Programming Techniques
S.L. Graham, R.L. Rivest Editors

On the Complexity of Computing the Measure of $\cup[a_i, b_i]$

Michael L. Fredman
The University of California at San Diego

Bruce Weide
Carnegie-Mellon University

The decision tree complexity of computing the measure of the union of n (possibly overlapping) intervals is shown to be Ω(n log n), even if comparisons between linear functions of the interval endpoints are allowed. The existence of an Ω(n log n) lower bound to determine whether any two of n real numbers are within ϵ of each other is also demonstrated. These problems provide an excellent opportunity for discussing the effects of the computational model on the ease of analysis and on the results produced.

Key Words and Phrases: analysis of algorithms, combinatorial problems, computational complexity, computational models, decision tree programs, lower bounds

CR Categories: 5.25, 5.26, 5.30, 5.39

The contribution of M. L. Fredman was supported in part by the National Science Foundation under grant number MCS 76-08543. That of Bruce Weide was supported in part by a National Science Foundation Graduate Fellowship. The main theorem of this paper was proved by the authors independently. They have since combined and edited their original manuscripts to produce this one.

Authors' addresses: Michael L. Fredman, Computer Science Division, Department of Applied Physics and Information Science, University of California at San Diego, La Jolla, CA 92093; Bruce Weide, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

I. Introduction

Victor Klee [5] has asked whether computing the measure of the union of n (possibly overlapping) intervals $[a_i, b_i]$ requires $\Omega(n \log n)$ steps.[1] He shows that $O(n \log n)$ comparisons are sufficient, and we show this number to be necessary, even if comparisons between linear functions of the inputs are allowed.

In Section II, we introduce the decision tree model with linear comparisons. Section III contains two proofs that the complexity of computing $\cup[a_i, b_i]$ is $\Omega(n \log n)$ under this model of computation. Part of one proof establishes the same bound for deciding whether any two of n real numbers are within ϵ of each other. In Section IV we give an algorithm for Klee's interval problem which shows that the bound proved in Section III is nearly the best possible; i.e., the complexity of the algorithm is within $O(n)$ of the lower bound. Finally, in Section V we consider other possible computational models which might be reasonable for these problems, and the effects of such alternative models on lower and upper bounds are discussed.

II. The Decision Tree Model with Linear Tests

A decision tree program with linear comparisons takes as input a list of real numbers $\{x_i\}$, which are considered to be indeterminates. Each internal and external (leaf) node of the tree is labeled with a linear function of these inputs. The algorithm is executed with control beginning at the root node. In general, when control is centered at any internal node, the linear function labeling that node is evaluated and the result is compared to zero. If it is greater than zero, control passes to the left son; otherwise, it passes to the right son. When control reaches a leaf node the linear function there is evaluated and put out as the answer. In a decision problem, where the answer is simply "yes" or "no" rather than a real number, each leaf node is labeled with "yes" or "no" rather than with a function. The complexity of such an algorithm is defined as the depth of the tree, which is simply the number of comparisons required in the worst case.[2]

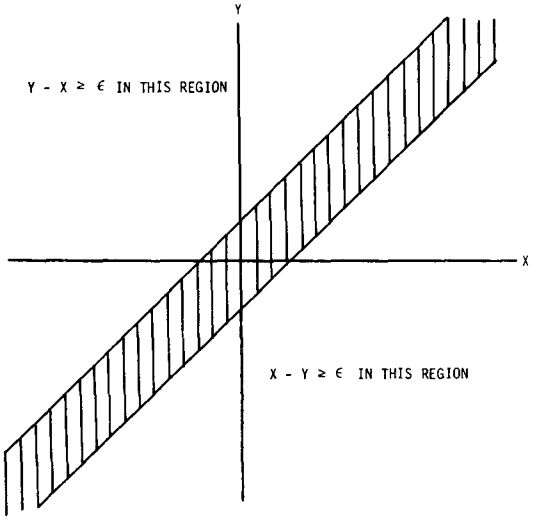
III. An $\Omega(n \log n)$ Lower Bound

For convenience, let us call the problem presented in [5] the MEASURE problem. Namely, given a list of $2n$ real numbers $\{a_i\}$ and $\{b_i\}$ which represent the endpoints of the n intervals $[a_i, b_i]$, compute the measure of the union of these intervals. This section is devoted to proving the following theorem in two different ways.

[1] We say $g(n) = O(f(n))$ if there is a constant c such that $|g(n)| \leq cf(n)$ for all sufficiently large n. Similarly, $g(n) = \Omega(f(n))$ if there is a constant $c > 0$ such that $|g(n)| \geq cf(n)$ for all sufficiently large n.

[2] By pruning the tree if necessary, we may assume without loss of generality that for each leaf there exists a sequence of inputs which follows the path through the tree leading to that leaf.

Fig. 1. The regions comprising L when $n = 2$.



THEOREM 1. MEASURE is of decision tree complexity $\Omega(n \log n)$, even if comparisons between linear functions of the inputs are allowed.

PROOF 1. Let the ϵ-CLOSENESS problem be defined as the task of determining whether any two of n real numbers $\{x_i\}$ are within ϵ of each other, where ϵ is a fixed parameter of the problem. The proof is based on two lemmas: The first will show how an algorithm for MEASURE could be used to solve ϵ-CLOSENESS, and the second will provide a lower bound on the complexity of solving ϵ-CLOSENESS. Together, these facts will enable us to establish the lower bound for MEASURE.

LEMMA 1: If MEASURE can be solved with $T(n)$ comparisons, then ϵ-CLOSENESS can be solved with $T(n) + 1$ comparisons.

PROOF: Given any algorithm for MEASURE which operates with $T(n)$ comparisons, an ϵ-CLOSENESS algorithm is easily constructed.

(1) With inputs $x_1, x_2, \ldots, x_n$ and parameter $\epsilon > 0$, form the intervals $[x_i, x_i + \epsilon]$, for $i = 1, 2, \ldots, n$.

(2) Find the measure of $\cup[x_i, x_i + \epsilon]$.

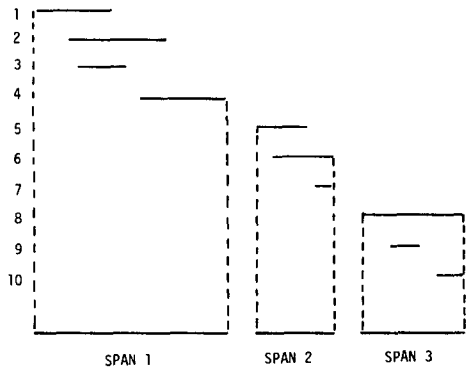
(3) If the answer from part (2) is exactly $n\epsilon$, then no two points are within ϵ of each other. If the answer is anything else, then there are two points within ϵ of each other.

The total number of comparisons required by this ϵ-CLOSENESS algorithm is $T(n) + 1$. Step (1) uses no comparisons, step (2) uses $T(n)$ by hypothesis, and step (3) requires just one more comparison. □

LEMMA 2. ϵ-CLOSENESS is of decision-tree complexity $\Omega(n \log n)$, even if comparisons between linear functions of the inputs are allowed.

PROOF: The argument is very much like that used in [3] to show that $\Omega(n \log n)$ comparisons are required by any decision tree program with linear tests to determine whether any two of n real numbers are equal. Consider the inputs $x_1, x_2, \ldots, x_n$ to be the coordinates of a point in n-dimensional Euclidean space E^n. Let the locus of points L be the set of input points for which $|x_i - x_j| \geq \epsilon$, for all $i \neq j$. Then ϵ-CLOSENESS is equivalent to the problem of determining membership in L.

Fig. 2. In this case, the 10 intervals form 3 spans.



The set L is the union of the disjoint and noncontiguous regions

$$L_\pi = \{x_{\pi(1)} < x_{\pi(2)} < \ldots < x_{n(n)} \text{ and } |x_i - x_j| \geq \epsilon \text{ for all } i \neq j\}.$$

where π is a permutation of the integers $1, 2, \ldots, n$. The number of such regions is $n!$ because there are $n!$ such permutations, each corresponding to a different region (see Figure 1 for the case $n = 2$).

To see that $L = \cup L_\pi$, note first that any point in L has every pair of coordinates differing by at least ϵ, so any such point belongs to L_π for some permutation π of the ranks of the coordinates. Similarly, any point in $\cup L_\pi$ has $|x_i - x_j| \geq \epsilon$ for all $i \neq j$, so such a point is, by definition, in L. The regions L_π are clearly disjoint because only one permutation can correspond to any point with distinct coordinate values. The regions are also noncontiguous because every pair of regions is separated by (at least) the hyperplane $x_i = x_j$, for some pair of coordinates i and j which are in a different order in the two permutations. We have therefore established that L consists of $n!$ disjoint, noncontiguous regions in E^n.

Now consider a decision tree algorithm for deciding membership in L. At any leaf node, the algorithm must answer "yes" or "no" to the question of whether $x_1, x_2, \ldots, x_n$ are the coordinates of a point in L. Let the set of points "accepted" at leaf node k be denoted by T_k (i.e., T_k is the set of points for which all comparisons in the tree have identical outcomes and lead to leaf node k, for which the algorithm answers "yes"). The leaf nodes of the tree partition E^n into disjoint convex regions because all comparisons are between linear functions of the coordinates of the input point, so in particular each of the accepting sets T_k is convex.

Let m be the number of accepting leaf nodes, and suppose $m < n!$. Then since L consists of $n!$ disjoint regions, some accepting node T_k must accept points in two regions, say L_α and L_β. Choose any points $P_1 \in T_k \cap L_\alpha$ and $P_2 \in T_k \cap L_\beta$. By the convexity of T_k, every point on the line P_1P_2 is in T_k, so for every such point the algorithm answers "yes." However, L_α and L_β are

disjoint and noncontiguous, so the line P_1P_2 contains points not in L. The algorithm must therefore fail to operate correctly if $m < n!$, so there must be at least $n!$ accepting leaf nodes.

The total number of leaf nodes must be at least $n!$, since that many are accepting, so the height of the decision tree (and hence the number of comparisons along some path) must be at least $[\log_2 n!] = \Omega(n \log n)$. The worst-case complexity of ϵ-CLOSENESS is therefore $\Omega(n \log n)$. A similar argument based on the external path length of the tree (see [6]) shows that ϵ-CLOSENESS also requires $\Omega(n \log n)$ comparisons on the average, assuming that the inputs $\{x_i\}$ are considered in random order. We note that this bound can be used to prove similar lower bounds for some nearest-neighbor search problems (for example, the problems in [2] and [8]). □

Now the main theorem follows immediately. By Lemmas 1 and 2, $T(n) + 1 = \Omega(n \log n)$, where $T(n)$ is the time required to solve MEASURE. This implies that $T(n) = \Omega(n \log n)$. □

PROOF 2: The first proof relies on problem reduction, thereby introducing the auxiliary problem ϵ-CLOSENESS, which is a decision problem. However, the main theorem can be proved directly, as follows. The argument is similar to that used in [4].

Let I_i denote the interval $[i, i + 1]$, $1 \leq i \leq n$. We define a class C consisting of the $n!$ sequences of n such intervals. For each permutation π on $\{1, 2, \ldots, n\}$, the interval sequence $J_1, J_2, \ldots, J_n$, where $J_i = I_{\pi(i)}$, is included in C. Thus, all intervals of a sequence in C have length 1, and the measure of the union of the intervals in any such sequence is n. We argue below that no two sequences in C can define the same path through the decision tree representing a correct algorithm for MEASURE. It will then follow that such a tree must have at least $n!$ leaf nodes, thereby proving the theorem.

Suppose that there exist two distinct sequences $J_1, J_2, \ldots, J_n$ and $J'_1, J'_2, \ldots, J'_n$ in C which define the same path through the tree, and let π and π' be their associated permutations. The inputs $\{a_i\}$ and $\{b_i\}$ can be thought of as the coordinates of a point in $2n$-dimensional Euclidean space; let P and P' be the points in this space associated with $J_1, J_2, \ldots, J_n$ and $J'_1, J'_2, \ldots, J'_n$, respectively. The liner inequalities along the common path followed for the input sequences $J_1, J_2, \ldots, J_n$ and $J'_1, J'_2, \ldots, J'_n$ define a convex region of $2n$-space to which the interval endpoints of both sequences belong. Any point P'' on the line segment joining P and P' defines a sequence of unit-length intervals $J''_1, J''_2, \ldots, J''_n$, which also follows the same path through the tree. Since the output value associated with the leaf node at the end of this path is a linear function of the endpoints of the intervals, the output produced for $J''_1, J''_2, \ldots, J''_n$ will be n, because the output is n for $J_1, J_2, \ldots, J_n$ and for $J'_1, J'_2, \ldots, J'_n$.

However, if we imagine P'' sliding continuously along the line segment from P to P', the intervals $J''_1, J''_2, \ldots, J''_n$ will shift continuously, and there will be an instant at which two previously different intervals J''_r and J''_s, $r \neq s$, will exactly coincide. (These may be any two intervals r and s for which $\pi(r) - \pi(s)$ and $\pi'(r) - \pi'(s)$ have different signs.) At such an instant, the measure of the union of $J''_1, J''_2, \ldots, J''_n$ cannot exceed $n - 1$, contradicting the value n produced as output by the hypothetically correct tree. This contradiction shows that no two sequences of C can define the same path through the tree, which, as explained above, proves the theorem. □

IV. More Precise Bounds

Let $T(n)$ be the minimum complexity among all decision tree algorithms which solve MEASURE using linear tests. In this section we prove the following result.

THEOREM 2. For $T(n)$ defined as above,

$$n \log_2 n - 1.443n + O(\log n) \leq T(n) \leq n \log_2 n + .671n + O(\log n)$$

PROOF: The lower bound, being the expansion of $\log_2 n!$, follows immediately from the arguments of Section III.

The following algorithm for MEASURE provides an upper bound on $T(n)$. As before, the input consists of n intervals $[a_i, b_i]$, with $a_i \leq b_i$. The endpoints $\{a_i\}$ and $\{b_i\}$ are given in the arrays A and B, respectively. After sorting the intervals by left endpoints (keeping the right endpoints together with their corresponding left endpoints), $A[i]$ contains the ith smallest left endpoint, and $B[i]$ contains the right endpoint corresponding to $A[i]$. The left endpoints are then scanned in increasing order, while the algorithm keeps track of the current "span" of intervals considered so far (see Figure 2). Whenever a new interval has its left endpoint greater than the rightmost point of the current span, the total measure is updated and a new span is started.

```
begin
(1) sort A in increasing order, keeping B[i] with A[i];
    measure := 0;
    leftspan := A[1];
    rightspan := B[1];
    for i := 2 until n do
(2) if A[i] ≤ rightspan
(3) then rightspan := max (rightspan, B[i])
      else begin
(4)       measure := measure + rightspan − leftspan;
          leftspan := A[i];
          rightspan := B[i]
    end;
(5) measure := measure + rightspan − leftspan;
    output (measure)
end
```

All comparisons are between linear functions of the inputs. These comparisons occur in the statements numbered (1), (2), and (3). Statement (1) costs at most $n \log_2 n - 1.329n + O(\log n)$ comparisons if sorting is done using the merge-insertion algorithm (see [6], section 5.3.1, exercise 15). Statement (2) is encountered exactly $n - 1$ times, at a cost of one comparison each time, while

statement (3) could be executed at most $n - 1$ times, again at a cost of a single comparison each time. The total number of comparisons used by this algorithm therefore satisfies

$$T(n) \leq n \log_2 n + 0.671n + O(\log n),$$

completing the proof. We see that $T(n)$ is determined to within $2.114n + O(\log n)$ comparisons in the decision tree model with linear tests. □

It is of some interest that the algorithm given in [5] is only within a factor of two of the lower bound, since it involves sorting all $2n$ endpoints together, while the algorithm presented here sorts only the left endpoints.

V. Remarks About the Model of Computation

The typical (if not altogether realistic) restriction to comparisons between linear functions of the inputs is used in both proofs to establish the convexity of the regions identified with the leaf nodes of a decision tree. If this restriction is removed, or the decision tree model is abandoned entirely, then bounds are more difficult to obtain.

Jon Bentley and Michael Shamos note that ϵ-CLOSENESS can be solved in linear time under a different model of computation: a RAM (Random Access Machine) with exact real arithmetic, including division, "floor" function $\lfloor x \rfloor$,[3] and unbounded storage. An algorithm which accomplishes this is due to Yuval [8]. Aho, Hopcroft, and Ullman ([1], exercise 2.12) show how to avoid initializing unused memory locations, so that even though the amount of storage must be unbounded, this does not increase the asymptotic order of the time required by the algorithm.

The key to the algorithm is the computation of the floor function, which is available as a single operation on most computers. It is, however, merely a consequence of the radix representation of numbers on actual digital machines which makes this possible. In general, a piecewise-linear function like $\lfloor x \rfloor$ is a perfectly acceptable quantity to compute in our decision tree model of computation, so long as a legitimate cost function is applied. For a piecewise-linear function with p pieces, the cost should be at least $\lceil \log_2 p \rceil$ comparisons. The fact that $\lfloor x \rfloor$ is an "elementary" operation on real computers is a happy coincidence. (Note also that even while the floor function helps solve ϵ-CLOSENESS, it is not clear how it could help solve MEASURE.)

Similarly, if we are allowed to compute products of the inputs, then ϵ-CLOSENESS can be solved with only $2n - 1$ comparisons, as follows. For any collection of nonoverlapping intervals $[a_i, b_i]$, a number z will not fall inside any of these intervals if and only if

$$\prod_i (a_i - z) \cdot (b_i - z) \geq 0$$

since an odd number of terms in the product will be negative if and only if z lies inside one of the intervals. Solving ϵ-CLOSENESS for the inputs $\{x_i\}$ is equivalent to asking whether the intervals $[x_i, x_i + \epsilon]$ are nonoverlapping. Assuming that the x_i are distinct and that the first k intervals are nonoverlapping, the first $k + 1$ intervals will be nonoverlapping if and only if both x_{k+1} and $x_{k+1} + \epsilon$ do not fall inside any of the first k intervals. (Since all intervals under consideration have length ϵ, it is impossible for the $k + 1$th interval to contain one of the other intervals.)

The following program uses these ideas to decide ϵ-CLOSENESS using only $2n - 1$ comparisons. The first step of the program, which tests whether any of the x_i are equal, also illustrates how the element uniqueness problem can be solved using only one comparison if products may be computed and compared. If two of the x_i are within ϵ of each other, the program outputs "yes"; otherwise, it outputs "no."

```
begin
  if ∏_{i<j} (x_i − x_j) = 0 then output ("yes");
  for j := 1 until n do y_j := x_j + ε;
  J := 2;
  while j ≤ n
    and ∏_{i<j} (x_i − x_j)·(y_i − x_j) ≥ 0
    and ∏_{i<j} (x_i − y_j)·(y_i − y_j) ≥ 0 do j := j + 1;
  if j ≤ n then output ("yes") else output ("no")
end
```

On the other hand, a lower bound of $n - 1$ comparisons can be demonstrated for ϵ-CLOSENESS if comparisons between meromorphic functions[4] of the inputs are allowed. This result is a special consequence of a general theorem due to Rabin [7].

If tests between meromorphic functions are allowed for the MEASURE problem, it is still not apparent how they can be used to reduce the number of comparisons below $\Omega(n \log n)$. However, the best lower bound we can demonstrate in this model is only $\Omega(n)$.

THEOREM 3: Let $T^*(n)$ denote the minimum complexity of any decision tree algorithm for MEASURE in which all comparisons are between meromorphic functions of the inputs. Then $T^*(n) \geq 2n + O(\log n)$.

PROOF. A decision tree algorithm solving MEASURE must have as many leaf nodes as there are distinct meromorphic functions required to express the measure of the union of n intervals. As a consequence of a basic result of complex variable theory, if a meromorphic function agrees with a linear function in any neighborhood, then the two functions are identical everywhere. Therefore, the number of leaf nodes must be at least as large as the number of distinct linear functions which can express the measure of a union of n intervals. The set of such functions is given by

$$\{\sum_{j \in J} b_j - \sum_{k \in K} a_k \mid J \text{ and } K \text{ are nonempty subsets of } \{1, 2, \ldots, n\} \text{ with } |J| = |K|\}$$

[3] $\lfloor x \rfloor$ is the greatest integer less than or equal to x.

[4] A meromorphic function $f(x_1, x_2, \ldots, x_n)$ is defined here as the quotient of two power series in $x_1, x_2, \ldots, x_n$, each of which is convergent for all values of its arguments, and for which the denominator is not identically zero.

The number of such functions is $\binom{2n}{n} - 1$, and the theorem follows upon evaluating $\log_2(\binom{2n}{n} - 1)$. □

In each of these cases, reducing the number of comparisons has introduced $O(n^2)$ arithmetic operations for computing the nonlinear functions. In general, however, the tradeoffs involved between comparisons and arithmetics are not well understood. It is usually much easier to prove lower bounds on the number of comparisons than on the number of arithmetic operations, especially if the comparisons must be between linear functions of the inputs. The place where this becomes important in our lower bound proofs is, of course, in guaranteeing the convexity of the sets at the leaf nodes of the decision tree.

Acknowledgments. The authors would like to thank Jon Bentley and Michael Shamos for sharing their insights.

Received May 1977; revised November 1977

References

1. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. Bentley, J.L. Divide and conquer algorithms for closest point problems in multidimensional space. Rep. No. TR-76-103, U. of North Carolina, Chapel Hill, N.C., Dec. 1976.
3. Dobkin, D., and Lipton, R. On the complexity of computations under varying sets of primitives. Comp. Sci. Tech. Rep. No. 42, Yale U., New Haven, Conn., 1975.
4. Fredman, M.L. On computing the length of longest increasing subsequences. *Discrete Math. 11* (1975), 29–35.
5. Klee, V. Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *Amer. Math. Monthly 84,* 4 (April 1977), 284–285.
6. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
7. Rabin, M.O. Proving simultaneous positivity of linear forms. *J. Comptr. Syst. Sci. 6,* 6 (Dec. 1972), 639–650.
8. Yuval, G. Finding nearest neighbours. *Inform. Proc. Letters 5,* 3 (Aug. 1976), 63–65.

Programming Techniques

S. L. Graham, R. L. Rivest, Editors

An $O(n)$ Algorithm for Determining a Near-Optimal Computation Order of Matrix Chain Products

Francis Y. Chin
University of Alberta

This paper discusses the computation of matrix chain products of the form $M_1 \times M_2 \times \cdots \times M_n$ where M_i's are matrices. The order in which the matrices are computed affects the number of operations. A sufficient condition about the association of the matrices in the optimal order is presented. An $O(n)$ algorithm to find an order of computation which takes less than 25 percent longer than the optimal time T_{opt} is also presented. In most cases, the algorithm yields the optimal order or an order which takes only a few percent longer than T_{opt} (less than 1 percent on the average).

Key Words and Phrases: approximate algorithm, heuristic algorithm, matrix multiplication, matrix chain product

CR Categories: 5.14

1. Introduction

Consider the evaluation of the product of n matrices

$$M = M_1 \times M_2 \times \cdots \times M_n$$

where M_i is a $k_{i-1} \times k_i$ matrix with each $k_i \geq 1$. Our main tool is the associativity of matrix multiplication. The order in which the matrices are multiplied does not

This research was supported in part by a Canadian National Research Council Grant.

Author's address: Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.