C \leq \lambda$ and halt the algorithm. If all $4n$ updates have been processed but it is always $|M(G)| < n$, then we report $\lambda_C > \lambda$. Based on Theorem 1, we have the result below.

Theorem 2 *Given a value λ, we can determine whether $\lambda_C \leq \lambda$ in $O(n \log^2 n)$ time for the decision version of the min-max problem.*

3 The Optimization Version of the Min-Max Problem

In this section, we consider the optimization version of the min-max problem, and present an $O(n \log^3 n)$ time algorithm for it. The main task is to compute the value λ_C.

To compute λ_C, we will first show in Sect. 3.1 that there exist a set D of distances such that $\lambda_C \in D$. Consequently, λ_C can be determined by searching D and using the decision algorithm in Theorem 2. However, since D is too large, it would take too much time to search D. To resolve the issue, in Sect. 3.2 we find a much smaller subset of D such that λ_C is still in the subset. Further, we show in Sect. 3.3 that the smaller subset can be determined implicitly such that the value λ_C can be computed in totally $O(n \log^3 n)$ time by using the decision algorithm in Theorem 2.

3.1 Preliminaries

Let o be the center of C. For simplicity of discussion, we assume that no sensor lies at o. Denote by X_i and Y_i the two points on ∂C which are closest and farthest to each sensor A_i, respectively. Clearly, X_i and Y_i are the two intersection points of ∂C with the line passing through A_i and the center o of C (see Fig. 1(a)). The lemma below has been proved in [23].

Lemma 2 [23] *Suppose an optimal solution for the min-max optimization problem is achieved with $\lambda_C = |A_i A'_i|$ for some $i \in \{0, \ldots, n-1\}$. Then either A'_i is the point X_i, or there is another sensor A_j ($j \neq i$) such that $\lambda_C = |A_j A'_j|$ also holds. In the latter case, any slight rotation of the regular n-gon that achieves λ_C in either direction causes the value of λ_C to increase (i.e., it makes one of the two distances $|A_i A'_i|$ and $|A_j A'_j|$ increase and the other one decrease).*

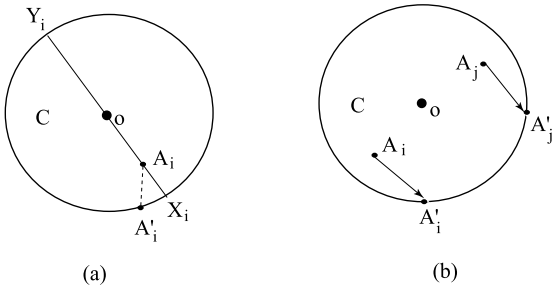

Fig. 1 (**a**) The points X_i and Y_i on ∂C for A_i; (**b**) $|A_iA'_i| = |A_jA'_j|$

The points on ∂C satisfying the conditions specified in Lemma 2 may be considered as those defining candidate values for λ_C, i.e., they can be considered as some vertices of possible regular n-gons on ∂C in an optimal solution. The points X_h of all sensors A_h $(0 \leq h \leq n-1)$ can be easily determined. Define $D_1 = \bigcup_{h=0}^{n-1}\{|A_hX_h|\}$, which can be computed in $O(n)$ time. But, the challenging task is to handle all the pairs (A_i, A_j) $(i \neq j)$ such that the distance from A_i to a vertex of a regular n-gon is equal to the distance from A_j to another vertex of that n-gon and a slight rotation of the n-gon in either direction monotonically increases one of these two distances but decreases the other. We refer to such distances as the *critical equal distances*. Denote by D_2 the set of all critical equal distances. Let $D = D_1 \cup D_2$. By Lemma 2, $\lambda_C \in D$. Thus, if D is somehow available, then λ_C can be determined by using our algorithm in Theorem 2 in a binary search process. Since D_1 is readily available, the key is to deal with D_2 efficiently. An easy observation is $\max_{0\leq h\leq n-1}|A_hX_h| \leq \lambda_C$. We can use the algorithm in Theorem 2 to check whether $\lambda_C \leq \max_{0\leq h\leq n-1}|A_hX_h|$, after which we know whether $\lambda_C = \max_{0\leq h\leq n-1}|A_hX_h|$. Below, we assume $\max_{0\leq h\leq n-1}|A_hX_h| < \lambda_C$ (otherwise, we are done). Thus, we only need to focus on finding λ_C from the set D_2.

It has been shown in [23] that $|D_2| = O(n^3)$. Of course, our goal is to avoid an $O(n^3)$ time solution. To do so, first we determine a subset D'_2 of D_2 such that $\lambda_C \in D'_2$ but with $|D'_2| = O(n^2)$. Furthermore, we do not compute D'_2 explicitly. Specifically, our idea is as follows. We show that the elements of D'_2 are the y-coordinates of a subset of intersection points among a set F of $O(n)$ functional curves in the plane such that each curve is x-monotone and any two such curves intersect in at most one point at which the two curves cross each other. (Such a set of curves is sometimes referred to as *pseudolines* in the literature.) Let $\mathcal{A}_F$ be the arrangement of F and $|\mathcal{A}_F|$ be the number of vertices of $\mathcal{A}_F$. Without computing $\mathcal{A}_F$ explicitly, we will generalize the techniques in [10] to compute the k-th highest vertex of $\mathcal{A}_F$ for any integer k with $1 \leq k \leq |\mathcal{A}_F|$ in $O(n\log^2 n)$ time. Consequently, with Theorem 2, the value λ_C can be computed in $O(n\log^3 n)$ time. The details are given below.

3.2 Determining the Set D'_2

Let P be an arbitrary regular n-gon with its vertices $P_0, P_1, \ldots, P_{n-1}$ placed clockwise on ∂C. Suppose the distances of all the pairs between a sensor and a vertex of P are $d_1 \leq d_2 \leq \cdots \leq d_{n^2}$ in sorted order. Let $d_0 = 0$. Clearly, $d_0 < \lambda_C \leq d_{n^2}$ (the case of $\lambda_C = 0$ is trivial). Hence, there exists an integer k with $0 \leq k < n^2$ such that

$\lambda_C \in (d_k, d_{k+1}]$. One can find d_k and d_{k+1} by first computing all these n^2 distances explicitly and then utilizing our algorithm in Theorem 2 in a binary search process. But that would take $\Omega(n^2)$ time. In the following lemma, we give a faster procedure without having to compute these n^2 distances explicitly.

Lemma 3 *The two distances d_k and d_{k+1} can be obtained in $O(n \log^3 n)$ time.*

Proof We apply a technique, called *binary search in sorted arrays* [8], as follows. Given M arrays A_i, $1 \leq i \leq M$, each containing $O(N)$ elements in sorted order, the task is to find a certain element $\delta \in A = \bigcup_{i=1}^{M} A_i$. Further, assume that there is a "black-box" decision procedure Π available, such that given any value a, Π reports $a \leq \delta$ or $a > \delta$ in $O(T)$ time. An algorithm is given in [8] to find the sought element δ in $A = \bigcup_{i=1}^{M} A_i$ in $O((M+T)\log(NM))$ time. We use this technique to find d_k and d_{k+1}, as follows.

Consider a sensor A_i. Let $S(A_i)$ be the set of distances between A_i and all vertices of P. In $O(\log n)$ time, we can implicitly partition $S(A_i)$ into two sorted arrays in the following way. By binary search, we can determine an index j such that X_i lies on the arc of ∂C from P_j to P_{j+1} clockwise (the indices are taken as module by n). Recall that X_i is the point on ∂C closest to A_i. If a vertex of P is on X_i, then define j to be the index of that vertex. Similarly, we can determine an index h such that Y_i (i.e., the farthest point on ∂C to A_i) lies on the arc from P_h to P_{h+1} clockwise. If a vertex of P is on Y_i, then define h to be the index of that vertex. Both j and h can be determined in $O(\log n)$ time, after which we implicitly partition $S(A_i)$ into two sorted arrays: One array consists of all distances from A_i to $P_j, P_{j-1}, \ldots, P_{h+1}$, and the other consists of all distances from A_i to $P_{j+1}, P_{j+2}, \ldots, P_h$ (again, all indices are taken as module by n). Note that both these arrays are sorted increasingly and each element in them can be obtained in $O(1)$ time by using its index in the corresponding array.

Thus, we obtain $2n$ sorted arrays (represented implicitly) for all n sensors in $O(n \log n)$ time, and each array has no more than n elements. Therefore, by using the technique of binary search in sorted arrays, with our algorithm in Theorem 2 as the black-box decision procedure, both d_k and d_{k+1} can be found in $O(n \log^3 n)$ time. The lemma thus follows. □

By applying Lemma 3, we have $\lambda_C \in (d_k, d_{k+1}]$. Below, for simplicity of discussion, we assume $\lambda_C \neq d_{k+1}$. Thus $\lambda_C \in (d_k, d_{k+1})$. Since $\max_{0 \leq h \leq n-1} |A_h X_h| < \lambda_C$, we redefine $d_k := \max\{d_k, \max_{0 \leq h \leq n-1} |A_h X_h|\}$. We still have $\lambda_C \in (d_k, d_{k+1})$. Let D'_2 be the set of all critical equal distances in the range (d_k, d_{k+1}). Then $\lambda_C \in D'_2$. We show below that $|D'_2| = O(n^2)$ and λ_C can be found in $O(n \log^3 n)$ time without computing D'_2 explicitly.

Suppose we rotate the regular n-gon $P = (P_0, P_1, \ldots, P_{n-1})$ on ∂C clockwise by an arc distance $2\pi/n$ (this is the arc distance between any two adjacent vertices of P). Let $A_i(P_h(t))$ denote the distance function from a sensor A_i to a vertex P_h of P with the time parameter t during the rotation. Clearly, the function $A_i(P_h(t))$ increases or decreases monotonically, unless the interval of ∂C in which P_h moves contains the point X_i or Y_i; if that interval contains X_i or Y_i, then we can further

divide the interval into two sub-intervals at X_i or Y_i, such that $A_i(P_h(t))$ is monotone in each sub-interval. The functions $A_i(P_h(t))$, for all P_h's of P, can thus be put into two sets S_{i1} and S_{i2} such that all functions in S_{i1} monotonically increase and all functions in S_{i2} monotonically decrease. Let $m = |S_{i1}|$. Then $m \leq n$. Denote by $d_1^i < d_2^i < \cdots < d_m^i$ the sorted sequence of the initial values of the functions in S_{i1}. Also, let $d_0^i = 0$ and $d_{m+1}^i = 2$ (recall that the radius of C is 1). It is easy to see that the range (d_k, d_{k+1}) obtained in Lemma 3 is contained in $[d_j^i, d_{j+1}^i]$ for some $0 \leq j \leq m$. The same discussion can be made for the distance functions in the set S_{i2} as well.

Since we rotate P by only an arc distance $2\pi/n$, during the rotation of P, each sensor A_i can have at most two distance functions (i.e., one decreasing and one increasing) whose values may vary in the range (d_k, d_{k+1}). We can easily identify these at most $2n$ distance functions for the n sensors in $O(n \log n)$ time. Denote by F' the set of all such distance functions. Clearly, all critical equal distances in the range (d_k, d_{k+1}) can be generated by the functions in F' during the rotation of P. Because every such distance function either increases or decreases monotonically during the rotation of P, each pair of one increasing function and one decreasing function can generate at most one critical equal distance during the rotation. (Note that by Lemma 2, a critical equal distance cannot be generated by two increasing functions or two decreasing functions.) Since $|F'| \leq 2n$, the total number of critical equal distances in (d_k, d_{k+1}) is bounded by $O(n^2)$, i.e., $|D_2'| = O(n^2)$. For convenience of discussion, since we are concerned only with the critical equal distances in (d_k, d_{k+1}), for each function in F', we restrict it to the range (d_k, d_{k+1}) only.

Let the time t be the x-coordinate and the function values be the y-coordinates of the plane. Then each function in F' defines a curve segment that lies in the strip of the plane between the two horizontal lines $y = d_k$ and $y = d_{k+1}$. We refer to a function in F' and its curve segment interchangeably, i.e., F' is also a set of curve segments. Clearly, a critical equal distance generated by an increasing function and a decreasing function is the y-coordinate of the intersection point of the two corresponding curve segments. Note that every function in F' has a simple mathematical description. Below, we simply assume that each function in F' is of $O(1)$ complexity. Thus, many operations on them can each be performed in $O(1)$ time, e.g., computing the intersection of a decreasing function and an increasing function.

The set D_2' can be computed explicitly in $O(n^2)$ time, after which λ_C can be easily found by binary search. Below, we develop a faster solution without computing D_2' explicitly, by utilizing the property that each element of D_2' is the y-coordinate of the intersection point of a decreasing function and an increasing function in F' and generalizing the techniques in [10].

3.3 Computing λ_C

A slope selection algorithm for a set of points in the plane was given in [10]. We will extend this approach to solve our problem. The following lemma is needed.

Lemma 4 *For any two increasing (resp., decreasing) functions in F', if the curve segments defined by them are not identical to each other, then the two curve segments*

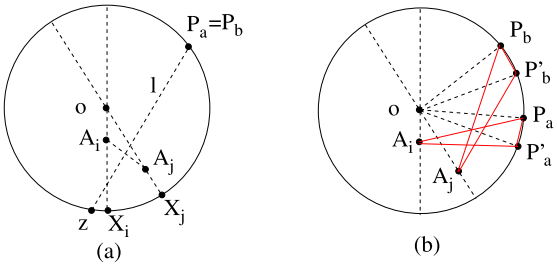

Fig. 2 Illustrating the proof of Lemma 4: (**a**) $P_a = P_b$; (**b**) $P_a \neq P_b$

intersect in at most one point and they cross each other at their intersection point (*if any*).

Proof We only prove the decreasing case. The increasing case can be proved similarly. Let $A_i(P_a(t))$ and $A_j(P_b(t))$ be two decreasing curves in F', where $A_i(P_a(t))$ (resp., $A_j(P_b(t))$) is the distance function between the sensor A_i (resp., A_j) and the vertex P_a (resp., P_b) of the regular n-gon P, and the two curve segments defined by $A_i(P_a(t))$ and $A_j(P_b(t))$ are not the same. Since each sensor has at most one decreasing function in F', we have $A_i \neq A_j$. We assume that during the (clockwise) rotation of P, $A_i(P_a(t)) = A_j(P_b(t))$ at the moment $t = t_1$ and t_1 is the first such moment. Below, we prove that $A_i(P_a(t)) = A_j(P_b(t))$ cannot happen again for any $t > t_1$ in the rotation. There are two cases: $P_a = P_b$ and $P_a \neq P_b$.

For any two points p and q, let $l(p, q)$ denote the line passing through the two points and $\overline{pq}$ denote the line segment with endpoints p and q whose length is $|pq|$. Recall that o is the center of the circle C. Let $P_a(t_1)$ and $P_b(t_1)$ be the positions of P_a and P_b at the moment t_1, respectively.

- $P_a = P_b$. Clearly, $P_a(t_1) = P_b(t_1)$. Let l be the perpendicular bisector of the line segment $\overline{A_i A_j}$. At the moment t_1, since $|A_i P_a(t_1)| = |A_j P_a(t_1)|$, $P_a(t_1)$ is at one of the two intersection points of l and ∂C. Further, since $A_i(P_a(t))$ is a decreasing function, $P_a(t_1)$ must be on the right side of the line $l(A_i, o)$ if we walk from A_i to o (see Fig. 2(a)). Similarly, $P_a(t_1)$ must be on the right side of the line $l(A_j, o)$ (going A_j to o). Let z be the other intersection point of l and ∂C. It is easy to see that z is on the left side of either the line $l(A_i, o)$ or the line $l(A_j, o)$. Note that $d_k \geq \max_{0 \leq h \leq n-1} |A_h X_h|$. Thus, $d_k \geq |A_i X_i|$ and $d_k \geq |A_j X_j|$. During the rotation of P, since both $A_i(P_a(t))$ and $A_j(P_b(t))$ are always larger than d_k, $P_a(t)$ cannot pass any of X_i and X_j, and thus $P_a(t)$ cannot arrive to the position z during the rotation. Hence, $A_i(P_a(t)) = A_j(P_b(t))$ cannot happen again after t_1.

  Further, recall that t_1 is the first moment from the beginning of the rotation with $A_i(P_a(t)) = A_j(P_b(t))$. Without loss of generality, we assume $A_i(P_a(t)) < A_j(P_b(t))$ for any time $t < t_1$ (as the example shown in Fig. 2(a)). It is easy to see that $A_i(P_a(t)) > A_j(P_b(t))$ for any time $t > t_1$, which implies that the two functions cross each other at their intersection point.
- $P_a \neq P_b$. At the moment t_1, we have $|A_i P_a(t_1)| = |A_j P_b(t_1)|$. Assume to the contrary that at some moment $t_2 > t_1$, we also have $A_i(P_a(t_2)) = A_j(P_b(t_2))$. Suppose at the moment t_2, $P_a(t_2)$ is at the position P'_a and $P_b(t_2)$ is at the position P'_b (see Fig. 2(b)). Then $|A_i P'_a| = |A_j P'_b|$. Since P_a and P_b are rotated

simultaneously, the arc distance from $P_a(t_1)$ to P'_a is equal to the arc distance from $P_b(t_1)$ to P'_b, and thus $|P_a(t_1)P'_a| = |P_b(t_1)P'_b|$. Consider the two triangles $\triangle P_b(t_1)A_jP'_b$ and $\triangle P_a(t_1)A_iP'_a$ (shown with red solid segments in Fig. 2(b)). Since $|A_iP_a(t_1)| = |A_jP_b(t_1)|$, $|A_iP'_a| = |A_jP'_b|$, and $|P_a(t_1)P'_a| = |P_b(t_1)P'_b|$, $\triangle P_b(t_1)A_jP'_b$ is congruent to $\triangle P_a(t_1)A_iP'_a$. Thus, the two angles $\angle A_iP_a(t_1)P'_a = \angle A_jP_b(t_1)P'_b$. Further, it is easy to see $\angle oP_a(t_1)P'_a = \angle oP_b(t_1)P'_b$. Consequently, we have $\angle oP_a(t_1)A_i = \angle oP_b(t_1)A_j$.

But, if $\angle oP_a(t_1)A_i = \angle oP_b(t_1)A_j$, then we can show that the two functions $A_i(P_a(t))$ and $A_j(P_b(t))$ define exactly the same curve segment. The proof is nothing but the inverse of the above argument. Specifically, consider any time moment $t_3 > t_1$ before the end of the rotation. Suppose at the moment t_3, P_a is at the position P''_a and P_b is at the position P''_b. Since $\angle oP_a(t_1)A_i = \angle oP_b(t_1)A_j$ and $\angle oP_a(t_1)P''_a = \angle oP_b(t_1)P''_b$, we have $\angle A_iP_a(t_1)P''_a = \angle A_jP_b(t_1)P''_b$. Further, since $|A_iP_a(t_1)| = |A_jP_b(t_1)|$ and $|P_a(t_1)P''_a| = |P_b(t_1)P''_b|$, $\triangle P_b(t_1)A_jP''_b$ is congruent to $\triangle P_a(t_1)A_iP''_a$. Thus, $|A_iP''_a| = |A_jP''_b|$, i.e., $A_i(P_a(t)) = A_j(P_b(t))$ at any time $t = t_3 > t_1$. Similarly, we can also show that at any time moment $t_3 < t_1$, $A_i(P_a(t_3)) = A_j(P_b(t_3))$. Hence, $A_i(P_a(t))$ and $A_j(P_b(t))$ define exactly the same curve segment. But this contradicts with the fact that the curve segments defined by these two functions are not the same. This implies that $A_i(P_a(t)) = A_j(P_b(t))$ cannot happen again at any moment $t > t_1$.

Further, without loss of generality, we assume $A_i(P_a(t)) < A_j(P_b(t))$ for any time $t < t_1$ (as the example shown in Fig. 2(b)). We then show that $A_i(P_a(t_3)) > A_j(P_b(t_3))$ for any time $t_3 > t_1$, which means that the two functions cross each other at their intersection point. We briefly discuss this. Again, suppose at the moment t_3, P_a is at the position P''_a and P_b is at the position P''_b. First, since $A_i(P_a(t)) < A_j(P_b(t))$ for any time $t < t_1$, it must be $|oA_j| > |oA_i|$ (this can be proved by similar techniques as above and we omit the details). Consider the two triangles $\triangle oA_iP_a(t_1)$ and $\triangle oA_jP_b(t_1)$ (at the moment t_1). Since $|A_iP_a(t_1)| = |A_jP_b(t_1)|$, $|oP_a(t_1)| = |oP_b(t_1)|$, and $|oA_j| > |oA_i|$, we have $\angle oP_b(t_1)A_j > \angle oP_a(t_1)A_i$, which further implies $\angle A_jP_b(t_1)P''_b < A_iP_a(t_1)P''_a$. Consider the triangles $\triangle P_b(t_1)A_jP''_b$ and $\triangle P_a(t_1)A_iP''_a$. Due to $|A_iP_a(t_1)| = |A_jP_b(t_1)|$, $|P_a(t_1)P''_a| = |P_b(t_1)P''_b|$, and $\angle A_jP_b(t_1)P''_b < A_iP_a(t_1)P''_a$, it must be $|A_jP''_b| < |A_iP''_a|$. In other words, $A_i(P_a(t)) > A_j(P_b(t))$ at any time $t = t_3 > t_1$.

The lemma thus follows. □

We further extend every curve segment in F' into an x-monotone curve, as follows. For each increasing (resp., decreasing) curve segment, we extend it by attaching two half-lines with slope 1 (resp., -1) at the two endpoints of that curve segment, respectively, such that the resulting new curve is still monotonically increasing (resp., decreasing). Denote the resulting new curve set by F. Obviously, an increasing curve and a decreasing curve in F intersect once and they cross each other at their intersection point. For any two different increasing (resp., decreasing) curves in F, by Lemma 4 and the way we extend the corresponding curve segments, they can intersect in at most one point and cross each other at their intersection point (if any). In other words, F can be viewed as a set of pseudolines. Let $\mathcal{A}_F$ be the arrangement of F. Observe that the elements in D'_2 are the y-coordinates of a subset of the vertices

of $\mathcal{A}_F$. Since $\lambda_C \in D'_2$, λ_C is the y-coordinate of a vertex of $\mathcal{A}_F$. Denote by $|\mathcal{A}_F|$ the number of vertices in $\mathcal{A}_F$. Of course, we do not want to compute the vertices of $\mathcal{A}_F$ explicitly. By generalizing some techniques in [10], we have the following lemma.

Lemma 5 *The value $|\mathcal{A}_F|$ can be computed in $O(n \log n)$ time. Given an integer k with $1 \le k \le |\mathcal{A}_F|$, the k-th highest vertex of $\mathcal{A}_F$ can be found in $O(n \log^2 n)$ time.*

Proof First of all, because every function in F' is of $O(1)$ complexity, we can determine in $O(1)$ time whether a curve segment in F' intersects a given horizontal line, and if "yes", then compute the intersection. Thus, for every curve in F, we can also compute its intersection with any horizontal line in $O(1)$ time. Let $N = |F| \le 2n$.

Recall that the curve segments in F' are all in the horizontal strip between $y = d_k$ and $y = d_{k+1}$. Thus, all vertices of $\mathcal{A}_F$ above the horizontal line $y = d_{k+1}$ are intersections of the newly attached half-lines. We can easily determine the highest vertex of $\mathcal{A}_F$ in $O(n \log n)$ time, e.g., by using the approach in [10]. Let l be a horizontal line higher than the highest vertex. Denote by $f_1, f_2, \ldots, f_N$ the sequence of the curves of F sorted in increasing order of the x-coordinates of their intersections with l. Similarly, we can determine the lowest vertex of $\mathcal{A}_F$ in $O(n \log n)$ time. Let $f_{\pi(1)}, f_{\pi(2)}, \ldots, f_{\pi(N)}$ be the sequence of the curves of F sorted in increasing order of the x-coordinates of their intersections with a horizontal line below the lowest vertex of $\mathcal{A}_F$. Since the curves in F can be viewed as a set of pseudolines, as in [10], the number of inversions in the permutation π, which can be computed in $O(n \log n)$ time, is equal to $|\mathcal{A}_F|$. In summary, we can compute $|\mathcal{A}_F|$ in $O(n \log n)$ time.

To compute the k-th highest vertex of $\mathcal{A}_F$, we choose to generalize the $O(n \log^2 n)$ time algorithm in [10]. Let L be a set of n lines in the plane and $\mathcal{A}_L$ be the arrangement of L. An $O(n \log^2 n)$ time algorithm was given in [10] for computing the k-th highest vertex of $\mathcal{A}_L$ ($1 \le k \le |\mathcal{A}_L|$) in $O(n \log^2 n)$ time based on parametric search [9, 21]. The main property used in the algorithm [10] is the following one. Denote by $l_1, l_2, \ldots, l_n$ the sequence of lines in L sorted in increasing order of their intersections with a horizontal line above the highest vertex of $\mathcal{A}_L$. Given any horizontal line l', let $l_{\pi(1)}, l_{\pi(2)}, \ldots, l_{\pi(n)}$ be the sequence of lines of L sorted in increasing order of their intersections with l'. Then, the number of vertices of $\mathcal{A}_L$ above l' is equal to the number of inversions in the permutation π.

In our problem, since any two curves in F can intersect each other in at most one point and they cross each other at their intersection point, the above property still holds for $\mathcal{A}_F$. Thus, the $O(n \log^2 n)$ time algorithm in [10] is applicable to our problem. Therefore, we can find the k-th highest vertex of $\mathcal{A}_F$ in $O(n \log^2 n)$ time, and the lemma follows. □

A remark: Optimal $O(n \log n)$ time algorithms were also given [10, 16] for finding the k-th highest vertex of $\mathcal{A}_L$. However, these algorithms are overly complicated. Further, even they can be made work for our problem, it does not benefit our overall solution for the optimization version because its total time is dominated by other parts of the algorithm. Hence, the much simpler $O(n \log^2 n)$ time solution (for finding the k-th highest vertex of $\mathcal{A}_F$) suffices for our purpose. In addition, the randomized $O(n \log n)$ time algorithms [11, 20] may also be used to finding the k-th highest vertex of $\mathcal{A}_F$.

Recall that λ_C is the y-coordinate of a vertex of $\mathcal{A}_F$. Our algorithm for computing λ_C then works as follows. First, compute $|\mathcal{A}_F|$. Next, find the $(|\mathcal{A}_F|/2)$-th highest vertex of $\mathcal{A}_F$, and denote its y-coordinate by λ_m. Determine whether $\lambda_C \leq \lambda_m$ by the algorithm in Theorem 2, after which one half of the vertices of $\mathcal{A}_F$ can be pruned away. We apply the above procedure recursively on the remaining vertices of $\mathcal{A}_F$, until λ_C is found. Since there are $O(\log n)$ recursive calls to this procedure, each of which takes $O(n \log^2 n)$, the total time for computing λ_C is $O(n \log^3 n)$.

Theorem 3 *The min-max optimization problem is solvable in $O(n \log^3 n)$ time.*

4 The Min-Sum Problem

In this section, we present our new algorithms for the min-sum problem. We show that the boundary case of this problem is solvable in $O(n^2)$ time, which improves the $O(n^4)$ time result in [23]. We also give an $O(n^2)$ time approximation algorithm with approximation ratio 3 for the general min-sum problem, which improves the $(1+\pi)$-approximation $O(n^2)$ time algorithm in [2]. Note that a PTAS algorithm was give in [2], which has a substantially larger polynomial time bound.

4.1 The Boundary Case

For the boundary case, the $O(n^4)$ time algorithm in [23] uses the $O(n^3)$ time Hungarian algorithm to compute a minimum weight perfect matching in a complete bipartite graph. However, the graph for this case is very special in the sense that all its vertices lie on the boundary of a circle. By using the result in [5], we can actually find a minimum weight perfect matching in such a graph in $O(n)$ time. Therefore, if we follow the algorithmic scheme in [23] but replace the Hungarian algorithm by the algorithm in [5], the boundary case can be solved in $O(n^2)$ time. For completeness, we give the details below.

Recall that in the boundary case of the min-sum problem, all sensors are on the boundary ∂C of C. Let $A_0, A_1, \ldots, A_{n-1}$ denote the initial positions of the n sensors on ∂C, and $A'_0, A'_1, \ldots, A'_{n-1}$ denote their goal positions on ∂C that form a regular n-gon. Denote by Δ_C the sum of the distances traveled by all n sensors in an optimal solution of the min-sum problem, i.e., $\Delta_C = \min \sum_{i=0}^{n-1} |A_i A'_i|$. Note that the moving paths of the sensors are not restricted on the boundary of C. The following lemma has been proved in [23].

Lemma 6 [23] *There exists an optimal solution for the boundary case of the min-sum problem with the following property*: *There exists a sensor A_i which does not move, i.e., $A_i = A'_i$.*

Based on Lemma 6, the boundary case can be solved as follows. For each sensor A_i, $0 \leq i \leq n-1$, let $P(A_i)$ be the regular n-gon on ∂C such that A_i is one of its vertices. Denote by H_i the complete bipartite graph between the set of all sensors and the set of all vertices of $P(A_i)$ such that the weight of an edge connecting a sensor

and a vertex of $P(A_i)$ is defined as their Euclidean distance. We compute a minimum weight perfect matching M_i in H_i, for each $0 \leq i \leq n-1$, and the one that gives the minimum weight defines an optimal solution for our original problem. Here, the weight of a perfect matching is the sum of all edge weights of the matching.

The running time of the above algorithm is dominated by the step of computing the minimum weight perfect matchings in the graphs H_i. The algorithm in [23] uses the $O(n^3)$ time Hungarian algorithm for computing such matchings in the graphs H_i.

Let H be a complete bipartite graph with two vertex sets of cardinalities n_1 and n_2, respectively, such that all its vertices lie on the boundary of a circle and each edge weight is the Euclidean distance between two such vertices (the edges are represented implicitly). A maximum cardinality matching of H consists of $\min\{n_1, n_2\}$ edges. An algorithm was given in [5] for computing a minimum weight maximum cardinality matching in H in $O(n_1 + n_2)$ time (i.e., the total sum of edge weights in the output maximum cardinality matching is as small as possible). Note that the graph H has the quasi-convex property [5].

Since in our algorithm, all vertices of every complete bipartite graph H_i lie on ∂C, the linear time algorithm in [5] can be applied to compute a minimum weight maximum cardinality matching of H_i in $O(n)$ time, for $0 \leq i \leq n-1$. (Note that a maximum cardinality matching in the graph H_i is a perfect matching, and vice versa.) Consequently, the total running time of our algorithm is $O(n^2)$.

Theorem 4 *The boundary case of the min-sum problem can be solved in $O(n^2)$ time.*

4.2 The General Case

We give our 3-approximation $O(n^2)$ time algorithm for the general min-sum problem.

Let $A_0, \ldots, A_{n-1}$ be the sensors in C. Our approximation algorithm works as follows. (1) For each sensor A_i, $i = 0, \ldots, n-1$, compute the point X_i on ∂C that is closest to A_i. (2) By using the algorithm in Theorem 4, solve the following min-sum boundary case problem: Viewing the n points $X_0, X_1, \ldots, X_{n-1}$ as *pseudo-sensors* (which all lie on ∂C), find n points on ∂C as the goal positions for the pseudo-sensors such that the sum of the distances traveled by all n pseudo-sensors is minimized. Let X_i' be the goal position for each X_i $(0 \leq i \leq n-1)$ in the optimal solution thus obtained. We then let X_i' be the goal position for each sensor A_i, $0 \leq i \leq n-1$, for our original min-sum problem. This completes the description of our approximation algorithm.

Clearly, with Theorem 4, the time complexity of the above approximation algorithm is $O(n^2)$. The lemma below shows that the approximation ratio of this algorithm is 3.

Lemma 7 *The approximation ratio of our approximation algorithm is* 3.

Proof Let $\Delta = \sum_{i=0}^{n-1} |A_i X_i'|$. Let $A_0^*, A_1^*, \ldots, A_{n-1}^*$ be the goal positions of all sensors (i.e., A_i^* is the goal position for each sensor A_i, $0 \leq i \leq n-1$) in an optimal solution for the min-sum problem. Let $\Delta_C = \sum_{i=0}^{n-1} |A_i A_i^*|$. Our task is to prove $\Delta \leq 3 \cdot \Delta_C$.

First, $\sum_{i=0}^{n-1} |X_i X_i'| \le \sum_{i=0}^{n-1} |X_i A_i^*|$, and $|A_i X_i| \le |A_i A_i^*|$ holds for each $0 \le i \le n-1$. Then,

$$\begin{aligned}
\Delta &= \sum_{i=0}^{n-1} |A_i X_i'| \le \sum_{i=0}^{n-1} \big(|A_i X_i| + |X_i X_i'|\big) \quad \text{(triangle inequality)} \\
&= \sum_{i=0}^{n-1} |A_i X_i| + \sum_{i=0}^{n-1} |X_i X_i'| \le \sum_{i=0}^{n-1} |A_i X_i| + \sum_{i=0}^{n-1} |X_i A_i^*| \\
&\le 2 \cdot \sum_{i=0}^{n-1} |A_i X_i| + \sum_{i=0}^{n-1} |A_i A_i^*| \quad \text{(triangle inequality)} \\
&\le 3 \cdot \sum_{i=0}^{n-1} |A_i A_i^*| = 3 \cdot \Delta_C.
\end{aligned}$$

The lemma thus follows. □

Hence, we conclude with the following result.

Theorem 5 *There exists an $O(n^2)$ time approximation algorithm for the min-sum problem with approximation ratio* 3.

5 Conclusions

In this paper, we present new algorithms for the min-max and min-sum versions for moving points to cover circular regions. Our results significantly improve the previous work. We also develop an algorithm for dynamically maintaining the maximum matching of a circular convex bipartite graph, which is of independent interest. One open problem is whether the general min-sum version is solvable in polynomial time.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Comput. Netw. **38**(4), 393–422 (2002)
2. Bhattacharya, B., Burmester, B., Hu, Y., Kranakis, E., Shi, Q., Wiese, A.: Optimal movement of mobile sensors for barrier coverage of a planar region. Theor. Comput. Sci. **410**(52), 5515–5528 (2009)
3. Bremner, D., Chan, T.M., Demaine, E.D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Taslakian, P.: Necklaces, convolutions, and X + Y. In: Proc. of the 14th Conference on Annual European Symposium on Algorithms, pp. 160–171 (2006)
4. Brodal, G., Georgiadis, L., Hansen, K.A., Katriel, I.: Dynamic matchings in convex bipartite graphs. In: Proc. of the 32nd International Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, vol. 4708, pp. 406–417. Springer, Berlin (2007)
5. Buss, S., Yianilos, P.: Linear and $O(n \log n)$ time minimum-cost matching algorithms for quasi-convex tours. SIAM J. Comput. **27**(1), 170–201 (1998)
6. Chang, M.S., Tang, C.Y., Lee, R.C.T.: Solving the Euclidean bottleneck matching problem by k-relative neighborhood graphs. Algorithmica **8**, 177–194 (1992)

7. Chen, A., Kumar, S., Lai, T.: Designing localized algorithms for barrier coverage. In: Proc. of the 13th Annual ACM International Conference on Mobile Computing and Networking, pp. 63–73 (2007)
8. Chen, D.Z., Wang, C., Wang, H.: Representing a functional curve by curves with fewer peaks. Discrete Comput. Geom. **46**(2), 334–360 (2011)
9. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. J. ACM **34**(1), 200–208 (1987)
10. Cole, R., Salowe, J., Steiger, W., Szemerédi, E.: An optimal-time algorithm for slope selection. SIAM J. Comput. **18**(4), 792–810 (1989)
11. Dillencourt, M.B., Mount, D.M., Netanyahu, N.S.: A randomized algorithm for slope selection. Int. J. Comput. Geom. Appl. **2**, 1–27 (1992)
12. Efrat, A., Katz, M.J.: Computing Euclidean bottleneck matchings in higher dimensions. Inf. Process. Lett. **75**, 169–174 (2000)
13. Efrat, A., Itai, A., Katz, M.: Geometry helps in bottleneck matching and related problems. Algorithmica **31**(1), 1–28 (2001)
14. Gabow, H., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. J. Comput. Syst. Sci. **30**, 209–221 (1985)
15. Hu, S.: 'Virtual Fence' along border to be delayed. Washington Post, February 28, 2008
16. Katz, M., Sharir, M.: Optimal slope selection via expanders. Inf. Process. Lett. **47**(3), 115–122 (1993)
17. Kumar, S., Lai, T., Arora, A.: Barrier coverage with wireless sensors. Wirel. Netw. **13**(6), 817–834 (2007)
18. Liang, Y.D., Blum, N.: Circular convex bipartite graphs: maximum matching and Hamiltonian circuits. Inf. Process. Lett. **56**, 215–219 (1995)
19. Lipski, W. Jr., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. Acta Inform. **15**(4), 329–346 (1981)
20. Matoušek, J.: Randomized optimal algorithm for slope selection. Inf. Process. Lett. **39**, 183–187 (1991)
21. Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms. J. ACM **30**(4), 852–865 (1983)
22. Steiner, G., Yeomans, J.: A linear time algorithm for maximum matchings in convex, bipartite graphs. Comput. Math. Appl. **31**(2), 91–96 (1996)
23. Tan, X., Wu, G.: New algorithms for barrier coverage with mobile sensors. In: Proc. of the 4th International Workshop on Frontiers in Algorithmics. Lecture Notes in Computer Science, vol. 6213, pp. 327–338. Springer, Berlin (2010)