The Longest Common Subsequence with String Exclusion in Run-Length Encoded Format *

En-An Sung[a], Chang-Biau Yang[a]†and Kuo-Tsung Tseng [b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[b]Department of Shipping and Transportation Management
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

This paper studies the constrained LCS problem with excluding a constraint string as a substring in the answer, denotes as STR-EC-LCS. For the STR-EC-LCS problem with run-length encoded (RLE) strings, we are given two RLE strings X and Y of M and N, respectively, and a constraint string P of length r. We combine the state transition table and the dynamic programming approach to the solve the STR-EC-LCS problem with RLE strings. The time complexity of our algorithm is $(r(MN + \rho))$ time, where ρ denotes the numbers of cells in the bottom and right boundaries of the matched blocks. The previous algorithm for STR-EC-LCS without RLE is of time complexity $O(mnr)$, where m and n represent the lengths of the two given sequences, respectively. Clearly, $M \leq m$ and $N \leq n$. Thus, our algorithm improves the previous algorithm in time complexity.

Keywords: longest common subsequence, constrained longest common subsequence with excluding a substring, dynamic programming, state transition, run-length encoded

1 Introduction

The *longest common subsequence* (LCS) is a well-known measurement for calculating the similarity of two strings (sequences) in computer science. The LCS has been widely used in the past decades, such as article comparison, pattern matching, and alignment of of bio-sequences (DNA, RNA, proteins) [1]. One of the most well-known algorithms was proposed by Wagner and Fischer in 1974 [10]. They solved it with the dynamic programming (DP) method in O(mn) time and O(mn) space, where m and n denotes the lengths of the two input sequences.

A sequence is an ordered list of characters over an alphabet Σ. Let $X = x_1x_2 \dots x_m$ be a sequence of length m. A subsequence of X is obtained by deleting an arbitrary number of characters in X; and a substring of X, denoted as $X_{i\dots j}$, $1 \leq i \leq j \leq m$, is formed consecutively from index i to index j of X. Let $Y = y_1y_2 \dots y_n$ be a sequence of length n. Then, the LCS of the X and Y is the longest subsequence which is contained in both X and Y.

For example, suppose $X =$ `abdcac` and $Y =$ `bacdac`. Then, the LCS of X and Y is `adac`, `acac`, `bcac` or `bdac`, with LCS length 4.

The *constrainted longest common subsequence* (CLCS) problem is one of the famous variants of the LCS problem. The CLCS problem is further branched into four versions with various constraints: string inclusion (STR-IC-LCS), string exclusion (STR-EC-LCS), sequence inclusion (SEQ-IC-LCS) and sequence exclusion (SEQ-EC-LCS). The detailed definitions are given in Table 1.

Table 1: The four versions of the CLCS problem.

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

*This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 109-2221-E-110-040-MY2.

†Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

For the STR-EC-LCS problem, Chen and Chao [4] proposed an algorithm to solve it. However, their algorithm cannot correctly solve the problem, pointed out by Ann *et al.* [2]. Wang [11] employed a dynamic programming to solve the problem in O(mnr) time, where m and n denote the lengths of the two given sequences X and Y, respectively, and r denotes the length of the constraint string P. Yamada *et al.* [12] applied the diagonal algorithm, proposed by Nakatsu *et al.* [8] for solving the LCS problem, to solving the problem in O($(L+1)(m-L+1)r$) time, where L denotes the length of STR-EC-LCS(X, Y, P). Furthermore, the STR-EC-LCS problem with multiple constraint strings was proposed by Ann *et al.* [2]. They employed the KMP string matching algorithm [5, 7] to solve it in O(mnR) time, where R is the total length of all constraint strings.

The organization of this paper is given as follows. Section 2 introduces some preliminaries of the STR-EC-LCS problem. In Section 3, we introduce the dynamic programming method for solving the STR-EC-LCS problem with the state transition table. In Section 4, we propose an algorithm for solving the STR-EC-LCS problem, where the two input sequences X and Y are in the run-length encoded format, but the constraint string P is not encoded. Finally, the conclusion is given in Section 5.

2 Preliminaries

We give some preliminaries of the STR-EC-LCS problem in this section.

Run-length encoded (RLE) format is used in lossless compression It represents a string as several runs, where each run consists of several consecutive identical characters. In an RLE string, each run is represented by its character and its length. For examp le, suppose X = `aaabccccddaa`. Then, X can be encoded as $\mathtt{a}^3\mathtt{b}^1\mathtt{c}^4\mathtt{d}^2\mathtt{a}^2$ in the RLE format. And Y = `aacaddadacbc` is encoded as $\mathtt{a}^2\mathtt{c}^1\mathtt{a}^1\mathtt{d}^2\mathtt{a}^1\mathtt{d}^1\mathtt{a}^1\mathtt{c}^1\mathtt{b}^1\mathtt{c}^1$.

Given three strings X, Y and P, where $|X| = m$, $|Y| = n$ and, $|P| = r$, the STR-EC-LCS problem is to obtain an LCS of X and Y that excludes P as a substring. .

Definition 1. (Longest prefix suffix) [5] *Given two strings and $S = s_1 s_2 \dots s_t$ and $P = p_1 p_2 \dots p_r$, we denote the longest suffix of S that matches the prefix of P as $LPS(S, P) = k$. In other words, $S_{t-k+1..t} = P_{1..k}$, but $S_{t-k'+1..t} \neq P_{1..k'}$ for any $k' > k$.*

For example, suppose that S = `abbaba` and P = `babaa`. Then, $LPS(S, P) = 4$, since $S_{3..6}$ = `baba` = $P_{1..4}$.

Definition 2. (State of two strings) *Given a string $S = s_1 s_2 \dots s_t$ and a constraint string $P = p_1 p_2 \dots p_r$, if P is a substring of S, then we denote $state(S, P) = r$; otherwise, $state(S, P) = LPS(S, P)$.*

For example, suppose S_1 = `abbaba` and P = `babaa`. Then, $state(S_1, P) = LPS(S_1, P) = 4$. As another example, suppose S_2 = `abbabaac`. Then $state(S_2, P) = |P| = 5$, because P is a substring of S_2.

Definition 3. (State Transition [2, 11]) *Given two strings S and P over an alphabet set Σ, let $\delta(k, \alpha)$ denote $state(S \oplus \alpha, P)$, where $state(S, P) = k$, $\alpha \in \Sigma$, and $\oplus$ represents the concatenation of two strings.*

For given a constraint string P and an alphabet Σ, we can construct the state diagram of the automaton for representing P and Σ when another string P is input sequentially. For example, suppose that the constraint string P = `abab` and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Figure 1 demonstrates the state transition diagram of the automaton constructed from P and Σ with $|P| + 1$ states. It can also be represented as the state transition table, shown in Table 2. Also suppose S = `bbaa`. Then, we have $state(S, P) = 1$. If we append `b` to the tail of S, then we get $state(S \oplus \mathtt{b}, P) = 2$. Thus, in the table, we present this transition as $\delta(1, \mathtt{b}) = 2$.

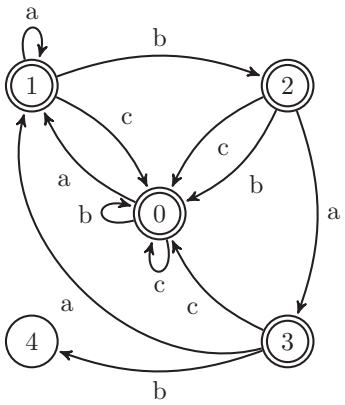


Figure 1: The state diagram of the automaton constructed from P = `abab` and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

Table 2: The state transition table of $P =$ abab and $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Here, ϵ denotes an empty string.

Input \ *State*	$0(\epsilon)$	$1(a)$	$2(ab)$	$3(aba)$
a	$1(a)$	$1(a)$	$3(aba)$	$1(a)$
b	$0(\epsilon)$	$2(ab)$	$0(\epsilon)$	$4(abab)$
c	$0(\epsilon)$	$0(\epsilon)$	$0(\epsilon)$	$0(\epsilon)$

3 A Dynamic Programming Method with the Transition Table

Given a constraint string $P = p_1p_2 \dots p_r$ and two input strings $X = x_1x_2 \dots x_m$ and $Y = y_1y_2 \dots y_n$ over an alphabet set Σ. Let $S(i, j, k)$ denote the LCS content of $X_{1\dots i}$ and $Y_{1\dots j}$ whose longest suffix matching with the prefix of P is k. That is, $state(S(i, j, k), P) = k$. Let $L(i, j, k)$ denote the length of $S(i, j, k)$.

In the preprocessing stage, we can build the state transition table for the constraint string P and the alphabet set Σ. Then, we can calculate $L(i, j, k)$ by the following DP formula with the transition table, for $1 \le i \le m$, $1 \le j \le n$ and $1 \le k \le r - 1$.

$$L(i,j,k) = \begin{cases} max\{L(i-1,j,k), & \text{if } x_i = y_j \text{ and} \\ \quad L(i,j-1,k), & \delta(k', x_i) = k, \\ \quad L(i-1,j-1,k')+1\} & \\ \\ max\{L(i-1,j,k), & \text{otherwise.} \\ \quad L(i,j-1,k)\} & \end{cases} \tag{1}$$

The boundary conditions are set as follows.
$L(i, 0, 0) = 0$, for $0 \le i \le m$;
$L(0, j, 0) = 0$, for $0 \le j \le n$;
$L(i, 0, k) = -\infty$, for $0 \le i \le m$ and $1 \le k \le r-1$;
$L(0, j, k) = -\infty$, for $0 \le j \le n$ and $1 \le k \le r-1$.

Note that $L(i, j, r) = -\infty$, since the answer cannot contain P as a substring. For example, Table 3 shows an example of $L(i, j, k)$ for solving the STR-EC-LCS problem. In the table, $L(5, 4, 1) = 2$ denotes the LCS of bbaba and abab with $state(S(5, 4, 1), P) = 1$.

We show an example to illustrate the case of $x_i = y_j$ and $\delta(k', x_i) = k$ in Equation 1. See the calculation of $L(5, 5, 1)$ in Table 3. The possible k' for $\delta(k', x_i = \mathtt{a}) = k = 1$ is 0, 1, 3, as shown in Table 2. $P =$ abab $L(5, 5, 1) = \max\{L(4, 5, 1), L(5, 4, 1), L(4, 4, 0) + 1, L(4, 4, 1) + 1, L(4, 4, 3) + 1\} = max\{3, 2, 2 + 1, 2 + 1, -\infty + 1\} = 3$.

Though the possible values of k' for computing $L(i, j, k)$ may be more than one, a specific value of k' with a specific character can be used exactly once for computing $L(i, j, k)$. Then, the amortized time for calculating each $L(i, j, k)$ is O(1). Thus, the total time complexity for solving the STR-EC-LCS problem with the transition table is O(mnr).

Table 3: An example for solving STR-EC-LCS with two input strings $X =$ bbabaa, $Y =$ abababa, and a constraint string $P =$ abab. Its state transition table is shown in Table 2 without the last row. Here, the four tables are $L(i, j, 0)$, $L(i, j, 1)$, $L(i, j, 2)$ and $L(i, j, 3)$. Each '*' means $-\infty$.

		a	b	a	b	a	b
	0	0	0	0	0	0	0
b	0	0	1	1	1	1	1
b	0	0	1	1	2	2	2
a	0	0	1	1	2	2	2
b	0	0	1	1	2	2	3
a	0	0	1	1	2	2	3
a	0	0	1	1	2	2	3

$k = 0$ (ϵ)

		a	b	a	b	a	b
	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
a	*	1	1	2	2	3	3
b	*	1	1	2	2	3	3
a	*	1	1	2	2	3	3
a	*	1	1	2	2	4	4

$k = 1$ (a)

		a	b	a	b	a	b
	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
a	*	*	*	*	*	*	*
b	*	*	2	2	3	3	4
a	*	*	2	2	3	3	4
a	*	*	2	2	3	3	4

$k = 2$ (ab)

		a	b	a	b	a	b
	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
a	*	*	*	*	*	*	*
b	*	*	*	*	*	*	*
a	*	*	*	3	3	4	4
a	*	*	*	3	3	4	4

$k = 3$ (aba)

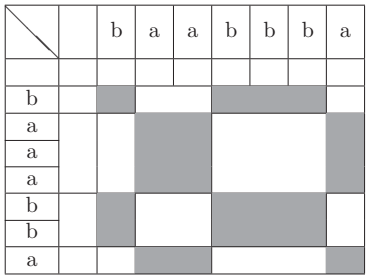
Theorem 1. *The STR-EC-LCS problem can be solved in O(mnr) time with the state transition table, where m and n denote the lengths of the two input sequences, respectively, and r denotes the length of the constraint string.*

4 The Algorithm for Run-Length Encoded Strings

Definition 4. (STR-EC-LCS with the RLE format) *For the STR-EC-LCS problem with the RLE strings, we are given two input RLE strings X, Y, and a constraint string $P = p_1p_2 \dots p_r$ without encoding. The alphabet set of X, Y and P is denoted by Σ. The problem is to find an LCS of X and Y that excludes P as a substring.*

Let $X = x_1x_2 \dots x_m$ be represented as $X = RX_1RX_2 \dots RX_M$, where each RX_i is a run with the RLE format, $|RX_i| = m_i$ denotes the length of RX_i and the character of RX_i is denoted as α_i. Similarly, $Y = y_1y_2 \dots y_n$ is encoded as $Y = RY_1RY_2 \dots RY_N$, where $|RY_i| = n_i$ and the character of RX_i is denoted as β_i. That is, there

Table 4: The gray and white blocks of the DP table with X = baaabba and Y = baabbba.



are M and N runs in X and Y, respectively. Thus, $\Sigma_{i=1}^{M} m_i = m$ and $\Sigma_{i=1}^{N} n_i = n$.

For example, suppose X = aaabccccddaa = $\mathtt{a}^3\mathtt{b}^1\mathtt{c}^4\mathtt{d}^2\mathtt{a}^2$ and Σ ={a, b, c, d}. Then, α_1 = a, α_2 = b, α_3 = c, α_4 = d, α_5 = a, $m_1 = 3$, $m_2 = 1$, $m_3 = 4$, $m_4 = 2$ and $m_5 = 2$.

Let $S'(i, j, \mu, \nu, k)$ denote the LCS content of $RX_1RX_2 \ldots RX_{i-1} \oplus \alpha_i^{\mu}$ and $RY_1RY_2 \ldots RY_{j-1} \oplus \beta_j^{\nu}$ that $state(S'(i, j, \mu, \nu, k), P) = k$. We denote the length of $S'(i, j, \mu, \nu, k)$ as $L'(i, j, \mu, \nu, k)$.

Definition 5. (Gray blocks and white blocks [3]) *Given two RLE strings $X = RX_1RX_2 \ldots RX_M$, $Y = RY_1RY_2 \ldots RY_N$, the DP table can be divided into $M \times N$ blocks. If $\alpha_i = \beta_j$, then block (i, j) is a gray block; otherwise, block (i, j) is a white block. Furthermore, there are $m_i \times n_j$ cells in block (i, j).*

Table 4 shows that the DP table is divided into grey and white blocks.

To compute the DP table efficiently, for a gray block (with $\alpha_i = \beta_j$), we calculate only the cells on the bottom boundary and the right boundary. For a white block (with $\alpha_i \neq \beta_j$), we calculate only the cell in the bottom right corner. The other cells, called *empty cells*, are not calculated. We illustrate an example in Table 6.

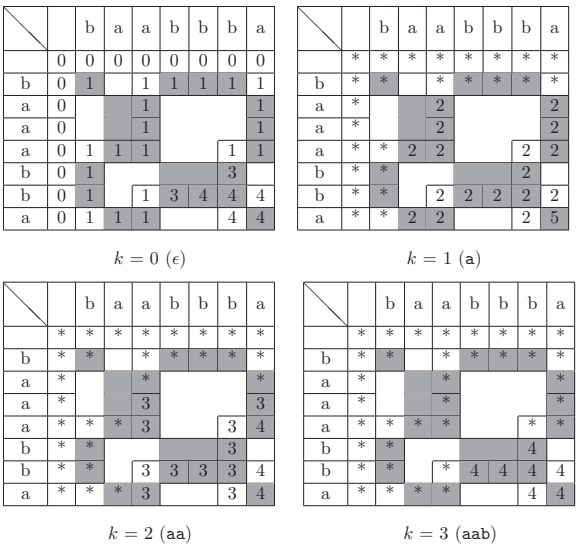
Definition 6. (State transition with the RLE format) *Given two strings S and P over an alphabet set Σ, let $\delta(k, \sigma^u)$ denote $state(S \oplus \sigma^u, P)$, where $state(S, P) = k$, $\sigma \in \Sigma$, u is the amount of character σ and $\oplus$ represents the concatenation of two strings.*

To solve the STR-EC-LCS problem with the RLE format, we can build the state transition table with the RLE format in the preprocessing stage. See the example shown in Table 5. In the table, state 4 is not acceptable, since it is to include P as substring in the answer. We have to consider the case of duplicate characters $\delta(k,\sigma^u)$

Table 5: The state transition table with two input RLE strings X = baaabba, Y = baabbba, and a constraint constraint string P = aabb.

Input \ *State*	$0(\epsilon)$	$1(a)$	$2_{(aa)}$	$3_{(aab)}$
$\mathtt{a}^1$	$1(a)$	$2(aa)$	$2(aa)$	$1(a)$
$\mathtt{a}^2$	$2(aa)$	$2(aa)$	$2(aa)$	$2(aa)$
$\mathtt{b}^1$	$0(\epsilon)$	$0(\epsilon)$	$3(aab)$	$4(aabb)$
$\mathtt{b}^2$	$0(\epsilon)$	$0(\epsilon)$	$4(aabb)$	$4(aabb)$

Table 6: An example for solving STR-EC-LCS with two input RLE strings X = baaabba, Y = baabbba, and a constraint string P = aabb without encoding.



in the transition table, for $0 \leq k \leq r-1$, $1 \leq u \leq \min\{r_1, r_2\}$ and $\sigma \in \Sigma$, where $r_1 = \max_{1 \leq i \leq M} \{m_i | \alpha_i = \sigma\}$, $r_2 = \max_{1 \leq j \leq N} \{n_j | \beta_j = \sigma\}$. For $u \geq 2$, we have $\delta(k, \sigma^u) = \delta(\delta(k, \sigma^{u-1}), \sigma)$.

If $\delta(k, \sigma^{u-1}) = \delta(k, \sigma^u)$ for $\sigma \in \Sigma$, then we do not need to continue to calculate $\delta(k, \sigma^{u'})$ for $u' > u$, because the state is convergent. That is, $\delta(k, \sigma^{u'}) = \delta(k, \sigma^u)$.

For example in Table 5, $X = \mathtt{b}^1\mathtt{a}^3\mathtt{b}^2\mathtt{a}^1$, $Y = \mathtt{b}^1\mathtt{a}^2\mathtt{b}^3\mathtt{a}^1$ and P = aabb. For the case of σ = a, we need to build $\delta(k, \mathtt{a}^1)$ and $\delta(k, \mathtt{a}^2)$.

Definition 7. ([3]) *For a cell at (i, j, μ, ν, k), let $Up(i, j, \mu, \nu, k)$ be the upper nearest cell which is at the same column, and $Left(i, j, \mu, \nu, k)$ be the left nearest cell which is not empty and at the same row.*

For example, in Table 6, we have $Up(3, 3, 2, 2, 0) = (1, 3, 1, 2, 0)$ and $Left(3, 3, 2, 2, 0) = (3, 3, 2, 1, 0)$.

$$L'(i,j,\mu,\nu,k) = \begin{cases} \max\{L'(i-1,j,m_{i-1},n_j,k), & \text{if } \alpha_i \neq \beta_j \\ \quad L'(i,j-1,m_i,n_{j-1},k),\} & \\ \max\{L'(UP(i,j,\mu,\nu,k)), & \text{if } \alpha_i = \beta_j, \\ \quad L'(Left(i,j,\mu,\nu,k)), & \ k = \delta(k',\alpha_i^{\mu}) \\ \quad L'(UP(i-1,j,m_{i-1},\nu-\mu,k'))+\mu, & \text{and } \mu < \nu \\ \quad L'(Left(i-1,j,m_{i-1},\nu-\mu,k'))+\mu & \\ \max\{L'(UP(i,j,\mu,\nu,k)), & \text{if } \alpha_i = \beta_j, \\ \quad L'(Left(i,j,\mu,\nu,k)), & \ k = \delta(k',\alpha_i^{\nu}) \\ \quad L'(UP(i,j-1,\mu-\nu,n_{j-1},k'))+\nu, & \text{and } \mu > \nu \\ \quad L'(Left(i,j-1,\mu-\nu,n_{j-1},k'))+\nu & \\ \max\{L'(UP(i,j,\mu,\nu,k)), & \text{if } \alpha_i = \beta_j, \\ \quad L'(Left(i,j,\mu,\nu,k)), & \ k = \delta(k',\alpha_i^{\mu}) \\ \quad L'(i-1,j-1,m_{i-1},n_{j-1},k')+\mu\} & \text{and } \mu = \nu \\ \max\{L'(UP(i,j,\mu,\nu,k)), & \text{otherwise.} \\ \quad L'(Left(i,j,\mu,\nu,k))\} & \end{cases} \quad (2)$$

Figure 2: The DP formula for STR-EC-LCS in the RLE format with the transition table.

The DP formula for solving the STR-EC-LCS problem with the RLE format is presented by Equation 2, as shown in Figure 2.

Table 6 shows an example for solving the STR-EC-LCS problem with Equation 2. There are five possible cases for calculating $L'(i,j,\mu,\nu,k)$, as shown as follows.

Case 1: $\alpha_i \neq \beta_j$. An example:
$L'(2,3,3,3,0) = \max\{L'(1,3,1,3,0),\ L'(2,2,3,2,0)\} = 1$.

Case 2: $\alpha_i = \beta_j$, $k = \delta(k',\alpha_i^{\mu})$ and $\mu < \nu$. An example:
$L'(2,2,1,2,1) = \max\{L'(1,2,1,2,1),\ L'(2,0,1,1,1),\ L'(0,2,1,1,0)+1,\ L'(1,1,1,1,0)+1,\ L'(0,2,1,1,3)+1,\ L'(1,1,1,1,3)+1\} = 2$.

Case 3: $\alpha_i = \beta_j$, $k = \delta(k',\alpha_i^{\nu})$ and $\mu > \nu$. An example:
$L'(3,3,2,1,3) = \max\{L'(1,3,1,1,3),\ L'(3,2,2,2,3),\ L'(2,2,3,2,2)+1,\ L'(3,1,1,1,2)+1\} = 4$.

Case 4: $\alpha_i = \beta_j$, $k = \delta(k',\alpha_i^{\mu})$ and $\mu = \nu$. An example:
$L'(3,3,2,2,0) = \max\{\ L'(1,3,1,2,0),\ L'(3,3,2,1,0),\ L'(2,2,3,2,0)+2,\ L'(2,2,3,2,1)+2\ \} = 4$.

Case 5: Otherwise (There exists no such k'.). An example:
$L'(3,3,2,3,3) = \max\{L'(3,3,1,3,3), L'(3,3,2,2,3)\} = 3$.

Our algorithm for solving the STR-EC-LCS problem with the RLE strings is formally presented in Algorithm 1. Its time complexity is $O(r(MN+\rho))$ time, where ρ denotes the number of cells on the bottom boundary and the right boundary of the matched blocks. The computation of each cell requires O(1) amortized time, and $r(MN+\rho)$ cells are needed to be calculated.

For the detailed analysis of each cell, it is obvious that each cell can be computed in O(1) time when $\alpha_i \neq \beta_j$. In the case of $\alpha_i = \beta_j$ and $p < q$, to calculate $L'(i,j,\mu,\nu,k)$, for $0 \leq k \leq r-1$, we need to get the set $S_k = \{k' | \delta(k',\alpha_i^{\mu}) = k\}$, for $0 \leq k \leq r-1$. We have $|S_0| + |S_1| + \ldots |S_{r-1}| = r$. Hence these r cells $L'(i,j,\mu,\nu,k)$ for all $0 \leq k \leq r-1$ can be computed in $O(r)$ time. Thus, the amortized time for computing each cell is O(1). Other cases can be analyzed similarly.

Theorem 2. *The STR-EC-LCS problem with RLE strings can be solved in $O(r(MN+\rho))$ time, where M and N denote the number of runs in the two input RLE strings, respectively, r denotes the length of the constraint string, ρ denotes the number of cells on the bottom boundary and the right boundary of the matched blocks.*

5 Conclusion and Discussion

In this paper, we solve the STR-EC-LCS problem with RLE strings in $O(r(MN+\rho))$ time and

Algorithm 1 Computation of the length of STR-EC-LCS(X, Y, P)

Input: Two RLE strings $X = RX_1RX_2 \dots RX_M$, $Y = RY_1RY_2 \dots RY_N$, and a constraint string $P = p_1, p_2, \dots, p_r$.
Output: Length of STR-EC-LCS(X, Y, P).

```
1: Construct the transition table δ.
2: Set the boundaries of the DP table, L'.
3: for i = 1 to M do
4:     for j = 1 to N do
5:         for k = 0 to r − 1 do
6:             if α_i ≠ β_j then                                   ▷ block(i, j) is white.
7:                 L'(i, j, m_i, n_j, k) ← max{L'(i − 1, j, m_{i−1}, n_j, k), L'(i, j − 1, m_i, n_{j−1}, k) }
8:             else
9:                 for μ = 1 to m_i do
10:                    Calculate L'(i, j, μ, n_j, k) with Equation 2.
11:                for ν = 1 to n_j do
12:                    Calculalte L'(i, j, m_i, ν, k) with Equation 2.
13: return max_{1≤k≤r−1}{L'(M, N, m_M, n_N, k)}
```

$O(r(MN + \rho))$ space. In other algorithms without RLE format [2, 11], the computation of each block(i, j) (matched or mismatched) needs $O(m_i \times n_j)$. To compute a matched block, our algorithm reduced the required time to $O(m_i + n_j)$; and our algorithm reduced the required time to $O(1)$ for a mismatched block.

Our algorithm also can be applied to other CLCS problems, such as STR-IC-LCS, SEQ-IC-LCS and SEQ-EC-LCS. For those kinds of problems, we only need to build the state transition table, similar to Definition. 2 and the examples in Tables 2 and 5. However, when the state transition method is applied to STR-IC-LCS, it is not so efficient because it has been solved with $O(mn)$ time [6, 9]. To overcome this situation, the state transition table may be built with considering a substring, not only a single character.

References

[1] I. Alsmadi and M. Nuser, "String matching evaluation methods for DNA comparison," *International Journal of Advanced Science and Technology*, Vol. 47, No. 1, pp. 13–32, 2012.

[2] H.-Y. Ann, C.-B. Yang, and C.-T. Tseng, "Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion," *Journal of Combinatorial Optimization*, Vol. 28, No. 4, pp. 800–813, 2014.

[3] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, and C.-Y. Hor, "A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings," *Information Processing Letters*, Vol. 108, No. 6, pp. 360–364, 2008.

[4] Y. C. Chen and K. M. Chao, "On the generalized constrained longest common subsequence problems," *Journal of Combinatorial Optimization*, Vol. 21, No. 3, pp. 383–392, 2011.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[6] S. Deorowicz, "Quadratic-time algorithm for a string constrained LCS problem," *Information Processing Letters*, Vol. 112, No. 11, pp. 423–426, 2012.

[7] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, Vol. 6, No. 2, pp. 323–350, 1977.

[8] N. Nakatsu, Y. Kambayashi, and S. Yajima, "A longest common subsequence algorithm suitable for similar text strings," *Acta Informatica*, Vol. 18, No. 2, pp. 171–179, 1982.

[9] C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Efficient algorithms for the longest common subsequence problem with sequential substring constraints," *Journal of Complexity*, Vol. 29, No. 1, pp. 44–52, 2013.

[10] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, Vol. 21, No. 1, pp. 168–173, 1974.

[11] L. Wang, X. Wang, Y. Wu, and D. Zhu, "A dynamic programming solution to a generalized lcs problem," *Information Processing Letters*, Vol. 113, No. 19, pp. 723 – 728, 2013.

[12] K. Yamada, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda, "Faster STR-EC-LCS computation," *Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 125–135, Limassol, Cyprus, 2020.