The (1|1)-Centroid Problem on the Plane Concerning Distance Constraints*

Hung-I Yu[1], Tien-Ching Lin[2], and Der-Tsai Lee[3]

1 **Institute of Information Science, Academia Sinica, Taipei 115, Taiwan**
herbert@iis.sinica.edu.tw
2 **Institute of Information Science, Academia Sinica, Taipei 115, Taiwan**
kero@iis.sinica.edu.tw
3 **Institute of Information Science, Academia Sinica, Taipei 115, Taiwan**
dtlee@iis.sinica.edu.tw

Abstract

In 1982, Drezner proposed the (1|1)-centroid problem on the plane, in which two players, called the leader and the follower, open facilities to provide service to customers in a competitive manner. The leader opens the first facility, and then the follower opens the second. Each customer will patronize the facility closest to him (ties broken in favor of the leader's one), thereby decides the market share of the two players. The goal is to find the best position for the leader's facility so that his market share is maximized. The best algorithm for this problem is an $O(n^2 \log n)$-time parametric search approach, which searches over the space of possible market share values.

In the same paper, Drezner also proposed a general version of (1|1)-centroid problem by introducing a minimal distance constraint R, such that the follower's facility is not allowed to be located within a distance R from the leader's. He proposed an $O(n^5 \log n)$-time algorithm for this general version by identifying $O(n^4)$ points as the candidates of the optimal solution and checking the market share for each of them. In this paper, we develop a new parametric search approach searching over the $O(n^4)$ candidate points, and present an $O(n^2 \log n)$-time algorithm for the general version, thereby closing the $O(n^3)$ gap between the two bounds.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases competitive facility, Euclidean plane, parametric search

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.64

1 Introduction

In 1929, economist Hotelling introduced the first competitive location problem in his seminal paper [13]. Since then, the subject of competitive facility location has been extensively studied by researchers in the fields of spatial economics, social and political sciences, and operations research, and spawned hundreds of contributions in the literature. The interested reader is referred to the following survey papers [1, 4, 8, 9, 10, 12, 17, 19].

Hakimi [11] and Drezner [6] independently proposed a series of competitive location problems in a leader-follower framework. The framework is briefly described as follows. There are n customers in the market, and each is endowed with a certain buying power. Two players, called the *leader* and the *follower*, sequentially open facilities to attract the

* Research supported by the Ministry of Science and Technology under Grants MOST 104-2221-E-001-031, 104-2221-E-001-032.

buying power of customers. At first, the leader opens his p facilities, and then the follower opens another r facilities. Each customer will patronize the closest facility with all buying power (ties broken in favor of the leader's ones), thereby deciding the market share of the two players. Since both players ask for market share maximization, two competitive facility location problems are defined. Given that the leader locates his p facilities at the set X_p of p points, the follower wants to locate his r facilities in order to attract the most buying power, called the $(r|X_p)$-*medianoid* problem. On the other hand, knowing that the follower will react with maximization strategy, the leader wants to locate his p facilities in order to retain the most buying power against the competition, called the $(r|p)$-*centroid* problem.

Drezner [6] first proposed to study the two competitive facility location problems on the Euclidean plane. Since then, several related results [5, 7, 12, 14] have been obtained for different values of r and p. Due to page limit, here we introduce only previous results about the case $r = p = 1$. For the $(1|X_1)$-medianoid problem, Drezner [6] showed that there exists an optimal solution arbitrarily close to X_1, and solved the problem in $O(n \log n)$ time by a sweeping technique. Later, Lee and Wu [14] obtained an $\Omega(n \log n)$ lower bound for the $(1|X_1)$-medianoid problem, and thus proved the optimality of Drezner's result. For the (1|1)-centroid problem, Drezner [6] developed a parametric search approach that searches over the space of $O(n^2)$ possible market share values, along with an $O(n^4)$-time test procedure constructing and solving a linear program of $O(n^2)$ constraints, and gave an $O(n^4 \log n)$-time algorithm. Then, by improving the test procedure via Megiddo's linear-time result [16] for solving linear programs, Hakimi [12] reduced the time complexity to $O(n^2 \log n)$.

In [6], Drezner also proposed a more general setting for the leader-follower framework by introducing a *minimal distance constraint* $R \geq 0$ into the $(1|X_1)$-medianoid problem and the (1|1)-centroid problem, such that the follower's facility is not allowed to be located within a distance R from the leader's. The augmented problems are respectively called the $(1|X_1)_R$-*medianoid* problem and $(1|1)_R$-*centroid* problem in this paper. Drezner showed that the $(1|X_1)_R$-medianoid problem can also be solved in $O(n \log n)$ time by using nearly the same proof and technique as for the $(1|X_1)$-medianoid problem. However, for the $(1|1)_R$-centroid problem, he argued that it is hard to generalize the parametric approach approach for the (1|1)-centroid problem to solve this general version, due to the change of problem properties. Then, he gave an $O(n^5 \log n)$-time algorithm by identifying $O(n^4)$ candidate points on the plane, which contain at least one optimal solution, and performing medianoid computation on each of them. So far, the $O(n^3)$ gap between the two centroid problems remains open.

In this paper, we propose an $O(n^2 \log n)$-time algorithm for the $(1|1)_R$-centroid problem on the Euclidean plane, thereby closing the gap that has existed for decades. Instead of searching over market share values, we develop a new approach based on a different parametric search technique by searching over the $O(n^4)$ candidate points mentioned in [6]. This is made possible by making a critical observation on the distribution of optimal solutions for the $(1|X_1)_R$-medianoid problem given X_1, which provides us a useful tool to prune candidate points with respect to X_1. We then extend the usage of this tool to design a key procedure to prune candidates with respect to a given vertical line. Due to page limits, most of the proofs are omitted.

The rest of this paper is organized as follows. Section 2 gives formal problem definitions and describes previous results in [6, 12]. In Section 3, we make the observation on the $(1|X_1)_R$-medianoid problem, and use it to find a "local" centroid on a given line. This result is then extended as a new pruning procedure with respect to any given line in Section 4, and utilized in our parametric search approach for the $(1|1)_R$-centroid problem. Finally, in Section 5, we give some concluding remarks.

2 Notations and Preliminary Results

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of n points on the Euclidean plane $\mathbb{R}^2$, as the representatives of the n customers. Each point $v_i \in V$ is assigned a positive weight $w(v_i)$, representing its buying power. To simplify the presentation, we assume that the points in V are in general position, that is, no three points are collinear and no two points share a common x or y-coordinate.

Let $d(u, w)$ denote the Euclidean distance between any two points $u, w \in \mathbb{R}^2$. For any set Z of points on the plane, we define $W(Z) = \sum\{w(v)|v \in V \bigcap Z\}$. Suppose that the leader has located his facility at $X_1 = \{x\}$, which is shortened as x for simplicity. Due to the minimal distance constraint R mentioned in [6], any point $y' \in \mathbb{R}^2$ with $d(y', x) < R$ is infeasible to be the follower's choice. If the follower locates his facility at some feasible point y, the set of customers patronizing y instead of x is defined as $V(y|x) = \{v \in V | d(v, y) < d(v, x)\}$, with their total buying power $W(y|x) = W(V(y|x))$. Then, the largest market share that the follower can capture is denoted by the function $W^*(x) = \max_{y \in \mathbb{R}^2, d(y,x) \geq R} W(y|x)$, which is called the *weight loss* of x. Given a point $x \in \mathbb{R}^2$, the $(1|x)_R$-medianoid problem is to find a $(1|x)_R$-*medianoid*, which is a feasible point $y^* \in \mathbb{R}^2$ such that $W(y^*|x) = W^*(x)$.

In contrast, the leader tries to minimize the weight loss of his own facility by finding a point $x^* \in \mathbb{R}^2$ such that $W^*(x^*) \leq W^*(x)$ for any point $x \in \mathbb{R}^2$. The $(1|1)_R$-centroid problem is to find a $(1|1)_R$-*centroid*, which is a point x^* minimizing its own weight loss. Note that, when $R = 0$, the two problems degenerate to the $(1|x)$-medianoid problem and $(1|1)$-centroid problem.

2.1 Previous approaches

In this subsection, we briefly review previous results for the $(1|x)_R$-medianoid, $(1|1)$-centroid, and $(1|1)_R$-centroid problems in [6, 12], so as to derive properties essential to our approach.

Let L be an arbitrary line, which partitions the Euclidean plane into two half-planes. For any point $y \notin L$, we define $H(L, y)$ as the closed half-plane including L and y, and $H^-(L, y)$ as the open half-plane $H(L, y) \backslash L$. For any two distinct points $x, y \in \mathbb{R}^2$, let $B(y|x)$ denote the perpendicular bisector of $\overline{xy}$, the line segment connecting x and y.

Given an arbitrary point $x \in \mathbb{R}^2$, we first describe the algorithm for finding a $(1|x)_R$-medianoid in [6]. Let y be a feasible point other than x, and y' be some point on the open line segment $\overline{xy} \backslash \{x, y\}$. We can see that $H^-(B(y|x), y) \subset H^-(B(y'|x), y')$, which implies the fact that $W(y'|x) = W(H^-(B(y'|x), y')) \geq W(H^-(B(y|x), y)) = W(y|x)$, It shows that moving y toward x does not diminish its weight capture, thereby follows the lemma.

▶ **Lemma 1** ([6]). *There exists a $(1|x)_R$-medianoid in $\{y \mid y \in \mathbb{R}^2, d(x, y) = R\}$.*

For any point $z \in \mathbb{R}^2$, let $C_R(z)$ and $C_\gamma(z)$ be the circles centered at z with radii R and $\gamma = R/2$, respectively. By Lemma 1, finding a $(1|x)_R$-medianoid can be reduced to searching a point y on $C_R(x)$ maximizing $W(y|x)$. Since the perpendicular bisector $B(y|x)$ of each point y on $C_R(x)$ is a tangent line to the circle $C_\gamma(x)$, the searching of y on $C_R(x)$ is equivalent to finding a tangent line to $C_\gamma(x)$ that partitions the most weight from x. The latter problem can be solved in $O(n \log n)$ time as follows. For each $v \in V$ outside $C_\gamma(x)$, we calculate its two tangent lines to $C_\gamma(x)$. Then, by sorting these tangent lines according to the polar angles of their corresponding tangent points with respect to x, we can use an angle sweeping technique to check how much weight they partition.

▶ **Theorem 2** ([6]). *Given a point $x \in \mathbb{R}^2$, the $(1|x)_R$-medianoid problem can be solved in $O(n \log n)$ time.*

Next, we describe the algorithm of the $(1|1)_R$-centroid problem in [6]. Let S be a subset of V. We define $\mathcal{C}(S)$ to be the set of all circles $C_\gamma(v)$, $v \in S$, and $CH(\mathcal{C}(S))$ to be the convex hull of these circles. For any positive number W_0, let $I(W_0)$ be the intersection of all convex hulls $CH(\mathcal{C}(S))$, where $S \subseteq V$ and $W(S) \geq W_0$. Drezner [6] argued that the set of all $(1|1)_R$-centroids is equivalent to the intersection $I(W_0)$ for the smallest possible W_0. We slightly clarify his argument below. Let $\mathcal{W} = \{W(y|x) \mid x, y \in \mathbb{R}^2, d(x,y) \geq R\}$. The following lemma can be obtained.

▶ **Lemma 3.** *Let W_0^* be the smallest number in $\mathcal{W}$ such that $I(W_0^*)$ is not null. A point x is a $(1|1)_R$-centroid if and only if $x \in I(W_0^*)$.*

Although it is hard to compute $I(W_0^*)$ itself, we can find its vertices as solutions to the $(1|1)_R$-centroid problem. Let $\mathcal{T}$ be the set of outer tangent lines of all pairs of circles in $\mathcal{C}(V)$. For any subset $S \subseteq V$, the boundary of $CH(\mathcal{C}(S))$ is formed by segments of lines in $\mathcal{T}$ and arcs of circles in $\mathcal{C}(V)$. Since $I(W_0)$ is an intersection of such convex hulls, its vertices must fall within the set of intersection points between lines in $\mathcal{T}$, between circles in $\mathcal{C}(V)$, and between one line in $\mathcal{T}$ and one circle in $\mathcal{C}(V)$. Let $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{C}(V)$, and $\mathcal{T} \times \mathcal{C}(V)$ denote the three sets of intersection points, respectively. We have the lemma below.

▶ **Lemma 4** ([6])**.** *There exists a $(1|1)_R$-centroid in $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{C}(V)$, and $\mathcal{T} \times \mathcal{C}(V)$.*

Obviously, there are at most $O(n^4)$ intersection points, which can be viewed as the *candidates* of being $(1|1)_R$-centroids. Drezner thus gave an algorithm by evaluating the weight loss of each candidate by Theorem 2.

▶ **Theorem 5** ([6])**.** *The $(1|1)_R$-centroid problem can be solved in $O(n^5 \log n)$ time.*

We remark that, when $R = 0$, $CH(\mathcal{C}(S))$ for any $S \subseteq V$ degenerates to a convex polygon, so does $I(W_0)$ for any given W_0, if not null. Drezner [6] proved that in this case $I(W_0)$ is equivalent to the intersection of all half-planes H with $W(H) \geq W_0$. Thus, whether $I(W_0)$ is null can be determined by constructing and solving a linear program of $O(n^2)$ constraints, which takes $O(n^2)$ time by Megiddo's result [16]. Since $|\mathcal{W}| = O(n^2)$, according to Lemma 3 the $(1|1)$-centroid problem can be solved in $O(n^2 \log n)$ time [12], by applying parametric search over $\mathcal{W}$ for W_0^*. Unfortunately, it is hard to generalize this idea to the case $R > 0$.

3 Local $(1|1)_R$-Centroid within a Line

In this section, we analyze the properties of $(1|x)_R$-medianoids of a given point x in Subsection 3.1, and derive a procedure that prunes candidate points with respect to x. Applying this procedure, we study a restricted version of the $(1|1)_R$-centroid problem in Subsection 3.2, in which the leader's choice is limited to a given line L, and obtain an $O(n \log^2 n)$-time algorithm. The algorithm is then extended as the basis of the vertical-line test procedure for the parametric search approach in Section 4.

3.1 Pruning with Respect to a Point

Given a point $x \in \mathbb{R}^2$ and an angle θ between 0 and 2π, let $y(\theta|x)$ be the point on $C_R(x)$ with polar angle θ with respect to x.[1] We define $MA(x) = \{\theta \,|\, W(y(\theta|x)|x) = W^*(x), 0 \leq \theta < 2\pi\}$, that is, the set of angles θ maximizing $W(y(\theta|x)|x)$. It can be observed that, for any

[1] We assume that a polar angle is measured counterclockwise from the positive x-axis.

$\theta \in MA(x)$ and sufficiently small ϵ, both $\theta + \epsilon$ and $\theta - \epsilon$ belong to $MA(x)$, because each $v \in V(y(\theta|x)|x)$ does not intersect $B(y(\theta|x)|x)$ by definition. This implies that angles in $MA(x)$ form open angle interval(s) of non-zero length.

To simplify the terms, let $W(\theta|x) = W(y(\theta|x)|x)$ and $B(\theta|x) = B(y(\theta|x)|x)$ in the remaining parts. Also, let $F(\theta|x)$ be the line passing through x and parallel to $B(\theta|x)$. The following lemma provides the basis for pruning candidates.

▶ **Lemma 6.** *Let $x \in \mathbb{R}^2$ be an arbitrary point, and θ be an angle in $MA(x)$. For any point $x' \notin H^-(F(\theta|x), y(\theta|x))$, $W^*(x') \geq W^*(x)$.*

This lemma tells us that, given a point x and an angle $\theta \in MA(x)$, all points not in $H^-(F(\theta|x), y(\theta|x))$ can be ignored while finding $(1|1)_R$-centroids, as their weight losses are no less than that of x. Besides, the distribution of angles in $MA(x)$ is also meaningful. Let $CA(x)$ be the minimum angle interval covering all angles in $MA(x)$, and $\delta(CA(x))$ be its angle span in radians. Since $MA(x)$ consists of open angle interval(s) of non-zero length, $CA(x)$ is also an open interval and $\delta(CA(x)) > 0$. Moreover, we can derive the following.

▶ **Lemma 7.** *If $\delta(CA(x)) > \pi$, x is a $(1|1)_R$-centroid.*

We call a point x satisfying Lemma 7 a *strong $(1|1)_R$-centroid*, since its discovery gives an immediate solution to the $(1|1)_R$-centroid problem. Note that there are problem instances in which no strong $(1|1)_R$-centroids exist.

Suppose that $\delta(CA(x)) \leq \pi$ for some point $x \in \mathbb{R}^2$. Let $Wedge(x)$ denote the *wedge* of x, defined as the intersection of the two half-planes $H(F(\theta_b|x), y(\theta_b|x))$ and $H(F(\theta_e|x), y(\theta_e|x))$, where θ_b and θ_e are the beginning and ending angles of $CA(x)$, respectively. $Wedge(x)$ consists of the two half-lines extending from x, defined by $F(\theta_e|x)$ and $F(\theta_b|x)$, and the infinite region lying between them. The counterclockwise (CCW) angle between the two half-lines is denoted by $\delta(Wedge(x))$. Since $0 < \delta(CA(x)) \leq \pi$, we have that $Wedge(x) \neq \emptyset$ and $0 \leq \delta(Wedge(x)) < \pi$.

It should be emphasized that $Wedge(x)$ is a computational byproduct of $CA(x)$ when x is not a strong $(1|1)_R$-centroid. In other words, not every point has its wedge. Therefore, we make the following assumption (or restriction) in order to avoid the misuse of $Wedge(x)$.

▶ **Assumption 8.** *Whenever $Wedge(x)$ is mentioned, the point x has been found not to be a strong $(1|1)_R$-centroid, either by computation or by properties. Equivalently, $\delta(CA(x)) \leq \pi$.*

The following lemma makes $Wedge(x)$ our main tool for prune-and-search. (Note that its proof is not trivial, since by definition θ_b and θ_e do not belong to $CA(x)$ and $MA(x)$.)

▶ **Lemma 9.** *Let $x \in \mathbb{R}^2$ be an arbitrary point. For any point $x' \notin Wedge(x)$, $W^*(x') \geq W^*(x)$.*

The computation of $Wedge(x)$ is simple. We first compute $W^*(x)$ in $O(n \log n)$ time by Theorem 2. Then, by reusing the sweeping technique, we can obtain $MA(x)$ and $CA(x)$ in $O(n)$ time and, if x is not a strong $(1|1)_R$-centroid, $Wedge(x)$ in $O(1)$ time.

▶ **Lemma 10.** *Given a point $x \in \mathbb{R}^2$, $MA(x)$, $CA(x)$, and $Wedge(x)$ can be computed in $O(n \log n)$ time.*

3.2 Searching on a Line

Although wedges can be used to prune candidate points, the performance is not stable, since wedges of different points have distinct angle intervals and spans. However, they work fine

with lines by Assumption 8. Here we show how to use the wedges to compute a *local optimal* point on a given line, i.e. a point x with $W^*(x) \leq W^*(x')$ for any point x' on the line.

Let L be an arbitrary line, which is assumed to be non-horizontal for ease of discussion. For any point x on L, we can compute $Wedge(x)$ and make use of it for pruning purposes by defining its *direction* with respect to L. Since $\delta(Wedge(x)) < \pi$ by definition, there are only three categories of directions according to the intersection of $Wedge(x)$ and L:
Upward – the intersection is the half-line of L above and including x;
Downward – the intersection is the half-line of L below and including x;
Sideward – the intersection is x itself.
If $Wedge(x)$ is sideward, x is a local optimal point on L, since by Lemma 9 $W^*(x) \leq W^*(x') \ \forall \ x' \in L$. Otherwise, either $Wedge(x)$ is upward or downward, the points on the opposite half of L can be pruned by Lemma 9. It shows that computing wedges acts as a predictable tool for pruning points on L.

Next, we list sets of *breakpoints* on L in which a local optimal point exists. Recall that $\mathcal{T}$ is the set of outer tangent lines of all pairs of circles in $\mathcal{C}(V)$. We define the *$\mathcal{T}$-breakpoints* as the set $L \times \mathcal{T}$ of intersection points between L and lines in $\mathcal{T}$, and the *$\mathcal{C}$-breakpoints* as the set $L \times \mathcal{C}(V)$ of intersection points between L and circles in $\mathcal{C}(V)$. (Note that outer tangent lines parallel to L can be ignored while defining breakpoints.) We have the following lemma for breakpoints.

▶ **Lemma 11.** *There exists a local optimal point x^*_L which is also a breakpoint.*

Since $|L \times \mathcal{T}| = O(n^2)$ and $|L \times \mathcal{C}(V)| = O(n)$, we can sort all breakpoints on L in $O(n^2 \log n)$ time according to the decreasing order of their y-coordinates, and, by Lemma 11, perform binary search via wedges to find a local optimal point x^*_L among them in $O(n \log n \times \log n) = O(n \log^2 n)$ time. Thus, the restricted problem is trivially solved in $O(n^2 \log n)$ time. In the following, we however give a more complicated algorithm to deal with the case that the line L is given as a query. The algorithm consists of an $O(n^2 \log n)$-time preprocessing and an $O(n \log^2 n)$-time procedure to find x^*_L on L.

The preprocessing itself is very simple. For each point $v \in V$, we compute a sequence $P(v)$, consisting of points in $V \backslash \{v\}$ sorted in increasing order of their polar angles with respect to v. The computation for all $v \in V$ takes $O(n^2 \log n)$ time in total. We will show that, for any given line L, $O(n)$ sorted sequences of breakpoints can be obtained from these pre-computed sequences in $O(n \log n)$ time, and can be used to replace the role of the sorted sequence of all breakpoints while performing binary search on L.

For any two points $v \in V$ and $z \in \mathbb{R}^2$, let $T^r(z|v)$ be the outer tangent line of $C_\gamma(v)$ and $C_\gamma(z)$ to the right of the line from v to z. Similarly, let $T^l(z|v)$ be the outer tangent line to the left. Moreover, let $t^r_L(z|v)$ and $t^l_L(z|v)$ be the points at which $T^r(z|v)$ and $T^l(z|v)$ intersect with L, respectively. We partition $\mathcal{T}$ into $O(n)$ sets $\mathcal{T}^r(v) = \{T^r(v_i|v)|v_i \in V \backslash \{v\}\}$ and $\mathcal{T}^l(v) = \{T^l(v_i|v)|v_i \in V \backslash \{v\}\}$ for $v \in V$, and for each set consider its corresponding $\mathcal{T}$-breakpoints independently.

We discuss the set of $\mathcal{T}$-breakpoints $L \times \mathcal{T}^r(v)$ first. Let v be an arbitrary point in V. By general position assumption, we can observe that, in some consecutive subsequences of $P(v)$, points v_i are listed in the same order as their corresponding breakpoints $t^r_L(v_i|v)$ in decreasing y-coordinates, whereas the order of points in other consecutive subsequences correspond to that of breakpoints in increasing y-coordinates. Thus, we can partition $P(v)$ into $O(1)$ consecutive subsequences to represent $L \times \mathcal{T}^r(v)$, as shown in the following.

▶ **Lemma 12.** *For each $v \in V$, we can construct $O(1)$ sequences of $\mathcal{T}$-breakpoints on L in $O(\log n)$ time, which satisfy the following statements:*

(a) *Each sequence is of length* $O(n)$.
(b) *Breakpoints in each sequence are sorted in decreasing y-coordinates.*
(c) *The union of breakpoints in all sequences form* $L \times \mathcal{T}^r(v)$.

By Lemma 12, the $O(1)$ sorted sequences can replace the role of $L \times \mathcal{T}^r(v)$. Symmetrically, we can also obtain a similar lemma constructing another $O(1)$ sorted sequences of breakpoints to replace $L \times \mathcal{T}^l(v)$. By applying such a construction to all $v \in V$, in $O(n \log n)$ time we can construct total $O(n)$ sorted sequences of length $O(n)$, whose union is equivalent to $L \times \mathcal{T}$. Moreover, since $|L \times \mathcal{C}(V)| = O(n)$, we can directly arrange them into a sorted sequence in $O(n \log n)$ time. Consequently, all breakpoints on L are partitioned into $N_0 = O(n)$ sequences, each of length $O(n)$ and sorted in decreasing y-coordinates.

The searching of x_L^* in the N_0 sorted sequences is done by parametric search technique for parallel binary searches, introduced in [2]. For each sorted sequence, we obtain its middle element, and associate it with a weight. Then, we compute the *weighted median* x of the N_0 middle elements [18]. Finally, we apply Lemma 10 on x, and prune breakpoints not in $Wedge(x)$ for every sequence. By proper weighting scheme (details omitted), the total number of breakpoints in all sequences will be reduced by a constant factor. By repeating the above process, we can find x_L^* in at most $O(\log n)$ iterations.

The running time is analyzed as follows. As discussed above, constructing the N_0 sorted sequences takes $O(n \log n)$ time. The pruning process requires at most $O(\log n)$ iterations. At each iteration, we compute the weighted median x in $O(N_0) = O(n)$ time by [18], and $Wedge(x)$ in $O(n \log n)$ time by Lemma 10. Finally, pruning every sequences takes $O(n)$ time. Thus, the total running time is $O(n \log n) + O(\log n) \times O(n \log n) = O(n \log^2 n)$ time.

▶ **Lemma 13.** *With an* $O(n^2 \log n)$*-time preprocessing, given an arbitrary line* L*, a local optimal point* x_L^* *on* L *can be computed in* $O(n \log^2 n)$ *time.*

4 $(1|1)_R$-Centroid on the Plane

In this section, we study the $(1|1)_R$-centroid problem and propose an improved algorithm of time complexity $O(n^2 \log n)$. This algorithm is as efficient as the best-so-far algorithm for the (1|1)-centroid problem in [12], but based on a completely different approach.

In Subsection 4.1, we extend the algorithm of Lemma 13 to develop a procedure allowing us to prune candidate points on the plane with respect to a given vertical line. Then, in Subsection 4.2, we show how to compute a $(1|1)_R$-centroid in $O(n^2 \log n)$ time based on this newly-developed pruning procedure.

4.1 Pruning with Respect to a Vertical Line

Let L be an arbitrary vertical line on the plane. We call the half-plane strictly to the left of L the *left plane* of L and the one strictly to its right the *right plane* of L. A sideward wedge of some point on L is said to be *rightward* (resp. *leftward*) if it intersects the right (resp. left) plane of L. We can observe that, if there is some point $x \in L$ such that $Wedge(x)$ is rightward, every point x' on the left plane of L can be pruned, since $W^*(x') \geq W^*(x)$ by Lemma 9. Similarly, if $Wedge(x)$ is leftward, points on the right plane of L can be pruned. Although the power of wedges is not fully exerted in this way, pruning via vertical lines and sideward wedges is superior than directly via wedges due to predictable pruning regions.

Therefore, in this subsection we describe how to design a procedure that enables us to prune either the left or the right plane of a given vertical line L. As mentioned above, the key point is the searching of sideward wedges on L. It is achieved by carrying out three

conditional phases. In the first phase, we try to find some proper breakpoints with sideward wedges. If failed, we pick some representative point in the second phase and check its wedge to determine whether or not sideward wedges exist. Finally, in case of their nonexistence, we show that their functional alternative can be computed, called the *pseudo wedge*, that still allows us to prune the left or right plane of L. In the following, we develop a series of lemmas to demonstrate the details of the three phases.

▶ **Lemma 14.** *Let x be an arbitrary point on L. If $Wedge(x)$ is either upward or downward, for any point $x' \in L \backslash Wedge(x)$, $Wedge(x')$ has the same direction as $Wedge(x)$.*

Following from this lemma, if there exist two arbitrary points x_1 and x_2 on L with their wedges downward and upward, respectively, we can derive that x_1 must be strictly above x_2, and that points with sideward wedges or even strong $(1|1)_R$-centroids can lie only between x_1 and x_2. Thus, we can find sideward wedges between some specified downward and upward wedges. Let x_D be the lowermost breakpoint on L with its wedge downward, x_U the uppermost breakpoint on L with its wedge upward, and G_{DU} the open segment $\overline{x_D x_U} \backslash \{x_D, x_U\}$. (For ease of discussion, we assume that both x_D and x_U exist on L, and show how to resolve this assumption later by constructing a bounding box.) Again, x_D is strictly above x_U. Also, we have the following corollary by their definitions.

▶ **Corollary 15.** *If there exist breakpoints in the segment G_{DU}, for any such breakpoint x, either x is a strong $(1|1)_R$-centroid or $Wedge(x)$ is sideward.*

Given x_D and x_U, the first phase can thus be done by checking whether there exist breakpoints in G_{DU} and picking any of them if exist. Supposing that the picked one is not a strong $(1|1)_R$-centroid, a sideward wedge is found by Corollary 15 and can be used for pruning. Notice that, when there are two or more such breakpoints, one may question whether their wedges are of the same direction, as different directions result in inconsistent pruning results. The following lemma answers the question in the positive.

▶ **Lemma 16.** *Let x_1, x_2 be two distinct points on L, where x_1 is strictly above x_2 and none of them is a strong $(1|1)_R$-centroid. If $Wedge(x_1)$ and $Wedge(x_2)$ are both sideward, they are either both rightward or both leftward.*

The second phase deals with the case that no breakpoint exists between x_D and x_U by determining the wedge direction of a representative point of all inner points in G_{DU}. The following lemma enables us to pick an arbitrary point in G_{DU} as the representative.

▶ **Lemma 17.** *When there is no breakpoint between x_D and x_U, any two distinct points x_1, x_2 in G_{DU} have the same wedge direction, if they are not strong $(1|1)_R$-centroids.*

We choose the bisector point x_B of x_D and x_U as the representative. If x_B is not a strong $(1|1)_R$-centroid and $Wedge(x_B)$ is sideward, the second phase finishes with a sideward wedge found. Otherwise, if $Wedge(x_B)$ is downward or upward, we can derive the following and have to invoke the third phase.

▶ **Lemma 18.** *If there is no breakpoint between x_D and x_U and $Wedge(x_B)$ is not sideward, there exist neither strong $(1|1)_R$-centroids nor points with sideward wedges on L.*

When L satisfies Lemma 18, it consists of only points with downward or upward wedges, and is said to be *non-leaning*. Obviously, our pruning strategy via sideward wedges could not apply to such non-leaning lines. The third phase overcomes this obstacle by constructing a functional alternative of sideward wedges, called the pseudo wedge, on either x_D or x_U, so that pruning with respect to L is still achievable. We start with auxiliary lemmas.

▶ **Lemma 19.** *If L is non-leaning, $W^*(x_D) \neq W^*(x_U)$.*

Let $W_1 = \max\{W^*(x_D), W^*(x_U)\}$. We are going to define the pseudo wedge on either x_U or x_D, depending on which one has the smaller weight loss. We consider first the case that $W^*(x_D) > W^*(x_U)$, and obtain the following.

▶ **Lemma 20.** *If L is non-leaning and $W^*(x_D) > W^*(x_U)$, there exists one angle θ for x_U, where $\pi \leq \theta \leq 2\pi$, such that $W(H(B(\theta|x_U), y(\theta|x_U))) \geq W_1$.*

Let θ_U be an arbitrary angle satisfying the conditions of Lemma 20. We apply the line $F(\theta_U|x_U)$ for trimming the region of $Wedge(x_U)$, so that a sideward wedge can be obtained. Let $PW(x_U)$, called the *pseudo wedge* of x_U, denote the intersection of $Wedge(x_U)$ and $H(F(\theta_U|x_U), y(\theta_U|x_U))$. Deriving from the three facts that $Wedge(x_U)$ is upward, $\delta(Wedge(x_U)) < \pi$, and $\pi \leq \theta_U \leq 2\pi$, we can observe that either $PW(x_U)$ is x_U itself, or it intersects only one of the right and left planes of L. In the two circumstances, $PW(x_U)$ is said to be *closed* or *sideward*, respectively. The pseudo wedge has similar functionality as wedges, as shown in the following corollary.

▶ **Corollary 21.** *For any point $x' \notin PW(x_U)$, $W^*(x') \geq W^*(x_U)$.*

By this lemma, if $PW(x_U)$ is found to be sideward, points on the opposite half-plane with respect to L can be pruned. If $PW(x_U)$ is closed, x_U becomes another kind of strong $(1|1)_R$-centroids, in the meaning that it is also an immediate solution to the $(1|1)_R$-centroid problem. Without confusion, we call x_U a *conditional* $(1|1)_R$*-centroid* in the latter case.

On the other hand, considering the opposite case that $W^*(x_D) < W^*(x_U)$, we can also obtain an angle θ_D and a pseudo wedge $PW(x_D)$ for x_D by symmetric arguments. Then, either $PW(x_D)$ is sideward and the opposite side of L can be pruned, or x_D itself is a conditional $(1|1)_R$-centroid. Thus, the third phase overcomes the obstacle of the nonexistence of sideward wedges.

Recall that the three phases of searching sideward wedges is based on the existence of x_D and x_U on L, which was not guaranteed before. Here we show that, by constructing appropriate border lines, we can guarantee the existence of x_D and x_U while searching between these border lines. The *bounding box* is defined as the smallest axis-aligned rectangle that encloses all circles in $\mathcal{C}(V)$. Clearly, any point x outside the box satisfies that $W^*(x) = W(V)$ and must not be a $(1|1)_R$-centroid. Thus, given a vertical line not intersecting the box, the half-plane to be pruned is trivially decided. Moreover, let T_{top} and T_{btm} be two arbitrary horizontal lines strictly above and below the box, respectively. We can obtain the following.

▶ **Lemma 22.** *Let L be an arbitrary vertical line intersecting the bounding box, and x'_D and x'_U denote its intersection points with T_{top} and T_{btm}, respectively. $Wedge(x'_D)$ is downward and $Wedge(x'_U)$ is upward.*

According to this lemma, by inserting T_{top} and T_{btm} into $\mathcal{T}$, the existence of x_D and x_U is enforced for any vertical line intersecting the bounding box. Besides, the insertion does not affect the correctness of all lemmas developed so far.

Summarizing the above discussion, the whole picture of our desired pruning procedure can be described as follows. In the beginning, we perform a preprocessing to obtain the bounding box and then add T_{top} and T_{btm} into $\mathcal{T}$. Now, given a vertical line L, whether to prune its left or right plane can be determined by the following steps.

1. If L does not intersect the bounding box, prune the half-plane not containing the box.
2. Compute x_D and x_U on L.

3. Find a sideward wedge or pseudo wedge via three forementioned phases. (Terminate whenever a strong or conditional $(1|1)_R$-centroid is found.)
   a. If breakpoints exist between x_D and x_U, pick any of them and check it.
   b. If no such breakpoint, decide whether L is non-leaning by checking x_B.
   c. If L is non-leaning, compute $PW(x_U)$ or $PW(x_D)$ depending on which of x_U and x_D has smaller weight loss.
4. Prune the right or left plane of L according to the direction of the sideward wedge or pseudo wedge.

The correctness of this procedure follows from the developed lemmas. Any vertical line not intersecting the bounding box is trivially dealt with in Step 1, due to the property of the box. When L intersects the box, by Lemma 22, x_D and x_U can certainly be found in Step 2. The three sub-steps of Step 3 correspond to the three searching phases. When L is not non-leaning, a sideward wedge is found, either at some breakpoint between x_D and x_U in Step 3(a) by Corollary 15, or at x_B in Step 3(b) by Lemma 17. Otherwise, according to Lemma 20 or its symmetric version, a pseudo wedge can be built in Step 3(c) for x_U or x_D, respectively. In Step 4, whether to prune the left or right plane of L can be determined via the just-found sideward wedge or pseudo wedge, by respectively Lemma 9 or Corollary 21.

The time complexity of this procedure is analyzed as follows. Computing the bounding box takes $O(n)$ time. In Step 2, x_D and x_U can be found by using the binary-search discussed in 3.2. Although the algorithm is not designed for this purpose, a slightly modification to its objective satisfies our need, and Step 2 can be done in $O(n \log^2 n)$ time by Lemma 13.

In Step 3(a), all breakpoints between x_D and x_U can be obtained in $O(n \log n)$ time as follows. As done in Lemma 13, we list all breakpoints on L as $O(n)$ sorted sequences, and prune breakpoints not in G_{DU} from each sequence by binary search. In Step 3(a) or 3(b), checking a picked point is done in $O(n \log n)$ time by invoking Lemma 10. The pseudo wedge $PW(x_U)$ or $PW(x_D)$ in Step 3(c) can be computed in $O(n \log n)$ time by using a sweeping technique to find the angle θ_U satisfying Lemma 20, or symmetrically θ_D, in $O(n \log n)$ time. Summarizing the above, these steps require $O(n \log^2 n)$ time in total. Since the invocation of Lemma 13 needs an additional $O(n^2 \log n)$-time preprocessing, we have the following result.

▶ **Lemma 23.** *With an $O(n^2 \log n)$-time preprocessing, whether to prune the right or left plane of a given vertical line L can be determined in $O(n \log^2 n)$ time.*

4.2 Searching on the Euclidean Plane

In this subsection, we come back to the $(1|1)_R$-centroid problem. Recall that, by Lemma 4, at least one $(1|1)_R$-centroid can be found in the three sets of intersection points $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{T}$, and $\mathcal{C}(V) \times \mathcal{C}(V)$, which consist of total $O(n^4)$ points. Let $\mathcal{L}$ denote the set of all vertical lines passing through these $O(n^4)$ intersection points. By definition, there exists a vertical line $L^* \in \mathcal{L}$ such that its local optimal point is a $(1|1)_R$-centroid. Conceptually, with the help of Lemma 23, L^* can be derived by applying prune-and-search approach to $\mathcal{L}$. However, it costs too much to explicitly generate and maintain the $O(n^4)$ lines. In the following, we show how to implicitly maintain these lines, by dealing with each of the above three sets separately, so that prune-and-search approaches can be applied.

Let $\mathcal{L}_\mathcal{T}$, $\mathcal{L}_\mathcal{M}$, and $\mathcal{L}_\mathcal{C}$ be the sets of all vertical lines passing through the intersection points in $\mathcal{T} \times \mathcal{T}$, $\mathcal{C}(V) \times \mathcal{T}$, and $\mathcal{C}(V) \times \mathcal{C}(V)$, respectively. A *local optimal line* of $\mathcal{L}_\mathcal{T}$ is a vertical line $L_t^* \in \mathcal{L}_\mathcal{T}$, such that its local optimal point has weight loss no larger than those of other lines in $\mathcal{L}_\mathcal{T}$. The local optimal lines L_m^* and L_c^* can be similarly defined for $\mathcal{L}_\mathcal{M}$ and $\mathcal{L}_\mathcal{C}$, respectively. We will adopt different prune-and-search techniques to find the

local optimal lines of the three sets, so that a $(1|1)_R$-centroid can be found on one of them. Since some of the algorithms are fairly complicated, due to page limit, we provide only the overview of our approaches in the following.

To deal with up to $O(n^4)$ vertical lines in $\mathcal{L}_\mathcal{T}$, we apply the ingenious idea of parametric search via parallel sorting algorithms, proposed by Megiddo [15]. In this approach, the process of pruning vertical lines in $\mathcal{L}_\mathcal{T}$ to find L_t^* is reduced to the problem of sorting the $O(n^2)$ lines of $\mathcal{T}$ according to their intersection points on the undetermined vertical line L_t^*, in which each comparison between two lines of $\mathcal{T}$ can be resolved by deciding whether L_t^* is to the right or left of their intersection point. Obviously, the decision can be done by invoking Lemma 23 on the vertical line passing through the point.

Given a batch of k such comparisons, Megiddo showed how to resolve them in $O(k+\tau \log k)$ time, where τ is the time required to resolve one comparison. Then, he found that executing parallel sorting algorithms in a sequential way serves as good batching schemes. For example, the parallel merge sort algorithm [3] sorts N_1 items in $O(\log N_1)$ parallel steps on $O(N_1)$ processors. Executing this algorithm sequentially forms a sorting framework that takes $O(\log N_1)$ iterations, in each of which a batch of $k = O(N_1)$ comparisons has to be resolved. Thus, by letting $N_1 = |\mathcal{T}|$ and $\tau = O(n \log^2 n)$, our sorting problem can be solved in $O((k + \tau \log k) \times \log N_1) = O(n^2 \log n)$ time.

▶ **Lemma 24.** *A local optimal line L_t^* of $\mathcal{L}_\mathcal{T}$ can be found in $O(n^2 \log n)$ time.*

By similar observation as made in Lemma 12, for any two points $u, v \in V$, $C_\gamma(u) \times \mathcal{T}^r(v)$ and $C_\gamma(u) \times \mathcal{T}^l(v)$ can be represented by $O(1)$ consecutive subsequences of $P(v)$ in $O(\log n)$ time. Thus, for $\mathcal{C}(V) \times \mathcal{T}$ we can construct in $O(n^2 \log n)$ time $O(n^2)$ sequences of $O(n)$ breakpoints, each sorted in increasing x-coordinates. Correspondingly, $\mathcal{L}_\mathcal{M}$ can be represented by $O(n^2)$ sorted sequences of vertical lines. Then, finding L_m^* can be done by applying prune-and-search to the $O(n^2)$ sequences of vertical lines via parallel binary searches, like in Lemma 13, which takes $O(\log n)$ iterations and $O(n^2 + n \log^2 n)$ time per iteration.

▶ **Lemma 25.** *A local optimal line L_m^* of $\mathcal{L}_\mathcal{M}$ can be found in $O(n^2 \log n)$ time.*

Since $|\mathcal{C}(V) \times \mathcal{C}(V)| = O(n^2)$, a sorted sequence of $\mathcal{L}_\mathcal{C}$ can be obtained in $O(n^2 \log n)$ time. Then, L_c^* can be easily found by binary search with Lemma 23 in $O(n \log^3 n)$ time.

▶ **Lemma 26.** *A local optimal line L_c^* of $\mathcal{L}_\mathcal{C}$ can be found in $O(n^2 \log n)$ time.*

By definition, L^* can be found among L_t^*, L_m^*, and L_c^*, which can be computed in $O(n^2 \log n)$ time by Lemmas 24, 25, and 26, respectively. Then, a $(1|1)_R$-centroid can be computed as the local optimal point of L^* in $O(n \log^2 n)$ time by Lemma 13. Combining with the $O(n^2 \log n)$-time preprocessing for computing the angular sorted sequence $P(v)$s and the bounding box enclosing $\mathcal{C}(V)$, we have the following theorem.

▶ **Theorem 27.** *The $(1|1)_R$-centroid problem can be solved in $O(n^2 \log n)$ time.*

5 Concluding Remarks

In this paper, we revisited the $(1|1)$-centroid problem on the Euclidean plane under the consideration of minimal distance constraint between facilities, and proposed an $O(n^2 \log n)$-time algorithm, which closes the bound gap between this problem and its unconstrained version. Starting from a critical observation on the medianoid solutions, we developed a pruning tool with indefinite region remained after pruning, and made use of it via multi-level structured parametric search approach, which is different to the previous approach in [6, 12].

Considering distance constraint between facilities in various competitive facility location models is both of theoretical interest and of practical importance. However, similar constraints are rarely seen in the literature. It would be good starting points by introducing the constraint to the facilities between players in the $(r|X_p)$-medianoid and $(r|p)$-centroid problems, maybe even to the facilities between the same player.

References

1 Aritra Banik, Jean-Lou De Carufel, Anil Maheshwari, and Michiel Smid. Discrete voronoi games and ε-nets, in two and three dimensions. *Computational Geometry*, 55:41–58, 2016.

2 Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, January 1987.

3 Richard Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, 1988.

4 Abdullah Dasci. Conditional location problems on networks and in the plane. In Horst A. Eiselt and Vladimir Marianov, editors, *Foundations of Location Analysis*, pages 179–206. Springer US, Boston, MA, 2011.

5 Ivan Davydov, Yury Kochetov, and Alexandr Plyasunov. On the complexity of the $(r|p)$-centroid problem in the plane. *TOP*, 22(2):614–623, 2014.

6 Zvi Drezner. Competitive location strategies for two facilities. *Regional Science and Urban Economics*, 12(4):485–493, 1982.

7 Zvi Drezner and E. Zemel. Competitive location in the plane. *Annals of Operations Research*, 40(1):173–193, 1992.

8 Horst A. Eiselt and Gilbert Laporte. Sequential location problems. *European Journal of Operational Research*, 96(2):217–231, 1997.

9 Horst A. Eiselt, Gilbert Laporte, and Jacques-Francois Thisse. Competitive location models: a framework and bibliography. *Transportation Science*, 27(1):44–54, 1993.

10 Horst A. Eiselt, Vladimir Marianov, and Tammy Drezner. Competitive location models. In Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama, editors, *Location Science*, pages 365–398. Springer International Publishing, Cham, 2015.

11 S. Louis Hakimi. On locating new facilities in a competitive environment. *European Journal of Operational Research*, 12(1):29–35, 1983.

12 S. Louis Hakimi. Locations with spatial interactions: competitive locations and games. In Pitu B. Mirchandani and Richard L. Francis, editors, *Discrete location theory*, pages 439–478. Wiley, 1990.

13 Harold Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929.

14 D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1(1):193–211, 1986.

15 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, October 1983.

16 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

17 Frank Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

18 Angelika Reiser. A linear selection algorithm for sets of elements with weights. *Information Processing Letters*, 7(3):159–162, 1978.

19 D. R. Santos-Peñate, R. Suárez-Vega, and P. Dorta-González. The leader–follower location model. *Networks and Spatial Economics*, 7(1):45–61, 2007.