Excursions into Geometry

by

David P. Dobkin, Richard J. Lipton
& Steven P. Reiss

Research Report #71

CONTENTS

Introduction

The geometry of n dimensions has been a topic of mathematical interest for the last century. Until very recently, however, little work has been done on the application of geometric principles to the study of algorithms or on the study of algorithms for geometric problems. In the research presented here, models are proposed for the study of the complexity of higher dimensional geometry problems. A major goal of these researches is the generation of better lower and upper bounds on commonly performed operations which have a geometric flavor. Typical of such operations are problems of linear and integer programming as well as problems involving the searching of geometric objects in Euclidian spaces. The results presented here comprise a set of research projects undertaken at Yale over the past three years. While each of these reports will eventually appear elsewhere in a format which may differ from that given here, we have gathered all reports together here in order to present a unified study of researches on the problem. The goal of this section will be to present a unified introduction to the research reported here.

The geometric results were derived from studies of the complexity of problems with the flavor of searching in Euclidian spaces and of problems involving properties of convex polytopes and point sets in Euclidian spaces. The former studies are motivated by the integer programming knapsack problems while the latter relate to the complexity of standard linear programming problems. For neither of these problems is an algorithm known of complexity which grows at less than an exponential rate in the size of the problem input. And no lower bound of greater than linear complexity is known for deterministic Turing machine computation which solves either of these problems. Yet each is

of major practical importance.

The knapsack problem can be simply stated as the problem of finding whether a given point lies on any of a set of hyperplanes. We consider linear search tree programs for solving general problems of this type. The model is described in [3] along with methods of obtaining lower bounds on problems of this type. Using these methods, lower bounds of nlogn on the 2-dimensional knapsack problem and $\frac{1}{2}n^2$ on the general problem are given in [1] and [4]. A method of obtaining upper bounds on such problems is given in [2]. This method yields an upper bound of 3logn on the complexity of searching a set of n lines in the plane which have been prepared in a manner which generalizes sorting. Extensions of this method to searching k-dimensional objects in Euclidian n-spaces are also given. This algorithm gives insight into methods of achieving upper bounds on the knapsack problems and into the framework necessary to prove better lower bounds on the problem. This framework is based largely on an understanding of partitions of the plane by lines or partitions of n space by hyperplanes. Properties of such partitions derived from connections with threshold logic lead to the $\frac{1}{2}n^2$ lower bound of [4]. Further study of this problem in the plane, with the intention of fully classifying partitions of the plane by lines is a first step towards a complete understanding of partitions of n-space by hyperplanes. It is only through this approach that better lower bounds on the knapsack problem can be obtained. This first step is undertaken in [5] where all of these concepts are outlined in detail.

A constant problem of interest in operations research is the determination of the complexity of the linear programming problem. We consider the time [7] and space [6] complexity of this problem. All known fast algorithms for this problem involve finding paths for traversing the facets of a convex polytope. We show that any fast (i.e. subexponential) algorithm for this problem must give rise to fast algorithms for a number of problems in the geometry of convex figures and point sets in n-dimensional Euclidian space. Furthermore, all of our reducibilities are 2-way. For example, an algorithm which works in polynomial time to solve the hemisphere problem (i.e. given a set of p points on the unit sphere in E^n, do they share a common hemisphere?) gives rise to a polynomial time algorithm for solving all linear programming problems. Problems of complexity equivalent to linear programming are said to be LP-complete. Because the class of LP-complete problems are closed under complementation and the class of NP-complete problems are conjectured to be not closed under complementation, we present strong evidence that linear programming is mostly likely not NP-complete. Linear programming is, however, shown to be P-complete [6]. That is every problem solvable in polynomial time on a deterministic Turing machine is solvable in no more space than required by the linear programming problem.

Throughout the preparation of these reports, we have received generous support from a number of sources. In particular we would like to thank the National Science Foundation for their support under a graduate fellowship (S.R.) and grants GJ-43157 (D.D.), DCR-74-24193 (R.L.), and DCR-74-12870 (R.L.), the Army Research Office for grant DAHC04-75-G-0037, (R.L.) the Office of Naval Research for grants N00014-75-C-0450 (D.D.&S.R.) and N00014-75-C-0752 (R.L.)

References

[1] D. Dobkin
A non-linear lower bound on linear search tree programs for solving knapsack problems.
JCSS (to appear).

[2] D. Dobkin and R. Lipton
Multi-dimension Searching Problems
SIAM Journal of Computing (to appear).

[3] D. Dobkin and R. Lipton
On the complexity of computations under varying sets of primitive operations in Automata Theory and Formal Languages, Spring-Verlag Lecture Notes in Computer Science, #33.

[4] D. Dobkin and R. Lipton
A lower bound of $\frac{1}{2}n^2$ on linear search tree programs for the knapsack problem.
Mathematical Foundations of Computer Science Conference, Gdansk, Poland, September, 1976 (to appear).

[5] D. Dobkin and R. Lipton
The complexity of searching lines in the plane.

[6] D. Dobkin, R. Lipton and S. Reiss
Linear Programming is P-complete.

[7] S. Reiss and D. Dobkin
The complexity of linear programming.

A non-linear lower bound on linear search tree programs for solving knapsack problems[1]

David Dobkin

Department of Computer Science
Yale University
New Haven, Connecticut 06520

Abstract

By the method of region counting, a lower bound of $n \log_2 n$ queries is obtained on linear search tree programs which solve the n-dimensional knapsack problem. The region counting involves studying the structure of a subset of the hyperplanes defined by the problem. For this subset of hyperplanes, the result is shown to be tight.

[1] Portions of this research were supported by the ONR under Grant N00014-75-C-0450 and NSF Travel Grant DCR75-15624.

1. Introduction

In a previous paper [2], we showed that any linear search tree which determines membership in a union of a disjoint family of k open sets requires at least $\log_2 k$ queries. This result can actually be extended [4] to show that any search tree using queries which are polynomials of degree $\leq p$ requires at least $\frac{1}{p} \log_2 k$ queries to determine membership in a union of a disjoint family of k open sets. In the present paper we will use these results to show that any linear search tree for solving the knapsack problem of dimension n requires at least $n \log_2 n$ queries. This result will follow by showing that a subset of this problem gives rise to at least $\frac{1}{2} n!$ regions. Although the linear search tree model does not appear to have all of the power of the Turing machine model for which the P vs. NP model was first proposed, this non-linear lower bound is derived on the model which is actually used in practice for solving knapsack-type problems. To begin, we review some definitions from [2] to set notation for this paper. The knapsack problem of dimension n is commonly stated as follows: Given an n+1-tuple $\{x_1, \ldots, x_n, b\}$, does there exist a vector $(a_1, \ldots, a_n)$ all of whose components are 0 or 1 such that $\sum_{i=1}^{n} a_i x_i = b$?

We can restate this problem in a geometric fashion by observing that determining whether the n+1-tuple $(x_1, \ldots, x_n, b)$ gives rise to a solvable knapsack problem is equivalent to determining whether the point $(x_1/b, x_2/b, \ldots, x_n/b)$ lies on any of a set of hyperplanes in

R^n, n dimensional Euclidean space. In particular, if $1 \le i \le 2^n-1$ is represented in its binary expansion as $i = i_n 2^{n-1} + i_{n-1} 2^{n-2} + \ldots + i_2 \cdot 2 + i_1$, we can represent the i^{th} of these hyperplanes as $H_i(\underline{v}) - 1 = 0$ where $H_i(\underline{v}) = \sum_{j=1}^{n} i_j v_j$. In all that follows we shall use this second formulation which is trivially equivalent to the first.

The linear search tree model is defined as the set of programs consisting of statements of three types

L_i: if $f(x)$ R 0 then go to L_j else go to L_k
L_m: Halt and accept
L_n: Halt and reject

where $f(x)$ is a linear form in the components of the n-vector x and R is one of the relations >, < or =. This model was first studied by Rabin [3]. This model can be extended to polynomial search trees of degree $\le p$ by allowing $f(x)$ to be a polynomial of degree $\le p$. The result of [2] can then be stated as follows,

Theorem 1: Any linear search tree for determining membership in $\bigcup_{i \in I} A_i$ where each A_i is an open subset of $\mathbb{R}^n$ and the A_i are pairwise disjoint requires at least $\log_2|I|$ queries in the worst case.

and as observed we have the corollary

Corollary: Any polynomial search tree of degree $\le p$ for determining membership in $\bigcup_{i \in I} A_i$ for $\{A_i\}_{i \in I}$ a family of open subsets of $\mathbb{R}^n$ which are pairwise disjoint requires $\frac{1}{p} \log_2|I|$ queries in the worst case.

Our main result will be to show that a subset of the knapsack hyperplanes divide $\mathbb{R}^n$ into at least $\frac{1}{2}n!$ disjoint open sets, so that solving the knapsack problem (i.e. determining membership in these sets) requires at least $n \log_2 n$ queries worst case. The hyperplanes which we will consider are those H_i such that the binary expansion of i has exactly 2 ones in it corresponding to solutions of the problem: given $(x_1, \ldots, x_n, b)$ do there exist $i \ne j$ such that $x_i + x_j = b$? Since the number of regions generated by all the hyperplanes is at least as many as those generated by this subset, this lower bound provides a lower bound on the entire problem. Furthermore, we will show that this lower bound is tight for the problem at hand by demonstrating an appropriate algorithm.

2. A Lower Bound

In this section, we give a characterization of the regions generated by the hyperplanes in the problem under consideration. To begin, we observe that we are seeking to determine how many regions exist which can be expressed as the intersections of halfspaces of the form $x_i + x_j > 1$ or $x_i + x_j < 1$ for $1 \leq i < j \leq n$. We may represent these spaces by $\underline{S}_{ij} = \{ \underline{x} \in \mathbb{R}^n \mid x_i + x_j < 1 \}$ and $\overline{S}_{ij} = \{ \underline{x} \in \mathbb{R}^n \mid x_i + x_j > 1 \}$ for $1 \leq i < j \leq n$. We may define $\mathcal{T}_n$ as the set of all pairs of the form (i,j) for $1 \leq i \neq j \leq n$ and then for each subset of K of $\mathcal{T}_n$; we ask whether

$\delta_K = \bigcap_{(i,j)\in K} \underline{S}_{ij} \cap \bigcap_{(i,j)\in \overline{K}} \overline{S}_{ij}$ is empty or not. There are $2^{\binom{n}{2}}$ such subsets and at first one is tempted to believe that each subset gives rise to an open subset of $\mathbb{R}^n$. However, we may observe that for $n = 4$, if $K = \{(1,2),(3,4)\}$, then

$\delta_K = \underline{S}_{12} \cap \underline{S}_{34} \cap \overline{S}_{13} \cap \overline{S}_{24} \cap \overline{S}_{14} \cap \overline{S}_{23}$ is empty since a point in this set would satisfy $x_1 + x_2 < 1$, $x_3 + x_4 < 1$ as well as $x_1 + x_3 > 1$, $x_2 + x_4 > 1$, a contradiction. Furthermore, it is clear that if K_1 and K_2 are different subsets of $\mathcal{T}_n$ such that δ_{K_1} and δ_{K_2} are non-empty, then these intersections are disjoint open sets. Openness follows since these sets are the intersections of finite collections of open sets and to show that they are disjoint we observe that there is an (i,j) which belongs to K_1 and not K_2 (or to K_2 and not K_1)

so that every point in δ_{K_1} satisfies $x_i + x_j < 1$ and every point in δ_{K_2} satisfies $x_i + x_j > 1$.

Now, we wish to determine conditions of $K \subset \mathcal{T}_n$ such that δ_K is nonempty. We begin with one such set and apply the appropriate permutations to generate at least $\frac{1}{2}n!$ others.

<u>Lemma</u>: If $K = \{(i,j) \mid i + j \leq n + 1\}$, then δ_K is non-empty.

<u>Proof</u>: Let $x_i = \frac{i}{n + 3/2}$ for $1 \leq i \leq n$; then $x_i + x_j < 1$ if and only if $i + j < n + 3/2$ if and only if $(i,j) \in K$. Hence the point $(x_1,\ldots,x_n)$ is an element of K, which is therefore non-empty.

We observe that K is a subset of $\mathcal{T}_n$ consisting of $\left\lfloor \frac{n^2}{2} \right\rfloor$ elements and that i occurs in exactly $n - i$ pairs of K if $i \leq \left\lceil \frac{n}{2} \right\rceil$ and $n + 1 - i$ pairs if $i > \left\lceil \frac{n}{2} \right\rceil$. For each permutation π of n elements we define $\pi(K)$ as $\{(\pi(i),\pi(j) \mid (i,j) \in K\}$ and observe that, for two permutations π_1 and π_2, $\pi_1(k)$ and $\pi_2(k)$ are different if either of the following is not true:

(i) $\pi_1(i) = \pi_2(i)$ if $i \notin \{ \left\lceil \frac{n}{2} \right\rceil, \left\lceil \frac{n}{2} \right\rceil + 1 \}$,

(ii) $\{\pi_1(\left\lceil \frac{n}{2} \right\rceil), \pi_1(\left\lceil \frac{n}{2} \right\rceil + 1)\} = \{\pi_2(\left\lceil \frac{n}{2} \right\rceil), \pi_2(\left\lceil \frac{n}{2} \right\rceil + 1)\}$.

Thus, no three permutations generate the same $\pi(k)$.

<u>Theorem 2</u>: There are at least $\frac{1}{2}$ n! different subsets K of $\mathcal{T}_n$ which give rise to non-empty sets δ_K.

Using this theorem in concert with Theorem 1, we obtain the main result of this paper.

Theorem 3: Any linear search tree for solving the n-dimensional knapsack problem (or even the n-dimensional knapsack problem restricted to solutions of the form $x_i + x_j = b$) must require $n \log n$ - l.o.t. queries in the worst case.

Furthermore, since polynomial queries can be simulated by linear queries, we have

Corollary: Any polynomial search tree of degree $\le p$ requires at least $\frac{1}{p} n \log_2 n$ queries in the worst case to solve the n-dimensional knapsack problem.

3. An Upper Bound

In this section, a method is given to match the lower bound given in the previous section for the restricted knapsack problem mentioned there. Given an input $(x_1, \ldots, x_n, b)$, we wish to determine if there exist distinct integers i and j such that $x_i + x_j = b$. To do so, we may use the following

Algorithm KS2

I. Sort $x_1, \ldots x_n$ to yield a sorted list $y_1, \ldots y_n$ such that for $i > j$, $y_i \ge y_j$.

II. Test to determine if any y_i is $\frac{1}{2}$, if two or more are, halt and accept the input. If one is, drop this element from the list and proceed to Step III. If none is, proceed to Step III.

III. For each i, $1 \le i \le n$, determine the least j_i such that $y_i + y_{j_i} < 1$, if $y_i + y_{j_i - 1} = 1$ halt and accept, otherwise continue.

IV. Halt and reject.

Step I in this algorithm requires $n \log_2 n$ steps, Step II can be done in $\log_2 n$ steps by binary search and Step III can be done in at most $2n$ steps by a merging strategy. Therefore the entire algorithm requires $n \log_2 n$ + l.o.t. steps matching the leading term in the upper bound.

4. Conclusions and Research Problems

The major results of this paper are a lower bound of $n \log n$ queries for the solution of the knapsack problem with a linear search tree and a similar upper bound for the restricted version of the problem under consideration. These results may possibly be improved by considering less restricted versions of the problem. For example, it is reasonable to conjecture that a better lower bound could be obtained by considering the regions generated by all 2^n knapsack hyperplanes rather than merely a subset of $\binom{n}{2}$ such hyperplanes. Unfortunately, Strassen [4] has observed that no better lower bound than $O(n^2)$ can be obtained by applying Theorem 1, as at most $O(2^{n^2})$ regions will be generated by a set of 2^n hyperplanes in $\mathbb{R}^n$. A lower bound of this type is given in [2].

5. References

[1] D. Dobkin and R. Lipton. A Lower Bound of $\frac{1}{2}n^2$ on the Knapsack Problem. In preparation.

[2] D. Dobkin and R. Lipton. On the Complexity of Computations under Varying Sets of Primitive Operations. In the Proceedings of the 2nd GI- Symposium on Automata Theory and Formal Languages, Springer-Verlag, 110-119.

[3] M. Rabin. Proving Simultaneous Positivity of Linear Forms. JCSS 6:639-650, 1972.

[4] V. Strassen, Personal communication May 1975.

Multidimensional Searching Problems

David Dobkin and Richard J. Lipton

Research Report #34

Department of Computer Science
Yale University
New Haven, Connecticut 06520

Abstract

Classic binary search is extended to multidimensional search problems. This extension yields efficient algorithms for a number of tasks such as a secondary searching problem of Knuty, region location in planar graphs, and speech recognition.

An earlier version of some of the results presented here was given in a paper titled "On Some Generalizations of Binary Search" and presented at the Sixth Annual ACM Symposium on the Theory of Computing, Seattle, Washington, April 1974. Portions of the work of the first author were supported by NSF Grant GJ-43157.

1. Introduction

One of the most basic operations performed on a computer is searching. A search is used to decide whether or not a given word is in a given collection of words. Since many searches are usually performed on a given collection, it is generally worthwhile to organize the collection into a more desirable form so that searching is efficient. The organization of the collection--called preprocessing--can be assumed to be done at no cost relative to the cost of the numerous searches.

One of the basic searching methods is the binary searching method (Knuth [1]). For the purposes of this paper we can view binary search as follows:

Data: A collection of m points on a line.

Query: Given a point, does it equal any of the m points?

Binary search, since it organizes the points into a balanced binary tree, can answer this query in $\lfloor \log m \rfloor + 1$ "steps" where a step is a single comparison.* Note the preprocessing needed to form the balanced binary tree is a sort which requires $O(m \log m)$ steps. For the algorithms under consideration here, we will define a step in an algorithm as a comparison of two scalars or the determination of whether a point in 2-dimensional Euclidean space lies on, above, or below a given line. For notational simplicity we will define $g(m)$ as the number of steps necessary to perform a search through a set of m objects. Thus, $g(m) = \lfloor \log m \rfloor + 1$.

This paper generalizes binary search to higher dimensions. Through-

*Throughout this paper all logarithms are taken to base 2.

out it is assumed that data can be organized in any manner desired at no cost. Thus, our cost criterion for evaluating the relative efficiencies of searching algorithms will be the number of steps required to make a single query into the reorganized data.

The search problems considered are specified by a collection of data and a class of queries. These problems include:

1. Data: A set of m lines in the plane.
   Query: Given a point, does it lie on any line?

2. Data: A set of n regions in the plane.
   Query: Given a point, in which region does it lie?

3. Data: A set of m points in the plane.
   Query: Given a new point, to which of the original points is it closest?

4. Data: A set of m lines in n-dimensional space.
   Query: Given a point does it lie on any line?

5. Data: A set of m k-dimensional objects in n-dimensional space.
   Query: Given a point does it lie on any of the objects?

6. Data: A set of m hyperplanes ($n-1$ dimensional objects) in n-dimensional space.
   Query: Given a point does it lie on any hyperplane?

These examples form the basis for some important problems in diverse areas of computer science. Examples 1, 2, and 3 are fundamental to certain operations in computer graphics [2] and secondary searching. In particular example 3 is a reformulation of an important problem discussed by Knuth [1] concerning information retrieval. Examples 4, 5, and 6 are generalizations

of the widely studied knapsack problem.

The main results of this paper are that fast algorithms exist for problems (1)-(6). In particular: problems (1)-(3) have $O(\log m)$ algorithms; problems (4)-(6) have $O(f(n)\log m)$ algorithms where $f(n)$ is some function of the dimension of the space ($f(n)$ is determined more exactly later). The existence of these fast algorithms is somewhat surprising. For instance, lines in the plane (problem 1) are not ordered in any obvious way; hence, it is not at all clear how one can use binary search to obtain fast searches.

II. Basic Algorithm in E^2

All of our fast algorithms are extensions of a fast algorithm for computing the predicate:

$$\exists\, 1 \le i \le m \ [(x,y) \text{ is on } L_i]$$

where $L_1, \ldots, L_m$ are lines and (x,y) is a point in 2-dimensional Euclidean space (E^2) . This predicate merely consists of querying whether a point in the plane lies on any of a given set of lines. Therefore, we begin with a proof that this predicate can be computed in $O(\log m)$ steps.

Theorem 1. For any set of lines $L_1, \ldots, L_m$ in the plane, there is an algorithm that computes $\exists\, 1\le i\le m \ [(x,y) \text{ is on } L_i]$ in $3g(m)$ steps.

Proof. Let the intersections of the lines be given by the points $z_1, \ldots, z_n$ $(n \le \frac{m(m-1)}{2})$ and let the projections of these points onto the x-axis be given by $p_1, \ldots, p_n$. These points define a set of intervals $I_1, \ldots, I_{n+1}$ on the x-axis such that $I_1 = (-\infty, p_1)$, $I_i = (p_{i-1}, p_i)$, $i = 2, \ldots, n$, $I_{n+1} = (p_n, \infty)$ and within the slice of the plane defined by each of these intervals, no two of the original lines intersect. Thus, we can define the relation $<_i$ $(1 \leq i \leq n+1)$ as follows:

$$L_j <_i L_k \text{ if and only if } \forall x \in E^1 [\text{if } p_i \leq x \leq p_{i+1}, \text{ then } L_j(x) \leq L_k(x)] \text{ .}$$

(Note, $L(x)$ is equal to the value of y such that $(x,y) \in L$ and we set $p_0 = -\infty$, $p_{n+1} = \infty$.) By a simple continuity argument it follows that each $<_i$ is a linear ordering on the lines $L_1, \ldots, L_m$. We can thus define a set of permutations $\pi(i,1), \ldots, \pi(i,m)$ such that

$$L_{\pi(i,1)} <_i L_{\pi(i,2)} <_i \cdots <_i L_{\pi(i,m)}$$

for $i = 1, \ldots, n+1$. An algorithm consisting of a binary search into a set of at most $\frac{m(m-1)}{2} + 2$ objects (the points $\{p_i\}$) and a binary search into a set of m objects (the lines $L_{\pi(i,1)}, \ldots, L_{\pi(i,m)}$ for the proper choice of i) requires at most $g(m) + g(\frac{m(m-1)}{2} + 2)$ steps and since g is a monotonically increasing function with $g(m^2) \leq 2g(m)$, this quantity is at most $3g(m)$. Degeneracies which may be introduced into the above algorithm by lines perpendicular to the x-axis may be removed by a rotation of the axes to a situation where no line is perpendicular to the new x-axis. □

Before studying applications of this algorithm to the problems mentioned above, it is worthwhile to examine its structure in more detail. What we have done is to find a method of applying an ordering to a set of lines in the plane. For a set of lines in the plane, no natural ordering exists and thus it is reasonable to assume that any search algorithm which is "global" (i.e. considers the entire plane at once) must use a number of

of steps which grows linearly with the number of lines. The algorithm presented in Theorem 1 defines a set of regions of the plane in which the lines are ordered. In this sense, the algorithm is "local" and the two steps consist of finding the region in which to search and then to do a local search on an ordered set. The orderings are found during preprocessing of the data. The projections of intersection points (i.e. $\{p_i\}$) define the local regions into which the plane can be subdivided and the permutations (i.e. $\pi(i,\cdot)\}$) define the orderings within each of the subdivisions of the plane. Moreover, it is clear that the algorithm not only determines whether the point lies on any line but also between which lines the point lies, if it does not lie on any line. Using this information, we can determine in which region of the plane determined by the given lines the point lies. Thus, we have,

Corollary. Given a set of regions formed by m lines in the plane, we can determine in $3g(m)$ steps in which region a given point lies.

This algorithm forms the basis for what follows. We proceed by studying extensions of this algorithm to higher dimensions and applications of our basic algorithm and its extensions to some interesting problems of computer science.

III. Extensions to E^n

We have seen that searching in a set of m 0-dimensional objects in 1-dimensional space can be done in $g(m)$ steps and that searching in a set of m 1-dimensional objects in 2-dimensional space can be done in $3g(m)$ steps given that the original objects can be preprocessed before any searches are undertaken. In the present section, we extend the search question to seek methods of searching in a set of m k-dimensional objects in n-dimensional spaces. In order to provide a clearer exposition, a series of lemmas will be presented, each of which can be viewed as a generalization of Theorem 1.

Lemma 1. For any set of lines $L_1, \ldots, L_m$ in n-dimensional Euclidean space $(n \geq 2)$, there is an algorithm which computes $\exists\ 1 \leq i \leq m\ [x$ is on $L_i]$ for x a point in E^n in $(n+1)g(m)$ steps.

Proof. The proof is by induction on n and follows from Theorem 1 for $n = 2$. Now, suppose that $n > 2$. It is possible to find a hyperplane H such that none of the lines is perpendicular to H. Projecting the lines onto H yields a set of lines $L_1, \ldots, L_m$ on H and projecting x onto H yields a point x' on H. Furthermore if x lies on L_i then x' lies on L_i'. By the induction hypothesis, we can determine on which lines of the set $\{L_1', \ldots, L_m'\}$, x' lies on, in $ng(m)$ steps. If x' doesn't lie on any L_i', then x doesn't lie on any L_i. And if x' lies on $\{L_{i_1}', \ldots, L_{i_k}'\}$, the lines $L_{i_1}, \ldots, L_{i_k}$ are linearly ordered at x' with respect to the projected co-ordinate and with a logarithmic search we can determine if x lies on any of $\{L_{i_1}, \ldots, L_{i_k}\}$. Since $i_k \leq m$, this search requires at most $g(m)$ steps and m lines in E^n can be searched in $(n+1)g(m)$ steps. □

Lemma 2. For any set of hyperplanes $H_1, \ldots, H_m$ in E^n $(n \geq 2)$, there is an algorithm which determines, for any point x, whether x is on any hyperplane or which hyperplanes it is between in at most $(3 \cdot 2^{n-2} + (n-2))g(m)$ steps.

Proof. Let $h(m,n)$ be the time required to do the search. From Theorem 1, we know that $h(m,2) \leq 3g(m)$ and we will show here that $h(m,n) \leq h(m^2, n-1) + g(m)$. Let K be a hyperplane which is not identical to any of the original hyperplanes. Then, we proceed by forming the set of $n-2$ dimensional objects $J_1, \ldots, J_k$ formed as intersections of pairs of the hyperplanes we considered. Thus, for example $J_1 = H_1 \cap H_2$, $\ldots, J_k = H_{n-1} \cap H_n$ and $k \leq \frac{m(m-1)}{2} < m^2$. From these hyperplanes, we form their projections $J_1', \ldots, J_k'$ on to K. If the point x projects onto x', we can by the induction hypothesis determine in less than $h(m^2, n-1)$ steps in which region of $n-1$ dimensional space x' lies. With respect to each of these regions, the hyperplanes $H_1, \ldots, H_n$ are ordered and can be searched in $g(m)$ steps. Thus, if x' doesn't lie on any J_k', the lemma holds. And, if x' lies on a hyperplane J_i', a search requiring less than $g(m)$ will determine in which region of E^n the point x lies. This proves that $h(m,n) \leq h(m^2, n-1) + g(m)$. Solving this recursion yields $h(m,n) \leq h(m^{2k}, n-k) + kg(m)$ or

$$h(m,n) \leq h(m^{2^{n-2}}, 2) + (n-2)g(m) = 3 \cdot 2^{n-2} + (n-2))g(m)\ . \quad \square$$

Combining the results and proof techniques of Lemmas 1 and 2 yields the following general theorem on searching k-dimensional objects in E^n.

Theorem 2. For any set of k-dimensional objects $\theta_1, \ldots, \theta_m$ in E^n, there is an algorithm that computes $\exists\ 1 \leq i \leq m[x$ is on $\theta_i]$ in $(3 \cdot 2^{k-1} + (n-2))g(m)$ steps for any point x in E^n.

Proof. Let $f(m,k,n)$ be the number of steps required by the search. Then, if $k < n-1$, we can by an argument similar to that used in the proof of Lemma 1 projects the objects onto a hyperplane in E^n and proceeding as there, it is clear that $f(m,k,n) \leq f(m, k, n-1) + g(m)$ if $k < n-1$. Continuing this induction yields $f(m,k,n) \leq f(m, k, k+1) + (n-k-1)g(m)$. Combining this result with the result of Lemma 2 that $f(m, k, k+1) \leq (3 \cdot 2^{k-1} + (k-1))g(m)$ yields the result $f(m,k,n) \leq (3 \cdot 2^{k-1} + (n-2))g(m)$ as in the statement of the theorem where we make use of the identity $h(m,n) = f(m, n-1, n)$.

4. Applications in E^2

Before presenting any applications the basic algorithm must be: (i) examined with respect to preprocessing, (ii) examined with respect to storage requirements, (iii) also extended to a slightly more general case.

Instead of m lines assume that we are given m lines or line segments. The problem is then to search the regions formed by these generalized lines. It is easy to see that the basic algorithm can be adapted here and it operates in time $3g(m)$. Let N be the number of intersection points formed by the m lines. The preprocessing is:

1. Find the N intersection points formed by the m lines.
2. Store these intersection points after they are projected onto the x-axis.
3. For each two adjacent intersection points t_1 and t_2 find the permutation of the m lines in the region $t_1 \leq x \leq t_2$.

Step (1) takes $O(m^2)$ since finding the intersection of two lines consists merely of finding the solution of a simple linear system of equations. Step (2) is a sort of N objects; hence, it takes $O(N\log N)$ time. Finally, Step (3) takes at most $O(m\log m)$ for each of the N regions: to determine the order of the m lines takes at most a sort of m objects. In summary, preprocessing takes

$$O(mN\log m) + O(N\log N) .$$

The storage requirements are easily seen to be: $O(N)$ from Step (1) and $O(mN)$ from Step (3). Thus the total storage needed is seen to be $O(Nm)$.

We will now study two applications of the basic algorithm.

4.1. Planar Graph Search

Suppose that we are given a planar graph G with m edges. How fast can we determine which region of G a new point is in? For example, this location problem is central to the finite element method [3].

Theorem 3. In $O(\log m)$ time and $O(m^2)$ storage, it is possible to determine in which region of a planar graph with m edges a given point lies.

Proof. By an application of Euler's relation [4] the m lines of G can only intersect in $O(m)$ points. Therefore, the basic algorithm--as modified--shows that planar graphs can be searched in $O(\log m)$ time. The preprocessing required is $O(m^2\log m)$; the storage required is $O(m^2)$. □

4.2. Post Office Problem

The post office problem is a search problem for which Knuth [1] states there is no known efficient solution. Suppose that we are given m cities or "post offices." How fast can we determine which post office is nearest to a new point? This is the post office problem. We will now show how to reduce it to the planar search problem of 4.1: Between each post offices x and y construct the line segment ℓ_{xy}. Then construct the perpendicular bisector of ℓ_{xy}, call it b_{xy} (see Figure 1). The line b_{xy} divided the plane into two regions. The points in the half plane containing x are nearer to x than y ; the points in the other half plane are all nearer to y than x.

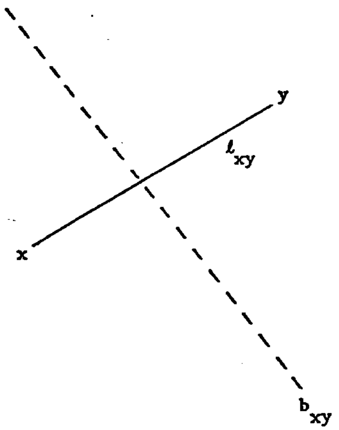


FIGURE 1. b_{xy} is the perpendicular bisector of ℓ_{xy}. Therefore, points below b_{xy} are closer to x than y and points above b_{xy} are closer to y than x.

In order to solve the post office problem it is sufficient to determine, for a given point, which region of the regions formed by the $\binom{m}{2}$ lines b_{xy} it lies in. By the basic algorithm this can be done in $3g(\binom{m}{2}) = O(\log m)$ time with $O(m^4)$ storage.

These applications of our basic algorithm are clearly optimal with respect to time to within a constant. (This follows since it takes at least $g(m)$ time to search m objects.) They, however, also demonstrate that our algorithms tend to use a large amount of storage. An interesting open question is therefore: can one search a planar graph's m regions in time $O(\log m)$ with storage $O(m)$? Or even $O(m\log m)$?

5. Applications in E^n

Most of the applications of the above algorithms in E^n are straightforward extensions of the applications given in the previous section. However, because of the exponential term in the operation count of Theorem 2, these extensions are only of interest if k, the dimension of the objects to be searched is small relative to m, the number of objects to be searched. Typically, we would require that m is larger than 2^k and hopefully as large as 2^{2^k}. However, in cases where k and m do satisfy these criteria, speedups do occur by applying the algorithms of Section 3. We study two applications of these algorithms here.

Consider first the problem of finding closest points in spaces of small dimension. An example of such a problem occurs in the area of speech recognition. Sounds can be classified according to a set of less than 8 characteristics [5] and thus we may consider a data base for a speech recognition system to consist of a set of points in E^8. When such a system is used to understand a speaker, the method used is to find for each sound uttered the closest sound in the data base. In order to develop a real time speech recognition system, it is reasonable to allow large quantities of preprocessing and storage to be arranged in advance as a tradeoff so that each sound uttered by the speaker can be identified as rapidly as possible. Thus, for a set of m sounds to be in the data base, the speed up of $O(m/g(m))$ afforded by an extension of the closest-point algorithm of the previous section to E^8 is very useful. Further studies of this extension are necessary to yield improvements in the storage requirements.

As a second application, we mention some extensions of the well-known knapsack problem (see e.g., [6], [7]). We may state this problem in the present context as

Knapsack Problem $(KS_{n,1})$. Given the hyperplanes $H_1, \ldots, H_{2^n-1}$ in E^n defined by $H_i(v_1, \ldots, v_n) = \sum_{j=1}^{n} c_{ij} v_{j-1}$ where $\sum_{j=1}^{n} c_{ij} 2^{j-1} = i$ for $i = 1, \ldots, 2^n-1$, the point $(x_1/b, \ldots, x_n/b)$ lies on one of the H_i if and only if there are 0-1 valued numbers $c_{i1}, \ldots, c_{in}$ such that $\sum_{j=1}^{n} c_{ij} x_j = b$ which is true if and only if the knapsack problem with input $(x_1, \ldots, x_n, b)$ has a solution.

Furthermore, we may consider the extended knapsack problem of seeking multiple solutions by

Knapsack Problem $(KS_{n,p})$. The point $(x_1/b, \ldots, x_n/b)$ lies on p of the hyperplane $H_1, \ldots, H_{2^n-1}$ and therefore on one of the $n-p$ dimensional objects $\theta_{i_1, \ldots, i_p} \triangleq H_{i_1} \cap H_{i_2} \cap \ldots \cap H_{i_p}$, $1 \leq i_1 < i_2 < \ldots < i_p \leq 2^n-1$

if and only if the knapsack problem with input $(x_1, \ldots, x_n, b)$ has p (or more) different solutions.

We then can establish the results

Theorem 4. The application of the algorithm of Theorem 2 yields an algorithm using at most $(3 \cdot 2^{n-p-1} + (n-2))g(2^{np})$ steps to solve $KS_{n,p}$.

Proof. On intersection, a set of $\binom{2n}{p}$ objects of dimension at most $n-p-1$ are formed. Straightforward application of Theorem 2 then yields the desired result. □

Further discussion of this result and its implications to an open problem in automata theory will appear in a future paper.

REFERENCES

[1] D. Knuth. The Art of Computer Programming. Volume 3: Sorting and Searching, Addison-Wesley, 1973.

[2] W. Newman and R. Sproull. Principles of Interactive Computer Graphics, McGraw-Hill, 1973.

[3] J.A. George. A Computer Implementation of the Finite Element Method. Ph.D. thesis, Stanford University, 1971.

[4] C. Liu. Introduction to Combinatorial Mathematics, Addison-Wesley.

[5] S. Levinson. Private Communication.

[6] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. Cornell University Computer Science Technical Report 72-134, July 1972.

[7] R. Karp. Reducibility among combinatorial problems. Complexitv of Computer Computations, ed. by R. Miller and J. Thatcher, Plenum Press, 1972.

On the Complexity of Computations

under Varying Sets of Primitives

David P. Dobkin and Richard J. Lipton
Department of Computer Science
Yale University
New Haven, Connecticut 06520

The first author's research was supported in part by the Office of Naval Research under Grant N00014-75-C-0450.

An earlier version of this paper was presented at the 2nd GI-Professional Conference on Automata Theory and Formal Language, at Kaiserslautern, Germany, in May 1975.

1. Introduction

The principal goal of research in computational complexity is the determination of tight lower bounds on the complexity, in terms of primitive operation executions, of solving problems or performing larger operations. While algorithms now exist that yield better than naive upper bounds for various operations (e.g. [1,9,11,12]), finding lower bounds for the solution of a problem using a general model has proved to be more difficult. In order to circumvent these difficulties, many authors (e.g. [3,5,6,8,13]) have chosen to work with models that place some restriction on the primitive operations that can be used or on the flow of output that can occur. Typical of the restrictions that have been placed on models are: allowing the use of only a monotone basis of functions [6,8] or requiring that all circuits be restricted to fan-out one [3,5,13]. The value of using such models is in the insights they produce into the general process of finding lower bounds; many of the actual lower bounds they produce are shown, however, to be invalid for more general models.

The goal of the current research is the study of lower bounds on the complexity of a set of searching problems under various restrictions on the nature of the primitive operation used to determine each branch in a search tree. Our model, to be described in more detail in the next section, has programs consisting of two types of statements. Query statements are of the form

L_k: if $f(x)\ \mathcal{R}\ 0$ then goto L_m else goto L_n

where $\mathcal{R}$ is one of the relations (> or =) and f is a function of restricted form on the input of x. An output statement of the form

L_s: accept (or reject)

occurs for each possible outcome of the problem.

The problems we consider all involve searching a set of geometric objects in Euclidean space to determine in which region of their partition of space a given point lies or whether the point lies in any of the given regions. Among the new results obtained are exponential lower bounds on searching for solutions to a knapsack problem, viewed as a hyperplane search problem, using various models involving restrictions on the primitive operations allowed. A non-linear (in the number of hyperplanes) lower bound in given for a generalized hyperplane search problem along with an O(n log n) bound for a problem in the plane.

2. Basic Model

Our model of computation is based on the notion of a search program. A search program P with input $(x_1,\ldots,x_n)$ is a finite list of instructions of the following three types:

```
1) $L_k$: if $f(x_1,\ldots,x_n)\ \mathcal{R}\ 0$ then goto $L_m$    ($\mathcal{R} \in \{>,=\}$)
                               else goto $L_p$
2) $L_k$: accept
3) $L_k$: reject
```

Control initially starts at the first instruction. An instruction of type (1) determines whether the indicated test is true: If it is true, control passes to the statement with label L_m; otherwise, control passes to the statement with label L_p. An instruction of type (2) denotes that the program has halted and it has accepted the input. Correspondingly, an instruction of type (3) denotes that the program has halted and it has rejected the input.

We will restrict search programs in two distinct ways. The functions allowed in instructions of type (1) are called primitives. Often we will restrict the class of allowed primitives. We will also restrict at times the relations $\mathcal{R}$ allowed in instructions of type (1). Thus an equality search program can have $\mathcal{R}$ equal only to =. On the other hand, a linear search program can have only functions f that are linear.

The complexity measure we will use on our search programs is "time." Each possible input $(x_1,\ldots,x_n)$ determines a computation through the search

program. The length of this computation is the number of steps associated with the input $(x_1,\ldots,x_n)$. We are always interested in the worst-case behavior, i.e. the maximum number of steps required by a given search program.

3. Restricted Linear Programs

In this section we will investigate the n-dimensional knapsack problem (KS_n). We can view this problem as follows: Given a point $(x_1,\ldots,x_n b) \in E^{n+1}$ we are to determine whether there exists an index set I such that

$$\sum_{i \in I} x_i - b = 0.$$

The first question we ask is: If we restrict our search programs to queries of the form

$$\sum_{i \in I} x_i \gtreqless b$$

can we show that they must require an exponential number of queries? The answer is yes:

Theorem 1: Any search program having as its primitive operation functions of the form

$$\sum_{i \in I} x_i - b$$

for some index set I and any tests >, =, and < must require $O(2^n)$ primitive steps to solve the n-dimensional Knapsack Problem.

Proof: We adopt an adversary approach and provide a set of data such that if fewer than $\binom{n}{n/2}$ primitive operations are executed the data can be altered so as to make it possible for the solution of the problem to change without changing previous results.

Our adversary will return answers to queries according to the following plan:

i) if $|I| < \frac{n}{2}$, then $\sum_{i \in I} x_i < b$

ii) if $|I| > \frac{n}{2}$, then $\sum_{i \in I} x_i > b$

iii) if $|I| = \frac{n}{2}$ and fewer than $\binom{n}{n/2} - 1$ tests on index sets of exactly $\frac{n}{2}$ elements have been done, then $\sum_{i \in I} x_i > b$.

We now make the claim that it is possible to provide three sets of data satisfying conditions (i), (ii), and (iii) such that each set yields a different result on the final query. From this claim, the theorem follows since although an algorithm knowing this adversary's strategy could eliminate all tests of index sets with cardinality not equal to $\frac{n}{2}$ the $\binom{n}{n/2}$ tests of index sets of cardinality $\frac{n}{2}$ must all be performed.

Claim: Assume the last test performed on an index set of exactly $\frac{n}{2}$ elements compares $x_1 + x_2 + \ldots + x_{n/2}$ to b; then there are choices of $x_1 = \gamma$, $x_2 = x_3 = \ldots = x_{n/2} = \alpha$ and $x_{(n/2)+1} = x_{(n/2)+2} = \ldots = x_n = \beta$ for $0 < \gamma \le \alpha \le \beta < b$ satisfying the three conditions of the adversary and yielding any of the three possible results $x_1 + \ldots + x_{n/2} \lesseqgtr b$.

Proof: The conditions (i), (ii), and (iii) can be restated as

i) $(\frac{n}{2} - 1)\cdot\beta < b$

ii) $\gamma + (\frac{n}{2} - 1)\cdot\alpha + \beta > b$

iii) $\gamma + (\frac{n}{2} - 2)\cdot\alpha + \beta > b$

and we observe that (iii) implies (ii), so that we need show only that conditions (i) and (iii) can be met along with the result of one of the cases:

Case I: $\gamma + (\frac{n}{2} - 1)\cdot\alpha > b$

Case II: $\gamma + (\frac{n}{2} - 1)\cdot\alpha < b$

Case III: $\gamma + (\frac{n}{2} - 1)\cdot\alpha = b$.

In the first case, the choice $\gamma = \alpha = \beta = \frac{2b}{n-1}$ works since

$$\text{i)}\ (\frac{n}{2} - 1)\cdot(\frac{2b}{n-1}) = (\frac{n-2}{n-1})\cdot b < b$$

$$\text{iii)}\ (\frac{2b}{n-1}) + (\frac{n}{2} - 2)\cdot(\frac{2b}{n-1}) + (\frac{2b}{n-1}) = (\frac{n}{n-1})\cdot b > b$$

$$\text{and}\ (\frac{2b}{n-1}) + (\frac{n}{2} - 1)\cdot(\frac{2b}{n-1}) = (\frac{n}{n-1})\cdot b > b.$$

The second case is handled by the choice $\gamma = b\cdot(\frac{2n-5}{(n - \frac{1}{2})^2})$,

$\alpha = 2b\cdot(\frac{1}{n-2} + \frac{2}{(n - \frac{1}{2})^2})$, $\beta = 2b\cdot(\frac{1}{n-2} - \frac{1}{(n - \frac{1}{2})^2})$ since

$$\text{i)}\ (\frac{n}{2} - 1)\cdot(2b\ (\frac{1}{n-2} - \frac{1}{(n - \frac{1}{2})^2})) = b\cdot(1 - \frac{n-2}{(n - \frac{1}{2})^2}) < b$$

$$\text{iii)}\ b\cdot(\frac{2n-5}{(n - \frac{1}{2})^2}) + (\frac{n}{2} - 2)\cdot 2b\cdot(\frac{1}{n-2} + \frac{2}{(n - \frac{1}{2})^2}) + 2b\cdot(\frac{1}{n-2} - \frac{1}{(n - \frac{1}{2})^2})$$

$$= b\cdot(\frac{4n-15}{(n - \frac{1}{2})^2} + 1) > b$$

$$\text{and}\ b\cdot(\frac{2n-5}{(n - \frac{1}{2})^2}) + (\frac{n}{2} - 1)\cdot 2b\cdot(\frac{1}{n-2} + \frac{2}{(n - \frac{1}{2})^2}) = b\cdot(\frac{4n-9}{(n - \frac{1}{2})^2} + 1) > b.$$

Finally, the choices $\gamma = \frac{2b(n-2)}{n^2-4}$, $\alpha = \frac{2bn}{n^2-4}$, and $\beta = \frac{2b(n+1)}{n^2-4}$ settle the third

case via

$$\text{i)}\ (\frac{n}{2} - 1)\cdot(\frac{2b(n+1)}{n^2-4}) = \frac{b(n+1)}{n+2} < b$$

$$\text{iii)}\ \frac{2b(n-2)}{n^2-4} + (\frac{n}{2} - 2)\cdot(\frac{2bn}{n^2-4}) + \frac{2b(n+1)}{n^2-4} > b$$

$$\text{and}\ \frac{2b(n-2)}{n^2-4} + (\frac{n}{2} - 1)\cdot(\frac{2bn}{n^2-4}) = b. \quad \square$$

The result of this theorem is that any polynomial-time algorithm for solving the knapsack problem must use comparisons to hyperplanes not in the original set but generated from the original set. While such an algorithm is possible, it is unlikely to exist as a general procedure but might rather exist as a set of procedures $\{P_i\}_{i=1}^{\infty}$ such that solving the n-dimensional knapsack problem involves using procedure P_n to generate new hyperplanes and solving the n+1-dimensional knapsack problem involves using (possibly different) procedure P_{n+1} to generate new hyperplanes. Examples of such procedures as well as a brief discussion of the implications of such a system for the question "P = NP?" are contained in [2]. The present result in conjunction with those discussions makes it extremely unlikely that P and NP are the same.

4. Linear Programs

Next we will study linear programs. That is, we will allow any test of the form

$$f(x_1,\ldots,x_n) \overset{\geq}{<} 0$$

where f is a linear function. The next theorem allows us to obtain lower bounds for the complexity of various membership problems.

Theorem 2: Any linear search tree that solves the membership problem for a disjoint union of a family $\{A_i\}_{i\epsilon I}$ of open subsets of R^n requires at least $\log_2 |I|$ queries in the worst case.

Proof: We prove that any such search tree T with leaves $D_1,\ldots,D_r$ has $r \geq |I|$ and hence a path of depth $\geq \log_2 |I|$. The leaves partition R^n and, for each j, D_j is an accepting leaf if $D_j \subseteq \bigcup_{i\epsilon I} A_i$ and a rejecting leaf otherwise. The theorem now follows from the observation that for each ℓ, $1 \leq \ell \leq r$, there is at most one i such that $D_\ell \cap A_i$ and $D_\ell \cap A_j$ are non-empty. If we choose points $x \in D_\ell \cap A_i$ and $y \in D_\ell \cap A_j$ then all points on the line joining x and y belong to D_ℓ by the convexity of D_ℓ. However, since A_i and A_j are disjoint, there is a point on this line that does not belong to $\bigcup_{i\epsilon I} A_i$. Hence D_ℓ can be neither an accepting nor a rejecting leaf and the theorem holds. □

Let us now generalize the knapsack problem (KS_n) to the generalized

knapsack problem (GKS_n): We are given 2^n hyperplanes $H_1,\ldots,H_{2n}$ in E^{n+1} space that form a simple arrangement, i.e. no n+2 hyperplanes have a common point. For each new point x we are to determine whether x lies in any of these hyperplanes. Note that we do <u>not</u> insist that the search tree determine which hyperplane x lies in; it must determine only whether or not x lies in some hyperplane.

From this result we obtain the following corollaries:

Corollary 1: The membership problem for GKS_n takes at least $O(n^2)$ queries for any search tree.

Proof: Since the hyperplanes of this problem form a simply arrangement, we can find a family $\{A_i\}_{i\epsilon I}$ of open subsets of R^n such that

$$x \in \bigcup_{i\epsilon I} A_i \leftrightarrow x \in GKS_n$$

and $|I| \geq O(2^{n^2})$ [4]. The corollary then follows from the theorem. □

This result improves a lower bound of O(n) due to Spira [10]. Further extensions of this result appear in [14,15].

Corollary 2 (Element Uniqueness Problem)*:* Let E_n be the set of points in R^n that have two coordinates equal; then any algorithm for determining membership in E_n requires at least O(n log n) queries.

Proof: Solving the membership problem for E_n corresponds to solving the membership problem for the family

$$\bigcup_{\pi\epsilon S_n} \{A_\pi\}$$

where

$$A_\pi = \{(x_1,\dots,x_n) \in R^n \mid x_{\pi(1)} < x_{\pi(2)} < \dots < x_{\pi(n)}\}$$

and S_n is the set of permutations on n objects. The result then follows from $|S_n| = n!$. □

Applications of this result may be found in [16].

5. Equality Programs

In the previous section, we considered the problem of determining whether a point belonged to the union of a family of open sets allowing linear search programs. Here, we extend our methodology to the problem of determining whether a point belongs to the union of a family of varieties allowing search programs that determine at each step whether the point is the root of an irreducible polynomial. Before proceeding, we state some results from algebraic geometry [7] that will be necessary to our development.

Definition: A variety $V(f_1,\dots,f_m)$ is a subset of R^n defined by $V(f_1,\dots,f_m) = \{(x_1,\dots,x_n) \in R^n \mid f_1(x_1,\dots,x_n) = \dots = f_m(x_1,\dots,x_n) = 0\}$ for polynomials $f_1,\dots,f_m$.

Definition: The polynomials f and g are said to be equivalent iff there exists a non-zero constant λ such that $f = \lambda g$.

Fact 1: If the dimension of $V(f_1,\dots,f_n)$ is denoted by $\dim(V(f_1,\dots,f_n))$ then

i) $\dim(A) = 0$ if and only if A is empty;

ii) if $R^n = \bigcup_{i=1}^{k} V(f_i)$, then one of the polynomials f_i is trivial;

iii) if f and g are non-trivial irreducible polynomials that are not equivalent, then $\dim(V(f,g)) < \dim(V(f))$.

Theorem 3: If $f_1,\dots,f_m$ are irreducible polynomials of n real variables that are not equivalent, then any equality search program for

$$\bigcup_{i=1}^{m} V(f_i)$$

using only irreducible polynomials requires at least m queries.

Proof: Let T be a search program of depth k that determines for any $x \in R^n$ whether or not $x \in \bigcup_{i=1}^{m} V(f_i)$. Select the path in T that always takes the NO branch; moreover, let $g_1(x) = 0,\ldots,g_\ell(x) = 0$ be the queries on this path $(\ell \le k)$. Define $F = V(f_1) \cup \ldots \cup V(f_m)$ and, for $1 \le i \le \ell$, $G_i = [V(g_1) \cup \ldots \cup V(g_i)]^c$ (where A^c = the complement of the set A). We now assert that if $\ell < m$ then

(1) $G_\ell \cap F \neq \phi$

and (2) $G_\ell \cap F^c \neq \phi$.

This will be a contradiction since $x \in G_k$ implies that x takes this path; hence, whether the leaf of this path is an ACCEPT or a REJECT we have a contradiction with either (1) or (2).

If (1) is false, i.e. if $G_\ell \cap F = \phi$, then there is some i such that for all j, $V(f_i) \neq V(g_j)$. This follows since $\ell < m$. Fix this i. Then $G_\ell \cap V(f_i) = \phi$, and hence $V(g_1) \cup \ldots \cup V(g_\ell) \cup V(f_i)^c = R^n$. Thus, $V(g_1) \cap V(f_i) \cup \ldots \cup V(g_\ell) \cap V(f_i) = V(f_i)$. But $\dim(V(g_j)\cap(V(f_i)) < \dim(V(f_i))$ by fact (iii). A contradiction.

Now assume that (2) is false, i.e. that $G_\ell \cap F^c = \phi$. This is equivalent to

$$V(g_1) \cup \ldots \cup V(g_\ell) \cup V(f_1) \cup \ldots \cup V(f_m) = R^n$$

which is impossible by Fact (ii). Hence $\ell \ge m$. □

Corollary 3: Any equality search program for KS_n that uses only irreducible polynomials requires at least 2^n queries.

References

[1] Blum, Floyd, Pratt, Rivest, Tarjan. Linear time bounds for selection. JCSS 7:448-461, 1973.

[2] Dobkin, Lipton. On some generalizations of binary search. ACM Symposium on the Theory of Computing, Seattle, Washington, May 1974.

[3] Fischer, Meyer, Paterson. Lower bounds on the size of Boolean formulas. ACM Symposium on the Theory of Computing, Albuquerque, New Mexico, May 1975.

[4] Grünbaum. Convex Polytopes. Interscience Publishers, 1967.

[5] Harper, Savage. On the complexity of the marriage problem. Advances in Mathematics 9:299-312, 1972.

[6] Kerr. The effect of algebraic structure on the computational complexity of matrix multiplication. PhD thesis, Cornell University, 1970.

[7] Lefschetz. Algebraic Geometry. Princeton University Press, 1953.

[8] Schnorr. A lower bound on the number of additions in monotone computations of monotone rational polynomials. Unpublished manuscript.

[9] Schönhage, Strassen. Fast multiplication of large numbers [in German]. Computing 1:182-196, 1966.

[10] Spira. On the number of comparisons necessary to rank an element. Computational Complexity Symposium, Courant Institute, 1973.

[11] Strassen. Gaussian elimination is not optimal. Numerische Mathematik 13:354-356, 1969.

[12] Tarjan. Depth-first search and linear graph algorithms. SIAM Journal on Computing 1, 1972.

[13] Vilfan. The complexity of finite functions. Technical Report 97, Project MAC, MIT, 1972.

[14] Dobkin. A non-linear lower bound on linear search tree programs for solving knapsack problems. Yale Computer Science Research Report 52, 1975.

[15] Dobkin, Lipton. A lower bound of 1/2 n^2 on the knapsack problem. In preparation.

[16] Shamos. Geometric complexity. PhD thesis, Yale University, to appear.

A Lower Bound of $\frac{1}{2}n^2$ on Linear Search Programs for the Knapsack Problem

David Dobkin

Richard J. Lipton

Research Report #70
Department of Computer Science
Yale University
New Haven, Connecticut 06520

Portions of this research were supported by the Office of Naval Research under grant N00014-75-C-0450 to the first author and the National Science Foundation under grant DCR-74-12870 to the second author.

1. Introduction

The purpose of this paper is to establish the following theorem:

Theorem: For each n, any linear search tree that solves the n-dimensional knapsack problem requires at least $\frac{1}{2}n^2$ comparisons for almost all inputs.

Previously the best known lower bound on this problem was nlogn [1]. The result presented here is the first lower bound of better than nlogn given for an NP-complete problem for a model that is actually used in practice. Previous non-linear lower bounds have been for computations involving only monotone circuits [8] or fanout limited to one. Our theorem is derived by combining results on linear search tree complexity [4] with results from threshold logic [11]. In section 2, we begin by presenting the results on linear search trees and threshold logic. Section 3 is devoted to using these results to obtain our main theorem.

2. Basic Concepts

In this section we introduce the basic concepts necessary to the understanding of our main theorem. To begin, we present the model for which our bounds hold. It has been previously studied in [6, 7, 10].

Definition. A linear search tree program is a program consisting of statements of one of the forms:

(a) L_i: if $f(x) > 0$ then go to L_j else go to L_k;

(b) L_i: halt and accept input x;

(c) L_j: halt and reject input x.

In (a) f(x) is an affine function (i.e., $f(x) = \sum_{i=1}^{n} a_i x_i + a_0$ for some $a_0, a_1, \ldots, a_n$) of the input $x = (x_1, \ldots, x_n)$ which is assumed to be from some euclidean space E^n. Moreover the program is assumed to be loop free.

In a natural way each linear search tree program computes some predicate on E^n. The complexity of such a program on a given input is the number of statements executed on this input. The complexity of such a program is defined as the maximum complexity over all inputs.

In proving our results, we shall make use of the following theorem which is proved here for completeness.

Theorem [4]: Any linear search tree program that determines membership in the set

$$\bigcup_{i \in I} A_i$$

where the A_i are pairwise disjoint nonempty open subsets of E^n requires at least $\log_2 |I|$ queries for almost all inputs.

Proof: We prove that any such search tree T with leaves $D_1, \ldots, D_r$ has $r \geq |I|$ and hence a path of depth $\geq \log_2 |I|$. The leaves partition E^n and, for each j, D_j is an accepting leaf if $D_j \subseteq \bigcup_{i \in I} A_i$ and a rejection leaf otherwise. The theorem then follows from the observation that the function $Y: \{I \rightarrow 1, \ldots, r\}$

defined by Y(i)'s being the least i such that $A_i \cap D$ is non-empty is an injective function. This observation is true since if $Y(i) = Y(j) = \ell$ for $i \neq j$ then there exist distinct points x and y such that $x \in A_i \cap D_\ell$ and $y \in A_j \cap D_\ell$. By the convexity of D_ℓ, each point on the line joining x and y lies in D_ℓ and hence is accepted as a point of $\bigcup_{i \in I} A_i$. Defining the function g: $L \rightarrow I$ by $g(Z) = k$ whenever $Z \in A_k$ yields the contradiction that g is the constant function i, since A_i is open and I is finite. □

In this paper we shall study the complexity of linear search trees for the n-dimensional knapsack problem, which we state as a geometric problem. It should be noted however that our methods can be applied to many other problems. We may state two equivalent versions of this problem.

Knapsack Problem (KSn):

i) Given a point $(x_1, \dots x_n) \in E^n$, does there exist a subset I such that

$$\sum_{i \in I} x_i = 1.$$

ii) Given the hyperplanes H_α, $\alpha \in \{0,1\}^n$ where

$$H_\alpha \triangleq (y_1, \dots, y_n) \in E^n \mid \sum_{i=1}^{n} \alpha_i y_i = 1\}$$

does $(x_1, \dots, x_n)$ lie on some hyperplane.

Clearly these two formulations are equivalent and they both correspond to the usual knapsack problem which is NP-complete [5].

The lower bound established here is proved by appealing to results from threshold logic. Before defining the necessary terms from this field, we demonstrate our method and the chief obstacle in applying it.

Let $\Gamma = \{0, 1\}^n - \{0^n\}$. Say a point x is above (below) the hyperplane H_α with $\alpha \in \Gamma$ provided

$$\sum_{i=1}^{n} \alpha_i x_i - 1$$

is positive (negative). Also let R_I for $I \subseteq \Gamma$ be the set

$$\{x \in E^n \mid x \text{ is above } H_\alpha \text{ with } \alpha \in I \text{ and below } H_\alpha \text{ with } \alpha \notin I\}.$$

Intuitively R_I is one of the regions formed by the hyperplanes. There are $2^{2^n - 1}$ possible such regions; however, many of these regions are empty. For example,

$$x_1 + x_2 > 1, \; x_3 + x_4 > 1, \; x_1 + x_3 < 1, \; x_2 + x_4 < 1$$

is empty. This example shows that the key problem is to determine how many regions are formed by the hyperplanes $\{H_\alpha\}\alpha \in \Gamma$.

The answer to this problem lies in threshold logic. We will now sketch the relevant results. Further details appear in [9].

Definition. Let A be a subset of $\{0, 1\}^n$. Then the partition of $\{0, 1\}^n$ into A and $\{0, 1\}^n - A$ corresponds to a threshold function provided there exist weights $w_1, \dots, w_n$ such that

(1) $x_1 \dots x_n \in A$ iff $w_1x_1 + \dots + w_nx_n > 1$.

(2) $x_1 \dots x_n \notin A$ iff $w_1x_1 + \dots + w_nx_n < 1$.

Note that (2) does not follow from (1).

Let N(n) be the number of such threshold functions, then [11] shows that

$$2^{\frac{1}{2}n^2} \le N(n) \le 2^{n^2}.$$

In the next section we use this result to obtain our lower bound.

3. Main Result

In this section we prove our main result, i.e., that any linear search tree for KS_n requires at least $\frac{1}{2}n^2$ comparisons. We first state a technical lemma:

Lemma: (1) R_I is an open set.

(2) $R_{I_1} = R_{I_2}$ implies that $I_1 = I_2$.

The proof of this is elementary and is omitted. This lemma shows (part (2)) that we need only prove that R_I is nonempty for many sets I in order to prove our theorem. The next lemma does this.

Lemma: Suppose that A partitions $\{0, 1\}^n$ and gives rise to a threshold function. Then R_A is nonempty.

Proof: Let $w_1, \dots, w_n$ be weights for A. Now we claim that $w = (w_1, \dots, w_n) \in R_A$.

(a) Let α be in A. Then w is above H_α since

$$\sum_{i=1}^{n} \alpha_i w_i > 1$$

by the definition of threshold function.

(b) Let α be in $\{0, 1\}^n - A$. Then w is below H_α since

$$\sum_{i=1}^{n} \alpha_i w_i < 1$$

and again this follows by the definition of threshold function.

Thus we have shown that $w \in R_A$. □

In summary we have shown that there are at least $2^{\frac{1}{2}n^2}$ distinct open sets R_I's. An appeal to our earlier theorem [4] yields the claimed lower bound.

Finding an upper bound on the linear search tree complexity of knapsack problem appears to be a trivial problem. Two possible methods of attack are available. In the first, an algorithm is sought that works uniformly in n. That is, we seek a single method of solving knapsack problems of all dimensions. The existence of such an algorithm that runs in polynomial time is unlikely because this would imply that P = NP. But, for each n, it may be possible to construct a linear search tree that solves all n-dimensional knapsack problems. To construct such a tree, it is necessary to study partitions of the set of knapsack regions by new hyperplanes in order to determine appropriate tests at each stage of the algorithm. Based on considerations of the structure of the regions of the knapsack problem, we conjecture that a polynomial-time algorithm does exist for this problem. The existence of such an algorithm would resolve an open question posed in [3] but would not show that P and NP are equal for the reason given there.

References

[1] D. Dobkin.
A non-linear lower bound on linear search tree programs for solving knapsack problems.
Yale Computer Science Research Report #52, 1975.

[2] D. Dobkin and R. Lipton.
The complexity of searching lines in the plane.
Submitted for publication.

[3] D. Dobkin and R. Lipton.
On some generalizations of binary search.
ACM Symposium on the Theory of Computing, Seattle, Washington, May 1974.

[4] D. Dobkin and R. Lipton.
On the complexity of computations under varying sets of primitive operations.
Automata Theory and Formal Languages, Springer-Verlag Lecture Notes in Computer Science #33.

[5] R. Karp.
Reducibilities among combinatiral problems.
In R. Miller and J. Thatcher, editors, Complexity of Computer Computations, 85-104, Plenum Press, 1972.

[6] M. Rabin.
Proving the simultaneous positivity of linear forms.
JCSS 6, 1972.

[7] E. Reingold.
Computing the maximum and the median.
Twelfth Annual Symposium on Switching and Automata Theory, 1971.

[8] C. Schnorr.
A lower bound on the number of additions in monotone computations of monotone rational polynomials.
Unpublished manuscript.

[9] E.L. Sheng.
Threshold Logic.
Academic Press, 1969.

[10] P. Spira.
Complete linear proofs of systems of linear inequalities.
JCSS, 6, pp. 205-216, .972.

[11] S. Yajima and T. Ibaraki.
A lower bound on the number of threshold functions.
IEEE EC14:926-929, 1965.

The Complexity of Searching Lines in the Plane: Preliminary Version

by

David P. Dobkin* Richard J. Lipton+

Department of Computer Science
Yale University
New Haven, Connecticut 06520
U.S.A.
(203)-436-8160

*Portions of this author's research were supported by ONR Grant N00014-75-C-0450.

+Portions of this author's research were supported by ONR Grant N00014-75-C-0752.

Abstract

Searching is a fundamental operation of computer science. Yet a number of key mathematical questions about searching in Euclidian spaces remains open. A number of such questions are formulated and answered here for searching lines in the plane. Relationships between the results here and higher dimensional analogs for other problems of interest are given. Among the new results is a mathematical framework in which questions about searching can be stated in a more uniform manner than was possible before. Specific results are also given on the searching complexity of various sets of lines in the plane. In particular, we show that there are easy and hard sets of lines to search and establish methods of generating upper and lower bounds on the search complexities of such sets.

(1) $x_1 \dots x_n \in A$ iff $w_1x_1 + \dots + w_nx_n > 1$.

(2) $x_1 \dots x_n \notin A$ iff $w_1x_1 + \dots + w_nx_n < 1$.

Note that (2) does not follow from (1).

Let N(n) be the number of such threshold functions, then [11] shows that

$$2^{\frac{1}{2}n^2} \le N(n) \le 2^{n^2}.$$

In the next section we use this result to obtain our lower bound.

3. Main Result

In this section we prove our main result, i.e., that any linear search tree for KS_n requires at least $\frac{1}{2}n^2$ comparisons. We first state a technical lemma:

Lemma: (1) R_I is an open set.

(2) $R_{I_1} = R_{I_2}$ implies that $I_1 = I_2$.

The proof of this is elementary and is omitted. This lemma shows (part (2)) that we need only prove that R_I is nonempty for many sets I in order to prove our theorem. The next lemma does this.

Lemma: Suppose that A partitions $\{0, 1\}^n$ and gives rise to a threshold function. Then R_A is nonempty.

Proof: Let $w_1, \dots, w_n$ be weights for A. Now we claim that $w = (w_1, \dots, w_n) \in R_A$.

(a) Let α be in A. Then w is above H_α since

$$\sum_{i=1}^{n} \alpha_i w_i > 1$$

by the definition of threshold function.

(b) Let α be in $\{0, 1\}^n - A$. Then w is below H_α since

$$\sum_{i=1}^{n} \alpha_i w_i < 1$$

and again this follows by the definition of threshold function.

Thus we have shown that $w \in R_A$. □

In summary we have shown that there are at least $2^{\frac{1}{2}n^2}$ distinct open sets R_I's. An appeal to our earlier theorem [4] yields the claimed lower bound.

Finding an upper bound on the linear search tree complexity of knapsack problem appears to be a trivial problem. Two possible methods of attack are available. In the first, an algorithm is sought that works uniformly in n. That is, we seek a single method of solving knapsack problems of all dimensions. The existence of such an algorithm that runs in polynomial time is unlikely because this would imply that P = NP. But, for each n, it may be possible to construct a linear search tree that solves all n-dimensional knapsack problems. To construct such a tree, it is necessary to study partitions of the set of knapsack regions by new hyperplanes in order to determine appropriate tests at each stage of the algorithm. Based on considerations of the structure of the regions of the knapsack problem, we conjecture that a polynomial-time algorithm does exist for this problem. The existence of such an algorithm would resolve an open question posed in [3] but would not show that P and NP are equal for the reason given there.

References

[1] D. Dobkin.
A non-linear lower bound on linear search tree programs for solving knapsack problems.
Yale Computer Science Research Report #52, 1975.

[2] D. Dobkin and R. Lipton.
The complexity of searching lines in the plane.
Submitted for publication.

[3] D. Dobkin and R. Lipton.
On some generalizations of binary search.
ACM Symposium on the Theory of Computing, Seattle, Washington, May 1974.

[4] D. Dobkin and R. Lipton.
On the complexity of computations under varying sets of primitive operations.
Automata Theory and Formal Languages, Springer-Verlag Lecture Notes in Computer Science #33.

[5] R. Karp.
Reducibilities among combinatiral problems.
In R. Miller and J. Thatcher, editors, Complexity of Computer Computations, 85-104, Plenum Press, 1972.

[6] M. Rabin.
Proving the simultaneous positivity of linear forms.
JCSS 6, 1972.

[7] E. Reingold.
Computing the maximum and the median.
Twelfth Annual Symposium on Switching and Automata Theory, 1971.

[8] C. Schnorr.
A lower bound on the number of additions in monotone computations of monotone rational polynomials.
Unpublished manuscript.

[9] E.L. Sheng.
Threshold Logic.
Academic Press, 1969.

[10] P. Spira.
Complete linear proofs of systems of linear inequalities.
JCSS, 6, pp. 205-216, .972.

[11] S. Yajima and T. Ibaraki.
A lower bound on the number of threshold functions.
IEEE EC14:926-929, 1965.

The Complexity of Searching Lines in the Plane: Preliminary Version

by

David P. Dobkin[*] Richard J. Lipton[+]

Department of Computer Science
Yale University
New Haven, Connecticut 06520
U.S.A.
(203)-436-8160

[*] Portions of this author's research were supported by ONR Grant N00014-75-C-0450.

[+] Portions of this author's research were supported by ONR Grant N00014-75-C-0752.

Abstract

Searching is a fundamental operation of computer science. Yet a number of key mathematical questions about searching in Euclidian spaces remains open. A number of such questions are formulated and answered here for searching lines in the plane. Relationships between the results here and higher dimensional analogs for other problems of interest are given. Among the new results is a mathematical framework in which questions about searching can be stated in a more uniform manner than was possible before. Specific results are also given on the searching complexity of various sets of lines in the plane. In particular, we show that there are easy and hard sets of lines to search and establish methods of generating upper and lower bounds on the search complexities of such sets.

I. Introduction

A fundamental operation of computer science is searching. Certainly the majority of actual computation involves the processing and organization of data into sets which are to be sorted in a manner to make repeated searches as simple as possible. Furthermore, Knuth [5] has devoted an entire chapter of his encyclopedic work on computer programming to the study of methods of computer searching. Despite this enormous focus on searching, a number of key mathematical issues regarding searching remain either unexplored or unanswered. Among these issues is the key issue of the searching of a set of geometric objects in Euclidian space. In addition to the existance of such problems as extensions and embellishments to previously studied problems of geometric complexity (see e.g. [2], [9]); this framework appears to be a natural setting for the generation of lower bounds on the knapsack, partition and travelling salesman problems as well as variants of the sorting problem. Furthermore, this methodology has also produced many good upper bounds which can be used to solve practical problems of such diverse areas as information retrieval, **numerical** analysis, and artificial intelligence. The main goal of this paper will be to lay the beginnings of a unified framework through which all questions of geometric searching can be resolved. To give an idea of the complexity of such a theory we pause to give an example of an elementary result within this theory which appears very anomalous. Consider the problem of determining membership of a point on or among a set of n lines in the plane which are in general position. That is, we are given a set of n lines in the plane with the condition that no three have a point in common and each pair has exactly one point in common (i.e. no two are parallel). We then wish to

ask questions about a new point determining at each query whether it lies to the left of right of one of the given lines. Our procedure halts after enough queries have been made to know whether the given point lies on any of the lines or if not, which lines bound the region in which it lies. A reasonable conjecture, given that any set of n lines of the plane in general position forms exactly $\frac{1}{2}(n^2+n+2)$ regions, is that the searching complexity of any set of n lines in general position is the same. Yet, as we shall see in subsequent sections of this paper there is a set of n lines which can be searched in $O(\log^2 n)$ queries while another set is shown to require n queries, an exponential gap. Such anomalies together with the guiding principle that "intuition about geometric problems is seldom correct" characterize this as a difficult problem. However, recent results [1,2,3,4,9] concerning searching complexities and lower bounds tend to characterize these as fruitful areas of research. Among the results reported in these papers are upper bounds of practical importance on some searching problems as well as lower bounds of $\frac{1}{2}n^2$ and n logn queries on linear search tree programs (i.e. each query is $f(\underline{x})$ R 0 where f is an affine function on the input $\underline{x}$ and R is >, = or < for the knapsack (i.e. Given $x_1,\ldots,x_n$, b does there exist $I \le \{1,\ldots,n\}$ such that $\sum_{i\in I} x_i = b$) and Element Uniqueness (i.e. Given $x_1,\ldots,x_n$, does there exist $i \neq j$ such that $x_i=x_j$) Problems.

In the current paper, we will focus our attention on problems involving searching lines in the plane. Such problems are of interest in themselves as well as a gateway to problems involving hyperplane searches in higher dimensional Euclidian spaces. Our goal will be one of classification of the complexity of searching different sets of lines. Two distinct cases exist, in the first only queries may be made of the original lines and in

I. Introduction

A fundamental operation of computer science is searching. Certainly the majority of actual computation involves the processing and organization of data into sets which are to be sorted in a manner to make repeated searches as simple as possible. Furthermore, Knuth [5] has devoted an entire chapter of his encyclopedic work on computer programming to the study of methods of computer searching. Despite this enormous focus on searching, a number of key mathematical issues regarding searching remain either unexplored or unanswered. Among these issues is the key issue of the searching of a set of geometric objects in Euclidian space. In addition to the existance of such problems as extensions and embellishments to previously studied problems of geometric complexity (see e.g. [2], [9]); this framework appears to be a natural setting for the generation of lower bounds on the knapsack, partition and travelling salesman problems as well as variants of the sorting problem. Furthermore, this methodology has also produced many good upper bounds which can be used to solve practical problems of such diverse areas as information retrieval, numerical analysis, and artificial intelligence. The main goal of this paper will be to lay the beginnings of a unified framework through which all questions of geometric searching can be resolved. To give an idea of the complexity of such a theory we pause to give an example of an elementary result within this theory which appears very anomalous. Consider the problem of determining membership of a point on or among a set of n lines in the plane which are in general position. That is, we are given a set of n lines in the plane with the condition that no three have a point in common and each pair has exactly one point in common (i.e. no two are parallel). We then wish to ask questions about a new point determining at each query whether it lies to the left of right of one of the given lines. Our procedure halts after enough queries have been made to know whether the given point lies on any of the lines or if not, which lines bound the region in which it lies. A reasonable conjecture, given that any set of n lines of the plane in general position forms exactly $\frac{1}{2}(n^2+n+2)$ regions, is that the searching complexity of any set of n lines in general position is the same. Yet, as we shall see in subsequent sections of this paper there is a set of n lines which can be searched in $O(\log^2 n)$ queries while another set is shown to require n queries, an exponential gap. Such anomalies together with the guiding principle that "intuition about geometric problems is seldom correct" characterize this as a difficult problem. However, recent results [1,2,3,4,9] concerning searching complexities and lower bounds tend to characterize these as fruitful areas of research. Among the results reported in these papers are upper bounds of practical importance on some searching problems as well as lower bounds of $\frac{1}{2} n^2$ and n logn queries on linear search tree programs (i.e. each query is $f(\underline{x})$ R 0 where f is an affine function on the input $\underline{x}$ and R is >, = or < for the knapsack (i.e. Given $x_1,\ldots,x_n$, b does there exist $I \leq \{1,\ldots,n\}$ such that $\sum_{i\in I} x_i = b$) and Element Uniqueness (i.e. Given $x_1,\ldots,x_n$, does there exist $i\neq j$ such that $x_i=x_j$) Problems.

In the current paper, we will focus our attention on problems involving searching lines in the plane. Such problems are of interest in themselves as well as a gateway to problems involving hyperplane searches in higher dimensional Euclidian spaces. Our goal will be one of classification of the complexity of searching different sets of lines. Two distinct cases exist, in the first only queries may be made of the original lines and in the second new lines may be added with queries made with respect to the original or new lines. Thus, if we define $c(A)$ and $\hat{c}(A)$ as the complexity under the first and second measures of searching the set of lines A, then $\hat{c}(A) \triangleq \min_{B} c(A \cup B)$ where B is any new set of lines. Among the results presented here are

$$2 \log |A| \leq \hat{c}(A) \leq 3 \log |A| \text{ for any set A where } |A| \text{ is the number of lines in A.}$$

And the existence for each n of sets A_1^n and A_2^n of n lines such that

$$c(A_1^n) \leq 3/4 \log^2 |A_1^n|$$

$$c(A_2^n) = |A_2^n| = n.$$

These results leave us unable to make general statements about the $c(.)$ function as we could about the $\hat{c}(.)$ function. Hence we concentrate our efforts on methods for determining for any set A, the value of $c(A)$. To do so, it is necessary to introduce new ideas to the standard mathematical notions of general position. And it is at this point where our work diverges from the standard mathematical literature on this subject. However, we believe that some of the methods and new ideas introduced here will, in addition to resolving questions regarding searching lines in planes, yield insight into methods of extending the known lower bound of $\frac{1}{2} n^2$ on the complexity of the knapsack problem in n-dimensions, as the issues there are merely higher-dimensional analogs of those introduced here.

The organization of the paper is as follows. In the next section, the exact problem which we are considering is presented in detail. The

concepts briefly spelled out above are concretely defined. Following that, some definitions and results concerning the geometry of intersecting lines in the plane are given. Some of these results belong to the classical mathematical literature on the problem while others were derived within the context of this problem. Results found by applying these results to the problems at hand are also surveyed.

II. Problem Statement

Searching problems in the plane will be our focus. Such a problem consists of a set of lines dividing the plane into regions. Our lines will be in general position, hence no two are parallel and no three have a point in common. Thus the number of regions formed by a set of n such lines will be $\frac{1}{2}(n^2+n+2)$. The searching problem for lines in the plane then consists of determining for a new point in which of these regions it lies. And our goal is to determine the complexity of searching any given set of lines in the plane. The algorithms we allow are linear tree progams which have been widely used before [3,7,10,11]. Such programs consist of three types of statements, branches of the form

S_k: if $f(x)$ R 0 then go to S_m else go to S_n,

and decision statements of the form

S_p: point x belongs to one of the lines

S_q: point x belongs to region R and none of the lines where f is a linear function on the input point x, R is one of the relations {<,=,>}, and R is a specification of one of the regions formed by the intersecting lines. The complexity of such an algorithm is defined as the longest path from its root to any decision statement.

Within this model, we consider two complexity measures on the searching of lines. In the first, the function f is restricted to represent one of the original lines. Thus, the problem here is to determine to which region a point belongs with only comparison to the original lines. We define the complexity of searching a set of lines, A, under this measure as c(A). One is tempted to believe that $c(A) = |A|$, the cardinaltiy of A, but the following example shows otherwise.

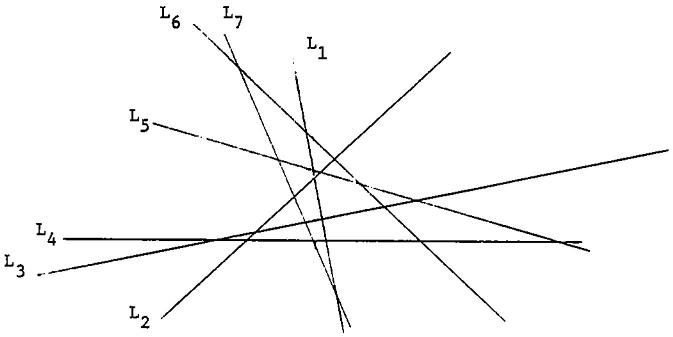


Figure 1: A set of 7 lines to be searched.

We observe that on the left of L_1, the lines L_4, L_5, and L_6 do not intersect and on the right, lines L_2, L_3 and L_7 do not intersect. Hence if x lies to the left of L_1, we can search L_4, L_5, and L_6 by a binary search algorithm and similarly for L_2, L_3 and L_7 if x lies to the right of L_1. Therefore, in at most 6 comparisons we can search these lines. Since all sets of lines are taken to be in general position, it would be reasonable to assume that c(A) is fixed for fixed $|A|$. This is untrue, since a set of 7 lines forming a septagon has a searching complexity of 7. We shall see in later sections that c(A) varies greatly with A for fixed $|A|$.

The second complexity measure we use allows for the introduction of new searching objects. The function f can now be any line in the plane. For this case, we represent the complexity of searching a set A of lines as $\hat{c}(A)$. It is easy to see that $\hat{c}(A) = \min_B c(A \cup B)$ taken over all sets of lines, B. In a previous paper [2], we showed that $\hat{c}(a) \le 3 \log|A|$ and a simple region counting arguments yields $\hat{c}(A) \ge 2 \log|A|$. However an exact bound on $\hat{c}(A)$ would be of value as this would yield insight into methods of generating better than information theoretic lower bounds on searching. In a related paper, applications of such results to tight bounds on the knapsack problem are studied [4].

Throughout, we shall use r(A) to denote the largest number of sides of any polygon formed by intersections of the lines in A. Clearly r(A) is a lower bound on c(A).

III. Results

In this section the basic structure of c(A), $\hat{c}(A)$, and r(A) is investigated. In addition to proving a number of simple but basic facts, we also demonstrate that understanding these functions is going to be a non-trivial task. This follows for two diffent but related reasons. First, the classical literature on arrangements of lines in the plane is filled with simple sounding assertions that are open. Indeed much of this literature is still trying to answer questions of the form "how many ... are there?". In contrast our research requires answers to questions of the form "how many ... are there and where are they with respect to ...?". Second, we are able to prove at least two results that are unexpected. Moreover, these results show that simple and intuitive arguments about even the function c(A) are possibly going to be incorrect. In particular we show that complexity behaves poorly with respect to disjoint union, i.e. there are disjoint sets A and B such that

$$c(A \cup B) << c(A) + c(B)$$

(<< means much smaller. See theorem 5 for details.) This result has a similar flavor to the result of Schnorr [8] on the corresponding result for Bookan circuits.

We first observe the following two easy lower bounds on c(A).

Theorem 1: Let A be a set of lines in the plane. Then $c(A) \geq r(A)$ and $c(A) \geq \log_2|A|$.

Proof: Recall that r(A) is the size of the largest region formed by A. Thus, a simple adversary argument demonstrates the lower bound of r(A). The lower bound of $\log_2|A|$ is the usual information theory argument. □

We now study a simple general method of obtaining upper bounds on c(A), $\hat{c}(A)$, and r(A).

Theorem 2: Let A and B be sets of lines in the plane. Then

(1) $c(A \cup B) \leq c(A) + c(B)$

(2) $r(A \cup B) \leq r(A) + r(B)$.

Proof:

(1) Any search trees for A and B respectively can be combined to form one for $A \cup B$ of size at most c(A) + c(B). (This uses the convexity of the regions that A and B form.)

(2) We sketch a proof that $r(A \cup B) \leq r(A) + r(B)$. Let R be the largest region of $A \cup B$; let $r_1,\ldots,r_k$ be the sides of R. Partition $r_1,\ldots,r_k$ into $s_1,\ldots,s_m$ and $t_1,\ldots,t_n$ such that each s_i is part of a line from A and each t_i is part of a line from B. By a convexity argument we can show that there is a region with at least m sides in A (alone) and one with at least n sides in B (alone). Thus, $r(A \cup B) = m+n \leq r(A) + r(B)$. The convexity argument is as follows: Consider the sides $s_1,\ldots,s_m$. Now extend them; they form a region with m sides. The other lines from A can not cut any of $s_1,\ldots,s_m$ by definition; hence, in A we must have a region of at least m sides. □

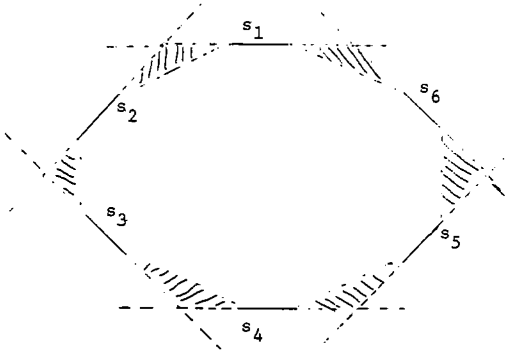


Figure 2: Only the shadowed regions can be cut by other lines of A; hence, a region of ≥m sides exists in A.

If the last theorem were tight one might hope that $c(A)$ would be about $|A|$. This is of course trivially false if A contains parallel lines. Thus a more interesting question is: Does $c(A)$ equal about $|A|$ for A in general position? We know from [2] that for $\hat{c}(A)$ this is false, i.e.

$$\hat{c}(A) \leq 3 \log_2 |A|.$$

We now show that $c(A)$ can also be very small compared to $|A|$. Note before we continue that $c(A) = |A|$ is possible for A in general position since there are such A with $r(A) = |A|$.

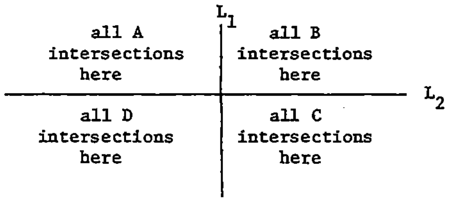
Theorem 3: For any n there is a set of lines $|A| = n$ in general position such that $c(A) = O(\log^2 |A|)$.

Proof: We proceed by induction. Let k be a constant such that for each $i<n$, there is a set, x_i, of i lines with $c(x_i) \leq k \log^2 |x_i|$. Construct a set x_n as follows:

I. Choose two lines L_1 and L_2 which divide the plane into four quadrants.

II. Choose four sets A, B, C and D such that each is a copy of $x_{\frac{n-2}{4}}$ and all intersections between lines in A occur within the first quadrant formed by L_1 and L_2, all B intersections in the second, ..., all D intersections in the fourth.

This yields the structure



Note that we put no restrictions on the locations of intersections of lines from different sets.

Now, we may search this set by first determining in which quadrant the point to be searched for lies. This requires 2 comparisons. Assume without loss of generality that the point lies in quadrant 1. We then consider the complexity of searching $A \cup B \cup C \cup D$ in the first quadrant. However,

$$c_1 (A \cup B \cup C \cup D) \leq c_1(A) + c_1(B) + c_1(c) + c_1(D)$$

where c_1 represents the complexity of searching in the first quadrant. We observe that $c_1(B)$, $c_1(C)$ and $c_1(D)$ are at most $\log_2(\frac{n-2}{4})$ as the lines in each of these sets have no intersections in the first quadrant and hence are totally ordered here. By induction, $c_1(A) \leq k \log^2(\frac{n-2}{4})$. Hence

$$c(x_n) \leq 2 + 3 \log_2(\tfrac{n-2}{4}) + k \log^2(\tfrac{n-2}{4}) \leq$$

$$k \log^2 n + (3-4k) \log n + (4k-4) \leq k \log^2 n \quad \text{for } k \geq 3/4$$

Q. E. D.

By methods of Lipton-Dobkin [6] we can use theorem 3 to demonstrate that there is a hierarchy in the following sense:

Corollary 4: For any monotone $f(n)$ such that $f(n) \leq n$ and $\frac{f(n)}{\log^2 n} \to \infty$ there is a family $\{A_n\}$ such that $|A_n| = n$ and

$$f(n) \leq c(A_n) \leq O(f(n)).$$

As stated earlier we will now show that $c(A)$ behaves poorly with respect to disjoint union.

Theorem 5: For any $n \geq 1$ there are $|A| = |B| = n$ sets of lines in general position such that $A \cup B$ is also in general position and

(1) $c(A \cup B) = O(\log^2 n)$

(2) $c(A) + c(B) \geq cn$ for some constant $c > 0$.

Sketch of Proof: Let A be a set of n lines in general position such that $r(A) = n$; let R be this region with n sides. Let B be the set of m lines constructed in theorem 3 positioned so that all the intersection points formed by the lines of B lie within R. Now to search $A \cup B$ we proceed as follows: First, determine where with respect to B we are. This can be done in $O(\log^2 m)$ steps. Second, if we are in a bounded region of B, then we must lie inside R and we are done. On the other hand, if we are in an unbounded region we argue as follows. The m+1 unbounded regions of B can be arranged with respect to A so that we can determine where we are in at most $O(\log^2 n)$ additional steps.

References

[1] D. Dobkin. A non-linear lower bound on linear search tree programs for solving knapsack problems. To appear.

[2] D. Dobkin and R. Lipton. On some generalizations of binary search. *ACM Symposium on the Theory of Computing*, Seattle, Washington, May 1974. (To appear in *SIAM Journal of Computing*.)

[3] D. Dobkin and R. Lipton. On the complexity of computations under varying sets of primitive operations. *Automata Theory and Formal Languages*, 2nd GI Conference, Spring-Verlag Lecture Notes in Computer Science, #33.

[4] D. Dobkin and R. Lipton. A lower bound of n^2 on the knapsack problem. In preparation.

[5] D. Knuth. *The Art of Computer Programming: Sorting and Searching*, III. Reading, Massachusetts: Addison Wesley, 1973.

[6] R. Lipton and D. Dobkin. Complexity measures and hierarchies for the evaluation of integers, polynomials and n-linear forms. *ACM Symposium on the Theory of Computing*, Albuquerque, New Mexico, May 1975.

[7] M. Rabin. Proving simultaneous positivity of linear forms. *JCSS* 6: 639-650, 1972.

[8] C. P. Schnorr. The network-complexity of equivalence and other applications of the network complexity. *Automata Theory and Formal Languages*, 2nd GI Conference, Spring-Verlag Lecture Notes in Computer Science, #33.

[9] M. Shamos. *Geometric Complexity*. PhD thesis, Yale University, New Haven, Connecticut. To appear.

[10] P. Spira. Complete linear proofs of systems of linear inequalities. *JCSS* 6:205-216, 1972.

[11] A. Yao. On the complexity of comparison problems using linear functions. *IEEE 16th Annual Symposium on Foundations of Computer Science*, Berkeley, California, October 1975.

Linear Programming is P-complete

David Dobkin
Richard J. Lipton
Steven Reiss

Department of Computer Science
Yale University
New Haven, Connecticut 06520

Portions of this research were funded by an NSF Graduate Fellowship, by the ONR under Grant N00014-75-0450 and by ONR under N00014-75-C-0752.

In this note, we study the space complexity of linear programming. In particular, we show that linear programming requires as much space to compute its solutions as any problem in P - the set of languages accepted in deterministic polynomial time by a multi-tape Turing machine. That is, if there exists a constant k such that linear programming is solvable in space $O(\log^k n)$, then all languages in P are accepted in $O(\log^{k'} n)$ space for some constant k'. This result is especially interesting since the membership of linear programming in P is currently in doubt [3]. We prove this result by showing that the problem of determining whether a contradiction can be reached by unit resolution for a formula in Conjunctive Normal Form (CNF) is log-space reducible to linear programming. Previously, this problem was shown to be P-complete in [2]. Another proof of this result may be derived by combining proofs that the hemisphere problem (i.e. given n points on the sphere in d dimensions, do all share a common hemisphere) is equivalent to linear programming [3] and that a restricted version of this problem (i.e. given the n points as before and an integer k, do any k of the original n share a common hemisphere) is NP-complete [1].

Our proof is based on a mapping from the clauses in CNF to inequalities. We define this mapping via

Definition: Suppose that $C = A_1 \vee \ldots \vee A_k \vee \sim B_1 \vee \ldots \vee \sim B_\ell$ is a clause, we say that C^* is the equation associated with C if C^* is the inequality

$$A_1 + \ldots + A_k + (1-B_1) + \ldots + (1-B_\ell) \geq 1.$$

Furthermore, if $\mathcal{C}$ is a set of clauses, we denote by $\mathcal{S}(\mathcal{C})$ the set of inequalities associated with clauses of $\mathcal{C}$.

This leads to a statement of our main result.

Theorem 1: Let $\mathcal{C} = (C_1, \ldots, C_m)$ be a set of clauses and $S(\mathcal{C})$ represent the set of inequalities associated with these clauses, then

i) If all clauses in $\mathcal{C}$ are simultaneously satisfiable, then all inequalities of $S(\mathcal{C})$ are simultaneously satisfiable on the interval $[0,1]$

ii) If there is a unit proof of a contradiction in $\mathcal{C}$, then $S(\mathcal{C})$ has no solution over the reals.

The proof of the first of these statements is trivial under the assignment of 1 to true literals and 0 to false literals. The second statement follows from the following lemma.

Lemma: If $\mathcal{C}_1$ follows from $\mathcal{C}_0$ by a unit resolution, then $S(\mathcal{C}_1)$ is solvable over the reals if $S(\mathcal{C}_0)$ is solvable over the reals.

Proof: Let $\mathcal{C}_0$ be the set of clauses $\{C_1, \ldots, C_p\}$ with $C_1 = A$ and $C_2 = \sim A \vee \alpha$ where α is a disjunction of literals. Then $\mathcal{C}_1$ can be represented as $\{C_0, C_3, \ldots, C_p\}$ where $C_0 = \alpha$. $S(\mathcal{C}_0)$ is the set of inequalities $\{A \geq 1,\ 1 - A + \alpha \geq 1,\ C_3^*, \ldots, C_p^*\}$ and $S(\mathcal{C}_1)$ is the set of inequalities $\{\alpha \geq 1,\ C_3^*, \ldots, C_p^*\}$. Clearly, a solution of $S(\mathcal{C}_0)$ is a solution of $S(\mathcal{C}_1)$ and hence the lemma holds.

Now, Jones and Laaser [2] show that determining whether a propositional formula in conjunctive normal form yields a contradiction by unit resolution is P-complete and this result may be combined with our theorem above to show our main result.

Theorem 2: Every language L in P is log-space reducible to linear programming.

Proof: By theorem 1, given a propositional formula in conjunctive normal form, determining whether a contradiction is found by unit resolution is log-space reducible to linear programming. And, by [2], this problem is P-complete.

This theorem is especially interesting since it is not known whether linear programming belongs to P or not (see [3] for more details on this problem.)

References

1 D. Johnson and F. Preparata, private communication.

2 N. Jones and W. Laaser, "Complete Problems for Deterministic Polynomial Time", Proceedings of the Sixth Annual ACM Symposium on the Theory of Computing, (1974), pp. 40-46.

3 S. Reiss and D. Dobkin, "The Complexity of Linear Programming", submitted for publication.

The Complexity of Linear Programming

by

Steven P. Reiss

David P. Dobkin

Department of Computer Science
Yale University
New Haven, Connecticut 06520

Portions of this research were supported by a NSF graduate fellowship and by ONR Grant N00014-75-C0450.

ABSTRACT

The complexity of linear programming and other problems in the geometry of d-dimensions is studied. A notion of LP-completeness is introduced, and a set of problems is shown to be of equivalent complexity to linear programming. Many of these problems involve simple extensions of constructions concerning convex hulls of polytopes, all of which are doable in O(nlogn) operations for d = 2. Finally, known results are surveyed in order to give an interesting characterization for the complexity of linear programming.

INTRODUCTION

Geometry is one of the oldest branches of mathematics. The geometry of d-dimensions has been an important area of study during the last century. In the last thirty years, this study has become far more important because it has served as a medium for investigating many problems. More recently, emphasis has been placed on the study of efficient computation for geometric problems. While the traditional mathematical approach to geometry gives us proofs of the existence of algorithms for given problems, the current emphasis is on finding efficient algorithms for these problems. In the two dimensional case, Shamos [28] has generated fast and asymptotically optimal algorithms for many geometric problems. One important extension of this work involves studying the complexity of some of these geometric problems in higher dimensions.

Four categories arise in extending these problems to arbitrary dimensions. First of all, many problems which are of polynomial complexity in the plane simply remain of polynomial complexity in d-dimensions because only a polynomial number of cases need be considered. Consider, for example, the problem of finding the closest pair of a set of n points in E^d, d-dimensional Euclidean space. At most $\binom{n}{2}$ pairs of points need to be considered, so the existence of a polynomial algorithm in any dimension is guaranteed. In contrast, some easily solved planar problems become obviously exponential in higher dimensions. For example, given a set of n points in E^d, their convex hull may have as many as $O(n^{d/2})$ facets. Thus any algorithm to find the convex hull in F^d must be exponential in d. Our focus in this paper will be on problems which belong to neither of these two classes. We consider problems whose complexity is not known to be either polynomial or exponential. Typical of such problems are the commonly studied problems of integer and linear programming. Karp and others [14] have shown that many problems of integer programming are NP-complete and, hence, probably of exponential complexity. However, the complexity of the rational analogs of many of these problems remains in doubt. In this paper we will study these rational analogs.

Our study consists of two parts. We begin by examining those problems in the geometry of d-dimensions which are not trivially polynomial or exponential in complexity and which are not known to be NP-complete. Since the most important such problem is that of linear programming, we first define a notion of LP-completeness and give the necessary geometric terminology. We then define the classical linear programming problem and some of its variations and discuss the geometric interpretation of these problems. Next we introduce a family of geometric problems that are not directly related to linear programming. Finally we present our main theoremwhich states that all of these problems are LP-complete, and prove this theorem.

The latter part of the paper is concerned with the complexity of these LP-complete problems, or, equivalently, of linear programming. Here we briefly discuss what is known concerning algorithms for solving linear programming problems and consider the location of linear programming in the P-NP complexity hierarchy. Our main conclusion based on current assumptions about the distinctions among P, NP and co-NP is

that either a polynomial algorithm exists for solving linear programming problems or linear programming is P-hard, i.e. not in P and not NP-complete.

Section I. Definitions

To begin, we set forth the notions of reducibility which will be used throughout this paper. Problems under consideration will be phrased as language recognition problems where we are interested in whether a given input is a member of the set of acceptable inputs, or as actual problems where we want to produce an answer. The length of the desired answer will in all cases be short enough to make lower bound arguments based on output length meaningless. Our basic notions of reducibility are equivalent to those used by Karp (1-1 reducibility)[14] or Ladner (many-one reducibility) [22] and are defined for recognition problems as

> Definition 1: Problem A is said to be (polynomial) reducible to problem B, denoted $A \propto B$, if and only if there exists a function f, computable in deterministic polynomial time, such that $x \in A$ if and only if $f(x) \in B$.
>
> Definition 2: Problems A and B are said to be (polynomial) equivalent, denoted $A \equiv B$, if and only if both $A \propto B$ and $B \propto A$.

In the case of problems where an answer is required, we extend these definitions to allow A to be reducible to B if and only if an algorithm for solving B yields an algorithm for solving A after polynomial transformation.

We will let P denote the class of problems that are solvable in polynomial time on a deterministic multitape Turing machine and let NP denote the class of problems that can be solved in polynomial time on a nondeterministic multitape Turing machine. A problem is called NP-complete if and only if it is in NP and every problem in NP is reducible to it. A problem is called P-hard if and only if it is in NP, not in P, and is not NP-complete. Finally a problem is said to be LP-complete if and only if it is equivalent to the problem of Linear Programming (LP).

As most of the problems we consider here have a geometric flavor, we will define the necessary geometric concepts before proceeding. E^d denotes d-dimensional Euclidean space and a point P in E^d is represented by a d-vector $(p_1,\dots,p_d)$. The function $(,): E^d \times E^d \to E$ is the usual dot product, i.e. $(P,Q) = \sum_{i=1}^{d} p_i q_i$. An affine sum of a set of points $P_1,\dots,P_n$ is a weighted sum $\sum_{i=1}^{n} x_i P_i$ such that $\sum_{i=1}^{n} x_i = 1$. A convex sum of a set of points is an affine sum such that each $x_i \geq 0$. An m-flat in E^d, $m < d$, is an m-dimensional surface. A 0-flat is a point; a 1-flat is a line; a 2-flat is a plane. A (d-1)-flat is called a hyperplane and can be written as $\{X \in E^d \mid (a,X)=b\}$ for some d-vector a and some scalar b. A (closed) halfspace is the set of points on or on one side of a hyperplane. It can be written as $\{X \in E^d \mid (a,X) \geq b\}$ or as $\{X \in E^d \mid (a,X) \leq b\}$. The corresponding hyperplane, $\{X \in E^d \mid (a,X)=b\}$, is called the determining hyperplane of the halfspace. A region in E^d is called convex if and only if for every pair of points in the region, the line segment connecting them lies completely inside the region. In particular, halfspaces and all flats are convex regions. The intersection of any number of convex regions is convex. Given a set of points, their convex hull is the smallest convex set containing these points. Convex regions in E^d determined by the intersection of a finite number of halfspaces are called polyhedra. If a polyhedron is bounded it is called a polytope. A hyperplane is called a supporting hyperplane of a polyhedron if and only if it has a non-empty intersection with the polyhedron and the polyhedron lies totally in one of the two halfspaces determined by the hyperplane. The intersection of a supporting hyperplane with the polyhedron is called a face of the polyhedron. An m-face, $m<d$, is a face that has dimension m, that is, the subspace that can be written as an affine sum of points from the face has dimension m. A 0-face is called

a vertex and a (d-1)-face is called a facet. For further details the reader is referred to [9, 10].

Section II: Linear Programming Problems

A lot of literature exists concerning linear programming problems. Throughout this literature the basic problem has been formulated in several different, although equivalent, ways. In this section we will summarize these equivalent forms of the problem. Our basic definition of the problem will be

LINEAR PROGRAMMING (LP)

Given: An integer $n \times d$ matrix A, integer n-vector b, integer d-vector c.

Find: A rational d-vector x such that $Ax \le b$ and $c^T x$ is maximized.

In this problem, $Ax \le b$ is a set of constraints of the form $(a,x) \le b_i$ and $c^T x$ is the objective function or the function to be maximized. The classical variants of this problem allow the inputs to be rationals rather than integers, allow the objective function to be minimized, or allow the constraints to take slightly different forms, either $Ax \ge b$ or $Ax=b$, $x \ge 0$. We can thus define a more general version of the linear programming problem, where any of the options in brackets may be chosen:

GENERAL LINEAR PROGRAMMING

Given: A rational $n \times d$ matrix A, rational n-vector b, rational d-vector c.

Find: A rational d-vector x such that $[Ax \le b;\ Ax \ge b;\ Ax=b \text{ and } x \ge 0]$ and $c^T x$ is [maximized; minimized].

If a problem is an instance of the general linear programming problem we will refer to it as a linear programming problem. A typical linear programming problem and its variants are given in figures 1, 2, and 3.

Maximize $4x_1+x_2$

Subject to:

$$\begin{aligned} x_1 - x_2 &\leq 1\\ x_1 + 3x_2 &\leq 4\\ -2x_1 + x_2 &\leq 3\\ - x_2 &\leq 0 \end{aligned}$$

Solution: $x_1 = \frac{7}{4}$, $x_2 = \frac{3}{4}$

maximum=31/4

Figure 1. A Linear Programming Problem.

Maximize $(4\ 1)\begin{pmatrix} x_1\\ x_2 \end{pmatrix}$

Subject to:

$$\begin{pmatrix} 1 & -1\\ 1 & 3\\ -2 & 1\\ 0 & -1 \end{pmatrix}\begin{pmatrix} x_1\\ x_2 \end{pmatrix} \leq \begin{pmatrix} 1\\ 4\\ 3\\ 0 \end{pmatrix}$$

Solution: $x = \begin{pmatrix} \frac{7}{4}\\ \frac{3}{4} \end{pmatrix}$

maximum=31/4

Figure 2. Matrix Form of Problem of Figure 1.

(a) Maximize $(4\ 1)\begin{pmatrix} x_1\\ x_2 \end{pmatrix}$

Subject to:

$$\begin{pmatrix} -1 & 1\\ -1 & -3\\ 2 & -1\\ 0 & -1 \end{pmatrix}\begin{pmatrix} x_1\\ x_2 \end{pmatrix} \leq \begin{pmatrix} -1\\ -4\\ -3\\ 0 \end{pmatrix}$$

Solution: $x = \begin{pmatrix} \frac{7}{4}\\ \frac{3}{4} \end{pmatrix}$ max.=31/4

(b) Minimize $(-4\ -1)\begin{pmatrix} x_1\\ x_2 \end{pmatrix}$

Subject to: $x \geq 0$ and

$$\begin{pmatrix} 1 & -1 & -1 & 1 & 0 & 0\\ 1 & 3 & -1 & 0 & 1 & 0\\ -2 & 1 & 2 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6 \end{pmatrix} = \begin{pmatrix} 1\\ 4\\ 3 \end{pmatrix}$$

Solution: $\begin{pmatrix} 7/4\\ 3/4\\ 0\\ 0\\ 0\\ 23/4 \end{pmatrix} = x$; min.=-31/4

Figure 3. General Forms of Pgoblem of Fiqure 1.

In the classical literature on linear programming (see e.g. [4]) there is a good deal of discussion concerning the dual problem. Each instance of a linear programming problem has a corresponding instance of a dual problem. For example, the dual to our basic problem is

DUAL LINEAR PROGRAMMING

Given: An integer n×d matrix A, integer n-vector b, integer d-vector c.

Find: A rational n-vector y such that $A^T y = c$, $y \geq 0$, and $b^T y$ is minimized.

Figure 4 here

Note that this is a linear programming problem in itself with inputs A^T, c, b, and, in general, the dual of a linear programming problem is a linear programming problem [4].

So far we have considered basically unrestricted linear programming problems. We next want to consider two problems in which we restrict the range of solutions and then, in addition, restrict the problem itself. It is clearly possible to derive a linear programming problem where the maximum value of the objective function is unbounded by the constraints. In some cases this is not desirable, hence we introduce the restricted form of the problem in which the solution is bounded:

BOUNDED LINEAR PROGRAMMING

Given: An integer n×d matrix A, integer n-vector b, integer d-vector c, rational M such that $c^T x \leq M$ for all x such that $Ax \leq b$.

Find : A rational d-vector x such that $Ax \leq b$ and $c^T x$ is maximized.

Figure 5 here

Minimize $\begin{pmatrix} 1 & 4 & 3 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$

Subject to:

$$\begin{bmatrix} 1 & 1 & -2 & 0 \\ -1 & 3 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

and

$y \geq 0$

Solution:

$$y = \begin{pmatrix} 11/4 \\ 5/4 \\ 0 \\ 0 \end{pmatrix}$$

Min. = 31/4

Figure 4. A Dual Problem to that of Figure 1.

Maximize $\begin{pmatrix} 4 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

Subject to:

$$\begin{pmatrix} 1 & -1 & -1 \\ 1 & 3 & -1 \\ -2 & 1 & 2 \\ 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 4 \\ 3 \\ 1000 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Solution:

$$x = \begin{pmatrix} 7/4 \\ 3/4 \\ 0 \end{pmatrix}$$

Max. = 31/4

Figure 5. Bounded Version of Problem of Figure 1.

Maximize $\begin{pmatrix} 4 & 1 & 1000 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - 1000$

Subject to:

$$\begin{pmatrix} 1 & -1 & 0 \\ 1 & 3 & 0 \\ -2 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 4 \\ 3 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Solution:

$$x = \begin{pmatrix} 7/4 \\ 3/4 \\ 1 \end{pmatrix}$$

Max = 31/4

Figure 6. A Bounded Positive Form of the Problem of Figure 1.

In addition it is helpful to consider this form of the problem with the added restriction that the vector $b>0$.

BOUNDED POSITIVE LINEAR PROGRAMMING

Given: An integer $n\times d$ matrix A, integer n-vector $b>0$, integer d-vector c, rational M such that $c^T x \le M$ for all x such that $Ax \le b$.

Find: A rational d-vector x such that $Ax \le b$ and $c^T x$ is maximized.

Figure 6 here

All the above characterizations of linear programming treat the problem as that of actually finding a solution. As has been done in complexity theory [14] we can also consider linear programming as a recognition problem:

LINEAR INEQUALITIES (LI)

Given: An integer $n\times d$ matrix A, integer n-vector b.

Determine: If there is a rational d-vector x such that $Ax \le b$.

In this problem we are only interested in determining if a solution exists rather than actually finding the solution. In this sense a recognition problem seems easier. We can also consider simpler linear programming recognition problems. One such problem is the recognition version of the Bounded Positive Linear Programming problem. This is

BOUNDED POSITIVE LINEAR INEQUALITIES

Given: Integer $n\times d$ matrix A, integer n-vector $b>0$, integer d-vector c, rational M such that $c^T x \le M$ for all x such that $Ax \le b$, rational E.

Determine: If there is a rational x such that $Ax \le b$ and $c^T x \ge E$.

In addition we can consider simpler versions of LI in which the vector b is fixed at zero. These include

SIMPLIFIED LINEAR PROGRAMMING I

Given: An integer $n\times d$ matrix A.

Determine: If there is a rational d-vector x such that $Ax=0$, $x \ne 0$, $x \ge 0$.

and

SIMPLIFIED LINEAR PROGRAMMING II

Given: An integer $n\times d$ matrix A.

Determinè : If there is a rational d-vector x such that $Ax \ge 0$ and $x \ne 0$.

We will show that all of these problems are equivalent and hence LP-complete, that is, that none of these simplifications can give a significantly simpler problem.

One of the questions that arises when a linear programming problem is presented is whether or not the set of constraints is redundant. We will say that this set is redundant if there is a constraint which is always satisfied when all the other constraints are satisfied. The problem of testing for this condition is

RELEVANCY

Given: A set of constraints $(a_0,x)\le b_0$, ... , $(a_n,x)\le b_n$.

Determine: If satisfying the last n constraints is equivalent to satisfying all n+1 constraints.

We shall see that this problem is equivalent to linear programming.

Finally, we will consider a problem that would seem to be more complicated than the basic problem of linear programming. This problem is the complement of the problem of Linear Inequalities, and here, rather than finding a solution or determining if one exists, we have to prove that no solution exists. The problem is

LINEAR PROGRAMMING COMPLEMENT (LPC)

Given: An integer n×d matrix A, integer n-vector b.

Show: That the system $Ax \leq b$ has no rational solutions.

Again we will show that this problem is LP-complete. This result will be important in our consideration of where LP-complete problems lie in the P-NP complexity hierarchy for it will make it unlikely that LP-completeness and NP-completeness are equivalent notions as it shows that the former is closed under complementation while the latter is probably not.

Section III: Geometric Linear Programming Problems

It has long been known that linear programming problems can be viewed as problems in the geometry of d-dimensional Euclidean space, E^d. This fact has been the basis of much of the study of linear programming and especially the study of the complexity of linear programming. Consider our basic linear programming problem. The solution vector x can be thought of as a point in E^d. Each of the constraints $(a,x) \leq b$, restricts the set of feasible solutions (possible points x that satisfy all the constraints) to a halfspace in E^d. As the solution must satisfy all the constraints simultaneously, the set of feasible solutions is the intersection of the various halfspaces determined by the constraints. This intersection is a convex polyhedron in E^d. Moreover, it can be shown [4] that a solution to a linear programming problem corresponds to a vertex of this polyhedron. Thus, we can reformulate linear programming as a geometric problem (see figures 7a, b) as

GEOMETRIC LINEAR PROGRAMMING

Given : A set of halfspaces $\{H_1,...,H_n\}$ and a d-vector x.

Find: The vertex v of the polyhedron formed by the intersection of the halfspaces at which (v,x) is maximized.

Just as we demonstrated restricted forms of the linear programming problem, we can consider restricted forms of geometric linear programming problems. In particular the geometric form of Bounded Positive Linear Programming is

BOUNDED POSITIVE GEOMETRIC LINEAR PROGRAMMING

Given: A set of halfspaces $\{H_1,\ldots,H_n\}$ such that their intersection is a polytope containing the origin, and a d-vector x.

Find: The vertex v of the polytope at which (v,x) is maximized.

In geometry there is a well-defined concept of a geometric dual. The dual is formed by a dimension-inverting mapping from E^d to E^d. That is, this mapping takes objects of dimension n in E^d into objects of dimension d-n-1 in E^d. In particular, points are mapped into hyperplanes and hyperplanes into points. There are several methods of constructing such a mapping, but the most common one is that of the polar dual. This mapping takes an object Q into an object $\overline{Q}$ such that

$$\overline{Q} = \{x \in E^d \mid (u,x) \leq 1 \text{ for all } u \in Q\}.$$

It is a well-known geometric result that this mapping has several nice properties:

Lemma 1: Let P be a polytope in E^d and let $\overline{P}$ be its polar dual. Then

(1) $0 \in P$ if and only if $\overline{P}$ is bounded, where 0 is the origin;

(2) There is a 1-1 onto mapping between the k-faces of P and the d-k-1 faces of $\overline{P}$.

(3) $\overline{\overline{P}}$, the dual of $\overline{P}$, is P.

(4) If the supporting hyperplane to a facet of P is $\{x \in E^d \mid (u,x)=1\}$, then the corresponding point in $\overline{P}$ is u.

(5) $\overline{P}$ is the convex hull of U where $U = \{u \in E^d \mid u$ corresponds to a facet of $P\}$.

Proof: These are known geometric results and can be found in [9].

For any geometric problem it is generally possible to consider the dual problem instead since we can map the original or primal problem into the dual, solve the dual problem, and then apply the dual mapping, which by (3) above is its own inverse, to construct the primal solution. In particular, the dual to the geometric version of the linear programming problem is

DUAL GEOMETRIC LINEAR PROGRAMMING

Given: A set of points $x_1,\ldots, x_n$ in E^d such that the origin is interior to their convex hull, and a ray r from the origin.

Find: The facet of the convex hull through which r passes.

Figure 7 here

In two dimensions this problem, and many others involving the convex hull of a set of n points, can be solved quickly, generally in time O(n) or O(nlogn), as the convex hull has at most n facets and can be found in time O(nlogn). However, in the d-dimensional case these arguments break down since the convex hull of a set of n points in E^d can have as many as $O(n^{d/2})$ facets. If a polynomial time algorithm is to exist for linear programming, it must be able to circumvent this difficulty by constructing only those portions of the convex hull necessary to solve the linear programming problem. At present, no subexponential algorithms are known which construct portions of the convex hull which are sufficient to solve the linear programming problem. For the remainder of this paper, we will give varying sets of sufficient constructions for solving this problem.

(a) Maximize $2x_1+x_2$

Subject to: $\begin{pmatrix} 1 & 1 \\ -2 & 0 \\ -4 & -2 \\ -2 & -4 \\ 0 & -4 \end{pmatrix} (x_1\ x_2) \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

Solution: $\left(\frac{5}{4}, \frac{-4}{4}\right)$ MAX = 9/4

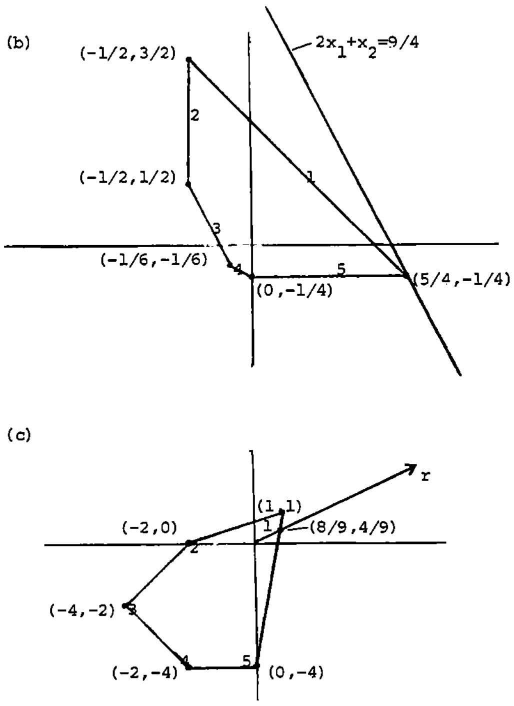


Figure 7. A Bounded Positive Linear Programming Problem (a), its geometric form (b), and its geometric dual form (c).

In addition to considering the geometric forms of the linear programming problem, we can consider the geometric versions of several of the related problems. In particular, the geometric forms of the problems of Linear Inequalities and of Bounded Positive Linear Inequalities respectively are

INTERSECTION OF HALFSPACES

Given: Closed halfspaces $H_1, \ldots, H_n$.

Determine: If $H_1 \cap \ldots \cap H_n$ is non-empty.

and

BOUNDED POSITIVE GEOMETRIC LINEAR INEQUALITIES

Given: Closed halfspaces $H_1, \ldots, H_n$ such that $H = H_1 \cap \ldots \cap H_n$ is a polytope containing the origin, and a hyperplane h.

Determine: If $h \cap H$ is non-empty.

Finally, the geometric versions of the problems of Simplified Linear Programming I and II can be stated respectively as

INTERSECTION OF FLAT AND ORTHANT

Given: An n-flat L through the origin in E^d.

Determine: If $L \cap 0^d$ contains any points other than the origin, where 0^d is the positive orthant.

and

INTERSECTION OF COMMON HALFSPACES

Given: Halfspaces $H_1, \ldots, H_n$ in E^d such that the origin is interior to their intersection.

Determine: If the origin is their intersection.

We shall see that all of these geometric versions of linear programming problems are equivalent to linear programming and, hence, are LP-complete.

Section IV: Geometric Problems

So far we have seen several forms of linear programming that are polynomial equivalent and hence LP-complete. We have also seen several geometric problems which are either restatements of some form of linear programming or are the geometric duals of such restatements. In this section we will consider several other geometric problems and will show that they too are LP-complete. The first class of such problems involves the identification of extreme points. We say that a point Q is extreme with respect to points $P_1,\ldots,P_n$ if and only if Q is exterior to the convex hull of $P_1,\ldots,P_n$. The basic problem here is

EXTREME POINT (EP)

Given: A set of points $P_0,P_1,\ldots,P_n$ in E^d.

Determine: If P_0 is extreme with respect to $P_1,\ldots,P_n$.

We can consider several variations of this problem. First of all, we can simplify the problem by placing all the points $P_1,\ldots,P_n$ on the unit sphere in E^d and letting P_0 be the origin. This yields

ORIGIN INTERIOR PROBLEM

Given: A set of points $P_1,\ldots,P_n$ on S^{d-1}, the unit sphere in E^d.

Determine: If the origin is extreme with respect to $P_1,\ldots,P_n$.

This version of the problem can also be restated as

HEMISPHERE PROBLEM

Given: A set of points $P_1,\ldots, P_n$ on S^{d-1}.

Determine: If $P_1,\ldots, P_n$ lie interior to some hemisphere.

Figure 8 here

This problem, as well as the previous two, can be solved in linear time in the plane, but remain LP-complete in the general case. It is interesting to note that Johnson and Preparata [12] have shown that a modification of this problem is NP-complete. In particular, if we give as input the n points on S^{d-1} and an integer $k < n$ and seek to determine whether k(or more) of the n points share a common hemisphere, this problem is reducible to MAXSAT2[7] which is known to be NP-complete. Furthermore, by applying their reduction to the HEMISPHERE PROBLEM, we are able to make contact with the work of Jones and Laaser [13] and show that if linear programming is solvable in poly-log space, then all problems in P are solvable in this space bound [5].

The geometric dual problem to EP is another variant here. It can be stated as

HYPERPLANE-HALFSPACE INTERSECTION

Given: A set of halfspaces $H_1,\ldots, H_n$ and a hyperplane h.

Determine: If h intersects $\bigcap_{i=1}^{n} H_i$.

This is the same problem as Bounded Positive Geometric Linear Inequalities. It is interesting to note that the algebraic form of this dual problem is the linear programming problem of Relevancy, that is, the problem of determining if a given constraint is relevant.

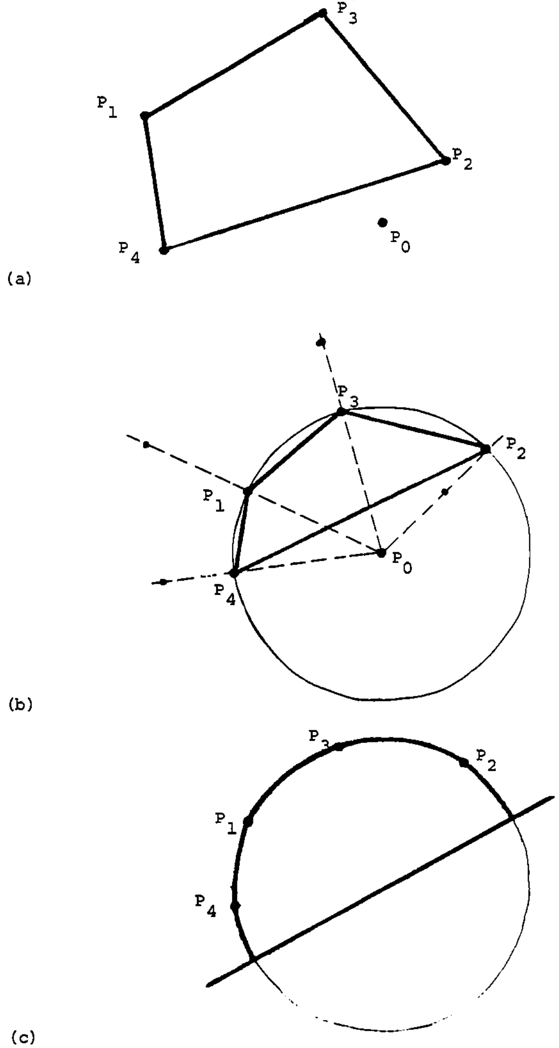


Figure 8. An Extreme Point Problem; the corresponding Origin Interior Problem; and the corresponding Hemisphere Problem.

A second class of geometry problems that can be shown to be LP-complete involves the notion of separability. Two point sets are said to be separable if and only if there is a hyperplane H such that all points of one set lie on one side of the hyperplane and all points of the other set lie on the other side (i.e., the hyperplane separates the points). Classical results show that two point sets in E^d are separable if and only if every subset of d+2 points is separable. This yields an algorithm of complexity $0(n^{d+2})$ for sets of n points in E^d. Shamos [28] has shown that this problem can be solved in time 0(nlogn) in two dimensions. While this is a vast improvement over the classical n^4, applying the same techniques in higher dimensions means forming the convex hull and hence yields an algorithm of exponential complexity in E_d. The basic problems here are

POINT-SET SEPARATION

Given: Points P_0, $P_1,\ldots, P_n$ in E^d.

Determine: If P_0 is separable from $\{P_1,\ldots, P_n\}$

and

SET-SET SEPARATION

Given: Points $P_1,\ldots, P_n$, $Q_1,\ldots, Q_m$ in E^d.

Determine: If $\{P_1,\ldots, P_n\}$ is separable from $\{Q_1,\ldots, Q_m\}$.

Figure 9 here

We can also consider simpler versions of these two problems. First of all we can restrict the points to the unit sphere. This gives

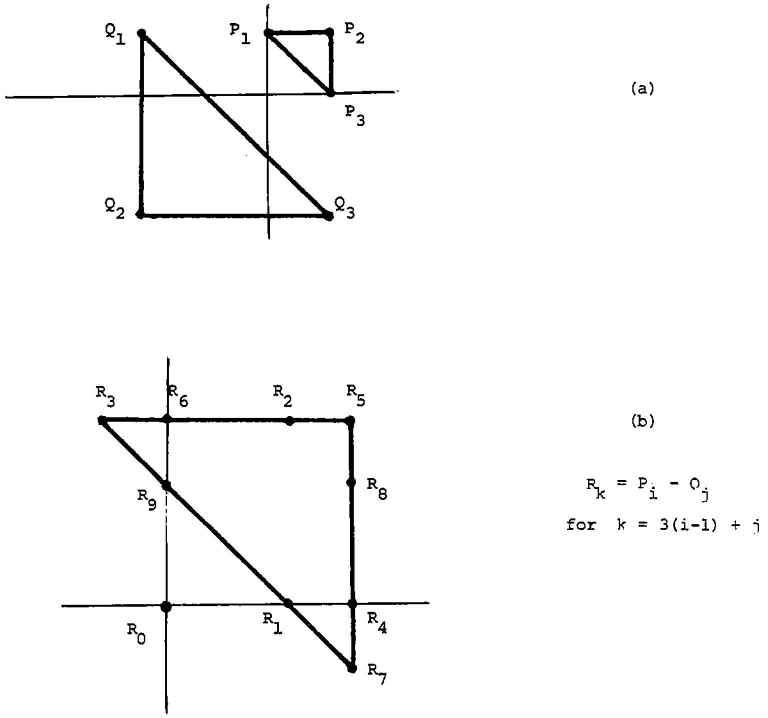


Figure 9. A Set-Set Separation Problem and the corresponding Point-Set Separation Problem.

SPHERICAL SEPARATION

Given: Points $P_1,\ldots, P_n, Q_1,\ldots, Q_m$ on S^{d-1} in E^d.

Determine: If $\{P_1,\ldots, P_n\}$ is separable from $\{Q_1,\ldots, Q_m\}$.

In addition, we can restrict ourselves to one set of points and test separation between it and its reflection through the origin. This yields

HEMISPHERE SEPARATION

Given: Points $P_1,\ldots, P_n$ on S^{d-1} in E^d.

Determine: If $\{P_1,\ldots, P_n\}$ is separable from $\{-P_1,\ldots, -P_n\}$.

All of the separability problems mentioned thus far have been stated as recognition problems. We can restate each of them as computational problems. Here we require that we not only determine if a separating hyperplane exists, but actually find one if it does. This gives us the problems

FINDING POINT-SET SEPARATION

Given: Points $P_0, P_1, \ldots, P_n$ in E^d.

Find: A hyperplane separating $\{P_0\}$ from $\{P_1,\ldots, P_n\}$.

FINDING SET-SET SEPARATION

Given: Points $P_1,\ldots, P_n, Q_1,\ldots, Q_m$ in E^d.

Find: A hyperplane separating $\{P_1,\ldots, P_n\}$ from $\{Q_1,\ldots, Q_m\}$.

FINDING SPHERICAL SEPARATION

Given: Points $P_1,\ldots, P_n, Q_1,\ldots, Q_m$ on S^{d-1} in E^d.

Find: A hyperplane separating $\{P_1,\ldots, P_n\}$ from $\{Q_1,\ldots, Q_m\}$.

FINDING HEMISPHERE SEPARATION

Given: Points $P_1,\ldots, P_n$ on S^{d-1} in E^d.

Find: A hyperplane separating $\{P_1,\ldots, P_n\}$ from $\{-P_1,\ldots, -P_n\}$.

Again we will show that both the computational and the recognition forms of these separation problems are LP-complete.

Section V: Main Result

We are now ready to present our main result:

Theorem 1: The following problems are LP-complete:

(1) Linear Programming

(2) General Linear Programming

(3) Dual Linear Programming

(4) Bounded Linear Programming

(5) Bounded Positive Linear Programming

(6) Linear Inequalities

(7) Bounded Positive Linear Inequalities

(8) Simplified Linear Programming I

(9) Simplified Linear Programming II

(10) Relevancy

(11) Linear Programming Complement

(12) Geometric Linear Programming

(13) Bounded Positive Geometric Linear Programming

(14) Bounded Positive Geometric Linear Inequalities

(15) Dual Geometric Linear Programming

(16) Intersection of Halfspaces

(17) Intersection of Flat and Orthant

(18) Intersection of Common Halfspaces

(19) Extreme Point

(20) Origin Interior Problem

(21) Hemisphere Problem

(22) Hyperplane-Halfspace Intersection

(23) Point-Set Separation

(24) Set-Set Separation

(25) Spherical Separation

(26) Hemisphere Separation

(27) Finding Point-Set Separation

(28) Finding Set-Set Separation

(29) Finding Spherical Separation

(30) Finding Hemisphere Separation.

PROOF OF MAIN THEOREM

Preliminary to proving the main theorem, we prove 21 basic reductions.

1. Linear Programming is LP-complete

This is trivially true based on the definition of LP-complete.

2. General Linear Programming is LP-complete

It suffices to show that any instance of General Linear Programming can be transformed into a problem like LP and vice versa. This is accomplished by showing

(1) Integer inputs can be used in place of rational ones;

(2) $A'x \le b'$ can be used in place of $Ax \ge b$;

(3) $A'x' \ge b'$ can be used in place of $Ax=b$, $x \ge 0$;

(4) $c'^T x$ maximized can be used in place of $c^T x$ minimized.

That integers can be used for input rather than rationals follows since we can express all rationals as ratios and then multiply A, b and c by the greatest common denominator of the ratios involved and still have the same problem. In place of $Ax \ge b$, let $A' = -A$ and $b' = -b$. Then $A'x \le b'$ if and only if $Ax \ge b$. In place of $Ax=b$, $x \ge 0$ we can let A' contain the constraints $x \ge 0$, $Ax \ge b$, and $-Ax \ge -b$. Moreover, constraints of the form $Ax \ge b$ can be replaced by constraints of the form $Ax=b$, $x \ge 0$ using the technique of slack variables [4]. Finally $c^T x$ is maximized if and only if $(-c)^T x$ is minimized. This can be seen in Figures 1, 2, and 3.

3. Dual Linear Programming ≡ LP

This is a classical result in the theory of linear programming (see [4] and Figure 4).

4. Simplified Linear Programming II ∝ LP

This follows since the former problem is just a restricted form of the latter.

5. LP ∝ Bounded Linear Programming

First of all, classical transformations can be used to change LP to a problem which includes the constraints $x \ge 0$ [see 4]. Then we can add a constraint $\sum_i x_i \le M$ for some sufficiently large M, and hence each variable, x_i, is constrained as $0 \le x_i \le M$. Such an M exists for it can be shown [4] that if the problem has a solution, then that solution corresponds to solving a system of equations $A'x=b$ for some submatrix A' of A, and hence each variable is bounded by the largest possible determinant of a submatrix of A, which in turn is bounded by $(d!)(a^d)$ where a is the maximum magnitude of any element of A. See also Figure 5.

6. Bounded Linear Programming ∝ Bounded Positive Linear Programming

Let the original problem be

$$\begin{array}{l} a_{1,1}x_1 + \dots + a_{1,n}x_n \le b_1 \\ \vdots \qquad\qquad\quad \vdots \\ a_{m,1}x_1 + \dots + a_{m,n}x_n \le b_m \\ \qquad \text{Maximize} \quad c_1x_1 + \dots + c_nx_n. \end{array}$$

Now if all $b_i > 0$ we are done. If not, choose r such that $r > -b_i$ for all i. Then consider the new problem

$$\begin{array}{rcl}
a_{1,1}x_1 + \cdots + a_{1,n}x_n + x_{n+1} & \leq & b_1 + r \\
\vdots \qquad\quad \vdots \qquad \vdots & & \vdots \\
a_{m,1}x_1 + \cdots + a_{m,n}x_n + x_{n+1} & \leq & b_m + r \\
x_{n+1} & \leq & r \\
- x_{n+1} & \leq & 1
\end{array}$$

$$\text{Maximize} \quad c_1x_1 + \cdots + c_nx_n + Mx_{n+1} - Mr.$$

This new problem remains bounded since $-1 \leq x_{n+1} \leq r$ and all other variables remain bounded. Moreover, this problem is equivalent to the original if $x_{n+1} = r$. M here is chosen to be larger than the maximum bounded value of c^Tx of the original problem. Now,.if we are able to solve the new problem, then, by our choice of M, if it has a solution with $x_{n+1} = r$, such a solution must be chosen. Hence it suffices to solve the new problem and if $x_{n+1} = r$ in the solution we are done, while, if not, there is no solution to the original problem. See also Figure 6.

7. a) Linear Inequalities ≡ LP

   b) Bounded Positive Linear Programming ≡ Bounded Positive Linear Inequalities

First note that $LI \propto LP$ is trivial. Hence it suffices to show that we can find the exact solution to a LP problem by testing guesses to a solution via the LI problem. This is done by adding a new constraint $c^Tx \geq B$ to the original problem, to test if the answer is less than B or not. The argument used above to establish an upper bound on a possible value also establishes an upper bound on the denominator of the solution and hence, using binary search techniques, we can approach the solution with fewer than $4d(\log d + \log r)$ applications of LI, where r is the largest magnitude in A and d is the number of variables. Hence $LP \propto LI$. The reduction between Bounded Positive Linear Programming and Bounded Positive Linear Inequalities is essentially the same.

8. Linear Programming Complement is LP-complete

First note that $LI \propto LPC$ trivially. To solve a LPC problem using linear programming, one merely needs to add artificial variables which are all constrained to be greater than or equal to zero, and solve the linear programming problem of minimizing the sum of these new variables. The details and the proof that this is conclusive is fundamental to the classical study of linear programming as it represents phase one of the Simplex algorithm [see 4]. This was also observed by Karp [22].

9. a) Geometric Linear Programming ≡ LP

   b) Bounded Positive Geometric Linear Programming ≡ Bounded Positive Linear Programming

   c) Intersection of Halfspaces ≡ LI

   d) Intersection of Flat and Orthant ≡ Simplified Linear Programming I

   e) Intersection of Common Halfspaces ≡ Simplified Linear Programming II

   f) Bounded Positive Geometric Linear Inequalities ≡ Bounded Positive Linear Inequalities

   g) Hyperplane-Halfspace Intersection ≡ Relevancy

These are all simple reductions as they all represent simple translations from an algebraic problem to the equivalent geometric one. (see Figure 7)

10. a) Dual Geometric Linear Programming ≡ Bounded Positive Geometric Linear Programming

    b) Extreme Point ≡ Hyperplane-Halfspace Intersection

    c) Extreme Point ≡ Bounded Positive Geometric Linear Inequalities

These reductions are based on the geometric duality concept and follow from the properties of the polar dual.

11. Extreme Point $\propto$ Simplified Linear Programming I

It can be shown [9] that P_0 is interior to the convex hull of $P_1,\dots,P_n$ if and only if P_0 can be written as $x_1P_1 + \dots + x_nP_n$ where each $x_i \geq 0$ and $\sum_{i=1}^{n} x_i = 1$. It is clear that the property of P_0 being extreme is not affected by a simple transformation. Hence we can assume that P_0 is the origin. Then it is sufficient to satisfy the constraints $x_1P_1 + \dots + x_nP_n = 0$, $x \geq 0$, and $\sum_{i=1}^{n} x_i = 1$. Suppose we can find a solution that satisfies $x_1P_1 + \dots + x_nP_n = 0$ and $x \geq 0$ such that $\sum_{i=1}^{n} x_i = S \neq 0$. Then we can let $y_i = x_i/S$ and then $y_1P_1 + \dots + y_nP_n = 0$ and $y \geq 0$ and $\sum_{i=1}^{n} y_i = 1$. Hence it is enough to satisfy $x_1P_1 + \dots + x_nP_n = 0$, $x \geq 0$ and $x = 0$ and we are done.

12. Origin Interior Problem $\equiv$ Extreme Point

Note that an Origin Interior Problem is a special case of an Extreme Point problem, and hence, $\propto$ is trivial. We need only show then that any Extreme Point problem can be solved by solving a problem where P_0 is the origin and all the other points lie on the unit sphere. We first can do a transformation to insure that P_0 is the origin. Now P_0 is interior to $P_1,\dots,P_n$ if and only if the origin is interior to the transformed points $P_1',\dots,P_n'$ if and only if $0 = \sum_{i=1}^{n} x_iP_i'$ where $x \geq 0$ and $\sum_{i=1}^{n} x_i = 1$. Let $Q_i = P_i'/|P_i'|$, be a point on the unit sphere corresponding to P_i'. Now let $r = \sum_{i=1}^{n} x_i|P_i'|$ and for each i, let $y_i = x_i|P_i'|$ and let $z_i = y_i/r$. Then $x_i \geq 0$ if and only if $y_i \geq 0$ if and only if $z_i \geq 0$ as $r \geq 0$. Also

$$0 = \sum_{i=1}^{n} x_iP_i' = \sum_{i=1}^{n} \frac{y_i}{|P_i'|} P_i' = \sum_{i=1}^{n} y_iQ_i = (1/r) \sum_{i=1}^{n} y_iQ_i = \sum_{i=1}^{n} z_iQ_i$$

while $\sum_{i=1}^{n} z_i = 1$. Hence the origin is interior to $P_1,\dots,P_n$ if and only if it is interior to $Q_1,\dots,Q_n$, and we are done. See Figure 8.

13. Hemisphere Problem $\equiv$ Origin Interior Problem

It is easy to see that these are actually the same problem as the origin is an extreme point if and only if there is a supporting hyperplane through the origin. But such a hyperplane exists if and only if the points on the unit sphere share a common hemisphere. See Figure 8.

14. Point-Set Separation $\equiv$ Extreme Point

This is trivial as it follows from the definitions of extreme point and separation that P_0 is separable from $\{P_1,\dots, P_n\}$ if and only if P_0 is extreme with respect to $\{P_1,\dots, P_n\}$.

15. a) Spherical Separation $\propto$ Set-Set Separation

b) Hemisphere Separation $\propto$ Spherical Separation

c) Finding Spherical Separation $\propto$ Finding Set-Set Separation

d) Finding Hemisphere Separation $\propto$ Finding Spherical Separation

These reductions all follow as in each case one problem is a special case of the other.

16. Set-Set Separation $\propto$ Point-Set Separation

It is sufficient to show how to express a given Set-Set Separation problem as a Point-Set Separation problem. Let the two sets to be separated be $\{P_1,\dots, P_n\}$ and $\{Q_1,\dots, Q_m\}$. Let P be the convex hull of

$P_1,\ldots,P_n$ and Q be the convex hull of $Q_1,\ldots,Q_m$. Then the sets are separable if and only if $P \cap Q$ is empty. Define

$$P-Q = \{x | x=p-q,\ p\in P \text{ and } q\in Q\}\ .$$

Then $P \cap Q$ is empty if and only if the origin is exterior to $P-Q$, or alternatively, if the origin is an extreme point of $P-Q$. It can be shown [9, 30] that $P-Q$ is the convex hull of U where

$$U = \{u | u = P_i - Q_j \text{ for } 1 \le i \le n \text{ and } 1 \le j \le m\}\ .$$

Hence it is sufficient to separate the origin from the set U, and since $|U| = mn$, we are done. See Figure 9.

17. Hemisphere Problem $\propto$ Hemisphere Separation

Let the points on the unit sphere be $P_1,\ldots,P_n$. Then these lie interior to some hemisphere if and only if there is a rotation such that every first coordinate is greater than zero. Then the hyperplane with first coordinate zero separates these points from their negatives, and we are done.

18. Hemisphere Separation $\propto$ Finding Hemisphere Separation

This is true since finding the separating hyperplane determines separability.

19. Finding Set-Set Separation $\propto$ Finding Point-Set Separation

Here we can use the same techniques as in our reduction from Set-Set Separation to Point-Set Separation (reduction 16). We can thus find a hyperplane $E_0 = \{x\in E^d \mid (a,x)=b_0\}$ that separates the origin from $P-Q$. Now we assume, without loss of generality, that for all points y in $P-Q$, $(a,y)<0$. Thus for all p in P and all q in Q we have

$$(a,p) - (a,q) < 0 \quad \text{or} \quad (a,p) < (a,q)\ .$$

Next we compute $r = \max_i\ (a,P_i)$ and $s = \min_i\ (a,Q_i)$. Then $r<s$. Finally, let $t = (r + s)/2$, and let

$$E = \{x\in E^d \mid (a,x)=t\}\ .$$

Now we claim that E separates P from Q. For any p in P and q in Q we have $(a,p) \le r < t < s \le (a,q)$.

20. Finding Point-Set Separation $\propto$ Dual Gemoetric Linear Programming

Let $\{P_1,\ldots, P_n\}$ be the set of points we are separating from P_0. Then $Q = (1/n) \sum_{i=1}^{n} P_i$ is interior to the convex hull of $P_1,\ldots,P_n$. We can transform all the points such that Q is the origin. Let r be the ray from Q through P_0 under the transformation. We can now find the facet F that this ray passes through. Given this facet, its affine hull is its supporting hyperplane. Let this hyperplane be $F_0 = \{x \mid (a,x)=b_0\}$. Let a parallel hyperplane containing the point P_0 be $E_1 = \{x \mid (a,x)=b_1\}$. Then let $E_2 = \{x \mid (a,x)=(b_0+b_1)/2\}$. Then E_2 is a separating hyperplane between P_0 and $\{P_1,\ldots, P_n\}$.

21. Simplified Linear Programming I $\propto$ Simplified Linear Programming II

It sufficies to show that we can solve the system $Ax=0,\ x\neq 0,\ x\ge 0$ by solving some other system $Bx\ge 0,\ x\neq 0$. Let the matrix B be a basis for the null space of A. Such a B can be found in deterministic polynomial time using techniques of linear algebra. Then the conditions $Ax=0,\ x\neq 0,\ x\ge 0$ can be rewritten as $x=By,\ By\neq 0,\ By\ge 0$ for some vector y. In particular, as y exists if and only if x exists and we can compute x given y, it suffices to find a y such that $By \ge 0$ and $By\neq 0$, and we are done.

We can now prove the theorem using these reductions as follows:

Statement	Reductions
Linear Programming $\equiv$ LP	1
General Linear Programming $\equiv$ LP	2
Dual Linear Programming $\equiv$ LP	3
Linear Inequalities $\equiv$ LP	7a
Linear Programming Complement $\equiv$ LP	8
Geometric Linear Programming $\equiv$ LP	9a
Intersection qf Halfspaces $\equiv$ LI	9c
LP $\propto$ Bounded Linear Programming $\propto$ Bounded Positive Linear Programming $\propto$ Bounded Positive Linear Inequalities $\propto$ Bounded Positive Geometric Linear Inequalities $\propto$ Extreme Point $\propto$ Simplified Linear Programming I $\propto$ Simplified Linear Programming II $\propto$ LP	5,6,7b,9f,10c,11,21,4
Bounded Positive Geometric Linear Programming $\equiv$ Bounded Positive Linear Programming	9b
Intersection of Flat and Orthant $\equiv$ Simplified Linear Programming I	9d
Intersection of Common Halfspaces $\equiv$ Simplified Linear Programming II	9e
Dual Geometric Linear Programming $\equiv$ Bounded Positive Geometric Linear Programming	10a
Hyperplane-Halfspace Intersection $\equiv$ EP	10b
Origin Interior Problem $\equiv$ EP	12
Hemisphere Problem $\equiv$ Origin Interior Problem	13
Point-Set Separation $\equiv$ EP	14
Relevancy $\equiv$ Hyperplane-Halfspace Intersection	9g
Hemisphere Problem $\propto$ Hemisphere Separation $\propto$ Spherical Separation $\propto$ Set-Set Separation $\propto$ Point-Set Separation	17,15b,15a,16

Hemisphere Separation $\propto$ Finding Hemisphere Separation $\propto$ Finding Spherical Separation $\propto$ Finding Set-Set Separation $\propto$ Finding Point-Set Separation $\propto$ Dual Geometric Linear Programming	18,15d,15c,19,20

Section VI: Complexity of Linear Programming

The purpose of this section is not to introduce any new results, but rather to take some known results and show their relevance to the various problems that are LP-complete. It should first be noted that fast (O(n) or O(nlogn)) algorithms exist for most of the problems mentioned here when these problems are restricted to two and in some cases three dimensions [8,11,26,28]. However, most of these algorithms depend on constructing the convex hull of a set of n points, and hence can only be extended to d dimensions at a cost of becoming exponential.

In a more positive sense though, one class of algorithms stands out in relation to linear programming problems. These are algorithms which first find an approximation to the solution and then methodically proceed to a new approximation until the actual solution is found. The most widely known example of such an algorithm is the Simplex algorithm. This algorithm solves a linear programming problem by finding an initial feasible solution and then, if it does not maximize the objective function, another feasible solution is found and the check for maximizing the objective function is made again. The value of the objective function at the new solution is always greater than or equal to its value at the previous solution. Moreover, under the proper assumptions, the method can be shown to always find the optimal solution if it exists. While the Simplex algorithm itself solves a linear programming problem, it is generally possible with the problems presented in this paper to find a similar algorithm to solve the particular problem [1, 2, 24, 25, 31].

Moreover, the nature of our notion of reducibility allows us to consider any of the LP-complete problems as a linear programming problem and then to apply the Simplex algorithm to solve it.

Empirical evidence for the complexity of the Simplex algorithm shows it to be quite efficient, usually running in time linear with the number of constraints and variables [4, 17, 23]. However, in recent years it has been shown that there are cases where the Simplex algorithm will take exponential time [20, 32]. Thus, although these problems can generally be solved quickly, the best upper bound known for their worst-case complexity remains exponential.

Given this behavior, we know turn to a study of the relationship of LP-complete problems to NP-complete problems. To begin with, we observe that all LP-complete problems belong to NP. This follows as

Lemma 2: LP ϵ NP

Proof: It has been shown [4] that a solution to a bounded linear programming problem with n constraints and d variables corresponds to a set of d of the n constraints. We can nondeterministically obtain these d constraints and thus obtain a solution. We have seen that the problem of Bounded Positive Linear Inequalities is LP-complete. As this problem can be solved by the method described above it is in NP. But then, by theorem 1, all LP-complete problems are in NP and in particular, LP ϵ NP.

This lemma puts an upper bound on the complexity of linear programming. It leaves four broad possibilities for the actual complexity. These can be summarized as

Theorem 2: One of the following is true:

(a) LP $\in$ P $\subsetneq$ NP

(b) LP is P-hard

(c) LP is NP-complete and NP is closed under complement

(d) LP $\in$ P = NP

Proof: If P = NP, then L $\in$ NP if and only if L $\in$ P = NP. Assume then that P $\neq$ NP. Then either LP is doable in polynomial time or it isn't. If it is, then LP $\in$ P $\subsetneq$ NP. If it is not, then either LP is NP-complete or it is not. If it is not then L $\in$ NP-P and LP not NP-complete implies that LP is P-hard. If LP is NP-complete then LI is also NP complete and the complement of LI, LPC, is also NP-complete. But then NP is closed under complement, and the theorem is proved. Parts of this theorem have been alluded to in [22].

Of these four possibilities, the last two must be considered unlikely. Although it is currently unknown whether NP is closed under complement or if P = NP, it is widely believed that both of these statements are false. Hence, we may conjecture that either statement (a) or statement (b) is true. If statement (a) is true, then polynomial time algorithms exist for all of the problems that are LP-complete. This would be interesting in terms of its impact on the study of geometric complexity as well as being of possible practical interest in operations research. If however, statement (b) is true, then we would have demonstrated a natural problem belonging to NP-P which is not NP-complete (assuming P $\neq$ NP). Such a problem would be of considerable interest to theoreticians. Thus, the resolution of theorem 2 will be an important and interesting result no

matter which of the alternatives is true. Hence the problems associated with determining the complexity of linear programming are indeed important problems of complexity theory.

Thus, it seems unlikely that the notions of LP-completeness and NP-completeness are equivalent. Yet, we may extend the observation of Johnson and Preparata [12] to transform each of these problems into an NP-complete problem by merely adding an additional parameter, k. For example, they showed that determining if any subset of size k of a set of n points on S^{d-1} share a common hemisphere is NP-complete. Here we observe that in the case where k = n, the problem becomes LP-complete. Similarly, we could restate NP-complete versions of the other problem. Typical of these statements would be

INTERSECTION OF COMMON HALFSPACES

Given: Halfspaces $H_1, \ldots, H_n$ in E^d such that the origin is interior to their intersection, an integer k.

Determine: If there is a subset consisting of k of the halfspaces which have the origin as their intersection.

EXTREME POINT

Given: A set of points $P_0, P_1, \ldots, P_n$ in E^d, an integer k

Determine: If there exist $1 \leq i_1 < i_2 \ \ldots < i_k \leq n$

INTERSECTION OF HALFSPACES

Given: Closed halfspaces $H_1, \ldots, H_n$, an integer k.

Determine: If there are $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq n$ such that

$$\bigcap_{j=1}^{k} H_{i_j}$$ is non-empty.

RELEVANCY

Given: A set of constraints $(a_0, x) \le b_0, \ldots, (a_n, x) \le b_n$, an integer k.

Determine: If determining some set of k constraints is equivalent to satisfying all of the constraints.

LINEAR INEQUALITIES

Given: An integer $n \times d$ matrix A, integer n-vector b, integer k.

Determine: If there is a rational d-vector x such that k components of $Ax=b$ are negative.

These results, expressing linear programming as an interesting limiting case of integer programming take on added interest as a possible means of finding a hierarchy of natural problems in their complexities. Furthermore, as observed in [5], connections with log-space completeness can also be made.

CONCLUSION

The results in this paper can be divided into two sections. In the first section we presented a list of LP-complete problems. These problems are both interesting and important. Some of them are fundamental to the study of d-dimensional geometry and others, especially those directlv involved with linear programming, have vast practical applications. Secondly, these problems are interesting because they represent a wide range of geometric problems that all have the same intrinsic complexity. Moreover, by our notions of reducibility, information regarding the complexity of any single problem, be it a fast alcorithm or a good lower bound, is directly applicable to all the other LP-complete problems as well.

The second part of the paper was devoted to a survey of known results on the complexity of linear programming. Here we have shown that it is probable that linear programming is either of polynomial complexity or is P-hard. In the light of the efforts that have been made toward finding a polynomial algorithm, and considering that all efforts have failed, it seems most probable that linear programming is indeed P-hard.

Bibliography

1. Balinsky, M. L. An algorithm for finding all vertices of convex polyhedral sets. SIAM J on Appl. Math. (1961), 9, pp. 72-88.
2. Burdet, C-A. Generating all the faces of a polyhedron. SIAM J on Applied Math. (1974), 26, pp. 479-489.
3. Chand, D. R., and Kapur, S. S. An algorithm for convex polytopes. JACM (1970), 17, pp. 78-86.
4. Dantzig, G. B. Linear programming and extensions. 1963, Princeton University Press, Princeton, New Jersey.
5. Dobkin, D., Lipton, R., and Reiss, S. Linear programming is P-complete, submitted for publication.
6. Gale, D. On the number of faces of a convex polytope. Canadian J of Math. (1964), 16, pp. 12-17.
7. Garey, M, Johnson, D., and Stockmeyer, L. Some simplified NP-complete problems. Proceedings of the Sixth Annual ACM Symposium on Theory of Computing. (1974), pp. 47-63.
8. Graham, R. L. An efficientt algorithm for determining the convex hull of a finite planar set. Information Processing Letters. (1972), 1, pp. 132-133.
9. Grunbaum, B. Convex polytopes. 1967, John Wiley and Sons, London.
10. Grunbaum, B. Polytopes, graphs, and complexes. Bull. of the AMS. (1970), 76, pp. 1131-1201.
11. Jarvis, R. A. On the identification of the convex hull of a finite set of points in the plane. Information Processing Letters. (1973), 2, pp. 18-21.

12. Johnson, D., and Preparata, F. private communication.
13. Jones, N., and Laaser, W. Complete problems for deterministic polynomial time. Proceedings of the Sixth Annual ACM Symposium on Theory of Computing. (1974), pp. 40-46.
14. Karp, R. M. Reducibility among combinatorial problems. in R. E. Miller and J. W. Thatcher (eds.), Complexity of computer computations. 1972, Plenum Press, New York.
15. Klee, V. A class of linear programming problems requiring a large number of iterations. Num. Math. (1965), 7, pp. 313-321.
16. Klee, V. Convex polyhedra and mathematical programming. 1974. Univ. of Washington Technical Report #50.
17. Klee, V. Convex polytopes and linear programming. in Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems. March 16-18, 1964, IBM Data Processing Division, White Plains, New York, 1966, pp. 123-158.
18. Klee, V. Heights of convex polytopes. Mathematical Analysis and Application. (1965), 11, pp. 176-190.
19. Klee, V. On the number of vertices of a convex polytope. Canadian J of Math. (1964), 16, pp. 701-720.
20. Klee, V., and Minty, G. J. How good is the simplex algorithm. in O. Shisha (ed.), Inequalities, Vol. III. 1972. Academic Press, New York. pp. 159-175.
21. Kuhn, H. W., and Tucker, A. W. Linear inequalities and related systems. Annals of Mathematics Studies 38. 1956, Princeton University Press, Princeton, NJ.
22. Ladner, R. E. On the structure of polynomial time reducibility. JACM. (1975), 22, pp. 155-171.

23. Liebling, T. M. On the number of iterations of the simplex method. Institut fur Operations Research der ETHZ.

24. Mattheiss, T. H. An algorithm for determining irrelevant constraints and all vertices in systems of linear inequalities. Operations Research. (1973), 21, pp. 247-260.

25. McRae, W. B., and Davidson, E. R. An algorithm for the extreme rays of a pointed convex polyhedral cone. SIAM J on Comput. (1973), 2, pp. 281-293.

26. Preparata, F. P., and Hong, S. J. Convex hulls of finite planar and spatial sets of points. UILU-ENG 75-2217, 1975.

27. Shamos, M. I. Closest-point problems. 1975. Proceedings of the 16th Annual Symposium on Foundations of Computer Science. pp. 151-162.

28. Shamos, M. I. Computational Geometry. PhD. Thesis, Yale University, New Haven, Conn., to appear.

29. Shamos, M. I. Geometric complexity. Proceedings of the Seventh Annual ACM Symposium on Theory of Computing. (1975), pp. 224-233.

30. Stoer, J., and Witzgall, C. Convexity and optimization in finite dimensions I. 1970, Springer-Verlag, New York.

31. Wets, R J-B., and Witzgall, C. Algorithms for frames and linearity spaces of cones. J of Research of the National Bureau of Standards. (1967), 71b, pp. 1-7.

32. Zadeh, N. A bad network problem for the simplex method and other minimum cost flow algorithms. Mathematical Programming. (1973), 5, pp. 255-266.