A Simple Polynomial Time Algorithm for the Generalized LCS Problem with Multiple Substring Exclusive Constraints

Daxin Zhu, Lei Wang, Jun Tian* and Xiaodong Wang*

Abstract—**In this paper, we present a simple polynomial time algorithm for a generalized longest common subsequence problem with multiple substring exclusion constraints. Given two sequences X and Y of lengths m and n, respectively, and a set of constraint strings P of total length r, we are to find a common subsequence z of X and Y which excludes each of strings in P as a substring and the length of z is maximized. The problem was declared to be NP-hard [1], but we finally found that this is not true. A new polynomial time solution for this problem is presented in this paper. The correctness of the new algorithm is proved. The time complexity of our algorithm is analyzed.**

Index Terms—**generalized longest common subsequence, NP-hard problems, dynamic programming, time complexity**

I. Introduction

In this paper, we consider a generalized longest common subsequence problem with multiple substring exclusive constraints. The longest common subsequence (LCS) problem is a well-known measurement for computing the similarity of two strings, and it is crucial in various applications. In this problem, we are interested in a longest sequence which is a subsequence of both sequences. The problem is well studied and is used in many applications, like DNA and protein analysis, text information retrieval, file comparing, music information retrieval, or spelling correction.

The most referred algorithm, proposed by Wagner and Fischer [29], solves the LCS problem by using a dynamic programming algorithm in quadratic time. Other advanced algorithms were proposed in the past decades [2]–[4], [16], [17], [19], [21].

Manuscript received January 18, 2015; revised May 22, 2015.

This work was supported in part by the Natural Science Foundation of Fujian under Grant No.2013J01247, Fujian Provincial Key Laboratory of Data-Intensive Computing and Fujian University Laboratory of Intelligent Computing and Information Processing.

Daxin Zhu is with Quanzhou Normal University, Quanzhou, China.(email:dex@qztc.edu.cn)

Lei Wang is with Facebook, 1 Hacker Way, Menlo Park, CA 94052, USA.

Jun Tian is with Fujian Medical University, Fuzhou, China.

Xiaodong Wang is with Fujian University of Technology, Fuzhou, China.

*Corresponding author.

If the number of input sequences is not fixed, the problem to find the LCS of multiple sequences has been proved to be NP-hard [23]. Some approximate and heuristic algorithms were proposed for these problems [6], [25].

Therefore, the constrained LCS (CLCS) problem, a recent variant of the LCS problem which was first addressed by Tsai (2003), has received much attention.

There are also a lot of generalizations of this similarity measure. Applying the constraints to the LCS problem is meaningful for some biological applications [24]. One of the recent variants of the LCS problem, the constrained longest common subsequence (CLCS) which was first addressed by Tsai [27], has received much attention. It generalizes the LCS measure by introduction of a third sequence, which allows to extort that the obtained CLCS has some special properties [26]. For two given input sequences X and Y of lengths m and n, respectively, and a constrained sequence P of length r, the CLCS problem is to find the common subsequences Z of X and Y such that P is a subsequence of Z and the length of Z is the maximum.

The most referred algorithms were proposed independently [5], [8], which solve the CLCS problem in $O(mnr)$ time and space by using dynamic programming algorithms. Some improved algorithms have also been proposed [11], [18]. The LCS and CLCS problems on the indeterminate strings were discussed in [20]. Moreover, the problem was extended to the one with weighted constraints, a more generalized problem [24].

Recently, a new variant of the CLCS problem, the restricted LCS problem, was proposed [14], which excludes the given constraint as a subsequence of the answer. The restricted LCS problem becomes NP-hard when the number of constraints is not fixed.

Some more generalized forms of the CLCS problem, the generalized constrained longest common subsequence (GC-LCS) problems, were addressed independently by Chen and Chao [7]. 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 [7] can be summarized in Table 1.

Table I
THE GC-LCS PROBLEMS

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

For the four problems in Table 1, $O(mnr)$ time algorithms were proposed [7]. However, their algorithm for STR-EC-LCS is not correct. In a recent paper, a correct $O(mnr)$ time dynamic programming algorithm was proposed [30]. For all four variants in Table 1, $O(r(m+n)+(m+n)\log(m+n))$ time algorithms were proposed by using the finite automata [12]. Recently, a quadratic algorithm to the STR-IC-LCS problem was proposed [10], and the time complexity of [12] was pointed out not correct.

The four GC-LCS problems can be generalized further to the cases of multiple constraints. In these generalized cases, the single constrained pattern P will be generalized to a set of d constraints $P = \{P_1, \cdots, P_d\}$ of total length r, as shown in Table 2.

Table II
THE MULTIPLE-GC-LCS PROBLEMS

Problem	Output
M-SEQ-IC-LCS	The longest common subsequence of X and Y including $P_i \in P$ as a subsequence
M-STR-IC-LCS	The longest common subsequence of X and Y including $P_i \in P$ as a substring
M-SEQ-EC-LCS	The longest common subsequence of X and Y excluding $P_i \in P$ as a subsequence
M-STR-EC-LCS	The longest common subsequence of X and Y excluding $P_i \in P$ as a substring

The problem M-SEQ-IC-LCS has been proved to be NP-hard in [13]. The problem M-SEQ-EC-LCS has also been proved to be NP-hard in [14], [28]. In addition, the problems M-STR-IC-LCS and M-STR-EC-LCS were also declared to be NP-hard in [7], but without a proof. The exponential-time algorithms for solving these two problems were also presented in [7].

We will discuss the problem M-STR-EC-LCS in this paper. The failure functions in the Knuth-Morris-Pratt algorithm [22] for solving the string matching problem have been proved very helpful for solving the STR-EC-LCS problem. It has been found by Aho and Corasick [1] that the failure functions can be generalized to the case of keyword tree to speedup the exact string matching of multiple patterns. This idea can be very helpful in our dynamic programming algorithm. This is the main idea of our new algorithm. A polynomial time algorithm is presented for the M-STR-EC-LCS problem based on this observation. The time complexity of our new dynamic programming algorithm for the M-STR-EC-LCS problem is $O(nmr)$, where n and m are the lengths of the two given input strings, and r is the total length of d constraint strings. This fact proves by reduction to absurdity that the M-STR-EC-LCS problem is not NP-hard.

The organization of the paper is as follows.

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

In Section 2 the preliminary knowledge for presenting our algorithm for the M-STR-EC-LCS problem is discussed. In Section 3 we give a new dynamic programming solution for the M-STR-EC-LCS problem with time complexity $O(nmr)$, where n and m are the lengths of the two given input strings, and r is the total length of d constraint strings. In Section 4 we discuss the issues to implement the algorithm efficiently. Some concluding remarks are in Section 5.

II. PRELIMINARIES

A sequence is a string of characters over an alphabet $\sum$. 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 $x_i \in \sum$ for any $i = 1, \cdots, n$. A substring of X from position i to j can be denoted as $X[i:j] = x_ix_{i+1}\cdots x_j$. If $i \neq 1$ or $j \neq n$, then the substring $X[i:j] = x_ix_{i+1}\cdots x_j$ is called a proper substring of X. 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.

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 set of d constraints $P = \{P_1, \cdots, P_d\}$ of total length r, the multiple STR-EC-LCS problem M-STR-EC-LCS is to find an LCS of X and Y excluding each of constraint $P_i \in P$ as a substring.

The most important difference between the problems STR-EC-LCS and M-STR-EC-LCS is the number of constraints. For ease of discussion, we will make the following two assumptions on the constraint set P.

Assumption 1: There are no duplicated strings in the constraint set P.

Assumption 2: No string in the constraint set P is a proper substring of any other string in P.

Keyword tree [2], [7] is a main data structure in our dynamic programming algorithm to process the constraint set P of the M-STR-EC-LCS problem.

Definition 1: The Keyword tree for set P is a rooted directed tree T satisfying 3 conditions: 1. each edge is labeled with exactly one character; 2. any two edges out of the same node have distinct labels; and 3. every string P_i in P maps to some node v of T such that the characters on the path from the root of T to v exactly spell out P_i, and every leaf of T is mapped to some string in P.

For example, Figure 1(a) shows the keyword tree T for the constraint set $P = \{aab, aba, ba\}$, where $d = 3, r = 8$. Clearly, every node in the keyword tree corresponds to a prefix of one of the strings in set P, and every prefix of a string P_i in P maps to a distinct node in the keyword tree T. The keyword tree for set P of total length r of all strings can be easily constructed in $O(r)$ time for a constant alphabet size. Because no two edges out of any node of T are labeled with the same character, the keyword tree T can be used to search for all occurrences in a text X of strings from P.

The failure functions in the Knuth-Morris-Pratt algorithm for solving the string matching problem can be generalized to the case of keyword tree to speedup the exact string matching of multiple patterns as follows.

In order to identify the nodes of T, we assign numbers $0, 1, \cdots, t-1$ to all t nodes of T in their preorder numbering. Then, each node will be assigned an integer $i, 0 \leq i < t$, as shown in Fig.1. For each node numbered i of a keyword tree T, the concatenation of characters on the path from the root to the node i spells out a string denoted as $L(i)$. The string $L(i)$ is also called the label of the node i in the keyword tree T. For any node i of T, define $lp(i)$ to be the length of the longest proper suffix of string $L(i)$ that is a prefix of some string in T.

It can be verified readily that for each node i of T, if A is an $lp(i)$-length suffix of string $L(i)$, then there must be a unique node $pre(i)$ in T such that $L(pre(i)) = A$. If $lp(i) = 0$ then $pre(i) = 0$ is the root of T.

Definition 2: The ordered pair $(i, pre(i))$ is called a failure link.

The failure link is a direct generalization of the failure functions in the KMP algorithm. For example, in Figure 1(a), failure links are shown as pointers from every node i to node $pre(i)$ where $lp(i) > 0$. The other failure links point to the root and are not shown.

The failure links of T define actually a failure function pre for the constraint set P.

For example, for the nodes $i = 1, 2, 3, 4, 5, 6, 7$ in Fig.1, the corresponding values of failure function are $pre(i) = 0, 1, 4, 6, 7, 0, 1$, as shown in Fig.1(a).

The failure function pre is used to speedup the search for all occurrences in a text X of strings from P. As stated in [7], the failure function pre can be computed in $O(r)$ time.

In the keyword tree application in our dynamic programming algorithm, a function σ will be mentioned frequently. For a string S and a given keyword tree T, if the label $L(i)$ of a node numbered i is also a suffix of S, then the node i is called a suffix node of S in T.

Definition 3: For any string S and a given keyword tree T, the unique suffix node of S in T with maximum depth is denoted as $\sigma(S)$. That is:

$$|L(\sigma(S))| = \max_{0 \leq i < t}\{|L(i)||L(i) \texttt{ is a suffix of } S\} \tag{1}$$

where t is the number of nodes in T.

For example, if $S = aabaaabb$, then in the keyword tree T of Fig.1, the node 6 is the only suffix node of S in T, therefore $\sigma(S) = 6$.

In our keyword tree application, we are only interested in the nonleaf nodes of the tree. So, we can renumber the nodes of the tree only for nonleaf nodes, omitting the leaf nodes of the tree, as shown in Fig.1(b). After renumbering, the failure function of the tree will also be changed accordingly.

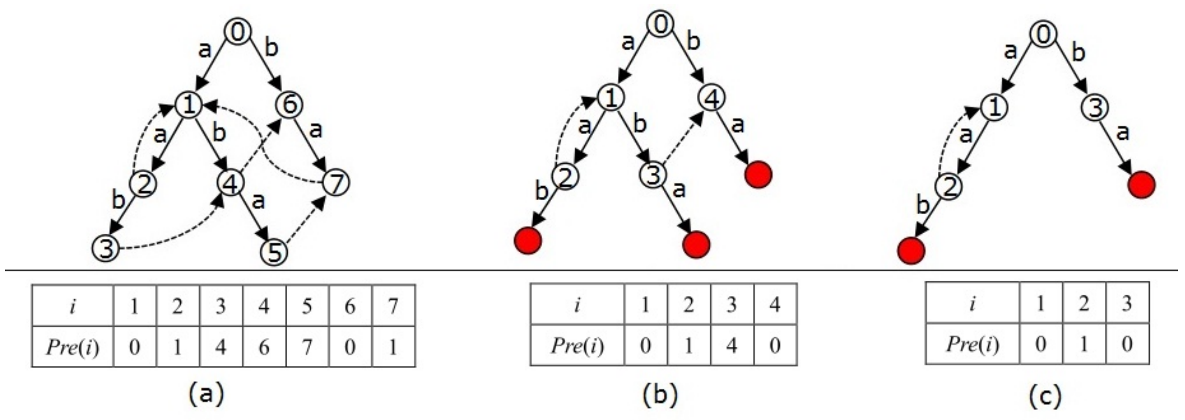
If a string P_i in the constraint set P is a proper substring of another string P_j in P, then an LCS of X and Y excluding P_i must also exclude P_j. For this reason, the constraint string P_j can be removed from constraint set P without changing the solution of the problem. For example, the string ba is a proper substring of the string aba in the keyword tree of Fig.1(b). Therefore, the string aba can be removed from the keyword tree, as shown in Fig.1(c). We will show shortly how to remove these redundant strings from constraint set P in $O(r)$ time. In the following sections, discussions are based on the Assumptions 1 and 2 on the constraint set P. The number of nonleaf nodes of the keyword tree for the constraint set P is denoted as s. In the worst case $s = r - d$. The root of the keyword tree is numbered 0, and the other nonleaf nodes are numbered $1, 2, \cdots, s-1$ in their preorder numbering. For example, in Fig.1(c), there are $s = 4$ nonleaf nodes in T. The labels for the four nonleaf nodes are $L(0) = \emptyset, L(1) = a, L(2) = aa$ and $L(3) = b$ respectively.

The symbol $\oplus$ is also used to denote the string concatenation. For example, if $S_1 = aaa$ and $S_2 = bbb$, then it is readily seen that $S_1 \oplus S_2 = aaabbb$.

III. A Simple Dynamic Programming Algorithm

In the following discussions, we will call 'a sequence excluding each of constraint string in P as a substring' a sequence excluding P for short.

Definition 4: Let $Z(i, j, k)$ denote the set of all LCSs of $X[i : n]$ and $Y[j : m]$ such that for each $z \in Z(i, j, k)$,



i	1	2	3	4	5	6	7
$Pre(i)$	0	1	4	6	7	0	1

(a)

i	1	2	3	4
$Pre(i)$	0	1	4	0

(b)

i	1	2	3
$Pre(i)$	0	1	0

(c)

Figure 1. Keyword Trees

$L(k) \oplus z$ excludes P, where $1 \leq i \leq n, 1 \leq j \leq m$, and $0 \leq k < s$. 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 $1 \leq i \leq n, 1 \leq j \leq m$, and $0 \leq k < s$ efficiently, then the length of an LCS of X and Y excluding P must be $f(1, 1, 0)$.

By using the keyword tree data structure described in the last section, we can give a recursive formula for computing $f(i, j, k)$ by 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 set of d constraints $P = \{P_1, \cdots, P_d\}$ of total length r, let $Z(i, j, k)$ and $f(i, j, k)$ be defined as in Definition 4. Suppose a keyword tree T for the constraint set P has been built, and the s nonleaf nodes of T are numbered in their preorder numbering. The label of the node numbered $k(0 \leq k < s)$ is denoted as $L(k)$. Then, for any $1 \leq i \leq n, 1 \leq j \leq m$, and $0 \leq k < s$, $f(i, j, k)$ can be computed by the following recursive formula (2).

where $q = \sigma(L(k) \oplus x_i)$, 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 $0 \leq k \leq s$.

Proof.

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

First of all, we notice 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 $f(i', j', k) \leq f(i, j, k)$, since a common subsequence z of $X[i' : n]$ and $Y[j' : m]$ satisfying $L(k) \oplus z$ excluding P is also a common subsequence of $X[i : n]$ and $Y[j : m]$ satisfying $L(k) \oplus z$ excluding P.

(1) In the case of $x_i \neq y_j$, we have $x_i \neq z_1$ or $y_j \neq z_1$.

(1.1)If $x_i \neq z_1$, then $z = z_1 \cdots z_t$ is a common subsequence of $X[i + 1 : n]$ and $Y[j : m]$ satisfying $L(k) \oplus z$ excluding P, and so $f(i + 1, j, k) \geq t$. On the other hand, $f(i + 1, j, k) \leq f(i, j, k) = t$. Therefore, in this case we have $f(i, j, k) = f(i + 1, j, k)$.

(1.2)If $y_j \neq z_1$, then 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 $x_i = y_j$ and $q < s$, there are also two subcases to be distinguished.

(2.1)If $x_i = y_j \neq z_1$, then $z = z_1 \cdots z_t$ is also a common subsequence of $X[i + 1 : n]$ and $Y[j + 1 : m]$ satisfying $L(k) \oplus z$ excluding P, and so $f(i + 1, j + 1, k) \geq t$. On the other hand, $f(i + 1, j + 1, k) \leq f(i, j, k) = t$. Therefore, in this case we have $f(i, j, k) = f(i + 1, j + 1, k)$.

(2.2)If $x_i = y_j = z_1$, then $f(i, j, k) = t > 0$ and $z = z_1 \cdots z_t$ is an LCS of $X[i : n]$ and $Y[j : m]$ satisfying $L(k) \oplus z$ excluding P, and thus $z' = z_2, \cdots, z_t$ is a common subsequence of $X[i + 1 : n]$ and $Y[j + 1 : m]$ satisfying $L(k) \oplus x_i \oplus z'$ excluding P. If $q = \sigma(L(k) \oplus x_i)$, then $L(q)$ is the longest suffix of $L(k) \oplus x_i$ that is also a label of a node of the keyword tree T, and therefore $z' = z_2, \cdots, z_t$ is also a common subsequence of $X[i+1 : n]$ and $Y[j+1 : m]$ satisfying $L(q) \oplus z'$ excluding P. In other words,

$$f(i + 1, j + 1, q) \geq t - 1 = f(i, j, k) - 1. \tag{3}$$

On the other hand, if $L(q)$ is the longest suffix of $L(k) \oplus x_i$, $f(i + 1, j + 1, q) = s$ and $v = v_1 \cdots v_s \in Z(i + 1, j + 1, q)$, then v is an LCS of $X[i + 1 : n]$ and $Y[j + 1 : m]$ satisfying $L(q) \oplus v$ excluding P. In this case $v' = x_i \oplus v$ is a common subsequence of $X[i : n]$ and $Y[j : m]$ satisfying $L(k) \oplus x_i \oplus v'$ excluding P, since $L(q)$ is the longest suffix of $L(k) \oplus x_i$ and $q < r$. Therefore,

$$f(i, j, k) \geq s + 1 = f(i + 1, j+, q) + 1. \tag{4}$$

Combining (3) and (4) we have, in this case,

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

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

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

(3) In the case of $x_i = y_j$ and $q = s$, we must have $x_i = y_j \neq z_1$, otherwise $L(k) \oplus z$ will including the string $L(k) \oplus x_i$ corresponding to a leaf node of the keyword tree T. Similar to the 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. ■

IV. The Implementation of the Algorithm

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

In Algorithm 1, s is the number of nonleaf nodes of the keyword tree T for set P. The root of the keyword tree is numbered 0, and the other nonleaf nodes are numbered $1, 2, \cdots, s-1$ in their preorder numbering. $L(t)$ is the label of node numbered t in the keyword tree T.

To implement our algorithm efficiently, the most important thing is to compte $\sigma(L(k) \oplus x_i)$ for each $0 \leq k < s$ and $x_i, 1 \leq i \leq n$, in line 9 efficiently.

It is obvious that $\sigma(L(k) \oplus x_i) = g$ if there is an edge (k, g) out of the node k labeled x_i. It will be more complex to compute $\sigma(L(k) \oplus x_i)$ if there is no edge out of the node k labeled x_i. In this case the matched node label has to be changed to the longest proper suffix of $L(k)$ that is a prefix of some string in T and the corresponding node h has an out edge (h, g) labeled x_i. Therefore, in this case, $\sigma(L(k) \oplus x_i) = g$.

This computation is very similar to the search algorithm in the keyword tree T for the multiple string matching problem [2], [7].

With pre-computed prefix function pre, the function $\sigma(L(k) \oplus ch)$ for each character $ch \in \sum$ and $1 \leq k \leq s$ can be described as the following Algorithm 2.

To speedup, we can pre-compute a table $\lambda(k, ch)$ of the function $\sigma(L(k) \oplus ch)$ for each character $ch \in \sum$ and $1 \leq k \leq s$.

When we precompute the prefix function pre, for every edge (k, g) labeled with character ch, the value of $\lambda(k, ch)$ can be assigned directly to g. The other values of the table

Algorithm 1 M-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 set of d constraints $P = \{P_1, \cdots, P_d\}$ of total length r

Output: The length of an LCS of X and Y excluding P

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

Algorithm 2 $\sigma(k, ch)$

Input: Integer k and character ch

Output: $\sigma(L(k) \oplus ch)$

```
1: while $k \geq 0$ do
2:     if there is an edge $(k, h)$ labeled $ch$ out of the node $k$ of $T$ then
3:         return $h$
4:     else
5:         $k \leftarrow pre(k)$
6:     end if
7: end while
8: return 0
```

λ can be computed by using the prefix function pre in the following recursive algorithm 3.

Algorithm 3 $\lambda(k, ch)$

Input: Integer k, character ch

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

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

The time cost of computing all values $\lambda(k, ch)$ of the table for each character $ch \in \sum$ and $1 \leq k \leq s$ by above preprocessing algorithm is obviously $O(s|\Sigma|)$. By using this pre-computed table λ, the value of function $\sigma(L(k) \oplus ch)$ for each character $ch \in \sum$ and $1 \leq k < s$ can be computed readily in $O(1)$ time.

With this pre-computed table λ, the loop body of above algorithm 1 requires only $O(1)$ time. Therefore, our dynamic programming algorithm for computing the length of an LCS of X and Y excluding P requires $O(nmr)$ time and $O(r|\Sigma|)$ preprocessing time.

Until now we have assumed that our algorithm is implemented under Assumption 1 and Assumption 2 on the constraint set P. We now describe how to relax the two assumptions.

If Assumption 1 is violated, then there must be some duplicated strings in the constraint set P. In this case, we can first sort the strings in the constraint set P, then duplicated strings can be removed from P easily and then Assumption 1 on the constraint set P is satisfied. It is clear that removed strings will not change the solution of the problem.

For Assumption 2, we first notice that a string A in the constraint set P is a proper substring of string B in P, if and only if in the keyword tree T of P, there is a directed path of failure links from a node v on the path from the root to the leaf node corresponding to string B to the leaf node corresponding to string A [7]. For example, in Fig.1(a), there is a directed path of failure links from node 5 to node 7 and thus we know the string ba corresponding to node 7 is a proper substring of string aba corresponding to node 5.

With this fact, if Assumption 2 is violated, we can remove all super-strings from the constraint set P as follows. We first build a keyword tree T for the constraint set P, then mark all nodes passed by a directed path of failure links to a leaf node in T by using a depth first traversal of T. All the strings corresponding to the marked leaf node can then be removed from P. Assumption 2 is now satisfied on the new constraint set and the keyword tree T for the new constraint set is then rebuilt. It is not difficult to do this preprocessing in $O(r)$ time. It is clear that the removed super-strings will not change the solution of the problem.

If we want to get the answer LCS of X and Y excluding P, but not just its length, we can also present a simple recursive back tracing algorithm for this purpose as the following Algorithm 4.

Algorithm 4 $back(i, j, k)$

Comments: A recursive back tracing algorithm to construct the answer LCS

```
1: if $i > n$ or $j > m$ then
2:     return
3: end if
4: if $x_i = y_j$ and $f(i, j, k) = 1 + f(i+1, j+1, \lambda(k, x_i))$ 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
```

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

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

Finally we summarize our results in the following Theorem.

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

V. Concluding Remarks

We have suggested a new dynamic programming solution for the M-STR-EC-LCS problem. The M-STR-IC-LCS problem is another interesting generalized constrained longest common subsequence (GC-LCS) which is very similar to the M-STR-EC-LCS problem. The M-STR-IC-LCS problem is to find an LCS of two main sequences, in which a set of constraint strings must be included as its substrings. It is not clear that whether the same technique of this paper can be applied to this problem to achieve an efficient algorithm. We will investigate the problem further.

References

[1] Aho A.V., Corasick M.J., Efficient string matching: an aid to bibliographic search, *Commun ACM* 18(6), 1975, pp. 333-340.

[2] Ann H.Y., Yang C.B., Tseng C.T., Hor C.Y., A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings, *Inform Process Lett* 108(11), 2008, pp.360-364.

[3] Ann H.Y., Yang C.B., Peng Y.H., Liaw B.C., Efficient algorithms for the block edit problems, *Inf Comput* 208(3),2010, pp. 221-229.

[4] Arslan A.N., Egecioglu O., Algorithms for the constrained longest common subsequence problems, *Int J Found Comput Sci* 16(6), 2005, pp. 1099-1109.

[5] Blum C., Blesa M.J., Lpez-Ibnez M., Beam search for the longest common subsequence problem, *Comput Oper Res* 36(12), 2009, pp. 3178-3186.

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

[7] Chin F.Y.L.,Santis A.D.,Ferrara A.L.,Ho N.L.,Kim S.K., A simple algorithm for the constrained sequence problems, *Inform Process Lett* 90(4), 2004, pp. 175-179.

[8] Crochemore M.,Hancart C., and Lecroq T., Algorithms on strings, Cambridge University Press, Cambridge, UK, 2007.

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

[10] Deorowicz S., Obstoj J., Constrained longest common subsequence computing algorithms in practice, *Comput Inform* 29(3), 2010, pp. 427-445.

[11] Farhana E., Ferdous J., Moosa T., Rahman M.S., Finite automata based algorithms for the generalized constrained longest common subsequence problems, In: Proceedings of the 17th international conference on string processing and information retrieval, SPIRE10, Los Cabos, Mexico, 2010, pp. 243-249.

[12] Gotthilf Z., Hermelin D., Lewenstein M., Constrained LCS: hardness and approximation. In: *Proceedings of the 19th annual symposium on combinatorial pattern matching, CPM'08*, Pisa, Italy, 2008, pp. 255-262.

[13] 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, 2010, pp. 250-257.

[14] Gusfield, D.,Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge, UK, 1997.

[15] Hirschberg D.S., Algorithms for the longest common subsequence problem, *J ACM* 24(4), 1977, pp. 664-675.

[16] Iliopoulos C.S., Rahman M.S., New efficient algorithms for the LCS and constrained LCS problems, *Inform Process Lett* 106(1), 2008, pp. 13-18.

[17] Iliopoulos C.S., Rahman M.S., A new efficient algorithm for computing the longest common subsequence, *Theor Comput Sci* 45(2), 2009, pp. 355-371.

[18] Iliopoulos C.S., Rahman M.S., Rytter W., Algorithms for two versions of LCS problem for indeterminate strings, *J Comb Math Comb Comput* 71, 2009, pp. 155-172.

[19] Iliopoulos C.S., Rahman M.S., Vorcek M., Vagner L., Finite automata based algorithms on subsequences and supersequences of degenerate strings, *J Discret Algorithm* 8(2), 2010, pp. 117-130.

[20] Knuth D.E., Morris J.H.Jr, Pratt V., Fast pattern matching in strings, *SIAM J Comput* 6(2), 1977, pp. 323-350.

[21] Maier D., The complexity of some problems on subsequences and supersequences, *J ACM* 25, 1978, pp. 322-336.

[22] 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* 21(1),2010, pp. 51-59.

[23] Shyu S.J., Tsai C.Y., Finding the longest common subsequence for multiple biological sequences by ant colony optimization, *Comput Oper Res* 36(1), 2009, pp. 73-91.

[24] Tang C.Y., Lu C.L., Constrained multiple sequence alignment tool development and its application to RNase family alignment, *J Bioinform Comput Biol* 1, 2003, pp. 267-287.

[25] Tsai Y.T., The constrained longest common subsequence problem, *Inform Process Lett* 88(4), 2003, pp. 173-176.

[26] Tseng C.T., Yang C.B., Ann H.Y., Efficient algorithms for the longest common subsequence problem with sequential substring constraints, *J Complexity* 29, 2013, pp. 44-52.

[27] Wagner R., Fischer M., The string-to-string correction problem, *J ACM* 21(1), 1974, pp. 168-173.

[28] Wang L., Wang X., Wu Y., Zhu D., A dynamic programming solution to a generalized LCS problem, *Inform Process Lett* 113(1), 2013, pp. 723-728.

[29] Yan J., Li M., and Xu J., An Adaptive Strategy Applied to Memetic Algorithms, IAENG International Journal of Computer Science, 42:2, pp73-84, 2015.

[30] Zhu D., Wang L., Tian J. and Wang X., Efficient Algorithms for a Generalized Shuffling Problem, IAENG International Journal of Computer Science, 41:4, pp237-248, 2014.