MATHEMATICS OF OPERATIONS RESEARCH
Vol. 8, No. 4, November 1983
Printed in U.S.A.

THE WEIGHTED EUCLIDEAN 1-CENTER PROBLEM*†

NIMROD MEGIDDO

Tel Aviv University

We present an $O(n(\log n)^3(\log\log n)^2)$ algorithm for the problem of finding a point (x, y) in the plane that minimizes the maximal weighted distance to a point in a set of n given points. The algorithm can be extended to higher dimensional spaces. For any fixed dimension our bound is $o(n^{1+\epsilon})$ for any $\epsilon > 0$.

1. Introduction. The weighted Euclidean 1-center problem is defined in the plane as follows. Given are n points $(x_1, y_1), \ldots, (x_n, y_n)$ together with positive weights $w_1, \ldots, w_n$. We wish to find a point (x, y) so as to minimize

$$\max_i \left\{ w_i \cdot \left((x - x_i)^2 + (y - y_i)^2\right)^{1/2} \right\}.$$

The special case where $w_i = 1$, $i = 1, \ldots, n$, was proposed by J. Sylvester in 1857 and amounts to finding the smallest circle that contains all the given points. The general case was introduced by Francis [3]. The most efficient algorithm known to date for the unweighted case is an $O(n \log n)$ algorithm by Shamos and Hoey [10].[1] This algorithm utilizes the data structure so-called "Farthest point Voronoi diagram." It is not clear whether the generalization of this concept into weighted Voronoi diagrams may yield equally efficient algorithms for problems such as the weighted 1-center problem. Previous algorithms and analysis of the unweighted case appeared in Rademacher and Toeplitz [8], Courant and Robbins [1], Smallwood [11], Nair and Chandrasekaran [6] and Elzinga and Hearn [2].

The weighted case is solvable in $O(n^3)$ time assuming that we can solve quadratic equations in constant time. The $O(n^3)$ solution is achieved as follows. The problem can be posed as of finding the minimal value r^* of the variable r such that the balls $B_i(r) = \{(x, y) : (x - x_i)^2 + (y - y_i)^2 \leqslant (r/w_i)^2\}$ have a nonempty intersection. By Helly's Theorem [9] this intersection is nonempty if and only if every three of the balls intersect. We can thus consider triples of balls as follows. Let $r_{ijk} = \min\{r : B_i(r) \cap B_j(r) \cap B_k(r) \neq \emptyset\}$. Then $r^* = \max r_{ijk}$ and the point (x, y) is selected as the unique point in $B_i(r^*) \cap B_j(r^*) \cap B_k(r^*)$ where i, j, k are such that $r_{ijk} = r^*$. The goal of the present paper is to develop an algorithm for this weighted 1-center problem which runs in $O(n(\log n)^3(\log\log n)^2)$ time.

*Received December 16, 1981.
AMS 1980 subject classification. Primary: 90C25; Secondary: 68C05.
OR/MS Index 1978 subject classification. Primary: 185 Facilities/equipment/location.
Key words. Sylvester's problem, weighted 1-center problem, convex piecewise quadratic minimization, parallel computation, multi-parametric computing.
†This research was supported in part by the National Science Foundation under Grants ECS-8121741 and ECS-8218181.

[1] The present author has recently improved this bound. In a paper entitled "Linear-Time Algorithms for Linear Programming in R^3 and Related Problems" (to appear in *SIAM J. Comput.*; also in the proceedings of the 1982 IEEE Symposium on Foundations of Computer Science, pp. 329–338), a linear-time algorithm is presented for the unweighted problem. This algorithm generalizes to any fixed dimension in linear-time (N. Megiddo, "Linear Programming in Linear Time When the Dimension Is Fixed," to appear in *J. Assoc. Comput. Mach.*). The method developed in the latter papers, combined with that of the present one, also allows for an improvement in the weighted Euclidean case and a bound of $O(n \log^2 n)$ is possible in the planar case. Also, the weighted rectilinear 1-center problem can be solved in linear-time in any fixed dimension.

0364-765X/83/0804/0498$01.25

Unlike the notion of Voronoi diagram our methods here do generalize to higher dimensional spaces. Thus they yield bounds which are better than any previously known ones even for the unweighted case in more than two dimensions.

The algorithm is based on maximizing in the set of the weighted distances from a variable point while searching for the optimal point. It utilizes ideas of solving parametrized problems introduced by the author in [4], [5]. The new features here are that we work with quadratic functions and that our problem involves more than just a single parameter.

2. Minimizing convex piecewise quadratic functions. The central problem of the present paper is to minimize the function $\max_i\{w_i \cdot ((x - x_i)^2 + (y - y_i)^2)^{1/2}\}$. This is equivalent to minimizing the function $f(x, y) = \max_i\{w_i^2((x - x_i)^2 + (y - y_i)^2)\}$. Clearly, $f(x, y)$ is convex. This implies that for every y the function $h(x) = f(x, y)$ is convex and also the function $g(y) = \min_x f(x, y)$ is convex. We will first show how to evaluate $g(y)$ at a given y. This evaluation amounts to minimizing a convex piecewise quadratic function of one variable, namely, $h(x)$.

To simplify the notation let $h(x) = \max_i\{a_i x^2 + b_i x + c_i\}$ where $a_i > 0$, $i = 1, \ldots, n$. We note that the minimum of $h(x)$ is attained either at the minimum of one of the defining parabolas or at a point of intersection of two parabolas. We will work under a model where it takes constant-time to find the intersection of two parabolas, i.e., the square-root operation takes constant-time. It is obvious that under such a model $h(x)$ can be minimized in $O(n^2)$ time, namely, by identifying in advance all the intersections of two parabolas and then searching for the minimum. We will however develop an $O(n \log n \log\log n)$ algorithm for minimizing $h(x)$. The method is an extension of a basic principle presented by the author in [4] and further developed in [5].

Let x_0 denote the minimizer of $h(x)$. We first observe that given any x it is easy to recognize whether $x < x_0$, $x = x_0$ or $x > x_0$. This is achieved by observing the derivative, or the one-sided derivatives $h'_+(x), h'_-(x)$ of h at x. Formally, we evaluate $h(x)$ and consider the set $I = \{i : a_i x^2 + b_i x + c_i = h(x)\}$. The one-sided derivatives are given by $h'_+(x) = \max\{2a_i x + b_i : i \in I\}$ and $h'_-(x) = \min\{2a_i x + b_i : i \in I\}$. If $h'_+(x) < 0$ then $x < x_0$. If $h'_-(x) > 0$ then $x > x_0$. Otherwise $h'_-(x) \leqslant 0 \leqslant h'_+(x)$ in which case $x = x_0$. These rules enable us to search for x_0. We will work as if we had to find the maximum (with respect to i) of the numbers $a_i x_0^2 + b_i x_0 + c_i$ even though x_0 will not be known exactly. Typically, we will have an interval $[e', e'']$ which contains x_0 and such that all the comparisons between two parabolas made so far yield the same outcome for all values of x in that interval. If the outcome of the following comparison between two parabolas is not uniform over the current interval then this interval will have to be updated. Specifically, suppose we need to decide which is the larger between $a_i x_0^2 + b_i x_0 + c_i$ and $a_j x_0^2 + b_j x_0 + c_j$ while x_0 is known to belong to $[e', e'']$. We then solve the equation $a_i x^2 + b_i x + c_i = a_j x^2 + b_j x + c_j$ and obtain at most two solutions e_1, e_2 in the interval, $e' \leqslant e_1 \leqslant e_2 \leqslant e''$. Note that if there is no solution in the interval then it is obvious how to find which of the two parabolas is above the other one over the interval. The function $\max(a_i x^2 + b_i x + c_i, a_j x^2 + b_j x + c_j)$ coincides with one of the parabolas over each of the subintervals $[e', e_1], [e_1, e_2], [e_2, e'']$. We therefore compute the one-sided derivatives of $h(x)$ at e_1 and e_2 and that enables us to find out which of the subintervals contains x_0. We then update our interval, i.e., we rename the appropriate subinterval $[e', e'']$ and now the maximum between the two remains in the form of a parabola (rather than a piecewise quadratic function) over the new interval which is known to contain x_0.

We have thus found an alternative way of minimizing $h(x)$ in $O(n^2)$. This is by using a simple algorithm for finding the maximum of n numbers. This requires $n - 1$ comparisons. In our case we will compare parabolas instead of numbers. Each

comparison of the maximum-finding algorithm will be translated to solving a quadratic equation (i.e., finding the intersection of two parabolas) followed by evaluation of one-sided derivatives of $h(x)$ and updating the interval. This process will eventually yield an interval which contains x_0 such that the maximum envelope of the collection of parabolas coincides with one of the parabolas over that interval. The determination of x_0 is then straightforward. In this procedure each comparison of the original maximum-finding algorithm corresponds to a step which requires linear-time in the generalized problem of maximizing over the set of parabolas. This establishes the $O(n^2)$ bound. We will however improve this bound to $O(n \log n \log\log n)$ by using a good parallel algorithm for maximum-finding along the lines presented in [5].

The improved procedure for minimizing $h(x)$ is based on a parallel maximum-finding algorithm by Valiant [12]. This algorithm consists of $O(\log\log n)$ steps where during a single step each of n parallel processors executes one comparison. Valiant's algorithm works as follows. Given the set C_i of the candidates for the maximum after i steps, we determine the n comparisons to be made during step $i+1$ according to a graph G_i on $|C_i|$ vertices with the following properties: The graph G_i has no more than n edges and they form cliques of almost equal sizes, i.e., the cardinality difference between any two cliques is at most one. By comparing every two members of a clique only one candidate for the maximum will remain after step $i+1$ in each clique.

Using Valiant's parallel maximum-finding algorithm we can do the following for minimizing $h(x)$. We will run in $O(\log\log n)$ steps corresponding to the steps in Valiant's algorithm. During a single step we will find the intersections of n pairs of parabolas and thus obtain at most $2n$ critical values in the current interval: $e' = e_0 \leqslant e_1 \leqslant \cdots \leqslant e_{2n} \leqslant e_{2n+1} = e''$. Now, in order to proceed to the next round of comparisons we first need to know the exact outcome of each comparison at $x = x_0$. This amounts to locating x_0 in a subinterval $[e_i, e_{i+1}]$, $0 \leqslant i \leqslant 2n$. This search for x_0 can be carried out as a binary search where the test at any critical value e_j amounts to computing the one-sided derivatives of $h(x)$ at e_j. This search can hence be carried out in $O(n \log n)$ time. Since we have $O(\log\log n)$ steps like this the $O(n \log n \log\log n)$ bound follows.

3. Parallel minimization of $h(x)$. The analysis of the preceding section was with respect to a fixed value of y. Thus, the minimizer x_0 is in fact a function of y, $x_0 = x_0(y)$. Our ultimate goal is to minimize $f(x, y)$ with respect to both x and y. We shall thus have to develop an algorithm for minimizing $g(y) = f(x_0(y), y)$. To that end we will apply the basic idea of using parallelism for efficiently searching for the correct value of a parameter (i.e., the variable y in our case). We therefore need a good parallel algorithm for minimizing $h(x)$. The variables of a program for finding $x_0(y)$ will be functions of y. We wish to have a parallel algorithm where the depth, in terms of the number of operations in which y is involved, is minimized.

Suppose that we employ n processors. They will work in $O(\log\log n)$ phases. Consider a single phase. After each processor has produced its respective critical value of x we perform a binary search which requires $O(\log n)$ evaluations of h and its one-sided derivatives at critical values. A single evaluation takes $O(\log\log n)$ parallel time by n processors. Thus, the minimization of $h(x)$ (in other words, evaluating $x_0(y)$ at a given y) is carried out by n processors in $O(\log n(\log\log n)^2)$ parallel time.

As a matter of fact, this bound can be improved if we employ $n \log n$ processors. This is done as follows. Again, we run in $O(\log\log n)$ phases. A phase starts with the parallel calculation of $n \log n$ critical values of x, one by each processor. Next, these values are sorted by the $n \log n$ processors in $O(\log n)$ time, using Preparata's parallel sorting scheme [7]. We then select $\log n$ critical values equally spaced in the set of

critical values. Formally, if the critical values are $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_{n \log n}$ then we select $e_{n/2}, e_{3n/2}, \ldots$. We allocate n processors to each e_i of the selected values and evaluate $h(e_i)$ as well as the one-sided derivatives in $O(\log\log n)$ time. We can then identify an index j such that $e_{(2j-1)n/2} \leqslant x_0 \leqslant e_{(2j+1)n/2}$. Again, we select $n \log n$ critical values from those between $e_{(2j-1)n/2}$ and $e_{(2j+1)n/2}$ and repeat the same idea of parallel evaluation of the derivatives followed by a reduction of the set of critical values by a factor of $\log n$. This is repeated until we find an index k such that $e_k \leqslant x_0 \leqslant e_{k+1}$. The number of rounds required to reach this situation is

$$\log(n \log n)/\log(\log n),$$

i.e., $O(\log n/\log\log n)$. Each takes $O(\log\log n)$ time and hence a single phase runs in $O(\log n)$ time. Thus, the entire procedure for minimizing $h(x)$ on $n \log n$ processors takes $O(\log n \log\log n)$ time.

4. The doubly-parametrized algorithm. We now turn to the variable y and our aim is to find the optimal value y^*, i.e., to minimize $g(y)$. The search for y^* will again be based on the fundamental method of running a parametric version of an algorithm which evaluates g, while maintaining an interval which is known to contain the optimal value y^*. We have already developed both serial and parallel algorithms for evaluating $g(y)$. However, we still need a mechanism that will tell us whether a given value of y is greater than, equal to, or less than y^*. This requires some analysis of the one-sided derivatives of the function g.

4.1. *On the derivative of* $g(y)$. We will describe two methods for distinguishing among the cases $y_0 < y^*$, $y_0 = y^*$ and $y_0 > y^*$ when y_0 is any given value of y.

Method A. Given y_0, we evaluate $g(y_0)$ in $O(n \log n \log\log n)$ time and obtain the optimal value $x_0 = x_0(y_0)$ of x (see §2). We now consider the set I_0 of indices at which the maximum is attained, i.e., $I_0 = \{i : w_i^2((x_0 - x_i)^2 + (y_0 - y_i)^2) = f(x_0, y_0)\}$. Let (ξ_i, η_i) denote the vector difference between (x_i, y_i) and (x_0, y_0), i.e., $(\xi_i, \eta_i) = (x_i - x_0, y_i - y_0)$. Clearly, the function $f(x, y)$ decreases in the direction of a vector (u, v) if and only if $\xi_i u + \eta_i v > 0$ for all $i \in I_0$. Thus, $y^* < y_0$ if and only if there exist u and v such that $v < 0$ and $\xi_i u + \eta_i v > 0$ for all $i \in I_0$. This is equivalent to the existence of u such that $\xi_i u > \eta_i$ $(i \in I_0)$. A necessary and sufficient condition for this to hold is that

$$\max\{\eta_i/\xi_i : \xi_i > 0, i \in I_0\} < \min\{\eta_i/\xi_i : \xi_i < 0, i \in I_0\} \quad \text{and}$$

$$\max\{\eta_i : \xi_i = 0, i \in I_0\} < 0.$$

Similarly, $y^* > y_0$ if and only if

$$\max\{\eta_i/\xi_i : \xi_i < 0, i \in I_0\} < \min\{\eta_i/\xi_i : \xi_i > 0, i \in I_0\} \quad \text{and}$$

$$\min\{\eta_i : \xi_i = 0\} > 0.$$

The analysis based on these conditions takes linear-time once $x_0(y_0)$ has been computed. However, this method does not seem to generalize efficiently when we work in higher dimensional spaces. Our second method is easier to generalize.

Method B. This second method is based on maintaining the derivatives with respect to y of the variables in the program for minimizing $h(x)$ (equivalently, evaluating $g(y)$). First, consider the critical values of the variable x as being functions of y. A critical value is obtained by solving an equation of the form

$$w_i^2\big((x - x_i)^2 + (y - y_i)^2\big) = w_j^2\big((x - x_j)^2 + (y - y_j)^2\big).$$

The set of pairs (x, y) solving this equation is either a straight line (when $w_i = w_j$) or a

circle. At most two critical values arise. Denote by $e_{ij}(y)$ one of the critical values. In any case it is easy to find the derivative of $e_{ij}(y)$ at y_0. Now, we know that the optimal value x_0 is either a minimum of a parabola or a point of intersection of two parabolas. We will now study the behavior of these two types of a minimum as a function of the parameter y. First, note that the minimum of $h_i(x) = w_i^2((x - x_i)^2 + (y - y_i)^2)$ is independent of y; it occurs at $x = x_i$. On the other hand, if x_0 is at a point of intersection of two parabolas then it does depend on y. Suppose x_0 coincides with a critical value $e_{ij}(y_0)$. It may happen that x_0 coincides with several such critical values $e_{i_1j_1}(y_0), e_{i_2j_2}(y_0), \ldots$. Let $e_{ij}(y)$ be any one of these and consider the function $f_{ij}(y) = f(e_{ij}(y), y)$ in the neighborhood of y_0. The derivative of f_{ij} at y_0 can of course be computed in constant time. However, for our purposes we need only to know the signs of the derivatives $f'_{ij}(y_0)$ for those pairs i, j such that $e_{ij}(y_0) = x_0(y_0)$. Specifically, if they are all positive then $y^* > y_0$ and if they are all negative then $y^* < y_0$; otherwise $y^* = y_0$.

4.2. *Minimizing* $g(y)$. As pointed out earlier, the minimization of $g(y)$ will be carried out by running a parametrized version of the parallel algorithm for minimizing $h(x)$. More formally, the variables of the program for minimizing $h(x)$ are themselves functions of the parameter y. In order for these functions to remain tractable we maintain an interval $[d', d'']$ which contains the optimal value y^*. This interval will gradually be reduced so that all the current variables remain quadratic (rather than piecewise quadratic) functions of y over the current interval.

Recall that the minimization of $h(x)$ is carried out in $O(\log\log n)$ phases. During a single phase we produce $n \log n$ critical values of x. After the $n \log n$ critical values of x have been produced, we run a sequence of $O(\log n/\log\log n)$ stages at the end of which we locate x_0 between two consecutive critical values of x. During a single stage we select $\log n$ critical values, evaluate $h(x)$, $h'_+(x)$ and $h'_-(x)$ at each of them and locate x_0 between two critical values which are consecutive in the set of $\log n$ values we have selected.

We will now review the execution of a single phase and examine the role of the parameter y during each of the different steps constituting the phase.

(i) Computing critical values of x. The basic operation here is solving (for x) an equation of the form $w_i^2((x - x_i)^2 + (y - y_i)^2) = w_j^2((x - x_j)^2 + (y - y_j)^2)$. The cases $w_i = w_j$ and $w_i \neq w_j$ are fundamentally different. If $w_i = w_j$ then the solution x is a linear function of y. Suppose $w_i \neq w_j$. Here the solution has the form of a circle: $(x - a_{ij})^2 + (y - b_{ij})^2 = c_{ij}^2$. In this case two critical values of y arise, namely, $b_{ij} \pm c_{ij}$. Specifically, if $|y - b_{ij}| < c_{ij}$ then two critical values of x are produced while if $|y - b_{ij}| \geqslant c_{ij}$ then no critical values of x arise (i.e., the outcome of the comparison between the two parabolas is uniform in x when $|y - b_{ij}| \geqslant c_{ij}$). During this step of producing the critical values of x (as functions of y) we also produce critical values of y. We then narrow down the interval $[d', d'']$ so that it will contain y^* but none of these critical values of y in its interior.

(ii) Sorting the critical values of x. In order for us to be able to select $\log n$ critical values adequately, we will sort the set of critical values of x over an interval which contains y^* and over which the sorting permutation is constant. The sorting operation requires comparisons between critical values. The outcome of a comparison may depend on the value of y. For example, suppose that we need to compare $e_i(y)$ which is defined by $(e_i(y) - a_i)^2 + (y - b_i)^2 = c_i^2$ with $e_j(y)$ which is defined by

$$(e_j(y) - a_j)^2 + (y - b_j)^2 = c_j^2 .$$

Then the critical values of y are determined by the points of intersection of these two circles. The case when a critical value is a linear function of y is similar. The sorting

will be carried out by following Preparata's sorting scheme with $n(\log n)^2$ processors (since we sort $n \log n$ elements). There will be $O(\log n)$ rounds of comparisons. During each round we obtain $O(n(\log n)^2)$ critical values of y (intersections of circles) and then update $[d', d'']$ in $O(n(\log n)^2 \log\log n)$ time. Thus, the sorting step takes $O(n(\log n)^3 \log\log n)$ time.

We now consider the $O(\log n/\log\log n)$ rounds where during each round we test $\log n$ critical values of x and seek to locate x_0 among them. Given a critical value $e_k(y)$ of x, we need to look at the signs of the one-sided derivatives of $h(x)$ at $e_k(y)$. This requires that we first identify the set of maximizing indices:

$$I(e_k(y)) = \{ i : w_i^2((e_k(y) - x_i)^2 + (y - y_i)^2) = f(e_k(y), y) \}.$$

Moreover, we have to find a neighborhood of y^* in which this set is constant.

(iii) Evaluating $I(e_k(y))$. To find the set $I(e_k(y))$, we run a parallel maximum-finding algorithm on the set of functions $f_i(y) = w_i^2((e_k(y) - x_i)^2 + (y - y_i)^2)$. Comparisons between $f_i(y)$'s yield critical values of y and we then search for y^* among them. We will shortly discuss these comparisons but it is already clear that it will take $O(\log\log n)$ rounds of such comparisons, each followed by a binary search that takes $O(n(\log n)^2 \log\log n)$ time, to reduce the interval so that eventually the set $I(e_k(y))$ is constant. Thus, the set $I(e_k(y^*))$ is determined in $O(n(\log n)^2(\log\log n)^2)$ time. We now return to the comparisons between the $f_i(y)$'s. Such a comparison requires that we solve an equation of the form

$$w_i^2((e_k(y) - x_i)^2 + (y - y_i)^2) = w_j^2((e_k(y) - x_j)^2 + (y - y_j)^2).$$

Replacing $e_k(y)$ by a variable ξ, we obtain either a circle or a straight line on which the pair (ξ, y) must lie. On the other hand the relation $\xi = e_k(y)$ determines another circle (or, possibly, a straight line) on which the pair (ξ, y) must lie. The critical values of y arising here are therefore obtained by finding the points of intersection of two circles (or, possibly, a circle and a line or even two lines). Thus, at most two such values are produced per each comparison.

(iv) Evaluating $h'(e_k(y))$. Given an index $i \in I(e_k(y^*))$, a critical value of y is determined by substituting $e_k(y)$ for x in the partial derivative of

$$w_i^2((x - x_i)^2 + (y - y_i)^2)$$

with respect to x and equating to zero. This is simply when $e_k(y) = x_i$, i.e., the value of y where the line $x = x_i$ intersects the circle (or line) which determines $e_k(y)$. We will work on all the selected $\log n$ $e_k(y)$'s simultaneously. Thus, we will produce $O(\log n)$ critical values of y and then search for y^* among them. This search takes $O(n(\log n)^2 \log\log n)$ time and eventually we know the position of $x_0(y^*)$ relative to each of the selected $e_k(y)$'s.

The entire search for x_0 consists of $O(\log n/\log\log n)$ rounds, where during each round it takes $O(n(\log n)^2(\log\log n)^2)$ time to narrow down the interval. Thus, the entire search for x_0 in one phase takes $O(n(\log n)^3 \log\log n)$ time. We conclude that the entire procedure takes $O(n(\log n)^3(\log\log n)^2)$ time.

5. Extension to higher dimensional spaces. Our procedure can be extended to higher dimensional spaces. When passing from R^d to R^{d+1} the $(d+1)$st coordinate serves as a parameter and we search for its optimal value by employing a parallel algorithm for the d-dimensional problem. Let $T_p(d)$ and $T_s(d)$ denote the parallel time and serial time, respectively, required for solving the d-dimensional problem. Then we obtain $T_p(d+1) \leqslant (T_p(d) \cdot \log n) \cdot T_p(d)$ and $T_s(d+1) \leqslant (T_s(d) \cdot \log n) \cdot T_p(d)$; this is by following the parallel algorithm for the d-dimensional problem and during each phase running a binary search which amounts to $O(\log n)$ solutions of the d-

dimensional problem. This is not the best possible construction since it does not utilize the possibility of selecting $\log n$ critical values and testing them simultaneously. However, the improvement by using this idea is relatively small. We thus obtain the following bounds:

$$T_p(d) = O\left((\log n)^{2^d-1}(\log\log n)^{2^{d-1}}\right) \quad \text{and}$$

$$T_s(d) = O\left(n(\log n)^{2^d-1}(\log\log n)^{2^{d-1}}\right).$$

These bounds are of course not very appealing when we think of the problem in a general dimension. However, for a fixed dimension we obtain a bound which is asymptotically less than $n^{1+\epsilon}$ for every $\epsilon > 0$. We note that a brute-force method would yield a bound of $O(n^d)$. Also, no polynomial algorithm is known for the weighted problem in genera! dimension.[2]

[2] R. Chandrasekaran ("The Weighted Euclidean 1-Center Problem," *Oper. Res. Lett.* **1** (1982), 111–112) has pointed out that a polynomial algorithm exists for the problem in general dimension. However, in his model the number of arithmetic operations is bounded by a polynomial function of the sum of logarithms of the coordinates and not only in n and the dimension. The existence of a genuinely-polynomial algorithm (i.e., the number of arithmetic operations bounded by a polynomial in n and the dimension) implies the existence of a genuinely polynomial algorithm for linear programming.

References

1. Courant, R. and Robbins, H. (1941). *What Is Mathematics*?, Oxford University Press, New York.
2. Elzinga, J. and Hearn, D. W. (1972). Geometrical Solutions for Some Minimax Location Problems. *Transportation Sci.* **6** 379–394.
3. Francis, R. L. (1967). Some Aspects of a Minimax Location Problem. *Oper. Res.* **15** 1163–1168.
4. Megiddo, N. (1979). Combinatorial Optimization with Rational Objective Functions. *Math. Oper. Res.* **4** 414–424.
5. ———. (1981). Applying Parallel Computation Algorithms in the Design of Serial Algorithms. Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science, 399–408. To appear in *J. Assoc. Comput. Mach.*
6. Nair, K. P. K. and Chandrasekaran, R. (1971). Optimal Location of a Single Service Center of Certain Types. *Naval Res. Logist. Quart.* **18** 503–510.
7. Preparata, F. P. (1978). New Parallel-Sorting Schemes. *IEEE Trans. Comput.* **C-27** 669–673.
8. Rademacher, H. and Toeplitz, O. (1957). *The Enjoyment of Mathematics*. Princeton University Press, Princeton, New Jersey.
9. Rockafellar, R. T. (1970). *Convex Analysis*. Princeton University Press, Princeton, New Jersey.
10. Shamos, M. I. and Hoey, D. (1975). Closest-Point Problems. Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science, 151–162.
11. Smallwood, R. D. (1965). Minimax Detection Station Placement. *Oper. Res.* **13** 636–646.
12. Valiant, L. G. (1975). Parallelism in Comparison Problems. *SIAM J. Comput.* **4** 348–355.

DEPARTMENT OF COMPUTER SCIENCE, STANFORD UNIVERSITY, STANFORD, CALIFORNIA 94305