Algorithms for Minimizing the Movements of Spreading Points in Linear Domains

Shimin Li* Haitao Wang†

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 of all points is minimized. 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. In addition, we present a linear-time algorithm for a similar facility-location moving problem, which improves the previous work.

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 of 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 of 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 [8, 9], 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 is minimized. Dumitrescu and Jiang [4] solved this problem 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 points-spreading problem in 2D was proposed by Demaine et al. [3] (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 the 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 proposed by Cabello [1]. Dumitrescu and Jiang [5] gave improvement on the approximation algorithms and also presented algorithms for the problem in high-dimensional spaces. In fact, Fiala et al. [6] studied the dispersion problems on a more general problem settings. Another variation of the dispersion problems 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 [10, 11]. The problem is generally NP-hard (e.g., in 2D) but polynomial time algorithms are available in the one-dimensional space [10, 11]. In addition, Chandra and Halldórsson [2] studied dispersion problems on other problem settings.

The facility-location movement problem was first introduced by Demaine et al. [3] in graphs, which was proved to be NP-hard. A 2-approximation algorithm

*Department of Computer Science, Utah State University, Logan, UT 84322, USA. shiminli@aggiemail.usu.edu

†Department of Computer Science, Utah State University, Logan, UT 84322, USA. haitao.wang@usu.edu

was presented in [3] for this problem in graphs, 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 k as the parameter) were also given in [4].

1.2 Our Approaches

For solving the line version of the points-spreading problem, essentially we first solve a "one-direction" 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 in the solution of the above one-direction 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 $d/2$.

For solving the cycle version of the problem, essentially we also first solve a one-direction case in which points are only allowed to move counterclockwise on C. If d is the maximum movement in the solution of the one-direction 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-direction 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-direction 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 show the correctness.

For solving the facility-location movement problem, Dumitrescu and Jiang [4] presented an $O((m+n)\log(m+n))$ 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.

Due to the space limit, proofs of all lemmas and observations in Section 3 are in the appendix.

2 The Points-Spreading Problem on a Line

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 $P = \{p_1, p_2, \ldots, p_n\}$ are sorted by their x-coordinates from left to right. For each $i \in [1, n]$, 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 two points of P is at least δ and the maximum moving distance, i.e., $\max_{1\le i\le n} |x_i x'_i|$, 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 *configuration* as a specification of the location of each point p_i of P on L. 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. Below, 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 $[1, n]$, define

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

As discussed 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}.

Lemma 1 $d_{opt} \ge \max_{1\le i<j\le n} \frac{w(i,j)}{2}$.

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 $1 \le i \le n$, let x_i^* be the location of p_i in OPT.

Consider any i and j with $1 \le i < j \le n$. Our goal is to prove $d_{opt} \ge w(i, j)/2$. Since the points of P in OPT have the same order as in F_0, for each k with $i < k \le j$, we have $|x_{k-1}^* x_k^*| \ge \delta$ because OPT is a feasible solution. Hence, $|x_i^* x_j^*| = \sum_{k=i+1}^{j} |x_{k-1}^* x_k^*| \ge (j - i) \cdot \delta$.

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

If $|x_i^* x_j^*| - |x_i x_j| > 0$, then the difference of $|x_i^* x_j^*|$ and $|x_i x_j|$ are due to the moving of p_i and p_j. It is not difficult to see that $\max\{|x_i x_i^*|, |x_j x_j^*|\} \ge (|x_i^* x_j^*| - |x_i x_j|)/2$ (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 $d_{opt} \ge \max\{|x_i x_i^*|, |x_j x_j^*|\}$,

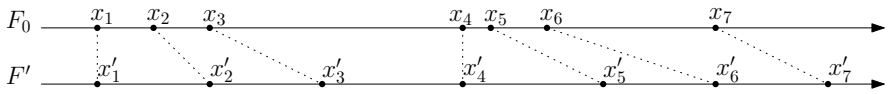

Figure 1: Illustrating our algorithm for computing the configuration F.

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 $d_{opt} \geq w(i,j)/2$. □

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 $w(i,j) = \max_{1 \leq k \leq n} |x_k x'_k|$, 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'' be 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, since $0 \leq x'_k - x_k \leq w(i,j)$, i.e., x'_k is to the right of x_k at most $w(i,j)$, 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, $|x_k x''_k| \leq w(i,j)/2$. □

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

We present a linear-time algorithm for finding a canonical configuration. Comparing with the original problem, now we only need to consider the rightward movements.

Initially, we set $x'_1 = x_1$. Then we consider the points $p_2, p_3, \ldots, p_n$ 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).

Lemma 3 *F' is a canonical configuration.*

Proof. First of all, based on our way of setting x'_i for $i = 1, 2, \ldots, n$, 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 $i \in [1,n]$. 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 $k \in [i+1, j]$, 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. □

Lemmas 2 and 3 together lead to Theorem 4.

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 a *one-direction 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 of all points of P is minimized.

3 The Points-Spreading Problem on a Cycle

In the cycle version, the points of $P = \{p_1, \ldots, p_n\}$ 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 $i \in [1,n]$, we use x_i 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 before, we refer to a *configuration* as a specification of the location of each point of P on C. 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. Below, we assume F_0 is not feasible, and thus $d_{opt} > 0$.

To solve 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 $[1, n]$, define

$$w(i, j) = \big[(n + j - i) \mod n\big] \cdot \delta - |C(x_i, x_j)|.$$

In words, if $i < j$, then $w(i, j) = (j-i)\cdot\delta-|C(x_i, x_j)|$; otherwise, $w(i, j) = (n + j - i) \cdot \delta - |C(x_i, x_j)|$. Since $|C| \geq \delta \cdot n$, it can be verified that $w(i, j) \leq |C|$.

As discussed 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 can prove Lemma 5, which is analogous to Lemma 2.

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

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 $[1, n]$ 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 $w(i, j)/2$.*

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

We present a linear-time algorithm for finding a canonical configuration. Now we only need to consider the counterclockwise movements.

Recall that the points $p_1, p_2, \ldots, p_n$ 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. Define x_1 as the origin with coordinate zero. 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 on the second round, which is to consider all points again from p_1 and use the same strategy to set the new locations of the points. 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 $i \in [2, n]$, suppose we have already moved p_{i-1} to x'_{i-1}, then we define x'_i as follows:

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

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 $x_i \leq |C|$, by Equation (1), $x'_i = (x'_{i-1} + \delta) \mod |C|$, 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.*

By 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 on the second round, as follows.

3.2.2 The second round

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

We first define x''_1. Recall that we proceed on 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

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

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

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

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

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.

Observation 1 *There must be a point p_i with $i \in [2, n]$ such that p_i does not move in the first round of the algorithm (i.e., $x_i = x'_i$).*

Observation 2 *If a point p_i does not move in the second round, then for each point p_j with $j \in [i, n]$, p_j does not move in the second round either.*

Lemma 8 *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., $x_k = x'_k = x''_k$.*

Based on the proof of Lemma 8, we have the following two corollaries.

Corollary 9 *The configuration F_2 is feasible.*

Corollary 10 *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 $1 \leq i \leq n$.*

With the previous observations, Lemma 11 finally shows that F_2 is a canonical configuration.

Lemma 11 *F_2 is a canonical configuration.*

Clearly, both rounds of our algorithm run in $O(n)$ time. Combining Lemmas 6, 7, and 11, we have the following result.

Theorem 12 *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-direction 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 of 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 solve 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. 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 $\{[p, q] \mid p, q \in S \cup P\}$. For any two points p and q on L, let $d(p, q)$ denote the distance between them.

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of servers sorted on L from left to right. Let $Q = \{q_1, q_2, \ldots, q_n\}$ 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.

The servers of S partition the clients of Q into $k + 1$ subsets, defined as follows. For each $i \in [1, k - 1]$, 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 $Q_0, Q_2, \ldots, Q_k$ in $O(n + k)$ time. In the following, for simplicity of discussion, we assume Q_i is not empty for each $i \in [0, k]$. This implies that the rightmost client q_n is to the right of the rightmost server s_k and the leftmost client q_1 is to the left of the leftmost server s_1. For each $i \in [1, k]$, let $Q'_i = \{s_i\} \cup Q_i$.

4.2 A Dynamic Programming Algorithm [4]

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 $\{s_1, s_2, \ldots, s_i\}$, 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 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 $\alpha(q')$ for all $q' \in Q'_{i-1}$ are known, $\alpha(q)$ can be computed in $O(\log |Q'_{i-1}|)$ time by binary search.

In this way, $\alpha(q_n)$ 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.

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 13.

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 $g(q)$ be the point in Q'_{i-1} such that $\alpha(q) = f(g(q))$ (if there is more than one such point, we let $g(q)$ refer to the rightmost one).

Lemma 13 *Either $g(r(q)) = g(q)$ or $g(r(q))$ is strictly to the right of $g(q)$.*

Proof. We only give an "intuitive" 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, $g(q)$ corresponds to the intersection of the two functions $\alpha(q')$ and $d(r(q'), q)$ for $q' \in Q'_{i-1}$ (e.g., see Figure 2). Similarly, for the point $r(q)$, which is still in Q'_i, $g(r(q))$ 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 $g(r(q))$ cannot be strictly to the left of $g(q)$. The lemma thus follows. □

Lemma 13 essentially says that if we consider all points $q \in Q'_i$ from left to right, then $g(q)$ in Q'_{i-1} are also sorted on L from left to right. Due to this monotonicity property on $g(q)$, we can compute $g(q)$ 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 we have computed $g(q)$ and $\alpha(q)$ for some $q \in Q'_i$; then if $r(q)$ is still in Q'_i, we can compute $g(r(q))$ and $\alpha(r(q))$ by scanning the points of Q'_{i-1} starting from $g(q)$ to the right.

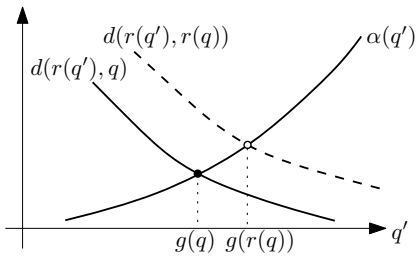

Figure 2: Illustrating the three functions $\alpha(q')$, $d(r(q'), q)$, and $d(r(q'), r(q))$ for $q' \in Q'_{i-1}$.

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 14 *The facility-location movement problem can be solved in $O(n+k)$ time.*

Acknowledgment. The authors would like to thank Minghui Jiang for bringing these problems to them. The research was supported in part by NSF under Grant CCF-1317143

References

[1] S. Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62:49–73, 2007.

[2] B. Chandra and M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38:438–465, 2001.

[3] E. Demaine, M. Hajiaghayi, H. Mahini, A. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3), 2009. Article No. 30.

[4] A. Dumitrescu and M. Jiang. Constrained k-center and movement to independence. *Discrete Applied Mathematics*, 159:859–865, 2011.

[5] A. Dumitrescu and M. Jiang. Dispersion in disks. *Theory of Computing Systems*, 51:125–142, 2012.

[6] J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145:306–316, 2005.

[7] Z. Friggstad and M. Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms*, 7(3), 2011. Article No. 28.

[8] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[9] L. G. Khachiyan. Polynomial algorithm in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

[10] S. Ravi, D. Rosenkrantz, and G. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[11] D. Wang and Y.-S. Kuo. A study on two geometric location problems. *Information Processing Letters*, 28:281–286, 1988.