

Article

A Simple Algorithm for Solving for the Generalized Longest Common Subsequence (LCS) Problem with a Substring Exclusion Constraint

Daxin Zhu ¹ and Xiaodong Wang ^{1,2,*}

¹ Faculty of Mathematics & Computer Science, Quanzhou Normal University, Quanzhou 362000, China; E-Mail: dex@qztc.edu.cn

² Faculty of Mathematics & Computer Science, Fuzhou University, Fuzhou 350108, China

* Author to whom correspondence should be addressed; E-Mail: wangxiaodong@qztc.edu.cn; Tel.: +86-595-22916878; Fax: +86-595-22796091.

Received: 14 June 2013; in revised form: 15 July 2013 / Accepted: 24 July 2013 /

Published: 15 August 2013

Abstract: This paper studies the string-excluding (STR-EC)-constrained longest common subsequence (LCS) problem, a generalized LCS problem. For the two input sequences, X and Y , of lengths n and m and a constraint string, P , of length r , the goal is to find the longest common subsequence, Z , of X and Y that excludes P as a substring. The problem and its solution were first proposed by Chen and Chao, but we found that their algorithm cannot solve the problem correctly. A new dynamic programming solution for the STR-EC-LCS problem is then presented in this paper, and the correctness of the new algorithm is proven. The time complexity of the new algorithm is $O(nmr)$.

Keywords: constrained LCS; string-excluding; dynamic programming

1. Introduction

In this paper, we consider a generalized longest common subsequence problem. The longest common subsequence (LCS) problem is a well-known measurement for computing the similarity of two strings. This problem can be widely applied in diverse areas, such as file comparison, pattern matching and computational biology [1].

A sequence is an ordered list of characters over an alphabet, Σ . A subsequence of a sequence, X , is obtained by deleting zero or more characters (not necessarily contiguous) from X . A substring of a sequence, X , is a subsequence of successive characters within X .

For a given sequence, $X = x_1x_2 \cdots x_n$, of length n , the i th character of X is denoted, $x_i \in \Sigma$, for any $i = 1, \dots, n$. A substring of X from position i to j can be denoted as $X[i : j] = x_ix_{i+1} \cdots x_j$. A substring, $X[i : j]$, is called a prefix of X if $i = 1$ and a suffix of X if $j = n$.

Given two sequences, X and Y , the LCS problem is finding a subsequence of X and Y whose length is the longest among all common subsequences of the two given sequences.

For some biological applications, some constraints must be applied to the LCS problem. These types of variants of the LCS problem are called constrained LCS (CLCS) problems [2].

A recent variant of the LCS problem, which was first addressed in [2], has received considerable attention. The most cited algorithms solve the CLCS problem based on dynamic programming algorithms. Some improved algorithms have also been proposed in [3,4]. The LCS and CLCS problems on indeterminate strings were also discussed in [4]. A bit-parallel algorithm for solving the CLCS problem was proposed in [3]. The problem was extended to have weighted constraints, a more generalized problem, in [5]. A variant of the CLCS problem with multiple constraints, the restricted LCS problem, which excludes the given constraint as a subsequence of the answer, was proposed in [6]. This restricted LCS problem becomes non-deterministic polynomial-time hard (NP-hard) when the number of constraints is not fixed [6].

Recently, Chen and Chao [7] proposed a more generalized form of the CLCS problem, the generalized-constrained-LCS (GC-LCS) problem. For the two input sequences, X and Y , of lengths n and m , respectively, and a constraint string, P , of length r , the GC-LCS problem is a set of four problems that find the LCS of X and Y that includes/excludes P as a subsequence/substring. The four generalized constrained LCSs are summarized in Table 1 [7].

Table 1. The generalized-constrained-longest common subsequence (GC-LCS) problems. STR-EC, string-excluding.

Problem	Input	Output
SEQ-IC-LCS	X, Y , and P	The LCS of X and Y that includes P as a subsequence
STR-IC-LCS	X, Y , and P	The LCS of X and Y that includes P as a substring
SEQ-EC-LCS	X, Y , and P	The LCS of X and Y that includes P as a subsequence
STR-EC-LCS	X, Y , and P	The LCS of X and Y that includes P as a substring

We will discuss the STR-EC-LCS problem in this paper. We found that a previously proposed dynamic programming algorithm for the STR-EC-LCS problem [7] cannot correctly solve the problem. Let $L(i, j, k)$ denote the length of an LCS of $X[1 : i]$ and $Y[1 : j]$, excluding $P[1 : k]$ as a substring. Chen and Chao gave a recursive Formula (1) for computing $L(i, j, k)$ as follows.

$$L(i, j, k) = \begin{cases} L(i-1, j-1, k) & \text{if } k=1 \text{ and } x_i = y_j = p_k, \\ 1 + \max\{L(i-1, j-1, k-1), L(i-1, j-1, k)\} & \text{if } k \geq 2 \text{ and } x_i = y_j = p_k, \\ 1 + L(i-1, j-1, k) & \text{if } x_i = y_j \text{ and } (k > 0 \text{ and } x_i \neq p_k), \\ \max\{L(i-1, j, k), L(i, j-1, k)\} & \text{if } x_i \neq y_j \end{cases} \quad (1)$$

The boundary conditions of this recursive formula are $L(i, 0, k) = L(0, j, k) = 0$ for any $0 \leq i \leq n$, $0 \leq j \leq m$ and $0 \leq k \leq r$.

The algorithm presented in [7] was stated without strict proof. Thus, the correctness of the proposed algorithm cannot be guaranteed. For example, if $X = abbb$, $Y = aab$ and $P = ab$, the values of $L(i, j, k)$, $1 \leq i \leq 4$, $1 \leq j \leq 3$, $0 \leq k \leq 2$ computed by recursive Formula (1) are listed in Table 2.

Table 2. $L(i, j, k)$ computed by recursive Formula (1).

	k = 0				k = 1			k = 2	
$i = 1$	1	1	1	0	0	0	1	1	1
$i = 2$	1	1	2	0	0	1	1	1	2
$i = 3$	1	1	2	0	0	1	1	1	2
$i = 4$	1	1	2	0	0	1	1	1	2

From Table 2, we know that the final answer is $L(4, 3, 2) = 2$, which is computed by the formula, $L(4, 3, 2) = 1 + L(3, 2, 2)$, since, in this case, $k \geq 2$ and $a_4 = b_3 = p_2 = b'$. However, this is a wrong answer, since the correct answer should be one.

A new dynamic solution for the STR-EC-LCS problem is presented in this paper, and the correctness of the new algorithm is proven. The time complexity of the new algorithm is $O(nmr)$.

The organization of the paper is as follows.

In the following three sections, we describe our dynamic programming algorithm for the STR-EC-LCS problem.

In Section 2, we present a new dynamic programming solution for the STR-EC-LCS problem with time complexity, $O(nmr)$, from a novel perspective. In Section 3, we discuss the issues involved in implementing the algorithm efficiently. Some concluding remarks are provided in Section 4.

2. A Simple Dynamic Programming Solution

For the two input sequences, $X = x_1x_2 \cdots x_n$ and $Y = y_1y_2 \cdots y_m$, of lengths n and m , respectively, and a constraint string, $P = p_1p_2 \cdots p_r$, of length r , we want to find an LCS of X and Y that excludes P as a substring.

In the description of our new algorithm, a function, σ , will be mentioned frequently. For any string, S , and a fixed constraint string, P , the length of the longest suffix of S that is also a prefix of P is denoted by the function, $\sigma(S)$.

The function, σ , refers to both P and S . Because the string, S , is a variable and the constraint string, P , is fixed, the notation, $\sigma(S)$, will not cause confusion, even though it does not reflect its dependence on P .

The symbol, $\oplus$, is also used to denote string concatenation.

For example, if $P = aaba$ and $S = aabaaab$, then substring aab is the longest suffix of S that is also a prefix of P ; therefore, $\sigma(S) = 3$.

It is readily seen that $S \oplus P = aabaaabaaba$.

Let $Z(i, j, k)$ denote the set of all LCSs of $X[i : n]$ and $Y[j : m]$ that exclude P as a substring of $P[1 : k] \oplus z$ for each $z \in Z(i, j, k)$, $1 \leq i \leq n$, $1 \leq j \leq m$, $0 \leq k \leq r$. $P[1 : k]$ is an empty string if $k = 0$. The length of an LCS in $Z(i, j, k)$ is denoted $f(i, j, k)$.

If we can compute $f(i, j, k)$ for any $1 \leq i \leq n$, $1 \leq j \leq m$ and $0 \leq k < r$ efficiently, then the length of an LCS of X and Y that excludes P as a substring must be $f(1, 1, 0)$.

We can obtain a recursive formula for computing $f(i, j, k)$ with the following theorem.

Theorem 1 For the two input sequences, $X = x_1x_2 \cdots x_n$ and $Y = y_1y_2 \cdots y_m$, of lengths n and m , respectively, and a constraint string, $P = p_1p_2 \cdots p_r$, of length r , let $Z(i, j, k)$ denote the set of all LCSs of $X[i : n]$ and $Y[j : m]$ that exclude P as a substring of $P[1 : k] \oplus z$ for each $z \in Z(i, j, k)$.

The length of an LCS in $Z(i, j, k)$ is denoted, $f(i, j, k)$.

For any $1 \leq i \leq n$, $1 \leq j \leq m$ and $0 \leq k < r$, $f(i, j, k)$ can be computed with the following recursive Formula (2):

$$f(i, j, k) = \begin{cases} \max \{f(i+1, j+1, k), 1 + f(i+1, j+1, q)\} & \text{if } x_i = y_j \text{ and } q < r \\ \max \{f(i+1, j, k), f(i, j+1, k)\} & \text{otherwise} \end{cases} \quad (2)$$

where $q = \sigma(P[1 : k] \oplus x_i)$, and the boundary conditions are $f(i, m+1, k) = f(n+1, j, k) = 0$ for any $1 \leq i \leq n$, $1 \leq j \leq m$ and $0 \leq k \leq r$.

Proof. For any $1 \leq i \leq n$, $1 \leq j \leq m$ and $0 \leq k < r$, suppose $f(i, j, k) = t$ and $z = z_1, \dots, z_t \in Z(i, j, k)$.

First, we note that for each pair, (i', j') , $1 \leq i' \leq n$, $1 \leq j' \leq m$, such that $i' \geq i$ and $j' \geq j$, we have $f(i', j', k) \leq f(i, j, k)$, because a common subsequence, z , of $X[i' : n]$ and $Y[j' : m]$ that excludes P as a substring of $P[1 : k] \oplus z$ is also a common subsequence of $X[i : n]$ and $Y[j : m]$ that excludes P as a substring of $P[1 : k] \oplus z$.

(1) When $x_i \neq y_j$, we have $x_i \neq z_1$ or $y_j \neq z_1$.

(1.1) If $x_i \neq z_1$, then $z = z_1, \dots, z_t$ is a common subsequence of $X[i+1 : n]$ and $Y[j : m]$ that excludes P as a substring of $P[1 : k] \oplus z$; thus, $f(i+1, j, k) \geq t$. In contrast, $f(i+1, j, k) \leq f(i, j, k) = t$. Therefore, in this case, we have $f(i, j, k) = f(i+1, j, k)$.

(1.2) If $y_j \neq z_1$, then in a similar manner, we can prove that $f(i, j, k) = f(i, j+1, k)$ in this case.

Combining the two subcases, we conclude that when $x_i \neq y_j$, we have

$$f(i, j, k) = \max \{f(i+1, j, k), f(i, j+1, k)\}$$

(2) When $x_i = y_j$ and $q < r$, there are also two subcases to be distinguished.

(2.1) If $x_i = y_j \neq z_1$, then $z = z_1, \dots, z_t$ is also a common subsequence of $X[i+1 : n]$ and $Y[j+1 : m]$ that excludes P as a substring of $P[1 : k] \oplus z$ and, thus, $f(i+1, j+1, k) \geq t$. In contrast, $f(i+1, j+1, k) \leq f(i, j, k) = t$. Therefore, we have $f(i, j, k) = f(i+1, j+1, k)$ in this case.

(2.2) If $x_i = y_j = z_1$, then $f(i, j, k) = t > 0$ and $z = z_1, \dots, z_t$ is an LCS of $X[i : n]$ and $Y[j : m]$ that excludes P as a substring of $P[1 : k] \oplus z$, and thus, $z' = z_2, \dots, z_t$ is a common subsequence of $X[i+1 : n]$ and $Y[j+1 : m]$ that excludes P as a substring of $P[1 : k] \oplus x_i \oplus z'$.

If $q = \sigma(P[1 : k] \oplus x_i)$, then $P[1 : q]$ is the longest suffix of $P[1 : k] \oplus x_i$ that is also a prefix of P . It follows that $P[1 : q] \oplus z'$ is a suffix of $P[1 : k] \oplus x_i \oplus z'$. Therefore, a sequence that excludes P as a substring of $P[1 : k] \oplus x_i \oplus z'$ is also a sequence that excludes P as a substring of $P[1 : q] \oplus z'$. It follows from the fact that $z' = z_2, \dots, z_t$ is a common subsequence of $X[i + 1 : n]$ and $Y[j + 1 : m]$ that excludes P as a substring of $P[1 : k] \oplus x_i \oplus z'$ that $z' = z_2, \dots, z_t$ is also a common subsequence of $X[i + 1 : n]$ and $Y[j + 1 : m]$ that excludes P as a substring of $P[1 : q] \oplus z'$. In other words:

$$f(i + 1, j + 1, q) \geq t - 1 = f(i, j, k) - 1 \quad (3)$$

In contrast, if $P[1 : q]$ is the longest suffix of $P[1 : k] \oplus x_i$, $f(i + 1, j + 1, q) = s$ and $v = v_1, \dots, v_s \in Z(i + 1, j + 1, q)$, then v is an LCS of $X[i + 1 : n]$ and $Y[j + 1 : m]$ that excludes P as a substring of $P[1 : q] \oplus v$. In this case, $v' = x_i \oplus v$ is a common subsequence of $X[i : n]$ and $Y[j : m]$ that excludes P as a substring of $P[1 : k] \oplus x_i \oplus v'$, because $P[1 : q]$ is the longest suffix of $P[1 : k] \oplus x_i$ and $q < r$. Therefore:

$$f(i, j, k) \geq s + 1 = f(i + 1, j + 1, q) + 1 \quad (4)$$

Combining (3) and (4), we have:

$$f(i, j, k) = 1 + f(i + 1, j + 1, q) \quad (5)$$

Combining the two subcases, where $x_i = y_j$ and $q < r$, we conclude that the recursive Formula (2) is correct for this case.

(3) When $x_i = y_j$ and $q = r$, we must have $x_i = y_j \neq z_1$; otherwise, $P[1 : k] \oplus z$ will include the string, $P[1 : k] \oplus x_i = P$. Similar to Subcase (2.1), we can conclude that in this case,

$$f(i, j, k) = f(i + 1, j + 1, k) = \max\{f(i + 1, j, k), f(i, j + 1, k)\}$$

The proof is complete. ■

3. Implementation of the Algorithm

According to Theorem 1, our new algorithm for computing $f(i, j, k)$ is a standard dynamic programming algorithm. With the recursive Formula (2), the new dynamic programming algorithm for computing $f(i, j, k)$ can be implemented as the following Algorithm 1.

To implement our new algorithm efficiently, it is important to compute $\sigma(P[1 : k] \oplus x_i)$ for each $0 \leq k < r$ and x_i , where $1 \leq i \leq n$ efficiently in line 8.

It is clear that $\sigma(P[1 : k] \oplus x_i) = k + 1$ when $x_i = p_{k+1}$. It will be more complex to compute $\sigma(P[1 : k] \oplus x_i)$ when $x_i \neq p_{k+1}$. In this case, the length of the matched prefix of P must be shortened to the largest $t < k$, such that $p_{k-t+1} \dots p_k = p_1 \dots p_t$ and $x_i = p_{t+1}$. Therefore, in this case, $\sigma(P[1 : k] \oplus x_i) = t + 1$.

This computation is very similar to the computation of the prefix function in the Knuth-CMorris-CPratt string searching algorithm (KMP algorithm) for solving the string matching problem [8].

Algorithm 1 STR-EC-LCS.

Input: Strings $X = x_1 \cdots x_n$, $Y = y_1 \cdots y_m$ of lengths n and m , respectively, and a constraint string, $P = p_1 \cdots p_r$, of lengths r

Output: The length of an LCS of X and Y that excludes P as a substring

```

1: for all  $i, j, k$ ,  $1 \leq i \leq n$ ,  $1 \leq j \leq m$ , and  $0 \leq k \leq r$  do
2:    $f(i, m+1, k) \leftarrow 0$ ,  $f(n+1, j, k) \leftarrow 0$  {boundary condition}
3: end for
4: for  $i = n$  down to 1 do
5:   for  $j = m$  down to 1 do
6:     for  $k = 0$  to  $r-1$  do
7:        $f(i, j, k) \leftarrow \max\{f(i+1, j, k), f(i, j+1, k)\}$ 
8:        $q \leftarrow \sigma(P[1:k] \oplus x_i)$ 
9:       if  $x_i = y_j$   $q < r$  then
10:         $f(i, j, k) \leftarrow \max\{f(i+1, j+1, k), 1 + f(i+1, j+1, q)\}$ 
11:       end if
12:     end for
13:   end for
14: end for
15: return  $f(1, 1, 0)$ 

```

For a given string, $S = s_1 \cdots s_n$, the prefix function, $kmp(i)$, denotes the length of the longest prefix of $s_1 \cdots s_{i-1}$ that matches a suffix of $s_1 \cdots s_i$. For example, if $S = ababaa$, then $kmp(1), \dots, kmp(6) = 0, 0, 1, 2, 3, 1$.

For the constraint string, $P = p_1 \cdots p_r$, of length r , its prefix function, kmp , can be pre-computed in $O(r)$ time by Algorithm 2.

Algorithm 2 Prefix Function.

Input: String $P = p_1 \cdots p_r$

Output: The prefix function kmp of P

```

1:  $kmp(0) \leftarrow -1$ 
2: for  $i = 2$  to  $r$  do
3:    $k \leftarrow 0$ 
4:   while  $k \geq 0$   $p_{k+1} \neq p_i$  do
5:      $k \leftarrow kmp(k)$ 
6:   end while
7:    $k \leftarrow k + 1$ 
8:    $kmp(i) \leftarrow k$ 
9: end for

```

With this pre-computed prefix function, kmp , the function, $\sigma(P[1:k] \oplus ch)$, for each character, $ch \in \Sigma$ and $1 \leq k \leq r$, can be described as Algorithm 3.

To accelerate the processing, we can pre-compute a table, $\lambda(k, ch)$, of the function, $\sigma(P[1 : k] \oplus ch)$, for each character, $ch \in \Sigma$ and $1 \leq k \leq r$. It is clear that $\lambda(k - 1, P[k]) = k$ for each $1 \leq k \leq r$. The other values of the table, λ , can be computed by using the prefix function, kmp , in the following recursive Algorithm 4.

Algorithm 3 $\sigma(k, ch)$.

Input: String $P = p_1 \cdots p_r$, integer k and character ch

Output: $\sigma(P[1 : k] \oplus ch)$

```

1: while  $k \geq 0$   $p_{k+1} \neq ch$  do
2:    $k \leftarrow kmp(k)$ 
3: end while
4: return  $k + 1$ 

```

Algorithm 4 $\lambda(k, ch)$.

Input: Integer k , character ch

Output: Value of $\lambda(k, ch)$

```

1: if  $k > 0$   $\lambda(k, ch) = 0$  then
2:    $\lambda(k, ch) \leftarrow \lambda(kmp(k), ch)$ 
3: end if
4: return  $\lambda(k, ch)$ 

```

The time cost of the above preprocessing algorithm is clearly $O(r|\Sigma|)$. By using this pre-computed table, λ , the value of function $\sigma(P[1 : k] \oplus ch)$ for each character, $ch \in \Sigma$ and $1 \leq k < r$, can be computed readily in $O(1)$ time.

With this pre-computed table, λ , the loop body of the above Algorithm 1 requires only $O(1)$ time, because $\lambda(k, x_i)$ can be computed in $O(1)$ time for each x_i , $1 \leq i \leq n$ and any $0 \leq k < r$. Therefore, our new algorithm for computing the length of an LCS of X and Y that excludes P as a substring requires $O(nmr)$ time and $O(r|\Sigma|)$ preprocessing time.

If we want to obtain the actual LCS of X and Y that excludes P as a substring, not only its length, we can also present a simple recursive back-tracing algorithm for this purpose as Algorithm 5.

At the end of our new algorithm, a function call, $back(1, 1, 0)$, will produce the resultant LCS accordingly.

Because the cost of the computation of $\lambda(k, x_i)$ is $O(1)$, the algorithm, $back(i, j, k)$, will cost $O(n + m)$ in the worst case.

Finally, we summarize our results in the following theorem:

Theorem 2 Algorithm 1 solves the STR-EC-LCS problem correctly in $O(nmr)$ time and $O(nmr)$ space, with preprocessing time $O(r|\Sigma|)$.

Algorithm 5 *back*(i, j, k).

Comments: A recursive back tracing algorithm to construct the actual LCS.

```

1: if  $i > n$   $j > m$  then
2:   return
3: end if
4: if  $x_i = y_j$   $f(i, j, k) = 1 + f(i + 1, j + 1, \lambda(k, x_i))$  then
5:   print  $x_i$ 
6:   back( $i + 1, j + 1, \lambda(k, x_i)$ )
7: else if  $f(i + 1, j, k) > f(i, j + 1, k)$  then
8:   back( $i + 1, j, k$ )
9: else
10:  back( $i, j + 1, k$ )
11: end if

```

4. Conclusions

We have suggested a new dynamic programming solution for the STR-EC-LCS problem. The new algorithm corrects a previously presented dynamic programming algorithm with the same time and space complexities.

The STR-IC-LCS problem is another interesting GC-LCS, which is very similar to the STR-EC-LCS problem.

The STR-IC-LCS problem, introduced in [7], is to find an LCS of two main sequences, in which a constraining sequence of length r must be included as its substring. In [7], an $O(nmr)$ -time algorithm was presented to solve this problem. Almost immediately, the presented algorithm was improved to a quadratic-time algorithm and to accept many main input sequences [9,10].

It is not clear whether the same improvement can be applied to our presented $O(nmr)$ -time algorithm for the STR-EC-LCS problem to achieve a quadratic-time algorithm. We will investigate the problem further.

Acknowledgments

The authors acknowledge the financial support of the Natural Science Foundation of Fujian under Grant No. 2013J0101 and the Haixi Project of Fujian under Grant No. A099.

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

Conflict of Interest

The authors declare no conflict of interest.

References

1. Tang, C.Y.; Lu, C.L. Constrained multiple sequence alignment tool development and its application to RNase family alignment. *J. Bioinform. Comput. Biol.* **2003**, *1*, 267–287.
2. Tsai, Y.T. The constrained longest common subsequence problem. *Inf. Process. Lett.* **2003**, *88*, 173–176.
3. Deorowicz, S.; Obstoj, J. Constrained longest common subsequence computing algorithms in practice. *Comput. Inf.* **2010**, *29*, 427–445.
4. Iliopoulos, C.S.; Rahman, M.S. A new efficient algorithm for computing the longest common subsequence. *Theory Comput. Sci.* **2009**, *45*, 355–371.
5. Peng, Y.H.; Yang, C.B.; Huang, K.S.; Tseng, K.T. An algorithm and applications to sequence alignment with weighted constraints. *Int. J. Found. Comput. Sci.* **2010**, *21*, 51–59.
6. Gotthilf, Z.; Hermelin, D.; Landau, G.M.; Lewenstein, M. Restricted LCS. In Proceedings of the 17th International Conference on String Processing and Information Retrieval, SPIRE'10, Los Cabos, Mexico, 11–13 October 2010; pp. 250–257.
7. Chen, Y.C.; Chao, K.M. On the generalized constrained longest common subsequence problems. *J. Comb. Optim.* **2011**, *21*, 383–392.
8. Knuth, D.E.; Morris, J.H., Jr.; Pratt, V. Fast pattern matching in strings. *SIAM J. Comput.* **1977**, *6*, 323–350.
9. Deorowicz, S. Quadratic-time algorithm for a string constrained LCS problem. *Inf. Process. Lett.* **2012**, *112*, 423–426.
10. Tseng, C.T.; Yang, C.B.; Ann, H.Y. Efficient algorithms for the longest common subsequence problem with sequential substring constraints. *J. Complex.* **2013**, *29*, 44–52.

© 2013 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>).