Information Processing Letters 113 (2013) 723–728

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A dynamic programming solution to a generalized LCS problem

Lei Wang [a], Xiaodong Wang [b,c,*], Yingjie Wu [c], Daxin Zhu [b]

[a] *Microsoft AdCenter, Bellevue, WA 98004, USA*
[b] *Faculty of Mathematics & Computer Science, Quanzhou Normal University, China*
[c] *College of Mathematics & Computer Science, Fuzhou University, Fuzhou, China*

ARTICLE INFO

Article history:
Received 30 January 2013
Accepted 4 July 2013
Available online 10 July 2013
Communicated by Jinhui Xu

Keywords:
Design of algorithms
Longest common subsequence
Generalized LCS
Dynamic programming

ABSTRACT

In this paper, we consider a generalized longest common subsequence problem, the string-excluding constrained LCS problem. For the two input sequences X and Y of lengths n and m, and a constraint string P of length r, the problem is to find a common subsequence Z of X and Y excluding P as a substring and the length of Z is maximized. The problem and its solution were first proposed by Chen and Chao (2011) [1], 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. The correctness of the new algorithm is proved. The time complexity of the new algorithm is $O(nmr)$.

© 2013 Elsevier B.V. All rights reserved.

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. It can be widely applied in diverse areas, such as file comparison, pattern matching and computational biology [2,5,6,8,9].

A sequence is a string of characters over an alphabet Σ. A subsequence of a sequence X is obtained by deleting zero or more characters from X (not necessarily contiguous). 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 ith character of X is denoted as $x_i \in \Sigma$ for any $i = 1, \ldots, n$. 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 $X[i:j] = x_ix_{i+1}\cdots x_j$ is called a prefix or a suffix of X if $i = 1$ or $j = n$, respectively.

Given two sequences X and Y, the longest common subsequence (LCS) problem is to find 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 kinds of variants of the LCS problem are called the constrained LCS (CLCS) problem [10]. Recently, Chen and Chao [1] proposed the more generalized forms of the CLCS problem, the generalized constrained longest common subsequence (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 which are to find the LCS of X and Y including/excluding P as a subsequence/substring, respectively. The four generalized constrained LCS problems can be summarized in Table 1.

We will discuss the STR-EC-LCS problem in this paper. We have noticed that a previous proposed dynamic programming algorithm for the STR-EC-LCS problem [1] cannot correctly solve the problem. A new dynamic solution for the STR-EC-LCS problem is then presented in this paper. The correctness of the new algorithm is proved. The time complexity of the new algorithm is $O(nmr)$.

The organization of the paper is as follows.

* Corresponding author. Tel.: +86 13906930798; fax: +86 59522916878.
E-mail address: wangxiaodong@qztc.edu.cn (X. Wang).

0020-0190/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ipl.2013.07.005

Table 1
The GC-LCS problems.

Problem	Input	Output
SEQ-IC-LCS	X, Y, and P	The longest common subsequence of X and Y including P as a subsequence
STR-IC-LCS	X, Y, and P	The longest common subsequence of X and Y including P as a substring
SEQ-EC-LCS	X, Y, and P	The longest common subsequence of X and Y excluding P as a subsequence
STR-EC-LCS	X, Y, and P	The longest common subsequence of X and Y excluding P as a substring

In the following 4 sections we describe our presented dynamic programming algorithm for the STR-EC-LCS problem.

In Section 2 we review the dynamic programming algorithm for the STR-EC-LCS problem proposed by Chen and Chao [1]. We point out that their algorithm will not work for a simple counterexample. In Section 3 we give a new dynamic solution for the STR-EC-LCS problem with time complexity $O(nmr)$ in a different point of view. In Section 4 we discuss the issues to implement the algorithm efficiently. Some concluding remarks are in Section 5.

2. A proposed dynamic programming algorithm

In this section, we will focus on the STR-EC-LCS problem and its solution proposed previously by Chen and Chao [1]. As noted in Table 1, for the two input sequences X and Y of lengths n and m, and a constraint string P of length r, the STR-EC-LCS problem is to find an LCS Z of X and Y excluding P as a substring.

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.

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

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

The correctness of the recursive formula (1) was based on Theorem 3 of their paper [1] as follows.

Theorem 1. *(See Chen and Chao, 2011 [1].) Let $S_{i,j,k}$ denote the set of all LCSs of $X[1:i]$ and $Y[1:j]$ excluding $P[1:k]$ as a substring. If $Z[1:l] \in S_{i,j,k}$, the following conditions hold:*

(1) *If $x_i = y_j = p_k$ and $k = 1$, then $z_l \neq x_i$ and $Z[1:l] \in S_{i-1,j-1,k}$.*
(2) *If $x_i = y_j = p_k$ and $k \geqslant 2$, then $z_l = x_i = y_j = p_k$ and $z_{l-1} = p_{k-1}$ implies $Z[1:l-1] \in S_{i-1,j-1,k-1}$.*
(3) *If $x_i = y_j = p_k$ and $k \geqslant 2$, then $z_l = x_i = y_j = p_k$ and $z_{l-1} \neq p_{k-1}$ implies $Z[1:l-1] \in S_{i-1,j-1,k}$.*

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

	$k=0$			$k=1$			$k=2$		
$i=1$	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

(4) *If $x_i = y_j = p_k$ and $k \geqslant 2$, then $z_l \neq x_i$ implies $Z[1:l] \in S_{i-1,j-1,k}$.*
(5) *If $x_i = y_j$ and $x_i \neq p_k$, then $z_l = x_i = y_j$ and $Z[1:l-1] \in S_{i-1,j-1,k}$.*
(6) *If $x_i \neq y_j$, then $z_l \neq x_i$ implies $Z[1:l] \in S_{i-1,j,k}$.*
(7) *If $x_i \neq y_j$, then $z_l \neq y_j$ implies $Z[1:l] \in S_{i,j-1,k}$.*

Since a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding $P[1:k-1]$ as a substring is also a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding $P[1:k]$ as a substring, by the definition of $L(i,j,k)$, we know that $L(i,j,k) \geqslant L(i,j,k-1)$ is always true. Therefore, the recursive formula (1) can be further reduced to the recursive formula (2).

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

Furthermore, the most important thing is that the above theorem was only stated but without a strict proof. Therefore, the correctness of the proposed algorithm cannot be guaranteed. For example, if $X = abbb$, $Y = aab$ and $P = ab$, the values of $L(i,j,k)$, $1 \leqslant i \leqslant 4$, $1 \leqslant j \leqslant 3$, $0 \leqslant k \leqslant 2$ computed by recursive formulas (1) and (2) are listed in Table 2.

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

We have tried to modify the recursive formula (1) or (2) to a correct one, but failed.

In the next section, we will investigate the problem in a different way and finally present a correct dynamic solution for the STR-EC-LCS problem.

3. Our new 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 excluding 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 function $\sigma(S)$.

The symbol $\oplus$ is also used to denote the 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, and 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 $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma(z) = k$ for each $z \in Z(i, j, k)$. The length of an LCS in $Z(i, j, k)$ is denoted as $f(i, j, k)$.

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

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

Theorem 2. *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 $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma(z) = k$ for each $z \in Z(i, j, k)$.*

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

For any $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, and $0 \leqslant k < r$, $f(i, j, k)$ can be computed by the following recursive formula (3).

$$f(i,j,k) = \begin{cases} \max\{f(i-1,j,k), f(i,j-1,k)\} \\ \quad \text{if } x_i \neq y_j, \\ \max\{f(i-1,j-1,k), \\ \qquad 1+\max_{0\leqslant t<r}\{f(i-1,j-1,t) \mid \sigma(P[1:t]\oplus x_i)=k\}\} \\ \quad \text{if } x_i = y_j. \end{cases} \tag{3}$$

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

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

First of all, we notice that for each pair (i', j'), $1 \leqslant i' \leqslant n$, $1 \leqslant j' \leqslant m$, such that $i' \leqslant i$ and $j' \leqslant j$, we have $f(i', j', k) \leqslant f(i, j, k)$, since a common subsequence z of $X[1:i']$ and $Y[1:j']$ excluding P as a substring and $\sigma(z) = k$ is also a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma(z) = k$.

(1) In the case of $x_i \neq y_j$, we have $x_i \neq z_t$ or $y_j \neq z_t$.

(1.1) If $x_i \neq z_t$, then $z = z_1, \ldots, z_t$ is a common subsequence of $X[1:i-1]$ and $Y[1:j]$ excluding P as a substring and $\sigma(z_1, \ldots, z_t) = k$, and so $f(i-1, j, k) \geqslant t$. On the other hand, $f(i-1, j, k) \leqslant f(i, j, k) = t$. Therefore, in this case we have $f(i, j, k) = f(i-1, j, k)$.

(1.2) If $y_j \neq z_t$, then we can prove similarly that in this case, $f(i, j, k) = f(i, j-1, k)$.

Combining the two subcases we conclude that in the case of $x_i \neq y_j$, we have $f(i, j, k) = \max\{f(i-1, j, k), f(i, j-1, k)\}$.

(2) In the case of $x_i = y_j$, there are also two cases to be distinguished.

(2.1) If $x_i = y_j \neq z_t$, then $z = z_1, \ldots, z_t$ is also a common subsequence of $X[1:i-1]$ and $Y[1:j-1]$ excluding P as a substring and $\sigma(z_1, \ldots, z_t) = k$, and so $f(i-1, j-1, k) \geqslant t$. On the other hand, $f(i-1, j-1, k) \leqslant f(i, j, k) = t$. Therefore, in this case we have $f(i, j, k) = f(i-1, j-1, k)$.

(2.2) If $x_i = y_j = z_t$, then $f(i, j, k) = t > 0$ and $z = z_1, \ldots, z_t$ is an LCS of $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma(z_1, \ldots, z_t) = k$, and thus $z_1, \ldots, z_{t-1}$ is a common subsequence of $X[1:i-1]$ and $Y[1:j-1]$ excluding P as a substring.

Let $\sigma(z_1, \ldots, z_{t-1}) = q$ and $f(i-1, j-1, q) = s$. Then $z_1, \ldots, z_{t-1}$ is a common subsequence of $X[1:i-1]$ and $Y[1:j-1]$ excluding P as a substring and $\sigma(z_1, \ldots, z_{t-1}) = q$. Therefore, we have

$$f(i-1, j-1, q) = s \geqslant t-1. \tag{4}$$

Let $v = v_1, \ldots, v_s \in Z(i-1, j-1, q)$ be an LCS of $X[1:i-1]$ and $Y[1:j-1]$ excluding P as a substring and $\sigma(v_1, \ldots, v_s) = q$. Then $\sigma((v_1, \ldots, v_s) \oplus x_i) = \sigma(P[1:q] \oplus x_i) = k$, and thus $(v_1, \ldots, v_s) \oplus x_i$ is a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma((v_1, \ldots, v_s) \oplus x_i) = k$.

Therefore,

$$f(i, j, k) = t \geqslant s+1. \tag{5}$$

Combining (4) and (5) we have $s = t-1$. Therefore, $z_1, \ldots, z_{t-1}$ is an LCS of $X[1:i-1]$ and $Y[1:j-1]$ excluding P as a substring and $\sigma(z_1, \ldots, z_{t-1}) = q$.

In other words,

$$f(i,j,k) \leqslant 1 + \max_{0\leqslant q<r}\bigl\{f(i-1,j-1,q) \bigm| \sigma\bigl(P[1:q]\oplus x_i\bigr)=k\bigr\}. \tag{6}$$

On the other hand, for any $0 \leqslant q < r$, if $f(i-1, j-1, q) = s$ and $\sigma(P[1:q] \oplus x_i) = k$, then for any $v = v_1, \ldots, v_s \in Z(i-1, j-1, q)$, $v \oplus x_i$ is a common subsequence of $X[1:i]$ and $Y[1:j]$ and $\sigma(v \oplus x_i) = k$. Since v excludes P as a substring and $\sigma(v \oplus x_i) = k < r$, $v \oplus x_i$ is a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding P as a substring. Furthermore, $v \oplus x_i$ is a common subsequence of $X[1:i]$ and $Y[1:j]$ excluding P as a substring and $\sigma(v \oplus x_i) = k$. Therefore, $f(i,j,k) = t \geqslant 1 + s = 1 + f(i-1, j-1, q)$, and so we conclude that,

$$f(i,j,k) \geqslant 1 + \max_{0 \leqslant q < r} \{ f(i-1, j-1, q) \mid \sigma(P[1:q] \oplus x_i) = k \}. \tag{7}$$

Combining (6) and (7) we have, in this case,

$$f(i,j,k) = 1 + \max_{0 \leqslant q < r} \{ f(i-1, j-1, q) \mid \sigma(P[1:q] \oplus x_i) = k \}. \tag{8}$$

Combining the two subcases in the case of $x_i = y_j$, we conclude that the recursive formula (3) is correct for the case $x_i = y_j$.

The proof is complete. □

4. The implementation of the algorithm

According to Theorem 2, our new algorithm for computing $f(i,j,k)$ is a standard 2-dimensional dynamic programming algorithm. By the recursive formula (3), the new dynamic programming algorithm for computing $f(i,j,k)$ can be implemented as the following Algorithm 1.

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 length r
Output: The length of an LCS of X and Y excluding P as a substring

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

To implement our new algorithm efficiently, the most important thing is to compute $\sigma(P[1:k] \oplus x_i)$ for each $0 \leqslant k < r$ and x_i, $1 \leqslant i \leqslant n$, in line 10 efficiently.

It is obvious that $\sigma(P[1:k] \oplus x_i) = k + 1$ for the case of $x_i = p_{k+1}$. It will be more complex to compute $\sigma(P[1:k] \oplus x_i)$ for the case of $x_i \neq p_{k+1}$. In this case the length of matched prefix of P has to be shortened to the largest $t < k$ such that $p_{k-t+1} \cdots p_k = p_1 \cdots p_t$ 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 KMP algorithm for solving the string matching problem [3,7].

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 $kmp(1), \ldots, kmp(6) = 0, 0, 1, 2, 3, 1$.

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

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 $k \geqslant 0$ and $p_{k+1} \neq p_i$ 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 $1 \leqslant k \leqslant r$ can be described as follows.

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

Then, we can compute an index t^* such that

$$f(i-1, j-1, t^*) = \max_{0 \leqslant t < r} \{ f(i-1, j-1, t) \mid \sigma(P[1:t] \oplus x_i) = k \}$$

in line 10 of Algorithm 1 by the following Algorithm 4.

Algorithm 4 $\max\sigma(i,j,k)$.

Input: Integers i, j, k
Output: An index t^* such that

$$f(i-1, j-1, t^*) = \max_{0 \leqslant t < r} \{ f(i-1, j-1, t) \mid \sigma(P[1:t] \oplus x_i) = k \}$$

```
1: $tmp \leftarrow -1$, $t^* \leftarrow -1$
2: for $t = 0$ to $r - 1$ do
3:   if $\sigma(t, x_i) = k$ and $f(i-1, j-1, t) > tmp$ then
4:     $tmp \leftarrow f(i-1, j-1, t)$, $t^* \leftarrow t$
5:   end if
6: end for
7: return $t^*$
```

Then the value of u in line 10 of Algorithm 1 must be

$$u = f(i-1, j-1, t^*) = f(i-1, j-1, \max\sigma(i,j,k)).$$

We can improve the efficiency of the above algorithms further in the following two points.

First, we can pre-compute a table λ of the function $\sigma(P[1:k] \oplus ch)$ for each character $ch \in \Sigma$ and $1 \leqslant k \leqslant r$ to speed up the computation of $\max\sigma(i,j,k)$.

Algorithm 5 $\lambda(1:r, ch \in \Sigma)$.

```
Input: String $P = p_1 \cdots p_r$, alphabet $\Sigma$
Output: A table $\lambda$
1: for all $a \in \Sigma$ and $a \neq p_1$ do
2:   $\lambda(0,a) \leftarrow 0$
3: end for
4: $\lambda(0,p_1) \leftarrow 1$
5: for $t = 1$ to $r-1$ do
6:   for all $a \in \Sigma$ do
7:     if $a = p_{t+1}$ then
8:       $\lambda(t,a) \leftarrow t+1$
9:     else
10:      $\lambda(t,a) \leftarrow \lambda(kmp(t),a)$
11:    end if
12:  end for
13: end for
```

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

Second, the computation of function $\max\sigma(i,j,k)$ is very time consuming and many repeated computations are overlapped in the whole **for** loop of Algorithm 1. We can amortize the computation of function $\max\sigma(i,j,k)$ to each entry of $f(i,j,k)$ in the **for** loop on variable k of Algorithm 1 and finally reduce the time costs of the whole algorithm. The modified algorithm can be described as follows.

Algorithm 6 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 length $r$
Output: The length of an LCS of $X$ and $Y$ excluding $P$ as a substring
1: for all $i, j, k$, $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant m$, and $0 \leqslant k \leqslant r$ do
2:   $f(i,0,k) \leftarrow 0$, $f(0,j,k) \leftarrow 0$ {boundary condition}
3: end for
4: for $i = 1$ to $n$ do
5:   for $j = 1$ to $m$ do
6:     for $k = 0$ to $r$ do
7:       $f(i,j,k) \leftarrow \max\{f(i-1,j,k), f(i,j-1,k)\}$
8:     end for
9:     if $x_i = y_j$ then
10:      for $k = 0$ to $r$ do
11:        $t \leftarrow \lambda(k, x_i)$
12:        $f(i,j,t) \leftarrow \max\{f(i,j,t), 1 + f(i-1,j-1,k)\}$
13:      end for
14:    end if
15:  end for
16: end for
17: return $\max_{0 \leqslant t < r}\{f(n,m,t)\}$
```

Since $\lambda(k, x_i)$ can be computed in $O(1)$ time for each x_i, $1 \leqslant i \leqslant n$ and any $0 \leqslant k < r$, the loop body of the above algorithm requires only $O(1)$ time. Therefore, our new algorithm for computing the length of an LCS of X and Y excluding P as a substring requires $O(nmr)$ time and $O(r|\Sigma|)$ preprocessing time.

If we want to get the answer LCS of X and Y excluding P as a substring, but not just its length, we can also present a simple recursive back tracing algorithm for this purpose as the following Algorithm 7.

In the end of our new algorithm, we will find an index t such that $f(n,m,t)$ gives the length of an LCS of X and Y excluding P as a substring. Then, a function call $back(n,m,t)$ will produce the answer LCS accordingly.

Algorithm 7 $back(i,j,k)$.

```
Comments: A recursive back tracing algorithm to construct the answer LCS
1: if $i = 0$ or $j = 0$ then
2:   return
3: end if
4: if $x_i = y_j$ then
5:   if $f(i,j,k) = f(i-1,j-1,k)$ then
6:     $back(i-1,j-1,k)$
7:   else
8:     $back(i-1,j-1,\max\sigma(i,j,k))$
9:     print $x_i$
10:  end if
11: else if $f(i-1,j,k) > f(i,j-1,k)$ then
12:   $back(i-1,j,k)$
13: else
14:   $back(i,j-1,k)$
15: end if
```

Since the cost of the algorithm $\max\sigma(i,j,k)$ is $O(r)$ in the worst case, the algorithm $back(i,j,k)$ will cost $O(r\max(n,m))$.

Finally we summarize our results in the following theorem.

Theorem 3. *Algorithm 6 solves STR-EC-LCS problem correctly in $O(nmr)$ time and $O(nmr)$ space, with preprocessing time $O(r|\Sigma|)$.*

5. Concluding remarks

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 generalized constrained longest common subsequence (GC-LCS) which is very similar to the STR-EC-LCS problem.

The STR-IC-LCS problem, introduced in [1], is to find an LCS of two main sequences, in which a constraining sequence of length r must be included as its substring. In [1] an $O(nmr)$-time algorithm was given for it. Almost immediately the presented algorithm was improved to a quadratic-time algorithm and furthermore to many main input sequences [4].

It is not clear that 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.

References

[1] Y.C. Chen, K.M. Chao, On the generalized constrained longest common subsequence problems, J. Comb. Optim. 21 (3) (2011) 383–392.

[2] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, A simple algorithm for the constrained sequence problems, Inform. Process. Lett. 90 (4) (2004) 175–179.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd ed., MIT Press, Cambridge, MA, 2009.

[4] S. Deorowicz, Quadratic-time algorithm for a string constrained LCS problem, Inform. Process. Lett. 112 (11) (2012) 423–426.

[5] S. Deorowicz, J. Obstoj, Constrained longest common subsequence computing algorithms in practice, Comput. Inform. 29 (3) (2010) 427–445.
[6] C.S. Iliopoulos, M.S. Rahman, A new efficient algorithm for computing the longest common subsequence, Theor. Comput. Sci. 45 (2) (2009) 355–371.
[7] D.E. Knuth, J.H. Morris Jr, V. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350.
[8] Y.H. Peng, C.B. Yang, K.S. Huang, K.T. Tseng, An algorithm and applications to sequence alignment with weighted constraints, Int. J. Found. Comput. Sci. 21 (1) (2010) 51–59.
[9] C.Y. Tang, C.L. Lu, Constrained multiple sequence alignment tool development and its application to RNase family alignment, J. Bioinform. Comput. Biol. 1 (2003) 267–287.
[10] Y.T. Tsai, The constrained longest common subsequence problem, Inform. Process. Lett. 88 (4) (2003) 173–176.