JOURNAL OF ALGORITHMS 14, 466–477 (1993)

On the Set LCS and Set–Set LCS Problems

BIING-FENG WANG AND GEN-HUEY CHEN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

AND

KUNSOO PARK

Department of Computing, King's College London, Strand, London WC2R 2LS, United Kingdom

Received October 1991; revised May 1992

We consider two generalizations of the longest common subsequence (LCS) problem: the Set LCS problem and the Set–Set LCS problem. We present algorithms for the two problems that are faster than the previous ones by Hirschberg and Larmore. © 1993 Academic Press, Inc.

1. INTRODUCTION

A *string* is a sequence of symbols in an alphabet Σ. String $\alpha = a_1 \cdots a_m$ is a *subsequence* of string $\beta = b_1 \cdots b_n$ if there is a mapping f: $\{1, \ldots, m\} \to \{1, \ldots, n\}$ such that $f(i) = j$ only if $a_i = b_j$ and f is a strictly increasing function; i.e., α is obtained by deleting zero or more symbols from β. String γ is a *common subsequence* of strings α and β if γ is a subsequence of α and also a subsequence of β. The longest common subsequence (LCS) problem is: Given two strings α and β, find a longest common subsequence of α and β.

We consider two generalizations of the LCS problem. Given a sequence of sets $\mathscr{A} = A_1 \cdots A_p$ (let $m = |A_1| + \cdots + |A_p|$), where each A_i is a set over Σ, we say that a string $\alpha = a_1 \cdots a_m$ is a *flattening* of $\mathscr{A}$ if α is the concatenation of p strings, the ith of which is a permutation of A_i. The Set LCS problem is: Given $\mathscr{A}$ and a string $\beta = b_1 \cdots b_n$, find a longest common subsequence of α and β, where α ranges over all flattenings of $\mathscr{A}$. The Set–Set LCS problem is: Given $\mathscr{A}$ and $\mathscr{B} = B_1 \cdots B_q$ (let

0196-6774/93 $5.00

$n = |B_1| + \cdots + |B_q|$), find a longest common subsequence γ of α and β, where α ranges over all flattenings of $\mathscr{A}$, and β over all flattenings of $\mathscr{B}$. We say that γ is a longest common subsequence of $\mathscr{A}$ and $\mathscr{B}$. These two problems have applications in computer-driven music accompaniment [2, 3, 4].

Hirschberg and Larmore gave an $O(mn)$ time algorithm for the Set LCS problem, and asked whether an $O(pn + m \log m)$ time algorithm is possible [3]. Later, they gave an $O(mn)$ time algorithm for the Set–Set LCS problem [4], which solves the Set LCS problem as a special case in the same time complexity. Huang and Asuri [5] proposed algorithms for the Set LCS and Set–Set LCS problems based on Hunt and Szymanski's approach [6] for the LCS problem. Unfortunately, their algorithms are faulty.

We give an improved version of Hirschberg and Larmore's algorithm for the Set–Set LCS problem [4]. Our algorithm maintains much less information during its computation than Hirschberg and Larmore's. We also show that it can be implemented in $O(pn + qm)$ time. Note that it still takes $O(mn)$ time for the Set LCS problem, since $q = n$. However, by changing a data structure of our Set–Set LCS algorithm, we obtain faster algorithms for the Set LCS problem: $O(m + pn \log(m/p))$ time or $O(m + pn\alpha(m/p + n, m/p + n))$ time, where $\alpha(\ )$ is a functional inverse of Ackermann's function. The time bound $O(m + pn \log(m/p))$ is never worse than $O(mn)$.

2. The Set–Set LCS Problem

We use dynamic programming to solve the Set–Set LCS problem. For $1 \le i \le p$ and $1 \le j \le q$, we define $E_A(i, j)$ and $E_B(i, j)$: $E_A(i, j)$ is the set of pairs (u, F) such that

(1) u is the length of a common subsequence γ of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_j$, and

(2) $F \subseteq A_i$ is the set of symbols of A_i which are not used by γ.

Similarly, $E_B(i, j)$ is the set of pairs (v, G) such that

(1) v is the length of a common subsequence γ of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_j$, and

(2) $G \subseteq B_j$ is the set of symbols of B_j which are not used by γ.

Then the length of a longest common subsequence of $\mathscr{A}$ and $\mathscr{B}$ is the largest u such that $(u, F) \in E_A(p, q)$ for some F. Initially, $E_A(i, 0) = \{(0, A_i)\}$ for $1 \le i \le p$, and $E_B(0, j) = \{(0, B_j)\}$ for $1 \le j \le q$.

LEMMA 1. *If* $(u, F) \in E_A(i, j-1)$ *and* S *is any subset of* $F \cap B_j$, *then* $(u + |S|, F - S) \in E_A(i, j)$ *and* $(u + |S|, B_j - S) \in E_B(i, j)$.

Proof. Let α be a common subsequence of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_{j-1}$ such that α has length u and $F \subseteq A_i$ is not used by α. Let β be any flattening of S. Then $\alpha\beta$ is a common subsequence of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_j$. $\alpha\beta$ has length $u + |S|$, and $F - S \subseteq A_i$ and $B_j - S \subseteq B_j$ are not used by $\alpha\beta$. □

We say that (u, F) *generates* $(u + |S|, F - S)$ and $(u + |S|, B_j - S)$. Similarly, $(v, G) \in E_B(i-1, j)$ generates elements of $E_A(i, j)$ and $E_B(i, j)$.

LEMMA 2 [4]. *An element of* $E_A(i, j)$ *and* $E_B(i, j)$ *is generated by either an element of* $E_A(i, j-1)$ *or an element of* $E_B(i-1, j)$.

Proof. Let $(u, F) \in E_A(i, j)$, and γ be a common subsequence of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_j$ such that γ has length u, and F and G for some $G \subseteq B_j$ are not used by γ. The flattenings of both $A_i - F$ and $B_j - G$ are suffixes of γ. Thus either $A_i - F \subseteq B_j - G$ or $B_j - G \subseteq A_i - F$. If $B_j - G \subseteq A_i - F$, let β be the suffix of γ which is a flattening of $S = B_j - G$, and α be the prefix of γ such that $\alpha\beta = \gamma$. Then α is a common subsequence of a flattening of $A_1 \cdots A_i$ and a flattening of $B_1 \cdots B_{j-1}$ such that α has length $u - |S|$, and $F \cup S \subseteq A_i$ is not used by α. $(u - |S|, F \cup S) \in E_A(i, j-1)$ generates (u, F). In the case $A_i - F \subseteq B_j - G$, (u, F) is generated by an element of $E_B(i-1, j)$. Similarly, $(v, G) \in E_B(i, j)$ is generated by either an element of $E_A(i, j-1)$ or an element of $E_B(i-1, j)$. □

By Lemmas 1 and 2, $E_A(i, j)$ and $E_B(i, j)$ can be computed from $E_A(i, j-1)$ and $E_B(i-1, j)$. A difficulty in this computation is that $E_A(i, j)$ and $E_B(i, j)$ may be very large sets. We eliminate many elements of $E_A(i, j)$ and $E_B(i, j)$ by the dominance relation: for $(u, F), (u', F') \in E_A(i, j)$ we say that (u, F) *dominates* (u', F') if

(1) $u \geq u'$, and

(2) $|F' - F| \leq u - u'$.

Similarly, for $(v, G), (v', G') \in E_B(i, j)$ we say that (v, G) dominates (v', G') if

(1) $v \geq v'$, and

(2) $|G' - G| \leq v - v'$.

LEMMA 3 [4]. *Any element of* $E_A(i, j)$ *and* $E_B(i, j)$ *which is not maximal with respect to the dominance relation can be discarded without affecting the length of a longest common subsequence of* $\mathscr{A}$ *and* $\mathscr{B}$.

Proof. By backward induction on i and j. The length of a longest common subsequence is obtained from a maximal element of $E_A(p, q)$ (or $E_B(p, q)$), and thus all elements in $E_A(p, q)$ that are not maximal may be discarded with no effect. Suppose $j < q$. Let $(u', F') \in E_A(i, j)$ be a nonmaximal element. Then (u', F') is dominated by some maximal element $(u, F) \in E_A(i, j)$. We will show that any element generated by (u', F') is dominated by an element generated by (u, F). Let $(u' + |S'|, F' - S')$ for $S' \subseteq F' \cap B_{j+1}$ be an element generated by (u', F'). Let $S = S' \cap F$. Then $(u + |S|, F - S)$ is generated by (u, F), since S is a subset of $F \cap B_{j+1}$. Furthermore,

(1) $(u + |S|) - (u' + |S'|) = (u - u') - |S' - S| \geq 0$, because $|S' - S| = |S' - (S' \cap F)| = |S' - F| \leq |F' - F| \leq u - u'$.

(2)

$$\begin{aligned} |(F' - S') - (F - S)| &= |(F' - F) - (S' - S)| \text{ (since } S = S' \cap F) \\ &= |F' - F| - |S' - S| \text{ (since } S' - S \subseteq F' - F) \\ &\leq (u - u') - |S' - S| \\ &= (u + |S|) - (u' + |S'|). \end{aligned}$$

Thus $(u + |S|, F - S)$ dominates $(u' + |S'|, F' - S')$. Similarly, $(u' + |S'|, B_{j+1} - S')$ generated by (u', F') is dominated by an element generated by (u, F). The case $i < p$ with $E_B(i, j)$ is similar. □

For $(u, F) \in E_A(i, j - 1)$, we define

$$f_A(u, F) = \big(u + |F \cap B_j|, F - B_j\big),$$
$$f_B(u, F) = \big(u + |F \cap B_j|, B_j - F\big).$$

For $(v, G) \in E_B(i - 1, j)$, we define

$$g_A(v, G) = \big(v + |A_i \cap G|, A_i - G\big),$$
$$g_B(v, G) = \big(v + |A_i \cap G|, G - A_i\big).$$

$f_A(u, F)$ is the maximal element of $E_A(i, j)$ that can be generated by (u, F), and $f_B(u, F)$ is the maximal element of $E_B(i, j)$ generated by (u, F). See Fig. 1.

We maintain only maximal elements of $E_A(i, j)$ and $E_B(i, j)$. Let $D_A(i, j)$ and $D_B(i, j)$ be the sets of maximal elements in $E_A(i, j)$ and $E_B(i, j)$, respectively. For all i and j, $D_A(i, 0) = E_A(i, 0)$ and $D_B(0, j) = E_B(0, j)$, since they contain only one element. To obtain $D_A(i, j)$, we compute $f_A(u, F)$ for all $(u, F) \in D_A(i, j - 1)$ and $g_A(v, G)$ for all $(v, G) \in$

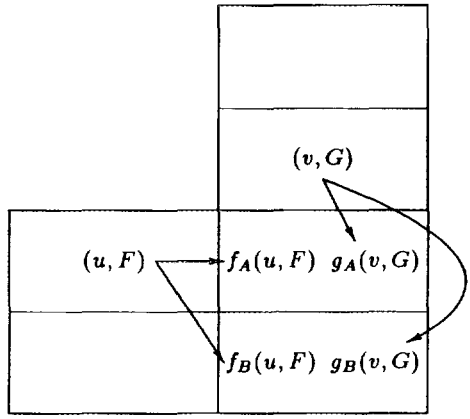

FIG. 1. Computing $E_A(i, j)$ and $E_B(i, j)$.

$D_B(i-1, j)$, and delete nonmaximal elements. Similarly, $D_B(i, j)$ is computed using f_B and g_B.

Consider an example $\mathscr{A}$ = {gredy} · {algorithm} · {cou} · {rse} and $\mathscr{B}$ = {comp} · {uter} · {scien} · {degr}, where commas are omitted inside braces. For each i and j, the top box contains $D_A(i, j)$ and the bottom box $D_B(i, j)$. The length of a longest common subsequence (omtrucser) is 9, as shown in Table 1.

Hirschberg and Larmore's algorithm [4] maintains triples (u, F, G) for each i and j; i.e., $E(i, j)$ is a set of triples. Their dominance relation is

TABLE 1

	comp	uter	scien	degr
gredy	(0, gredy)	(2, gdy)	(2, gdy)	(4, y)
	(0, comp)	(2, ut)	(2, scien)	(4, er)
algorithm	(2, algrith)	(4, algih)	(5, algh)	(6, alohm)
		(3, algorihm)	(4, algorhm)	(5, algoithm)
	(2, cp)	(4, ue)	(5, scen)	(6, der)
cou	(3, ou)	(5, co)	(6, ou)	(6, cou)
	(3, p)	(4, ter)	(6, sien)	(6, degr)
		(5, e)		
rse	(3, rse)	(6, rs)	(8, r)	(9, ∅)
	(3, p)	(6, t)	(7, cien)	(9, deg)
			(8, in)	

that (u, F, G) dominates (u', F', G') if

(1) $u \geq u'$,

(2) $|F' - F| \leq u - u'$, and

(3) $|G' - G| \leq u - u'$.

Our dominance relation is a stronger notion than Hirschberg and Larmore's. For example, (5, abc, d), (5, ab, de), and (5, a, def) $\in E(i, j)$ are all maximal by the definition above, whereas only (5, abc) $\in E_A(i, j)$ and (5, def) $\in E_B(i, j)$ are maximal in our dominance relation. Moreover, with Hirschberg and Larmore's dominance relation there is no obvious way to maintain only maximal triples; they maintained a larger set than the set of maximal triples.

The elements of $D_A(i, j)$ (also $D_B(i, j)$) satisfy a special property since they are maximal with respect to the dominance relation. Let $D_A(i, j) = \{(u_1, F_1), \ldots, (u_r, F_r)\}$. We define the *canonical sequence* of $D_A(i, j)$ to be $(u_1, F_1) \cdots (u_r, F_r)$ such that $u_1 \geq \cdots \geq u_r$. Lemma 4 will prove that the concept of canonical sequence is well defined.

LEMMA 4. *Let* $D_A(i, j) = \{(u_1, F_1), \ldots, (u_r, F_r)\}$. *The sequence* $(u_1, F_1) \cdots (u_r, F_r)$ *is the canonical sequence of* $D_A(i, j)$ *if and only if*

1. $u_1 > \cdots > u_r$,
2. $F_1 \subset \cdots \subset F_r$, and
3. $|F_{k+1} - F_k| \geq 2$ for $1 \leq k < r$.

Proof. First we prove the "only if" direction. We show by induction on i and j that $F_1 \subseteq \cdots \subseteq F_r$. It holds trivially for $D_A(i, 0)$ and $D_B(0, j)$. Suppose $i, j > 0$. The following three cases show that $F_1, \ldots, F_r$ are totally ordered by set inclusion.

(1) For $x = (u, F)$, $x' = (u', F') \in D_A(i, j - 1)$ such that $F \subseteq F'$, $f_A(x) = (u + |F \cap B_j|, F - B_j)$ and $f_A(x') = (u' + |F' \cap B_j|, F' - B_j)$. Thus $F - B_j \subseteq F' - B_j$.

(2) For $y = (v, G)$, $y' = (v', G') \in D_B(i - 1, j)$ such that $G \supseteq G'$, $g_A(y) = (v + |A_i \cap G|, A_i - G)$ and $g_A(x') = (v' + |A_i \cap G'|, A_i - G')$. Thus $A_i - G \subseteq A_i - G'$.

(3) For $x = (u, F) \in D_A(i, j - 1)$ and $y = (v, G) \in D_B(i - 1, j)$, $f_A(x) = (u + |F \cap B_j|, F - B_j)$ and $g_A(y) = (v + |A_i \cap G|, A_i - G)$. Since $F \subseteq A_i$ and $G \subseteq B_j$, $F - B_j \subseteq A_i - B_j \subseteq A_i - G$.

If the elements are maximal, $F_k \neq F_{k'}$ for $k < k'$, because otherwise (u_k, F_k) would dominate $(u_{k'}, F_{k'})$. Hence $F_1 \subset \cdots \subset F_r$. Also $u_1 > \cdots > u_r$ and $|F_{k+1} - F_k| \geq 2$ for $1 \leq k < r$ follow from the dominance relation. The other direction is obvious. □

Let $\alpha = (u_1, F_1) \cdots (u_r, F_r)$ and $\beta = (v_1, G_1) \cdots (v_s, G_s)$ be the canonical sequences of $D_A(i, j)$ and $D_B(i, j)$, respectively. Thanks to the special property of $D_A(i, j)$ and $D_B(i, j)$ we represent $D_A(i, j)$ by

$$d_A(\alpha) = (u_1, F_1) \cdot (u_2, F_2 - F_1) \cdots (u_r, F_r - F_{r-1}),$$

where each set is a list of symbols. This representation of $D_A(i, j)$ takes $O(|A_i|)$ space, and the number of elements in $D_A(i, j)$ is at most $\lfloor |A_i|/2 \rfloor + 1$ by Lemma 4. Since f_B and g_A reverse the order of set inclusion, we use the *reverse* sequence for $D_B(i, j)$. Furthermore, we maintain the *complement set* $B_j - G_k$ instead of G_k in order to keep the sets from smallest to largest; i.e., $(v_s, B_j - G_s) \cdots (v_1, B_j - G_1)$, which will be called the *reverse canonical sequence* of $D_B(i, j)$. We represent $D_B(i, j)$ by

$$d_B(\beta) = (v_s, B_j - G_s) \cdot (v_{s-1}, G_s - G_{s-1}) \cdots (v_1, G_2 - G_1),$$

because $(B_j - G_{k-1}) - (B_j - G_k) = G_k - G_{k-1}$. Note that $v_1 = u_1$, which is the length of a longest common subsequence of $A_1 \cdots A_i$ and $B_1 \cdots B_j$. For example, let $A_i = \{\text{abcdefghijkl}\}$, $B_j = \{\text{ghijklmn}\}$, and the canonical sequences of $D_A(i, j-1)$ and $D_B(i-1, j)$ be

$$\alpha = (14, \text{a}) \cdot (13, \text{abcg}) \cdot (11, \text{abcgdeh}),$$
$$\beta = (12, \text{m}) \cdot (11, \text{mkl}) \cdot (10, \text{mkljn}),$$

respectively. Then

$$d_A(\alpha) = (14, \text{a}) \cdot (13, \text{bcg}) \cdot (11, \text{deh}),$$
$$d_B(\beta) = (10, \text{ghi}) \cdot (11, \text{jn}) \cdot (12, \text{kl}).$$

Now we show how to compute $D_A(i, j)$ and $D_B(i, j)$ from $D_A(i, j-1)$ and $D_B(i-1, j)$. Let $\alpha = (u_1, F_1) \cdots (u_r, F_r)$ and $\beta = (v_1, G_1) \cdots (v_s, G_s)$ be the canonical sequences of $D_A(i, j-1)$ and $D_B(i-1, j)$, respectively. We define $f_A(\alpha) = f_A(u_1, F_1) \cdots f_A(u_r, F_r)$, $f_B(\alpha) = f_B(u_r, F_r) \cdots f_B(u_1, F_1)$, $g_A(\beta) = g_A(v_s, G_s) \cdots g_A(v_1, G_1)$, and $g_B(\beta) = g_B(v_1, G_1) \cdots g_B(v_s, G_s)$.

LEMMA 5. 1. $d_A(f_A(\alpha)) = (u'_1, F_1 - B_j) \cdot (u'_2, (F_2 - F_1) - B_j) \cdots (u'_r, (F_r - F_{r-1}) - B_j)$,

2. $d_B(f_B(\alpha)) = (u'_1, F_1 \cap B_j) \cdot (u'_2, (F_2 - F_1) \cap B_j) \cdots (u'_r, (F_r - F_{r-1}) \cap B_j)$,

3. $d_A(g_A(\beta)) = (v'_s, A_i - G_s) \cdot (v'_{s-1}, (G_s - G_{s-1}) \cap A_i) \cdots (v'_1, (G_2 - G_1) \cap A_i)$,

4. $d_B(g_B(\beta)) = (v'_s, B_j - (G_s - A_i)) \cdot (v'_{s-1}, (G_s - G_{s-1}) - A_i) \cdots (v'_1, (G_2 - G_1) - A_i)$,

where $u'_k = u_k + |F_k \cap B_j|$ and $v'_k = v_k + |A_i \cap G_k|$.

Proof. 1. Since $f_A(\alpha) = (u'_1, F_1 - B_j) \cdots (u'_r, F_r - B_j)$, we have $(F_k - B_j) - (F_{k-1} - B_j) = (F_k - F_{k-1}) - B_j$.

2. Since $f_B(\alpha) = (u'_r, B_j - F_r) \cdots (u'_1, B_j - F_1)$, its reverse canonical sequence is $(u'_1, B_j - (B_j - F_1)) \cdots (u'_r, B_j - (B_j - F_r))$. Since $B_j - (B_j - F_k) = F_k \cap B_j$, we have $(F_k \cap B_j) - (F_{k-1} \cap B_j) = (F_k - F_{k-1}) \cap B_j$.

3. Since $g_A(\beta) = (v'_s, A_i - G_s) \cdots (v'_1, A_i - G_1)$, we have $(A_i - G_{k-1}) - (A_i - G_k) = (G_k - G_{k-1}) \cap A_i$.

4. Since $g_B(\beta) = (v'_1, G_1 - A_i) \cdots (v'_s, G_s - A_i)$, its reverse canonical sequence is $(v'_s, B_j - (G_s - A_i)) \cdots (v'_1, B_j - (G_1 - A_i))$. We have

$$\begin{aligned}\big(B_j - (G_{k-1} - A_i)\big) - \big(B_j - (G_k - A_i)\big) &= (G_k - A_i) - (G_{k-1} - A_i) \\ &= (G_k - G_{k-1}) - A_i. \quad \square\end{aligned}$$

By Lemma 5, $d_A(f_A(\alpha))$ can be computed from $d_A(\alpha)$ in $O(|A_i| \times |B_j|)$ time: For each symbol $a \in F_k - F_{k-1}$ for $1 \le k \le r$ (assuming $F_0 = \varnothing$), delete a from $F_k - F_{k-1}$ if $a \in B_j$. Also $d_A(g_A(\beta))$ is computed from $d_B(\beta)$ in $O(|A_i| \times |B_j|)$ time. Let $d'_A(g_A(\beta))$ be $d_A(g_A(\beta))$ after subtracting $F_r - B_j$ from the symbol set of its first element (i.e., $d'_A(g_A(\beta)) = (v'_s, (A_i - G_s) - (F_r - B_j)) \cdot (v'_{s-1}, (G_s - G_{s-1}) \cap A_i) \cdots (v'_1, (G_2 - G_1) \cap A_i)$). Recall that $F_r - B_j$ is a subset of $A_i - G_s$ as shown in the proof of Lemma 4. Thus the sequence $d_A(f_A(\alpha)) \cdot d'_A(g_A(\beta))$ contains the elements of $D_A(i, j)$, including possible nonmaximal elements of $E_A(i, j)$, and it has $O(|A_i| + |B_j|)$ elements. We scan the sequence and eliminate nonmaximal elements by the conditions of Lemma 4 in $O(|A_i| + |B_j|)$ time. In fact, nonmaximal elements can be discarded during the computation of d_A. Similarly, $D_B(i, j)$ can be computed from $d_A(\alpha)$ and $d_B(\beta)$ in $O(|A_i| \times |B_j|)$ time. With the previous example,

$$\begin{aligned} d_A(f_A(\alpha)) &= (14, \mathrm{a}) \cdot (14, \mathrm{bc}) \cdot (13, \mathrm{de}), \\ d_B(f_B(\alpha)) &= (14, \varnothing) \cdot (14, \mathrm{g}) \cdot (13, \mathrm{h}), \\ d_A(g_A(\beta)) &= (13, \mathrm{abcdefghi}) \cdot (13, \mathrm{j}) \cdot (12, \mathrm{kl}), \\ d_B(g_B(\beta)) &= (13, \mathrm{ghijkl}) \cdot (13, \mathrm{n}) \cdot (12, \varnothing). \end{aligned}$$

Thus $D_A(i, j)$ is $(14, \mathrm{abc}) \cdot (13, \mathrm{defghij}) \cdot (12, \mathrm{kl})$ and $D_B(i, j)$ is $(14, \varnothing)$.

The longest common subsequence can be obtained by maintaining a parent pointer from an element to the element that generated it. Summarizing all discussions so far, we have the following theorem.

THEOREM 1. *The Set–Set LCS problem is solved in $O(mn)$ time.*

A more efficient implementation is possible by using symbols as indices of arrays. We change symbols to integers by a bijective mapping from Σ to $\{1, 2, \ldots, \sigma\}$, where $\sigma = |\Sigma|$.

1. Create an array whose size is as large as the size of the symbol set of a machine implementation.

2. Take symbols in Σ one by one, and give numbers 1 to σ (no need to initialize the array).

The construction of this mapping takes $O(\sigma)$ time. If the alphabet is not known in advance or it is not finite, σ is the number of distinct symbols in the input. In this case we compute the symbol mapping by constructing a balanced binary search tree, which takes $O((n + m)\log \sigma)$ time.

To compute d_A and d_B efficiently, we use two arrays C_A and C_B of size σ. Initially, all entries of C_A and C_B are set to 0. For computing $D_A(i, j)$ and $D_B(i, j)$, we set $C_A[a] = 1$ for each $a \in A_i$, and $C_B[b] = 1$ for each $b \in B_j$. Now $d_A(f_A(\alpha))$ and $d_B(f_B(\alpha))$ are computed in $O(|A_i|)$ time because we can check whether $a \in \Sigma$ is in B_j, or not, in constant time. Similarly, $d_A(g_A(\beta))$ and $d_B(g_B(\beta))$ are computed in $O(|B_j|)$ time. After the computation of $D_A(i, j)$ and $D_B(i, j)$, we reset $C_A[a] = 0$ for each $a \in A_i$, and $C_B[b] = 0$ for each $b \in B_j$. For each i, j the procedure above takes $O(|A_i| + |B_j|)$ time; the total time is $O(pn + qm)$. The initialization of the arrays C_A and C_B takes $O(\sigma)$ time. Since either σ is a constant or it is bounded by $m + n$, the overall computation takes $O(pn + qm)$ time.

THEOREM 2. *If $\Sigma = \{1, \ldots, \sigma\}$, the Set–Set LCS problem is solved in $O(pn + qm)$ time.*

3. THE SET LCS PROBLEM

The Set LCS problem is a special case of the Set–Set LCS problem in which each B_j is a singleton set $\{b_j\}$. The time complexity of Theorem 2 is still $O(mn)$ for the Set LCS problem, since $q = n$. We improve this bound using a different representation for $D_A(i, j)$ and $D_B(i, j)$.

By Lemma 4 there is only one element (v, G) in $D_B(i, j)$, and G is either $\{b_j\}$ or $\varnothing$. Let $\alpha = (u_1, F_1) \cdots (u_r, F_r)$ be the canonical sequence of $D_A(i, j)$. We represent $D_A(i, j)$ by

$$d'_A(\alpha) = (u_1, F_1) \cdot (u_2 - u_1, F_2 - F_1) \cdots (u_r - u_{r-1}, F_r - F_{r-1}) \cdot (*, A_i - F_r),$$

where * denotes undefined. Note that the symbol sets of $d'_A(\alpha)$ are a partition of A_i. Huang and Asuri's algorithm for the Set LCS problem [5] maintains only (u_1, F_1) among the elements of $D_A(i, j)$. Thus their algorithm cannot get an LCS which comes from (u_k, F_k) for $k \geq 2$.

We show how to compute $D_A(i, j)$ and $D_B(i, j)$ from $D_A(i, j-1)$ and $D_B(i-1, j)$. Let $D_B(i-1, j) = \{(v, G)\}$, $\alpha = (u_1, F_1) \cdots (u_r, F_r)$ be the canonical sequence of $D_A(i, j-1)$, and $d'_A(\alpha) = (w_1, P_1) \cdots (w_{r+1}, P_{r+1})$. The computation of $D_B(i, j)$ is easy. Recall that $f_B(u, F) = (u + |F \cap B_j|, F - B_j)$. Since B_j contains only one symbol, $u_1 + |F_1 \cap B_j| \geq u_k + |F_k \cap B_j|$ for $k \geq 2$. Since $B_j - F_1 \supseteq B_j - F_k$, $f_B(u_1, F_1)$ dominates $f_B(u_k, F_k)$ for $k \geq 2$. Thus $D_B(i, j)$ is the dominant element of $f_B(u_1, F_1)$ and $g_B(v, G)$.

For $D_A(i, j)$, we do the following:

(1) If $b_j \in P_1$, then replace (w_1, P_1) by $(w_1 + 1, P_1 - B_j)$.

(2) If $b_j \in P_k$ for $2 \leq k \leq r$, then replace (w_k, P_k) by $(w_k + 1, P_k - B_j)$. This new element dominates (w_{k-1}, P_{k-1}) if $w_k + 1 = 0$, in which case we combine the two elements into $(w_{k-1}, P_{k-1} \cup (P_k - B_j))$. If $b_j \notin P_k$ for any $1 \leq k \leq r$, then $(w_1, P_1) \cdots (w_r, P_r)$ remains the same.

(3) Add $g_A(v, G) = (v + |A_i \cap G|, A_i - G)$ to the end of the new sequence. Since we maintain $P_{r+1} = A_i - F_r$, new sets $(A_i - G) - F_r$ and $A_i - (A_i - G)$ can be obtained by possibly deleting b_j from P_{r+1} and creating a set $\{b_j\}$.

(4) Determine the dominance relation between $g_A(v, G)$ and neighboring elements. To determine the dominance relation efficiently, we store the number $|P_k|$ in the data structure (described below) implementing the symbol set P_k for all k, and we also maintain u_r separately. Since G is either $\{b_j\}$ or $\emptyset$, the size of new set $(A_i - G) - F_r$ is easily computed. Now the dominance relation can be determined by comparing numbers (i.e., u's and sizes). Lemma 6 shows that $g_A(v, G)$ can dominate at most two elements.

LEMMA 6. *Let* $g_A(v, G) = (v', A_i - G)$, *and* $f_A(u_r, F_r) = (u', F')$. *Then* $v' \leq u' + 1$.

Proof. Let (v'', G'') be the element of $D_B(i-1, j-1)$. Then $v = v''$ or $v = v'' + 1$ because B_j contains one element. Furthermore, if $v = v'' + 1$, then $G = \emptyset$. Thus $v' = v''$ or $v' = v'' + 1$. Since $g_A(v'', G'')$ is an element of $D_A(i, j-1)$, $u_r \geq v''$. Thus $v' \leq u' + 1$. □

Consequently, the computation of $D_A(i, j)$ and $D_B(i, j)$ requires a constant number of the following operations.

1. create(a): Create a set $\{a\}$.
2. find(a): Find the set containing symbol a.
3. merge(S_1, S_2): Merge two sets S_1 and S_2 into one.
4. delete(a, S): Delete symbol a from set S.

The 2–3 trees [1] support each of these operations in $O(\log N)$ time, where N is the number of symbols, once a way to locate symbols in the data structure is given. To locate symbols, we can use an array of pointers whose size is σ. As in the Set–Set LCS problem, one array can be used for all i. Thus the array initialization for all i takes $O(\sigma + m)$ time. Since $O(|A_i|)$ time is required to construct the data structure for $D_A(i, 0)$, the time required for each i with 2–3 trees is $O(|A_i| + n \log|A_i|)$. The total time is $O(m + n(\log|A_1| + \cdots + \log|A_p|))$, which is $O(m + pn \log(m/p))$ by the concavity of the log function. Note that this time bound is never worse than $O(mn)$.

The disjoint set union algorithms [7, 8] support M operations of the first three in $O(M\alpha(M, N))$ time, where $\alpha(M, N)$ is a functional inverse of Ackermann's function. The algorithms do not support the delete operation, but there is a method to accommodate it, which is called *lazy delete*: To delete a symbol, we just mark it deleted. With lazy delete, the data structure keeps growing as we do delete(a, S) and create(a) operations. Thus for each i, $O(|A_i| + n\alpha(|A_i| + n, |A_i| + n))$ time is required. The total time is $O(m + pn\alpha(m/p + n, m/p + n))$.

THEOREM 3. *The Set LCS problem is solved either in* $O(m + pn \log(m/p))$ *time or in* $O(m + pn\alpha(m/p + n, m/p + n))$ *time.*

4. CONCLUSION

We have presented efficient algorithms for the set LCS and Set–Set LCS problems. Like Hirschberg and Larmore's algorithm [4], our algorithms can be adapted to a more general problem where each A_i (and B_j) is a multiset (i.e., elements of A_i are not necessarily distinct). We modify the algorithms as follows.

1. $O(mn)$ algorithm: In computing $d_A(f_A(\alpha))$ of Lemma 5 (similar changes for $d_B(f_B(\alpha))$, $d_A(g_A(\beta))$, and $d_B(g_B(\beta))$), all elements of B_j are unmarked initially. If $a \in F_k - F_{k-1}$ is an unmarked symbol in B_j, then delete a from $F_k - F_{k-1}$ and also mark a in B_j.

2. $O(pn + qm)$ algorithm: $C_A[a]$ contains the number of the symbol a in A_i. Similarly, $C_B[b]$ is the number of b in B_j. Now if $C_B[a] \geq 1$ for $a \in F_k - F_{k-1}$, then delete a from $F_k - F_{k-1}$ and decrease $C_B[a]$ by 1.

3. Set LCS algorithm: To locate symbols in the data structure, we use an array of size σ. Each element of the array points to a circular list whose elements point to symbols in the data structure. The order in a circular list

is induced by the order of P_k's in $d'_A(a)$. And find(a) returns the first set that contains a.

The time bounds of the algorithms remain unchanged.

Our algorithm for the Set LCS problem raises an interesting open problem. Is there a data structure that supports the four operations efficiently: create(a), find(a), merge(S_1, S_2), and delete(a, S)? For example, a data structure with $O(\log \log N)$ time for each operation or a data structure with a good amortized time might be pursued.

REFERENCES

1. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA, 1974.
2. J. J. BLOCH AND R. B. DANNENBERG, Real-time computer accompaniment of keyboard performances," *in* "Proc., 1985 International Computer Music Conference, August 1985."
3. D. S. HIRSCHBERG AND L. L. LARMORE, The set LCS problem, *Algorithmica* **2** (1987), 91–95.
4. D. S. HIRSCHBERG AND L. L. LARMORE, The set–set LCS problem, *Algorithmica* **4** (1989), 503–510.
5. S. S. HUANG AND H. S. ASURI, Algorithms for the set-LCS and the set–set LCS problems, *in* "Computing and Information" (R. Janicki and W. W. Koczkodaj, Eds.), pp. 81–87, Elsevier, Amsterdam/New York, 1989.
6. J. W. HUNT AND T. G. SZYMANSKI, A fast algorithm for computing longest common subsequences, *Comm. ACM* **20** (1977), 350–353.
7. R. E. TARJAN, "Data Structures and Network Algorithms," SIAM, Philadelphia, 1983.
8. R. E. TARJAN AND J. VAN LEEUWEN, Worst-case analysis of set union algorithms, *J. Assoc. Comput. Mach.* **31** (1984), 245–281.