Assigning Weights to Minimize the Covering Radius in the Plane*

Eunjin Oh[1] and Hee-Kap Ahn[2]

1 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
jin9082@postech.ac.kr

2 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
heekap@postech.ac.kr

Abstract

Given a set P of n points in the plane and a multiset W of k weights with $k \leq n$, we assign a weight in W to a point in P to minimize the maximum weighted distance from the weighted center of P to any point in P. In this paper, we give two algorithms which take $O(k^2n^2\log^4 n)$ time and $O(k^5 n\log^4 k + kn\log^3 n)$ time, respectively. For a constant k, the second algorithm takes only $O(n\log^3 n)$ time, which is near-linear.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Weighted center, facility location, weight assignment, combinatorial optimization, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.58

1 Introduction

Consider a set of robots lying at different locations in the plane. Each robot is equipped with a locomotion module so that it can move to a nearby facility to recharge its battery and return to its original location. We want to place a recharging facility for the robots such that the maximum travel time of them to reach the recharging facility is minimized. Then the Euclidean center of the robot locations may not be a good location for the recharging facility if the robots have huge differences in their speeds. For instance, consider three robots, each lying on a different corner of an equilateral triangle. If one of them has a much smaller speed compared to the speeds of the other two robots, the best location is very close to the corner where the low-speed robot lies. Hence, the recharging facility must be located at a *weighted center* of the robots by considering their speeds as weights of their placements.

In the weighted center problem, each input point $p \in P$ is associated with a positive weight and the *weighted distance* between an input point and a point of the plane is defined to be their distance divided by the associated weight of p. Then the point of the plane that minimizes the maximum weighted distance to input points is the center of the weighted input points, which we call the *weighted center*.

Dyer [8] studied the weighted center problem for a set of weighted points in the plane and gave a linear-time algorithm to compute their weighted center. Clearly, the weighted center coincides with the (unweighted) center if the associate weight is 1 for every input point.

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

Imagine now that we are allowed to *reassign* the locomotion modules of the robots. Or, if the mobile robots are identical, except their speeds, we are allowed to *relocate* the robots. A relocation of robots (or a reassignment of locomotion modules) may change the weighted center and the maximum travel time for mobile robots to reach the weighted center. In other words, a clever assignment of robots (or their locomotion modules) to given locations may decrease the minimum of the objective function.

In this paper, we formally define this relocation problem and present algorithms for it. The *weight assignment problem* is defined as follows: given an input consisting of a set P of n points in the plane and a multiset $W = \{w_1, \ldots, w_k\}$ of k weights of positive real values with $k \leq n$, find an assignment of the weights in W to input points such that the maximum weighted distance from the weighted center to input points is minimized. We assume that every input point of P has the default weight 1. We assign the k weights to k points of P such that every weight is assigned to a point and each point gets one weight in W or the default weight 1, which we call an *assignment of weights* of W to P.

We regard an assignment of weights as a function. To be specific, for an assignment f of weights, $f(p)$ denotes the weight of W assigned to point $p \in P$. We use $c(f)$ to denote the weighted center of P with the assignment f, and call the maximum weighted distance from $c(f)$ to input points the *covering radius* of the assignment f and denote it by $r(f)$.

Obviously, there are $\binom{n}{k}$ different combinations of selecting k points from P and $k!$ different ways of assigning the k weights to a combination, and therefore there are $\Theta(n^k)$ different assignments of weights. Our goal is to find an assignment f of weights of W to the points of P that minimizes the covering radius $r(f)$ over all possible assignments of weights.

Related Work. As mentioned earlier, Dyer [8] studied the weighted center problem for a set of weighted points in the plane. He reformulated the problem as an optimization problem with linear inequalities and one quadratic inequality. Then he gave a linear-time algorithm to compute their weighted center using the technique by Megiddo [12]. Later, Megiddo [13] gave a linear-time algorithm for the same problem using a different technique.

In contrast, to our best knowledge, no algorithm is known for the weight assignment problem while there are works on several related problems. In the *inverse 1-center problem* on graphs, we are given a graph and a target vertex, and we are to increase or decrease the lengths of edges of the graph so that the target vertex becomes a center of the modified graph. The goal is to minimize the modification of the lengths. This means that we give additive weights to edges of the graph. Cai et al. [6] showed that this problem is NP-hard on a general directed graph. Recently, Alizadeh and Burkard [3] gave an $O(n^2 r)$-time algorithm for this problem on a tree, where r is the compressed depth of the tree. A variant of this problem is the *reverse 1-center problem*, in which we are to decrease the lengths of edges of the graph under a given budget. This problem is also known to be NP-hard even on a bipartite graph [5], and there is an $O(n^2 \log n)$-time algorithm on a tree by Zhang et al. [15].

Our weight assignment problem is closely related to the weight balancing problem which was studied by Barba et al. [4]. The input consists of a simple polygon, a target point inside the polygon, and a set of weights. The goal is to put the weights on the boundary of the polygon so that the barycenter (center of mass) of the weights coincides with the target point. They showed the existence of such a placement of weights under the condition that no input weight exceeds the sum of the other input weights. They also gave an algorithm to find such a placement in $O(k + n \log n)$ time, where k is the number of the weights and n is the number of the vertices of P. Our problem can be considered as a discrete version of this problem, but with a different criteria (minimizing the covering radius), in the sense that we place the weights on predetermined positions.

Our Result. In this paper, we present two algorithms that compute an assignment f of weights minimizing $r(f)$. The first algorithm returns an optimal assignment in $O(k^2n^2\log^4 n)$ time using $O(kn)$ space. The second algorithm assumes that all weights in W are at most 1, and returns an optimal assignment in $O(k^5n\log^4 k + kn\log^3 n)$ time using $O(kn)$ space, which is faster than the first algorithm when k is sufficiently small ($k = o(n^{1/3})$). Moreover, it takes only $O(n\log^3 n)$ time when k is a constant.

Another merit of our algorithms is that they are based on useful geometric intuition and are easy to implement though they use parametric search, the optimization technique developed by Megiddo [10]. The technique is an important tool for solving many geometric optimization problems efficiently, but algorithms based on it are often not easily implemented [2]. A main difficulty lies in computing the roots of the polynomials exactly whose signs determine the outcome of the comparisons made by the algorithm. However, as we will see later, in our algorithms, such a root is a covering radius of at most three weighted points and it can be computed without resorting to complicated methods.

2 Preliminaries

One way to deal with our problem is to find the weighted centers for all possible assignments of weights and choose the one with the minimum radius. Although there are $\Theta(n^k)$ different assignments of weights, there are only $O(k^3n^3)$ different weighted centers. This is because a weighted center is determined by at most three weighted points. That is, given an assignment of weights, there always exist at most three weighted points of P whose center coincides with the center of the whole weighted points. Thus the number of all possible weighted centers is at most the number of all possible 6-tuples $(p_1, p_2, p_3, w_1, w_2, w_3)$ such that $p_i \in P$ and $w_i \in W \cup \{1\}$ for $i = 1, 2, 3$, which is $O(k^3n^3)$. Moreover, this bound is asymptotically tight by the following lemma. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 1.** *There exist a set of n points and a set of k weights such that the number of all possible weighted centers is $\Omega(k^3n^3)$.*

2.1 Deciding Feasibility for a Weighted Center of Three Points

As noted above, the weighted center and the covering radius of a weight assignment are determined by at most three weighted points. But not every three weighted points define such a center and its covering radius. Here we show how to test this for three weighted points efficiently.

Let W' be the union of W and the multiset consisting of $n - k - 3$ numbers of weight 1. Consider three points from P and an assignment of three weights from W' to the points. We first test whether the three weighted points define a point in the plane at the same weighted distance from them. If such a point does not exist, there is no weighted center of the three weighted points. Otherwise, there is only one such point which is the weighted center of them. Let c denote the weighted center and r denote the covering radius of the three weighted points. Let $\langle p_1, \ldots, p_{n-3}\rangle$ be the sequence of the points in P, except the three points, in the increasing order with respect to the Euclidean distance from c. Let $\langle w_1, \ldots, w_{n-3}\rangle$ be the sequence of the weights in W', except the three weights, in the increasing order, The following lemma directly gives us an $O(n)$-time algorithm to decide whether the three weighted points determine the weighted center and the covering radius of an assignment of weights. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 2.** *There exists an assignment f of weights such that $c(f) = c$ and $r(f) = r$ if and only if $d(p_i, c)/w_i \le r$ for $1 \le i \le n - 3$.*

It is possible that two weighted points p and q determine the weighted center of the whole weighted points. In this case, the weighted center lies in the line passing through p and q. To handle this case, we consider two points from P and an assignment of two weights from W to the points. We compute the weighted center and decide whether the two weighted points determine the weighted center and the covering radius using Lemma 2.

We need to sort the points in P repeatedly for each weighted center determined by a combination of three points (or two points) from P and three weights (or two weights) from W while it suffices to sort the weights in W just once. Thus, the total running time is $O(k^3 n^4 \log n)$.

Note that this algorithm returns all possible weighted centers. Thus it can be used for the problem with some different optimization criteria other than the minimization of the covering radius. For example, we can find the assignment f of weights such that the center $c(f)$ is the closest to a given point in the same time.

3 A Fast Algorithm using $O(kn)$ Space

In this section, we give an $O(k^2 n^2 \log^4 n)$-time algorithm for finding an assignment f of weights that minimizes $r(f)$. This algorithm does not consider all possible weighted centers. Instead, it uses parametric search due to Megiddo [10]. To apply this technique, we need to devise a decision algorithm which is used as a subprocedure of the main algorithm.

3.1 A Decision Algorithm

Let r be an input of the decision algorithm. The decision algorithm decides whether there is an assignment f of weights with $r(f) \le r$. In other words, it decides whether there are a point c and an assignment f of weights such that $d(p, c)/f(p) \le r$ for all points $p \in P$. If this is the case, we call such a point c an *r-center* with respect to f.

Instead of considering all $\Theta(k^3 n^3)$ combinations of three points in P and three weights in W, this algorithm considers all $O(kn)$ point-weight pairs. Our algorithm is based on the following lemma. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 3.** *For an assignment f of weights with $r(f) \le r$, there is an r-center c with respect to f satisfying $d(p', c)/f(p') = r$ for some point $p' \in P$.*

By the above lemma, there is a point $p \in P$ and a weight $w \in W \cup \{1\}$ such that the circle centered at p with radius wr contains an r-center c with respect to some assignment f of weights if $r(f) \le r$. Thus our strategy is to find an r-center, if it exists, lying on such a circle with respect to some weight assignment f satisfying that $f(p) = w$.

For each pair of a point $p \in P$ and a weight $w \in W \cup \{1\}$, we consider the circle C centered at p with radius wr. For a point $c \in C$, we check whether there exists an assignment f of weights such that the disk centered at q with radius $f(q)r$ contains c for all points $q \in P \setminus \{p\}$.

To this end, for each point $q \in P \setminus \{p\}$, we compute $k+1$ concentric circles centered at q with radius $w_1 r, w_2 r, \ldots, w_{k+1} r$, where $w_1, w_2, \ldots, w_{k+1}$ are weights in $W \cup \{1\}$. Then we compute the intersections of these circles with C and sort them along C in $O(kn \log(kn)) = O(kn \log n)$ time. (In a degenerate case, there can be more than one circle passing through the same

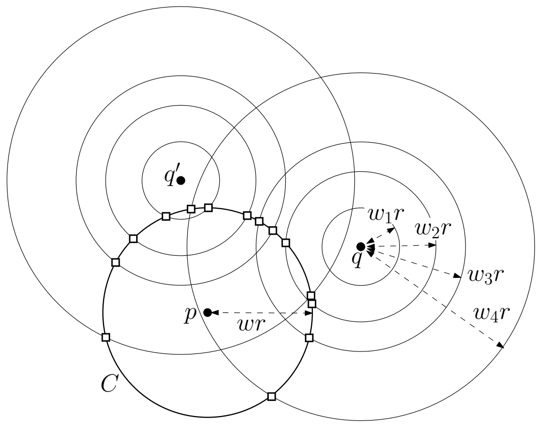

Figure 1 We compute the points marked with squares for all points in $q \in P \setminus \{p\}$ and sort them along C.

intersection point and we treat their intersection points as distinct points lying in the same position. Details can be found in the full version of this paper. In the following, we assume that exactly one circle passes through one intersection point.) See Figure 1 for an illustration.

Now, we have $O(kn)$ intersection points sorted along C. These intersection points subdivide the circle C into $O(kn)$ pieces, which we call *intervals* on C. We say an assignment f of weights is *feasible* for an interval if for a point c in the interval, $d(q,c)/f(q)$ is at most r for all points $q \in P$. Note that f is feasible for any point in the interval if f is feasible for a point in the interval. Therefore, for any point c lying in an interval, the set of feasible assignments of weights remains the same. We can test in $O(n \log n)$ time whether there exists a feasible assignment for an interval on C by slightly modifying Lemma 2.

Instead of applying this test repeatedly for each interval on C, which takes $O(kn^2 \log n)$ time in total, we can do this in $O(kn \log n)$ time in total for all intervals on C as follows. Consider the intervals one by one in clockwise order along C. Note that for any two consecutive intervals, there is only one disk centered at a point in P with radius rw for some $w \in W$ that contains one interval but does not contain the other interval. We use this fact in the following lemma.

▶ **Lemma 4.** *We can decide whether there is a feasible assignment of weights or not for every interval on C in $O(kn \log n)$ time in total.*

Proof. We first show how to check the existence of a feasible assignment for an interval μ in $O(kn)$ time. Then we show how we do this for all intervals on C efficiently.

We sort the weights in $W \cup \{1\}$ in the increasing order and denote the sorted list by $\langle w_1, \ldots, w_{k+1} \rangle$. Let ℓ_0 be the smallest index with $w_{\ell_0} = 1$, and c be a point in μ. For each point q in $P \setminus \{p\}$, let $\pi(q)$ be the smallest index such that $d(q,c)/w_{\pi(q)} \le r$ and let $\pi(p)$ be the index indicating w. The indices $\pi(q)$ for all points q in P can be computed in $O(kn)$ time in total. Then, we sort the points in P in the increasing order with respect to $\pi(\cdot)$ and denote the sorted list by $\langle q_1, \ldots, q_n \rangle$.

Consider the simple case that $k = n$. Then there exists a feasible assignment for μ if and only if $\pi(q_\ell) \le \ell$ for all indices $1 \le \ell \le n-1$. This is because for every point q in P, we have $d(q,c)/w_j \le r$ for all indices $j \ge \pi(q)$. Thus, we can check the existence of a feasible assignment for μ by comparing $\pi(q_\ell)$ and ℓ for all indices $1 \le \ell \le n$, which can be done in $O(n)$ time.

For the case that $k < n$, a similar property holds: there exists a feasible assignment for μ if and only if

- $\pi(q_\ell) \leq \ell$ for all indices $1 \leq \ell \leq \ell_0 - 1$,
- $\pi(q_\ell) \leq \ell_0$ for all indices $\ell_0 \leq \ell \leq \ell_0 + n - k - 1$, and
- $\pi(q_\ell) \leq \ell - n + k + 1$ for all indices $\ell_0 + n - k \leq \ell \leq n$.

Thus, we can check the existence of a feasible assignment for μ in $O(kn)$ time.

To check the existence of a feasible assignment for the interval μ' next to μ in clockwise order along C, we do not need to compute all such indices and compare them again. Recall that there is exactly one disk C' centered at a point $q \in P \setminus \{p\}$ with radius rw for some $w \in W$ that contains either μ or μ'. If C' contains μ but does not contain μ', $\pi(q)$ increases by one. If C' contains μ' but does not contain μ, $\pi(q)$ decreases by one. Note that $\pi(q_\ell)$ remains the same for all points $q_\ell \in P \setminus \{q\}$ for both cases.

To use this property, we maintain $2(k+1)$ pointers $U_1, \ldots, U_{k+1}$ and $L_1, \ldots, L_{k+1}$. For an index $1 \leq i \leq k+1$, the pointer U_i points to q_ℓ with the largest index ℓ among the points such that $\pi(q_\ell)$ is equal to i. Similarly, for an index $1 \leq i \leq k+1$, the pointer L_i points to q_ℓ with the smallest index ℓ among the points such that $\pi(q_\ell)$ is equal to i.

Note that we already know whether $\pi(q)$ increases or decreases by one when we move from μ to μ'. Here, we have to update not only $\pi(q)$ but also the pointers. Moreover, we have to reorder $\langle q_1, \ldots, q_n \rangle$ in the increasing order with respect to $\pi(\cdot)$.

We show how to do this for the case that $\pi(q)$ increases by one. The other case can be handled analogously. We first find the point q_u that the pointer $U_{\pi(q)}$ points to. Then we swap the positions for q and q_u on the sequence $\langle q_1, \ldots, q_n \rangle$ and let $U_{\pi(q)}$ point to q. This does not violate the property that $\pi(q_\ell) \leq \pi(q_{\ell+1})$ for all indices $1 \leq \ell < n$ since $\pi(q) = \pi(q_u)$. Then we update $\pi(q)$ to $\pi(q) + 1$ and update the pointers accordingly.

Here, we do not need to compare $\pi(q_\ell)$ and the index again for a point q_ℓ in $P \setminus \{q, q_u\}$. Thus to check the existence of a feasible assignment for μ', it suffices to compare $\pi(q_\ell)$ and the index for $q_\ell = q, q_u$. This can be done in constant time, which implies that each update can be done in the same time. Since there are $O(kn)$ intervals on C, updating the information takes $O(kn)$ time. Therefore, the running time of the procedure is dominated by the time for sorting the intersection points along C and computing the intervals, which is $O(kn \log n)$. ◀

With the argument in this section, the following lemma holds.

▶ **Lemma 5.** *Given a radius $r > 0$, we can decide in $O(k^2 n^2 \log n)$ time using $O(kn)$ space whether there exists an assignment f of weights with $r(f) \leq r$ or not.*

3.2 An Overall Algorithm

For ease of presentation, we first show how to compute an optimal solution in $O(k^2 n^2 \log^3 n)$ time using $O(k^2 n^2)$ space. At the end of this section, we show how to reduce the space into $O(kn)$ at the expense of increased time complexity by an $O(\log n)$ factor.

To obtain an optimal solution, we apply parametric search [10] using the decision algorithm in Section 3.1. Let r^* be the minimum of $r(f)$ over all possible assignments f of weights.

We consider the arrangement $A(r)$ of the circles $C_{p,w}(r)$ for all point-weight pairs (p, w), where $C_{p,w}(r)$ is the circle centered at p with radius rw. Let $\mathcal{C}(r)$ be the set of such circles $C_{p,w}(r)$ over all point-weight pairs (p, w). Here, $r > 0$ is a variable.

As r becomes larger, the combinatorial structure of $A(r)$ changes. To be specific, the combinatorial structure of $A(r)$ changes $O(k^3 n^3)$ times. To see this, we observe that there exists a vertex of $A(r)$ which is an intersection of three circles $C_{p_i,w_i}(r)$ (or two circles) in $\mathcal{C}(r)$ for $i = 1, 2, 3$ when $A(r)$ changes. Moreover, for any three point-weight pairs (p_i, w_i) for

$i = 1, 2, 3$ (or $i = 1, 2$), there exist at most two radii r such that the common intersection of $C_{p_i,w_i}(r)$ is not empty. This is because the trajectory of the intersections between two increasing circles forms a hyperbolic curve, and two hyperbolic curves cross at most twice. (Note that C_{p_i,w_i} is a circle, not a disk.)

These radii partition the real value space $\mathbb{R}$ into intervals such that for any value r in the same interval the combinatorial structure of $A(r)$ remains the same. We search the interval where r^* lies using the decision algorithm in Section 3.1. There are $\Theta(k^3n^3)$ such radii, but we do not consider all of them. Instead, we search for the interval containing r^* in $O(\log n)$ iterations. In each iteration, we consider $O(k^2n^2)$ radii and reduce the search space (intervals). Moreover, in each iteration, we apply the decision algorithm $O(\log n)$ times, which leads to the running time of $O(k^2n^2 \log^3 n)$. Details are described in the following lemma and its proof.

▶ **Lemma 6.** *The combinatorial structure of $A(r^*)$ can be computed in $O(k^2n^2 \log^3 n)$ time using (k^2n^2) space.*

Proof. Given a radius $r > 0$, we first introduce a simple way to compute the arrangement $A(r)$. Later, this will be used for computing $A(r^*)$ efficiently. For a point $p \in P$ and a weight $w \in W \cup \{1\}$, we compute the intersections of $C_{p,w}(r)$ and $C_{p_i,w_i}(r)$ for all points p_i in $P \setminus \{p\}$ and all weights $w_i \in W \setminus \{w\} \cup \{1\}$. Each circle $C_{p_i,w_i}(r)$ intersects $C_{p,w}(r)$ at most twice for any fixed radius $r > 0$. Let $\mathcal{I}_{p,w}(r)$ be the set of all such intersection points. We sort the points in $\mathcal{I}_{p,w}(r)$ along $C_{p,w}(r)$. Once this is done for all points p and all weights $w \in W \cup \{1\}$, we can construct $A(r)$.

Here, computing the combinatorial structure of $A(r)$ is equivalent to sorting the points in $\mathcal{I}_{p,w}(r)$ along $C_{p,w}(r)$ for all point-weight pairs (p, w). We apply this procedure to compute the combinatorial structure of $A(r^*)$ without computing r^* explicitly. To sort the points in $\mathcal{I}_{p,w}(r^*)$ along $C_{p,w}(r^*)$, we compare the relative positions for two points in the set $O(kn \log n)$ times since the number of points in the set is $O(kn)$.

Suppose that we want to compare the relative positions for two points $u_1(r^*), u_2(r^*)$ in the set along $C_{p,w}(r)$. As r increases, the relative positions between $u_1(r)$ and $u_2(r)$ change at most once. Moreover, they change when $C_{p,w}(r)$ and the two circles defining $u_1(r)$ and $u_2(r)$ meet at exactly one point. This happens at most twice. Let r_1 and r_2 be such two radii with $r_1 \le r_2$. Then we decide whether $r^* \le r_1$, $r_1 \le r^* \le r_2$, or $r_2 \le r^*$ using the decision algorithm in Section 3.1. With this, we can decide the relative positions between $u_1(r^*)$ and $u_2(r^*)$ without computing r^* explicitly. Thus, we can compare two points in $\mathcal{I}_{p,w}(r^*)$ in $O(k^2n^2 \log n)$ time. Since we do this $O(kn \log n)$ times, we can sort the points in $O(k^3n^3 \log^2 n)$ time. Since there are $O(kn)$ point-weight pairs (p, w), the total running time is $O(k^4n^4 \log^2 n)$.

Here, we apply the decision algorithm in Section 3.1 twice for each comparison, once with $r = r_1$ and once with $r = r_2$. We reduce the running time of the overall algorithm by reducing the number of executions of the decision algorithm. Suppose that we want to do m comparisons which are independent to each other. As we did in the previous procedure, we compute at most two radii from each comparison where the relative positions of the two points change. Then we have at most $2m$ radii. We sort them and apply binary search to compute the smallest interval containing r^*. After applying the decision algorithm $O(\log m)$ times, we can complete m comparisons.

In our problem, comparisons performed on two different circles are independent to each other. In each circle $C_{p,w}(r)$, we have $O(\log n)$ sets each of which consists of $O(kn)$ comparisons that are independent to each other. Indeed, Cole [7] gave a parallel algorithm to sort m elements in $O(\log m)$ time using $O(m)$ processors. Note that comparisons performed in different processors are independent to each other.

Thus, we sort the points in $\mathcal{I}_{p,w}(r^*)$ in $O(\log(kn)) = O(\log n)$ iterations and in each iteration we apply $O(kn)$ comparisons. We compute the whole combinatorial structure of $A(r^*)$ in $O(\log n)$ iterations and in each iteration we apply $O(k^2n^2)$ comparisons. This can be done in $O(T(k,n)\log^2 n + k^2n^2\log^2 n)$ time, where $T(k,n)$ is the running time of the decision algorithm. ◀

Now, we have the combinatorial structure of $A(r^*)$ while r^* is not known yet. The following lemma gives us a procedure to compute r^* in $O(k^2n^2\log^2 n)$ time.

▶ **Lemma 7.** *Given the combinatorial structure of $A(r^*)$, an optimal weight assignment and its covering radius r^* can be computed in $O(k^2n^2\log^2 n)$ time.*

Proof. In this proof, we use the notation defined in Lemma 6. We say that three circles $C_{p_i,w_i}(r)$ for $i = 1, 2, 3$ *define* a radius r' if they intersect at one point for $r = r'$. We already showed that there are at most three circles $C_{p_i,w_i}(r)$ for $i = 1, 2, 3$ that define r^*. Thus, in the set $\mathcal{I}_{p_1,w_1}(r^*)$ sorted along $C_{p_1,w_1}(r^*)$, a point corresponding to $C_{p_2,w_2}(r^*)$ and a point corresponding to $C_{p_3,w_3}(r^*)$ are consecutive. Note that when we sort the points in $\mathcal{I}_{p_1,w_1}(r^*)$, we cannot decide whether two points coincide or not. Instead, we give an arbitrary order for such points.

Let R be the set of radii r defined by three circles $C_{p_i,w_i}(r^*)$ for $i = 1, 2, 3$ such that the point corresponding to $C_{p_2,w_2}(r^*)$ and the point corresponding to $C_{p_3,w_3}(r^*)$ are consecutive in the set $\mathcal{I}_{p_1,w_1}(r^*)$. Since there are $O(k^2n^2)$ edges in the arrangement $A(r^*)$, there are the same number of such radii.

We sort the radii on R and apply binary search on it using the decision algorithm in Section 3.1 and find the smallest radius of R for which the decision algorithm returns "yes". Then the smallest radius is r^*. ◀

Since we maintain the whole combinatorial structure of the arrangement $A(r)$, we use $O(k^2n^2)$ space in the previous algorithm. We can reduce the space complexity to $O(kn)$ by computing only some partial information about the combinatorial structure of $A(r)$. However, this increases the running time of the algorithm by an $O(\log n)$ factor.

▶ **Lemma 8.** *We can compute an interval containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval in $O(k^2n^2\log^4 n)$ time using $O(kn)$ space.*

Proof. Here, we modify the algorithm in Lemma 6 as follows. Recall that in each iteration of the previous algorithm, we consider all point-weight pairs (p, w) and perform $O(kn)$ comparisons for sorting the intersection points in each $\mathcal{I}_{p,w}$. In this algorithm, we do this in $O(\log n)$ subiterations of an iteration as follows.

We maintain an interval u containing r^*. Initially, u is set to $[-\infty, +\infty]$. As we apply the algorithm, the interval becomes a smaller subinterval. In each subiteration, we consider $O(kn)$ radii corresponding to each comparison for a point-weight pair (p, w) and discard the radii which lie outside the interval u. Then we choose the median of them. We do this for all point-weight pairs (p, w), and we have $O(kn)$ medians in total. Then we sort the medians and apply binary search to compute the interval between two consecutive medians containing r^* in $O(T(n,k)\log n)$ time, where $T(n,k)$ is the running time of the decision algorithm. In total, each subiteration takes $O(T(n,k)\log n + k^2n^2)$ time using $O(kn)$ space.

Now, we show that in $O(\log n)$ subiterations, we complete $O(k^2n^2)$ comparisons. In the ith subiteration, we discard $(1/2 + 1/4 + \ldots + 1/2^i)|R|$ radii in total, where $|R| = O(k^2n^2)$ is the number of radii. Note that we discard the radius corresponding to some comparison once we complete the comparison. In $O(\log n)$ subiterations, we discard all radii, which means that we complete $O(k^2n^2)$ comparisons.

In this algorithm, each iteration takes $O(T(n,k)\log^2 n)$ time. Since we have $O(\log n)$ iterations in total as the algorithm in Lemma 6, the running time of this algorithm is $O(T(n,k)\log^3 n)$. ◀

Once we have an interval u containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval, the procedure described in the proof of Lemma 7 can also be improved.

▶ **Lemma 9.** *Given an interval containing r^* such that the combinatorial structure of $A(r)$ remains the same for any r in the interval, an optimal assignment and its covering radius r^* can be found in $O(k^2n^2\log n)$ time.*

Proof. We consider a point-weight pair (p,w) first. Instead of computing the whole arrangement, we compute the edges and the vertices lying on $C_{p,w}(r^*)$. This takes $O(kn\log n)$ time because we already have the smallest interval u containing r^*. Then we apply the algorithm in Lemma 7 on the edges and vertices lying on $C_{p,w}(r^*)$. The algorithm returns the minimum radius r in u such that the decision algorithm with input r answers "yes". We do this for all point-weight pairs (p,w). Then we have $O(kn)$ radii one of which is exactly r^*. We again apply binary search on these radii to find the minimum radius r over such radii that make the decision algorithm return "yes". Clearly, the minimum radius is exactly r^*. ◀

▶ **Theorem 10.** *Given a set P of n points in the plane and a multiset W of k weights with $k \le n$, an assignment f of weights that minimizes $r(f)$ can be found in $O(k^2n^2\log^4 n)$ time using $O(kn)$ space.*

4 A Faster Algorithm for a Small Set of Weights

We can improve the algorithm in Section 3 for the case that k is sufficiently small compared to n. In this algorithm, we restrict input weights to be at most 1.

Let f^* be an optimal assignment of weights, that is, $r(f^*) = r^*$, and let c^* be the weighted center of P with respect to f^*. A point p in P is called a *determinator* if $d(p,c^*)/f^*(p) = r^*$. We already observed that there exist two or three determinators for any point set P. Moreover, if there exist exactly two determinators, then the two determinators and c^* are collinear.

The algorithm in this section is based on the observation that if $f^*(p) = 1$ for a determinator p, then $d(p,c^*) \ge d(p',c^*)$ for any point p' in P. The following lemma provides a more general observation. A proof of the following lemma can be found in the full version of this paper.

▶ **Lemma 11.** *Let f^* be an optimal assignment of weights with minimum number of determinators. If a determinator p is the ith closest point of P from c^*, it is assigned the ith smallest element in W', where W' is the union of W and the multiset consisting of $n-k$ numbers of weight 1.*

Combining this with Lemma 2, we have the following corollary.

▶ **Corollary 12.** *There exists an optimal assignment f^* of weights that maps the ith closest point of P from $c(f^*)$ to the ith smallest element in W', where W' is the union of W and the multiset consisting of $n-k$ numbers of weight 1.*

Lemma 11 reduces the number of candidates for a determinator with its weight compared to the algorithm in Section 3. Recall that the algorithm considers each of $O(kn)$ point-weight pairs as a determinator and its weight.

We consider three different cases. The first case is that every determinator is assigned weight 1. The second case is that every determinator is assigned a weight strictly less than 1. The third case deals with all remaining situations.

Case 1: Every determinator is assigned weight 1. By Lemma 11, the determinators are the farthest points of c^* among all points in P in this case. This means that the (unweighted) center of P coincides with the weighted center $c(f^*)$ with the assignment f^*. After computing the (unweighted) center of P in $O(n)$ time [11], it suffices to check whether the center is valid or not by using the procedure by Lemma 2. Therefore, this case can be handled in $O(n)$ time in total, excluding the time for sorting the weights in W.

Case 2: Every determinator is assigned a weight smaller than 1. By Lemma 11, a determinator is one of the k closest points from c^*. This is related to the concept of the *order-k Voronoi diagram.* The order-k Voronoi diagram is a generalization of the standard Voronoi diagram. It partitions the plane into regions such that every point in the same region has the same k closest sites. The complexity of the order-k Voronoi diagram of n point sites is $O(kn)$ [9]. There are a number of algorithms to compute the order-k Voronoi diagram with different running times [1, 14]. Among them we use the algorithm in [14], which runs in $O(n\log n + nk2^{c\log^* k}) \leq O(n\log n + nk\log k)$ time using $O(kn)$ space.

In terms of the order-k Voronoi diagram, Lemma 11 can be interpreted as follows. All determinators are sites corresponding to the cell of the order-k Voronoi diagram of P containing c^*. To use this observation, we consider each cell of the order-k Voronoi diagram of P. For each cell, we assign the k weights in W to the k sites corresponding to the cell by applying the algorithm in Section 3 to the sites. This takes $O(k^4\log^4 k)$ time. Then we check whether the weighted center is valid or not. To this end, it suffices to check the distance from the center to the farthest point of the center by Lemma 2. This can be done in $O(\log n)$ time once we have the farthest-point Voronoi diagram of P. In total, this takes $O(k^5 n\log^4 k + T(n,k) + kn\log n)$ time, where $T(n,k)$ is the running time for computing the order-k Voronoi diagram of P.

Case 3: The remaining cases. Here, we apply parametric search. We first apply the procedures that deal with Case 1 and Case 2. Let r_U be the minimum radius of the results of the two procedures. To handle Case 3, we give a decision algorithm that returns "yes" with input $0 < r \leq r_U$ if and only if there exist an assignment f of weights and a point $c \in \mathbb{R}^2$ such that $d(p,c)/f(p) \leq r$ for all points $p \in P$ and $d(q,c) = r$ for some point in $q \in P$. In other words, the decision algorithm returns "yes" if and only if there is an assignment with covering radius r one of whose determinators is assigned weight 1. For a radius r, we call such a point c a *center* with radius r.

The following lemma enables us to apply parametric search. A proof of this lemma can be found in the full version of this paper.

▶ **Lemma 13.** *If an optimal solution belongs to Case 3, then the decision problem for any input r with $r^* \leq r \leq r_U$ returns "yes."*

Decision Algorithm for Case 3. Given a covering radius r, the decision algorithm first computes the intersection I of the disks $D(p,r)$ for all points $p \in P$, where $D(p,r)$ is the disk centered at p with radius r. If the answer for the decision problem is "yes", then a center with r lies in the intersection I. Moreover, a center with r lies on the boundary of I by the definition.

Thus the decision algorithm searches the boundary of I and checks whether there exists a center on the boundary of I. Here, we follow the framework of the algorithm in Section 3. That is, we consider $O(kn)$ circles $C_{p,w}(r)$ for all points $p \in P$ and all weights $w \in W \cup \{1\}$, where $C_{p,w}(r)$ is the circle centered at p with radius rw. Then we compute $O(kn)$ intersection points of each of the circles with the boundary of I and sort them along the boundary of I in $O(kn \log n)$ time. We apply the procedure in the proof of Lemma 4, which checks whether there exists a center with radius r lying on the boundary of I in $O(kn)$ time. Thus the decision algorithm takes $O(kn \log n)$ time.

Overall Algorithm for Case 3. As we did in the decision algorithm, we first compute the intersection $I(r^*)$ of the disks $D(p, r^*)$ for all points $p \in P$. Here, we are not given r^*. Instead of computing the intersection explicitly, we compute its combinatorial structure.

▶ **Lemma 14.** *The combinatorial structure of the intersection of the disks $D(p, r^*)$ for all points $p \in P$ can be computed in $O(n \log n + T(n) \log n)$ time, where $T(n)$ is the running time of the decision algorithm.*

Proof. As we increase the radius r from 0 to r_U, the combinatorial structure of the intersection of the disks $D(p, r)$ may change. We have two types of events where the combinatorial structure changes: an arc of a disk starts to appear in the structure or an existing arc of a disk disappears from the structure. At both types of events, such an arc becomes a point which is a degenerate arc. Moreover, this point is a vertex of the farthest-point Voronoi diagram of P.

To use this fact, we compute the farthest-point Voronoi diagram of P in $O(n \log n)$ time. Then for each vertex v of the diagram, we compute the Euclidean distance between v and its farthest point in P. There are $O(n)$ distances, and we sort them in the increasing order.

Then we apply binary search on the distances using the decision algorithm to find the smallest interval containing r^*. This can be done in $O(T(n) \log n)$ time.

Therefore, for any radius on the interval, the combinatorial structure of the intersection of the disks remains the same. ◀

Now we have the combinatorial structure of the intersection $I(r^*)$. As we did in the algorithm of Section 3, we sort the intersections of $O(kn)$ circles $C_{p,w}(r^*)$ for all point $p \in P$ and all weight $w \in W$ with the boundary of $I(r^*)$ without explicitly computing r^*. This can be done in $O(kn \log n + T(n) \log^2 n)$ time, where $T(n) = O(kn \log n)$ is the running time of the decision algorithm, in a way similar to Lemma 6. Then we find an optimal solution in a way similar to Lemma 7 in $O(kn)$ time if it belongs to Case 3. In total, Case 3 can be dealt in $O(kn \log^3 n)$ time using $O(kn)$ space.

Combining the three cases, we have the following theorem.

▶ **Theorem 15.** *Given a set P of n points in the plane and a multiset W of k weights smaller than or equal to 1 with $k \leq n$, we can compute an assignment f of weights that minimizes $r(f)$ in $O(k^5 n \log^4 k + kn \log^3 n)$ time using $O(kn)$ space.*

5 Concluding Remarks

We would like to mention that the approach in this paper also works under any convex distance function, including the L_p metric for $p \geq 1$. For the L_1 or the L_∞ metric, the optimal weighted center is not necessarily unique though. If the weight assigned to an input point p is subtracted from the distance between p and a point of the plane, the running times of the algorithms can be improved by an $O(\log n)$ factor.

References

1 Pankaj K. Agarwal, Mark de Berg, Jiří Matoušek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order voronoi diagrams. *SIAM Journal on Computing*, 27(3):654–667, 1998.

2 Pankaj K. Agarwal and Micha Sharir. *Computer Science Today: Recent Trends and Developments*, chapter Algorithmic techniques for geometric optimization, pages 234–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

3 Behrooz Alizadeh and Rainer E. Burkard. The inverse 1-center location problem on a tree. Technical Report 2009-03, Graz University of Technology, 2009.

4 Luis Barba, Otfried Cheong, Jean-Lou De Carufel, Michael Gene Dobbins, Rudolf Fleischer, Akitoshi Kawamura, Matias Korman, Yoshio Okamoto, János Pach, Yuan Tang, Takeshi Tokuyama, Sander Verdonschot, and Tianhao Wang. Weight balancing on boundaries and skeletons. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry (SoCG 2014)*, pages 436–443, 2014.

5 Oded Berman, Divinagracia I. Ingco, and Amedeo Odoni. Improving the location of minimax facilities through network modification. *Networks*, 24(1):31–41, 1994.

6 Mao-Cheong Cai, X. G. Yang, and J. Z. Zhang. The complexity analysis of the inverse center location problem. *Journal of Global Optimization*, 15(2):213–218, 1999.

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

8 Martin Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45, 1984.

9 Der-Tsai Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Trans. Computers*, 31(6):478–487, 1982.

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

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

12 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

13 Nimrod Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(6):605–610, 1989.

14 Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry (SoCG 1999)*, pages 390–399, 1999.

15 Jianzhong Zhang, Zhenhong Liu, and Zhongfan Ma. Some reverse location problems. *European Journal of Operational Research*, 124(1):77–88, 2000.