Journal of Discrete Algorithms 17 (2012) 67–73

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

The substring inclusion constraint longest common subsequence problem can be solved in quadratic time ☆

Muhammad Rashed Alam, M. Sohel Rahman *,1

AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh

ARTICLE INFO

Article history:
Received 11 November 2011
Received in revised form 16 October 2012
Accepted 30 November 2012
Available online 6 December 2012

Keywords:
Algorithms
Longest common subsequence
Dynamic programming

ABSTRACT

In this paper, we study some variants of the Constrained Longest Common Subsequence (CLCS) problem, namely, the substring inclusion CLCS (Substring-IC-CLCS) problem and a generalized version thereof. In the Substring-IC-CLCS problem, we are to find a longest common subsequence (LCS) of two given strings containing a third constraint string (given) as a substring. Previous solution to this problem runs in cubic time, i.e, $O(nmk)$ time, where n, m and k are the length of the 3 input strings. In this paper, we present simple $O(nm)$ time algorithms to solve the Substring-IC-CLCS problem. We also study the Generalized Substring-IC-LCS problem where we are given two strings of length n and m respectively and an ordered list of p strings and the goal is to find an LCS containing each of them as a substring in the order they appear in the list. We present an $O(nmp)$ algorithm for this generalized version of the problem.

1. Introduction

The longest common subsequence (LCS) problem is one of the classic and well-studied problems in computer science with extensive practical applications. The constraint LCS (CLCS) problem, where the computed LCS must contain a given third string as a subsequence, was introduced by Tsai [6] and later studied by a number of researchers [5,2,1,3]. This problem finds motivation from bioinformatics. Chen and Chao [2] recently introduced and studied several variants of the CLCS problem. In this paper, we are interested in one of the variants proposed in [2], where, given two strings $X = x_1, x_2, \dots, x_m$ and $Y = y_1, y_2, \dots, y_n$ and a third constraint string $S = s_1, s_2, \dots, s_k$, we are to find a longest common subsequence $C = c_1, c_2, \dots, c_\ell$ such that S is a *substring* of C. In what follows, we will be referring to this problem as the Substring-IC-LCS problem. Chen and Chao [2] presented an $O(nmk)$ algorithm for this problem. We on the other hand present two improved algorithms both of which run in $O(nm)$ time.

We also study the generalized version of this problem, where we are given two strings X, Y (as before) and an ordered list of constraint strings $\mathcal{S} = \{S_1, S_2, \dots, S_p\}$ (as opposed to a single constraint string) and we are to find a longest common subsequence C of X and Y containing each of the strings of $\mathcal{S}$ as a substring in the sequence they appear in the list. This problem was also handled by Chen and Chao [2] and they gave an algorithm with $O(n^2 \times \prod_{r=1}^{p} k_r)$ time where k_r is the length of the string S_r and we assume that $n > m$. We on the other hand present an $O(n^2 p)$ algorithm for the same problem.

☆ Part of this research work was carried out under the research project titled “Next Generation Algorithms on Sequences” funded by Ministry of Education, Government of the People’s Republic of Bangladesh.

* Corresponding author.
E-mail addresses: rashed.muhammad@yahoo.com (M.R. Alam), msrahman@cse.buet.ac.bd (M. Sohel Rahman).

1 Partially supported by a Commonwealth Fellowship and an ACU Titular Fellowship.

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2012.11.004

The rest of the paper is organized as follows. In Section 2, we present the preliminary concepts. Section 3 presents a simple quadratic time algorithm to solve the Substring-IC-LCS problem. In Section 4, we present our main dynamic programming algorithm to solve the same problem. In Section 5, we extend the algorithm of Section 4 to solve the generalized version of the problem. We briefly conclude in Section 6.

2. Preliminaries

Given a string, $X = x_1, x_2, \ldots, x_m$, we use x_i to denote the i-th letter in X. We use $X_{i\ldots i'}$, $i' \geqslant i$ to denote the substring $x_i x_{i+1} x_{i+2} \ldots x_{i'}$. On the other hand, $X_{i'\ldots i} = x_{i'}, x_{i'-1}, x_{i'-2}, \ldots, x_i$ denotes the reverse string of $X_{i\ldots i'}$, where $i' > i$. Given another string $S = s_1, s_2, \ldots, s_k$, we say that S *occurs* in the substring $X_{i\ldots i'}$ if and only if $X_{i\ldots i'}$ contains the string S as a sequence such that $x_i = s_1$ and $x_{i'} = s_k$. We use "." as the string concatenation operator. For example, we use $X.Y$ to denote the concatenation of X and Y. The problems we study in this paper are formally defined below.

Problem 1 *(Substring-IC-LCS Problem).* Suppose we are given two strings $X = x_1, x_2, \ldots, x_m$ and $Y = y_1, y_2, \ldots, y_n$ and a constraint string $S = s_1, s_2, \ldots, s_k$. We are to find a longest common subsequence $C = c_1, c_2, \ldots, c_\ell$ such that S is a substring of C.

Problem 2 *(Generalized Substring-IC-LCS Problem).* Suppose we are given two strings X, Y and a ordered list of constraint strings $\mathcal{S} = \langle S_1, S_2, \ldots, S_p \rangle$. We are to find a longest common subsequence C of X and Y containing each of the strings of $\mathcal{S}$ as a substring in the order it appears in the list.

Example 1. Suppose, $X = aatgcctaggc$, $Y = cgatctggac$, and $S = gtac$. Then, an LCS of X and Y is $atctggc$. And, given the constraint string S, a Substring-IC-LCS of X and Y is $C = gtac$.

Example 2. Suppose, $X = fabcfgbda$ and $Y = fabgcfbgda$. Now, consider an ordered list of two strings $\mathcal{S} = \langle abc, bda \rangle$. Then, a generalized Substring-IC-LCS of X and Y, given $\mathcal{S}$ is $C = fabcfbda$. Note that we have $C_{2\ldots4} = abc$ and $C_{6\ldots8} = bda$.

Example 3. Again, suppose $X = dabdcfedbgcef$ and $Y = dabcdegfe$. Also assume that $\mathcal{S} = \langle abc, bce \rangle$ is an ordered list of strings. Then, a generalized Substring-IC-LCS will be $C = dabcegfe$. Clearly, $C_{2\ldots4} = abc$ and $C_{3\ldots5} = bce$. Notably, we have overlapped occurrences of abc and bce in this example.

Given two strings X and Y of length m and n, respectively, for all $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$, we use $LCS[i, j]$ to denote the length of an LCS of $X_{1\ldots i}$ and $Y_{1\ldots j}$. On the other hand, $RevLCS[i, j]$ denotes the length of an LCS of $X_{i\ldots m}$ and $Y_{j\ldots n}$. Given $i < i'$ and $j < j'$, we further use $LCS(X_{i\ldots i'}, Y_{j\ldots j'})$ to denote a longest common subsequence of $X_{i\ldots i'}$ and $Y_{j\ldots j'}$ and $RevLCS(X_{i'\ldots i}, Y_{j'\ldots j})$ to denote a longest common subsequence of the reverse strings $X_{i'\ldots i}$ and $Y_{j'\ldots j}$.

Now, suppose that S occurs in X. Then, we say $S_X[i'] = i$ if and only if S occurs in $X_{i\ldots i'}$ and there exists no i_1, $i < i_1 < i'$ such that S occurs in $X_{i_1\ldots i'}$ and no i_2, $i < i_2 < i'$ such that S occurs in $X_{i\ldots i_2}$. In other words, $S_X[i']$ keeps track of the occurrences of S in X that ends at the $x_{i'}$ and starts at x_i such that no other occurrence of S starts (ends) at x_{i_1} (x_{i_2}) such that $i < i_1 < i'$ ($i < i_2 < i'$). If there is no occurrence of string S ending at Position i', then we set $S_X[i'] = 0$.

Example 4. Suppose $X = DABEABC$ and $S = ABC$. Then S occurs in the range $X_{2\ldots7}$ and $X_{5\ldots7}$. However we have $S_X[7] = 5$ (and not 2).

Now, suppose C is an LCS of X and Y such that S is a substring of C. Then we say that C is a Substring-IC-LCS of X and Y given a constraint string S. We use $StrLCS[i, j]$ to denote the length of a Substring-IC-LCS of $X_{1\ldots i}$ and $Y_{1\ldots j}$ containing the substring S.

To handle the Generalized Substring-IC-LCS Problem, we need to extend some of the above notations and definitions. In this version of the problem, we are given a constraint ordered list $\mathcal{S} = \langle S_1, S_2, \ldots, S_p \rangle$ instead of a single constraint string S. For each string S_r, $1 \leqslant r \leqslant p$ in $\mathcal{S}$, k_r denotes the length of string S_r. We extend the notion of $S_X[i']$ when we have a list $\mathcal{S}$ instead of a single constraint pattern S. In particular, we use $S_X[r, i']$ to denote $S_X[i']$ for the constraint pattern $S_r \in \mathcal{S}$.

Given two strings $A = a_1 \ldots a_{k_1}$ and $B = b_1 \ldots b_{k_2}$, we say that A and B *overlap* when we have $A_{i\ldots k_1} = B_{1\ldots j}$ for some $1 \leqslant i \leqslant k_1$ and $1 \leqslant j \leqslant k_2$. Note that the length of the overlap is $k_1 - i + 1 = j - 1 + 1$. In this case, the *merged pattern* of the above two overlapping strings is the string $a_1 a_2 \ldots a_i a_{i+1} \ldots a_{k_1} b_{j+1} \ldots b_{k_2} = a_1 a_2 \ldots a_{i-1} b_1 b_2 \ldots b_j b_{j+1} \ldots b_{k_2}$. In the context of the merged pattern of A and B above, the substring $b_{j+1} \ldots b_{k_2}$ is said to be the *non-overlapping pattern*. Note that, for this notion the order of A and B is important.

Now, in the list $\mathcal{S}$, we may have overlaps between the consecutive strings S_{r-1}, S_r, for all $2 \leqslant r \leqslant p$. We use Z_r to denote the merged pattern and NOV_r to denote the non-overlapping pattern of strings S_{r-1} and S_r, we have $NOV\ell_r$ to denote the length of NOV_r. The example below explains the above notions.

Example 5. Assume that $S_1 = ABCD$ and $S_2 = CDEFG$. Then their merged pattern $Z_2 = ABCDEFG$, non-overlapping pattern $NOV_2 = EFG$, which has length $NOV\ell_2 = 3$.

Finally, we use $NOV_X[r,i]$ to keep track the occurrence of NOV_r in X. Notably, since S_r contains NOV_r as a suffix, $S_X[r,i]$ is non-zero if, and only if, $NOV_X[r,i]$ is non-zero. Formally speaking, for $i_2 \leqslant i_1 \leqslant i$, if S_{r-1} occurs at $X_{i_2 \ldots i_1}$ and Z_r occurs at $X_{i_2 \ldots i}$, then we have $NOV_X[r,i] = i_1 + 1$; otherwise $NOV_X[r,i] = 0$.

3. A Simple algorithm for Substring-IC-LCS

If we want to include the string S to a common subsequence C of X and Y, then for some position i, we must have $C_{i \ldots i+k-1} = S$. Suppose that S occurs *only* at $X_{i \ldots i'}$ and $Y_{j \ldots j'}$. Then we can do the following to get a desired Substring-IC-LCS:

1. Compute $LCS(X_{1 \ldots i-1}, Y_{1 \ldots j-1})$. Let the length of the computed LCS is ℓ_1.
2. Compute $LCS(X_{i'+1 \ldots m}, Y_{j'+1 \ldots n})$. Let the length of the computed LCS is ℓ_2.
3. Return $LCS(X_{1 \ldots i-1}, Y_{1 \ldots j-1}).S.LCS(X_{i'+1 \ldots m}, Y_{j'+1 \ldots n})$ as a Substring-IC-LCS and $\ell_1 + |S| + \ell_2$ as the length.

Now we need to consider the general case when there are more than one pair of (i,i') $((j,j'))$ such that S occurs at $X_{i \ldots i'}$ $(Y_{j \ldots j'})$. From the above idea, we get the following algorithm for computing Substring-IC-LCS for the general case as follows.

1. Compute $S_X[i]$ for $1 \leqslant i \leqslant m$.
2. Compute $S_Y[i]$ for $1 \leqslant i \leqslant n$.
3. For each pair (i,j) such that $S_X[i] \neq 0$ and $S_Y[j] \neq 0$ compute $LCS(X_{1 \ldots S_X[i]-1}, Y_{1 \ldots S_Y[j]-1})$ and $LCS(X_{i+1 \ldots m}, Y_{j+1 \ldots n})$. Suppose the length of $LCS(X_{1 \ldots S_X[i]-1}, Y_{1 \ldots S_Y[j]-1})$ is ℓ_1 and the length of $LCS(X_{i+1 \ldots m}, Y_{j+1 \ldots n})$ is ℓ_2. Return $LCS(X_{1 \ldots S_X[i]-1}, Y_{1 \ldots S_Y[i]-1}).S.LCS(X_{i+1 \ldots m}, Y_{j+1 \ldots n})$ such that $\ell_1 + \ell_2$ is maximum. Also, return $\ell_1 + |S| + \ell_2$ as the length.

Clearly, $LCS(X_{i+1 \ldots m}, Y_{j+1 \ldots n})$ is equal to $RevLCS(X_{m \ldots i+1}, Y_{n \ldots j+1})$. The algorithm is formally presented in Algorithm Find-Substring-IC-LCS. The running time analysis is simple. Let us assume w.t.l.o.g that $n > m$. Then the computation of $S_X[i]$ and $S_Y[j]$ takes $O(n^2)$ time. Computation of table $LCS[i,j]$ and $RevLCS[i,j]$ takes $O(n^2)$ time. Finally, the computation of the Substring-IC-LCS length (see Steps 37 to 46) takes $O(n^2)$ time. Thus in total, the algorithm takes $O(n^2)$ time.

Find-Substring-IC-LCS

1: **for** $i \leftarrow 1$ **to** m **do**
2: $\quad S_X[i] \leftarrow 0$
3: **end for**
4: **for** $j \leftarrow 1$ **to** n **do**
5: $\quad S_Y[j] \leftarrow 0$
6: **end for**
7: **for** $i' \leftarrow 1$ **to** m **do**
8: $\quad$ **if** $s_1 = x_{i'}$ **then**
9: $\quad\quad p \leftarrow 1$
10: $\quad\quad$ **for** $i \leftarrow i'$ **to** m **do**
11: $\quad\quad\quad$ **if** $s_p = x_i$ **then**
12: $\quad\quad\quad\quad p \leftarrow p + 1$
13: $\quad\quad\quad\quad$ **if** $p = k$ **then**
14: $\quad\quad\quad\quad\quad S_X[i] \leftarrow i'$
15: $\quad\quad\quad\quad\quad$ **break**
16: $\quad\quad\quad\quad$ **end if**
17: $\quad\quad\quad$ **end if**
18: $\quad\quad$ **end for**
19: $\quad$ **end if**
20: **end for**
21: **for** $j' \leftarrow 1$ **to** n **do**
22: $\quad$ **if** $s_1 = y_{j'}$ **then**
23: $\quad\quad p \leftarrow 1$
24: $\quad\quad$ **for** $j \leftarrow j'$ **to** n **do**
25: $\quad\quad\quad$ **if** $s_p = y_j$ **then**
26: $\quad\quad\quad\quad p \leftarrow p + 1$
27: $\quad\quad\quad\quad$ **if** $p = k$ **then**
28: $\quad\quad\quad\quad\quad S_Y[j] \leftarrow j'$

```
                break
            end if
        end if
    end for
  end if
end for
compute LCS of X, Y
compute RevLCS of X, Y
for i ← 1 to m do
  for j ← 1 to n do
    if S_X[i] ≠ 0 and S_Y[j] ≠ 0 then
      d = LCS[S_X[i] − 1, S_Y[j] − 1] + k + RevLCS[i + 1, j + 1]
      if maxStrLCS > d then
        maxStrLCS = d
      end if
    end if
  end for
end for
```

Very recently, we came across a paper [4], that also solves the Substring-IC-LCS problem in $O(n^2)$ time. The algorithm of [4] is almost identical to our above algorithm. The only notable difference between the two algorithms is that while computing each of the occurrences of S, we keep the unique position while the algorithm in [4] keeps multiple positions which is redundant.

4. Our main result: a dynamic programming algorithm

In this section, we present a dynamic programming formulation to directly compute the Substring-IC-LCS. In a later section, we will discuss how this DP formulation can be extended to solve the generalized version of the problem. Property 1 shows the characterization of the structure of a solution to the Substring-IC-LCS problem.

Property 1. *If $C_{1\ldots\ell}$ is an LCS of $X_{1\ldots m}$ and $Y_{1\ldots n}$ including S as a substring such that $S = C_{\ell'-k+1\ldots\ell'}$ for some $k \leqslant \ell' \leqslant \ell$, then $C_{1\ldots\ell}$ is a concatenation of the following two substrings, for some $0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$:*

1. The prefix $C_{1\ldots\ell'}$: *$C_{1\ldots\ell'}$ is an LCS of $X_{1\ldots i}$ and $Y_{1\ldots j}$ including S as the suffix $C_{\ell'-k+1\ldots\ell'}$, and*
2. The suffix $C_{\ell'+1\ldots\ell}$: *$C_{\ell'+1\ldots\ell}$ is an LCS of $X_{i+1\ldots m}$ and $Y_{j+1\ldots n}$.*

A Dynamic programming formulation for the Substr-IC-LCS problem is given below.

$$LCS[i,j] = \begin{cases} 0 & \text{if } (i=0 \text{ or } j=0), \\ LCS[i-1,j-1]+1 & \text{if } (i,j>0) \text{ and } x_i = y_j, \\ \max(LCS[i-1,j], LCS[i,j-1]) & \text{if } (i,j>0) \text{ and } x_i \neq y_j. \end{cases} \tag{1}$$

$$t = \begin{cases} -\infty & \text{if } (i=0 \text{ or } j=0), \\ StrLCS[i-1,j-1]+1 & \text{if } (i,j>0) \text{ and } x_i = y_j, \\ \max(StrLCS[i-1,j], StrLCS[i,j-1]) & \text{if } (i,j>0) \text{ and } x_i \neq y_j. \end{cases} \tag{2}$$

$$StrLCS[i,j] = \begin{cases} t & \text{if } S_X[i]=0 \text{ or } S_Y[j]=0, \\ \max(t, LCS[S_X[i]-1, S_Y[j]-1]+k) & \text{if } S_X[i]\neq 0 \text{ and } S_Y[j]\neq 0. \end{cases} \tag{3}$$

As is evident, here, Eq. (1) basically computes the normal LCS. It is required by Eq. (3). Here, as usual, $LCS[i,j]$ stores the length of $LCS(X_{1\ldots i}, Y_{1\ldots j})$. Actual Substring-IC-LCS computation is done by Eqs. (2) and (3). Here, $StrLCS[i,j]$ stores the length of the Substring-IC-LCS of $X_{1\ldots i}$ and $Y_{1\ldots j}$. We use t for intermediate computation. Note that, if the length of the Substring-IC-LCS of $X[1\ldots i]$ and $Y[1\ldots j]$ is 0, we store $-\infty$ in $StrLCS[i,j]$.

Now, note that, when $i=0$ or $j=0$ there is no Substring-IC-LCS with respect to S. Also, if $S_X[i']=0$ or $S_X[j']=0$ for $1\leqslant i'\leqslant i$, $1\leqslant j'\leqslant j$, then we cannot have any Substring-IC-LCS with respect to S for $X[1\ldots i']$ and $Y[1\ldots j']$ for $1\leqslant i'\leqslant i$, $1\leqslant j'\leqslant j$. To handle this situation, when $i=0$ or $j=0$, we assume that the length of Substring-IC-LCS with respect to S is $-\infty$, where ∞ is conceptually a very very large value with respect to the values used in our computation. We further assume that any addition to $-\infty$ still results in $-\infty$. Now we prove the correctness of our DP formulation. We first report the following observation.

Observation 1. Suppose, $S_X[i]\neq 0$ and $S_Y[j]\neq 0$. Then the followings hold true.

1. If $S_X[i'] = 0$ or $S_Y[j'] = 0$ for all $1 \leqslant i' < i, 1 \leqslant j' < j$, then $StrLCS[i', j'] = -\infty$ and $StrLCS[i, j] = LCS[S_X[i] - 1, S_Y[j] - 1] + k$.
2. For all $i' > i$ or $j' > j$, $StrLCS[i', j'] > 0$

Lemma 1. *Eqs. (1) to (3) correctly compute Substring-IC-LCS.*

Proof. We prove the correctness based on a case by case analysis. We consider the computation of $StrLCS[i, j]$, i.e., the Substring-IC-LCS of $X[1 \ldots i]$ and $Y[1 \ldots j]$ with respect to the constraint pattern S.

Case 1: $S_X[i] = 0$ or $S_Y[j] = 0$.
From Eq. (3), it is clear that in this case, $StrLCS[i, j] = t$. We now have two subcases.

Case 1.a: *For all* $1 \leqslant i' \leqslant i, 1 \leqslant j' \leqslant j$ *we have* $S_X[i'] = 0$ *or* $S_Y[j'] = 0$.
We need to show that, in this case, $StrLCS[i, j] = -\infty$. From Eqs. (2) and (3), it is easy to see that $StrLCS[i, j]$ can get some value other than $-\infty$ if and only if the condition of $S_X[i] \neq 0$ and $S_Y[j] \neq 0$ holds in Eq. (3). Hence, clearly, $StrLCS[i, j] = -\infty$ in this case.

Case 1.b: *There exists* $i' < i, j' < j$ *such that* $S_X[i'] \neq 0$ *and* $S_Y[j'] \neq 0$.
Let us assume for some $i' < i, j' < j$, $S_X[i'] \neq 0$ and $S_Y[j'] \neq 0$ and for all $1 \leqslant i'' < i'$ and $1 \leqslant j'' < j'$, $S_X[i''] = 0$ and $S_Y[j''] = 0$. Then, by Observation 1, we have $StrLCS[i', j'] = LCS[S_X[i] - 1, S_Y[j] - 1] + k$. Hence we must have $StrLCS[i, j] > 0$. Now from Eqs. (2) and (3) it is easy to verify that $StrLCS[i, j]$ will get the correct value.

Case 2: $S_X[i] \neq 0$ *and* $S_Y[j] \neq 0$.
From Eq. (3), it is clear that in this case, $StrLCS[i, j] = \max(t, LCS[S_X[i] - 1, S_Y[j] - 1] + k)$. We now have two subcases.

Case 2.a: *For all* $1 \leqslant i' < i, 1 \leqslant j' < j$ *we have* $S_X[i'] = 0$ *or* $S_Y[j'] = 0$.
We need to show that, in this case, $t = -\infty$ and $StrLCS[i, j] = LCS[S_X[i] - 1, S_Y[j] - 1] + k$. From Eqs. (2) and (3), it is easy to see that $StrLCS[i', j']$ can get some value other than $-\infty$ if and only if the condition of $S_X[i'] \neq 0$ and $S_Y[j'] \neq 0$ holds in Eq. (3). Hence, clearly, $t = -\infty$ and $StrLCS[i, j] = LCS[S_X[i] - 1, S_Y[j] - 1] + k$.

Case 2.b: *There exists* $i' < i, j' < j$ *such that* $S_X[i'] \neq 0$ *and* $S_Y[j'] \neq 0$.
Let us assume that $S_X[i'] \neq 0$ and $S_Y[j'] \neq 0$ for some $i' < i, j' < j$. Then, by Observation 1, for all $i' \leqslant i'' < i$, $j' \leqslant j'' < j$ we have a value $StrLCS[i'', j''] > 0$. Hence from Eq. (2), $t > 0$ and from Eq. (3), $StrLCS[i, j] = \max(t, LCS[S_X[i] - 1, S_Y[j] - 1] + k)$, which is correct. □

The algorithm is formally presented in Algorithm Alt-Find-Substring-IC-CLCS. The running time analysis is simple. Let us assume w.t.l.o.g that $n > m$. Then the computation of $S_X[i]$ and $S_Y[j]$ for all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$ takes $O(n^2)$ time. Computation of the table $LCS[i, j]$ for all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$ takes $O(n^2)$ time. Finally, computation of the Substring-IC-LCS length in Steps 42 to 53 takes $O(n^2)$ time. Thus in total, the algorithm takes $O(n^2)$ time.

Alt-Find-Substring-IC-CLCS

```
1: for i ← 1 to m do
2:    S_X[i] ← 0
3: end for
4: for j ← 1 to n do
5:    S_Y[j] ← 0
6: end for
7: for i' ← 1 to m do
8:    if s_1 = x_{i'} then
9:       p ← 1
10:      for i ← i' to m do
11:         if s_p = x_i then
12:            p ← p + 1
13:            if p = k then
14:               S_X[i] ← i'
15:               break
16:            end if
17:         end if
18:      end for
19:   end if
20: end for
21: for j' ← 1 to n do
22:   if s_1 = y_{j'} then
23:      p ← 1
24:      for j ← j' to n do
25:         if s_p = y_j then
```

```
            p ← p + 1
            if p = k then
                S_Y[j] ← j'
                break
            end if
        end if
    end for
  end if
end for
compute LCS of X, Y
for i ← 1 to m do
  StrLCS[i, 0] ← −∞
end for
for j ← 1 to n do
  StrLCS[0, j] ← −∞
end for
for i ← 1 to m do
  for j ← 1 to n do
    if x_i = y_j then
      t ← StrLCS[i − 1, j − 1] + 1
    else
      t ← max(StrLCS[i − 1, j], StrLCS[i, j − 1])
    end if
    if S_X[i] ≠ 0 and S_Y[j] ≠ 0 then
      StrLCS[i, j] ← max(t, LCS[S_x[i] − 1, S_y[j] − 1] + k)
    else
      StrLCS[i, j] ← t
    end if
  end for
end for
```

5. Algorithm for generalized Substring-IC-LCS

In this section we consider the Generalized Substring-IC-LCS problem. In particular, we will extend the DP formulation of the Substring-IC-LCS problem from Section 4 to solve the generalized version of the problem. In what follows we will be using the notations $S_X[r, i]$ and Z_r extensively. Recall from Section 2 that $S_X[r, i]$ basically extends the notion of $S_X[i]$ when we are considering a list $\mathcal{S}$ of constraint pattern instead of a single one. On the other hand, Z_r denotes the *merged pattern* of strings S_{r-1} and S_r. Now we are ready to state Property 2 that shows the characterization of the structure of a solution to the Generalized Substring-IC-LCS problem.

Property 2. *If $C_{1\ldots\ell}$ is an LCS of $X_{1\ldots m}$ and $Y_{1\ldots n}$ including $\{S_1, S_2, \ldots S_r\}$ as substrings, in the given order such that $S_r = C_{\ell'-k_r+1\ldots\ell'}$ for some $k_r \leqslant \ell' \leqslant \ell$,then $C_{1\ldots\ell}$ is a concatenation of the following two substrings, for some $0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$:*

1. The prefix $C_{1\ldots\ell'}$: *$C_{1\ldots\ell'}$ is an LCS of $X_{1\ldots i}$ and $Y_{1\ldots j}$ including $\{S_1, S_2, \ldots S_{r-1}\}$ as substring and S_r as the suffix $C_{\ell'-k_r+1\ldots\ell'}$, and*
2. The suffix $C_{\ell'+1\ldots\ell}$: *$C_{\ell'+1\ldots\ell}$ is an LCS of $X_{i+1\ldots m}$ and $Y_{j+1\ldots n}$.*

Below we present a DP formulation for the Generalized Substring-IC-LCS problem. We need to consider following two cases:

Case 1: *For all $1 < r \leqslant p$ string S_r doesn't overlap with S_{r-1}.*
In this case, this problem is a simple extension of the Substring-IC-LCS problem. So we can adopt the following strategy. Let $StrLCS_0[i, j]$ corresponds to the computation of $LCS[i, j]$. Then we will compute $StrLCS_1$ considering substring S_1 using $StrLCS_0$; then we will compute $StrLCS_2$ using $StrLCS_1$ for substring S_2 and so on. Finally continuing in this way we will compute $StrLCS_r$ using $StrLCS_{r-1}$ for S_r. The dynamic programming formulation is given below.

$$StrLCS_0[i, j] = \begin{cases} 0 & \text{if } (i = 0 \text{ or } j = 0), \\ StrLCS_0[i-1, j-1] + 1 & \text{if } (i, j > 0) \text{ and } x_i = y_j, \\ \max(StrLCS_0[i-1, j], StrLCS_0[i, j-1]) & \text{if } (i, j > 0) \text{ and } x_i \neq y_j. \end{cases} \tag{4}$$

$$t = \begin{cases} -\infty & \text{if } (i = 0 \text{ or } j = 0), \\ StrLCS_r[i-1, j-1] + 1 & \text{if } (i, j > 0) \text{ and } x_i = y_j, \\ \max(StrLCS_r[i-1, j], StrLCS_r[i, j-1]) & \text{if } (i, j > 0) \text{ and } x_i \neq y_j. \end{cases} \tag{5}$$

$$StrLCS_r[i,j] = \begin{cases} t & \text{if } S_X[r,i] = 0 \text{ or } S_Y[r,j] = 0, \\ \max(t, StrLCS_{r-1}[S_X[r,i]-1, S_Y[r,j]-1] + k_r) & \text{if } S_X[r,i] \neq 0 \text{ and } S_Y[r,j] \neq 0. \end{cases} \tag{6}$$

The followings are the boundary conditions: $StrLCS_0[i,0] = StrLCS_0[0,j] = 0$ and $StrLCS_r[i,0] = StrLCS_r[0,j] = -\infty$. In Eq. (5), t is used for intermediate computation.

Case 2: *For some* $1 < r \leqslant p$ *string* S_r *overlaps with* S_{r-1}.

If for some $1 < r \leqslant p$, S_{r-1} and S_r overlaps then we will compute $NOV\ell_r$, $NOV_X[r,i]$ and $NOV_Y[r,j]$ for all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. For all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$, while computing Substring-IC-LCS for string S_r, we need to consider the merged pattern Z_r. The complete dynamic programming formulation for the problem is given below.

$$StrLCS_0[i,j] = \begin{cases} 0 & \text{if } (i = 0 \text{ or } j = 0), \\ StrLCS_0[i-1,j-1]+1 & \text{if } (i,j > 0) \text{ and } x_i = y_j, \\ \max(StrLCS_0[i-1,j], StrLCS_0[i,j-1]) & \text{if } (i,j > 0) \text{ and } x_i \neq y_j. \end{cases} \tag{7}$$

$$t = \begin{cases} -\infty & \text{if } (i = 0 \text{ or } j = 0), \\ StrLCS_r[i-1,j-1]+1 & \text{if } (i,j > 0) \text{ and } x_i = y_j, \\ \max(StrLCS_r[i-1,j], StrLCS_r[i,j-1]) & \text{if } (i,j > 0) \text{ and } x_i \neq y_j. \end{cases} \tag{8}$$

$$t' = \begin{cases} t & \text{if } NOV_X[r,i] = 0 \text{ or } NOV_Y[r,j] = 0, \\ \max(t, StrLCS_{r-1}[NOV_X[r,i]-1, NOV_Y[r,j]-1] + NOV\ell_r) & \text{if } NOV_X[r,i] \neq 0 \text{ and } NOV_Y[r,j] \neq 0. \end{cases} \tag{9}$$

$$StrLCS_r[i,j] = \begin{cases} t' & \text{if } S_X[r,i] = 0 \text{ or } S_Y[r,j] = 0, \\ \max(t', StrLCS_{r-1}[S_X[r,i]-1, S_Y[r,j]-1] + k_r) & \text{if } S_X[r,i] \neq 0 \text{ and } S_Y[r,j] \neq 0. \end{cases} \tag{10}$$

The boundary conditions are as follows: $StrLCS_0[i,0] = StrLCS_0[0,j] = 0$ and $StrLCS_r[i,0] = StrLCS_r[0,j] = -\infty$. In Eqs. (8) and (9), t and t', respectively, are used for intermediate computation.

The analysis is simple and follows readily from previous analysis. For all $1 \leqslant r \leqslant p$, at each step, the computation of $S_X[r,i]$, $S_Y[r,j]$, $NOV_X[r,i]$, $NOV_Y[r,j]$ and $StrLCS_r[i,j]$ takes $O(n^2)$ time. Thus the computation time is $O(n^2 p)$ in total.

6. Conclusion

In this paper, we have studied some variants of the CLCS problem, namely, the Substring-IC-LCS and generalized Substring-IC-LCS problems. We have presented two $O(n^2)$ time algorithms for solving the Substring-IC-LCS problem, improving the previously known $O(n^2k)$ time algorithm. Clearly this is a significant (from cubic to quadratic) improvement. An intriguing finding is that the algorithm is completely independent of the length of the constraint string. We have also presented an algorithm for the generalized Substring-IC-LCS in $O(n^2p)$ time where we are given an ordered list of p constraint patterns and we want to find an LCS String containing all of them as substring in the given order.

References

[1] A.N. Arslan, O. Eğecioğlu, Algorithms for the constraint longest common subsequence problems, International Journal of Foundations of Computer Science 16 (6) (2005) 1099–1109.
[2] Y.C. Chen, K.M. Chao, On the generalized constraint longest common subsequence problems, Journal of Combinatorial Optimization 21 (3) (2011) 383–392.
[3] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constraint longest common subsequence problems, Information Processing Letters 90 (4) (2004) 175–179.
[4] S. Deorowicz, Quadratic-time algorithm for the string constrained LCS problem, Information Processing Letters 112 (11) (2012) 423–426.
[5] C.S. Iliopoulos, M.S. Rahman, New efficient algorithms for the LCS and constrained LCS problems, Information Processing Letters 106 (1) (2008) 13–18.
[6] Y.T. Tsai, The constraint longest common subsequence problem, Information Processing Letters 88 (4) (2003) 173–176.