Open access • Journal Article • DOI:10.1145/321921.321922

Bounds on the Complexity of the Longest Common Subsequence Problem

Jeffrey D. Ullman, Alfred V. Aho, Daniel S. Hirschberg

Institutions: Princeton University, Bell Labs, Rice University

Published on: 01 Jan 1976 - Journal of the ACM (ACM)

Topics: Longest increasing subsequence, Longest common subsequence problem, Function problem, String (computer science) and Upper and lower bounds

Bounds on the Complexity of the Longest Common Subsequence Problem

A V. AHO

Bell Laboratories, Murray Hill, New Jersey

D S. HIRSCHBERG AND J D. ULLMAN

Princeton University, Princeton, New Jersey

ABSTRACT The problem of finding a longest common subsequence of two strings is discussed This problem arises in data processing applications such as comparing two files and in genetic applications such as studying molecular evolution The difficulty of computing a longest common subsequence of two strings is examined using the decision tree model of computation, in which vertices represent "equal - unequal" comparisons It is shown that unless a bound on the total number of distinct symbols is assumed, every solution to the problem can consume an amount of time that is proportional to the product of the lengths of the two strings A general lower bound as a function of the ratio of alphabet size to string length is derived The case where comparisons between symbols of the same string are forbidden is also considered and it is shown that this problem is of linear complexity for a two-symbol alphabet and quadratic for an alphabet of three or more symbols

KEY WORDS AND PHRASES longest common subsequence, algorithm, computational complexity, file comparison, molecular evolution

CR CATEGORIES 3 12, 3 73, 5 25

1. *Introduction*

A *subsequence* of a given string is any string obtained by deleting zero or more symbols from the given string. A *longest common subsequence (LCS)* of two strings is a subsequence of both that is as long as any other common subsequence. For example, "cled" and "coed" are the longest common subsequences of "schooled" and "encyclopedia".

Being able to determine a longest common subsequence of two strings is useful in data processing and genetic applications. In data processing a longest common subsequence is often used to measure the differences between two files of data. For example, we can consider a file to be a string in which each line of the file is treated as a single symbol A longest common subsequence of two files identifies those portions of the files that are identical. A genetic application arises in the study of the evolution of long molecules such as proteins; there a longest common subsequence

This research was partially supported by a National Science Foundation Fellowship to D S Hirschberg and by National Science Foundation Grant GJ-35570 to Princeton University

A preliminary version of this paper was presented at the 15th Annual IEEE Symposium on Switching and Automata Theory, October 14-16, 1974.

Authors' present addresses A. V Aho, Bell Laboratories, Inc , 600 Mountain Avenue, Murray Hill, NJ 07974, D S Hirschberg, Department of Electrical Engineering, Rice University, Houston, TX 77001, J D Ullman, Department of Electrical Engineering, Princeton University, Princeton, NJ 08540

is used to measure the correlation between two such molecules [11, 14].

Using dynamic programming an LCS of two strings A and B can be computed in time proportional to the product of their lengths. For special cases an LCS can be computed in time less than the product. For example, if A and B are length n strings of digits $1, 2, \ldots, n$, and no position of A matches more than one position of B, then an LCS of A and B can be computed in $O(n \log\log n)$ time by specializing the algorithms in [6, 9, 18] to integers and using van Emde Boas' integer merging technique [15]. Always being able to compute an LCS of two strings in time significantly less than the product of their lengths, however, appears very difficult [3].

For this reason we believe that an attempt at a lower bound is in order. To derive lower bounds a precise model for a class of algorithms is necessary. The model we choose is that of a decision tree [1] in which all decisions are whether or not two positions have or do not have the same symbol. This model fits various algorithms for the LCS problem which have appeared in the literature [7, 14, 16]. It has also been used to study the related string-to-string correction problem [17], the substring matching problem [5], and various problems on sets [13]. The model does not, however, fit the $O(n^2 \log\log n/\log n)$ algorithm of Paterson [12] nor the special case algorithms of [8, 9].

For the remainder of this paper A and B denote two strings of length n whose LCS we wish to compute [1] Throughout, s denotes the total number of distinct symbols that can appear in A and B (the alphabet size). $T(n, s)$ is the minimum number of comparisons under the decision tree model needed to find an LCS of A and B in the worst case

We shall derive both upper and lower bounds on $T(n, s)$. The use of lower bounds is clear They say that there are no algorithms of lower time complexity which can be modeled by a decision tree with "equal-unequal" comparisons. We are thus told something about the way algorithms for the LCS problem must behave, if they exist at all.

The need for upper bounds on $T(n, s)$ is less obvious. We shall use them to demonstrate that no stronger bounds on $T(n, s)$ can be shown. In principle, an upper bound on $T(n, s)$ is an algorithm for the LCS problem The algorithm, however, may involve essentially different decision trees for each value of n and s. Thus, it is possible that no uniform algorithm taking strings of arbitrary lengths and finding their LCS can be obtained from a sequence of decision trees for all n and s, and such appears to be the case here.

Our principal results are the following:

(1) $T(n, 2) = 2n-1$ for $n \geqslant 1$.

(2) For all $n \geqslant 1$·

(i) $\frac{s}{2}(n+\frac{s}{2}) \leqslant T(n, s) \leqslant \min[n^2, (s-1)(2n-\frac{s}{2})]$, for $2 \leqslant s \leqslant n$.

(ii) $3ns/4 \leqslant T(n, s) \leqslant n^2$, for $n \leqslant s \leqslant 4n/3$.

(iii) $T(n, s) = n^2$, for $s \geqslant 4n/3$.

These upper and lower bounds on $T(n, s)$ are shown in Figure 1.

(3) The special case where all comparisons are between symbols of different strings is shown to require $2n-1$ comparisons if $s = 2$ and n^2 comparisons if $s \geqslant 3$.

[1] We can, in a straightforward manner, generalize the results of this paper to the case where the strings are of unequal length

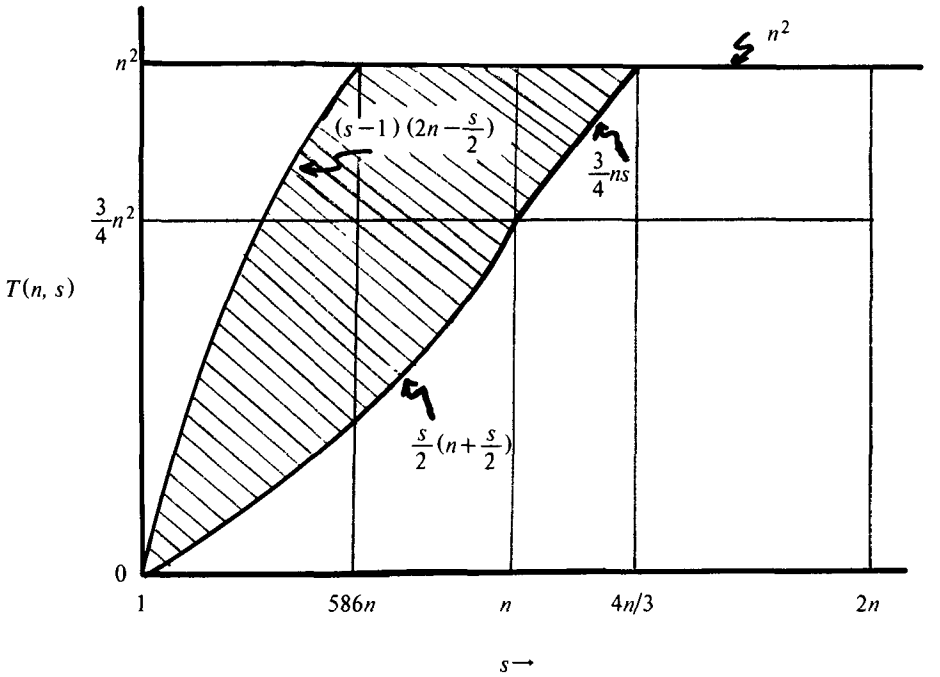

FIG 1. Upper and lower bounds on $T(n, s)$

2. *Decision Trees*

This section makes precise the decision tree model of computation Intuitively, each path starting at the root of a decision tree represents a sequence of comparisons made between various positions in the strings A and B. These comparisons give us all the information we currently know about A and B. The information is in the form of which positions in A and B must contain identical or distinct symbols.

More formally, we define a *decision tree with "equal-unequal" comparisons for the LCS problem* as a rooted binary tree in which each interior vertex is labeled with a pair of integers and each leaf is labeled by two lists of positions from A and B, respectively. A pair of integers $p.q$ at an interior vertex represents a comparison between the symbols in positions p and q of the two strings. (p and q can be positions in the same string.) Each list of positions at a leaf represents an LCS of A and B

Since the only information we get about A and B comes from "equal-unequal" comparisons among symbols of the two strings, we are always dealing with relative values of symbols in various positions in the strings. Consequently, it is convenient to define an (n, s)-*assignment* (or *assignment* when n and s are clear) as a setting of values from some s-symbol alphabet to the positions of A and B. Intuitively, an assignment is a representative of an equivalence class of pairs of input strings. Given a path $P = v_1, v_2, \ldots, v_m$ from the root to some vertex (not necessarily a leaf) in a decision tree, we say (n, s)-assignment C is *valid* for P if for each pair of positions $p_i:q_i$ at vertex v_i, $1 \leqslant i < m$, v_{i+1} is the left son of v_i if the symbols in positions p_i and q_i are equal according to C, and v_{i+1} is the right son of v_i otherwise. Thus an assignment C is valid for a path if C represents a class of pairs of input strings whose symbols are consistent with the outcomes of the comparisons made along the

path.

We say a decision tree D *solves the* (n, s)*-LCS problem* (or just the *LCS problem* if n and s are clear) if for every leaf w of D and for every (n, s)-assignment C valid for the path from the root of D to w, the two lists of positions found at w are an LCS in the first and second strings, respectively.

The *complexity* of a decision tree is the length of a longest path in that tree. We define $T(n, s)$ to be the minimum complexity over all decision trees that solve the (n, s)-LCS problem

A *free* decision tree is one which makes no comparisons whose outcomes are already known (For example, if the symbols at positions p and q and at positions q and r have been compared and found equal, then, by transitivity, the symbols at positions p and r are also known to be equal.) We can, without loss of generality, assume that all decision trees being considered are free. This assumption allows us to consider decision trees in which there are no unnecessary comparisons.

Example 1. To fix the model more closely, let us consider the case where $n = s = 2$. (That is, we are to find an LCS of two strings each of length 2, and each over the same two symbol alphabet.) For convenience we let $A = a_1 a_2$ and $B = b_1 b_2$. In Figure 2 we see a decision tree that solves the (2, 2)-LCS problem. It has complexity 3, which we shall see is the minimum for this problem. Thus $T(2, 2) = 3$. □

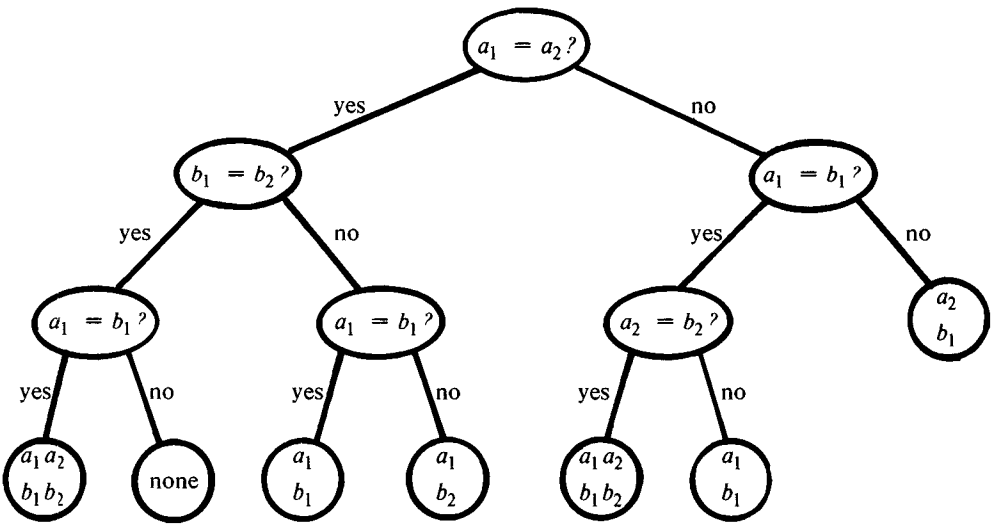

FIG 2. Decision tree solving the (2, 2)-LCS problem

3. *Upper Bounds*

There are two trivial strategies that can be used to construct decision trees for a fixed n and s. The first strategy is to compare each symbol of one string with each symbol of the other. It yields the following theorem.

THEOREM 1. *For all* s *and* n, $T(n, s) \leqslant n^2$.

For $s \geqslant 4n/3$ this result is the best possible under our model of computation. The second strategy is to use comparisons to determine which portions of the two strings hold identical symbols. We cannot, of course, determine the actual symbol at

a position with "equal - unequal" comparisons. If we know the partition of the two strings into equivalent positions, however, then we can surely select an LCS for the string without making any additional "equal-unequal" comparisons. We are thus motivated to make the following definitions.

The *(m, s)-string identification problem* is, given a string of length m, to determine which positions hold the same symbols, assuming all symbols are chosen from an s-symbol alphabet. A *decision tree with "equal-unequal" comparisons for the string identification problem* is defined as for the LCS problem, except the leaves are labeled with partitions of the integers $1, 2, \ldots, m$ into at most s equivalence classes.

The notions of assignment and validity of an assignment are defined as for the LCS problem. A decision tree *solves* the (m, s)-string identification problem if for each leaf w, all valid assignments for the path from the root to w have equal symbols at a pair of positions if and only if those positions are in the same block of the partition at w. Finally, we can define $I(m, s)$ to be the minimum over all decision trees D solving the (m, s)-string identification problem of the length of the longest path in D.

LEMMA 1. $T(n, s) \leqslant I(2n, s)$.

PROOF Concatenate the two strings of length n into one string of length $2n$, identify the equivalent positions, and determine from them an LCS for the two strings of length n. Note that no algorithm to solve the LCS problem for general n and s is implied by this strategy, but using it we can, for fixed n and s, build a decision tree for the (n, s)-LCS problem given a decision tree for the $(2n, s)$-string identification problem. □

LEMMA 2. $I(m, s) \leqslant (s-1)(m-\frac{s}{2})$ *for all* $1 \leqslant s \leqslant m$.

PROOF Visit in turn each position of the given string, comparing the symbol at that position with the representatives for each of the equivalence classes found so far. If the symbol matches the representative of some class, it is added to that class. If no match is found, the symbol becomes the representative of a new class. Hence, for $1 \leqslant i \leqslant s$, at most $i-1$ comparisons are needed for the ith position. For $i > s$, $s-1$ comparisons suffice, since the sth comparison will always succeed if all others have failed. The total number of comparisons is thus

$$\sum_{i=1}^{s-1} i + (m-s)(s-1) = (s-1)(m-\frac{s}{2}). \qquad \square$$

From Lemmas 1 and 2 we conclude:

THEOREM 2. *For all s and n,* $T(n, s) \leqslant (s-1)(2n-\frac{s}{2})$.

Note that Theorem 1 is stronger than Theorem 2 when $\frac{s}{n} \geqslant 2-\sqrt{2} \approx .586$ and Theorem 2 is stronger otherwise.

4. *String Identification*

Since the string identification problem was used in the proof of Theorem 2 to bound from above the complexity of the LCS problem, let us digress a moment and show that the upper bound on $I(m, s)$ of Lemma 2 is its exact value.

To prove this result we relate the string identification problem to graph coloring. Given a path P in a decision tree for the LCS or string identification problem we can associate with P an undirected graph G_P as follows. Let R_P relate two positions if they have been compared and found equal along path P. Let $\equiv_P$ be the least equivalence relation containing R_P. That is, $p \equiv_P q$ if and only if $p = q$ or the fact that p and q have the same symbol is implied by the outcomes along path P. Then the vertices of the graph G_P are the equivalence classes of $\equiv_P$, and there is an

edge between two vertices if members of their represented classes have been compared and found unequal along path P.

An undirected graph is *k-colorable* if there is a mapping (a *k-coloring*) from its vertices to a set of numbers (*colors*) such that no two adjacent vertices are mapped to the same color. A graph G is *uniquely k-colorable* if all k-colorings of G are the same up to a renaming of colors.

Example 2. Figure 3 shows a path P which is part of a hypothetical decision tree for the (6,2)-string identification problem. We use $a_1, a_2, \ldots, a_6$ for the positions of the string. R_P is given by:

$$a_1 \; R_P \; a_2$$
$$a_2 \; R_P \; a_3$$
$$a_4 \; R_P \; a_5$$

$\equiv_P$ has equivalence classes $\{a_1,a_2,a_3\}$, $\{a_4,a_5\}$, and $\{a_6\}$. Since $a_3 \neq a_4$ and $a_5 \neq a_6$, the graph G_P is as shown in Figure 4.

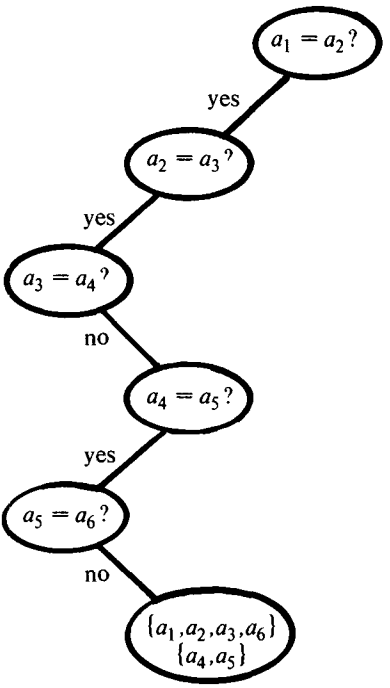

FIG 3. Path P

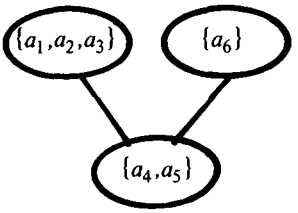

FIG 4. Graph G_P

We note that all 2-colorings for G_P must color $\{a_4,a_5\}$ and the other two vertices with the other color. Thus the 2-coloring of G_P is unique up to renaming of colors and explains one conclusion at the leaf of Figure 3 that a_6 has the same value as a_1, a_2, and a_3 although no equality among them is implied by $\equiv_P$. □

We may easily see that the notions of graph colorings and valid assignments for a path are related. It is therefore a restatement of definitions to prove:

LEMMA 3. *A decision tree solves the (m, s)-string identification problem if and only if for each path P, G_P is uniquely s-colorable.*

The following two lemmas are from [2].

LEMMA 4. *In a k-coloring of a uniquely k-colorable graph, the subgraph induced by the union of any two color classes is connected.*

LEMMA 5. *Every uniquely k-colorable graph with p vertices has at least* $(k-1)(p-\frac{k}{2})$ *edges.*

PROOF Let c_i be the number of vertices of color i in a unique k-coloring of the graph. Then $\sum_{i=1}^{k} c_i = p$. For each $i < j$ there are at least $c_i + c_j - 1$ edges connecting vertices of these two colors by Lemma 4. No edge surely connects two vertices of the same color, so to each edge we may assign a unique pair of states i and j such that the edge is counted among the $c_i + c_j - 1$ edges connecting the vertices of colors i and j. Thus the number of edges is at least $\sum_{i<j} c_i + c_j - 1$. Now $\sum_{i<j} 1 = k(k-1)/2$. For each m the term c_m appears exactly $k-1$ times among all the expressions $c_i + c_j$ for $i < j$. Thus

$$\sum_{i<j} c_i + c_j = (k-1)\sum_{m=1}^{k} c_m = (k-1)p.$$

Hence, $\sum_{i<j} c_i + c_j - 1 = (k-1)p - k(k-1)/2 = (k-1)(p-\frac{k}{2})$. □

THEOREM 3. $I(m, s) = (s-1)(m-\frac{s}{2})$ *for all* $1 \leqslant s \leqslant m$.

PROOF From Lemma 2 we have $I(m, s) \leqslant (s-1)(m-\frac{s}{2})$. Suppose we have a decision tree D for the string identification problem and consider that path P in D for which all outcomes are "not equal." The graph G_P has m vertices and must be uniquely s-colorable by Lemma 3. Thus by Lemma 5 G_P has at least $(s-1)(m-\frac{s}{2})$ edges, so P must be of at least that length. We have thus bounded from below the complexity of an arbitrary decision tree D for the string identification problem. Hence $I(m, s) \geqslant (s-1)(m-\frac{s}{2})$. □

5. *Lower Bounds for the LCS Problem*

We cannot show Theorems 1 and 2 to be exact bounds on $T(n, s)$, principally because we do not have an analog of Lemma 3 relating valid assignments to s-colorings. We can show the lower bound of Figure 2, and do so with a series of lemmas. The general strategy behind the proof is to exhibit a path P_* in any decision tree such that either lots of comparisons between positions of the two strings are made, or a lot of comparisons between positions of the same string are made to group positions into large equivalence classes under $\equiv_{P_*}$.

A *fundamental assignment* is an assignment of values to the positions of strings A and B such that there are at most $s/2$ different values per string and there are no values common to strings A and B. Thus, a fundamental assignment has an LCS of length 0.

A *side comparison* is a comparison between positions of the same string.

A *cross comparison* is a comparison between positions of different strings.

A *valid assignment* (for a particular sequence of comparisons) is an assignment of values to positions that is consistent with the results of all comparisons

We now define an "oracle" or decision rule by which a path is distinguished in each decision tree for the LCS problem.

Decision Rule (*): Return "unequal" whenever there exists a valid fundamental assignment consistent with that outcome. Otherwise, return "equal."

Let us fix on an arbitrary decision tree D for the (n, s)-LCS problem. Let P_* be the path from the root of D to a leaf such that every comparison has the outcome dictated by rule (*). Let $P_*^{(i)}$, $i \geqslant 0$, be the prefix of length i of P_*, and let $C_*^{(i)}$ be a (not necessarily unique) fundamental assignment presumed valid for $P_*^{(i)}$ by rule (*).

Define a *group* of positions (with respect to $P_*^{(i)}$) to be an equivalence class under $\equiv_{P_*^{(i)}}$. Note that a group may have size 1, and that all groups are contained within one string or the other, since by rule (*), all cross comparisons have outcome "unequal." If all members of a group have each been involved in at least $s/2$ side comparisons, call the group a *clan*.

Let us call the two strings being compared A and B. Let g_A and c_A be the number of groups and clans, respectively, in A with respect to path P_*. Since every clan is a group, $g_A \geqslant c_A$. Let g_B and c_B be defined analogously for string B.

It is easy to get a lower bound on the number of cross comparisons in P_*.

LEMMA 6. *P_* makes at least $g_A g_B$ cross comparisons.*

PROOF If not, then there are two groups, G_1 in A and G_2 in B, such that none of their positions have been compared. We know there is a valid fundamental assignment C_* for P_*, in which the LCS is necessarily of length 0. We can find another assignment C valid for P_* by changing the value of the G_1 positions in C_* to be equal to that of the G_2 positions.

To see that C is valid for P_*, consider any comparison $p_1 : p_2$ on that path. If neither p_1 nor p_2 is in G_1, their values are the same in C as in C_*. Thus the values assigned to positions p_1 and p_2 by C agree with the outcome of comparison $p_1 : p_2$ along P_*. If both p_1 and p_2 are in G_1, then they must have the same value in C_*, so the outcome of $p_1 : p_2$ was "equal". Since p_1 and p_2 were defined to have the same value in C, the outcome "equal" for comparison $p_1 . p_2$ is consistent with C.

If only one of p_1 and p_2, say p_1, is in G_1, then the outcome of $p_1 : p_2$ must be "unequal," and p_1 and p_2 have different values in C_*. They must have different values in C as well, unless p_2 is in G_2. But by hypothesis, no member of G_1 was compared with a member of G_2, so we can rule out this possibility. We conclude that C is valid for P_*.

Since C has an LCS of length equal to the smaller of G_1 and G_2, which is not zero, we conclude that the decision tree D of which P_* was a path does not solve the LCS problem. □

We now develop a lower bound on side comparisons by showing that in order for there to be any "equal" side comparisons, there must be many side comparisons with outcome "unequal." If there are few "equal" side comparisons, then Lemma 6 is sufficient to show P_* to be long. If there are many "equal" side comparisons, then we can use the number of "unequal" side comparisons to bound from below the length of P_*.

LEMMA 7. *Every group of size greater than one is a clan (i.e. all its members are involved in at least $s/2$ side comparisons).*

PROOF Suppose not. Then there must be some side comparison $p_1 : p_2$ on P_* with outcome "equal" along P_*, such that p_1 was previously in a group by itself and had been involved in at most $\frac{s}{2} - 2$ side comparisons, all of them with outcome "unequal." Let comparison $p_1 : p_2$ be the ith in P_* and consider a fundamental assignment $C_*^{(i-1)}$. Since p_1 has been compared with at most $\frac{s}{2} - 2$ other positions in its string, there is some value reserved for positions in that string possessed in $C_*^{(i-1)}$ by none of the positions with which p_1 has been compared in $P_*^{(i)}$.

Therefore, we can construct a valid fundamental assignment C from $C_*^{(i-1)}$ by giving p_1 that value. By an argument similar to that of Lemma 6, we can argue that C is valid for the path consisting of $P_*^{(i-1)}$ followed by the "unequal" branch after comparison $p_1:p_2$. Thus P_* should not follow the "equal" branch at that comparison, as supposed. □

LEMMA 8. *If there are any "equal" side comparisons in string A along P_*, then there are at least $s/2$ clans in string A, and similarly for B.*

PROOF Suppose the ith comparison along P_*, say $p_1:p_2$, has the outcome "equal." Suppose in contradiction that after this comparison there are at most $\frac{s}{2}-1$ clans in A. But then we can construct a fundamental assignment C valid for $P_*^{(i-1)}$ and the "unequal" outcome for $p_1:p_2$ as follows. C assigns to members of string B the same values assigned to them under $C_*^{(i-1)}$. After the $p_1:p_2$ comparison, p_1 and p_2 are together in a clan according to Lemma 7. Consider the other $\frac{s}{2}-2$ clans in A after this comparison. To each of these clans we assign a distinct value. This leaves free at least two distinct values which we can assign to p_1 and p_2 before the comparison. Nonclans are assigned values distinct from that of any position with which they have been compared. □

LEMMA 9. *If $g_A < n$, then $c_A \geqslant s/2$, and similarly for B.*

PROOF If $g_A < n$, then there has been a side comparison involving positions of A with outcome "equal." The result then follows from Lemma 8. □

LEMMA 10. *If $g_A < n$, then there have been at least $(n - g_A)\frac{s}{4} + \frac{s^2}{8}$ side comparisons along P_*, and similarly for B.*

PROOF Let h_A be the number of positions in A that are not in clans. By Lemma 7, all such positions are in single groups so $h_A = g_A - c_A$. Then the number of positions in clans is $n - h_A = n - g_A + c_A$. Each position in a clan is involved in at least $s/2$ side comparisons and each side comparison involves at most two members of clans. Thus the number of side comparisons within A in P_* is at least $(\frac{1}{2})(\frac{s}{2})(n - g_A + c_A) = (n - g_A)\frac{s}{4} + \frac{sc_A}{4}$. By Lemma 9, $c_A \geqslant s/2$, so the present result follows. □

THEOREM 4.

$$T(n, s) \geqslant \frac{s}{2}(n + \frac{s}{2}), \quad \text{for } s \leqslant n,$$

$$T(n, s) \geqslant 3ns/4, \quad \text{for } n \leqslant s \leqslant 4n/3, \text{ and}$$

$$T(n, s) \geqslant n^2, \quad \text{for } 4n/3 \leqslant s \leqslant 2n.$$

PROOF We consider the length of P_*, obtaining lower bounds in the cases when zero, one, or two of g_A and g_B are less than n. We then claim that $T(n, s)$ must exceed the smallest of these three. The three inequalities in the statement of the theorem reflect the analysis regarding which one of these cases yields the smallest lower bound on the length of P_* for varying values of s.

Case 0. $g_A = g_B = n$. The length of P_* is at least n^2 by Lemma 6.

Case 1. Let $g_A < n$ and $g_B = n$. Then there are at least ng_A cross comparisons by Lemma 6 and at least $(n - g_A)\frac{s}{4} + \frac{s^2}{8}$ side comparisons by Lemma 10. The length of P_* is at least $(n - g_A)\frac{s}{4} + \frac{s^2}{8} + ng_A$. Since $g_A \geqslant c_A$ is obvious, we have $g_A \geqslant s/2$ by Lemma 9. Let $g_A = s/2 + t$. Then $(n - g_A)\frac{s}{4} + \frac{s^2}{8} + ng_A = 3ns/4 + t(n - \frac{s}{4})$. Since we assume $s \leqslant 2n$, $3ns/4$ is a lower bound on the

length of P_* in this case.

Case 2. $g_A < n$ and $g_B < n$. By Lemmas 6 and 10, the length of P_* is at least

$$(2n - g_A - g_B)\frac{s}{4} + \frac{s^2}{4} + g_A g_B.$$

Since we know g_A and g_B are at least $s/2$, we have

$$g_A g_B = \frac{1}{2}g_A g_B + \frac{1}{2}g_A g_B \geqslant \frac{1}{2}g_A (s/2) + \frac{1}{2}g_B (s/2) = \frac{1}{4}s(g_A + g_B).$$

Thus $(2n - g_A - g_B)\frac{s}{4} + \frac{s^2}{4} + g_A g_B \geqslant \frac{ns}{2} + \frac{s^2}{4} = \frac{s}{2}(n + \frac{s}{2})$. We see that $\frac{s}{2}(n + \frac{s}{2})$ is a lower bound on the length of P_* in this case.

We may conclude that P_* is at least as long as the smallest of n^2, $3ns/4$ and $\frac{s}{2}(n + \frac{s}{2})$ The theorem then follows by comparing these three functions as s ranges from 1 to $2n$. □

It should be noted that the lower bound of Theorem 4 applies even if we only wish to find the length of an LCS.

If we compare two strings of lengths n_1 and n_2, respectively, where $n_1 > n_2$, then we can obtain the following lower bounds on $T(n_1, n_2, s)$, the number of comparisons needed to determine a longest common subsequence (or even its length):

$$T(n_1, n_2, s) \geqslant \frac{s}{4}(n_1 + n_2 + s) \text{ for } 0 \leqslant s \leqslant n_2,$$

$$T(n_1, n_2, s) \geqslant \frac{s}{4}(2n_2 + n_1) \text{ for } n_2 \leqslant s \leqslant \frac{4n_1 n_2}{2n_2 + n_1}, \text{ and}$$

$$T(n_1, n_2, s) \geqslant n_1 n_2 \text{ for } \frac{4n_1 n_2}{2n_2 + n_1} \leqslant s \leqslant n_1 + n_2.$$

For the special case $s = 2$, we can obtain exact upper and lower bounds on the LCS problem

THEOREM 5. $T(n, 2) = 2n - 1$.

PROOF By Theorem 2 we need only show that $T(n, 2) \geqslant 2n - 1$. Given any decision tree for the $(n,2)$-LCS problem, consider the path P along which all cross comparisons have outcome "unequal" and all side comparisons have outcome "equal." Suppose G_P has m vertices. Then there are exactly $2n - m$ side comparisons in P. We can find one assignment C_1 valid for P, in which all positions in one string are given one value, say 0, and all positions in the other have the other value, say 1. The LCS in C_1 is clearly of length 0. If G_P is not connected, then we could reverse the values defined by C_1 in one connected component to obtain valid assignment C_2 with LCS of length greater than 0. Thus G_P is connected, so there are at least $m - 1$ cross comparisons in P, for a total of $2n - m + (m - 1) = 2n - 1$ comparisons. □

6 *Algorithms That Use Only Cross Comparisons*

Various algorithms such as those of [7, 15] rely on cross comparisons only. It turns out that it is easy to get exact bounds on algorithms of this type, and the bounds are significantly higher for many values of s than for the unrestricted case.

THEOREM 6. *For $s = 2$, $2n-1$ cross comparisons and no side comparisons are sufficient to solve the LCS problem, and they are necessary even to determine the length of the LCS if side comparisons are forbidden.*

PROOF Necessity follows from Theorem 5. For sufficiency, we use an algorithm similar to that of Lemma 2. Compare the first position of each string with all positions of the other string, a total of $2n-1$ comparisons. Then the positions p_1 and p_2 in the same string have the same value if and only if they had the same outcome when compared with the first symbol of the other string. (Note that $s = 2$ is essential here.)

If p_1 and p_2 are in different strings, let a_1 and b_1 be the first positions of the strings containing p_1 and p_2, respectively. Then positions p_1 and p_2 hold the same value if and only if an odd number of the outcomes of the following three comparisons are "equal": (1) $p_1 : b_1$, (2) $p_2 : a_1$, (3) $a_1 : b_1$. (Again $s = 2$ is essential) □

THEOREM 7. *For $s \geq 3$, n^2 cross comparisons are sufficient for the LCS problem and necessary even to find the length of an LCS if side comparisons are prohibited.*

PROOF Sufficiency is obvious Consider the path P in a decision tree D, which makes cross comparisons only, such that all outcomes are "unequal." If positions p_1 in string A and p_2 in string B are not compared along P, then we may find two assignments C_1 and C_2 valid for P as follows. C_1 maps all positions in A to 0 and all positions in B to 1. It clearly has an empty LCS. C_2 maps all positions in A except p_1 to 0 and all positions in B except p_2 to 1. p_1 and p_2 are given value 2. Clearly the LCS of C_2 has length 1, and D does not solve the LCS problem. We conclude that all n^2 possible cross comparisons are present in P, so the complexity of D is at least n^2. □

7. *Conclusions*

We have demonstrated that any algorithm using "equal-unequal" comparisons for the LCS problem must, in the worst case, either be quadratic or must assume a fixed alphabet size. Even for a fixed alphabet of size greater than two, side comparisons must play an essential part if the algorithm is to run in less than quadratic time. There are, of course, many opportunities to use techniques that cannot be modeled by our decision trees. Indexing into arrays, as in [12], is one; sorting and hashing are other ideas which might be useful in constructing less than quadratic algorithms.

An obvious next step is to investigate the expected time complexity of the LCS problem. The primary difficulty here is defining a meaningful probability distribution on pairs of input strings. Chvatal and Sankoff [4] have computed bounds on the expected length of a longest common subsequence of two random sequences of the same length. However, in some applications random sequences may not be encountered. For example, in data processing applications where two files are being compared it is not reasonable to treat the two files as random pairs of strings.

The straightforward dynamic programming algorithm for the LCS problem has the disadvantage that it takes the same quadratic amount of time on all inputs. In practice we would prefer an algorithm which is more efficient on typical inputs and which may be less efficient on infrequent inputs. For example, Hunt and Szymanski's algorithm [9], which is based on an approach suggested by H. S. Stone, has the desirable property that it can be easily implemented to work in $O(n \log n)$ time on many inputs which occur in data processing applications, although its worst case time complexity is $O(n^2 \log n)$. More investigation of algorithms of this nature seems profitable.

ACKNOWLEDGMENTS The authors would like to thank Shen Lin and Doug McIlroy for their helpful comments. This paper was typeset using EQN on UNIX [10]. The cheerful assistance of Mike Lesk in the final preparation of this paper was sincerely appreciated.

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 CARTWRIGHT, D, AND HARARY, F On colorings of signed graphs *Elemente der Mathematik 23* (1968), 85-89
3 CHVATAL, V, KLARNER, D A, AND KNUTH, D E Selected combinatorial research problems STAN-CS-72-292, Stanford U, Stanford, Calif, 1972, p 26
4 CHVATAL, V, AND SANKOFF, D Longest common subsequences of two random sequences STAN-CS-75-477, Stanford U, Stanford, Calif, Jan 1975
5 FISCHER, M J, AND PATERSON, M S String matching and other products MAC Technical Memorandum 41, M I T, Cambridge, Mass, 1974
6 FREDMAN, M L On computing the length of longest increasing subsequences *Discrete Mathematics 11,* 1 (January 1975) 29-36
7 HIRSCHBERG, D S A linear space algorithm for computing maximal common subsequences *Comm ACM 18,* 6 (June 1975) 341-343
8 HIRSCHBERG, D S On finding maximal common subsequences TR-156, Computer Sciences Laboratory, Princeton U, Princeton, N J, 1974
9 HUNT, J W, AND SZYMANSKI, T G A fast algorithm for computing longest common subsequences Unpublished memorandum, Princeton University, September 1975
10 KERNIGHAN, B W, AND CHERRY, L L A system for typesetting mathematics *Comm ACM 18, 3* (March 1975), 151-157
11 NEEDLEMAN, S B, AND WUNSCH, C D A general method applicable to the search for similarities in the amino acid sequence of two proteins *J Mol Biol 48* (1970), 443-453
12 PATERSON, M S Unpublished manuscript University of Warwick, England, 1974
13 REINGOLD, E M On the optimality of some set algorithms *J ACM* **19**, 4 (October 1972), 649-659
14 SANKOFF, D Matching sequences under deletion/insertion constraints *Proc. Nat. Acad Sci. USA 69, 1* (Jan, 1972), 4-6
15 VAN EMDE BOAS, P Preserving order in a forest in less than logarithmic time *Sixteenth Annual IEEE Symposium on Foundations of Computer Science,* October 1975
16 WAGNER, R A, AND FISCHER, M J The string-to-string correction problem *J ACM 21,* 1 (Jan, 1974), 168-173
17 WONG, C K, AND CHANDRA, A K Bounds for the string editing problem IBM Research Center, Yorktown Heights, New York, 1974
18 YAO, A C, AND YAO, F F On computing the rank function for a set of vectors UIUCDCS-R-75-699, Dept of Computer Science, U of Illinois, Urbana, Ill, Feb, 1975

RECEIVED DECEMBER 1974, REVISED MAY 1975