Rearranging a Sequence of Points onto a Line*

Taehoon Ahn† Jongmin Choi† Chaeyoon Chung† Hee-Kap Ahn‡ Sang Won Bae§
Sang Duk Yoon¶

Abstract

Given a sequence of n weighted points $\langle p_1, p_2, \ldots, p_n \rangle$ in the plane, we consider the problem of finding a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than their weight difference, and the maximum distance between p_i and q_i over all i is minimized. We present efficient algorithms that compute an optimal rearrangement for three variants of the problem under the Euclidean metric. When the line is fully specified or partially specified by only its orientation, our algorithms take near-linear time. When we need to find a target line, onto which the input sequence can be rearranged with the optimal rearrangement cost, we present an $O(n^3 \operatorname{polylog} n)$-time algorithm.

1 Introduction

Consider an object moving in the plane and its trajectory data which can be represented by a sequence of pairs, each consisting of a time stamp and the coordinates of the object at the time. One popular problem concerning such trajectories is to determine whether the object follows a path of a certain shape. The quality of the trajectory with respect to the path can be measured by their similarity, that is, how closely the trajectory follows the path in increasing order of time stamps. Therefore, in a good trajectory, every trajectory point can be translated to a point in the path such that the translation distance is small and two consecutive trajectory points are translated to points close to each other along the path. Formally, we define this problem for linear paths as follows.

Weighted point-to-line rearrangement. Given a sequence of n points $\langle p_1, p_2, \ldots, p_n \rangle$ in the plane and their weights w_i for p_i with $w_1 \leq w_2 \leq \cdots \leq w_n$, find a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than $w_{i+1} - w_i$, and the maximum distance between p_i and q_i over all i is minimized. We call such a rearrangement an *optimal* rearrangement of the point sequence, and the maximum distance of an optimal rearrangement the *optimal* rearrangement cost.

Observe that the constraint on the distance of two consecutive points implies that any two points q_i and q_j $(i \leq j)$ are at distance no more than $w_j - w_i$. See Figure 1 for an illustration of rearranging five points onto a line ℓ.

We consider three variants of the problem: (1) the rearrangement line is given, (2) only the orientation of the rearrangement line is given, or (3) the rearrangement line is not specified at all. For the variants (2) and (3), we need to find a best line, onto which the input sequence can be rearranged with the optimal rearrangement cost, and realize such a rearrangement.

For ease of presentation, we will discuss a special case in which $w_i = i$ for every index i. We first describe our algorithms for this special case and then show how to extend to the general weighted problem without increasing time complexities. The special case is equivalent to the following unweighted problem.

(Unweighted) point-to-line rearrangement. Given a sequence of n points $\langle p_1, p_2, \ldots, p_n \rangle$ in the plane, find a rearrangement of the points, q_i for each p_i, onto a line such that any two consecutive points q_i and q_{i+1} are at distance no more than 1, and the maximum distance between p_i and q_i over all i is minimized.

Again, the constraint on the distance between two consecutive points implies that any two points q_i and q_j $(i \leq j)$ are at distance no more than $j - i$.

*T.Ahn, J.Choi, C.Chung, and H.-K.Ahn were supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)). S.W.Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07042755). S.D.Yoon was supported by "Cooperative Research Program for Agriculture Science & Technology Development (Project No. PJ015269032021)" Rural Development Administration, Republic of Korea.

†Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. {sloth, icothos, chaeyoon17}@postech.ac.kr

‡Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Pohang, Korea. heekap@postech.ac.kr

§Department of Computer Science, Kyonggi University, Suwon, Korea. swbae@kgu.ac.kr

¶Department of Service and Design Engineering, Sungshin Women's University, Seoul, Korea. sangduk.yoon@sungshin.ac.kr

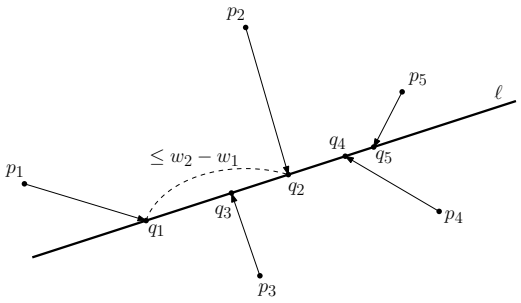

Figure 1: A rearrangement $\langle q_1, ..., q_5\rangle$ of five points $\langle p_1, ..., p_5\rangle$ onto ℓ. Any two points q_i and q_j $(i \leq j)$ are at distance no more than $w_j - w_i$. The cost of this rearrangement is the maximum distance between p_i and q_i over all $i = 1, \ldots, 5$.

Related work. There has been a fair amount of work on rearranging points with respect to certain objectives. One of those problems the most related to ours asks to find the *center line* that minimizes the maximum distance from the input points to the line. This can be solved in $O(n \log n)$ time by computing the center line of a minimum-width slab for the points from the convex hull [11, 12]. Notice that the center line problem implicitly assumes rearrangements of input points obtained by their orthogonal projections onto a line, and hence the center line problem is equivalent to our rearrangement problem where the proximity constraint of consecutive points is relaxed.

Similarly, a circle that aggregates a point set can be found from the minimum-width annulus for the points [9]. There is a deterministic $O(n^{8/5+\varepsilon})$-time algorithm [2] and an expected $O(n^{3/2+\varepsilon})$-time algorithm [1] for computing the minimum-width annulus for n points in the plane. When the radius r of the aggregating circle is fixed, there is an $O(n \log n)$-time algorithm [8, 10] that finds the minimum-width annulus with median radius r.

For a sequence of n points in the increasing order of x-coordinates in the plane, there has been a series of work to find an x-monotone curve with minimum error under various settings on the curve. When the curve is a polyline, the segmented least squares algorithm finds the polyline that minimizes a combination of the total squared errors and the number of segments in the polyline in $O(n^2)$ time [4].

When the weights of the input points are the same, we have $w_{i+1} - w_i = 0$ for all i, and any rearrangement $\langle q_1, \ldots, q_n\rangle$ has the same point for all q_i's. Thus, the optimal rearrangement is achieved by the point $q \in \ell$ such that the smallest disk centered at q and enclosing the input points has minimum radius among all enclosing disks centered at points of ℓ. There is an $O(n)$-time algorithm [14] for finding the center, and an $O(n \log n)$-time algorithm for k centers restricted to a line [18].

Our results. We present efficient algorithms for finding an optimal rearrangement among the rearrangements of the sequence S of n points onto a line under the Euclidean metric. For the case that the rearrangement line ℓ is given, we observe that the cost of the optimal rearrangement of S onto ℓ is determined by at most two points, and present a simple $O(n^2)$-time algorithm. Then we improve the running time to near-linear by applying several optimization techniques. We present an expected $O(n)$-time algorithm using randomized optimization [6] and a deterministic $O(n \log\log n)$-time algorithm using parametric search [7] with some additional preprocessing.

For the case that only the orientation of the rearrangement line is given, we compute an optimal rearrangement line ℓ among all lines of the orientation and an optimal rearrangement onto ℓ using a set of $O(n)$ convex functions, each representing the optimal rearrangement cost of a contiguous subsequence of S. The upper envelope of those functions coincides with the function of the optimal rearrangement cost of S. By applying convex programming, our algorithm computes an optimal rearrangement in $O(n \log n)$ time. For the case that the rearrangement line is not specified at all, we present an $O(n^3 \operatorname{polylog} n)$-time deterministic algorithm that finds an optimal rearrangement line ℓ and an optimal rearrangement onto ℓ. Due to the page limit, the detailed algorithms for the case that the rearrangement line is not specified is given in Appendix.

2 Preliminaries

Let $S = \langle p_1, \ldots, p_n\rangle$ denote the input sequence of points, and $w_1, \ldots, w_n$ be their weights with $w_1 \leq \cdots \leq w_n$. Throughout the paper, we mainly discuss the unweighted problem, so we assume $w_i = i$ for each $i = 1, \ldots, n$, unless stated otherwise. For any $1 \leq i \leq j \leq n$, we denote by S_{ij} the contiguous subsequence of S from p_i to p_j, that is, $S_{ij} = \langle p_i, \ldots, p_j\rangle$. Let $\|\cdot\|$ denote the Euclidean norm on the plane so that we use $\|p - q\|$ to denote the distance between two points p and q. A sequence $\langle q_i, \ldots, q_j\rangle$ of points is a *rearrangement* of S_{ij} onto a line ℓ if q_k lies on ℓ and $\|q_{k'} - q_k\| \leq w_{k'} - w_k$ for every k and k' with $i \leq k \leq k' \leq j$. Its *cost* is defined to be $\max_{i \leq k \leq j} \|q_k - p_k\|$.

An *optimal* rearrangement of S onto a line ℓ is a rearrangement with the minimum cost among all rearrangements of S onto ℓ. An optimal rearrangement of S onto a set of lines is a rearrangement with the minimum cost among all optimal rearrangement of S over the lines in the set. We use $\delta^*(\ell)$ to denote the cost of an optimal rearrangement of S onto ℓ. We may simply write δ^* if it is understood from the context.

For a real number $r \geq 0$, we denote by $I(r)$ the segment of length $2r$ joining two points $(-r, 0)$ and $(r, 0)$

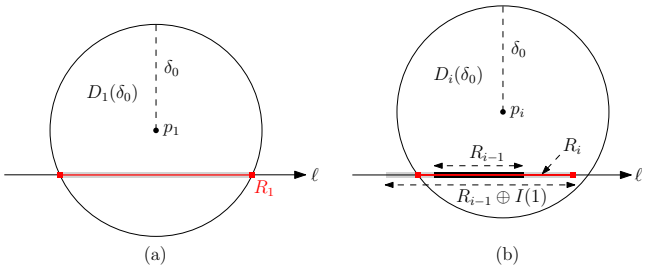

Figure 2: (a) R_1 is defined as the intersection of $D_1(\delta_0)$ and ℓ. (b) R_i is defined as the intersection of $D_i(\delta_0)$ and $R_{i-1} \oplus I(1)$.

on the x-axis. We will often consider the Minkowski sum of a compact set X in the plane and the segment $I(r)$, denoted by $X \oplus I(r)$.

3 Rearrangement onto a Fixed Line

In this section, we consider the problem when the rearrangement line is given as an input, so we are given a sequence $S = \langle p_1, \ldots, p_n \rangle$ and a line ℓ, and want to find an optimal rearrangement of S onto ℓ. Without loss of generality, assume that ℓ is the x-axis. For any real number r, we abuse the notation so that r also denotes the point $(r, 0)$ on the x-axis ℓ if there is no confusion from the context.

We first present an $O(n)$-time decision algorithm determining for a given real value $\delta_0 \geq 0$, whether there exists a rearrangement of S with cost at most δ_0. We compute a *feasible range* R_i for each point p_i of S, representing the range in the x-axis in which q_i of a rearrangement $\langle q_1, \ldots, q_i \rangle$ of S_{1i} onto ℓ with cost at most δ_0 can be placed as follows: $R_1 = \ell \cap D_1(\delta_0)$ and $R_i = (R_{i-1} \oplus I(1)) \cap D_i(\delta_0)$ for $1 < i \leq n$, where $D_i(\delta_0) := \{q \mid \|q - p_i\| \leq \delta_0\}$ denotes the disk with center p_i and radius δ_0. See Figure 2 for an illustration of the ranges.

Lemma 1 *There is a rearrangement of S_{1i} with cost at most δ_0 and p_i rearranged to q_i if and only if $q_i \in R_i$.*

Proof. We first prove the if part by induction. If $i = 1$, it is trivial. For any $i > 1$, we pick a point $q_i \in R_i$. Then we have $(q_i \oplus I(1)) \cap R_{i-1} \neq \emptyset$ as $q_i \in R_{i-1} \oplus I(1)$. Pick a point q_{i-1} in $(q_i \oplus I(1)) \cap R_{i-1}$. Then, by the induction hypothesis, there is a rearrangement $\langle q_1, \ldots, q_{i-1} \rangle$ of $S_{1(i-1)}$ with cost at most δ_0. Since $\|q_i - q_{i-1}\| \leq 1$, $\langle q_1, \ldots, q_{i-1}, q_i \rangle$ is a rearrangement of S_{1i} with cost at most δ_0.

We now prove the only if part by induction. It is trivial for $i = 1$. For any $i > 1$, let $\langle q_1, \ldots, q_i \rangle$ be a rearrangement of S_{1i} with cost at most δ_0. We have $q_i \in R_{i-1} \oplus I(1)$ because $\|q_{i-1} - q_i\| \leq 1$, and $q_{i-1} \in R_{i-1}$ by the induction hypothesis. We also have $q_i \in \ell \cap D_i(\delta_0)$ because the cost of the rearrangement is at most δ_0. Therefore, $q_i \in R_i$. □

By Lemma 1, the decision problem can be answered by checking whether $R_n \neq \emptyset$ (`yes`) or $R_n = \emptyset$ (`no`). Since we can compute R_1 in $O(1)$ time, and R_i in $O(1)$ time once we have R_{i-1}, we can compute R_n in $O(n)$ time. If $R_n \neq \emptyset$, we can compute a rearrangement $\langle q_1, \ldots, q_n \rangle$ of S in $O(n)$ time, by choosing q_n from R_n, and then choosing q_i from $(q_{i+1} \oplus I(1)) \cap R_i$ repeatedly for i from $n - 1$ to 1.

Lemma 2 *Given a point sequence S of n points, a line ℓ, and a real value δ_0, we can decide whether there exists a rearrangement of S onto ℓ with cost at most δ_0 in $O(n)$ time. If such rearrangement exists, we can compute a rearrangement of S with cost at most δ_0 in $O(n)$ time.*

We present some characterizations of an optimal rearrangement of S. We first show that there are at most two points of S which determine the cost $\delta^* = \delta^*(\ell)$ of an optimal rearrangement in the following lemma.

Lemma 3 *There exists an optimal rearrangement $\langle q_1, \ldots, q_n \rangle$ of S onto ℓ satisfying one of the followings.*

(1) There is a point p_j in S such that $\|p_j - q_j\| = \delta^*$ and q_j is the orthogonal projection of p_j onto ℓ.
(2) There are two points p_i and p_j $(i < j)$ in S such that $\|p_i - q_i\| = \|p_j - q_j\| = \delta^*$, $\|q_i - q_j\| = j - i$, and both q_i and q_j lie in between the orthogonal projections of p_i and p_j onto ℓ.

Proof. Among the feasible ranges of the points of S for δ^*, there must be a feasible range that is a single point. Otherwise, there is a real value $\epsilon > 0$ such that $R_n \neq \emptyset$ with cost $(\delta^* - \epsilon)$, which contradicts the optimality of δ^*.

If a feasible range R_j is a single point, then ℓ is tangent to $D_j(\delta^*)$, or $D_j(\delta^*)$ intersects $R_{j-1} \oplus I(1)$ only at an endpoint. The former case implies that q_j is the orthogonal projection of p_j onto ℓ with $\|p_j - q_j\| = \delta^*$, and thus we have case (1). The latter case implies that the common intersection of $D_i(\delta^*) \oplus I(j - i)$, ℓ, and $D_j(\delta^*)$ is just a single point for some i with $1 \leq i < j$. Then, $\|p_i - q_i\| = \|p_j - q_j\| = \delta^*$, $\|q_j - q_i\| = j - i$, and both q_i and q_j lie in between the orthogonal projections of p_i and p_j onto ℓ. Thus, we have case (2). □

For an optimal rearrangement, we call the points of S that satisfy cases (1) or (2) of Lemma 3 the *determinators* of the rearrangement. We now define a value δ_{ij} for every two indices $1 \leq i \leq j \leq n$. Let q_i and q_j be the points on ℓ minimizing $\max\{\|p_i - q_i\|, \|p_j - q_j\|\}$ with $\|q_i - q_j\| \leq j - i$. Then $\delta_{ij} = \max\{\|p_i - q_i\|, \|p_j - q_j\|\}$. (Figure 3). Note that δ_{jj} is the length of the orthogonal projection of p_j to ℓ. Then δ_{ij} denotes the minimum cost required by two points p_i and p_j of S such that there is a rearrangement of S.

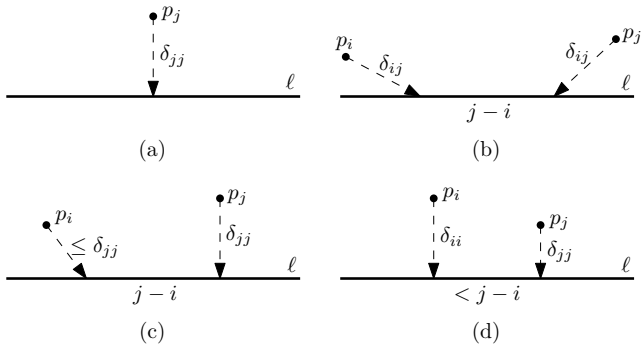

Figure 3: (a) δ_{jj} is the length of the projection from p_j to ℓ. (b) When the difference of x-coordinates between two points p_i and p_j $(i < j)$ is bigger than $j - i$, δ_{ij} is defined by two points in between the orthogonal projections of p_i and p_j with distance $j - i$. (c) One of the points that define δ_{ij} can be the orthogonal projection of p_j, so that $\delta_{ij} = \delta_{jj}$. (d) When the difference of x-coordinates between two points p_i and p_j is smaller than or equal to $j - i$, $\delta_{ij} = \max\{\delta_{ii}, \delta_{jj}\}$.

Lemma 4 $\delta^* = \max_{i,j} \delta_{ij}$.

Proof. By Lemma 3, there is an optimal rearrangement with determinators. If the optimal rearrangement belongs to case (1) of Lemma 3, then $\delta^* = \delta_{jj}$ for the determinator p_j (Figure 3(a)). If it belongs to case (2) of Lemma 3, then $\delta^* = \delta_{ij}$ for the determinators p_i and p_j. Therefore, $\delta^* \leq \max_{i,j} \delta_{ij}$ holds (Figure 3(b)).

If $\delta^* < \delta_{ii}$, there is no point $q \in \ell$ such that $\|p_i - q\| \leq \delta^*$, which is a contradiction. Assume that there are two indices i, j $(i < j)$ such that $\delta^* < \delta_{ij}$. There is an optimal rearrangement $Q^* = \langle q_1^*, \ldots, q_n^* \rangle$ with cost δ^*. However, $\|q_i^* - q_j^*\| > j - i$ holds by the assumption, which contradicts that Q^* is a rearrangement. Therefore, $\max_{i,j} \delta_{ij} \leq \delta^*$ holds. □

3.1 Randomized algorithm

This problem can be solved in $O(n)$ expected time using the randomized optimization technique by Chan [6] as follows. We consider the weighted version of the problem in which a sequence $S = \langle p_1, \ldots, p_n \rangle$ of n weighted points is given, the weight of p_i denoted by w_i, satisfying $w_i \leq w_j$ for every pair of indices i, j with $i < j$. The objective is to find a rearrangement $Q = \langle q_1, \ldots, q_n \rangle$ of S onto ℓ such that $\|q_i - q_j\| \leq w_j - w_i$ for every pair of indices i, j with $i \leq j$, and the rearrangement cost $\max_i \|p_i - q_i\|$ is minimized.

Observe that Lemmas 2, 3, and 4 also hold for this weighted version, by replacing the definition of R_i with $R_i = (R_{i-1} \oplus I(w_i - w_{i-1})) \cap D_i(\delta)$ (R_1 remains the same) and by replacing the condition $\|q_i - q_j\| = j - i$ in case (2) of Lemma 3 with $\|q_i - q_j\| = w_j - w_i$. Thus, we have the following corollary.

Corollary 5 *For a sequence S of n weighted points, a line ℓ, and a real value δ_0, we can decide whether there exists a rearrangement of S onto ℓ with cost at most δ_0 in $O(n)$ time.*

Let S_1, S_2 and S_3 be subsequences of S with length at most $\lceil n/3 \rceil$ and $S_1; S_2; S_3 = S$ where $A; B$ is the concatenation of two sequences A and B such that the elements of B comes after the last element of A in the concatenation. Then $\delta(S) = \max\{\delta(S_1; S_2), \delta(S_1; S_3), \delta(S_2; S_3)\}$ by Lemma 3, where $\delta(A)$ denotes the optimal rearrangement cost of a sequence A. Therefore, by applying the randomized optimization technique by Chan, we obtain a randomized algorithm to compute δ^*, which takes the time linear to the running time of the decision algorithm, $O(n)$. By Lemma 2, we can compute a rearrangement with cost δ^* using $O(n)$ additional time.

Theorem 6 *Given a sequence S of n weighted points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in expected $O(n)$ time.*

3.2 Deterministic algorithm

By Lemma 4, we get an $O(n^2)$-time deterministic algorithm to compute δ^* that computes δ_{ij} for every pair of indices i, j with $i \leq j$ and returns the maximum value among them. We present a more efficient deterministic algorithm for the problem. We first extend the definition of R_i to define a function representing the range that q_i can be placed with respect to the position of q_1 and the cost δ. We present sub-linear time sequential and parallel decision algorithms using those functions after preprocessing. By applying parametric search, we obtain an $O(n \log \log n)$-time algorithm to compute the optimal rearrangement cost δ^*.

Recall that R_i denotes the feasible range for p_i of S with a fixed cost. For the deterministic algorithm, we consider R_i as a function of the cost variable δ and a real value r, and thus we use $R_i(\delta, r)$ to denote the function. For a fixed cost δ_0 and a fixed value r_0, $R_i(\delta_0, r_0)$ represents the range in ℓ on which q_i of a rearrangement $\langle q_1 = r_0, \ldots, q_i \rangle$ of S_{1i} with cost at most δ_0 can be placed. Our algorithm takes S as input and computes $R_n(\delta, r)$ in the rearrangements of S for all cost values δ and real values r.

Characterization of $R_n(\delta, r)$ for a fixed cost δ_0. To characterize $R_i(\delta, r)$, we set δ to a fixed value δ_0, and use $R_i(r)$ to denote $R_i(\delta_0, r)$. Observe that $R_i(r)$ is an interval on ℓ and its two boundary points can be described by functions B_i and T_i defined on r such that $R_i(r) = [B_i(r), T_i(r)]$. In case that $R_i(r) = \emptyset$, B_i and

T_i are not defined for r. We use dom_i to denote the range of r for which $R_i(r) \neq \emptyset$, and thus B_i and T_i are defined for $r \in \mathsf{dom}_i$.

Observe that $R_i(r)$ can be defined inductively as R_i in the beginning of Section 3. We have $R_1(r) = [r, r]$ which is defined for $r \in \mathsf{dom}_1 = \ell \cap D_1(\delta_0)$, and $R_i(r) = (R_{i-1}(r) \oplus I(1)) \cap D_i(\delta_0)$ for $1 < i \leq n$.

Lemma 7 *There is a rearrangement $\langle q_1 = r, \ldots, q_n \rangle$ of S onto ℓ with cost at most δ_0 if and only if $q_n \in R_n(r)$.*

Proof. If S consists of one point, the lemma holds by the definition of $R_n(r)$. For S consisting of more than one point, we prove the lemma by induction.

Assume $q_n \in R_n(r)$. Then $R_{n-1}(r) \neq \emptyset$, and this implies $r \in \mathsf{dom}_{n-1}$. Also, by the definition of $R_n(r)$, $\|p_n - q_n\| \leq \delta_0$ and there exists a point $q' \in R_{n-1}(r)$ with $\|q_n - q'\| \leq 1$. By the induction hypothesis, we get a rearrangement $\langle q_1 = r, \ldots, q_{n-1} \rangle$ of $S_{1(n-1)}$ with cost at most δ_0. Then we obtain the rearrangement $\langle q_1 = r, \ldots, q_{n-1}, q_n \rangle$ of S with cost at most δ_0.

Let $Q = \langle q_1 = r, \ldots, q_{n-1}, q_n \rangle$ be a rearrangement of S onto ℓ with cost at most δ_0. By the induction hypothesis, we have $r \in \mathsf{dom}_{n-1}$ and $q_{n-1} \in R_{n-1}(r)$. Since we have $q_n \in (q_{n-1} \oplus I(1))$ and $q_n \in D_n(\delta_0)$, $q_n \in R_n(r)$ holds. □

Let b_i and t_i be the two boundary points of $\ell \cap D_i(\delta_0)$ with $b_i \leq t_i$. If $\ell \cap D_i(\delta_0) = \emptyset$, b_i and t_i are not defined. If ℓ is tangent to $D_i(\delta_0)$, $b_i = t_i$. We can check in $O(n)$ time whether b_i and t_i are defined for every i with $1 \leq i \leq n$. Since there is a rearrangement with cost δ_0 only if b_i and t_i are defined for every i, we assume that they are defined for every i in the remainder of this section. In the following lemmas, we show that $B_n(r)$, $T_n(r)$, and dom_n can be expressed using b_i's and t_i's.

Lemma 8 *For $r \in \mathsf{dom}_n$, $B_n(r) = \max_{1 \leq i \leq n}\{r - n + 1, b_i + i - n\}$ and $T_n(r) = \min_{1 \leq i \leq n}\{r + n - 1, t_i - i + n\}$.*

Proof. We prove the claim by induction. When $n = 1$, $[B_1(r), T_1(r)] = [r, r] = [r - 1 + 1, r + 1 - 1]$. Since $b_1 \leq r \leq t_1$ for any $r \in \mathsf{dom}_1$, the statement holds. For an index $j > 1$, $\mathsf{dom}_j \subseteq \mathsf{dom}_{j-1}$ holds since $R_j(r) \neq \emptyset$ only if $R_{j-1}(r) \neq \emptyset$. Therefore, for $r \in \mathsf{dom}_j$, $r \in \mathsf{dom}_{j-1}$. Then $[B_j(r), T_j(r)] = [B_{j-1}(r) - 1, T_{j-1}(r) + 1] \cap [b_j, t_j]$. Using the induction hypothesis, we can show that $B_j(r) = \max\{B_{j-1}(r) - 1, b_j\} = \max_{1 \leq i \leq j}\{r - j + 1, b_i + i - j\}$ holds. We can show the claim for $T_j(r)$ similarly. □

Lemma 9 *If $\mathsf{dom}_n \neq \emptyset$, $\mathsf{dom}_n = \bigcap_{1 \leq i \leq n}[b_i - i + 1, t_i + i - 1]$.*

Proof. Let R_j^{rev} denote the range in ℓ on which q_j of a rearrangement $\langle q_j, \ldots, q_n \rangle$ of S_{jn} with cost at most δ_0 can be placed. By Lemma 7, $r \in \mathsf{dom}_n$ if and only if there is a rearrangement $\langle q_1 = r, \ldots, q_n \rangle$ of S onto ℓ with cost at most δ_0. Therefore, $R_1^{\mathsf{rev}} = \mathsf{dom}_n$. We show that $R_j^{\mathsf{rev}} = \bigcap_{j \leq i \leq n}[b_i - i + j, t_i + i - j]$ by induction on j from n to 1.

As the base case, $R_n^{\mathsf{rev}} = \ell \cap D_n(\delta_0) = [b_n, t_n]$. For any $j < n$, let $R_j^{\mathsf{rev}} \neq \emptyset$. As $r \in R_j^{\mathsf{rev}}$ if and only if $r \in (R_{j+1}^{\mathsf{rev}} \oplus I(1))$ and $r \in D_j(\delta_0)$, $R_{j+1}^{\mathsf{rev}} \neq \emptyset$. Then by the induction hypothesis, we have the following equation.

$$R_{j+1}^{\mathsf{rev}} = \bigcap_{j+1 \leq i \leq n} [b_i - i + (j+1), t_i + i - (j+1)]$$

Since $R_j^{\mathsf{rev}} = (R_{j+1}^{\mathsf{rev}} \oplus I(1)) \cap [b_j, t_j]$, the claim holds. Therefore, we conclude $\mathsf{dom}_n = R_1^{\mathsf{rev}} = \bigcap_{1 \leq i \leq n}[b_i - i + 1, t_i + i - 1]$ if $\mathsf{dom}_n = R_1^{\mathsf{rev}} \neq \emptyset$. □

Observe that $\mathsf{dom}_n = \emptyset$ if $R_j^{\mathsf{rev}} = \emptyset$ for some j, even when $\bigcap_{1 \leq i \leq n}[b_i - i + 1, t_i + i - 1] \neq \emptyset$. Therefore, we have to check whether $\mathsf{dom}_n = \emptyset$. By Lemma 8, each of B_n and T_n consists of at most two segments as $\max_{1 \leq i \leq n}\{b_i - n + i\}$ and $\min_{1 \leq i \leq n}\{t_i + n - i\}$ remain unchanged for different r values. See Figure 4.

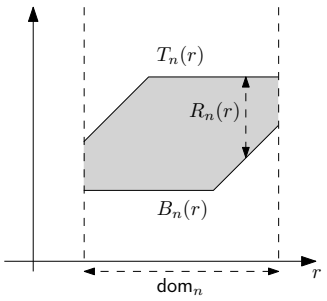

Figure 4: $R_n(r) = [B_n(r), T_n(r)]$, where B_n and T_n defined in dom_n have constant complexity.

Observation 1 *$B_n(r)$ consists of at most two segments, one with slope 0 followed by one with slope 1. $T_n(r)$ consists of at most two segments one with slope 1 followed by one with slope 0.*

Observe that $R_n(r) = [B_n(r), T_n(r)]$ is determined by at most four points of S with indices (a) $\arg\max_i\{b_i + i\}$ and (b) $\arg\min_i\{t_i - i\}$ from Lemma 8, and (c) $\arg\max_i\{b_i - i\}$ and (d) $\arg\min_i\{t_i + i\}$ from Lemma 9. If $\mathsf{dom}_n \neq \emptyset$, $\mathsf{dom}_n = [b_{(\mathrm{c})} - (\mathrm{c}) + 1, t_{(\mathrm{d})} + (\mathrm{d}) - 1]$. For a real value $r \in \mathsf{dom}_n$, $B_n(r) = \max\{r - n + 1, b_{(\mathrm{a})} + (\mathrm{a}) - n\}$ and $T_n(r) = \max\{r + n - 1, t_{(\mathrm{b})} - (\mathrm{b}) + n\}$. We call the set of those four points the *combinatorial structure* of $R_n(r)$ for δ_0.

Computing $R_n(\delta, r) = [B_n(\delta, r), T_n(\delta, r)]$. Let $B_{ij}(r)$ and $T_{ij}(r)$ denote the two boundary points of the feasible range $R_{ij}(r) = [B_{ij}(r), T_{ij}(r)]$ of p_j with respect to the subsequence S_{ij} with p_i rearranged to r. We can compute $B_n(r)$ and $T_n(r)$ using the feasible

ranges $[B_{1k}(r), T_{1k}(r)]$ and $[B_{kn}(r), T_{kn}(r)]$ of two subsequences S_{1k} and S_{kn} of S for any k with $1 < k < n$ in $O(1)$ time.

Lemma 10 *We can compute $B_n(r)$ and $T_n(r)$ of S in $O(1)$ time once we have $B_{1k}(r)$, $T_{1k}(r)$, $B_{kn}(r)$, and $T_{kn}(r)$ for any k with $1 < k < n$.*

Proof. We first check whether $\mathsf{dom}_n = \emptyset$. Let $\mathsf{dom}_{1k} = [r_1, r_2]$, which is the domain of functions $B_{1k}(r)$ and $T_{1k}(r)$. Note that $B_{1k}(r)$ and $T_{1k}(r)$ are monotonically increasing functions by Observation 1. Then the maximal range that q_k of a rearrangement $\langle q_1, \ldots, q_k \rangle$ of S_{1k} with cost at most δ_0 can be placed is $[B_{1k}(r_1), T_{1k}(r_2)]$. There is a rearrangement of S if and only if $[B_{1k}(r_1), T_{1k}(r_2)] \cap \mathsf{dom}_{kn} \neq \emptyset$, where dom_{kn} is the domain of $B_{kn}(r)$ and $T_{kn}(r)$. We can check whether $\mathsf{dom}_n = [B_{1k}(r_1), T_{1k}(r_2)] \cap \mathsf{dom}_{kn} = \emptyset$ in $O(1)$ time. If $\mathsf{dom}_n \neq \emptyset$, we can find the combinatorial structure of $R_n(r)$ from the combinatorial structures of $R_{1k}(r)$ and $R_{kn}(r)$ by $O(1)$ comparisions. After finding the combinatorial structure, we can compute $B_n(r)$ and $T_n(r)$ of S in $O(1)$ time. □

Now we treat δ also as a variable of the feasible range R_i and its boundary points B_i and T_i so that $q_i \in R_i(\delta, r) = [B_i(\delta, r), T_i(\delta, r)]$ holds if and only if there exists a rearrangement $\langle r, \ldots, q_i \rangle$ of S_{1i} with cost at most δ. We also use δ as a variable of two boundary points $b_i(\delta)$ and $t_i(\delta)$ of $\ell \cap D_i(\delta)$ with $b_i(\delta) \leq t_i(\delta)$. We compute $R_n(\delta, r)$ by storing all different combinatorial structures over δ in increasing order of δ.

Lemma 11 *The combinatorial structure of $R_n(\delta, r)$ changes $O(n)$ times over δ.*

Proof. We prove that $\arg\max_i \{b_i(\delta) + i\}$ changes at most $O(n)$ times for δ increasing from 0 to ∞. For any two indices i and j, $b_i(\delta) + i = b_j(\delta) + j$ holds for at most one δ value. Therefore, if $\arg\max_i \{b_i(\delta) + i\}$ changes from j to j' at δ', $j = \arg\max_i \{b_i(\delta) + i\}$ does not hold for $\delta > \delta'$. This implies that each index becomes $\arg\max_i \{b_i(\delta) + i\}$ for at most one interval, which bounds the number of changes to $O(n)$ in total. We can bound the numbers of changes of $\arg\min_i \{t_i(\delta) - i\}$, $\arg\max_i \{b_i(\delta) - i\}$, and $\arg\min_i \{t_i(\delta) + i\}$ in the same way. □

Let $B_{ij}(\delta, r)$ and $T_{ij}(\delta, r)$ denote the two boundary points of the feasible range $R_{ij}(\delta, r) = [B_{ij}(\delta, r), T_{ij}(\delta, r)]$ of p_j with respect to S_{ij} with p_i rearranged to r. We can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ using the feasible ranges $[B_{1k}(\delta, r), T_{1k}(\delta, r)]$ and $[B_{kn}(\delta, r), T_{kn}(\delta, r)]$ of two subsequences S_{1k} and S_{kn} of S for any k with $1 < k < n$ in $O(n)$ time.

Lemma 12 *We can compute $R_n(\delta, r)$ of S in $O(n)$ time once we have $R_{1k}(\delta, r)$, and $R_{kn}(\delta, r)$ for any k with $1 < k < n$.*

Proof. The functions $R_{1k}(\delta, r)$ and $R_{kn}(\delta, r)$ consist of $O(k)$ and $O(n - k)$ combinatorial structures, respectively, by Lemma 11. As they are stored in the increasing order of δ, we simply merge the functions to compute $R_n(\delta, r)$ in increasing order of δ. For each range where the combinatorial structures of $R_{1k}(\delta, r)$ and $R_{kn}(\delta, r)$ remain the same, we can compute $R_n(\delta, r)$ in $O(1)$ time by Lemma 10. As there are $O(n)$ such ranges, the algorithm takes $O(n)$ time in total. □

By Lemma 12, we can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ of S in $O(n \log n)$ time applying divide and conquer.

Lemma 13 *We can compute $B_n(\delta, r)$ and $T_n(\delta, r)$ in $O(n \log n)$ time.*

Optimization algorithm using parametric search. By Lemma 13, we have an $O(n \log n)$-time algorithm for computing an optimal rearrangement as follows: Compute $R_n(\delta, r) = [B_n(\delta, r), T_n(\delta, r)]$ in $O(n \log n)$ time, and find δ^* which is the minimum value of δ satisfying $\mathsf{dom}_n \neq \emptyset$. Using $O(n)$ additional time, the algorithm computes an optimal rearrangement by Lemma 2. Here, we present an algorithm using Cole's parametric search [7] to further improve the running time to $O(n \log \log n)$ time.

Before applying parametric search, our algorithm preprocesses the feasible ranges of subsequences. The algorithm subdivides S into $\lceil n/t \rceil$ subsequences, each consisting of at most $t + 1$ points for some parameter t, which will be chosen later. Every two consecutive subsequences share a point and S is entirely covered by the subsequences. For each subsequence, the algorithm computes the feasible range over all δ values. This procedure takes $O(t \log t)$ time for each subsequence, and thus it takes $O(n \log t)$ time in total.

After the preprocessing, the algorithm determines (sequentially) whether $\delta \geq \delta^*$ as follows. It finds the combinatorial structure of the feasible range of each subsequence with respect to δ using binary search. This takes $O(\log t)$ time for each subsequence, and thus it takes $O((n \log t)/t)$ time in total. Then the algorithm merges the feasible ranges in the sequential order to determine whether $\mathsf{dom}_n = \emptyset$ for δ. This takes $O(1)$ time for merging two feasible ranges in the order, and thus it takes $O(n/t)$ time in total, which is dominated by $O((n \log t)/t)$ time.

With the preprocessing, we present a parallel decision algorithm using $O(n/t)$ processors. The algorithm finds the combinatorial structure of the feasible range with respect to δ by assigning a processor for each subsequence. Then for the remaining steps, the algorithm merges two consecutive feasible ranges using a processor. After $O(\log n)$ merge steps, the algorithm computes $R_n(\delta, r)$. This procedure takes $O(1)$ time for each of $O(\log n)$ steps, after finding the combinatorial structure

of feasible range for each subsequence in $O(\log t)$ time. Thus, it takes $O(\log n)$ time in total. As each process of the parallel algorithm depends on at most two other processes, we can apply Cole's parametric search [7].

By applying parametric search, the algorithm can compute the optimal rearrangement cost δ^* in $O(PT_P + T_S(T_P + \log P)) = O((n \log n \log t)/t)$ time after $O(n \log t)$-time preprocessing, where $P = O(n/t)$ is the number of processors needed for parallel decision, T_P is the running time of our parallel decision algorithm, and T_S is the running time of our sequential decision algorithm. Setting $t = \log n$, we obtain an $O(n \log \log n)$-time algorithm. After finding δ^* in $O(n \log \log n)$ time, we can find an optimal rearrangement of S using $O(n)$ additional time by Lemma 2.

Theorem 14 *For a sequence S of n points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in $O(n \log \log n)$ time.*

3.3 Weighted version

In this section, we show that the deterministic algorithm of Section 3.2 can be extended to the weighted version without increasing the time complexity. Observe that Lemma 7 also holds for the weighted version, by replacing the definition of $R_i(r) = [B_i(r), T_i(r)]$ for fixed δ_0 with $R_i(r) = (R_{i-1}(r) \oplus I(w_i - w_{i-1})) \cap D_i(\delta_0)$ ($R_1(r)$ remains the same). Then we can show that $R_i(r)$ is determined by at most four points of S with indices (a) $\arg\max_i\{b_i + w_i\}$, (b) $\arg\min_i\{t_i - w_i\}$, (c) $\arg\max_i\{b_i - w_i\}$, and (d) $\arg\min_i\{t_i + w_i\}$ as shown in Lemmas 8 and 9. Observe that the points determining $R_i(\delta, r)$ change $O(n)$ times over δ as in Lemma 11, we obtain $O(n \log \log n)$-time algorithm using parametric search.

Theorem 15 *For a sequence S of n weighted points and a line ℓ, we can compute an optimal rearrangement of S onto ℓ in $O(n \log \log n)$ time.*

4 Rearrangement onto a Line with Fixed Orientation

Recall that $\delta^*(\ell)$ denotes the cost of an optimal rearrangement of S onto a line ℓ. Given a sequence $S = \langle p_1, \ldots, p_n \rangle$ of n points and an orientation $\vec{c}$, we find a line ℓ such that $\delta^*(\ell)$ is minimum among all lines parallel to $\vec{c}$ in the plane, and compute an optimal rearrangement of S onto ℓ.

Without loss of generality, we assume that the orientation is horizontal, and thus our target line is horizontal. For a real value h, we use $\ell(h)$ to denote a horizontal line with $y = h$.

We compute an optimal rearrangement using a function that represents the optimal rearrangement cost of S onto all horizontal lines. In doing so, we first compute a convex function for a contiguous subsequence of S that partially describes the optimal rearrangement cost of the subsequence. We do this for $O(n)$ contiguous subsequences of S so that the upper envelope of those functions coincides with the function of the optimal rearrangement cost of S. By applying convex programming on the upper envelopes of disjoint subsets of functions, we can find a horizontal line ℓ such that $\delta^*(\ell)$ is minimum among all horizontal lines, and get the optimal rearrangement cost of S onto ℓ.

We abuse the function δ^* so that for a real value h, $\delta^*(h) = \delta^*(\ell(h))$ denotes the optimal rearrangement cost of S onto $\ell(h)$, as defined in Section 3. Then our goal is to minimize $\delta^*(h)$ over all $h \in \mathbb{R}$. Let h^* denote a real value such that $\delta^*(h^*) = \min_h \delta^*(h)$, and let $\delta^* := \delta^*(h^*)$ denote the optimal cost over all horizontal lines. Note that once we know h^*, we can find an optimal rearrangement of S onto $\ell(h^*)$ with cost δ^* in $O(n)$ time by Lemma 2.

Let us define two distance functions, $\delta_{ij}(h)$ and $\sigma_{ij}(h)$, for two indices i, j with $1 \le i \le j \le n$ with respect to a horizontal line $\ell(h)$ for a real value h as follows.

Recall the definition of δ_{ij} from Section 3. We define a function $\delta_{ij}(h)$ of a real value h with respect to $\ell(h)$ for every two indices $i \le j$ similarly. Let q_i and q_j be the points on $\ell(h)$ minimizing $\max\{\|p_i - q_i\|, \|p_j - q_j\|\}$ with $\|q_i - q_j\| \le j - i$. Then $\delta_{ij}(h) = \max\{\|p_i - q_i\|, \|p_j - q_j\|\}$. By Lemma 4, $\delta^*(h) = \max_{i,j} \delta_{ij}(h)$ for any fixed h. Let $d(p, s) = \min_{q \in s} \|p - q\|$ denote the distance from a point p to a line segment s. For a point $p_i \in S$ and an index k, let $s_{ik} = p_i \oplus I(|i - k|)$ denote the horizontal line segment of length $2 \cdot |i - k|$ with midpoint p_i. In case $i = k$, s_{ik} is a degenerate line segment with length 0.

Lemma 16 $\delta_{ij}(h) = \min_{q \in \ell(h)} \max\{d(q, s_{ik}), d(q, s_{jk})\}$ *for any index k with $i \le k \le j$,*

Proof. If $i = j$, the lemma holds obviously as both sides denote the distance from p_j to $\ell(h)$ (Figure 5(a)).

Consider the case that $i < j$. If $|x(p_i) - x(p_j)| \le j - i$, we have $\delta_{ij}(h) = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ by definition. Moreover, there is a point $q \in \ell(h)$ such that the vertical line through q intersects both s_{ik} and s_{jk}. For any such point q, we have $\max\{d(q, s_{ik}), d(q, s_{jk})\} = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ (Figure 5(b)).

If $|x(p_i) - x(p_j)| > j - i$, there are two points $u_i, u_j \in \ell(h)$ such that $\|u_i - u_j\| = j - i$ and $\delta_{ij}(h) = \max\{\|p_i - u_i\|, \|p_j - u_j\|\}$ by the definition of $\delta_{ij}(h)$. Let q be the point on $\ell(h)$ such that $\|q - u_i\| = k - i$ and $\|q - u_j\| = j - k$. Then $\|p_i - u_i\| = d(q, s_{ik})$ and $\|p_j - u_j\| = d(q, s_{jk})$, and for any point $r \in \ell(h)$, either $d(r, s_{ik}) \ge d(q, s_{ik})$ or $d(r, s_{jk}) \ge d(q, s_{jk})$. Therefore, we conclude that $\delta_{ij}(h) = \max\{d(q, s_{ik}), d(q, s_{jk})\}$ (Figure 5(c)). □

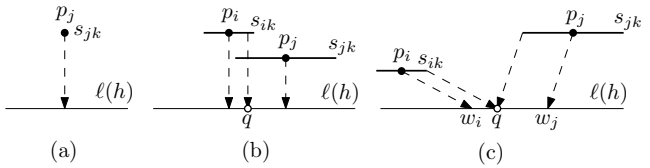

Figure 5: (a) If $i = j$, $\delta_{ij}(h)$ is the distance from p_j to $\ell(h)$. (b) If $|x(p_i) - x(p_j)| \leq j - i$, $\max\{d(q, s_{ik}), d(q, s_{jk})\} = \max\{\delta_{ii}(h), \delta_{jj}(h)\}$ for any point $q \in \ell(h)$ such that the vertical line through q intersects both s_{ik} and s_{jk}. (c) If $|x(p_i) - x(p_j)| > j - i$, we can find a point q on $\ell(h)$ such that $\delta_{ij}(h) = \max\{d(q, s_{ik}), d(q, s_{jk})\}$.

We now define another function $\sigma_{ij}(h)$ for two indices i, j with $1 \leq i \leq j \leq n$ as follows. Let $\mathsf{far}(p, A) = \max_{l \in A} d(p, l)$ for a point $p \in \mathbb{R}^2$ and a set A of horizontal line segments. Let $\mathsf{L}_{ab} = \{s_{ta} \mid a \leq t \leq b\}$ and $\mathsf{R}_{ab} = \{s_{tb} \mid a \leq t \leq b\}$ be sets of segments.

$$\sigma_{ij}(h) = \min_{q \in \ell(h)} \mathsf{far}(q, \mathsf{R}_{ik} \cup \mathsf{L}_{kj}), \text{ for } k = \lceil (i+j)/2 \rceil.$$

We now show that $\sigma_{ij}(h)$ is a convex function consisting of $O(j - i)$ pieces of quadratic functions. Also, we show that there are $O(n)$ pairs of indices such that the upper envelope of $\sigma_{ij}(h)$'s for the pairs coincides with $\delta^*(h)$. The *farthest-site Voronoi diagram* for line segments in A, denoted by $\mathsf{Vor}(A)$ decomposes the plane into regions such that every point p in the same region has the same *farthest* line segment l among the line segments in A, that is, $d(p, l) = \mathsf{far}(p, A)$ for every point p in the region. By computing $\mathsf{Vor}(A)$ for a given set A, we identify a full description of $\mathsf{far}(p, A)$. It is known by Aurenhammer et al. [3] that $\mathsf{Vor}(A)$ consists of $O(|A|)$ vertices, edges, and cells. Using this property, we prove the following lemma.

Lemma 17 *$\sigma_{ij}(h)$ is a convex function consisting of $O(j - i)$ pieces of quadratic functions.*

Proof. Let $A = \mathsf{R}_{ik} \cup \mathsf{L}_{kj}$ with $k = \lceil (i + j)/2 \rceil$. Since $d(q, l)$ is a convex function of $q \in \mathbb{R}^2$ for any fixed line segment $l \in A$, $F(q) := \mathsf{far}(q, A)$ is also convex. Therefore, $\sigma_{ij}(h) = \min_{q \in \ell(h)} F(q)$ is convex. For a value $h \in \mathbb{R}$, let $F_h(q) := F|_{\ell(h)}(q)$ be a function of $q \in \ell(h)$, and $Q(h)$ be the set of points in $\ell(h)$ minimizing F_h. Then $\delta_{ij}(h) = F_h(q) = \mathsf{far}(q, A)$ for any $q \in Q(h)$. Since F_h is convex, $Q(h)$ forms a line segment on $\ell(h)$, possibly being degenerate to a point.

Let $l \in A$ be a farthest segment from $Q(h)$, which is a farthest segment from every point $q \in Q(h)$. Then $\sigma_{ij}(h) = d(q, l)$ for any $q \in Q(h)$. Note that there can be more than two farthest segments from $Q(h)$. We analyze $\sigma_{ij}(h)$ when the number of the farthest segments from $Q(h)$ is (1) one, (2) two, or (3) more than two.

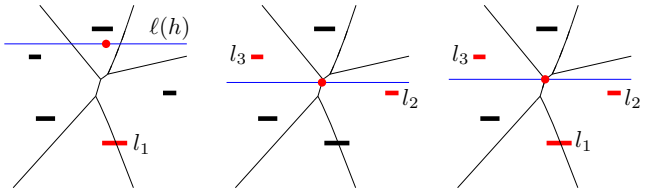

Figure 6: $q \in Q(h)$ is marked as a red dot. For each q, red segments are the farthest segments of A from q. (a) When there is one farthest segment, $\sigma_{ij}(h)$ is a linear function. (b) When there are two farthest segments, $\sigma_{ij}(h)$ is a quadratic function. (c) If there are more then two farthest segments, q lies on the Voronoi vertex of $\mathsf{Vor}(A)$.

(1) Let l_1 be the farthest segment of $Q(h)$. Then there is a point $q \in Q(h)$ lying in the interior of the cell of l_1 in $\mathsf{Vor}(A)$. Therefore, $\sigma_{ij}(h) = d(q, l_1)$ is the difference of y-coordinate of q and l_1, which is a linear function of h (Figure 6(a)). Note that l_1 is a segment with either the largest or the smallest y-coordinate value among the segments in A.

(2) Let l_2 and l_2 be the farthest segments of $Q(h)$. Then there must be a Voronoi edge defined by l_2 and l_3 in $\mathsf{Vor}(A)$, and $Q(h)$ is the intersection of $\ell(h)$ and the Voronoi edge. If the Voronoi edge is a parabolic curve, we can find a point $q \in \ell(h)$ that lies in the interior of the cell of l_2 or l_3 with $F(q) = \sigma_{ij}(h)$ as l_2 and l_3 are horizontal segments which makes one of two segments not farthest segment from $Q(h)$. Therefore, the Voronoi edge must be a line segment, which is the bisector of the endpoints of the segments. Thus, $\delta_{ij}(h)$ is a quadratic function of h (Figure 6(b)).

(3) Let l_1, l_2, and l_3 be three farthest segments of $Q(h)$. Then there must be a Voronoi vertex defined by the segments in $\mathsf{Vor}(A)$ and $Q(h)$ is the Voronoi vertex (Figure 6(c)).

There are at most two (unbounded) intervals of h with case (1). For the case (2), the farthest segments from $Q(h)$ do not change while increasing h until $\ell(h)$ hits a Voronoi vertex of $\mathsf{Vor}(A)$ to make the case (3). As there are $O(j - i)$ vertices of $\mathsf{Vor}(A)$, $\sigma_{ij}(h)$ consists of $O(j - i)$ partial quadratic functions. □

By Lemma 17, the algorithm can compute the function $\sigma_{ij}(h)$ from $\mathsf{Vor}(A)$ in $O(j-i)$ time as follows. The algorithm first identify $\mathsf{far}(p, A)$ from $\mathsf{Vor}(A)$ in $O(j-i)$ time. Then it computes $\sigma_{ij}(h)$ from $\mathsf{far}(p, A)$ by keeping track of $Q(h)$ and the farthest segments of $Q(h)$ while increasing h from $-\infty$ to ∞. As the farthest segments of $Q(h)$ changes $O(j - i)$ times and each change can be identified from a Voronoi vertex of $\mathsf{Vor}(A)$ in $O(1)$ time, the algorithm takes $O(j - i)$ time in total.

Since $\mathsf{far}(q, A)$ is induced by $\mathsf{Vor}(A)$, $\sigma_{ij}(h)$ is the radius of the smallest disk intersecting every segment in $\mathsf{R}_{ik} \cup \mathsf{L}_{kj}$ whose center is restricted to lie on $\ell(h)$. If the center is determined by a single line segment s_{ak}, then $\sigma_{ij}(h) = \delta_{aa}(h)$. If the center is determined by two line segments, s_{ak} and s_{bk} with $a \le k \le b$, $\sigma_{ij}(h) = \delta_{ab}(h)$. We prove these relations between $\sigma_{ij}(h)$ and $\delta_{ab}(h)$ in Lemmas 18 and 19.

Lemma 18 *For any value h, there is a pair (i', j') with $i \le i' \le j' \le j$ such that $\sigma_{ij}(h) \le \delta_{i'j'}(h)$.*

Proof. There is a segment s_{ak} with $i \le a \le j$ such that $\min_{q \in \ell(h)} d(q, s_{ak}) = \sigma_{ij}(h)$, or there are two segments s_{ak}, s_{bk} with $i \le a < b \le j$ such that $\min_{q \in \ell(h)} \max\{d(q, s_{ak}), d(q, s_{bk})\} = \sigma_{ij}(h)$.

In the first case, $\sigma_{ij}(h) = \delta_{aa}(h)$. In the second case, let q^* be a point on $\ell(h)$ such that $\sigma_{ij}(h) = \max\{d(q^*, s_{ak}), d(q^*, s_{bk})\}$. Then $\sigma_{ij}(h) = \delta_{ab}(h)$ for $a \le k$ and $k \le b$. When $b < k$, $s_{ab} \subset s_{ak}$ and $s_{bb} \subset s_{bk}$. Therefore, $\sigma_{ij}(h) = \max\{d(q^*, s_{ak}), d(q^*, s_{bk})\} \le \max\{d(q^*, s_{ab}), d(q^*, s_{bb})\} = \delta_{ab}(h)$. We can show that $\sigma_{ij}(h) \le \delta_{ab}(h)$ for $k < a$ similarly. □

Lemma 19 *For any index pair (i', j') with $i \le i' \le \lceil (i+j)/2 \rceil \le j' \le j$, we have $\delta_{i'j'}(h) \le \sigma_{ij}(h)$.*

Proof. For any fixed index a with $i \le a \le j$ and $k = \lceil (i+j)/2 \rceil$, $d(q, s_{ak})$ for $q \in \ell(h)$ is convex. Therefore, $\sigma_{ij}(h) = \max_{i \le a \le b \le j} \min_{q \in \ell(h)} \mathsf{far}(q, \{s_{ak}, s_{bk}\})$. As $\min_{q \in \ell(h)} \mathsf{far}(q, \{s_{ak}, s_{bk}\}) = \delta_{ab}(h)$ for $i \le a \le k$ and $k \le b \le j$, the inequality holds. □

Data structures and algorithm. Let $\mathcal{T}$ be a binary tree such that the root corresponds to $S = S_{1n}$, each internal node v corresponds to a subsequence S_{ab} of S for some $a \le b$ with its left and right children corresponding to S_{ac} and S_{cb}, respectively, for $c = \lceil (a+b)/2 \rceil$. Each leaf node of $\mathcal{T}$ corresponds to a subsequence consisting of a single point, so $\mathcal{T}$ has $O(n)$ nodes and its height is $O(\log n)$. At each node v of $\mathcal{T}$ corresponding to S_{ab}, we store $\delta_v(h) = \sigma_{ab}(h)$. Then we show the following lemma.

Lemma 20 *For any value h, $\delta^*(h) = \max_{v \in \mathcal{T}} \delta_v(h)$.*

Proof. For a node v of $\mathcal{T}$, $\delta_v(h) \le \max_{1 \le i \le j \le n} \delta_{ij}(h) = \delta^*(h)$ by Lemma 18. So we have $\max_{v \in \mathcal{T}} \delta_v(h) \le \delta^*(h)$.

We now show $\max_{v \in \mathcal{T}} \delta_v(h) \ge \delta^*(h)$. For any two indices i, j with $1 \le i \le j \le n$, let $v \in \mathcal{T}$ be the lowest node of $\mathcal{T}$ such that its corresponding subsequence S_{ab} contains both p_i and p_j, that is, $a \le i \le j \le b$. Note that $\delta_v(h) = \sigma_{ab}(h)$. Further, by our construction, $a \le i \le \lceil (a+b)/2 \rceil \le j \le b$. Then, by Lemma 19, $\delta_{ij}(h) \le \delta_v(h)$. Hence, $\delta^*(h) = \max_{i,j} \delta_{ij}(h) \le \max_v \delta_v(h)$. □

The algorithm computes $\delta_v(h)$ for nodes $v \in \mathcal{T}$ in bottom-up fashion. For each node v corresponding to S_{ab}, the algorithm stores $\mathsf{Vor}(\mathsf{R}_{ab})$ and $\mathsf{Vor}(\mathsf{L}_{ab})$. If v is a leaf node, the algorithm computes $\delta_v(h)$ in $O(1)$ time. If v is an internal node, the algorithm computes $\delta_v(h)$ in time linear to the length of the corresponding subsequence of v by using $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{L}_{cb})$ with $c = \lceil (a+b)/2 \rceil$ stored in the child nodes of v. The details on computing $\delta_v(h)$ for an internal node v are in the following.

Since every cell in $\mathsf{Vor}(A)$ is unbounded, there is a cyclic order of the cells of $\mathsf{Vor}(A)$ along a closed curve at infinity. The algorithm by Aurenhammer et al. [3] computes $\mathsf{Vor}(A)$ in two stages: finds out the cyclic order of the cells in $O(|A| \log |A|)$ time, and then constructs the diagram based on the order in additional $O(|A| \log |A|)$ time. Later, Khramtcova and Papadopoulou [13] showed that the second stage can be done in $O(|A|)$ time. From the two results, we have the following lemma.

Lemma 21 *Given $\mathsf{Vor}(A)$ and $\mathsf{Vor}(B)$ for two sets A and B of $O(n)$ line segments in total, we can compute $\mathsf{Vor}(A \cup B)$ in $O(n)$ time.*

Proof. We can compute the cyclic order of the cells of $\mathsf{Vor}(A)$ and the cyclic order of the cells of $\mathsf{Vor}(B)$ in $O(n)$ time. Then we can merge them into the cyclic order of cells of $\mathsf{Vor}(A \cup B)$ in $O(n)$ time using the algorithm by Aurenhammer et al. [3]. Finally, we can compute $\mathsf{Vor}(A \cup B)$ based on the order in $O(n)$ time [13]. □

Now we present an $O(n)$ time algorithm to compute $\mathsf{Vor}(\mathsf{R}_{ab})$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{R}_{cb})$ for $c = \lceil (a+b)/2 \rceil$. We first compute $\mathsf{Vor}(\mathsf{R}_{ac} \oplus I(b-c))$.

Lemma 22 *For a set A of horizontal line segments, the cyclic order of the cells of $\mathsf{Vor}(A)$ and the cyclic order of the cells of $\mathsf{Vor}(A \oplus I(r))$ are the same for any real value $r > 0$.*

Proof. For a direction $\vec{d}$, we use $C(\vec{d})$ to denote the cell of a farthest-site Voronoi diagram such that for any point p, there is a point q on the ray of direction $\vec{d}$ emanated from p such that $q \in C(\vec{d})$. For $\mathsf{Vor}(A)$, $C(\vec{d})$ is the cell of site $l \in A$ if and only if there exists an open half plane with inner normal vector $\vec{d}$ that intersects all the line segments of $A \setminus \{l\}$ but not l. Let h be an open half plane that intersects every line segment of $A \setminus \{l\}$. Let h' be the translate of h along the x-axis towards $\vec{d}$ by r. Then h' intersects every line segment of $(A \setminus \{l\}) \oplus I(r)$ but not $l \oplus I(r)$. Therefore, for any fixed direction $\vec{d}$, $\mathsf{Vor}(A)$ and $\mathsf{Vor}(A \oplus I(r))$ have the same site (line segment) defining $C(\vec{d})$ in their diagrams. □

Therefore, the algorithm can compute $\mathsf{Vor}(\mathsf{R}_{ac} \oplus I(b-c))$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ in linear time by Lemma 22. As

$\mathsf{R}_{ab} = (\mathsf{R}_{ac} \oplus I(b-c)) \cup \mathsf{R}_{cb}$, the algorithm can also compute $\mathsf{Vor}(\mathsf{R}_{ab})$ in linear time by Lemma 21. Computing $\mathsf{Vor}(\mathsf{L}_{ab})$ can be done in linear time as well.

For an internal node $v \in \mathcal{T}$, the algorithm computes $\mathsf{Vor}(\mathsf{R}_{ac} \cup \mathsf{L}_{cb})$ from $\mathsf{Vor}(\mathsf{R}_{ac})$ and $\mathsf{Vor}(\mathsf{L}_{cb})$ in linear time by Lemma 21. Then the algorithm computes $\delta_v(h)$ from $\mathsf{Vor}(\mathsf{R}_{ac} \cup \mathsf{L}_{cb})$ in linear time by Lemma 17. Therefore, we can conclude the following lemma.

Lemma 23 *We can compute an explicit description of function $\delta_v(h)$ for all nodes v of $\mathcal{T}$ in $O(n \log n)$ time.*

Convex programming with $\delta_v(h)$'s. Recall that $\delta_v(h)$ for each node v is convex by Lemma 17. To find the lowest point on the function $\delta^*(h)$, the algorithm computes h^* and $\delta^*(h^*)$ using convex programming by taking $\delta_v(h)$ as a constraint for each v and $f(h) = h$ as the objective function. Chan [5] showed that the convex programming can be done by $O(k \log k)$ primitive operations, where k is the number of constraints and the primitive operations are (1) to find a point that optimizes the objective function while satisfying two constraints, or (2) to find intersections between a constraint and a line. Our problem consists of $O(n)$ constraints ($\delta_v(h)$'s) and the primitive operation takes $O(\log n)$ time as there are $O(n)$ nodes in $\mathcal{T}$ and $\delta_v(h)$ consists of $O(n)$ partial functions. Therefore, we obtain an $O(n \log^2 n)$-time algorithm to compute h^* and $\delta^*(h^*)$. However, as the complexity of $\delta_v(h)$ varies through the nodes of $\mathcal{T}$, we can reduce the number of constraints without increasing the time complexity of primitive operation as follows.

Improving time complexity. For a node u with corresponding subsequence of length L with $\lceil \log n \rceil \leq L < 2\lceil \log n \rceil$, we compute the upper envelope of $\delta_v(h)$ for nodes v in the subtree with root u. Observe that the upper envelope consists of $O(\log n \log\log n \cdot 2^{\alpha(n)})$ partial functions, where $\alpha(\cdot)$ is the inverse Ackermann function [16]. The bound comes from the fact that every partial function of $\delta_v(h)$ is a quadratic function by Lemma 17, so that any two partial functions intersect at most twice. Thus we can compute the upper envelope in $O(\log n \log^2 \log n \cdot 2^{\alpha(n)})$ time. There are $O(n/\log n)$ such nodes, and thus computing the upper envelopes for all such nodes takes $O(n \log^2 \log n \cdot 2^{\alpha(n)})$ time. Thus, we have $O(n/\log n)$ convex constraints, each consisting of $O(n)$ partial functions. Therefore, convex programming can be done in $O(n \log n)$ time. Since all $\delta_v(h)$'s can be computed in $O(n \log n)$ time by Lemma 23, h^* and $\delta^*(h^*)$ can be computed in $O(n \log n)$ time. Then the algorithm computes an optimal rearrangement of S onto $\ell(h^*)$ in $O(n)$ time by Lemma 2.

Theorem 24 *For a sequence S of n points and an orientation, we can compute an optimal rearrangement of S onto a line with the orientation in $O(n \log n)$ time.*

4.1 Weighted version

In this section, we show that the algorithm above can be extended to weighted version without increasing the time complexity. By changing the definition of s_{ik} to $p_i \oplus I(w_k - w_i)$, Lemmas 17 and 20 hold for the weighted version. Therefore, by Lemmas 21 and 22, we can compute $\delta_v(h)$ for each node v in $\mathcal{T}$ in time linear to the length of the corresponding subsequence in bottom-up fashion. By computing $\delta_v(h)$'s for nodes v in $\mathcal{T}$, we obtain the following theorem.

Theorem 25 *For a sequence S of n weighted points and an orientation, we can compute an optimal rearrangement of S onto a line with the orientation in $O(n \log n)$ time.*

5 Rearrangement into an Arbitrary Line

Given a sequence of $S = \langle p_1, \ldots, p_n \rangle$ of n points in the plane, we find a line ℓ such that the optimal rearrangement cost $\delta^*(\ell)$ of S onto ℓ is minimum among all lines in the plane, and compute an optimal rearrangement of S onto ℓ.

Due to the limit of space, the detailed algorithm is given in Appendix.

Theorem 26 *For a sequence S of n points, we can compute an optimal rearrangement of S onto any line in $O(n^3 \operatorname{polylog} n)$ time.*

Theorem 27 *For a sequence S of n weighted points, we can compute an optimal rearrangement of S onto any line in $O(n^3 \operatorname{polylog} n)$ time.*

6 Conclusion

We consider the problem of finding an optimal rearrangement of sequence of n points onto a line. We present an expected $O(n)$-time algorithm and deterministic $O(n \log\log n)$-time algorithm for a given line. When the rearrangement line is given only by its orientation or not specfied at all, we present $O(n \log n)$-time and $O(n^3 \operatorname{polylog} n)$-time algorithm, respectively.

Our algorithms are described for solving the rearrangement problems under the Euclidean metric, but they can be applied to the cases that the underlying metric is L_p or has the distance function of constant complexity.

There are a few works to study. One is to improve the deterministic time complexity to linear for a given rearrangement line, or to show a tight time bound on the problem. Another one is to study the optimal rearrangement problem for more general objects.

References

[1] P. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Computational Geometry*, 16:317–337, 1996.

[2] P. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *Journal of Algorithms*, 17(3):292–318, 1994.

[3] F. Aurenhammer, R. Drysdale, and H. Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006.

[4] R. Bellman and R. Roth. Curve fitting by segmented straight lines. *Journal of the American Statistical Association*, 64(327):1079–1084, 1969.

[5] T. M. Chan. Deterministic algorithms for 2-d convex programming and 3-d online linear programming. *Journal of Algorithms*, 27(1):147–166, 1998.

[6] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete & Computational Geometry*, 22:547–567, 1999.

[7] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, Jan. 1987.

[8] M. de Berg, P. Bose, D. Bremner, S. Ramaswami, and G. Wilfong. Computing constrained minimum-width annuli of point sets. *Computer-Aided Design*, 30(4):267–275, 1998. Computational Geometry and Computer-Aided Design and Manufacturing.

[9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[10] C. A. Duncan, M. T. Goodrich, and E. A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 1997, page 121–130, USA, 1997. Society for Industrial and Applied Mathematics.

[11] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[12] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):761–765, 1988.

[13] E. Khramtcova and E. Papadopoulou. Linear-time algorithms for the farthest-segment Voronoi diagram and related tree structures. In *Proceedings of the 26th International Symposium on Algorithms and Computation*, ISAAC 2015, pages 404–414, 2015.

[14] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1982.

[15] F. P. Preparata. New parallel-sorting schemes. *IEEE Transactions on Computers*, C-27(7):669–673, 1978.

[16] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, USA, 2010.

[17] L. G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4(3):348–355, 1975.

[18] H. Wang and J. Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry & Applications*, 26(03n04):185–210, 2016.

Appendix

Detailed Algorithm of Section 5

We present a sequential decision algorithm and a parallel decision algorithm, both determining whether $\delta_0 \geq \delta^*$ for given δ_0, where δ^* is the minimum $\delta^*(\ell)$ among all lines ℓ in the plane. We apply Cole's parametric search [7] using the decision algorithms. For a line in the plane, we use the y-coordinate of y-intercept of the line as the height of the line. We say a line has an orientation α for $\alpha \in [0, \pi)$ if the counterclockwise angle from x-axis to the line is α. We use $\ell(h, \alpha)$ to denote the line of height h and an orientation α.

Sequential decision algorithm We are given a real value δ_0. For an orientation α, and $1 \leq i \leq j \leq n$, we define two functions $h_{ij}(\alpha)$ and $l_{ij}(\alpha)$. We use $h_{ij}(\alpha)$ to denote the maximum height h such that there are translations of p_i and p_j onto $\ell(h, \alpha)$ with length at most δ_0 and the distance between the translates is at most $j-i$. We define $l_{ij}(\alpha)$ as the minimum height as well. If there is no line with orientation α such that the translations exist, the functions are not defined for α. Note that for any distinct pairs of indices (i, j) and (i', j') with $i \leq j$ and $i' \leq j'$, h_{ij} and $h_{i'j'}$ intersect each other $O(1)$ times. We compute $H(\alpha) = \min_{1 \leq i \leq j \leq n} h_{ij}(\alpha)$ and $L(\alpha) = \max_{1 \leq i \leq j \leq n} l_{ij}(\alpha)$. Functions $H(\alpha)$ and $L(\alpha)$ are not defined for α if there are indices $i \leq j$ such that $h_{ij}(\alpha)$ and $l_{ij}(\alpha)$ are not defined.

Lemma 28 *There exists a rearrangement of S onto a line with cost at most δ_0 if and only if $H(\alpha) \geq L(\alpha)$ for some α.*

Proof. Let Q be a rearrangement of S onto $\ell(h, \alpha_0)$ with cost at most δ_0. By definition, $h_{ij}(\alpha_0) \geq h \geq l_{ij}(\alpha_0)$ for every index pair $i \leq j$. Therefore, $H(\alpha) \geq L(\alpha)$.

If $H(\alpha_0) \geq L(\alpha_0)$ for some α_0, let h be a height with $H(\alpha_0) \geq h \geq L(\alpha_0)$. By Lemma 4, there is a rearrangement of S onto $\ell(h, \alpha_0)$ with cost at most δ_0. □

The algorithm checks whether there is a certain orientation α satisfying $H(\alpha) \geq L(\alpha)$. As $h_{ij}(\alpha)$ and $h_{i'j'}(\alpha)$ intersects each other $O(1)$ times, $H(\alpha)$ consists of $O(n^2 \operatorname{polylog} n)$ partial functions and can be computed in $O(n^2 \operatorname{polylog} n)$ time. Similarly, the algorithm computes $L(\alpha)$ in $O(n^2 \operatorname{polylog} n)$ time. After computing $H(\alpha)$ and $L(\alpha)$, the algorithm checks whether $H(\alpha) \geq L(\alpha)$ holds for each partial intervals, which takes $O(n^2 \operatorname{polylog} n)$ time. In total, the algorithm takes $O(n^2 \operatorname{polylog} n)$ time.

Parallel decision algorithm The parallel decision algorithm computes $H(\alpha)$ and $L(\alpha)$ for a given real number δ_0 as follows. Let $H_i(\alpha) = \min_j h_{ij}(\alpha)$ and $L_i(\alpha) = \max_j l_{ij}(\alpha)$. They are not defined for α if there are indices $i \leq j$ such that $h_{ij}(\alpha)$ and $l_{ij}(\alpha)$ are not defined. Observe that $H_i(\alpha)$ and $L_i(\alpha)$ consist of $O(n \operatorname{polylog} n)$ partial functions. By assigning a processor to each index i, it takes $O(n \operatorname{polylog} n)$ time to compute $H_i(\alpha)$ and $L_i(\alpha)$. Then using $O(n^2)$ processors, the algorithm computes all the intervals such that $H_i(\theta) \geq L_j(\theta)$ for every index pair i, j. There are $O(n)$ such intervals for each index pair i, j. The algorithm sorts the endpoints of the intervals in $O(n \operatorname{polylog} n)$ time using $O(n^2)$ processors by a parallel sorting algorithm such as Preparata's [15] or Valiant's [17]. The algorithm finds whether there exists an interval of orientations α such that $H(\alpha) \geq L(\alpha)$ by checking every sorted intervals. In total, the algorithm takes $O(n \operatorname{polylog} n)$ time using $O(n^2)$ processors.

Parametric search Now we have an $O(n^2 \operatorname{polylog} n)$-time sequential decision algorithm and an $O(n \operatorname{polylog} n)$-time parallel decision algorithm using $O(n^2)$ processors. By applying parametric search, we can find an optimal rearrangement of S in $O(n^3 \operatorname{polylog} n)$ time. Therefore, Theorem 26 holds.

Weighted version Observe that both decision algorithms also work for the weighted version in same time bound by setting $h_{ij}(\alpha)$ and $l_{ij}(\alpha)$ to the maximum and the minimum height of the line with orientation α, respectively, such that p_i and p_j can be placed within distance $w_j - w_i$ with cost at most δ_0 for index pair $i \leq j$. Therefore, Theorem 27 holds.