


Algorithms for Minimizing the Movements of Spreading Points in Linear Domains*

Shimin Li ✉ 

Department of Computer Science, Winona State University, Winona, Minnesota 55987, USA

Haitao Wang ✉ 

Kahlert School of Computing, University of Utah, Salt Lake City, Utah 84112, USA

Abstract

We study a problem on spreading points. Given a set P of n points sorted on a line L and a distance value δ , we wish to move the points of P along L such that the distance of any two points of P is at least δ and the maximum movement over all points of P is minimized. Using the greedy strategy, we present an $O(n)$ time algorithm for this problem. Further, we extend our algorithm to solve (in $O(n)$ time) the cycle version of the problem where all points of P are on a cycle C . Previously, only weakly polynomial-time algorithms were known for these problems based on linear programming (of n variables and $\Theta(n)$ constraints). In addition, we present a linear-time algorithm for another similar facility-location moving problem, which improves on previous work.

Keywords and phrases points spreading, dispersion problem, facility-location, min-max, linear time

Digital Object Identifier 10.57717/cgt.v4i1.21

Acknowledgements We would like to thank Minghui Jiang for bringing these problems to us. We are also grateful to the anonymous referees for their careful reading and helpful suggestions that lead to improvements in the presentation of the paper.

1 Introduction

We consider the following *points-spreading* problem. Given a set P of n points sorted on a line L and a distance value $\delta \geq 0$, we wish to move the points of P along L such that the distance of any two points of P is at least δ and the maximum movement over all points of P is minimized. The above is the *line version*. We also consider the *cycle version* of the problem, where all points of P are given sorted cyclically on a cycle (one may view C as a simple closed curve). We wish to move the points of P on C such that the distance of any two points of P along C is at least δ and the maximum movement over all points of P along C is minimized. Note that since C is a cycle, the distance of any two points of C is defined to be the length of the shortest path on C between the two points.

Both versions of the problem have been studied before. By modeling them as linear programming problems (with n variables and $\Theta(n)$ constraints), Dumitrescu and Jiang [4] gave the first-known polynomial-time algorithms for both problems. Since there only exist weakly polynomial-time algorithms for linear programming [9, 10], it would be interesting to design strongly polynomial-time algorithms for the points-spreading problem. In this paper, we solve both versions of the problem not only in strongly polynomial time but also in $O(n)$ time (which is optimal). Our algorithms are based on a greedy strategy.

In addition, we consider a somewhat related problem, called the *facility-location movement* problem, defined as follows. Suppose we have a set of k “server” points and another set of n “client” points sorted on L . We wish to move all servers and all clients on L such that each client co-locates with a server and the maximum moving distance of all servers and clients

* A preliminary version of this paper appeared in Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015).



is minimized. Dumitrescu and Jiang [4] solved this problem in $O((n+k)\log(n+k))$ time (note that the algorithm works without assuming the servers and clients are given sorted on L ; but even if they are given sorted, their algorithm still runs in $O((n+k)\log(n+k))$ time). We present an $O(n+k)$ time algorithm based on their approach.

1.1 Related work

The 2D (Euclidean) version of the points-spreading problem was proposed by Demaine et al. [3] (also called “movement to independence” problem in [3, 4]). The problem in 2D is NP-hard and an approximation algorithm was given in [3]; the algorithm was improved later by Dumitrescu and Jiang [4].

The points-spreading problem is related to points dispersion problems which involve arranging a set of points as far away from each other as possible subject to certain constraints. For example, Fiala et al. in [6] studied such a problem in which one wants to place n given points, each inside its own, prespecified disk, with the objective of maximizing the distance between the closest pair of these points. The problem was shown to be NP-hard [6]. Approximation algorithms were given for this problem by Cabello [1]. Dumitrescu and Jiang [5] improved the approximation ratio and also proposed algorithms for the problem in high-dimensional spaces. In fact, Fiala et al. [6] studied the dispersion problem in a more general problem settings such as in other metric spaces. Another variation of the dispersion problem is to select a subset of facilities from a set of given facilities to maximize the minimum distance (or some other distance function) among all pairs of selected facilities [11, 12]. The problem is generally NP-hard (e.g., in 2D) but polynomial time algorithms are available in the one-dimensional space [11, 12]. In addition, Chandra and Halldórsson [2] studied dispersion problems in other problem settings, e.g., locating a set of points such that the sum of their distances to their nearest neighbor in the set is maximized.

Mehrdad Ghadiri and Sina Yazdanbod [8] studied a “min-sum” version of the points-spreading problem where all points on a line, and the objective is to minimize the total sum of the movements of all points. They gave an $O(n \log n)$ time algorithm for the problem.

The facility-location movement problem was first introduced by Demaine et al. [3] in general graphs, where servers and clients are all located on vertices of the graphs. The problem was proved to be NP-hard. A 2-approximation algorithm was presented in [3] for this problem, and later it was shown that the 2-approximation ratio cannot be improved unless $P=NP$ [7]. Dumitrescu and Jiang [4] studied the geometric version of this problem in the plane, and they showed that the problem is NP-hard to approximate within 1.8279. Fixed parameter algorithms (with the number of facilities k as the parameter) were also given in [4].

1.2 Our approach

To solve the line version of the points-spreading problem, essentially we first solve a “one-directional” case of the problem in which points are only allowed to move rightwards, by using a simple greedy algorithm. Suppose d is the maximum movement over all points in the solution of the above one-directional case. Then, we show that an optimal solution to the original problem can be obtained by shifting each point of P leftwards by the distance $d/2$ from its location in the above one-directional case solution.

To solve the cycle version of the problem, essentially we also first solve a one-directional case in which points are only allowed to move counterclockwise on C . If d is the maximum movement over all points in the solution of the one-directional case, then we also show that

an optimal solution to the original problem can be obtained by shifting each point of P clockwise by $d/2$. However, unlike the line version, the one-directional case of the problem becomes more difficult on the cycle. One straightforward idea is to cut the cycle C at a point of P (and extend C as a line) and then apply the algorithm for the one-directional case of the line version. However, the issue is that the last point may be too close to or even “cross” the first point if we put all points back on C . By observations, we show that if such a case happens, we can run the line-version algorithm for another round and the second round is guaranteed to find an optimal solution. Overall, the algorithm is still simple, but it is challenging to discover the idea and prove the correctness.

To solve the facility-location movement problem, Dumitrescu and Jiang [4] presented an $O((n+k)\log(n+k))$ time algorithm using dynamic programming. By discovering a monotonicity property on the dynamic programming, we improve Dumitrescu and Jiang’s algorithm to $O(n+k)$ time.

The rest of the paper is organized as follows. In Section 2, we present our algorithm for the line version of the points-spreading problem. The cycle version of the problem is solved in Section 3. Section 4 discusses our solution for the facility-location movement problem. Section 5 concludes the paper.

2 The line version of the points-spreading problem

In the line version, the points of P are given sorted on the line L . Without loss of generality, we assume L is the x -axis and $P = \{p_1, p_2, \dots, p_n\}$ is the set of points sorted by their x -coordinates from left to right. For each index $i \in \{1, \dots, n\}$, let x_i denote the location (or x -coordinate) of p_i on L . For any two locations x and x' of L , denote by $|xx'|$ the distance between x and x' , i.e., $|xx'| = |x - x'|$.

Our goal is to move each point $p_i \in P$ to a new location x'_i on L such that the distance of any pair of points of P is at least δ and the maximum moving distance, i.e., $\max_{1 \leq i \leq n} |x_i x'_i|$, is minimized. For simplicity of discussion, we make a general position assumption that no two points of P are at the same location in the input. The degenerate case can also be handled by our techniques but the discussions would be more tedious.

We refer to a specification of the location of each point p_i of P on L as a *configuration*. For example, in the input configuration each p_i is at x_i . Let F_0 denote the input configuration. A configuration is *feasible* if the distance between any pair of points of P is at least δ .

Denote by d_{opt} the maximum moving distance in any optimal solution. If the input configuration F_0 is feasible, then we do not need to move any point, implying that $d_{opt} = 0$. Since the points of P are sorted, we can check whether F_0 is feasible in $O(n)$ time by checking the distance between every adjacent pair of points of P . If F_0 is not feasible, then $d_{opt} > 0$. In the following, we assume F_0 is not feasible, and thus $d_{opt} > 0$.

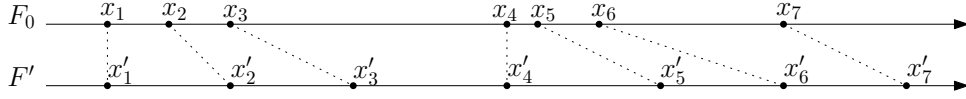
We first present some observations, based on which our algorithm will be developed.

2.1 Observations

For any two indices $i < j$ in $\{1, \dots, n\}$, define

$$w(i, j) = (j - i) \cdot \delta - |x_i x_j|.$$

As discussed by Dumitrescu and Jiang in [4], there exists an optimal solution in which the order of all points of P is the same as that in the input configuration F_0 . Based on this property, we prove Lemma 1 regarding the value d_{opt} .



■ **Figure 1** Illustrating our algorithm for computing the configuration F .

► **Lemma 1.** $d_{opt} \geq \max_{1 \leq i < j \leq n} \frac{w(i,j)}{2}$.

Proof. Consider any optimal solution OPT in which the order of all points of P is the same as that in F_0 . For each $1 \leq i \leq n$, let x_i^* be the location of p_i in OPT .

Consider any i and j with $1 \leq i < j \leq n$. Our goal is to prove $d_{opt} \geq w(i,j)/2$.

Since the points of P in OPT have the same order as in F_0 , for each k with $i < k \leq j$, we have $|x_{k-1}^* x_k^*| \geq \delta$ because OPT is a feasible solution. Hence, $|x_i^* x_j^*| = \sum_{k=i+1}^j |x_{k-1}^* x_k^*| \geq (j-i) \cdot \delta$.

If $|x_i^* x_j^*| - |x_i x_j| \leq 0$, then $|x_i x_j| \geq |x_i^* x_j^*| \geq (j-i) \cdot \delta$. Thus, $w(i,j) \leq 0$. Since $d_{opt} > 0$, $d_{opt} \geq w(i,j)/2$ holds.

If $|x_i^* x_j^*| - |x_i x_j| > 0$, then the difference of $|x_i^* x_j^*|$ and $|x_i x_j|$ is due to the moving of p_i and p_j . It is not difficult to see that $\max\{|x_i x_i^*|, |x_j x_j^*|\} \geq (|x_i^* x_j^*| - |x_i x_j|)/2$ (the equality happens when p_i moves leftwards by distance $(|x_i^* x_j^*| - |x_i x_j|)/2$ and p_j moves rightwards by the same distance). Since $d_{opt} \geq \max\{|x_i x_i^*|, |x_j x_j^*|\}$, it holds that $d_{opt} \geq (|x_i^* x_j^*| - |x_i x_j|)/2$. Due to $|x_i^* x_j^*| \geq (j-i) \cdot \delta$, we obtain that $d_{opt} \geq w(i,j)/2$.

The lemma thus follows. ◀

► **Lemma 2.** *If there exist i and j with $1 \leq i < j \leq n$ and a feasible configuration F' in which each point $p_k \in P$ moves rightwards to x'_k (i.e., $x_k \leq x'_k$) such that $w(i,j)$ satisfies $w(i,j) = \max_{1 \leq k \leq n} |x_k x'_k|$, then we can obtain an optimal solution by shifting each point of P in F' leftwards by distance $w(i,j)/2$.*

Proof. Let F'' denote the configuration obtained by shifting each point of P in F' leftwards by distance $w(i,j)/2$.

Consider any point $p_k \in P$. Let x''_k denote the location of p_k in F'' , i.e., $x''_k = x'_k - w(i,j)/2$. In order to prove that F'' is an optimal solution, by Lemma 1, it is sufficient to show that $|x_k x''_k| \leq w(i,j)/2$, as follows.

Indeed, $0 \leq x'_k - x_k \leq w(i,j)$, i.e., x'_k is at most $w(i,j)$ to the right of x_k . Therefore, after p_k is moved leftwards by $w(i,j)/2$ to x''_k , x''_k must be within distance $w(i,j)/2$ from x_k . Hence, $|x_k x''_k| \leq w(i,j)/2$. The lemma thus follows. ◀

We call a feasible configuration that satisfies the condition in Lemma 2 a *canonical configuration* (such as F' in Lemma 2). Due to Lemma 2, to solve the problem in linear time, it is sufficient to find a canonical configuration in linear time, which is our focus below.

2.2 Computing a canonical configuration

In this section, we present a linear-time algorithm that can find a canonical configuration. Comparing to the original problem, now we only need to consider the rightward movements.

Initially, we set $x'_1 = x_1$. Then we consider the rest of the points p_2, p_3, \dots, p_n from left to right. For each i with $2 \leq i \leq n$, suppose we have already moved p_{i-1} to x'_{i-1} . Then, we set $x'_i = \max\{x_i, x'_{i-1} + \delta\}$, and move p_i to x'_i . Refer to Fig. 1 for an example. The algorithm finishes after all points of P have been considered. Clearly, the algorithm runs in $O(n)$ time. Let F' denote the resulting configuration (i.e., each p_i is at x'_i).

In the following lemma, we show that F' is a canonical configuration.

► **Lemma 3.** F' is a canonical configuration.

Proof. First of all, based on our way of setting x'_i for $i = 1, 2, \dots, n$, it can be easily seen that every two points of P in F' are at least δ away from each other. Thus, F' is a feasible configuration. Note that $x'_i \geq x_i$ for any $i \in \{1, \dots, n\}$.

Next, we show that there exist i and j with $1 \leq i < j \leq n$ such that $w(i, j) = d_{max}$, where $d_{max} = \max_{1 \leq k \leq n} |x_k x'_k|$.

Recall that $d_{max} > 0$. Suppose the moving distance of p_j is the maximum, i.e., $d_{max} = |x_j x'_j|$. Let i be the largest index such that $i < j$ and p_i does not move in the algorithm (i.e., $x_i = x'_i$). Note that such a point p_i must exist as $x_1 = x'_1$ and $x'_j > x_j$.

For any point $p_k \in P$, if p_k is moved (rightwards) in F' (i.e., $x_k < x'_k$), then according to our way of setting x'_k , it must hold that $x'_k - x'_{k-1} = \delta$. By the definition of i , for each point p_k with $k \in \{i+1, \dots, j\}$, p_k is moved in F' , and thus $x'_k - x'_{k-1} = \delta$. Therefore, we obtain

$$|x'_i x'_j| = x'_j - x'_i = \sum_{i+1 \leq k \leq j} (x'_k - x'_{k-1}) = (j - i) \cdot \delta.$$

Since $x'_i = x_i$ and $x_j < x'_j$, we have $|x_i x'_j| = |x_i x_j| + |x_j x'_j|$. Hence, $d_{max} = |x_j x'_j| = |x_i x'_j| - |x_i x_j| = (j - i) \cdot \delta - |x_i x_j| = w(i, j)$.

This proves the lemma. ◀

Combining Lemmas 2 and 3, we conclude this section with the following theorem.

► **Theorem 4.** The line version of the points-spreading problem is solvable in $O(n)$ time.

Remark: One may verify that our algorithm for computing the canonical configuration F' essentially solves the following *one-directional case* of the line version problem: Move the points of P rightwards such that any pair of points of P are at least δ away from each other and the maximum moving distance over all points of P is minimized.

3 The cycle version of the points-spreading problem

In the cycle version, the points of $P = \{p_1, p_2, \dots, p_n\}$ are on a cycle C sorted cyclically, say, in the counterclockwise order. We use $|C|$ to denote the length of C . For any two locations x and x' on C , the distance between x and x' , denoted by $|xx'|$, is the length of the shortest path between x and x' on C . Clearly, $|xx'| \leq |C|/2$. For each $i \in \{1, \dots, n\}$, we use x_i to denote the location of p_i on C in the input. Our goal is to move each point $p_i \in P$ to a new location x'_i such that the distance of any pair of two points of P on C is at least δ and the maximum moving distance, i.e., $\max_{1 \leq i \leq n} |x_i x'_i|$, is minimized.

We assume $|C| \geq \delta \cdot n$ since otherwise there would be no solution. Again, for simplicity of discussion, we make a general position assumption that no two points of P are at the same location on C in the input.

As in the line version, we refer to a specification of the location of each point of P on C as a *configuration*. A configuration is *feasible* if the distance between any pair of points of P is at least δ . Let F_0 denote the input configuration.

Denote by d_{opt} the maximum moving distance in any optimal solution. If F_0 is feasible, then $d_{opt} = 0$. We can also check whether F_0 is feasible in $O(n)$ time. If F_0 is not feasible, then $d_{opt} > 0$. In the following, we assume F_0 is not feasible, and thus $d_{opt} > 0$.

To solve the cycle version of the problem, we extend our algorithm (and observations) for the line version in Section 2. Namely, we first move all points of P on C counterclockwise to

obtain a “canonical configuration”, and then shift all points clockwise. However, as will be seen later, the problem becomes much more difficult on the cycle.

Consider any two locations x and x' on C . We define $C(x, x')$ as the portion of C from x to x' counterclockwise. We use $|C(x, x')|$ to denote the length of $C(x, x')$. Note that $|xx'| = \min\{|C(x, x')|, |C(x', x)|\}$.

As in the line version, we first give some observations, based on which our algorithms will be developed.

3.1 Observations

For any two indices $i \neq j$ in $\{1, \dots, n\}$, define

$$w(i, j) = ((n + j - i) \bmod n) \cdot \delta - |C(x_i, x_j)|.$$

In words, if $i < j$, then $w(i, j) = (j - i) \cdot \delta - |C(x_i, x_j)|$; otherwise, $w(i, j) = (n + j - i) \cdot \delta - |C(x_i, x_j)|$. Roughly speaking, $w(i, j)$ measures the distance difference in the clockwise direction for two points p_i and p_j . Since $|C| \geq \delta \cdot n$, it can be verified that $w(i, j) \leq |C|$.

As discussed by Dumitrescu and Jiang in [4], there exists an optimal solution in which the order of all points of P is the same as that in the input configuration F_0 . Using this property, we prove Lemma 5, which is analogous to Lemma 2 for the line version.

► **Lemma 5.** $d_{opt} \geq \max_{1 \leq i, j \leq n} \frac{w(i, j)}{2}$.

Proof. Consider any optimal solution OPT in which the order of all points of P is the same as that in the input configuration F_0 . For each $1 \leq k \leq n$, let x_k^* be the location of x_k in OPT .

Consider any two indices $i \neq j$ in $\{1, \dots, n\}$. To prove the lemma, the goal is to show that $d_{opt} \geq w(i, j)/2$. Depending on whether $i < j$, there are two cases. Below we only prove the case $i < j$, and the other case is very similar.

First of all, we claim that $|C(x_i^*, x_j^*)| \geq (j - i) \cdot \delta$. Indeed, consider any $k \in \{i + 1, \dots, j\}$. Since OPT is an optimal solution, which is a feasible solution, $|x_{k-1}^* x_k^*| \geq \delta$ holds. Because $|x_{k-1}^* x_k^*| = \min\{|C(x_{k-1}^*, x_k^*)|, |C(x_k^*, x_{k-1}^*)|\}$, we obtain that $|C(x_{k-1}^*, x_k^*)| \geq \delta$. Since the order of the points of P in OPT is the same as that in F_0 , we have $C(x_i^*, x_j^*) = \cup_{k=i+1}^j C(x_{k-1}^*, x_k^*)$ and $|C(x_i^*, x_j^*)| = \sum_{k=i+1}^j |C(x_{k-1}^*, x_k^*)| \geq (j - i) \cdot \delta$. The claim is thus proved.

In the sequel, we prove $d_{opt} \geq w(i, j)/2 = ((j - i) \cdot \delta - |C(x_i, x_j)|) / 2$.

If $|C(x_i^*, x_j^*)| - |C(x_i, x_j)| \leq 0$, then since $|C(x_i^*, x_j^*)| \geq (j - i) \cdot \delta$, it holds that $|C(x_i, x_j)| \geq (j - i) \cdot \delta$. Hence, $w(i, j) \leq 0$, and it follows that $d_{opt} \geq w(i, j)/2$.

If $|C(x_i^*, x_j^*)| - |C(x_i, x_j)| > 0$, then the difference of $|C(x_i^*, x_j^*)|$ and $|C(x_i, x_j)|$ is due to the moving of p_i and p_j . Because the order of the points of P in OPT is the same as that in F_0 , the smallest moving distance of these two points happens when x_i and x_j move in opposite directions (i.e., x_i moves clockwise and x_j moves counterclockwise) by the same distance $(|C(x_i^*, x_j^*)| - |C(x_i, x_j)|) / 2$. Therefore, we obtain $\max\{|x_i x_i^*|, |x_j x_j^*|\} \geq (|C(x_i^*, x_j^*)| - |C(x_i, x_j)|) / 2$. Since $d_{opt} \geq \max\{|x_i x_i^*|, |x_j x_j^*|\}$, $d_{opt} \geq (|C(x_i^*, x_j^*)| - |C(x_i, x_j)|) / 2$ holds. Finally, because $|C(x_i^*, x_j^*)| \geq (j - i) \cdot \delta$, we obtain $d_{opt} \geq w(i, j)/2$. ◀

Based on Lemma 5, we obtain the following lemma, which is analogous to Lemma 3 for the line version.

► **Lemma 6.** *If there exist $i \neq j$ in $\{1, \dots, n\}$ and a feasible configuration F' in which each point $p_k \in P$ is at location x'_k such that $w(i, j) = \max_{1 \leq k \leq n} |C(x_k, x'_k)|$, then we can obtain an optimal solution by shifting every point of P in F' clockwise by distance $w(i, j)/2$.*

Proof. Let F'' denote the configuration obtained by shifting every point of P in F' clockwise by distance $w(i, j)/2$.

Consider any point $p_k \in P$. Let x_k'' denote the location of x_k in F'' . On the one hand, $|C(x_k, x_k')| \leq w(i, j)$ since $w(i, j) = \max_{1 \leq k \leq n} |C(x_k, x_k')|$. On the other hand, since the above shifting moves p_k from x_k' clockwise to x_k'' by distance $w(i, j)/2 \leq |C|/2$ (recall that $w(i, j) \leq |C|$), it holds that either $|C(x_k, x_k'')| \leq w(i, j)/2$ or $|C(x_k'', x_k)| \leq w(i, j)/2$. Consequently, $|x_k x_k''| = \min\{|C(x_k, x_k'')|, |C(x_k'', x_k)|\} \leq w(i, j)/2$.

The above shows that $\max_{1 \leq k \leq n} |C(x_k, x_k'')| \leq w(i, j)/2$, i.e., the maximum moving distance over all points of P in F'' is no more than $w(i, j)/2$. By Lemma 5, F'' is an optimal solution. The lemma is thus proved. \blacktriangleleft

We call a feasible configuration that satisfies the condition in Lemma 6 a *canonical configuration*. In light of Lemma 6, to solve the problem in linear time, it is sufficient to find a canonical configuration in linear time, which is our focus below.

3.2 Computing a canonical configuration

In this section, we present a linear-time algorithm that can find a canonical configuration. Comparing to the original problem, now we only need to consider the counterclockwise movements.

Recall that the points p_1, p_2, \dots, p_n are ordered on C counterclockwise in the input configuration F_0 . For convenience of discussion, we define coordinates for locations on C in the following way. We define x_1 as the origin with coordinate 0. For any other location $x \in C$, the coordinate of x is defined to be $|C(x_1, x)|$. Hence each location of C has a coordinate no greater than $|C|$.

Our algorithm has two rounds. In the first round, we will use the same approach as for the line version of the problem, and let F_1 denote the resulting configuration. However, the issue is that in F_1 the new location of p_n may be too close to p_1 or p_n may even “cross” p_1 , which might make F_1 not feasible. If p_n does not cross p_1 and p_n is at least δ away from p_1 in F_1 , then we will show that F_1 is a canonical configuration. Otherwise, we will proceed to the second round, which is to (starting from the configuration F_1) consider all points again from p_1 and use the same strategy to set the new locations of the points (to ensure the distance between p_n and p_1 is at least δ). We will show that the configuration F_2 obtained after the second round is a canonical configuration. The details are given below.

3.2.1 The first round

In the first round, we will move each point $p_i \in P$ from x_i along C counterclockwise to a new location x_i' . The way we set x_i' here is similar to that in the line version and the difference is that we have to take care of the cycle situation. Specifically, $x_1' = x_1$, i.e., p_1 does not move. For each $i \in \{2, \dots, n\}$, suppose we have already moved p_{i-1} to x_{i-1}' , then we define x_i' as follows:

$$x_i' = \begin{cases} x_i & \text{if } x_i \geq x_{i-1}' + \delta \\ (x_{i-1}' + \delta) \bmod |C| & \text{if } x_i < x_{i-1}' + \delta. \end{cases} \quad (1)$$

This finishes the first round of our algorithm. Denote by F_1 the resulting configuration.

Note that if $x_{i-1}' + \delta > |C|$, then since $x_i \leq |C|$, according to Equation (1), $x_i' = (x_{i-1}' + \delta) \bmod |C|$, which is equal to $x_{i-1}' + \delta - |C|$; in this case, we say that the counterclockwise movement of p_i *crosses* the origin x_1 .

► **Lemma 7.** *If p_n does not cross $x_1 (= x'_1)$ in the first round of the algorithm and $|C(x'_n, x'_1)| \geq \delta$, then F_1 is a canonical configuration.*

Proof. First of all, we show that F_1 is a feasible configuration, i.e., the distance between any two points of P in F_1 is at least δ . Consider any two indices i and j . Without loss of generality, assume $i < j$. Our goal is to show that $|x'_i x'_j| \geq \delta$. To this end, it is sufficient to show that $|C(x'_i, x'_j)| \geq \delta$ and $|C(x'_j, x'_i)| \geq \delta$.

On the one hand, $C(x'_i, x'_j)$ contains x'_{i+1} , implying that $C(x'_i, x'_{i+1}) \subseteq C(x'_i, x'_j)$ and thus $|C(x'_i, x'_{i+1})| \leq |C(x'_i, x'_j)|$ (note that $j = i + 1$ is possible). According to our first round algorithm (i.e., Equation (1)), it holds that $|C(x'_i, x'_{i+1})| \geq \delta$. Thus, $|C(x'_i, x'_j)| \geq \delta$.

On the other hand, since p_n does not cross $x_1 = x'_1$, $C(x'_j, x'_i)$ contains both x'_n and x'_1 , and in other words, $C(x'_n, x'_1) \subseteq C(x'_j, x'_i)$. Due to $|C(x'_n, x'_1)| \geq \delta$, we obtain $|C(x'_j, x'_i)| \geq |C(x'_n, x'_1)| \geq \delta$.

Therefore, F_1 is a feasible configuration.

Let d'_{max} be the maximum counterclockwise movement over all points of P in the first round, i.e., $d'_{max} = \max_{1 \leq k \leq n} |C(x_k, x'_k)|$. To show that F_1 is a canonical configuration, we also need to show that there exist i and j such that $d'_{max} = w(i, j)$. In the following, we will find two indices i and j with $i < j$ such that $d'_{max} = w(i, j)$. Recall that when $i < j$, $w(i, j) = (j - i) \cdot \delta - |C(x_i, x_j)|$.

Since the input configuration F_0 is not feasible, it must hold that $d'_{max} > 0$. Let j be the index such that $d'_{max} = |C(x_j, x'_j)|$. Let i be the largest index such that $i < j$ and $x'_i = x_i$. Note that such an index i must exist since $x_1 = x'_1$.

According to the definition of i , each point x_k with $i + 1 \leq k \leq j$ is moved in the first round algorithm, which implies that $|C(x'_{k-1}, x'_k)| = \delta$ according to Equation (1). Hence, we obtain $|C(x'_i, x'_j)| = \sum_{k=i+1}^j |C(x'_{k-1}, x'_k)| = (j - i) \cdot \delta$. On the other hand, since the movement of p_n does not cross x_1 and p_i does not move, the movement of p_j does not cross $x_i = x'_i$. Thus, $C(x'_i, x'_j) = C(x_i, x_j) \cup C(x_j, x'_j)$ and $|C(x'_i, x'_j)| = |C(x_i, x_j)| + |C(x_j, x'_j)|$.

Therefore, we obtain $d'_{max} = |C(x_j, x'_j)| = |C(x'_i, x'_j)| - |C(x_i, x_j)| = (j - i) \cdot \delta - |C(x_i, x_j)| = w(i, j)$.

We conclude that F_1 is a canonical configuration. ◀

According to Lemma 7, if p_n does not cross $x_1 = x'_1$ in the first round and $|C(x'_n, x'_1)| \geq \delta$ in F_1 , then we have found a canonical configuration and our algorithm stops. Otherwise, we proceed to the second round, as follows.

3.2.2 The second round

In the second round, we will move each point $p_i \in P$ from x'_i counterclockwise to a new location x''_i , defined as follows.

We first define x''_1 . Recall that we proceed to the second round because either p_n crosses $x_1 = x'_1$ in the first round or $|C(x'_n, x'_1)| < \delta$. In either case we define

$$x''_1 = (x'_n + \delta) \pmod{|C|}. \quad (2)$$

Hence, $|C(x'_n, x''_1)| = \delta$.

For each $i = 2, 3, \dots, n$, suppose p_{i-1} has been moved to x''_{i-1} ; then we move p_i from x'_i counterclockwise to x''_i , with

$$x''_i = \max\{x'_i, (x''_{i-1} + \delta) \pmod{|C|}\} \quad (3)$$

This finishes the second round of our algorithm. Let F_2 be the resulting configuration. In the sequel we show that F_2 is a canonical configuration.

Recall that $|C| \geq n \cdot \delta$. We first have the following observation on the first round of the algorithm.

► **Observation 8.** *There must be a point p_i with $i \in \{2, \dots, n\}$ such that p_i does not move in the first round of the algorithm (i.e., $x_i = x'_i$).*

Proof. Assume to the contrary that every point p_i with $i \in \{2, \dots, n\}$ is moved in the first round. Then, by our first round algorithm (i.e., Equation (1)), $|C(x'_{i-1}, x'_i)| = \delta$ for each $2 \leq i \leq n$. Hence, $|C(x'_1, x'_n)| = \sum_{i=2}^n |C(x'_{i-1}, x'_i)| = (n-1) \cdot \delta$. Further, since either p_n crosses $x_1 = x'_1$ or $|C(x'_n, x'_1)| < \delta$, we obtain that $n \cdot \delta > |C|$, which contradicts with the fact that $|C| \geq n \cdot \delta$. ◀

► **Observation 9.** *If a point p_i does not move in the second round, then for each point p_j with $j \in \{i, \dots, n\}$, p_j does not move in the second round either.*

Proof. If $i = n$, then the observation trivially follows. We assume $i < n$.

According to the first round algorithm, it holds that $|C(x'_{k-1}, x'_k)| \geq \delta$ for any $k \in \{2, \dots, n\}$. Since p_i does not move in the second round, $x''_i = x'_i$ holds. Due to $|C(x'_i, x'_{i+1})| \geq \delta$, according to our second round algorithm (e.g., Equation (3)), $x''_{i+1} = x'_{i+1}$. By the same reasoning, $x''_j = x'_j$ for any $j \in \{i+1, \dots, n\}$, which leads to the observation. ◀

With Observations 8 and 9, we can prove the following lemma.

► **Lemma 10.** *Suppose k is the largest index such that p_k does not move in the first round of the algorithm; then p_k does not move in the second round of the algorithm either, i.e., $x_k = x'_k = x''_k$.*

Proof. According to the first round algorithm, it holds that $|C(x'_{i-1}, x'_i)| \geq \delta$ for any $i \in \{2, \dots, n\}$.

By Observation 8, $k \in \{2, \dots, n\}$. We first discuss the case where $k \in \{3, \dots, n-1\}$. Indeed, this is the most general case. As shown later, the case where $k = 2$ or $k = n$ can be proved by similar but simpler techniques.

By the definition of k , the points $p_{k+1}, p_{k+2}, \dots, p_n$ are moved in the first round. Hence, for each $i \in \{k+1, \dots, n\}$, according to our first round algorithm (i.e., Equation (1)), $|C(x'_{i-1}, x'_i)| = \delta$. Thus,

$$|C(x'_k, x'_n)| = \sum_{i=k+1}^n |C(x'_{i-1}, x'_i)| = (n-k) \cdot \delta. \quad (4)$$

Recall that p_1 is moved in the second round, and according to Equation (2),

$$|C(x'_n, x''_1)| = \delta. \quad (5)$$

If there is any $i \in \{2, \dots, k-1\}$ such that p_i does not move in the second round, then by Observation 9, p_k does not move in the second round either, which leads to the lemma.

Otherwise, since every point p_i with $i \in \{2, \dots, k-1\}$ is moved in the second round, according to our second round algorithm (i.e., Equation (3)), $|C(x''_{i-1}, x''_i)| = \delta$ holds. Hence, we obtain

$$|C(x''_1, x''_{k-1})| = \sum_{i=2}^{k-1} |C(x''_{i-1}, x''_i)| = (k-2) \cdot \delta. \quad (6)$$

Based on Equations (4), (5), and (6), we obtain $|C(x'_k, x'_n)| + |C(x'_n, x''_1)| + |C(x''_1, x''_{k-1})| = (n-1) \cdot \delta$. This implies that in the second round the counterclockwise movement of p_{k-1} from x'_{k-1} to x''_{k-1} does not cross $x_k = x'_k$, due to $|C| \geq n \cdot \delta$. Further, $|C(x''_{k-1}, x'_k)| = |C| - |C(x'_k, x''_{k-1})| = |C| - (|C(x'_k, x'_n)| + |C(x'_n, x''_1)| + |C(x''_1, x''_{k-1})|) = |C| - (n-1) \cdot \delta \geq \delta$. According to our second round algorithm (i.e., Equation (3)), $x''_k = x'_k$, i.e., p_k does not move in the second round.

The above proves the lemma for the case where $k \in \{3, \dots, n-1\}$.

If $k = 2$ or $k = n$, the proof is very similar.

If $k = 2$, then we still have Equations (4) and (5). Thus, $|C(x'_2, x'_n)| + |C(x'_n, x''_1)| = (n-1) \cdot \delta$. This implies that in the second round the counterclockwise movement of p_1 from x'_1 to x''_1 does not cross $x_2 = x'_2$, due to $|C| \geq n \cdot \delta$. Further, $|C(x''_1, x'_2)| = |C| - |C(x'_2, x''_1)| = |C| - (|C(x'_2, x'_n)| + |C(x'_n, x''_1)|) = |C| - (n-1) \cdot \delta \geq \delta$. According to our second round algorithm (i.e., Equation (3)), $x''_2 = x'_2$, i.e., p_2 does not move in the second round. Hence, the lemma is proved.

If $k = n$, then we still have Equations (5) and (6). Thus, $|C(x'_n, x''_1)| + |C(x''_1, x''_{n-1})| = (n-1) \cdot \delta$. This implies that in the second round when p_{n-1} moved from x'_{n-1} to x''_{n-1} , p_{n-1} does not cross $x_n = x'_n$, due to $|C| \geq n \cdot \delta$. Further, $|C(x''_{n-1}, x'_n)| = |C| - |C(x'_n, x''_{n-1})| = |C| - (|C(x'_n, x''_1)| + |C(x''_1, x''_{n-1})|) = |C| - (n-1) \cdot \delta \geq \delta$. According to our second round algorithm (i.e., Equation (3)), $x''_n = x'_n$, i.e., p_n does not move in the second round. Hence, the lemma follows.

In summary, p_k does not move in the second round of the algorithm. ◀

Recall that F_2 is the configuration after the second round of the algorithm. Our goal is to prove that F_2 is a canonical configuration. Based on the proof of Lemma 10, we have the following two corollaries.

► **Corollary 11.** *The configuration F_2 is feasible.*

Proof. Suppose p_k is the point specified in Lemma 10. Hence, $k \in \{2, \dots, n\}$ and p_k does not move in the two rounds of our algorithm. We only prove the case where $k \in \{2, \dots, n-1\}$, and the case $k = n$ can be proved by similar (but simpler) techniques.

After the first round, it holds that $|C(x'_{i-1}, x'_i)| \geq \delta$ for each $i \in \{k+1, \dots, n\}$. Since x_k does not move in the second round, by Observation 9, $x''_i = x'_i$ for any $i \in \{k, \dots, n\}$. Hence, for each $i \in \{k+1, \dots, n\}$, it holds that $|C(x''_{i-1}, x''_i)| \geq \delta$.

On the other hand, according to our second round algorithm, $|C(x'_n, x''_1)| \geq \delta$ and $|C(x''_{i-1}, x''_i)| \geq \delta$ for each $i \in \{2, \dots, k\}$. Since $x'_n = x''_n$, it holds that $|C(x''_n, x''_1)| = |C(x'_n, x''_1)| \geq \delta$.

The above discussion leads to the following observation: $x''_1, x''_2, \dots, x''_n$ are ordered counterclockwise on C , and further, for each $i \in \{2, \dots, n\}$, $|C(x''_{i-1}, x''_i)| \geq \delta$, and $|C(x''_n, x''_1)| \geq \delta$.

To show that F_2 is feasible, our goal is to prove that $|x''_i x''_j| \geq \delta$ for any $i \neq j \in \{1, \dots, n\}$. Consider any $i \neq j \in \{1, \dots, n\}$. Without loss of generality, we assume $i < j$. To prove $|x''_i x''_j| \geq \delta$, it is sufficient to show that $|C(x''_i, x''_j)| \geq \delta$ and $|C(x''_j, x''_i)| \geq \delta$.

Based on the above observation, we can obtain $C(x''_i, x''_{i+1}) \subseteq C(x''_i, x''_j)$ and $|C(x''_i, x''_j)| \geq |C(x''_i, x''_{i+1})| \geq \delta$. On the other hand, $C(x''_n, x''_1) \subseteq C(x''_j, x''_i)$. Since $|C(x''_n, x''_1)| \geq \delta$, we have $|C(x''_j, x''_i)| \geq |C(x''_n, x''_1)| \geq \delta$.

Therefore, $|x''_i x''_j| \geq \delta$ holds. The corollary thus follows. ◀

► **Corollary 12.** *The total counterclockwise moving distance of each point of P in the two rounds of the algorithm is at most $|C| - \delta$, which implies that $|C(x_i, x''_i)| \leq |C| - \delta$ for each $1 \leq i \leq n$.*

Proof. By Lemma 10, suppose p_k does not move in the two rounds of our algorithm. For each other point p_i with $i \neq k$, since p_k does not move in the algorithm, the counterclockwise movement of p_i in the two rounds of the algorithm does not cross x_k . Further, as shown in the proof of Corollary 11, both $|C(x_k, x_i'')| \geq \delta$ and $|C(x_i'', x_k)| \geq \delta$ hold. Hence, the maximum counterclockwise movement of p_i in the two rounds is no more than $|C| - \delta$. The corollary follows. \blacktriangleleft

Finally, the next lemma shows that F_2 is a canonical configuration.

► **Lemma 13.** *The configuration F_2 is a canonical configuration.*

Proof. Corollary 11 has already shown that F_2 is a feasible configuration. To prove the lemma, it is sufficient to prove that there exist i and j in $\{1, \dots, n\}$ such that $d_{max} = w(i, j)$, where $d_{max} = \max_{1 \leq k \leq n} |C(x_k, x_k'')|$.

Let j be the index such that $d_{max} = |C(x_j, x_j'')|$. We define another index i as follows. If $j = 1$, or $j > 1$ but all points of p_1, p_2, \dots, p_{j-1} are moved in the two rounds of the algorithm, let i be the largest index in $\{j+1, \dots, n\}$ such that p_i does not move in the two rounds of the algorithm; otherwise (i.e., $j > 1$ and at least one point of p_1, p_2, \dots, p_{j-1} does not move in the two rounds of the algorithm), let i be the largest index in $\{1, \dots, j-1\}$ such that p_i does not move in the two rounds of the algorithm. By Lemma 10, such an index i must exist. In the following, we prove that $d_{max} = w(i, j)$.

Depending on whether $i \in \{1, \dots, j-1\}$ or $i \in \{j+1, \dots, n\}$, there are two cases.

1. If $i \in \{1, \dots, j-1\}$, then by the definition of i , all points $p_{i+1}, p_{i+2}, \dots, p_j$ are moved in the algorithm. Since p_i does not move in the second round, by Observation 9, for each $k \in \{i+1, \dots, n\}$, p_k does not move in the second round. This implies that every point of $p_{i+1}, p_{i+2}, \dots, p_j$ is moved in the first round of the algorithm. According to our first round algorithm, $|C(x'_{k-1}, x'_k)| = \delta$ for each $k \in \{i+1, \dots, j\}$. Hence, $|C(x_i, x_j'')| = |C(x_i'', x_j'')| = \sum_{k=i+1}^j |C(x''_{k-1}, x''_k)| = (j-i) \cdot \delta$ (because $x''_k = x'_k$ for each $k \in \{i, \dots, n\}$).

Since $i < j$ and $x_i = x'_i = x''_i$, $C(x_i, x_j'') = C(x_i, x_j) \cup C(x_j, x_j'')$. Thus, $d_{max} = |C(x_j, x_j'')| = |C(x_i, x_j'')| - |C(x_i, x_j)| = (j-i) \cdot \delta - |C(x_i, x_j)|$, which is equal to $w(i, j)$ since $i < j$.

Hence, the lemma is proved for this case.

2. If $i \in \{j+1, \dots, n\}$, we only discuss the general case where $i < n$. The special case where $i = n$ can be proved by similar (but simpler) techniques.

Consider any point p_k with $k \in \{i+1, \dots, n\}$. Since p_i does not move in the two rounds of the algorithm, by Observation 9, p_k does not move in the second round. According to the definition of i , p_k is moved in the algorithm. Hence, p_k is moved in the first round. According to our first round algorithm (i.e., Equation (1)), $|C(x'_{k-1}, x'_k)| = \delta$. Further, since $x''_k = x'_k$, $|C(x''_{k-1}, x''_k)| = \delta$ holds. Therefore, $|C(x_i'', x_n'')| = \sum_{k=i+1}^n |C(x''_{k-1}, x''_k)| = (n-i) \cdot \delta$.

Since p_1 is moved in the second round, by Equation (2), $|C(x'_n, x''_1)| = \delta$. We have shown above that p_k does not move in the second round for any $k \in \{i+1, \dots, n\}$. Hence, $x''_n = x'_n$ and $|C(x''_n, x''_1)| = \delta$.

If $j = 1$, then $|C(x_i'', x''_1)| = |C(x_i'', x''_n)| + |C(x''_n, x''_1)| = (n+1-i) \cdot \delta$. Further, since p_i does not move in the algorithm (i.e., $x''_i = x'_i = x_i$), $d_{max} = |C(x_1, x''_1)| = |C(x_i, x''_1)| - |C(x_i, x_1)| = (n+1-i) \cdot \delta - |C(x_i, x_1)|$, which is equal to $w(i, 1)$. The lemma thus follows.

In the following, we discuss the case $j > 1$.

Consider any point p_k with $k \in \{2, \dots, j\}$.

We claim that p_k is moved in the second round (i.e., $x'_k \neq x''_k$). We prove the claim by induction. Indeed, by the definition of i , p_k is moved in the two rounds of the algorithm. Recall that p_1 is moved in the second round. For any $k \in \{2, \dots, j\}$, suppose p_{k-1} is moved in the second round. Assume to the contrary that p_k does not move in the second round. Then, p_k must be moved in the first round. According to our first round algorithm (i.e., Equation (1)), $|C(x'_{k-1}, x'_k)| = \delta$. Since p_{k-1} is moved in the second round, p_k must be moved as well.

In light of the above claim and according to our second round algorithm, $|C(x''_{k-1}, x''_k)| = \delta$ for each $k \in \{2, \dots, j\}$. Therefore, we derive $|C(x''_1, x''_j)| = \sum_{k=2}^j |C(x''_{k-1}, x''_k)| = (j-1) \cdot \delta$. Based on the above discussions, $|C(x''_i, x''_j)| = |C(x''_i, x''_n)| + |C(x''_n, x''_1)| + |C(x''_1, x''_j)| = (n+j-i) \cdot \delta$. Since $x_i = x''_i$, $d_{\max} = |C(x_j, x''_j)| = |C(x_i, x''_j)| - |C(x_i, x_j)| = (n+j-i) \cdot \delta - |C(x_i, x_j)|$, which is equal to $w(i, j)$.

As a summary, F_2 is a canonical configuration. ◀

Clearly, both rounds of our algorithm run in $O(n)$ time. Combining Lemmas 6, 7, and 13, we have the following result.

► **Theorem 14.** *The cycle version of the points-spreading problem is solvable in $O(n)$ time.*

Remark: One may verify that our algorithm for computing the canonical configuration F_2 essentially solves the following *one-directional case* of the cycle version problem: Move the points of P counterclockwise such that any pair of points of P are at least δ away from each other and the maximum counterclockwise moving distance over all points of P is minimized.

4 The facility-location movement problem

In this section, we present our linear-time algorithm for the facility-location movement problem. In this problem, we are given a set S of k “server” points and a set Q of n “client” points sorted on a line L , and the goal is to move all servers and clients on L such that each client co-locates with a server and the maximum moving distance of all servers and clients is minimized.

As shown by Dumitrescu and Jiang [4], the problem is equivalent to finding k intervals (i.e., line segments) on L such that each interval contains at least one server, each client is covered by at least one interval, and the maximum length of these intervals is minimized. In the following, we will focus on solving this *interval coverage* problem (also called *constrained k -center* problem in [4]).

Dumitrescu and Jiang [4] presented an $O((n+k) \log(n+k))$ time algorithm using dynamic programming. We discover a monotonicity property on their dynamic programming scheme, and consequently improve their algorithm to $O(n+k)$ time. Below, we first review the algorithm in [4] and then show our improvement.

4.1 Preliminaries

Without loss of generality, we assume L is the x -axis. Let $S = \{s_1, s_2, \dots, s_k\}$ be the set of servers sorted on L from left to right. Let $Q = \{q_1, q_2, \dots, q_n\}$ be the set of clients sorted on L from left to right. For ease of exposition, we assume no two points in $S \cup Q$ are at the same location.

For any two points p and q on L with p to the left of q , we use $[p, q]$ to denote the interval on L with left endpoint at p and right endpoint at q . An easy observation is that there exists

an optimal solution consisting of k intervals in $\{[p, q] \mid p, q \in S \cup Q\}$, i.e., every interval in the optimal solution starts and ends at a point in $S \cup Q$. For any two points p and q on L , let $d(p, q)$ denote the distance between them.

The servers of S partition the clients of Q into $k + 1$ subsets, defined as follows. For each $i \in \{1, \dots, k - 1\}$, let Q_i be the subset of the clients of Q between s_i and s_{i+1} on L . In addition, we let Q_0 be the subset of the clients of Q to the left of s_1 , and let Q_k be the subset of the clients of Q to the right of s_k . Since both S and Q are already given sorted, we can obtain the subsets Q_0, Q_1, \dots, Q_k in $O(n + k)$ time. In the following, for simplicity of discussion, we assume Q_i is not empty for each $i \in \{0, \dots, k\}$. This implies that the leftmost client q_1 is to the left of the leftmost server s_1 and the rightmost client q_n is to the right of the rightmost server s_k . For each $i \in \{1, \dots, k\}$, let $Q'_i = \{s_i\} \cup Q_i$.

4.2 A dynamic programming algorithm

Consider any Q'_i with $1 \leq i \leq k$. Let q be any point in Q'_i . Consider the *subproblem* at q : Finding i intervals on L such that each interval contains at least one server of $\{s_1, s_2, \dots, s_i\}$, each client to the left of q (including q if $q \neq s_i$) must be covered by at least one interval, and the maximum length of these i intervals is minimized. Define $\alpha(q)$ as the maximum length of the intervals in an optimal solution of the above subproblem at q . Our goal for the interval coverage problem is to solve the subproblem at q_n and compute the value $\alpha(q_n)$.

For any point $q \in S \cup Q$, we use r_q to denote the right neighboring point of q on L in $S \cup Q$ (i.e., the closest point of $S \cup Q$ to q strictly to the right of q). Note that after merging S and Q into one sorted list, we can obtain r_q for each $q \in S \cup Q$ in constant time.

Initially, for each $q \in Q'_1$, $\alpha(q) = d(q_1, q)$ (recall that q_1 is to the left of s_1).

In general, consider any $q \in Q'_i$ for any $2 \leq i \leq k$. It holds that

$$\alpha(q) = \min_{q' \in Q'_{i-1}} \max\{\alpha(q'), d(r_{q'}, q)\}.$$

In words, in order to solve the subproblem at q , we use the $i - 1$ intervals for the subproblem at q' along with an additional interval $[r_{q'}, q]$. To compute $\alpha(q)$, Dumitrescu and Jiang [4] used the following observation: As we consider the points q' of Q'_{i-1} from left to right, $\alpha(q')$ is monotonically increasing and $d(r_{q'}, q)$ is monotonically decreasing. Hence, if $\alpha(q')$ is known for all $q' \in Q'_{i-1}$, $\alpha(q)$ can be computed in $O(\log |Q'_{i-1}|)$ time by binary search.

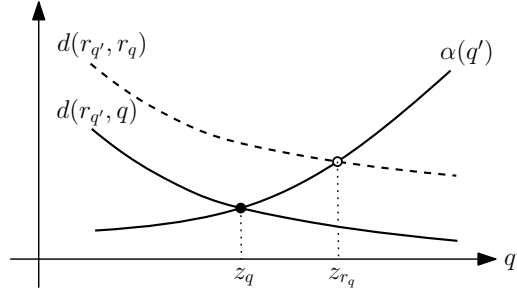
In this way, the value $\alpha(q_n)$ can be computed in $O((n + k) \log(n + k))$ time (more precisely, $O((n + k) \log n)$ time) and an optimal solution can be found correspondingly. Note that the algorithm of Dumitrescu and Jiang [4] does not assume that points of $S \cup Q$ are sorted. But even if they are sorted, their dynamic programming algorithm still takes $O((n + k) \log(n + k))$ time.

4.3 An improved implementation

We give an $O(n + k)$ time implementation for the above dynamic programming scheme. To this end, we find a new monotonicity property in Lemma 15.

Consider any point $q \in Q'_i$ such that r_q is still in Q'_i . For any point $q' \in Q'_{i-1}$, define $f(q') = \max\{\alpha(q'), d(r_{q'}, q)\}$. Hence, $\alpha(q) = \min_{q' \in Q'_{i-1}} f(q')$. Let z_q be the point in Q'_{i-1} such that $\alpha(q) = f(z_q)$ (if there is more than one such point, we let z_q refer to the rightmost one).

► **Lemma 15.** *Either $z_{r_q} = z_q$ or z_{r_q} is strictly to the right of z_q .*



■ **Figure 2** Illustrating the three functions $\alpha(q')$, $d(r_{q'}, q)$, and $d(r_{q'}, r_q)$ for $q' \in Q'_{i-1}$.

Proof. Recall that as we consider the points q' of Q'_{i-1} from left to right, $\alpha(q')$ is monotonically increasing and $d(r_{q'}, q)$ is monotonically decreasing. Intuitively, z_q corresponds to the intersection of the two functions $\alpha(q')$ and $d(r_{q'}, q)$ for $q' \in Q'_{i-1}$ (e.g., see Fig. 2). Similarly, for the point r_q , which is still in Q'_i , z_{r_q} corresponds to the intersection of the two functions $\alpha(q')$ and $d(r_{q'}, r_q)$ for $q' \in Q'_{i-1}$. An observation is that we can obtain the function $d(r_{q'}, r_q)$ by shifting $d(r_{q'}, q)$ upwards by the value $d(q, r_q)$ (e.g., see Fig. 2). This implies that z_{r_q} cannot be strictly to the left of z_q .

The above is an “intuitive” proof. We now provide a more rigorous argument. According to the above discussion, if we consider the points q' of Q'_{i-1} from left to right, $\max\{\alpha(q'), d(r_{q'}, q)\}$ is first monotonically decreasing and then increasing. By the definition of z_q , $\max\{\alpha(q''), d(r_{q''), q)\} \geq \max\{\alpha(z_q), d(z_q, q)\}$ must hold, where $q'' \in Q'_{i-1}$ is the left neighboring point of z_q . Notice that $d(r_{q''), r_q) = d(r_{q''), q) + d(q, r_q)$ and $d(z_q, r_q) = d(z_q, q) + d(q, r_q)$. Therefore, we obtain $\max\{\alpha(q''), d(r_{q''), r_q)\} \geq \max\{\alpha(z_q), d(z_q, r_q)\}$. This further implies that z_{r_q} is either z_q or strictly to the right of z_q . The lemma thus follows. ◀

Lemma 15 essentially says that if we consider all points $q \in Q'_i$ from left to right, then z_q in Q'_{i-1} are also sorted on L from left to right. Due to this monotonicity property on z_q , we can compute z_q and $\alpha(q)$ for all $q \in Q'_i$ in a total of $O(|Q'_{i-1}| + |Q'_i|)$ time by scanning the points of Q'_{i-1} from left to right. More specifically, suppose that we have computed z_q and $\alpha(q)$ for some $q \in Q'_i$. If r_q is still in Q'_i , to compute z_{r_q} and $\alpha(r_q)$, we scan the points of Q'_{i-1} starting from z_q to the right until a point $q' \in Q'_{i-1}$ such that $\max\{\alpha(q''), d(r_{q''), r_q)\} \geq \max\{\alpha(q'), d(r_{q'}, r_q)\} < \max\{\alpha(r_{q'}), d(r_{r_{q'}}, r_q)\}$, where $q'' \in Q'_{i-1}$ is the left neighboring point of z_q . Then, we set $z_{r_q} = q'$ and $\alpha(r_q) = \max\{\alpha(q'), d(r_{q'}, r_q)\}$.

In this way, the value $\alpha(q_n)$ can be computed in $O(n + k)$ time, and an optimal solution can be found correspondingly. Hence, we have the following theorem.

► **Theorem 16.** *If all servers and clients are sorted on the line L , then the facility-location movement problem can be solved in $O(n + k)$ time.*

5 Concluding remarks

In this paper, we studied the points-spreading problem for both a line version and a cycle version. We also considered a related facility-location movement problem. We presented linear-time algorithms for all three problems, which are clearly optimal.

As an application, our algorithm for Theorem 16 can be used to solve the cycle version of the same problem, where all servers and clients are given on a cycle. Dumitrescu and Jiang [4] showed that the cycle version can be solved by solving at most $(n + k)/k$ instances of the

above line version of the problem. More specifically, there must be an adjacent pair of servers such that there are at most n/k clients between them; cutting the cycle between each adjacent pair of the above clients will result in an instance of the line version, with a total of no more than $(n+k)/k$ instances. By using their line-version algorithm of $O((n+k)\log(n+k))$ time, Dumitrescu and Jiang [4] solved the cycle version of the problem in $O(\frac{1}{k}(n+k)^2\log(n+k))$ time. By applying our improved algorithm for the line version (and assuming that the servers and clients are all given sorted on the cycle), the cycle version can be solved in $O(\frac{1}{k}(n+k)^2)$ time. It would be interesting to see whether a better algorithm is possible.

As mentioned in Section 1, the min-sum version of the points-spreading problem for points on a line can be solved in $O(n\log n)$ time [8]. To the best of our knowledge, the cycle version of the problem has not been studied before. An interesting future work would be to see whether the techniques in our paper can be somehow adapted to tackle the problem.

References

- 1 Sergio Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62:49–73, 2007. doi:10.1016/j.jalgor.2004.06.009.
- 2 Barun Chandra and Magnús M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38:438–465, 2001. doi:10.1006/jagm.2000.1145.
- 3 Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5:30:1–30:30, 2009. doi:10.1145/1541885.1541891.
- 4 Adrian Dumitrescu and Minghui Jiang. Constrained k -center and movement to independence. *Discrete Applied Mathematics*, 159:859–865, 2011. doi:10.1016/j.dam.2011.01.008.
- 5 Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory of Computing Systems*, 51:125–142, 2012. doi:10.1007/s00224-011-9331-x.
- 6 Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145:306–316, 2005. doi:10.1016/j.dam.2004.02.018.
- 7 Zachary Friggstad and Mohammad R. Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms*, 7(28:1–28:22), 2011. doi:10.1145/1978782.1978783.
- 8 Mehrdad Ghadiri and Sina Yazdanbod. Minimizing the total movement for movement to independence problem on a line. In *Proceedings of the 28th Canadian Conference on Computational Geometry (CCCG)*, pages 15–20, 2016.
- 9 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984. doi:10.1007/BF02579150.
- 10 Leonid G. Khachiyan. Polynomial algorithm in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980. doi:10.1016/0041-5553(80)90061-0.
- 11 S.S. Ravi, Daniel J. Rosenkrantz, and Giri K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994. doi:10.1287/opre.42.2.299.
- 12 D.W. Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information Processing Letters*, 28:281–286, 1988. doi:10.1016/0020-0190(88)90174-3.