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 Knuth, region location in planar graphs, and speech recognition.

Keywords

binary search, secondary search, efficient algorithms, planar graphs, finite element methods

Multidimensional Searching Problems

David Dobkin and Richard J. Lipton

Research Report #34

October 1974

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:

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

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 $\exists\ 1 \leq i \leq m\ [(x,y) \text{ is on } L_i]$ in $3g(m)$ steps.

Proof. Let the intersections of the lines be given by the points $z_1, \ldots, z_n$ $(n \leq \frac{m(m-1)}{2})$ 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:

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

(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

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

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 $\exists\ 1 \leq i \leq m[x \text{ is on } \theta_i]$ 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 .

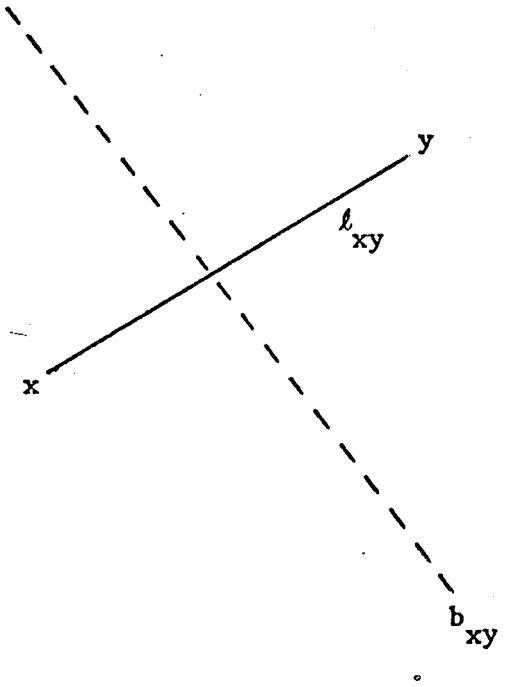

FIGURE 1. b_{xy} is the perpendicular bisector of ℓ_{xy}. Therefore, points below b_{xy} are closer to x than y and points above b_{xy} are closer to y than x.

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 $0(m/g(m))$ 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. Complexity of Computer Computations, ed. by R. Miller and J. Thatcher, Plenum Press, 1972.