Linear Programming in Linear Time When the Dimension Is Fixed

NIMROD MEGIDDO

Tel Aviv University, Tel Aviv, Israel

Abstract. It is demonstrated that the linear programming problem in d variables and n constraints can be solved in $O(n)$ time when d is fixed. This bound follows from a multidimensional search technique which is applicable for quadratic programming as well. There is also developed an algorithm that is polynomial in both n and d provided d is bounded by a certain slowly growing function of n.

Categories and Subject Descriptors: F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations on matrices*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; *sorting and searching*; G.1.6 [**Mathematics of Computing**]: Optimization—*linear programming*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Genuinely polynomial time, multidimensional search, quadratic programming, smallest ball problem, linear time algorithms

1. *Introduction*

The computational complexity of linear programming is one of the questions that has attracted many researchers since the invention of the simplex algorithm [3]. A major theoretical development in the field was Khachian's algorithm [7], which proved that the problem could be solved in time that is polynomial in the sum of logarithms of the (integral) coefficients. This notion of polynomiality is not altogether satisfactory (see [10]), and, moreover, it is not expected that the algorithm will prove more practical than the simplex method. Khachian's result left an open question as to the existence of an algorithm that requires a number of arithmetic operations which is polynomial in terms of the size nd of the underlying matrix (where d is the number of variables and n is the number of constraints). We call such an algorithm genuinely polynomial. This question is closely related to other interesting open questions in the theory of convex polytopes [6] concerning the diameter, height, etc., of polytopes. Obviously, Khachian's results has not advanced our knowledge about these problems.

With the central question still open, it is interesting to investigate special classes of linear programming problems. Systems of linear inequalities with no more than two variables per inequality have been shown by the author [13] to have a genuinely polynomial algorithm. When either n or d is fixed, then the simplex algorithm is

This work was supported in part by the National Science Foundation under Grants ECS-8121741 and ECS-8218181.

Author's present address: Computer-Science Department, Stanford University, Stanford, CA 94305.

of course genuinely polynomial (while Khachian's algorithm is not). However, the crude bound of $O(n^d)$ (assuming $d < n$) means that no matter how slowly the dimension grows, we have a superpolynomial bound. Even the more refined bound of $O(n^{d/2})$ (see [8]) does not help in this respect. Here we develop an algorithm that is genuinely polynomial even if d grows slowly with n.

In this paper we study the complexity of linear programming in a fixed space. In other words, considering linear programming problems of n constraints in R^d, we study the asymptotic time complexity with respect to n for any fixed value of d. Surprisingly, this turns out to be *linear* in n, the coefficient of proportionality rapidly growing with d. We remark that in the worst case any version of the simplex algorithm requires $O(n)$ time *per pivot* and hence is at least quadratic in terms of n even in the two-dimensional case. Our result implies a genuinely polynomial bound for classes of linear programs, where the dimension grows slowly with n, namely, $d = O(\log n/\log\log n)^{1/2})$.

Our results rely on a novel multidimensional search technique developed in the present paper. One version of it may be described roughly as follows: to search relative to n objects in d space, we solve, recursively, two problems each with $n/2$ objects in $(d-1)$ space and then our problem is reduced to one with $n/2$ objects in d space. This should not be confused with Bentley's multidimensional divide-and-conquer [2], where an $n \times d$ problem is reduced to solving two $(n/2) \times d$-problems and then one $n \times (d-1)$ problem.

We note that even though the results of this paper are interesting from the theoretical point of view in the general case, they are very practical for a small number of dimensions. In a previous paper [12] the cases of $d = 2$ and 3 were described, and other classical problems were shown to succumb to the same method of solution. Essentially, some of the common single-facility location problems relative to either the Euclidean or the rectilinear metric are solvable by this technique. Current computational experience with $d = 3$ looks very successful. Direct applications of linear programming in which the dimension is normally small are the following:

(1) *Linear separability.* Given n points in R^d, organized in two disjoint sets, find a hyperplane (if there is one) that separates the two sets. This problem is useful in statistics and in pattern recognition [4, 14].

(2) *Chebyshev approximation* [16]. Given n points $a_i = (a_{i1}, \ldots, a_{id}) \in R^d$ $(i = 1, \ldots, n)$, we wish to find a linear function

$$f(x_1, \ldots, x_{d-1}) = \sum_{j=1}^{d-1} \alpha_j x_j + \alpha_d$$

so as to minimize $\max\{|\sum_{j=1}^{d-1} \alpha_j a_{ij} + \alpha_d - a_{id}| : i = 1, \ldots, n\}$. For other related problems see [18]. We also note that the results can be extended to solve minimization convex quadratic programming problems. For the details of the extension to quadratic programming, as well as other related problems solvable by similar techniques, the reader is referred to [12]. Thus, the problem of finding the smallest ball enclosing n points in R^d can be solved in $O(n)$ time for fixed d.

We start with an overview of the method, given in Section 2. In Section 3 we describe two closely related methods of multidimensional techniques, based on an oracle that can decide the "correct" side of any given hyperplane, relative to the point we are looking for. The applications of these abstract search techniques to linear programming are described in Section 4, where we discuss the linear programming oracle. The final estimations of time efforts are given in Section 5.

2. *An Overview*

We are interested in solving a d-dimensional linear programming problem of n constraints:

$$\text{minimize} \quad \sum_{j=1}^{d} c_j x_j$$

$$\text{so that} \quad \sum_{j=1}^{d} a_{ij} x_j \geq b_i \qquad (i = 1, \ldots, n).$$

It is well known [3] that usually only relatively few constraints are tight at the optimal solution and hence, if we could identify those active constraints, we could solve our problem by solving a set of equations. The fundamental idea of our algorithm is to repeatedly reduce the set of constraints until the problem can be solved directly. For this idea to be successful, we need to drop relatively many constraints simultaneously. Fortunately, there is a way to drop a fixed fraction of the set of constraints regardless of the size of that set.

For the convenience of the presentation, let us transform the problem into the form

$$\begin{aligned} &\text{minimize} && x_d \\ &\text{so that} && x_d \geq \sum_{j=1}^{d-1} a_{ij} x_j + b_i && (i \in I_1), \\ & && x_d \leq \sum_{j=1}^{d-1} a_{ij} x_j + b_i && (i \in I_2), \\ & && \sum_{j=1}^{d-1} a_{ij} x_j + b_i \leq 0 && (i \in I_3), \end{aligned}$$

where $|I_1| + |I_2| + |I_3| = n$. This transformation is always possible. (Set $Z = \sum c_j x_j$; eliminate x_d; rename Z to be x_d.) In other words, our problem is

$$\begin{aligned} &\text{minimize} && x_d \\ &\text{so that} && x_d \geq \max\{\textstyle\sum a_{ij} x_j + b_i : i \in I_1\}, \\ & && x_d \leq \min\{\textstyle\sum a_{ij} x_j + b_i : i \in I_2\}, \\ & && \max\{\textstyle\sum a_{ij} x_j + b_i : i \in I_3\} \leq 0. \end{aligned}$$

It should be noted that the transformation is made only for the convenience of presentation. It merely reflects our consideration of the behavior of constraints in a subspace orthogonal to the gradient of the objective function. Repeated transformations might cause difficulties from the point of view of numerical analysis. However, the algorithm can be programmed with no transformations.

Consider a pair of inequalities with indices i, k in the same set I_ν ($1 \leq \nu \leq 3$). For example, suppose $\{1, 2\} \subset I_1$ and consider the constraints $x_d \geq \sum_{j=1}^{d-1} a_{1j} x_j + b_1$ and $x_d \geq \sum_{j=1}^{d-1} a_{2j} x_j + b_2$. The relationship between these two constraints can be summarized as follows:

(1) If $(a_{11}, \ldots, a_{1,d-1}) = (a_{21}, \ldots, a_{2,d-1})$, then one of the constraints is redundant and can readily be dropped.

(2) If $(a_{11}, \ldots, a_{1,d-1}) \neq (a_{21}, \ldots, a_{2,d-1})$, then the equation

$$\sum_{j=1}^{d-1} a_{1j} x_j + b_1 = \sum_{j=1}^{d-1} a_{2j} x_j + b_2$$

describes a hyperplane in R^{d-1} which divides the space into two domains of domination; namely, on one side of this hyperplane $\sum_{j=1}^{d-1} a_{1j}x_j + b_1 < \sum_{j=1}^{d-1} a_{2j}x_j + b_2$, whereas on the other side the reverse inequality prevails. Thus, if we could tell on what side of this hyperplane the optimal solution lies, then we could drop one of our constraints.

In general, the task of finding out on what side of a hyperplane the solution lies is not so easy, and we are certainly not going to test a hyperplane merely for dropping one constraint. We test relatively few hyperplanes, and our particular method of selecting those hyperplanes enables us to drop relatively many constraints. This seems paradoxical; however, by carefully choosing the hyperplanes we can know the outcomes of tests relative to many other hyperplanes without actually performing those tests. We can summarize the characteristics of the method in general terms as follows. Given n inequalities in d variables, we test a constant number of hyperplanes (i.e., a number that is independent of n but does depend on the dimension) as to their position relative to the optimal solution. A single test amounts to solving up to three $(d-1)$-dimensional linear programs of n constraints. Given the results of the tests for $B \cdot n$ pairs of inequalities (where $0 < B \leq \frac{1}{2}$ is a constant independent of n but dependent on d), we can tell which member of the pair may be dropped without affecting the optimal solution. At this point the problem is reduced to a d-dimensional linear programming problem of $(1-B)\cdot n$ constraints, and we proceed recursively.

For the complete description of the algorithm, we need to specify the following things. We need to explain what we mean by testing a hyperplane and we have to design an algorithm for a single test. Then we have to design the organization of the tests so as to establish a complete linear programming algorithm. These issues are discussed separately in the following sections. In Section 3 we discuss an abstract multidimensional search problem which we believe may also be applicable to problems other than linear programming. The implementation for linear programming is then discussed in Section 4.

3. *A Multidimensional Search Problem*

3.1 THE PROBLEM. Consider first a familiar one-dimensional search problem. Suppose there exists a real number x^* which is not known to us; however, there exists an oracle that can tell us for any real number x whether $x < x^*$, $x = x^*$, or $x > x^*$. Now, suppose we are given n numbers $x_1, \ldots, x_n$, and we wish to know the position of each of them relative to x^*. The question is how many queries we need to address the oracle (and also what is the other computational effort) in order to tell the position of each x_i relative to x^*. Obviously, we may sort the numbers in $O(n \log n)$ time and then perform a binary search for locating x^* in the sorted set. This amounts to $O(\log n)$ queries. An asymptotically better performance is obtained if we employ a linear-time median-finding algorithm [1, 17]. We can first find the median of x_i's, inquire about its position relative to x^*, and then drop approximately one-half of the set of the x_i's and proceed recursively. Thus, one query at the appropriate point suffices for telling the outcome with respect to half of the set. This yields a performance of $O(n)$ time for median finding in sets of cardinalities $n, n/2, n/4, \ldots$ plus $O(\log n)$ queries. It is also important to note that a practical version of this idea is as follows. Pick a random x_i and test it. Then drop all the elements on the wrong side and pick another element from a uniform distribution over the remaining set. This approach also leads to the expected

number of $O(\log n)$ queries and $O(n)$ comparisons. An analysis of essentially the same thing appears in [9]. An even more efficient idea is to select the median of a random sample of three or five.

We now generalize the one-dimensional problem to higher dimensions. Suppose that there exists a point $x^* \in R^d$ which is not known to us, but that there is an oracle that can tell us for any $a \in R^d$ and a real number b whether $a^T x^* < b$, $a^T x^* = b$, or $a^T x^* > b$. In other words, the oracle can tell us the position of x^* relative to any hyperplane in R^d.

Suppose we are given n hyperplanes and we wish to know the position of x^* relative to each of them. The question, again, is how many queries will we need to address the oracle and what is the other computational effort involved in finding out the position of x^* relative to all the n given hyperplanes? The general situation is of course extremely more complicated than the one-dimensional case. The major difference is that in higher dimensions we do not have the natural linear order that exists in the one-dimensional case. It is therefore quite surprising that the result from the one-dimensional case generalizes to higher dimensions in the following way. We show that for every dimension d there exists a constant $C = C(d)$ (i.e., C is independent of n) such that only $C \cdot \log n$ queries suffice for determining the position of x^* relative to all the n given hyperplanes. There is naturally some additional computational effort involved, which turns out to be proportional to n (linear time), the coefficient of proportionality depending on d.

3.2. The Solution. The procedure that we develop here is described recursively. The underlying idea is as follows. We show that for any dimension d there exist constants $A = A(d)$ (a positive integer) and $B = B(d)$ $(0 < B \leq \frac{1}{2})$, which are independent of n, such that A queries suffice for determining the position of x^*, relative to at least Bn hyperplanes out of n given ones in R^d. The additional computational effort is linear in terms of n. Given that this is possible, we can drop Bn hyperplanes and proceed recursively. Each round of A queries reduces the number of remaining hyperplanes by a factor of $1 - B$. Thus, after approximately $\log_{1/(1-B)} n$ rounds we will know the position of x^* relative to all the hyperplanes. Thus the total number of queries is $O(\log n)$. The other computational effort amounts to $cn + c(1 - B)n + c(1 - B)^2 + \cdots$, which is no more than $(c/B) \cdot n$, and hence linear in terms of n.

The constants A and B are derived recursively with respect to the dimension. We already know that when $d = 1$, we can inquire once and drop half of the set of hyperplanes. We may therefore define $A(1) = 1$, $B(1) = \frac{1}{2}$. Henceforth, we assume $d \geq 2$.

Let $H_i = \{x \in R^d : a_i^T x = b_i\}$ $(i = 1, \ldots, n)$ be n given hyperplanes (where $a_i = (a_{i1}, \ldots, a_{id})^T \neq 0$ and b_i is a real number, $i = 1, \ldots, n$). We first note that it will be convenient (for the description of the procedure) to transform the coordinate system. Of course, we cannot transform the unknown point x^*, but we can always transform a hyperplane back to the original system before asking the oracle about the position of the hyperplane. More precisely, if a vector $x \in R^d$ is represented as $x = My$, where M is a $(d \times d)$ nonsingular matrix, then a hyperplane $a^T x = b$ is represented by $(a^T M)y = b$, and we may work with the vectors $a' = M^T a$. If we need to inquire about a hyperplane $a'^T y = b$, then we look at $a^T x = b$, where $a^T = a'^T M^{-1}$.

We would like to choose the coordinate system so that each hyperplane intersects the subspace of x_1 and x_2 in a line (i.e., the intersection should be neither empty nor equal to the entire subspace). Since we have a finite number of hyperplanes, it

is always possible to achieve this situation by applying a nonsingular transformation. Specifically, we would like to have $(a_{i1}, a_{i2}) \neq (0, 0)$ for every i ($i = 1, \ldots, n$). It is possible to find (in $O(n)$ time) a basis for R^d relative to which $a_{ij} \neq 0$ for all i, j. This is based on the observation that $v = v(\epsilon) = (1, \epsilon, \epsilon^2, \ldots, \epsilon^{d-1})^T$ is orthogonal to some a_i for at most $n(d-1)$ values of ϵ. Using this observation, we can successively select our basic vectors so that none is orthogonal to any a_i. Alternatively (just for arguing that the linear-time bound is correct), in order to avoid repeated transformation of this kind, we can say that we can simply ignore those hyperplanes with $a_{i1} = a_{i2} = 0$; if there are too many of them, then we can simply select variables other than x_1 and x_2, and if the same trouble arises with every pair, then the entire situation is extremely simple (i.e., many constraints with only one positive cofficient a_{ij}).

Now, each hyperplane H_i intersects the (x_1, x_2) subspace in a straight line $a_{i1}x_1 + a_{i2}x_2 = b_i$. We define the *slope* of H_i to be the slope of that straight line. More precisely, this slope equals $+\infty$ if $a_{i2} = 0$ and $-a_{i1}/a_{i2}$ if $a_{i2} \neq 0$. We would like at least half of our hyperplanes to have a nonnegative slope and at least half of them to have a nonpositive slope. This can be achieved by a linear transformation of the (x_1, x_2) subspace. In particular, we may find the median of the set of slopes and apply a linear transformation that will take this median slope to zero. The transformation takes linear time and in fact is needed here only for simplicity of presentation. The algorithm can be programmed so that the same manipulations are applied to the original data. Thus, assume for simplicity of presentation that the original coefficients satisfy this requirement.

The first step in our procedure is to form disjoint pairs of hyperplanes where each pair has at least one member with a nonnegative slope and at least one member with a nonpositive slope. We now consider the relationship between two members of a typical pair. Suppose, for example, that H_i has a nonnegative slope, while H_k has a nonpositive slope, and that we have matched H_i with H_k. Assume, first, that the defining equations of H_i and H_k are linearly independent. Let $H_{ik}^{(1)}$ denote the hyperplane defined by the equation

$$\sum_{j=1}^{d} (a_{k1}a_{ij} - a_{i1}a_{kj})x_j = a_{k1}b_i - a_{i1}b_k.$$

This equation is obtained by subtracting a_{i1} times the equation of H_k from a_{k1} times the equation of H_i. Thus, the intersection of H_i, H_k and $H_{ik}^{(1)}$ is $(d-2)$ dimensional. Moreover, the coefficient of x_1 in $H_{ik}^{(1)}$ is zero. This property is essential for our recursive handling of hyperplanes of the form of $H_{ik}^{(1)}$, which takes place in a lower dimensional setting. Analogously, let $H_{ik}^{(2)}$ denote the hyperplane defined by

$$\sum_{j=1}^{d} (a_{k2}a_{ij} - a_{i2}a_{kj})x_j = a_{k2}b_i - a_{i2}b_k.$$

The hyperplane $H_{ik}^{(2)}$ has characteristics similar to those of $H_{ik}^{(1)}$. To understand the significance of $H_{ik}^{(1)}$ and $H_{ik}^{(2)}$, note that if we know the position of x^* relative to both of these hyperplanes, then we can readily tell the position of x^* relative to one of either H_i or H_k. This is illustrated in Figure 1. Note that the intersection of the four hyperplanes is still $(d-2)$ dimensional, and their relative positions are fully characterized by their intersection with the (x_1, x_2) subspace. The fact that H_i and H_k have slopes of opposite signs is essential for our conclusion. If, for example, x^* is known to lie to the "left" of $H_{ik}^{(2)}$ and "above" $H_{ik}^{(1)}$, then we definitely know that it lies "northwest" of H_i.

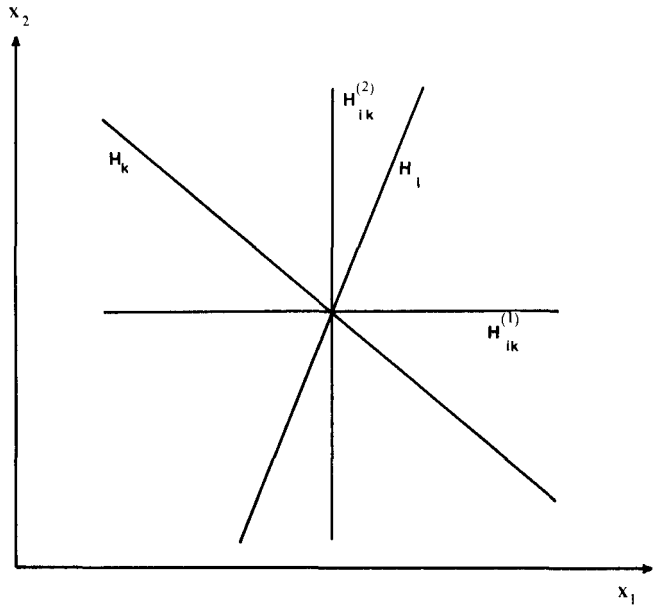

FIGURE 1

For a more precise argument for why this is true, suppose, for example, that H_i has positive slope and $a_{i1} < 0 < a_{i2}$. (All other cases are handled similarly.) Suppose

$$\sum_{j=1}^{d} (a_{k1}a_{ij} - a_{i1}a_{kj})x_j^* < a_{k1}b_i - a_{i1}b_n$$

and

$$\sum_{j=1}^{d} (a_{k2}a_{ij} - a_{i2}a_{kj})x_j^* < a_{k2}b_i - a_{i2}b_n.$$

These relations tell us that x^* lies on certain sides of $H_{ik}^{(1)}$ and $H_{ik}^{(2)}$. Multiplying the former by a_{i2} and the latter by a_{i1} yields (after adding the two) either $\sum_{j=1}^{d} a_{ij}x_j^* > b_i$ or $\sum_{j=1}^{d} a_{ij}x_j^* < b_i$, depending on the sign of $a_{k1}a_{i2} - a_{k2}a_{i1}$. In other words, in this example the side of H_i is known.

If H_i and H_k are linearly dependent, then we claim that $a_{i1} = a_{k1} = 0$. This follows from the fact that H_i and H_k have the same slope, which must be both nonnegative and nonpositive. In this case let $H_{ik}^{(1)}$ be defined as the hyperplane that is parallel to both H_i and H_k and lies in the middle between them. Formally, there is a real number $\lambda \neq 0$ such that $a_{ij} = \lambda a_{kj}$ $(j = 1, \ldots, d)$. The equation which defines $H_{ik}^{(1)}$ is therefore

$$\sum a_{ij}x_j = \frac{1}{2}(b_i + \lambda b_k).$$

Obviously, $H_{ik}^{(1)}$ has a zero coefficient for x_1 in this case too. Moreover, if the position of x^* relative to $H_{ik}^{(1)}$ is known, then we readily know its position relative to one of either H_i or H_k (see Figure 2).

The next step is to consider the hyperplanes $H_{ik}^{(1)}$ (for those pairs (i, k) that we have formed). Since all of them have zero coefficients for x_1, they may be perceived as hyperplanes in the $(d-1)$-dimensional subspace of $(x_2, \ldots, x_d)$. More precisely,

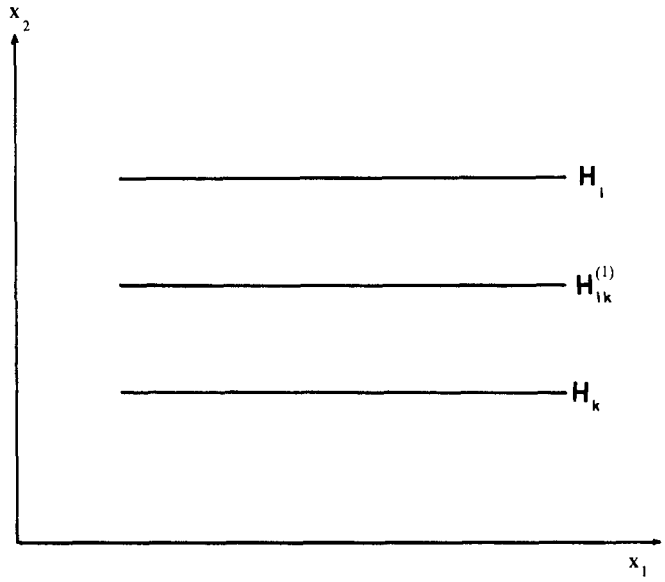

FIGURE 2

we may apply to the collection of these hyperplanes the search procedure that is recursively known for hyperplanes in R^{d-1}. The oracle we have for x^* in R^d may be used as an oracle for the projection of x^* on the subspace of $(x_2, \ldots, x_d)$. More specifically, since $H_{ik}^{(1)}$ has a zero coefficient for x_1, it follows that the projection of x^* on $\{(x_2, \ldots, x_d)\}$ lies on a certain side of the projection of $H_{ik}^{(1)}$ on that space if and only if x^* lies on the corresponding side of $H_{ik}^{(1)}$ in the grand space. There are two different approaches toward applying the recursion (see Figure 3):

Approach I. We may inquire $A(d-1)$ times and obtain the information about the position of x^* relative to $B(d-1)\cdot(n/2)$ hyperplanes of the form $H_{ik}^{(1)}$ (where (i, k) is a pair that we have formed). We now turn to the hyperplanes of the form $H_{ik}^{(2)}$, but here we consider only those pairs (i, k) for which the position of x^* is known relative to $H_{ik}^{(1)}$. Analogously, we can inquire $A(d-1)$ more times and obtain the information about $B(d-1)\cdot B(d-1)\cdot(n/2)$ hyperplanes. At this point we have $(B(d-1))^2(n/2)$ pairs of hyperplanes H_i, H_k such that for each pair we know the position of x^* relative to at least one member of the pair. It therefore follows that we have inquired $2A(d-1)$ times and have obtained the information about a fraction of $\frac{1}{2}(B(d-1))^2$ of the hyperplanes. Thus, we may define $A(d) = 2A(d-1)$ and $B(d) = \frac{1}{2}(B(d-1))^2$. The solution of these recursive equations is simple: $A(d) = 2^{d-1}$ and $B(d) = 2^{1-2^d}$. This implies that within $C(d)\cdot\log n$ queries and an additional effort of $O(F(d)\cdot n)$, we find the position relative to all hyperplanes. The coefficients $C(d)$ and $F(d)$ are of order $2^{O(2^d)}$.

Approach II. Suppose that we recursively find out the position of x^* relative to *all* the hyperplanes $H_{ik}^{(1)}$, $H_{ik}^{(2)}$ (where i, k is a pair that we have formed). Let $Q(n, d)$ denote the number of queries required for n hyperplanes in R^d and let $T(n, d)$ denote the additional effort (i.e., pairing hyperplanes, median finding, etc.) in the same problem. Then, by solving two $(d-1)$-dimensional problems with $n/2$ hyperplanes (i.e., the collection of $H_{ik}^{(1)}$'s and the collection of $H_{ik}^{(2)}$'s), we know the outcome relative to half of the n given hyperplanes. Obviously, $Q(n, d) \leq n$.

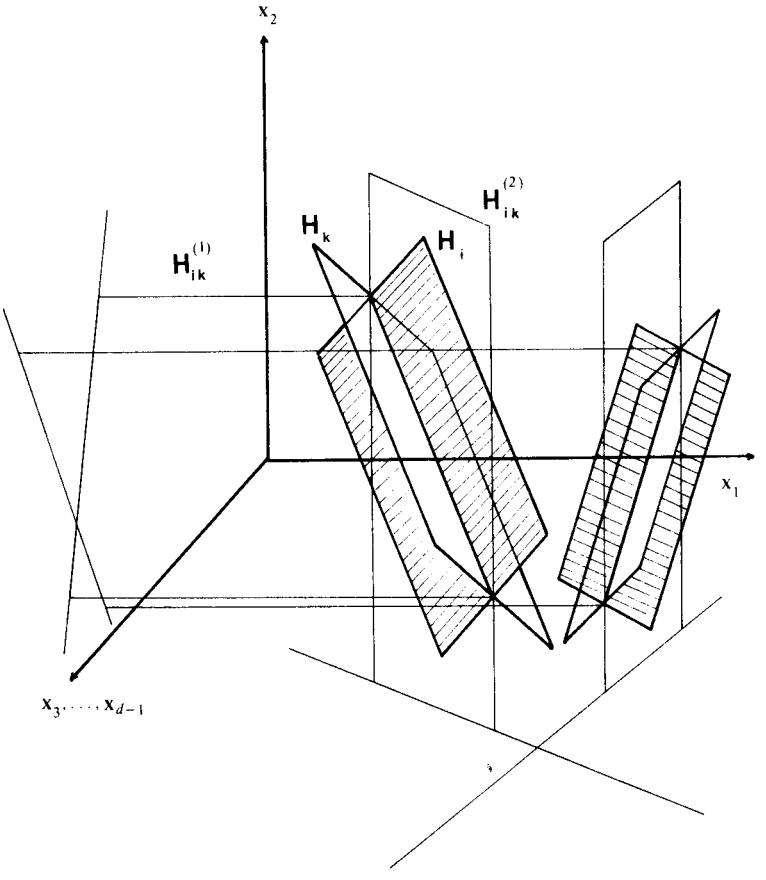

FIGURE 3

We can therefore write down the following recurrence:

$$Q(n, d) = \min\left\{n, 2Q\left(\frac{n}{2}, d-1\right) + Q\left(\frac{n}{2}, d\right)\right\},$$

with boundary conditions $Q(n, 1) = 1 + \lfloor \log_2 n \rfloor$ and $Q(1, d) = 1$. It is easier to solve $Q_1(n, d) = 2Q_1(n/2, d-1) + Q_1(n/2, d)$ with the same boundary conditions and then use the relation $Q(n, d) \leq \min(n, Q_1(n, d))$. However, note that $Q(n, d)$ is not necessarily equal to $\min(n, Q_1(n, d))$; for example, $Q(32, 2) = 26$, $Q_1(32, 2) = 31$.

If $n = 2^L$ where L is an integer, then instead of Q_1 we may consider a recurrence of the form

$$F(L, d) = 2F(L-1, d-1) + F(L-1, d)$$

with boundary conditions $F(L, 1) = L + 1$ and $F(0, d) = 1$. The solution is (see [15] for an interesting solution method):

$$F(L, d) = 2^d \sum_{i=1}^{L+2-d} i\binom{L-i}{d-2} + \sum_{j=0}^{d-2}\binom{L}{j}2^j.$$

It follows that $F(L, d) < (2L)^d/(d-2)!$ so that for fixed d we have $Q(n, d) = O(\log^d n)$ with a surprisingly favorable constant $C = C(d) = 2^d/(d-2)!$.

The other effort involved in preparations for queries satisfies

$$T(n, d) \leq 2T\left(\frac{n}{2}, d-1\right) + T\left(\frac{n}{2}, d\right) + \theta(nd).$$

Consider the recurrence

$$F(L, d) = 2F(L-1, d-1) + F(L-1, d) + 2^L d$$

with boundary conditions $F(0, d) = 1$ and $F(L, 1) = 2^L$. Surprisingly, this turns out to be *linear* in 2^L for every fixed d, with constants $C(d)$ that satisfy $C(d) \leq 2(C(d-1) + d)$. Thus $T(n, d) < d2^d \cdot n$.

It thus appears that the second approach may be more practical than the first, provided n is not extremely large relative to d. These two different approaches to multidimensional search give rise to two different algorithms for the linear programming problem. These algorithms are discussed after we provide the details of the oracle for linear programming.

4. *The Oracle for Linear Programming*

In this section we specify what we mean by testing a hyperplane in R^d. We first define what we require from a procedure for linear programming.

Given a d-variable linear programming problem

$$\text{minimize} \quad \sum_{j=1}^{d} c_j x_j$$

$$\text{so that} \quad \sum_{j=1}^{d} a_{ij} x_j \geq b_i \qquad (i = 1, \ldots, n),$$

we require that the procedure either provide an optimal solution, report that the problem is unbounded, or (in case the problem is infeasible) provide a vector $x' = (x_1', \ldots, x_d')$ which minimizes the function $f(x_1, \ldots, x_d) = \max\{b_i - \sum_{j=1}^{d} a_{ij}x_j : i = 1, \ldots, n\}$. Note that the requirement in case the problem is infeasible is rather unusual; it is needed here for recursive purposes.

The procedure we have developed so far solves this extended notion of a linear programming problem, provided we have a suitable oracle. Specifically, when given a hyperplane $\sum_{j=1}^{d} a_j x_j = b$, we need to know one of the following:

(1) The hyperplane contains the final outcome; that is, either there is an optimal solution on the hyperplane, the problem is unbounded even on the hyperplane, or the hyperplane contains a minimizer of the function f.

(2) The final outcome is on a certain side of the hyperplane. However, in this case we do not expect to know the nature of the final outcome, that is, whether the problem is feasible and bounded.

We have to clarify that such an approach is valid even if the linear programming problem has multiple optima. This follows from convexity of the set of optimal solutions. If the hyperplane does not contain an optimal point (i.e., a point that may be interpreted as x^*), then all optimal points lie on one side of the hyperplane (i.e., the oracle will respond in the same way for all possible choices of x^* in the optimal set).

We now show that the oracle is no more than a recursive application of the master problem (i.e., the extended linear programming problem discussed earlier in this section) in a lower dimensional space.

Given the linear programming problem and the hyperplane, we first consider the same problem with the *equation* of the hyperplane as an additional constraint. This is a d-variable problem of $n + 1$ constraints, but we can eliminate one variable and thus have an equivalent $(d - 1)$-variable problem of n constraints. Moreover, for simplicity of presentation we may assume that the given hyperplane is simply $\{x_d = 0\}$. (Otherwise, apply an affine transformation that takes the given hyperplane to the latter.) Thus the $(d - 1)$-dimensional problem is obtained by dropping the variable x_d. We assume of course that $d \geq 2$; the case $d = 1$ is easy since then the "hyperplane" is a single point. If the problem is feasible and unbounded even when confined to $\{x_d = 0\}$, we are done. Otherwise, we obtain (recursively) either an optimal solution $x^* = (x_1^*, \ldots, x_{d-1}^*, 0)$ (optimality is relative to $\{x_d = 0\}$) or a vector $x' = (x_1', \ldots, x_{d-1}', 0)$ which minimizes the function f on $\{x_d = 0\}$. We now need to determine whether we already have the final outcome or, alternatively, which side of $\{x_d = 0\}$ should be searched. Distinguish two cases:

Case I. First consider the case in which we obtain an optimal solution x^*. We would like to know whether there exists a vector $y = (y_1, \ldots, y_d)$ such that $y_d > 0$, $\sum c_j y_j < \sum c_j x_j^*$ and $\sum a_{ij} y_j \geq b_i$ $(i = 1, \ldots, n)$. Convexity of the feasible domain and linearity of the objective function imply that if a vector y satisfies these requirements, then so does the vector $(1 - \epsilon)x^* + \epsilon y$ for any ϵ $(0 < \epsilon \leq 1)$. It is therefore sufficient to look only in the neighborhood of x^*. Specifically, consider the following $(d - 1)$-dimensional problem:

$$\text{minimize} \quad \sum_{j=1}^{d-1} c_j x_j$$

$$\text{so that} \quad \sum_{j=1}^{d-1} a_{ij} x_j + a_{id} \geq 0 \qquad (i \in I),$$

where $I = \{i : \sum a_{ij} x_j^* = b_i\}$. Here we impose only those constraints that are tight at x^* and we set $x_d = 1$. This is because we are interested only in the local behavior in the neighborhood of x^* and we wish to determine the existence of a *direction* of improvement. It is easy to see that this $(d - 1)$-dimensional problem has a solution with a negative objective-function value if and only if there is a direction with the following property: Moving from x^* in that direction into the half-space $\{x_d > 0\}$, we remain in the feasible domain if the move is sufficiently small (yet positive) while the objective function decreases. Thus, if the problem is infeasible or if, in turn, its optimal value is nonnegative, then there is nothing to look for in that half-space. In the latter case we then consider the other half-space by solving the following problem:

$$\text{minimize} \quad \sum_{j=1}^{d-1} c_j x_j$$

$$\text{so that} \quad \sum_{j=1}^{d-1} a_{ij} x_j - a_{id} \geq 0 \qquad (i \in I).$$

Analogously, the half-space $\{x_d < 0\}$ contains solutions that are better than x^* if and only if this problem has a solution with a negative objective-function value. We note that since x^* is a relative optimum, it follows that at most one of these auxiliary problems can have a solution with a negative objective-function value. If neither has solutions with negative values, then x^* is a global optimum and we are done with the original problem.

Case II. The second case is that in which the problem is infeasible on the hyperplane $\{x_d = 0\}$. Here we obtain a vector $x' = (x', \ldots, x'_{d-1}, 0)$ that minimizes the function $f(x_1, \ldots, x_d)$ (defined above) on $\{x_d = 0\}$. Note that this function is convex so that it is sufficient to look at the neighborhood of x' in order to know which half-space may contain a feasible point. Formally, let $I' = \{i: f(x'_1, \ldots, x'_{d-1}, 0) = b_i - \sum_{j=1}^{d-1} a_{ij}x'_j\}$. We are now interested in the following questions:

Q1. Is there a vector $y = (y_1, \ldots, y_d)$ such that $y_d > 0$ and

$$\sum_{j=1}^{d} a_{ij}y_j > 0 \qquad \text{for} \quad i \in I'?$$

Q2. Is there a vector $z = (z_1, \ldots, z_d)$ such that $z_d < 0$ and

$$\sum_{j=1}^{d} a_{ij}z_j > 0 \qquad \text{for} \quad i \in I'?$$

Note that if the answer to Q1 is in the affirmative, then we proceed into the half-space $\{x_d > 0\}$; if the answer to Q2 is in the affirmative, then we proceed into $\{x_d < 0\}$; if both answers are in the negative, then we conclude that the original problem is infeasible and that x' is a global minimum of the function f.

We answer the above questions in a somewhat tricky way. Consider the following $(d-1)$-dimensional problem:

$$\begin{aligned} &\text{minimize} \quad 0 \\ &\text{so that} \quad \sum_{j=1}^{d-1} a_{ij}y_j \geq -a_{id} \qquad (i \in I'). \end{aligned}$$

If this problem is feasible, then there is a vector y such that $y_d > 0$ and $\sum_{j=1}^{d} a_{ij}y_j \geq 0$ $(i \in I')$. We claim that in this case the answer to Q2 is in the negative. This follows from the fact that if there is also a vector z such that $z_d < 0$ and $\sum_{j=1}^{d} a_{ij}z_j > 0$ $(i \in I')$, then there is a vector x such that $x_d = 0$ and for $i \in I'$, $\sum_{j=1}^{d} a_{ij}x_j > 0$ (x is a suitable convex combination of y and z); and this contradicts the assumption that x' minimizes the function f. Similarly, if there is a vector z such that $\sum_{j=1}^{d-1} a_{ij}z_j \geq a_{id}$ $(i \in I')$, then the answer to Q1 is in the negative. Thus, the procedure in the present case can be summarized as follows. We consider two sets of inequalities in $d-1$ variables:

$$\sum_{j=1}^{d-1} a_{ij}y_j \geq -a_{id} \qquad (i \in I'), \tag{1}$$

$$\sum_{j=1}^{d-1} a_{ij}z_j \geq a_{id} \qquad (i \in I'). \tag{2}$$

If both (1) and (2) are feasible or infeasible, then we conclude that x' is a global minimum of the function f and the original problem is infeasible. If precisely one is feasible, then we proceed into the half-space corresponding to the other one; that is, if (1) is the feasible one, we proceed into $\{x_d < 0\}$, whereas if (2) is the feasible one, we proceed into $\{x_d > 0\}$.

This completes the description of what the oracle does.

We note that the oracle may need to solve three $(d-1)$-dimensional problems of (possibly) n constraints each. However, the number of constraints is usually

much smaller in two of the problems. More precisely, the cardinality of the sets I and I' is not greater than d if we assume nondegeneracy [3]. So, two of the three problems the oracle needs to solve are rather easy.

5. *Conclusion*

If Approach I is used for the multidimensional search, then by solving not more than $3 \cdot 2^{d-1}$ problems of order $n \times (d-1)$, we reduce a problem of order $n \times d$ to a problem of order $n(1 - 2^{1-2^d}) \times d$. Hence the total effort $LP_1(n, d)$ using this approach satisfies

$$LP_1(n, d) \leq 3 \cdot 2^{d-1} LP_1(n, d-1) + LP_1(n(1 - 2^{1-2^d}), d) + \theta(nd).$$

It can be proved by induction on d that for every d there exists a constant $C(d)$ such that $LP_1(n, d) < C(d) \cdot n$. It can further be seen that $C(d) \leq 3 \cdot 2^{2^d+d-2} C(d-1)$, which implies $C(d) < 2^{2^{d+2}}$; that is, $C(d) < 2^{O(2^d)}$. It is interesting to compare this result with the epilogue in [8], where questions are raised with regard to the average complexity of the simplex algorithm as the number of variables tends to infinity while the number of constraints is fixed. The answer is highly sensitive to a probabilistic model to be adopted. Smale [19] has shown that, under a certain model, the average number of *pivot steps* is $o(n^\epsilon)$ for every $\epsilon > 0$ whenever the number of constraints is fixed.

Using Approach II, we reduce a problem of order $n \times d$ to one of order $(n/2) \times d$ by solving $O((2\log(n/2))^d/(d-2)!)$ problems of order $n \times (d-1)$, incurring an additional effort of $O(d2^d n)$. The resulting total effort satisfies

$$LP_2(n, d) \leq c \frac{(2\log(n/2))^d}{(d-2)!} LP_2(n, d-1) + LP_2\left(\frac{n}{2}, d\right) + O(d2^d n).$$

It can be proved that for fixed d, $LP_2(n, d) = O(n(\log n)^{d^2})$, with a rather unusual constant $C(d) < 2^{d^2} / \prod_{k=1}^{d-2} k!$

The argument that the worst case complexity is linear relies heavily on the fact that we can find the median in linear time. However the linear-time, median-finding algorithms [1, 17] are not altogether practical. In practice we would simply select a random element from the set of critical values rather than the median. It appears that the best practical selection is of the median of a random 3-sample or 5-sample. Since this is repeated independently many times, we do achieve expected linear-time performance (where the expectation is relative to our random choices). The analysis is close to that of algorithm FIND (see [9]). Another approach is to employ a probabilistic selection algorithm like that in [5], but, again, it is not required that the exact median be found.

Finally, we remark that a *hybrid* multidimensional search (i.e., picking, recursively, the better between the two approaches whenever a search is called for) may improve the bounds presented here.

ACKNOWLEDGMENTS. Fruitful discussions with Zvi Galil are gratefully acknowledged. The author also thanks the referees for their constructive comments.

REFERENCES

(Note. Reference [11] is not cited in the text.)

1. AHO, A.V., HOPCROFT, J.E., AND ULLMAN, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. BENTLEY, J.L. Multidimensional divide-and-conquer. *Commun. ACM 23*, 4 (Apr. 1980), 214–229.

3. DANTZIG, G.B. *Linear Programming and Extensions.* Princeton University Press, Princeton, N.J., 1963.
4. DUDA, R.O., AND HART, P.E. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.
5. FLOYD, R.W., AND RIVEST, R.L. Expected time bounds for selection. *Commun. ACM 18*, 3 (Mar. 1975), 165–172.
6. GRÜNBAUM, B. *Convex Polytopes.* Wiley, New York, 1967.
7. KHACHIAN, L.G. A polynomial algorithm in linear programming. *Soviet Math. Dokl. 20* (1979), 191–194.
8. KLEE, V., AND MINTY, G.J. How good is the simplex algorithm? In *Inequalities*, vol. 3. Academic Press, New York, 1972, pp. 159–175.
9. KNUTH, D.E. Mathematical analysis of algorithms. In *Information Processing 71.* Elsevier North-Holland, New York, 1972, pp. 19–27.
10. MEGIDDO, N. Is binary encoding appropriate for the problem-language relationship? *Theor. Comput. Sci. 19* (1982), 337–341.
11. MEGIDDO, N. Solving linear programming when the dimension is fixed. Dept. of Statistics, Tel Aviv University, April 1982.
12. MEGIDDO, N. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput. 12*, 4 (Nov. 1983).
13. MEGIDDO, N. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput. 12*, 2 (May 1983), 347–353.
14. MEISEL, W.S. *Computer-Oriented Approaches to Pattern Recognition.* Academic Press, New York, 1972.
15. MONIER, L. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *J. Algorithms 1* (1980), 60–74.
16. RICE, J. *The Approximation of Functions.* Vol. 1; *The Linear Theory.* Addison-Wesley, Reading, Mass., 1964.
17. SCHÖNHAGE, A., PATERSON, M., AND PIPPENGER, N. Finding the median. *J. Comput. Syst. Sci. 13* (1976), 184–199.
18. SHAMOS, M.I. *Computational geometry.* Ph.D. dissertation, Dept. of Computer Science, Yale Univ., New Haven, Conn., 1978.
19. SMALE, S. On the average speed of the simplex method of linear programming. To appear in *Math. Program.*

RECEIVED MAY 1982; REVISED MARCH 1983; ACCEPTED MAY 1983