# Fast Approximations for Sums of Distances, Clustering and the Fermat-Weber Problem[†]

Prosenjit Bose [‡] Carleton University

Anil Maheshwari[‡] Carleton University

Pat Morin [§] McGill University

**Abstract**

We describe two data structures that preprocess a set $S$ of $n$ points in $\mathbb{R}^d$ ($d$ constant) so that the sum of Euclidean distances of points in $S$ to a query point $q$ can be quickly approximated to within a factor of $\epsilon$. This preprocessing technique has several applications in clustering and facility location. Using it, we derive an $O(n \log n)$ time deterministic and $O(n)$ time randomized $\epsilon$-approximation algorithm for the so called Fermat-Weber problem in any fixed dimension.

## 1 Introduction

Let $S = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$, with $d$ constant. For a query point $q$ we define the *weight* of $q$ as

$$\mathrm{w}(q) = \sum_{i=1}^{n} \mathrm{d}(q, p_i) \ , \tag{1}$$

where $\mathrm{d}(x, y)$ denotes the Euclidean distance between $x$ and $y$. This function appears frequently as the objective function in facility location and clustering problems [2, 3, 5, 6, 10, 20].

Unfortunately, even with preprocessing, it appears that little can be done in order to speed up the evaluation of $\mathrm{w}(q)$ for an arbitrary query point $q$, and the only known result is the trivial one, namely to evaluate $\mathrm{w}(q)$ in $\Theta(n)$ time using (1) directly. In fact, in any realistic model of computation, it may be impossible to evaluate $\mathrm{w}(q)$ exactly, since it contains square roots that can be irrational numbers [1].

A famous problem related to the function $w$ is the *Fermat-Weber problem* [4] which asks for the point $p_{\mathrm{opt}}$ that minimizes (1). Currently, no exact solution to the Fermat-Weber problem is known, even in the real RAM model of computation. Indeed, Bajaj [1] shows

[†]This research was funded by the Natural Sciences and Engineering Research Council of Canada.

[‡]School of Computer Science, Carleton University, 1125 Colonel By Dr., Ottawa, Ontario, CANADA, K1S 5B6, {jit,maheshwa}@scs.carleton.ca.

[§]School of Computer Science, McGill University, 3480 University Street, McConnell Engineering Building., Room 318, Montreal, Quebec, CANADA, H3A 2A7, morin@cgm.cs.mcgill.ca.

that even for 5 points, the coordinates of $p_{\text{opt}}$ may not be representable even if we allow radicals. In particular, it is impossible to construct an optimal solution by means of ruler and compass.

A review of the literature shows that very little has (or can be) done to get around the numerical difficulties associated with the function $\mathrm{w}(q)$. In fact, in many cases, the function $w'(q) = \sum_{i=1}^{n} (\mathrm{d}(q, p_i))^2$ is used simply because it is more convenient and can be evaluated in constant time after a linear amount of preprocessing. It has been pointed out that the use of Euclidean distance is statistically more robust than squared Euclidean distance [11], and is preferable in many geographic situations [19, 7]. These observations motivate research into approximations of $\mathrm{w}(q)$ that can be evaluated more efficiently.

In this paper we show how to preprocess $S$ so that we can quickly evaluate an approximation to $\mathrm{w}(q)$. More specifically, we describe two data structures for computing approximations to $\mathrm{w}(q)$. The first data structure is based on range trees and evaluates a function $\mathrm{w}_k(q)$ that can be evaluated in $O(k \log^{d-1} n)$ time after preprocessing requiring $O(kn \log^{d-1} n)$ time and space. The value of $\mathrm{w}_k(q)$ obeys the inequalities

$$\mathrm{w}(q) \leq \mathrm{w}_k(q) \leq (1+\epsilon) \cdot \mathrm{w}(q) \ , \tag{2}$$

where $\epsilon$ is a constant that decreases as $k$ increases.

The second data structure is based on quadtrees and evaluates a function $\mathrm{w}_\epsilon(q)$ in $O(k \log n)$ time after preprocessing requiring $O(kn \log n)$ time and using $O(n)$ space. Again, $k$ is a function of $\epsilon$ and $d$ and $\mathrm{w}_\epsilon(q)$ satisfies $(1-\epsilon)\mathrm{w}(q) \leq \mathrm{w}_\epsilon(q) \leq (1+\epsilon)\mathrm{w}(q)$.

The running time and storage requirements of the quadtree data structure are asymptotically better than those of the range-tree based data structure which raises the question "why talk about it?" The reason we describe both data structures is that the range tree data structure has three advantages. The first is that the constants in the big-Oh notation are significantly lower, especially in the query time. The second advantage is that it generalizes immediately to the weighted case in which each point is assigned a weight $\mathrm{w}_i$ and the objective function is $\mathrm{w}(q) = \sum_{i=1}^{n} \mathrm{w}_i \mathrm{d}(q, p_i)$. The third advantage is that it also generalizes immediately to the dynamic case in which we insert and delete points in the set $S$.

We also study applications of these preprocessing techniques to clustering and facility location. One of these applications is an $O(n \log n)$ time deterministic and $O(n)$ time randomized $\epsilon$-approximation algorithm for the Weber-Fermat problem in any fixed dimension $d$.

The remainder of this paper is organized as follows. Section 2 reviews the data structuring techniques used by our preprocessing and querying algorithms. Section 3 describes our range tree based data structure. Section 4 describes the quadtree based data structure. Section 5 presents applications of our techniques to clustering and facility location problems. Finally, Section 6 summarizes and concludes with directions for continuing work.

## 2 Data Structures

In this section we review the main data structures used in our preprocessing and query algorithms. Throughout this section, and in the remainder of the paper we will use the

notation $x[i]$ for the $i$th coordinate of a point $x$.

Let $x_1, \ldots, x_n$ be a sequence of real numbers in increasing order. A *segment tree* on $x_1, \ldots, x_n$ is a complete binary search tree whose leaves correspond to the intervals

$$[x_1, x_2), [x_2, x_3), \ldots, [x_{n-1}, x_n), [x_n, x_n]$$

and for which an internal node $v$ corresponds to the interval $x_i, x_j$ spanned by the subtree rooted at $v$.

The *range tree* is defined recursively as follows: A range tree $T$ of dimension 1 is a balanced binary search tree. A range tree of dimension $d$ consists of a primary segment tree $T'$ on the set $p_1[1], \ldots, p_n[1]$. Each node $v$ of $T'$ contains a pointer to a $d-1$ dimensional range tree that contains all points $p_i$ such that $p_i[1]$ is contained in the interval of $v$.

We say that a point $p_i$ *dominates* a point $p_j$, denoted $p_i \geq p_j$ if and only if $p_i[k] \geq p_j[k]$ for all $1 \leq k \leq d$. Range trees can be used to answer *dominance queries*[1] of the form: Report the set of points $S' \subseteq S$ that dominate the query point $q$. Indeed, using standard data structuring techniques, range trees can be used to compute any associative function on the points of $S'$. We call such queries *generalized dominance counting queries*. In our particular case, we use range trees to store sums of coordinates of points in $S'$.

The performance of range trees is described by the following theorems [16]:

**Theorem 1.** *For a static set $S$ of $n$ points in $\mathbb{R}^d$, a $d$-dimensional range tree can be constructed in $O(n \log^{d-1} n)$ time and space that answers generalized dominance counting queries in $O(\log^{d-1} n)$ time.*

**Theorem 2.** *The $d$ dimensional range tree supports insertion, deletion and generalized dominance counting queries in $O(\log^d n)$ time and requires $O(n \log^{d-1} n)$ space, where $n$ is the maximum number of points stored in the range tree at any given time.*

The second data structure we use is a *quadtree* [18]. Let $S = \{p_1, \ldots, p_n\}$ be a set of $n$ points in $\mathbb{R}^d$ contained in a hypercube $C$ of side length $l$. A quad tree $T$ is constructed recursively as follows: The root of $T$ corresponds to the hypercube $C$. The root has $2^d$ children corresponding to the $2^d$ subcubes of $C$ of side length $l/2$. The leaves of $T$ are nodes with side length $l\epsilon/n$. Associated with each leaf $v$ of $T$ is a list of the points of $S$ contained in the hypercube spanned by $v$. Associated with each internal node is the number of points contained in the hypercubes spanned by the children of $v$.

From the above definition, it follows that $T$ has $\Theta(n^d)$ nodes. However, a *compressed quadtree* reduces this size to $O(n)$ by removing nodes not containing any points of $S$ and eliminating nodes having only one child. The following theorem describes the performance of the compressed quadtree [18]:

**Theorem 3.** *For a static set $S$ of $n$ points in $\mathbb{R}^d$, a $d$-dimensional compressed quadtree can be constructed in $O(n \log n)$ time and $O(n)$ space.*

[1] These are sometimes called $d$ sided range queries.

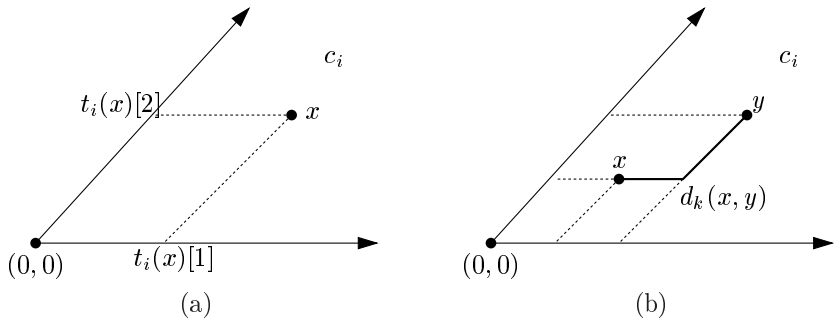


Figure 1: The definition of (a) $t_i(x)$ and (b) $\mathrm{d}_k(x,y)$ (shown in bold).

# 3 A Range Tree Based Data Structure

In this section we describe our range tree based data structure for approximating the sum $\mathrm{w}(q)$. We begin by defining a distance function $\mathrm{w}_k(q)$ that approximates $\mathrm{w}(q)$ and then show how $\mathrm{w}_k(q)$ can be evaluated exactly using range trees.

## 3.1 The $\mathrm{d}_k$ Distance Function

A *simplicial cone* in $\mathbb{R}^d$ is the intersection of $d$ (open or closed, as convenient) half-spaces, each of whose supporting planes contain the origin, $O$. Note that for any set of $d$ points on the unit hypersphere, there is a unique minimal simplicial cone $c$ that contains these points. This set of $d$ points defines a set of $d$ rays, where each ray originates at the origin and contains one of the points. We call these the *axes* of $c$. A simplicial cone $c$ *has diameter bounded by* $\theta$ if for any two points $x$ and $y$ in $c$, $\angle xOy \leq \theta$.

Let $C = \{c_1, \ldots, c_k\}$ be a set of simplicial cones with diameter bounded by $\theta$ and that form a partition of $\mathbb{R}^d$. It has been shown by Yao [21] that such sets of cones exist, and that the number of cones, $k$, is a function only of $d$ and $\theta$. For example, in the plane $C$ could be the set of cones defined by directions $\{0, 2\pi/k, 4\pi/k, 8\pi/k, \ldots, 2\pi(k-1)/k\}$.

For a point $x$ in $\mathbb{R}^d$, we use the notation $\|x\|$ to denote the sum $\sum_{j=1}^{d} x[j]$. We use the notation $t_i(x)$ to denote $x$ represented in the coordinate system whose axes are parallel to the axes of $c_i$. See Fig. 1.a for an illustration.

We are now ready to define our distance function: The $k$-oriented distance from the origin $O$ to a point $x$ contained in $c_i$, denoted $\mathrm{d}_k(O,x)$, is $\|t_i(x)\|$, i.e., the $k$-oriented distance from $O$ to $x$ is the length of the shortest path from $O$ to $x$ traveling only in directions parallel to the axes of $c_i$. The $k$-oriented distance between two points $x$ and $y$ is obtained by translating $x$ to the origin, i.e., $\mathrm{d}_k(x,y) = \mathrm{d}_k(O, y-x)$. See Fig. 1.b for an illustration.

The following lemma follows easily from trigonometry [17].

**Lemma 1.** *For all $a, b \in \mathbb{R}^d$ and any fixed $d$, $\mathrm{d}(a,b) \leq \mathrm{d}_k(a,b) \leq (1+\epsilon) \cdot \mathrm{d}(a,b)$, where $\epsilon$ is a positive constant that decreases as $k$ increases.*

In the important special case of $\mathbb{R}^2$, the exact bound is $\mathrm{d}(a,b) \leq \mathrm{d}_k(a,b) \leq 1/\cos(\theta/2)\mathrm{d}(a,b)$

The set of points $y$ such that $\mathrm{d}_k(x,y) \leq 1$ form a convex polytope whose vertices lie on the unit circle centered at $x$, i.e., under the $\mathrm{d}_k$ distance function, a "unit disk" is a convex polytope inscribed in the standard (Euclidean) unit circle. It is also worth noting that the $\mathrm{d}_k$ distance function is not necessarily symmetric.

### 3.2 Fast Evaluation of $\mathrm{w}_k(q)$

Next we show how to preprocess the set $S$ so that for any query point $q$ we can evaluate the sum

$$\mathrm{w}_k(q) = \sum_{i=1}^{n} \mathrm{d}_k(q, p_i) \tag{3}$$

in $O(k \log^{d-1} n)$ time. By Lemma 1 this gives an approximation of $\mathrm{w}(q)$ that is accurate to within a factor of $\epsilon$.

Let $c'_j$ be $c_j$ translated so that it's apex lies at $q$. With some manipulation, we can rewrite (3) as

$$\mathrm{w}_k(q) = \sum_{j=1}^{k} \left( \sum_{i=1}^{n} \|t_j(p_i)\| \cdot [t_j(q) \leq t_j(p_i)] - \|t_j(q)\| \cdot |\{p_i : t_j(q) \leq t_j(p_i)\}| \right) \quad .^2 \tag{4}$$

Next, note that

$$\sum_{i=1}^{n} \|t_j(p_i)\| \cdot [t_j(q) \leq t_j(p_i)] \tag{5}$$

can be expressed as a dominance counting query that asks for the sum of all points in $\{t_j(p_1), \ldots, t_j(p_n)\}$ that dominate $t_j(q)$. Similarly,

$$\|t_j(q)\| \cdot |\{p_i : t_j(q) \leq t_j(p_i)\}| \tag{6}$$

is the number of points in $\{t_j(p_1), \ldots, t_j(p_n)\}$ that dominate $q$ times $\|q\|$. Therefore, by storing $\{t_j(p_1), \ldots, t_j(p_n)\}$ in a range tree, for each $1 \leq j \leq k$ we can evaluate (4) in $O(k \log^{d-1} n)$ time.

To summarize, our preprocessing is as follows:

1: **for** $1 \leq j \leq k$ **do**
2: Construct a range tree $T_j$ containing $\{t_j(p_1), \ldots, t_j(p_n)\}$
3: **end for**

[2] We adopt Kenneth Iverson's notation where $[X]$ takes on the value 1 if the predicate $X$ is true and 0 otherwise [12].

The query algorithm is as follows:

1: $s \leftarrow 0$
2: **for** $1 \leq j \leq k$ **do**
3: Use $T_j$ to compute $s \leftarrow s + (5) - (6)$
4: **end for**

The correctness of this algorithm follows from the above discussion, while the running time follows from the running times of range tree operations. In the first case, fractional-cascading is used to reduce the running time by a logarithmic factor.

**Theorem 4.** *Given a set $S$ of $n$ points in $\mathbb{R}^d$, $S$ can be preprocessed in $O(kn \log^{d-1} n)$ time and space so that for any query point $q$, $\mathrm{w}_k(q)$ can be evaluated in $O(k \log^{d-1} n)$ time.*

In a dynamic setting, we obtain the following result.

**Theorem 5.** *There exists a dynamic data structure supporting, in $O(k \log^d n)$ time and $O(kn \log^{d-1} n)$ space, insertion, deletion, and queries of the form: Compute $\mathrm{w}_k(q)$ for an arbitrary query point $q$.*

# 4 A Quadtree Based Data Structure

In this section, we show how the same result can be obtained using quadtrees. Suppose that the smallest axis-aligned hypercube containing $S$ has side length $l$ and that we have computed a compressed quadtree $T$ on $S$.

Consider a node $v$ of $T$ corresponding to a hypercube $C_v$ of side-length $l_v$. Let $c_v$ denote the center of $C_v$ and suppose that $\mathrm{d}(q, c_v) > l_v/\epsilon$. Then it is clear that for any point $x \in C_v$,

$$\mathrm{d}(q, c_v) - l_v\sqrt{d}/2 \leq \mathrm{d}(q, x) \leq \mathrm{d}(q, c_v) + l_v\sqrt{d}/2$$

Since $\mathrm{d}(q, c_v) > l_v/\epsilon$ we have

$$(1 - \epsilon\sqrt{d}/2)\mathrm{d}(q, c_v) \leq \mathrm{d}(q, x) \leq (1 + \epsilon\sqrt{d}/2)\mathrm{d}(q, c_v)$$

Therefore, the point $c$ is a good substitute for any point in $C$. Let $c_v$ denote the center of the hypercube for node $v$, $\mathrm{card}(v)$ denote the number of points of $S$ contained in this hypercube and $\mathrm{children}(v)$ denote the children of $v$. The following algorithm, applied to the root of $T$, gives a method of computing the approximate value of $\mathrm{w}(q)$.

COMPUTE-SUM$(q, v, s)$

1: **if** $v$ is a leaf **or** $\mathrm{d}(q, c_v) > l_v/\epsilon$ **then**
2: $s \leftarrow s + \mathrm{card}(v) \cdot \mathrm{d}(q, c_v)$
3: **return** $s$
4: **else**
5: **for** $v' \in \mathrm{children}(v)$ **do**
6: $s \leftarrow$COMPUTE-SUM$(q, v', s)$
7: **end for**

8: **return** $s$
9: **end if**

**Lemma 2.** *The above algorithm, when applied to the root of $T$ runs in $O(k \log n)$ time and outputs a value $\mathrm{w}_\epsilon(q)$ satisfying $(1 - \epsilon\sqrt{d})\mathrm{w}(q) \leq \mathrm{w}_\epsilon(q) \leq (1 + \epsilon\sqrt{d})\mathrm{w}(q)$. The value of $k$ is a function of $d$ and $\epsilon$.*

*Proof.* First we prove the upper and lower bounds on $\mathrm{w}_\epsilon(q)$. Let $v_i$ denote the unique node of $T$ for which line 2 of the algorithm was executed and which contains the point $p_i$. Partition the point set $S$ into two sets $S_1$ and $S_2$. If $v_i$ is a leaf, then we assign $p_i$ to $S_1$, otherwise we assign $v_i$ to $S_2$.

We proceed by bounding the error in the computation of the value $s$. The leaves of $T$ are associated with hypercubes of side length $l\epsilon/n$. Therefore, since there are at most $n$ elements of $S_1$, the error introduced by elements of $S_1$ is at most $l\epsilon\sqrt{d}/2$. Furthermore, from the preceding discussion, an element $p_i$ of $S_2$ contributes at most $(\epsilon\sqrt{d}/2)\mathrm{d}(q, p_i)$ to the error. Therefore,

$$\mathrm{w}(q) = \sum_{i=1}^{n} \mathrm{d}(q, p_i) = s \pm (\epsilon\sqrt{d}/2)\left(\sum_{i=1}^{n} \mathrm{d}(q, p_i) + l\right) \subseteq s \pm (\epsilon\sqrt{d})\mathrm{w}(q) \ .$$

(The subset relationship follows from the observation that $\mathrm{w}(q) \geq l$.)

To prove the bound on the running time, it is sufficient to count the number of times that line 5 is executed. Therefore, we count how many nodes of $T$ at each level $i$ do not satisfy the conditions of line 1. The set of points $c_v$ that do not satisfy $\mathrm{d}(q, c_v) > l/\epsilon 2^i$ form the hypersphere of radius $l/\epsilon 2^i$ centered at $q$. Therefore, the hypercube $C_v$ associated with each node $v$ at level $i$ that does not satisfy the conditions of line 2 is contained in the hypersphere of radius $(l + \sqrt{d}/2)/\epsilon 2^i$ centered at $q$ and this hypersphere has volume less than $((l + \sqrt{d}/2)/\epsilon 2^i)^d$. Since the hypercube associated with a node a level $i$ occupies a volume of $(l/2^i)^d$ and no two such hypercubes overlap, there are at most $(1/\epsilon + \sqrt{d}/2)^d$ nodes a level $i$ that do not satisfy the conditions of line 2. □

Combined with the existing results on quadtree construction, we obtain the following theorem.

**Theorem 6.** *Given a set $S$ of $n$ points in $\mathbb{R}^d$, $S$ can be preprocessed in $O(kn \log n)$ time and $O(kn)$ space so that for any query point $q$, $\mathrm{w}_\epsilon(q)$ can be evaluated in $O(k \log n)$ time. The value of $\mathrm{w}_\epsilon(q)$ satisfies $(1 - \epsilon\sqrt{d})\mathrm{w}(q) \leq \mathrm{w}_\epsilon(q) \leq (1 + \epsilon\sqrt{d})\mathrm{w}(q)$ and the value of $k$ is a function of $d$ and $\epsilon$.*

# 5 Applications

In this section we discuss some applications of our preprocessing technique. In the following $P(n)$ denotes the preprocessing time and $Q(n)$ denotes the query time of an approximate data structure for evaluating $\mathrm{w}(q)$. In the case or range trees, $P(n) = O(kn \log^{d-1} n)$ and $Q(n) = O(k \log^{d-1} n)$ while for quadtrees $P(n) = O(kn \log n)$ and $Q(n) = O(k \log n)$.

## 5.1 $k$-Medoids Clustering

Clustering involves partitioning a set of points into "similar" groups. The definition of similarity depends on the clustering method being used. The dissimilarity measure used in the *k-medoids* clustering method [11, 14] is

$$g(S) = \frac{1}{n} \cdot \min \{ \mathrm{w}(p_i) : 1 \leq i \leq n \} \quad .$$

No method is known to compute $g(S)$ exactly in subquadratic time. However, it is possible to approximate the sum in $O(P(n) + nQ(n))$ time by first building an approximate data structure for evaluating $\mathrm{w}(q)$ and querying that data structure for each of the points in $S$. Of course, this approximation obeys the inequalities $(1-\epsilon)g(S) \leq g_\epsilon(S) \leq (1+\epsilon)g(S)$.

## 5.2 The Weber Problem

Let $S = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$. The *Weber problem*[3] is a facility location problem that asks us to find the point $p$ that minimizes $\mathrm{w}(p)$. Currently, no exact solution to this problem (even in the real RAM model of computation) is known. Under the $L_1$ (Manhattan) metric, the Weber problem can be solved in linear time by finding a point whose coordinates are the median in each dimension. Under the Euclidean metric, Chandrasekaran and Tamir [4] give a polynomial time approximation scheme based on the ellipsoid method (c.f., [15]).

Given the apparent difficulty in solving the Weber problem exactly, it seems reasonable to try and approximate the solution by finding the point $p'$ that minimizes $\mathrm{w}_k(p')$. Towards this end, we give a simple and efficient prune-and-search algorithm that works in $O(P(n) + Q(n)\log n)$ time when the dimension $d$ is fixed.

Our approximation algorithm solves the Weber problem exactly under the $\mathrm{w}_k$ distance function. To understand the algorithm, one should consider the surface $z = \mathrm{w}_k(p)$ in $\mathbb{R}^{d+1}$. It follows from the definition of $\mathrm{w}_k$ that this surface is convex away from the origin (i.e, any local minimum is a global minimum) and piecewise linear.

Consider the set of simplicial cones used in our data structure for evaluating $\mathrm{w}_k$. These cones give rise to a set of hyperplanes in the following way: Each cone $c$ is the intersection of $d$ hyperplanes. Let $P$ be the set of all such hyperplanes. Consider the arrangement of hyperplanes $A$ obtained by adding for each point $p_i \in S$ a copy of each hyperplane in $P$ translated so that it contains $p_i$. Then it should be clear that each cell of $A$ is a linear piece of the surface $z = \mathrm{w}_k(p)$. It follows then that $\mathrm{w}_k$ is minimized at a vertex of $A$.

In order to find the lowest point on the surface $z = \mathrm{w}_k(p)$ we use the following pruning algorithm. Initially, we have $k$ sets $H_0, \ldots, H_{k-1}$ of parallel hyperplanes corresponding to the hyperplanes in the arrangement $A$. The algorithm proceeds in rounds. During round $i$ we take the median hyperplane $h \in H_{i \bmod k}$ that splits the hyperplanes in $H_{i \bmod k}$ into two sets of roughly equal size. Note that $h$ partitions $\mathbb{R}^d$ into two half spaces $h_1$ and $h_2$, one (or both) of which contains a minimum of $\mathrm{w}_k$. We then determine which of $h_1$ or $h_2$ contains a minimum of $\mathrm{w}_k$ ($h_2$, say), at which point we can discard all the hyperplanes $h \in H_{i \bmod k}$

[3] Also called the Fermat-Weber problem [8, 13].

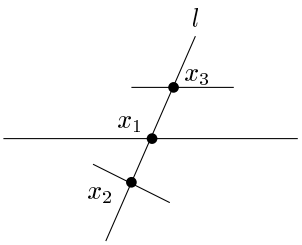


Figure 2: Finding the minimum of $\mathrm{w}_k$ restricted to $l$.

that are contained in $h_1$. It is clear that this process terminates after $O(k \log n)$ rounds. In the following, we show how to implement each round in $O(u + Q(n) \log n)$ time, where $u$ is the size of the set $H_{i \bmod k}$ at the beginning of the round.

Each round consists of two phases. In the first phase, we find a point $p_h$ on $h$ that minimizes $\mathrm{w}_k$. In the second phase, we use $p_h$ to determine whether a minimum lies in $h_1$ or $h_2$.

To implement the first phase, we first note that the problem of minimizing $\mathrm{w}_k$ constrained to $h$ is a $d-1$ dimensional instance of our minimization problem. Thus, we can recurse on the dimension. In the 1 dimensional case, we have a line $l$, and we want to find the point $p_l$ on $l$ that minimizes $\mathrm{w}_k$. Since $z = \mathrm{w}_k$ is piecewise linear, $p_l$ lies on an intersection of $l$ with some other hyperplane in $\cup\{H_1, \ldots, H_k\}$. Therefore, we compute all the intersections of $l$ with hyperplanes in $\cup\{H_1, \ldots, H_k\}$. We then find the median intersection point $x_1$ and the two intersection points $x_2$ and $x_3$ that are adjacent to $x_1$ on $l$ (see Fig. 2). By evaluating $\mathrm{w}_k(x_1)$, $\mathrm{w}_k(x_2)$ and $\mathrm{w}_k(x_3)$ we can determine whether $p_l$ comes before, after, or is $x_1$ itself.

Therefore, if we have $u$ candidate points, then half of these can be eliminated in $O(u + Q(n))$ time. The cost of finding $p_l$ is then given by the recurrence $T(u) = T(u/2) + O(u + Q(n))$, which solves to $O(u + Q(n) \log u)$. Therefore, the overall cost of computing the point $p_h$ on $h$ that minimizes $\mathrm{w}_k$ is given by the recurrence

$$T(d, n) = \begin{cases} O(u + Q(n) \log u) & \text{if } d = 1 \\ T(d-1, n) + O(u) & \text{otherwise,} \end{cases}$$

which solves to $O(u + Q(n) \log u)$ for any fixed $d$.

Next we show how, given a point $p_h$ on $h$ that minimizes $\mathrm{w}_k$, to decide whether there is a global minimum of $\mathrm{w}_k$ in the halfspace $h_1$ or $h_2$ or on $h$ itself. Suppose that $h_2$ contains a minimum of $\mathrm{w}_k$. Then $p_h$ is a point in $h_1$ that minimizes $\mathrm{w}_k$. Thus, if we can find a point $p'_h$ in $h_2$, such that $\mathrm{w}_k(p'_h) \leq \mathrm{w}_k(p_h)$ then it must be that a minimum of $\mathrm{w}_k$ is contained in $h_2$. Similar statements also hold for $h_1$.

To find the point $p'_h$, we consider the two opposite rays $r_1$ and $r_2$ originating at $p_h$, orthogonal to $h$ and contained in $h_1$ and $h_2$, respectively. By examining each hyperplane in $H$, we find the first hyperplane $h_{r_1} \in H$ and $h_{r_2} \in H$ intersected by $r_1$, respectively, $r_2$. Let $p_1$, respectively $p_2$, be the intersection of $h_{r_1}$ and $r_2$, respectively $h_{r_2}$ and $r_2$. There are then four cases which can be easily verified:

**Case 1:** $\mathrm{w}_k(p_1) \leq \mathrm{w}_k(p_h) \leq \mathrm{w}_k(p_2)$. In this case, $p_1 = p'_h$ is witness to the fact that a minimum of $\mathrm{w}_k$ is contained in $h_1$.

**Case 2:** $\mathrm{w}_k(p_1) \geq \mathrm{w}_k(p_h) \geq \mathrm{w}_k(p_2)$. In this case, $p_2 = p'_h$ is witness to the fact that a minimum of $\mathrm{w}_k$ is contained in $h_2$.

**Case 3:** $\mathrm{w}_k(p_1) < \mathrm{w}_k(p_h) > \mathrm{w}_k(p_2)$. This case can not occur, because then $p_h$ would not be a minimum of $\mathrm{w}_k$ in $h_1$ or $h_2$.

**Case 4:** $\mathrm{w}_k(p_1) > \mathrm{w}_k(p_h) < \mathrm{w}_k(p_2)$. In this case, $p_h$ is a minimum of $\mathrm{w}_k$, since it is a minimum of $\mathrm{w}_k$ in $h_1$ and a minimum of $\mathrm{w}_k$ in $h_2$.

Thus, determining whether to discard $h_1$ or $h_2$ requires computing all the intersections of hyperplanes in $H$ with $l'$ and two calls to our data structure for evaluating $\mathrm{w}_k$. The computational cost of determining whether to discard $h_1$ or $h_2$ is therefore $O(u + Q(n))$

In summary, our algorithm consists of $O(k \log n)$ rounds. The cost of the $ki$th round is $O(kn/2^i + Q(n) \log(kn/2^i))$. Therefore, the time to compute a point $p'$ that minimizes $\mathrm{w}_k$ is $O(P(n) + kn + Q(n) \log n) = O(P(n))$. Compared to the point $p$ that minimizes $\mathrm{w}(p)$, $p'$ satisfies $(1 - \epsilon)\mathrm{w}(p) \leq \mathrm{w}(p') \leq (1 + \epsilon)\mathrm{w}(p)$.

Indyk [9] has given a randomized data structure for testing whether $(1 - \epsilon)\mathrm{w}(p) > \mathrm{w}(q)$ for any two query points $p$ and $q$. The algorithm uses linear preprocessing time, has polylogarithmic query time and answers correctly with high probability. When combined with the prune-and-search technique described above, Indyk's data structure yields a linear-time algorithm for the Fermat-Weber problem that delivers an $\epsilon$ approximation with high probability.

## 5.3 The Constrained Facility Location (Weber) Problem

Another version of the Weber problem is the so called *constrained facility location problem* which asks us to find the point $p$ which minimizes $\mathrm{w}(q)$ and is contained in a constraining polyhedron $P$. We note that, if $P$ is convex, then using the same techniques used to solve the unconstrained Weber problem, it is possible to solve this problem approximately in $O(P(n))$ time with the same approximation factor as for the Weber problem.

## 5.4 Constrained Obnoxious Facility Location

Let $S = \{p_1, \ldots, p_n\}$ be a set of points in $\mathbb{R}^d$ and let $P$ be a polyhedron in $\mathbb{R}^d$ with $m$ vertices. The *constrained obnoxious facility location problem* is that of finding a point $p$ in $P$ that maximizes $\mathrm{w}(p)$.

Since $w$ is concave, it follows that $p$ lies on a vertex of the convex hull of $P$. Therefore, a straightforward way to solve the constrained obnoxious facility location problem is to evaluate $\mathrm{w}(v)$ at every vertex $v$ of $P$ and take the maximum. The running time of this algorithm is clearly $O(nm)$. Under the Euclidean distance measure, no better algorithm is known. Under the $L_1$ metric, it is not very difficult to derive an $O(n \log n + m \log n)$ time algorithm for this problem.

We note simply that by using the preprocessing technique described here to approximately evaluate $\mathrm{w}(v)$ at each vertex $v$ of $P$, it is possible to find, in $O(P(n) + Q(n))$ time,

a point $p'$ such that $(1-\epsilon)\mathrm{w}(p') \leq \mathrm{w}(p) \leq (1+\epsilon)\mathrm{w}(p')$.

# 6 Conclusions

In this paper we have given algorithms for preprocessing a set of points so that an approximation to the objective function $w$ can be computed quickly. Using this approximation we have shown how to approximately solve some fundamental problems in clustering and facility location for which no exact subquadratic time algorithms are currently known.

Another way of using the approximation algorithm would be as a filter. In cases where the number of potential locations is finite (as in the obnoxious facility location problem), it may be efficient to use our preprocessing algorithm to compute an exact solution. Indeed, by approximately evaluating $\mathrm{w}(q)$ it is possible to determine a (possibly empty) set of locations that are surely non-optimal. By increasing the quality of the approximation until this set contains all but one element the optimal solution is found. The resulting algorithm has a "precision-sensitive" running time and has the advantage that it can be interrupted at any time and yield an approximate solution with tight bounds on its quality.

# Acknowledgement

The authors would like to thank Piotr Indyk for bringing his paper [9] to our attention and an anonymous referee for suggesting the quadtree based data structure.

# References

[1] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete and Computational Geometry*, 3:177–191, 1988.

[2] S. Bespamyatnikh, K. Kedem, and M. Segal. Optimal facility location under various distance functions. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures (WADS'99)*, pages 318–329, 1999.

[3] J. Brimberg and G. O. Wesolowsky. The rectilinear distance minsum problem with minimum distance constraints. *Location Science*, 3(3):203–215, 1995.

[4] R. Chandrasekaran and A. Tamir. Algebraic optimization: The Fermat-Weber location problem. *Mathematical Programming*, 46(2):219–224, 1990.

[5] R. L. Church and R. S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12(2):107–119, 1978.

[6] H. Elgindy and M. Keil. Efficient algorithms for the capacitated 1-median problem. *ORSA Journal of Computers*, 4:418–424, 1982.

[7] V. Estivill-Castro and M. E. Houle. Robust distance-based clustering with applications to spatial data mining. *Algorithmica*, 2000. To appear.

[8] R. L. Francis. *Facility Layout and Location: An Analytical Approach*. Prentice-Hall, 1974.

[9] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st ACM Symposium on Theory of Computing (STOC'99)*, 1999.

[10] M. J. Katz, K. Kedem, and M. Segal. Improved algorithms for placing undesirable facilities. In *Abstracts for the Eleventh Canadian Conference on Computational Geometry (CCCG'99)*, pages 65–67, 1999. `http://www.cs.ubc.ca/conferences/CCCG/elec_proc/elecproc.html`.

[11] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[12] D. E. Knuth. Two notes on notation. *American Mathematical Monthly*, 99:403–422, 1992.

[13] Z. A. Melzak. *Companion to Concrete Mathematics; Mathematical Techniques and Various Applications*. John Wiley & Sons, 1973.

[14] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th Conference on Very Large Databases (VLDB)*, pages 144–155, 1994.

[15] C. H. Papadimitriou and M. Yannakakis. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[16] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[17] J. Ruppert and R. Seidel. Approximating the $d$-dimensional complete Euclidean graph. In *Proceedings of the Third Canadian Conference on Computational Geometry (CCCG'91)*, pages 207–210, 1991.

[18] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[19] C. Watson-Gandy. A note on the centre of gravity in depot location. *Management Science*, 18:B478–481, 1972.

[20] R. Webster and M. A. Oliver. *Statistical Methods in Land and Resource Survey*. Oxford University Press, 1990.

[21] A. C-C. Yao. On constructing minimum spanning trees in $k$-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.