BIT 21 (1981), 20–30

THE STRING MERGING PROBLEM*

STEPHEN Y. ITOGA

Abstract.

The string merging problem is to determine a merged string from a given set of strings. The distinguishing property of a solution is that the total cost of editing all of the given strings into this solution is minimal. Necessary and sufficient conditions are presented for the case where this solution matches the solution to the string-to-string correction problem. A special case where deletion is the only allowed edition operation is shown to have the longest common subsequence of the strings as its solution.

Key Words and Phrases: String modification, string matching, editing, costs of editing, longest common subsequence, shortest common supersequence.

1. Introduction.

Wagner and Fischer [11] posed the fundamental string-to-string correction problem (STS-problem) that has received much interest [1, 6, 7, 12]. Recently there has been work done to improve the time and space complexity requirements of their original algorithm [4, 5, 8]. Here we generalize their original problem to address the case where more than two strings are considered and where none of the strings are given any special designation. This approach incorporates some of the ideas found in [7] and [10]. In section 2 the problem is described in its most general form. Necessary and sufficient conditions for this problem to match the STS-problem are described in section 3. The special case where deletion is the only allowed editing operation is shown to solve the longest common subsequence problem (LCS-problem) in section 4. An algorithm to solve the problem in this situation is then given along with a complexity analysis of the algorithm using results from [4, 7, 8]. The special case where insertion is the only allowed editing operation is shown to solve the shortest common supersequence problem (SCS-problem) in section 5. The complexity of an algorithm to solve this problem is then discussed. A summary of our results and some conclusions are presented in section 6.

2. String merging

For consistency with [11] we will use the following notation. We let Σ denote an arbitrary finite set of symbols, Σ^* the monoid freely generated from Σ under

*This research was supported by the U.S. Army Research Office.
Received Feb. 21, 1980. Revised Jan. 22, 1981.

concatenation, and $P(\Sigma^*)$ the set of all subsets of Σ^*. If X is a string (finite sequence) of symbols then $X\langle i\rangle$ is the ith symbol of X, $X\langle i{:}j\rangle$ is the substring $X\langle i\rangle\, X\langle i+1\rangle \ldots X\langle j\rangle$, and $|X|$ is the length (number of symbols) of X, i.e., $X = X\langle 1{:}|X|\rangle$.

For a fixed Σ, an *edit operation* s is any mapping from Σ^* to $P(\Sigma^*)$. We will use S to denote an arbitrary finite set of edit operations. Then for $A, B \in \Sigma^*$ and $s \in S$, B is a result of editing A under s if $B \in s(A)$. An *edit sequence* s is any finite sequence of edit operations. The action of $s = s_1, s_2, \ldots, s_m$ on string X is defined to be $s(X) = s_m(s_{m-1}(\ldots s_1(X))\ldots)$. We let ψ represent the null sequence and define $\psi(X) = X$. For a sequence of strings $X_1, X_2, \ldots, X_m$, a *merge sequence of operations* is any sequence of edit sequences $\mu = s_1, s_2, \ldots, s_m$ such that $\bigcap_{i=1}^{m} s_i(X_i) \neq \emptyset$ (is not empty). I.e., a merge sequence of operations can edit a sequence of strings into a common string. For a fixed Σ, a set S of operations is said to be *complete* if for any finite sequence of strings $X_1, X_2, \ldots, X_m$ from Σ^* there always exists a merge sequence from S. The concept of an edit operation and an edit sequence is extended in the natural way for arguments which are sets.

For example, if k, l, r, and s are positive integers such that $\Sigma = \{a_1, a_2, \ldots, a_k\}$, $w = a_{i_1}a_{i_2}\ldots a_{i_l}$ and $s \leqq k$, then we can define some of the standard edit operations such as

1) insertion: $\operatorname{ins}_{r,s}(w) = \begin{cases} a_{i_1}a_{i_2}\ldots a_{i_{r-1}}a_s a_{i_r}a_{i_{r+1}}\ldots a_{i_l} & \text{if } r \leqq l \\ w \text{ otherwise} \end{cases}$

2) deletion: $\operatorname{del}_r(w) = \begin{cases} a_{i_1}a_{i_2}\ldots a_{i_{r-1}}a_{i_{r+1}}a_{i_{r+2}}\ldots a_{i_l} & \text{if } r \leqq l \\ w \text{ otherwise} \end{cases}$

and

3) change: $\operatorname{cha}_{r,s}(w) = \begin{cases} a_{i_1}a_{i_2}\ldots a_{i_{r-1}}a_s a_{i_{r+1}}a_{i_{r+2}}\ldots a_{i_l} & \text{if } r \leqq l \\ w \text{ otherwise} \end{cases}$

As in [11] we let γ be an arbitrary cost function such that $\gamma(s)$ is a nonnegative real number for all $s \in S$. Then γ is extended to edit sequences $s = s_1, s_2, \ldots, s_m$ via $\gamma(s) = \sum_{i=1}^{m} \gamma(s_i)$. For consistency we set $\gamma(\psi) = 0$. In addition γ is extended to merge sequences $\mu = s_1, s_2, \ldots, s_m$ via $\gamma(\mu) = \sum_{i=1}^{m} \gamma(s_i)$. We now define the *merge distance* for a finite sequence of strings $X_1, X_2, \ldots, X_m$ to be the minimal cost of editing all the strings into one common string. Formally, $S(X_1, X_2, \ldots, X_m) = \min\{\gamma(\mu) \mid \mu \text{ is a merge sequence for } X_1, X_2, \ldots, X_m\}$. Since the merge distance for a sequence of strings is independent of the particular arrangement of the strings, we can then use the terms *merge set of operations* and *merge distance* for a set of strings without any ambiguity.

This notation enables us to state the *string merging problem* (SM-problem). Given a finite set of strings $\{X_1, X_2, \ldots, X_m\}$, a complete set of edit operations S, and an associated cost function γ, the problem is to determine the merge distance $D(X_1, X_2, \ldots, X_m)$ and any particular member of $\bigcap_{i=1}^{m} s_i(X_i)$ where $D(X_1, X_2, \ldots, X_m) = \sum_{i=1}^{m} \gamma(s_i)$. In the following sections we will try to justify this rather abstract version of the elegant problem originally posed in [11].

3. Comparison with the STS-problem.

The general requirements for edit operations include the fundamental operations of insertion, deletion, and change [11] along with more esoteric operations like interchange [6]. The string to string correction problem (STS problem) was defined by Wagner and Fischer [11] to be the problem of finding a minimum cost sequence of edit operations to convert one string into another. We note in passing that a minimum complete set of operations for it consists of just insertion and deletion. In contrast, for the SM-problem just insertion alone or just deletion alone is sufficient to form a complete set. For the rest of this section we will assume that each edit operation $s \in S$ has an associated inverse operation in S, denoted by INV(s), such that for all $X \in \Sigma^*$, we have $X \in \text{INV}(s)\ (s(X))$. The generalization to edit sequences is obvious. We also assume that each operation is necessary in the sense that it constitutes a sequence in the merge sequence for some instance of the SM-problem. Again, the operations mentioned above all enjoy this property.

For any two strings X_1 and X_2, let s_1, s_2 represent a solution to the SM-problem and s_{STS} a solution to the STS-problem. I.e., $s_1(X_1) \cap s_2(X_2) \neq \emptyset$ and $X_2 \in s_{STS}(X_1)$. Since the edit sequence s_1, INV(s_2) edits X_1 into X_2 and since the sequence s_{STS}, ψ is a valid merge sequence we have the following relationships.

LEMMA 1.

a) *Suppose s_{STS}, s_1, and s_2 are as stated above. Then $\gamma(s_2) \leqq \gamma(s_{STS}) - \gamma(s_1) \leqq \gamma(INV(s_2))$.*

b) *Let $X = s_1(X_1) = s_2(X_2)$. Then s_1, ψ is a solution to the SM-problem for $\{X_1, X\}$. By symmetry s_2, ψ is a solution for $\{X_2, X\}$.*

PROOF.

Lemma 1. a) follows from the discussion preceding the lemma. To prove lemma 1. b) suppose there are edit sequences s_1', s_2' such that $s_1'(X_1) = s_2'(X)$ and $\gamma(s_1') + \gamma(s_2') < \gamma(s_1)$. Then the sequences s_1' and s_2, s_2' would constitute a merge sequence for X_1, X_2 with a lower cost than s_1, s_2. ∎

With this information we can now state a condition for the equivalence of our string merging problem and the string-to-string correction problem. By this we mean that from any solution to the SM-problem we can construct a solution to the STS-problem with identical cost and vice-versa.

THEOREM 2.

A necessary and sufficient condition for the equivalence of the STS-problem and the SM-problem for pairs of strings is that $\gamma(INV(s)) = \gamma(s)$ for all operations $s \in S$.

PROOF

Let s_{STS}, s_1 and s_2 be edit sequences as described previously. That the condition is sufficient follows from the fact that $\gamma(INV(s))=\gamma(s)$ for any edit sequence s. Hence, lemma 1.a) implies that $\gamma(s_{STS})=\gamma(s_1)+\gamma(s_2)$. Conversely, if s_1, s_2 is a solution to the SM-problem and we construct $s_1, INV(s_2)$ as a solution to the STS-problem with identical cost, then necessarily $\gamma(s_2)=\gamma(INV(s_2))$. Our assumptions on the members of S then indicate that the condition is necessary. ∎

The following example illustrates several of the points made in this section.

EXAMPLE 3.

Let S be the set of insertion and deletion operations. Let c_i be the cost of any insertion operation and c_d the cost of any deletion operation.

Case 1 If $c_i=c_d$ then $\gamma(s)=\gamma(INV(s))$ for all edit operations so the SM-problem and STS problem are equivalent.

Case 2 If $c_i<c_d$ then take $X_1=ab$ and $X_2=bc$. The merge solution has cost $2c_i$ which is less than the cost for the STS-problem. Note that the merge solution (abc) gives the shortest common supersequence of X_1 and X_2 as its solution.

Case 3 If $c_i>c_d$ then the same example from case 2 has a merge solution cost $2c_d$ which is less than the cost for the STS-problem. In this case, the merge solution (b) gives the longest common subsequence as its solution.

In the remainder of this paper we will discuss two special cases of the SM-problem, namely the longest common subsequence (LCS) problem [1, 4, 5, 7, 8, 9, 12] and the shortest common supersequence (SCS) problem [7]. Here, given a set of strings, the LCS problem is to find a longest string that is a substring of every member of the set. Conversely, the SCS problem is to find a shortest string such that every member of the set is a substring of it.

4. The LCS-problem

In this section we will present a concise algorithm for solving the LCS-problem given any finite set of strings $X_1, X_2, \ldots, X_m$. The key notion is that a solution to the SM-problem when S only consists of single symbol delete operations is also a solution to the LCS-problem. This is apparent when one realizes that the result of any merge sequence of delete operations must be a common subsequence of the original set of strings. Hence, for the remainder of this section, we will assume that $S=\{del_r \mid r\geq 1\}$ and that the cost of applying any operation is some positive fixed constant c_d. To avoid a trivial situation we will also assume that none of the strings is the empty string.

For the sake of clarity, we will develop the algorithm first for the case where there are only two strings to be merged and then later generalize the result to the case where there are an arbitrary number of strings to be merged. For arbitrary

strings X and Y and integers i, j such that $0 \leqq i \leqq |X|$ and $0 \leqq j \leqq |Y|$ let $C(i,j) = \gamma(X\langle 1:i\rangle, Y\langle 1:j\rangle)$, i.e., $C(i,j)$ is the cost of merging substrings of length i and j. If i or j is zero, this corresponds to the cost of merging everything to the empty string. Hence the cost of the merge solution is $D(X,Y) = C(|X|, |Y|)$. In order to establish the correctness of the algorithm we will need the following results.

LEMMA 3.

Let i and j be integers such that $1 \leqq i \leqq |X|$ and $1 \leqq j \leqq |Y|$. Then $C(i,j)$ must equal one of the following expressions:

i) $C(i-1,j-1)$,
ii) $C(i-1,j)+c_d$, *or*
iii) $C(i,j-1)+c_d$.

PROOF

Consider any merge sequence for $X\langle 1:i\rangle$ and $Y\langle 1:j\rangle$ with cost $C(i,j)$. There are only three possible results for the editing operations on the symbols $X\langle i\rangle$ and $Y\langle j\rangle$. Either neither is deleted (and hence $X\langle i\rangle = Y\langle j\rangle$) so that i) holds; or $X\langle i\rangle$ is deleted so that ii) holds; or otherwise $Y\langle j\rangle$ is deleted so that iii) holds. ∎

THEOREM 4.

Let i and j be integers such that $1 \leqq i \leqq |X|$ and $1 \leqq j \leqq |Y|$. If $X\langle i\rangle = Y\langle j\rangle$ then $C(i,j) = C(i-1,j-1)$.

PROOF

Suppose that $X\langle i\rangle = Y\langle j\rangle$ and $C(i,j)$ does not equal $C(i-1,j-1)$, i.e., case i) of lemma 3 does not hold. By symmetry it would suffice to show that ii) does not hold for then lemma 3 would be contradicted. Assume $C(i,j) = C(i-1,j)+c_d$. It follows that we can find a minimum cost merge sequence s_x, s_y for $X\langle 1:i\rangle$ and $Y\langle 1:j\rangle$ such that the edit sequence s_x has a delete operation for $X\langle i\rangle$. If s_y has a delete operation for $Y\langle j\rangle$ then by eliminating both of these delete operations (one from s_x and one from s_y) we would have a merge sequence that had a cost $C(i-1,j)-c_d < C(i,j)$. Since this is a contradiction to our assumptions, the only alternative is that $Y\langle j\rangle$ appears in the final solution of the merge sequence. Hence it must be matched by some symbol $X\langle k\rangle$ where $1 \leqq k \leqq i-1$. This implies that s_x contains delete operations for all symbols $X\langle l\rangle$ where $k < l \leqq i-1$. Hence we could create a new edit sequence s_x' from s_x by dropping the delete operation for $X\langle i\rangle$ and replacing it with a delete operation for $X\langle k\rangle$. This new minimum cost merge sequence would then match $X\langle i\rangle$ with $Y\langle j\rangle$ implying that $C(i,j) = C(i-1,j-1)$. ∎

Lemma 3 and theorem 4 justify the correctness of the following algorithm for computing the cost array C for strings X and Y.

Algorithm 5.

5.1 $C(0,0):=0$
5.2 **for** $i:=1$ **to** $|X|$ $C(i,0):=ic_d$
end for i
5.3 **for** $j:=1$ **to** $|Y|$ $C(0,j):=jc_d$
end for j
5.4 **for** $i:=1$ **to** $|X|$
for $j:=1$ **to** $|Y|$
if $(X\langle i\rangle = Y\langle j\rangle)$
then $C(i,j):=C(i-1,j-1)$
else $C(i,j):=\min\{C(i-1,j),\ C(i,j-1)\}+c_d$
end for j
end for i

In order to generalize our results we define a function α such that for any integer i and j and any finite set of integers K,

$$\alpha(i,j,K) = \begin{cases} j & \text{if } i \in K \\ j-1 & \text{if } i \notin K\ . \end{cases}$$

For notational convenience we will let $\alpha(i_1, i_2, \ldots, i_m, K)$ represent the sequence $\alpha(1,i_1,K),\ \alpha(2,i_2,K),\ldots,\alpha(m,i_m,K)$.

Algorithm 6.

Let $X_1, X_2, \ldots, X_m$ be non-empty strings. Then the cost array for this situation (and hence the general solution to the LCS-problem) can be computed as follows.

6.1 **for** $(i_1:=0$ **to** $|X_1|,\ i_2:=0$ **to** $|X_2|,\ldots,i_m:=0$ **to** $|X_m|)$
if $(i_j=0$ for some $1\leqq j\leqq m)$
then $C(i_1,i_2,\ldots,i_m):=c_d\sum_{k=1}^{m} i_k$
end for
6.2. **for** $(i_1:=1$ **to** $|X_1|,\ i_2:=1$ **to** $|X_2|,\ldots,i_m:=1$ **to** $|X_m|)$
if $(X_1\langle i_1\rangle = X_2\langle i_2\rangle = \ldots = X_m\langle i_m\rangle)$
then $C(i_1,i_2,\ldots,i_m):=C(i_1-1,\ i_2-1,\ldots,i_m-1)$
else $C(i_1,i_2,\ldots,i_m):=$
$\min_{a\in\Sigma}\{C(\alpha(i_1,i_2,\ldots,i_m,K))+(m-|K|)c_d | K=\{|X_k\langle i_k\rangle = a\}\neq\emptyset\}$.
end for

In step 6.1, the initialization of C takes advantage of the fact that if one string is the empty string then all strings must be deleted completely. In step 6.2, the set K (a proper subset of $\{1,2,\ldots m\}$) represents the symbols in $\{X_1\langle i_1\rangle, X_2\langle i_2\rangle,\ldots,X_m\langle i_m\rangle\}$ that can be matched together. Hence $(m-|K|)$ deletes are needed to make the other sequences match.

As an added bonus, the particular formulation of this problem enables us to

compute the length of the longest common subsequence of $\{X_1, X_2, \ldots, X_m\}$ by setting $c_d = 1$ and evaluating the expression

$$\left(\sum_{i=1}^{m} |X_i| - C(|X_1|, |X_2|, \ldots, |X_m|) \right) \Big/ m .$$

Another interesting statistical quantity would be the term

$$\varrho_{\mathrm{LCS}} = 1 - C(|X_1|, |X_2|, \ldots, |X_m|) \Big/ \left(\sum_{i=1}^{m} |X_i| \right)$$

since it serves as an indication of the amount of similarity that exists among the strings $\{X_1, X_2, \ldots, X_m\}$. Here, ϱ_{LCS} is the ratio of the length of the longest common subsequence to the average length of the original strings. Since $0 \leqq C(|X_1|, |X_2|, \ldots, |X_m|) \leqq \sum_{i=1}^{m} |X_i|$ then $0 \leqq \varrho_{\mathrm{LCS}} \leqq 1$. In both cases, note that the expressions can be evaluated without actually determining any member of the set of longest common subsequences.

In order to illustrate how we can determine a representative longest common subsequence for $\{X_1, X_2, \ldots, X_m\}$ we introduce two additional *m-dimensional arrays* L and P. The L array will keep track of the manner in which elements of the C array were determined from other elements of the C array, i.e., it records the "linkage" between various elements of the C array. The P array is used to determine if a particular element of the C array can correspond to a member of a longest common subsequence. We will use the expression $[i_1, i_2, \ldots, i_m]$ to represent any encoding of the sequence $i_1, i_2, \ldots, i_m$. Then step 6.2 can be modified to also compute

if $(X_1\langle i_1 \rangle = X_2\langle i_2 \rangle = \ldots = X_m\langle i_m \rangle)$
  then $L(i_1, i_2, \ldots, i_m) := [i_1 - 1, i_2 - 1, \ldots, i_m - 1]$
    $P(i_1, i_2, \ldots, i_m) := 1$
  else $L(i_1, i_2, \ldots, i_m) := [\alpha(i_1, i_2, \ldots, i_m, K)]$
    $P(i_1, i_2, \ldots, i_m) := 0$

Here it has been assumed that some algorithm has been given to select a particular $K \subseteqq \{1, 2, \ldots, m\}$ that satisfies the minimality requirement of step 6.2. The expression for calculating the elements of L clearly illustrates the "linkage" property previously mentioned. The expression for elements of the P array indicates that a value of "1" corresponds to the case where a member of the common subsequence has been found when one is using the L array to "backtrack" over the computations for the C array.

With this information it is evident that the following algorithm will print out a longest common subsequence in reverse order.

Algorithm 7.

7.1 **for** $j=1$ **to** m
$i_j=|X_j|$
end for j
7.2 **while** $(\{k|1\leqq k\leqq m,\ i_k=0\}=\emptyset)$
if $(P(i_1,i_2,\ldots,i_m)=1)$
then print $X_1\langle i_1\rangle(=X_2\langle i_2\rangle=\ldots=X_m\langle i_m\rangle)$.
Update $i_1,i_2,\ldots,i_m$ to be the sequence corresponding to $L(i_1,i_2,\ldots,i_m)$.
end while

Hirschberg [4] showed that the LCS-problem for sets of size two can be solved within linear space constraints. Maier [7] showed that the LCS-problem is NP-complete for $|\Sigma|\geqq 2$. As in [11], the following analysis of algorithm 6 takes the number of assignment statements executed as an indication of the complexity of the algorithm. In order to avoid some cumbersome notation we will also assume that all strings have the same length. Thus the complexity of the algorithm will be described in terms of m, the number of strings, and n, the length of each string.

Consider step 6.1 which is executed exactly $(n+1)^m-n^m$ times. In terms of the overall number of assignments for this algorithm this step takes up $1-1/(1+1/n)^m$ of the whole. Hence for a fixed string length, this step dominates the complexity requirements as the number of strings increases.

The number of times 6.2 is executed is n^m. This clearly dominates the number of comparisons made during each execution which is $O(m|\Sigma|)$ and hence serves as an appropriate measure of the complexity of this step. Thus with respect to the total number of assignments for the algorithm this step takes up $1/(1+1/n)^m$ of the whole. Accordingly, for a fixed number of strings, this step dominates the complexity requirements as the length of the strings increase.

5. The SCS-problem.

In this section we will present an algorithm for solving the SCS-problem. The treatment parallels the work done in section 4 so many of the details will be omitted. We will make use of the fact that a solution to the SM-problem when S consists of single symbol insert operations is also a solution to the SCS-problem. This is a direct consequence of the fact that the result of any merge sequence of insert operations must be a common supersequence of the original set of strings. Hence, for the remainder of this section, we will assume that $S=\{\text{ins}_{r,s}\mid r,s\geqq 1\}$ and that the cost of applying any operation is some positive fixed cost c_i.

Following the same approach that led to the development of algorithm 5 we can justify the correctness of the following algorithm for computing the cost array C for strings X and Y.

Algorithm 8.

Repeat the instructions in algorithm 5 substituting c_i for c_d throughout the program.

Using the same function α as in section 4 we can now present the general solution to the SCS-problem.

Algorithm 9.

9.1 $C(0,0,\ldots,0)=0$

9.2 **for** $(i_1:=0$ **to** $|X_1|$, $i_2:=0$ **to** $|X_2|,\ldots,i_m:=0$ **to** $|X_m|)$

if $(X_1\langle i_1\rangle = X_2\langle i_2\rangle = \ldots = X_m\langle i_m\rangle)$

then $C(i_1,i_2,\ldots,i_m)=C(i_1-1,i_2-1,\ldots,i_m-1)$

else $C(i_1,i_2,\ldots,i_m)=$

$$\min_{a\in\Sigma}\{C(\alpha(i_1,i_2,\ldots,i_m,K))+(m-|K|)c_i | K=\{k|X_k\langle i_k\rangle=a\}\neq\emptyset\}$$

end for

Unlike step 6.1 the initialization step 9.1 can only assign a value of zero to $C(0,0,\ldots,0)$. The reason for this is that for a set of strings containing an empty string, the solution for the LCS-problem must necessarily be the empty string, but the solution for the SCS-problem depends on the non-empty strings in the set. In step 9.2 the set K (a proper subset of $\{1,2,\ldots,m\}$) represents the elements that can be matched together for a minimum cost sequence. Hence $(m-|K|)$ inserts are needed to make the other sequences match. In terms of the terminology of [7], K represents the symbols that are to be "threaded" together.

This particular solution to the SCS-problem enables us to compute the length of a solution to the set $\{X_1,X_2,\ldots,X_m\}$ by setting $c_i=1$ and evaluating the expression

$$\left(\sum_{i=1}^{m}|X_i|+C(|X_1|,|X_2|,\ldots,|X_m|)\right)\Big/m\,.$$

A statistical quantity related to the similarity that exists amongst the strings is the term

$$\sigma_{\mathrm{SCS}} = 1\Big/\left(1+C(|X_1|,|X_2|,\ldots,|X_m|)\Big/\sum_{i=1}^{m}|X_i|\right)$$

which is the ratio of the average length of the original strings to the length of the shortest common supersequence. Since $0\leqq C(|X_1|,|X_2|,\ldots,|X_m|) \leqq (m-1)\sum_{i=1}^{m}|X_i|$, then $1/m\leqq\varrho_{\mathrm{SCS}}\leqq 1$. Again note that these expressions can be evaluated without actually determining any member of the set of shortest common supersequences. (The procedure to actually obtain a representative shortest common supersequence is almost identical to that for the LCS-problem and is omitted here.)

Since the total complexity of the algorithm is concentrated in step 9.2 and is $O((n+1)^m)$ we conclude that the algorithm is polynomial with respect to the length of the strings but goes exponentially with respect to the number of strings.

The following example illustrates the fact that a given set of strings may seem very homogeneous in terms of the solution to one problem, but very dissimilar in another case.

Example 10.

For any positive integer $m>1$ let $\Sigma=\{a_1, a_2, \ldots, a_m\}$ be a set of m distinct symbols. For $1\leqq i\leqq m$ let X_i be the string $a_1a_2\ldots a_{i-1}a_{i+1}\ldots a_m$ of length $m-1$ where a_i is the only symbol missing from Σ. Let the cost of insertion and deletion have unit value. Since the longest common subsequence is the empty string, but the shortest common supersequence is $a_1a_2\ldots a_m$ then $\varrho_{\mathrm{LCS}}=0$ but $\varrho_{\mathrm{SCS}}=(m-1)/m$. Hence, as m incresses, the difference between these "similarity" parameters approaches the maximum difference of one.

6. Summary.

We have formulated a string editing problem that encompasses the editing concepts in [11] and the consideration of an arbitrary number of strings as in [7]. An example illustrating the difference between this approach and that found in [11] was presented in section 2.

This approach was shown to lend itself very naturally to the problems of finding longest common subsequences and shortest common supersequences. In particular, we suggest that by modifying the algorithms presented in this paper with the techniques in [8] very efficient algorithms would be generated for these problems. Comparing the computational requirements for step 5.4 with steps 5 through 10 in algorithm X of [11], we see that our cost could be less if the strings have symbols in common. Here we should note that we have explicitly singled out the comparison between end symbols ($X\langle i\rangle$ and $Y\langle j\rangle$) whereas this action is implied in [11] by the expression $\gamma(A\langle i\rangle\rightarrow B\langle j\rangle)$.

The biological and data compression applications mentioned in [7] are still valid under the present formulation. Since we are not actually constrained to looking for solutions to the LCS-problem and SCS-problem, we actually have broadened the scope of possible applications. For example, in the case of data compression for several files, depending on the cost function γ, it may be desirable to store something else than the solution to the LCS or SCS problems. Also, in the field of molecular biology when studying the amino acid sequences of several proteins, perhaps the solution to the SM-problem for particular γ and S would be of considerable interest.

As a last remark we would like to suggest the following topic for further research. Our general formulation allows any mapping from strings to sets of

strings to be an edit operation. Clearly there is a large gap between this definition and the concrete examples used in the literature. Perhaps by limiting the operations to classes which enjoy well defined and well understood properties one may extend the theory presented in this paper. For instance, the basis examples of insertion, deletion and change can be viewed as a-transducer mappings [3]. From this point of view a natural hope would be to tap the wealth of knowledge in this area (AFL theory) for further results on the SM-problem.

REFERENCES

1. A. V. Aho, D. S. Hirschberg and J. D. Ullman. *The longest common subsequence problem.* J. ACM. 23,1 (Jan. 1976), 1–12.
2. V. L. Artazarov, E. A. Dinic, M. A. Kronrod and I. A. Faradzev. *On economical construction of the transitive closure of an oriented graph.* Soviet Math Dokl. 11,5 (1970), 1209–1210.
3. S. Ginsburg and S. A. Greibach, *Abstract families of languages.* In studies in *Abstract Families of Languages*, Memoir 87, Amer. Math. Soc., Providence, R.I., 1969, 1–32.
4. D. S. Hirschberg. *A linear space algorithm for computing maximal common subsequences.* Comm. ACM. 18,6 (June 1975), 341–343.
5. J. W. Hunt and T. G. Szymanski. *A fast algorithm for computing longest common subsequences.* Comm. ACM. 20,5 (May 1977), 350–353.
6. R. Lowrance and R. A. Wagner. *An extension of the string-to-string correction problem.* J. ACM 22,2 (April 1975), 177–183.
7. D. Maier. *The complexity of some problems on subsequences and supersequences.* J. ACM. 25, 2 (April 1978), 322–336.
8. W. J. Masek and M. S. Paterson. *A faster algorithm computing string edit distances.* TM–105, MIT Laboratory for Computer Science, (May 1978), 1–26.
9. D. Sankoff. *Matching sequences under deletion/insertion constraints.* Proc. Nat. Acad. Sci. USA 69,1 (Jan. 1972), 4–6.
10. P. H. Sellers. *An algorithm for the distance between two finite sequences.* J. Combin. Theory (A), 16 (1974), 253–258.
11. R. A. Wagner and M. J. Fischer. *The string to string correction problem.* J. ACM 21, 1 (Jan. 1974), 168–173.
12. C. K. Wong and A. K. Chandra. *Bounds for the string editing problem.* J. ACM. 23,1 (Jan. 1976), 13–16.

DEPARTMENT OF INFORMATION AND COMPUTER SCIENCE
UNIVERSITY OF HAWAII
HONOLULU, HI 96822