An Optimal Algorithm for the Weighted Median Problem

Daxin Zhu[a], Xiaodong Wang[a,b,*]
[a]Faculty of Mathematics & Computer Science, Quanzhou Normal University, China
Email: dex@qztc.edu.cn
[b] Faculty of Mathematics & Computer Science, Fuzhou University, China
Email: wangxiaodong@qztc.edu.cn

***Abstract*— In this paper, we consider the weighted rectilinear min-sum facility problem to minimize the sum of the weighted rectilinear distance between the given points and a new added point. The core problem is the problem in its one dimensional cases noted as the weighted median problem. We present efficient algorithms for optimally solving the weighted median problem. The computational experiments demonstrate that the achieved results are not only of theoretical interest, but also that the techniques developed may actually lead to considerably faster algorithms.**

***Index Terms*— Weighted median, facility location, binary search, pivot selection, linear time, optimal algorithm**

I. Introduction

In a typical weighted rectilinear min-sum facility location problem we are given a finite set of points $S = \{p_0, p_1, \ldots, p_{n-1}\}$ of n points in R^d with positive weights $w_0, w_1, \ldots, w_{n-1}$, and the goal is to find a point $y^* = (y_0^*, y_1^*, \ldots, y_{d-1}^*) \in R^d$ which minimizes the sum of weighted rectilinear distances (with l_1 norm) between the points in S and y^*.

A possible application of this problem is finding a location for a service center in a town in which all the streets are parallel to the axes. The location of the service center should minimize the sum of the weighted rectilinear distance between n customer locations given in S, and the service center.

When deciding where to place a facility that serves geographically scattered client sites - whether the facility is a delivery center, a distribution center, a transportation hub, a fleet dispatch location, etc. - a typical objective is to minimize the sum of the distances from the facility's location to the client sites. Furthermore, we may want to give the distance to certain sites more "weight" in the calculation, for example if a site requires several deliveries daily as opposed to one delivery daily to other sites, resulting in higher fuel and other costs. Or, perhaps we want to ensure particularly good service to a certain site. In these cases, these higher-weighted client sites may have a greater relative influence on the final location.

Manuscript received August 10, 2011; revised January 2, 2012; accepted April 16, 2012. © 2005 IEEE.
* Corresponding author
This work was supported in part by the Natural Science Foundation of Fujian (Grant No.2013J0101), and the Haixi Project of Fujian (Grant No.A099).

This problem arises in the area of facility location in operations research known as weighted rectilinear 1-median problem, proposed by Hakimi [8], [9], and many variants of it have been studied (see, for example, [1], [3], [4], [6], [7], [12], [15]).

The weighted rectilinear min-sum facility location problem in R^d can be formulated as

$$\min_{y \in R^d} g(y) = \sum_{i=0}^{n-1} \sum_{j=0}^{d-1} w_i |p_{ij} - y_j| \tag{1}$$

where $p_i = (p_{i,0}, p_{i,1}, \ldots, p_{i,d-1}) \in R^d$, $i = 0, 1, \ldots, n-1$, are the n given points in the set S.

The l_1 metric is separable, in the sense that the weighted distance between two points is the sum of their weighted distances of every coordinates. Therefore we can solve the problem for every coordinates separately. So we can consider the problem for the first coordinate without loss of generality. The problem is reduced to the one dimensional case:

$$\min_{x \in R} f(x) = \sum_{i=0}^{n-1} w_i |x_i - x| \tag{2}$$

where $x_i,\ i = 0, 1, \ldots, n-1$, are the n real numbers.

The problem in its one dimensional case is also noted as a weighted median problem. It is the core problem of the weighted rectilinear min-sum facility problem. The complexity of the problem is unknown [13].

We present optimal algorithms for the problems described above by exploiting the convexity of the problems. We report on computational experience with these algorithms and their time and space complexities.

The organization of the paper is as follows.

In the following 4 sections we describe our presented algorithms and our computational experience with these algorithms. In section 2 we present a new optimal algorithm for the weighted median problem. Its correctness and complexities are then analysed rigorously in section 3. In section 4, several improvements on the time complexities of the basic algorithm are proposed. In section 5 we give a computational study of the presented algorithms which demonstrates that the achieved results are not only of theoretical interest, but also that the techniques developed may actually lead to practical algorithms. Some concluding remarks are in section 6.

doi:10.4304/jcp.9.2.257-265

A preliminary conference version of this paper was presented at International Conference on Computational Intelligence and Software Engineering (CISE 2010) [17]. In this paper the correctness and complexities are proved rigorously, but not just stated in intuitively. More experiment details are described in this version of the paper.

II. Description of the Algorithms

In order to design an efficient algorithm for the weighted median problem, we first investigate some properties of the function $f(x)$.

Lemma 1:

$$\min_{x\in R} f(x) = \min_{0\le i<n} \{k_i x_i + b_i\} \tag{3}$$

$$\max_{x\in R} f(x) = \max_{0\le i<n} \{k_i x_i + b_i\} \tag{4}$$

Proof. The function $f(x)$ in formula (2) is obviously a piecewise linear function. Without loss of generality, let us assume that

$$-\infty = x_{-1} < x_0, \ldots, \le x_{n-1} < x_n = +\infty \tag{5}$$

The piecewise linear function $f(x)$ can be expressed as follows.

$$f(x) = f_j(x) = k_j x + b_j,\ x \in (x_{j-1}, x_j] \tag{6}$$

where

$$k_j = \sum_{i=0}^{j-1} w_i - \sum_{i=j}^{n-1} w_i,\ b_j = \sum_{i=j}^{n-1} w_i x_i - \sum_{i=0}^{j-1} w_i x_i \tag{7}$$

The values of k_j and b_j can be computed iteratively as follows.

$$k_0 = -\sum_{i=0}^{n-1} w_i,\ k_{j+1} = k_j + 2w_j,\ 0 \le j < n \tag{8}$$

$$b_0 = \sum_{i=0}^{n-1} w_i x_i,\ b_{j+1} = b_j - 2w_j x_j,\ 0 \le j < n \tag{9}$$

$$k_0 = -k_n,\ b_0 = -b_n \tag{10}$$

It is easy to verify that at the position x_j we have

$$f(x_j) = f_j(x_j) = f_{j+1}(x_j),\ 0 \le j < n \tag{11}$$

It follows that function $f(x)$ has a global minimum if $k_0 < 0$ and function $f(x)$ has a global maximum if $k_0 > 0$. The function $f(x)$ has both a global minimum and a global maximum when $k_0 = 0$.

If the function $f(x)$ has a global minimum, then

$$\begin{aligned}&\min_{x\in R} \left\{\sum_{i=0}^{n-1} w_i|x_i - x|\right\} = \min_{x\in R} f(x)\\ &= \min\{f(x_0), f(x_1), \ldots, f(x_{n-1})\}\end{aligned} \tag{12}$$

If the function has a global maximum, then

$$\begin{aligned}&\max_{x\in R} \left\{\sum_{i=0}^{n-1} w_i|x_i - x|\right\} = \max_{x\in R} f(x)\\ &= \max\{f(x_0), f(x_1), \ldots, f(x_{n-1})\}\end{aligned} \tag{13}$$

Therefore, the formulas (3) and (4) are established.

The proof is completed. ■

In the following we describe a linear time algorithm for finding the global minimum of the function $f(x)$. This algorithm is based on an extension of the well-known median-finding algorithm.

Algorithm 1 Binary-Search(x, w)

```
first ← 0, last ← n, k ← k_0/2
while last − first > 1 do
    mid ← (first + last)/2
    Select(x, w, first, mid, last)
    m ← k + Σ_{i=first}^{mid−1} w_i
    if m < 0 then
        k ← m, first ← mid
    else
        last ← mid
    end if
end while
return first
```

The inputs of the above algorithm Binary-Search are n real numbers $x_0, x_1, \cdots, x_{n-1}$ and the associated n positive weights $w_0, w_1, \cdots, w_{n-1}$.

III. The Correctness and Complexities

The correctness of the above algorithm Binary-Search is based on the following Theorem.

Theorem 1: If all the weights $w_0, w_1, \ldots, w_{n-1}$ are positive, then a necessary and sufficient condition for the function $f(x)$ in formula (2) achieves its global minimum $f(x_j) = k_j x_j + b_j$ at the position x_j, is $k_j < 0$ and $k_{j+1} \ge 0$, where $k_j,\ 0 \le j < n$, are computed by (8).

Proof. Since all the weights are positive, we have $k_0 = -\sum_{i=0}^{n-1} w_i < 0$. Therefore function $f(x)$ has a global minimum. It is easy to see from the formula (8) for computing k_j that $0 > k_0 < k_1 < \cdots < k_{n-1} > 0$. The function $f(x)$ is a convex piecewise linear function. Therefore, a local minimum of $f(x)$ is also a global minimum of $f(x)$.

The left and right derivatives of $f(x)$ are given respectively by

$$f'_-(x) = \sum_{i:x_i>x} w_i - \sum_{i:x_i\le x} w_i \tag{14}$$

$$f'_+(x) = \sum_{i:x_i\ge x} w_i - \sum_{i:x_i<x} w_i \tag{15}$$

Since the function $f(x)$ is convex and piecewise linear, a necessary and sufficient condition for x to solve (2) is:

$$f'_+(x) \leq 0 \ \texttt{and} \ f'_+(x) \geq 0 \tag{16}$$

It is easy to see that at the position x_j, we have $f'_-(x_j) = k_j$ and $f'_+(x_j) = k_{j+1}$. Therefore the necessary and sufficient condition for the function $f(x)$ achieving its global minimum at the position x_j is that the function $f(x)$ changes the sign of its slop k_j at the position x_j. That is, the function $f(x)$ achieves its global minimum $f(x_j) = k_j x_j + b_j$ at the position x_j, if and only if $k_j < 0$ and $k_{j+1} \geq 0$.

The proof is completed. ■

The complexities of the above algorithm Binary-Search can be concluded by the following Theorem.

Theorem 2: The weighted rectilinear min-sum facility location problem for the finite set of points $S = \{x_0, x_1, \ldots, x_{n-1}\}$ of n points with positive weights $w_0, w_1, \ldots, w_{n-1}$ can be solved by the algorithm Binary-Search in optimal $O(n)$ time and $O(n)$ space.

Proof.

In the algorithm Binary-Search, the variables $first$, mid and $last$ are used to represent $begin$, $middle$ and end positions of the search interval respectively.

The variable k is actually $k_{first}/2$ computed by formula (8) for the current search interval $[first, last)$. Similarly the variable m is actually $k_{mid}/2$ computed by formula (8) for the middle position of the current search interval $[first, last)$.

The variables $first$ and $last$ are initialized with number 0 and n respectively. The initial value of the variable k is $k_0/2$.

The algorithm is guaranteed that the global minimum of function $f(x)$ is located in the interval $[first, last)$. In the above algorithm Binary-Search, the median of the sequence $x_{first},\ x_{first+1}, \cdots,\ x_{last-1}$ is chosen by the algorithm Select which is an optimal linear time median-finding algorithm. The sequence $x_{first},\ x_{first+1}, \cdots,\ x_{last-1}$ is partitioned such that the median is located in the position mid.

For all indices $i,\ first \leq i < mid,\ x_i \leq x_{mid}$.

For all indices $j,\ mid < j \leq last,\ x_j \geq x_{mid}$.

Then the value of $m = k_{mid}/2$ is computed in row 5 according to formula (8).

There are two cases to be distinguished.

In the case of $m < 0$, the global minimum of function $f(x)$ is located in the interval $[mid, last)$ by Theorem 1. The interval $[first, mid - 1]$ is discarded and the search interval is reduced to $[mid, last)$.

In the case of $m \geq 0$, the global minimum of function $f(x)$ is located in the interval $[first, mid)$ by Theorem 1. The interval $[mid, last)$ is discarded and the search interval is reduced to $[first, mid)$. In either case the size of search interval is reduced to half.

Let the time complexity of the algorithm Binary-Search is $T(n)$ in the worst case. The algorithm Select can be completed in linear time in the worst case due to Blum, Floyd, Pratt, Rivest and Tarjan [2] (see alternatively [5]). Remaining computation in the while loop of the algorithm can be completed in linear time obviously. Therefore the time complexity of the algorithm $T(n)$ is determined by the following formulas:

$$T(n) = \begin{cases} T(n/2) + O(n) & n > 2 \\ 1 & n \leq 2 \end{cases} \tag{17}$$

The solution for the above recurrence is

$$T(n) = O(n) \tag{18}$$

The space used by the algorithm is obviously $O(n)$.

The proof is completed. ■

IV. Speeding up the Algorithm

A. Quick Pivot Search Algorithm

In the Binary-Search algorithm described above, a linear-time median-finding algorithm is used for choosing the partition pivot element. This is the key point for the algorithm achieving the optimal linear computing time complexity, but the linear-time median-finding algorithm is more time consuming.

1) A Random Pivot Search Algorithm: Another possibility to choose the partition pivot element is to use a randomized version of the pivot element choosing scheme of quick sort, in which pivot elements are chosen at random. With this modification, we can design a random pivot selecting algorithm for finding the global minimum of the function $f(x)$ as follows.

Algorithm 2 Random-Search(x, w)

1: $first \leftarrow 0, last \leftarrow n, k \leftarrow k_0/2$
2: **while** $last - first > 1$ **do**
3: $pivot \leftarrow$ a random element in $[first, last)$
4: $mid \leftarrow partition(first, last, pivot)$
5: $m \leftarrow \sum_{i=first}^{mid-1} w_i$
6: **if** $k + m < 0$ **then**
7: $k \leftarrow k + m, first \leftarrow mid$
8: **else**
9: $last \leftarrow mid$
10: **end if**
11: **end while**
12: **return** $first$

The inputs of the above algorithm Random-Search are n real numbers $x_0, x_1, \cdots, x_{n-1}$ and the associated n positive weights $w_0, w_1, \cdots, w_{n-1}$.

The algorithm is guaranteed that the global minimum of function $f(x)$ is located in the interval $[first, last)$.

The difference between the algorithm Random-Search and the algorithm Binary-Search is their pivot element choosing scheme.

In the algorithm Random-Search, the partition pivot element $pivot$ for the current search interval $[first, last)$ is chosen at random and then the sequence $x_{first},\ x_{first+1}, \cdots,\ x_{last-1}$ is partitioned by the algorithm $partition$. The algorithm $partition$ arranges elements in the range $[first, last)$ such that the elements

less than $pivot$ are before the elements not less than $pivot$. The algorithm $partition$ returns an index mid at the first element not less than $pivot$.

For all indices $i,\ first \le i < mid,\ x_i \le x_{mid}$.

For all indices $j,\ mid < j \le last,\ x_j \ge x_{mid}$.

Then the value of $m = \sum_{i=first}^{mid-1} w_i$ is computed.

There are two cases to be distinguished.

In the case of $k + m < 0$, the global minimum of function $f(x)$ is located in the interval $[mid, last)$ by Theorem 1. The interval $[first, mid - 1]$ is discarded and the search interval is reduced to $[mid, last)$.

In the case of $k + m \ge 0$, the global minimum of function $f(x)$ is located in the interval $[first, mid)$ by Theorem 1. The interval $[mid, last)$ is discarded and the search interval is reduced to $[first, mid)$.

Suppose in either case the size of search interval is reduced to $k(n)$.

The time complexity of the algorithm Random-Search can be analyzed similarly. Let the time complexity of the algorithm Random-Search is $Q(n)$. The algorithm $partition$ can be completed in linear time. Remaining computation inside the while loop of the algorithm can be completed in linear time obviously.

Therefore the time complexity of the algorithm $Q(n)$ is determined by the following formulas:

$$Q(n) = \begin{cases} Q(k(n)) + O(n) & n > 2 \\ 1 & n \le 2 \end{cases} \qquad (19)$$

The solution for the above recurrence is related to function $k(n)$. Since the partition pivot elements are chosen at random in the algorithm Random-Search, the function $k(n)$ can be as large as $n - 1$ in the worst case, which leads to an $O(n^2)$ solution of $Q(n)$.

The time complexity of the algorithm $Q(n)$ is therefore $\Omega(n^2)$ in the worst case, but the probability of there being many bad partitions to achieve this bound is extremely low.

2) The Median-of-3 Search Algorithm and Variations: In the algorithm Random-Search, the partition pivot elements are chosen at random. Several variations of the partition pivot elements choosing scheme may yield significant improvements in the algorithm's performance. For instance, instead of taking a random element as the pivot, the median of the first, middle and last elements of the current search interval is used as the pivot element. The idea is that subfiles are more likely not degenerate. This strategy, called the Median-of-three variant, is very well understood in the case of the quick sort algorithm. Of course, the Median-of-three strategy can also be used in the context of our algorithms for the weighted median problem.

With this modification, we can design a median-of-three pivot searching algorithm for finding the global minimum of the function $f(x)$ as follows.

The algorithm is guaranteed that the global minimum of function $f(x)$ is located in the interval $[first, last)$.

Algorithm 3 Median-of-3-Search(x, w)

```
1: first ← 0, last ← n, k ← k_0/2
2: while last − first > 1 do
3:    pivot ←Median3(first, last)
4:    mid ←Partition(first, last, pivot)
5:    m ← Σ_{i=first}^{mid−1} w_i
6:    if k + m < 0 then
7:       k ← k + m, first ← mid
8:    else
9:       last ← mid
10:   end if
11: end while
12: return first
```

The difference between the algorithm Median-of-3-Search and the algorithm Random-Search is their pivot element choosing scheme.

In the algorithm Median-of-3-Search, the partition pivot element of the current search interval $[first, last)$ is chosen by the algorithm Median3$(first, last)$ which is the median of three elements first, middle and last elements in the search interval $[first, last)$. Then the sequence $x_{first},\ x_{first+1}, \cdots,\ x_{last-1}$ is partitioned by the algorithm Partition$(first, last, pivot)$.

The algorithm Partition$(first, last, pivot)$ arranges elements in the range $[first, last)$ such that the elements less than $pivot$ are before the elements not less than $pivot$.

The algorithm Partition$(first, last, pivot)$ returns an index mid at the first element not less than $pivot$.

For all indices $i,\ first \le i < mid,\ x_i \le x_{mid}$.

For all indices $j,\ mid < j \le last,\ x_j \ge x_{mid}$.

This method produces good partitions in most cases, including the sorted or almost sorted cases that cause the simpler pivoting methods to blow up.

Several improvements of the median-of-3 partition pivot elements choosing scheme can also be applied to improve the algorithm's performance.

A variation of the partition pivot elements choosing scheme is the median-of-$(2t+1)$ scheme. In this scheme, the median of $2t+1$ random elements of the sequence is chosen as pivot element.

Another variation of the partition pivot elements choosing scheme is Tukey's 'ninther' [11], the median of the medians of three samples, each of three elements.

Nevertheless, all these algorithms can not avoid their quadratic behavior in the worst case.

For example, there are sequences that can cause Median-of-3-Search to make many bad partitions and take quadratic time.

For example, Musser's Median-of-3 killer sequence [11], [16] can cause median-of-3-Search to take quadratic time.

Let k be any even positive integer. Musser's median-of-3 killer sequence can be obtained by applying the following permutations to a sorted sequence of length $2k$.

$$\begin{pmatrix} 1,2,3,4,5,\ldots,k-2,k-1,k,k+1,k+2,k+3,\ldots,2k-1,2k \\ 1,k+1,3,k+3,5,\ldots,2k-3,k-1,2k-1,2,4,6,\ldots,2k-2,2k \end{pmatrix}$$

Suppose the median-of-3 killer sequence is stored in an array $a[1..2k]$. The three elements chosen by Median-of-3-Search in the first run will be $a[1], a[k+1]$, and $a[2k]$. The median value chosen as the pivot for partitioning is 2. In the partition, the only elements exchanged are $k+1$ and 2, resulting in

$$\begin{pmatrix} 1,2,3,4,5,\ldots,k-2,k-1,k,k+1,k+2,k+3,\ldots,2k-1,2k \\ 1,2,3,k+3,5,\ldots,2k-3,k-1,2k-1,k+1,4,6,\ldots,2k-2,2k \end{pmatrix}$$

Now the array $a[3..k]$ holds a median-of-3 killer sequence of length $2k-2$. Since this step can be repeated $k/2$ times, for any even integer n divisible by 4, the median-of-3 killer sequence of length n can produce a sequence of $n/4$ partitions into subsequences of length 2 and $n-2$, 2 and $n-4$, ..., 2 and $n/2$. All partitions at each level requires $\Theta(n)$ time. The total time for all levels takes $\Omega(n^2)$ time.

B. A Practical Linear Time Algorithm

The quick pivot search algorithms and their variations have small constant factors and excellent performance for most inputs as described in the next section.

In this section we will modify the quick pivot search algorithms to practical linear time algorithms guaranteed to run in linear time in the worst case, while retaining the small constant factors and excellent performance in average.

The key modification to the quick pivot search algorithm to avoid quadratic worst-case behavior is to limit the sum of the sizes of all partitions generated to some constant times h of the input size.

In the new algorithm, a quick pivot search algorithm is used first. When the sum of the sizes of all partitions generated exceeds h times of the input size then the algorithm switches to the linear time algorithm Binary-Search.

In the algorithm Quick-Search, a variable $limit$ initialized with the input size n is used to limit the sum of the sizes of all partitions. This limit is implemented by subtracting $1/h$ of the length of the subsequence prior to partitioning from the variable $limit$.

In order to facilitate description, we divide the algorithm by stages according to each call of Partition. Suppose in the stage i, the size of the current subsequence be s_i. It is readily seen that

$$n = s_0 > s_1 > \ldots > s_{i-1} > s_i > \ldots > 1 \qquad (20)$$

The condition for the algorithm entering stage k is $s_k/h \le limit$ or equivalently $s_k/h \le s_0 - \sum_{i=1}^{k-1} s_i/h$. Since $s_0 = n$, this condition is equivalent to $\sum_{i=1}^{k} s_i \le hn$.

If there exists an integer k such that

$$\sum_{i=1}^{k} s_i \le hn \quad \texttt{and} \quad \sum_{i=1}^{k+1} s_i > hn \qquad (21)$$

Algorithm 4 Quick-Search(x, w)

```
1: first ← 0, last ← n, limit ← n, k ← k_0/2
2: while last − first > 1 do
3:    if (last − first)/h ≤ limit then
4:       limit ← limit − (last − first)/h
5:       pivot ←Selet-Pivot(first, last)
6:       mid ←Partition(first, last, pivot)
7:       m ← Σ_{i=first}^{mid−1} w_i
8:       if k + m < 0 then
9:          k ← k + m, first ← mid
10:      else
11:         last ← mid
12:      end if
13:   else
14:      return Binary-Search(first, last)
15:   end if
16: end while
17: return first
```

then the algorithm will switch to the linear time algorithm Binary-Search with an input size of s_{k+1} after the stage k.

Suppose in the stage i the algorithm require $S(s_i)$ time in the worst case and the time complexity of the algorithm Binary-Search be $T(n)$, the time complexity of the algorithm Quick-Search $R(n)$ will be

$$R(n) = \sum_{i=0}^{k} S(s_i) + T(s_{k+1}) \qquad (22)$$

Since $S(s_i)$ is dominated by the time complexity of Partition which can be completed in linear time and remaining computation in the while loop of the algorithm can be completed in linear time obviously, we have $S(s_i) = O(s_i)$.

From formula (18) we know $T(s_{k+1}) = O(s_{k+1})$.

Therefore,

$$R(n) = \sum_{i=0}^{k} O(s_i) + O(s_{k+1}) = O\left(\sum_{i=0}^{k+1} s_i\right) \qquad (23)$$

From formula (20) (21) and (23), the time complexity of the algorithm in the worst case is determined by

$$R(n) = O((h+1)n) = O(n) \qquad (24)$$

In other words , the time complexity of the algorithm Quick-Search is linear and thus optimal in the worst case. In the next section we will see that the algorithm Quick-Search is also very practical.

V. Computational Experiments

In this section, we performed a series of experiments on the performance of our presented searching algorithms for the weighted median problem.

We evaluated the algorithms listed in Table 1.

In order to make the comparisons fair, all of the searching algorithms evaluated share the same common partitioning code, derived from the original STL implementation [10].

We evaluated the algorithms using a range of different input sequences, listed in Table 2, designed to test the performance of the algorithms under a range of conditions.

The most common cases of the random and sorted sequences should all be handled with virtually no additional overhead over the algorithms tested. The data sets M3killer and Rotated cause the algorithms based on median-of-3 pivot selection to exhibit worst-case performance, the latter sequence for the particular partitioning algorithm used in many implementations of the STL. The twofaced and organpipe sequences are examples of inputs that require more than the average number of partitioning steps.

The results reported here were obtained on a personal computer with Pentium(R) Dual Core CPU 2.10 GHz and 2.0 Gb RAM, using the Microsoft Visual C++ version 8.0 compilers. The word size of the processor is $w = 32$.

The running times in seconds of the 10 algorithms for the data sets Random, Sorted, Rotated, M3killer, Twofaced and Organpipe are compared in Table 3 through Table 8. In the tables, the entries marked with a "$******$" could not be solved due to insufficient memory or time.

The experiment results show that the techniques suggested in this paper produce good algorithms that provide superior performance over a wide range of inputs.

The 10 algorithms experimented can be divided into three groups.

The algorithm Binary-Search is in the first group. In this group, the necessary and sufficient condition for the function $f(x)$ achieving its global minimum is exploited and a linear-time median-finding algorithm is used to design a binary search algorithm achieving the linear computing time complexity. Since the input size of the problem is $O(n)$, the $O(n)$ time algorithm Binary-Search of the first group for the weighted median problem is obviously optimal. From Table 3 to Table 8 we can also see that the algorithm Binary-Search exhibits its linear time costs for all data sets.

The algorithms Random-Search, Med3-Search, Med3-Random, Medt-Search-2, Medt-Search-3 and Med3t-Search are in the second group. In this group, a quick sort like pivot element choosing scheme is used to choose the partition pivot element. From Table 3 to Table 8 we can see that the quick pivot search algorithms in this group have small constant factors and excellent performance for most inputs. Nevertheless, all these algorithms can not avoid their quadratic behavior in the worst case for some inputs such as the data sets M3killer and Rotated.

The third group includes the algorithms Quick-Search, Quick-Random and Quick-Med3t. They are all hybrid algorithms of the linear time algorithm Binary-Search and the quick pivot search algorithm with variant pivot search schemes. In our experiment the tuning constant 4 is chosen for the parameter h used in the algorithms in the third group. These algorithms combine the merits of the algorithms in the first two groups. They not only have good performance for all inputs but also provide insurance against the quadratic behavior of the algorithms. As expected, in our experiments the algorithm Quick-Random performs best for most of our test data sets and it is also an optimal algorithm in the worst case.

VI. Concluding Remarks

We have suggested new techniques for solving the weighted median problems, and shown that they provide significant performance improvements. Our techniques of the algorithms in the third group can be viewed as a hybrid between always choosing pivots randomly, versus always using a deterministic technique. The number of random pivot selections is determined adaptively,responding to how ill-behaved the input is. Doing so will save the time for random pivot selection and not sacrifice performance on best-case inputs.

In our experiments the rotated sequence is an exceptionally difficult input; this is somewhat surprising since it is very nearly already sorted. As noted previously, this behavior only occurs for the specific partitioning algorithm used. A variation of this algorithm that does not induce worst-case performance of the rotated sequence sorts the first, middle, and last elements of the array in the first, second, and last positions, and places the pivot element in its correct sorted position following partitioning [14]. Using this partitioning scheme in our experiments resulted in a slightly worse performance when sorting already sorted or M3killer inputs, but better performance on other inputs.

We chose the tuning constant $h = 4$ used in the implementation of the algorithms in third group more or less arbitrarily; future work should experiment to determine what the optimal settings are.

Quick-Search like algorithms in the third group provide insurance against quadratic behavior; the techniques described in this paper allow this insurance to be obtained at practically no cost, and in addition improve performance in instances that are not necessarily worst case, but nonetheless are worse than average (e.g., the organpipe sequence).

The computational experiments in Section 3 demonstrate that the achieved results are not only of theoretical interest, but also that the techniques developed may actually lead to considerably faster algorithms.

References

[1] J.Bleiholder, F.Naumann, Data Fusion, ACM Comput. Surv. 41 (2008)1-41.

[2] M.Blum, R.Floyd, V.Pratt, R.Rivest, R.Tarjan, Time bounds for selection, J. Comput. Syst. Sci. 7 (1973)448-461.

[3] N.Boland, P.D.Marin, S.Nickel, J.Puerto, Exact procedures for solving the discrete ordered median problem, Comput. & Oper. Res. 33 (2006)3270-3300.

[4] M.Chrobak, C.Kenyon, N.Young, The reverse greedy algorithm for the metric k-median problem, Inf. Process. Lett. 97 (2006)68-72.
[5] T.H.Cormen, C.E.Leiserson, R.L.Rivest: *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, 1990.
[6] T.Drezner, Z.Drezner, C.H.Scott, Location of a facility minimizing nuisance to or from a planar network, Comput. & Oper. Res. 36 (2009)135-148.
[7] R.Z.Farahani, Z.Drezner, N.Asgari, Single facility location and relocation problem with time dependent weights and discrete planning horizon, Ann. Oper. Res. 167 (2009)353-368.
[8] S.L.Hakimi, Optimum locations of switching centers and the absolute centers and medians of a graph, Oper. Res. 12 (1964)450-469.
[9] S.L.Hakimi, Optimum distribution of switching centers in a communication network and some related graph-theoretic problems, Oper. Res. 13 (1965)462-475.
[10] Lijun Lun,Xin Chi,Xuemei Ding, Edge Coverage Analysis for Software Architecture Testing, Journal of Software ,7(5), (2012)1121-1128.
[11] D.R.Musser, Introspective sorting and selection algorithms. Softw.łPract. and Exp., 27, 983C993, 1997.
[12] A.P.Punnen, Y.P.Aneja, On k-sum optimization, Oper. Res. Lett. 18 (1996)233-236.
[13] K.Rosen, Location theory, In S. Louis Hakimi editor, *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, New York, 2000.
[14] R.Sedgewick, Implementing quicksort programs. CACM 21 (1978)847C856.
[15] A.Tamir. Sorting weighted distances with applications to objective function evaluations in single facility location problems. Oper. Res. Lett. 32 (2004)249-257.
[16] J.D.Valois, Introspective sorting and selection revisited. Softw.łPract. and Exp. 30 (2000)617C638.
[17] X. Wang, L. Wang, Y. Wu, Comparative Study of the Algorithms for the Weighted Rectilinear Min-sum Facility Location Problem, Proc. of the 2010 International Conference on Computational Intelligence and Software Engineering ,(2010)1-5.
[18] Jia Wu, Zhihua Cai, Xiaolin Chen, Meng Li, Bin Guo, Weighted Clone Selection Algorithm based on Rough Set Theory, Journal of Software , 8(6),(2013)1333-1338.
[19] Jingben Yin,Yutang Liu,Baolin Ma,Dongwei Shi, An Effective Computational Algorithm for a Class of Linear Multiplicative Programming, Journal of Software , 8(1),(2013)110-117.

TABLE I.
SEARCHING ALGORITHMS EVALUATED

Algorithm	Description
Binary-Search	A linear time binary search algorithm for the weighted median problem.
Random-Search	A random pivot selecting algorithm which chooses a random element of the sequence as the pivot element.
Med3-Search	A median-of-three pivot selecting algorithm which chooses the median of three elements first, middle and last elements of the sequence as the pivot element.
Med3-Random	A random median-of-three pivot selecting algorithm which chooses the median of three random elements of the sequence as the pivot element.
Medt-Search-2	A median-of-$(2t+1)$ pivot selecting algorithm with $t=2$ which chooses the median of 5 random elements of the sequence as pivot element.
Medt-Search-3	A median-of-$(2t+1)$ pivot selecting algorithm with $t=3$ which chooses the median of 7 random elements of the sequence as pivot element.
Med3t-Search	Tukey's ninther pivot selecting algorithm which chooses the median of the medians of three samples, each of three random elements of the sequence.
Quick-Search	A hybrid algorithm of the linear time algorithm Binary-Search and the quick pivot search algorithm Random-Search.
Quick-Random	A hybrid algorithm of the linear time algorithm Binary-Search and the random median-of-three pivot selecting algorithm Med3-Random.
Quick-Med3t	A hybrid algorithm of the linear time algorithm Binary-Search and the Tukey's ninther pivot selecting algorithm Med3t-Search.

TABLE II.
DATA SETS USED TO EVALUATE ALGORITHMS

Sequence	Description
Random	A random permutation of the integers 1 through n.
Sorted	The integers 1 through n in increasing order.
Rotated	A sorted sequence rotated left once.
M3killer	Musser's median-of-3 killer sequence.
Twofaced	Musser's two-faced sequence, obtained by randomly permuting the elements of a median-of-3 killer sequence in positions $4\log n$ through $n/2$ and $n/2+4\log n$ through n.
Organpipe	The integers 1 through $n/2$ in increasing order, followed by $n/2$ through 1 in decreasing order, where n is even.

TABLE III.
COMPARING ALGORITHMS FOR DATA SET RANDOM : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.03	0.17	1.76	17.97
Random-Search	0.00	0.02	0.12	1.78	18.39
Med3-Search	0.00	0.02	0.11	1.09	13.09
Med3-Random	0.00	0.02	0.14	1.00	11.31
Medt-Search-2	0.00	0.02	0.12	0.98	12.07
Medt-Search-3	0.00	0.02	0.11	0.97	12.15
Med3t-Search	0.00	0.03	0.11	1.04	11.09
Quick-Search	0.00	0.03	0.11	1.08	12.99
Quick-Random	0.00	0.02	0.14	1.01	11.25
Quick-Med3t	0.00	0.03	0.09	1.01	11.06

TABLE IV.
COMPARING ALGORITHMS FOR DATA SET SORTED : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.01	0.08	0.84	8.47
Random-Search	0.55	0.50	0.58	1.11	11.42
Med3-Search	0.00	0.01	0.09	0.92	9.28
Med3-Random	0.00	0.02	0.09	0.95	9.50
Medt-Search-2	0.00	0.02	0.09	0.92	9.13
Medt-Search-3	0.00	0.01	0.09	0.97	9.53
Med3t-Search	0.00	0.02	0.09	0.92	9.28
Quick-Search	0.00	0.02	0.09	0.94	9.28
Quick-Random	0.00	0.02	0.09	0.95	9.52
Quick-Med3t	0.00	0.01	0.09	0.94	9.30

TABLE V.
COMPARING ALGORITHMS FOR DATA SET ROTATED : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.01	0.11	1.19	11.78
Random-Search	0.00	0.02	0.11	1.09	10.92
Med3-Search	0.00	0.73	65.32	6500.13	******
Med3-Random	0.00	0.02	0.09	0.95	9.52
Medt-Search-2	0.00	0.02	0.09	0.92	9.13
Medt-Search-3	0.00	0.02	0.11	0.95	9.50
Med3t-Search	0.00	0.02	0.11	0.92	9.30
Quick-Search	0.00	0.02	0.19	1.84	18.33
Quick-Random	0.00	0.01	0.11	0.95	9.52
Quick-Med3t	0.00	0.01	0.09	0.94	9.31

TABLE VI.
COMPARING ALGORITHMS FOR DATA SET M3KILLER : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.03	0.20	2.00	19.39
Random-Search	0.53	0.02	0.09	1.01	10.54
Med3-Search	0.00	0.38	32.68	3254.69	******
Med3-Random	0.00	0.02	0.13	1.20	11.86
Medt-Search-2	0.00	0.01	0.13	1.45	12.49
Medt-Search-3	0.00	0.02	0.11	1.29	9.81
Med3t-Search	0.00	0.02	0.09	1.04	9.87
Quick-Search	0.00	0.03	0.25	2.51	24.96
Quick-Random	0.00	0.02	0.13	1.22	1.89
Quick-Med3t	0.00	0.01	0.09	1.03	9.91

TABLE VII.
COMPARING ALGORITHMS FOR DATA SET TWOFACED : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.03	0.17	2.25	18.56
Random-Search	0.50	0.52	0.20	1.26	15.76
Med3-Search	0.00	0.06	0.59	6.63	77.83
Med3-Random	0.00	0.01	0.13	1.29	10.15
Medt-Search-2	0.00	0.02	0.13	1.51	12.14
Medt-Search-3	0.00	0.02	0.11	1.50	11.12
Med3t-Search	0.00	0.02	0.09	1.42	10.55
Quick-Search	0.00	0.03	0.23	2.89	29.44
Quick-Random	0.00	0.02	0.12	1.29	10.14
Quick-Med3t	0.00	0.02	0.09	1.42	10.55

TABLE VIII.
COMPARING ALGORITHMS FOR DATA SET ORGANPIPE : IN SECONDS

Size (n)	100	1000	10000	100000	1000000
Binary-Search	0.00	0.03	0.19	1.67	16.65
Random-Search	0.00	0.50	0.11	1.54	11.22
Med3-Search	0.00	0.03	0.31	4.98	51.04
Med3-Random	0.00	0.02	0.09	0.95	9.59
Medt-Search-2	0.00	0.03	0.14	1.45	14.51
Medt-Search-3	0.00	0.03	0.14	1.36	13.60
Med3t-Search	0.00	0.03	0.11	1.14	11.48
Quick-Search	0.00	0.03	0.27	2.73	27.72
Quick-Random	0.00	0.01	0.09	0.97	9.59
Quick-Med3t	0.00	0.01	0.13	1.15	11.42