



Contents lists available at ScienceDirect

Information and Computationjournal homepage: www.elsevier.com/locate/inco

The generalized constrained longest common subsequence in the run-length encoded format[☆]

En-An Song^a, Chang-Biau Yang^{a,*}, 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**ARTICLE INFO****Article history:**

Received 20 February 2024

Received in revised form 22 September 2024

Accepted 10 May 2025

Available online 19 May 2025

Keywords:

Longest common subsequence

Constrained longest common subsequence with substring exclusion

Dynamic programming

State transition

Run-length encoded string

ABSTRACT

The longest common subsequence (LCS) is commonly used as a similarity measurement between two given sequences (strings). Several variants of the LCS problem have been developed. The generalized constrained LCS (CLCS) problems have four variants: substring exclusion (STR-EC-LCS), substring inclusion, subsequence exclusion and subsequence inclusion. In the STR-EC-LCS problem, a given constraint string is excluded as a substring from the answer. For the STR-EC-LCS problem with run-length encoded (RLE) strings, we are given two strings X and Y , along with a constraint string P , consisting of M , N and R runs, respectively. In their plain (non-RLE) representations, the lengths of these strings are $|X| = m$, $|Y| = n$ and $|P| = r$. We combine the state transition with the dynamic programming approach to solve the STR-EC-LCS problem with RLE strings. The time complexity of our algorithm is $O(r(Mn + mN))$. The previous algorithm for STR-EC-LCS without RLE has a time complexity of $O(mnr)$. Since $M \leq m$ and $N \leq n$, our algorithm improves the previous algorithm in time complexity. Additionally, our algorithm, which is based on state transitions, is universal and can be applied to the other three variants of the generalized constrained LCS problem with the same complexity.

© 2025 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

1. Introduction

The *longest common subsequence* (LCS) [19,20,23,34] is a well-known measure used in computer science to calculate the similarity between two strings (sequences). There are wide applications for the LCS over the past decades, such as article comparison, pattern matching, and alignment of bio-sequences (DNA, RNA, and proteins) [2]. One of the most well-known algorithms for solving the LCS problem was proposed by Wagner and Fischer in 1974 [34]. They solved it with the dynamic programming (DP) method in $O(mn)$ time and space, where m and n denote the lengths of the two input sequences.

Given two strings (sequences) $X = x_1x_2\dots x_m$ and $Y = y_1y_2\dots y_n$, the LCS [19,20,23,34] of X and Y is the longest subsequence which is contained in both X and Y , while maintaining the original order of elements.

The *constrained longest common subsequence* (CLCS) problem [13,32] is one of the variants of the LCS problem. The CLCS problem is further branched into four versions with different constraints [12]: substring inclusion (STR-IC-LCS), substring ex-

[☆] A preliminary version of this paper was presented at the 38th Workshop on Combinatorial Mathematics and Computation Theory, June 30, 2021, Taipei, Taiwan.

* Corresponding author.

E-mail address: cbyang@cse.nsysu.edu.tw (C.-B. Yang).

Table 1

The four versions of the constrained LCS problem, where X and Y are two input strings, and P represents the constraint string.

Problem	Description
STR-IC-LCS	LCS of X, Y , including P as a substring
STR-EC-LCS	LCS of X, Y , excluding P as a substring
SEQ-IC-LCS	LCS of X, Y , including P as a subsequence
SEQ-EC-LCS	LCS of X, Y , excluding P as a subsequence

Table 2

The previous studies of the generalized constrained LCS problems. m, n, r : lengths of the input sequences X, Y , and the constraint string P , respectively; r' : total length of all constraint strings; M, N, R : runs of sequences X, Y and P , respectively; L : answer length.

LCS with substring exclusion (STR-EC-LCS)

Year	Author(s)	Time	Notes
2011	Chen and Chao [12]	$O(mnr)$	DP, incorrect
2013	Wang et al. [35]	$O(mnr)$	DP
2014	Ann et al. [3]	$O(mnr')$	DP, multiple substrings Aho-Corasick automaton
2020	Yamada et al. [36]	$O((L+1)(m-L+1)r)$	DP, diagonal
2025	This paper	$O(r(Mn+mN))$	DP, RLE, state transition

LCS with substring inclusion (STR-IC-LCS)

Year	Author(s)	Time	Notes
2011	Chen and Chao [12]	$O(mnr)$	DP
2013	Tseng et al. [33]	$O(mn)$	multiple substrings
2017	Kuboi et al. [26]	$O(Mn+mN)$	DP, RLE
2024	Yonemoto et al. [37]	$O(nr/\log r + n(n-L+1))$	linear space
2025	This paper	$O(r(Mn+mN))$	DP, RLE, state transition

LCS with subsequence inclusion (SEQ-EC-LCS)

Year	Author(s)	Time	Notes
2011	Chen and Chao [12]	$O(mnr)$	DP
2014	Deorowicz & Grabowski [18]	$O(mnr/\log n)$	DP, lookup table
2025	This paper	$O(r(Mn+mN))$	DP, RLE, state transition

clusion (STR-EC-LCS), subsequence inclusion (SEQ-IC-LCS) and subsequence exclusion (SEQ-EC-LCS). The detailed definitions are given in Table 1.

Given two sequences X and Y , along with a constraint string P , the STR-EC-LCS problem aims to seek the LCS of X and Y , but excluding P as a substring. Here, we use STR-EC-LCS(X, Y, P) to denote the answer. For the STR-EC-LCS problem, Chen and Chao [12] proposed an algorithm to solve it. However, their algorithm cannot correctly solve the problem, pointed out by Ann et al. [3]. Wang et al. [35] employed a dynamic programming to solve the problem in $O(mnr)$ time, where $m = |X|$ and $n = |Y|$, and $r = |P|$. Yamada et al. [36] applied the diagonal concept, proposed by Nakatsu et al. [30] for solving the LCS problem, achieving a time complexity of $O((L+1)(m-L+1)r)$, 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. [3]. They employed the KMP string matching algorithm [25,14] to solve it in $O(mnr')$ time, where r' is the total length of all constraint strings.

The previous studies of the generalized constrained LCS problems are summarized in Tables 2 and 3.

The *run-length encoded* (RLE) format represents a string as a series of 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 example, $X = \text{baaabba}$ is encoded as $b^1a^3b^2a^1$, or simply ba^3b^2a , in the RLE format.

The RLE format is one of the common string compression methods. For example, in the secondary structure of proteins, neighboring residues often have the same structure, making RLE an effective way to represent these structures [5]. Another example is the bi-level image, such as character images. The algorithms designed for RLE tend to be more efficient when the input data is highly repetitive.

Some algorithms offer the RLE format in solving the LCS problem. Bunke and Csirik [11] introduced the concept of gray-white blocks and employed a DP algorithm to solve the LCS problem in $O(Mn+mN)$ time, where M and N denote the numbers of runs in sequences X and Y , respectively. Mitchell [29] solved the problem by reducing it to the geometric shortest path problem in $O((M+N+b)\log(M+N+b))$ time, where b denotes the amount of the matched (gray) blocks in the DP table. Apostolico et al. [7] collected the forced subpath to solve the problem in $O(MN+\log(MN))$ time. Arbell et al. [8] proposed a simple algorithm with $O(Mn+mN)$ time. Liu et al. [28] proposed a recursive formula and solved the problem row by row with $O(\min\{Mn, mN\})$ time. Ann et al. [4] employed the technique of *range minimum query* (RMQ) to solve the problem in $O(MN+\psi)$ time, where ψ denotes the number of cells on the bottom boundaries and the right boundaries of the matched (gray) blocks.

Table 3

The previous studies of the constrained LCS with subsequence inclusion [21]. w : word size of a computer; $L' = |LCS(A, B)|$; Σ : alphabet set; λ : number of match pairs in X and Y ; D : size of the domination set; others: same as those in Table 2.

LCS with subsequence inclusion (SEQ-IC-LCS)			
Year	Author(s)	Time	Notes
2003	Tsai [32]	$O(m^2n^2r)$	DP
2004	Chin et al. [13]	$O(mnr)$	DP, seq. alignment
2005	Arslan Eğecioğlu [9]	$O(mnr)$	DP, seq. alignment
2007	Deorowicz [15]	$O(r(mL' + \lambda) + n)$	match-point
2008	Iliopoulos & Rahman [24]	$O(r\lambda \log \log n + n)$	bounded heap, match-point
2010	Becerra et al. [10]	$O(L'\lambda D)$	space-efficient, match-point
2010	Deorowicz [16]	$O(mnr/ \Sigma + nr[m/w])$	bit-parallel
2010	Peng et al. [31]	$O(mnr)$	weighted seq. alignment
2012	Deorowicz & Obstoj [17]	$O(r(mL' + \lambda) + n)$	entry-exit, match-point
2012	Ann et al. [5]	$O(Mnr + mNr + mnR)$	RLE, RMQ
2017	Ho et al. [21]	$O(mnr)$	similar, small alphabet
2018	Hung et al. [22]	$O(rL(m - L))$	diagonal, next match
2025	This paper	$O(r(Mn + mN))$	DP, RLE, state transition

In this paper, we efficiently solve the STR-EC-LCS problem with input strings X and Y , as well as the constraint string P , all in the RLE format. Our algorithm combines the state transition table for duplicate characters with a DP method. The time complexity of our algorithm is $O(r(Mn + mN))$, where M and N denote the runs of X and Y , respectively, $|X| = m$, $|Y| = n$ and $|P| = r$ represent the lengths of these strings in their plain (non-RLE) form. This algorithm can also be applied to the other three variants of the CLCS problem, including STR-IC-LCS, SEQ-IC-LCS, and SEQ-EC-LCS.

The organization of this paper is as follows. Section 2 introduces the state transition table for the STR-EC-LCS problem, and discusses the associated algorithm. In Section 3, we propose an algorithm for solving the STR-EC-LCS problem by combining a state transition table with a DP approach. The two input sequences X and Y , along with the constraint string P , are all in RLE format. Section 4 demonstrates how this algorithm can be applied to other constrained LCS problems. Finally, Section 5 provides the conclusion.

2. Preliminaries

2.1. Notations

We use an upper-case letter to denote a sequence or string, such as X and Y . The elements in a given string X are represented by lower-case letters, meaning that x_i is the i th element of X , with indexing starting at 1. $i..j$ represents an index range from indices i to j . $X_{i..j}$ is a substring of X ranging from positions i to j , for $1 \leq i \leq j \leq |X|$, where $|X|$ denotes the length of X . Specifically, $X_{1..j}$ is a prefix of X .

In a run-length encoded (RLE) string, each run is represented as its character followed by its length. For example, suppose $X = aaabccccddaa$. Then, X is encoded as $a^3b^1c^4d^2a^2$ in the RLE format. For simplicity, if a character repeats only once, it can be represented by the character itself without the superscript. For example, X can be more simply represented as $a^3bc^4d^2a^2$.

2.2. The state transition

Definition 1 (Longest common prefix suffix [14]). Given two strings $S = s_1s_2\dots s_t$ and $P = p_1p_2\dots p_r$, the length of the longest suffix of S that matches the prefix of P is denoted as $LPS(S, P) = k$. In other words, $S_{t-k+1..t} = P_{1..k}$, but for any $k' \geq k + 1$, $S_{t-k'+1..t} \neq P_{1..k'}$.

Definition 2 (State of two strings). Given a string $S = s_1s_2\dots s_t$ and a constraint string $P = p_1p_2\dots p_r$, if P is a substring of S , then we denote $state(S, P) = r$; otherwise, $state(S, P) = LPS(S, P)$.

For example, suppose $A = abbaba$ and $P = babaa$. Then, $state(A, P) = LPS(A, P) = 4$, since $A_{3..6} = baba = P_{1..4}$. As another example, suppose $B = abbabaac$. Then $state(B, P) = |P| = 5$, because P is a substring of B .

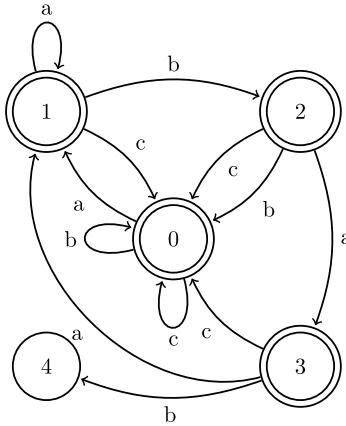
Definition 3 (State transition [3,35]). Given two strings S and P over an alphabet set Σ , let $\delta(k, \sigma)$ denote $state(S \oplus \sigma, P)$, where $k = state(S, P)$, $\sigma \in \Sigma$, and \oplus represents the concatenation of two strings.

For example, Table 4 shows an example of the transition table $\delta(\cdot)$ for an arbitrary input string S and the constraint string $P = abab$, where each character of S is in $\Sigma = \{a, b, c\}$. Here, S can be viewed as an LCS answer of two strings X and Y . The automaton of the state transition is shown in Fig. 1. If P is not allowed to be contained in an answer of a problem as a substring, then only state 4 is unacceptable, while the others are acceptable.

Table 4

The state transition table $\delta(\cdot)$ of $P = abab$ and $\Sigma = \{a, b, c\}$. Here, ϵ denotes the empty string. Note that each input element is a common character between the two input strings X and Y , which can be any character in Σ . And each state represents ϵ or a prefix of P . The transitions for state 4 are not displayed, since state 4 is considered unacceptable.

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

**Fig. 1.** The state transition diagram of the automaton constructed from $P = abab$ and $\Sigma = \{a, b, c\}$.

Theorem 1. [3] Given a string P over an alphabet set Σ , the state transition table can be constructed in $O(|P| \times |\Sigma|)$ time.

2.3. A dynamic programming method with the transition table

Given two input strings $X = x_1x_2\dots x_m$ and $Y = y_1y_2\dots y_n$ with a constraint string $P = p_1p_2\dots p_r$ over an alphabet set Σ , let $S(i, j, k)$ denote the STR-EC-LCS answer of $X_{1\dots i}$ and $Y_{1\dots j}$ whose longest suffix matching with the prefix of P has length k . In other words, $\text{state}(S(i, j, k), P) = k$. Additionally, let $L_W(i, j, k)$ denote the length of $S(i, j, k)$.

$L_W(i, j, k)$ can be calculated by the following DP formula with the state transition table, proposed by Ann et al. [3] and Wang et al. [35], for $1 \leq i \leq m$, $1 \leq j \leq n$ and $0 \leq k \leq r - 1$.

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

The boundary conditions are set as follows.

$$\begin{aligned} L_W(i, 0, 0) &= 0, \text{ for } 0 \leq i \leq m; \\ L_W(0, j, 0) &= 0, \text{ for } 0 \leq j \leq n; \\ L_W(i, 0, k) &= -\infty, \text{ for } 0 \leq i \leq m \text{ and } 1 \leq k \leq r - 1; \\ L_W(0, j, k) &= -\infty, \text{ for } 0 \leq j \leq n \text{ and } 1 \leq k \leq r - 1. \end{aligned}$$

Note that $L_W(i, j, r)$ does not need to be calculated, as its computation would imply that P is included as a substring. However, since P cannot appear as a substring in the STR-EC-LCS answer, this computation is unnecessary.

3. The constrained longest common subsequence with string exclusion problem

3.1. The state transition with RLE

For the STR-EC-LCS problem with RLE strings, we are given two input strings X and Y , along with a constraint string P , all in the RLE format. The alphabet set of X , Y and P is denoted by Σ . The task 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 in 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 $Y =$

Table 5

The gray and white blocks [11] of the DP table for solving the LCS problem with $X = \text{baaabba}$ and $Y = \text{baabbba}$. Here, i^j represents the j th element in the i th run.

		Y	0 ¹	1 ¹	2 ¹	2 ²	3 ¹	3 ²	3 ³	4 ¹
			b	a	a	b	b	b	a	
X	0 ¹									
	1 ¹	b								
2 ¹	a									
	2 ²	a								
3 ¹	a									
	3 ²	b								
4 ¹	b									
	a									

$RY_1RY_2\dots RY_N$, where $|RY_i| = n_i$ and the character of RY_i is denoted as β_i . In addition, $P = p_1p_2\dots p_r$ is represented as $P = RP_1RP_2\dots RP_r$, where $|RP_i| = r_i$ and the character of RP_i is denoted as ρ_i . That is, there are M , N and R runs in X , Y and P , respectively. Thus, $\sum_{i=1}^M m_i = m$, $\sum_{i=1}^N n_i = n$, and $\sum_{i=1}^R r_i = r$.

Definition 4 (Gray block and white block [11]). Given two RLE strings $X = RX_1RX_2\dots RX_M$, $Y = RY_1RY_2\dots RY_N$, the DP table for solving the LCS problem 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 5 illustrates an example of gray and white blocks in the DP table used for solving the LCS problem [11]. Note that two neighboring blocks cannot both be gray; otherwise, they could be merged into a larger gray block. However, two neighboring blocks can both be white, as they are formed by different characters.

Let $W(i, j, \mu, \nu, k)$ denote the STR-EC-LCS answer of $RX_1RX_2\dots RX_{i-1} \oplus \alpha_i^\mu$ and $RY_1RY_2\dots RY_{j-1} \oplus \beta_j^\nu$, $1 \leq \mu \leq m_i$ and $1 \leq \nu \leq n_j$, with the condition that $\text{state}(W(i, j, \mu, \nu, k), P) = k$. The string $W(i, j, \mu, \nu, k)$ is denoted as $RS_1RS_2\dots RS_t$, where $|RS_i| = q_i$, the character of RS_i is denoted as γ_i , and the length of $W(i, j, \mu, \nu, k)$ is denoted by $L(i, j, \mu, \nu, k)$.

To efficiently compute the DP table of STR-EC-LCS, for each gray block ($\alpha_i = \beta_j$), we only calculate the cells on the bottom boundary or the right boundary. Conversely, for each white block ($\alpha_i \neq \beta_j$), we calculate only the cell at the bottom-right corner. All other cells are not computed.

Definition 5 (empty cell and non-empty cell). In a gray block, each cell on the bottom or right boundary is a *non-empty cell*; in a white block, only the cell at the bottom-right corner is a *non-empty cell*. All other cells are referred to as *empty cells*.

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

To solve the STR-EC-LCS problem with the RLE format, we first build the state transition table with the RLE format with respect to X , Y and P in the preprocessing stage. Refer to the example shown in Table 6. In this table, state 4 is not acceptable, since it includes P as a substring in the answer.

Ann et al. [3] employed the dictionary matching algorithm (Aho-Corasick algorithm [1]) to create Aho-Corasick automaton for handling the multiple constraint strings P . They then used this automaton to build the state transition table for individual characters. And, the result in Theorem 1 describes the time complexity for a single constraint string P , specifically for the calculation of $\delta(k, \sigma)$, where $0 \leq k \leq r - 1$ and $\sigma \in \Sigma$.

In the RLE format, we further have to handle the case of duplicate characters in $\delta(k, \sigma^u)$, for $u \geq 2$. We have $\delta(k, \sigma^u) = \delta(\delta(k, \sigma^{u-1}), \sigma) = \delta(k', \sigma)$, where $k' = \delta(k, \sigma^{u-1})$. The calculation of $\delta(k', \sigma)$ is based on a single character transition, not on multiple occurrences of σ . Thus, we can apply the method proposed by Ann et al. [3] to construct the state transition table. This means that we can build $\delta(\cdot, \sigma^2), \delta(\cdot, \sigma^3), \dots, \delta(\cdot, \sigma^\eta)$ incrementally, where $\eta = \min\{\eta_1, \eta_2\}$, $\eta_1 = \max_{1 \leq i \leq M} \{m_i \mid \alpha_i = \sigma\}$, and $\eta_2 = \max_{1 \leq j \leq N} \{n_j \mid \beta_j = \sigma\}$. For each character $\sigma \in \Sigma$, the duplicate characters in a single match between X and Y are up to σ^η . Thus, calculating $\delta(k, \sigma^u)$, $1 \leq u \leq \eta$, is sufficient for dealing with the RLE matches for σ in X and Y (where $\alpha_i = \beta_j = \sigma$).

Clearly, each $\delta(k, \sigma^u)$, $u \geq 2$, can be determined in $O(1)$ time.

In Table 6, the duplicate characters of a single match in the RLE format for X and Y can include up to a^2 or b^2 . Thus, $\delta(k, a)$, $\delta(k, a^2)$, $\delta(k, b)$ and $\delta(k, b^2)$, $0 \leq k \leq r - 1$, are calculated.

Theorem 2. For two strings X and Y , along with a constraint string P , all in the RLE format and over an alphabet set Σ , the state transition table with the RLE format can be computed in $O(r(m + n))$ time and space, where $m = |X|$, $n = |Y|$, and $r = |P|$.

Table 6

The state transition table $\delta(\cdot)$ of STR-EC-LCS with two input RLE strings $X = \text{baaabba} = \text{ba}^3\text{b}^2\text{a}$, $Y = \text{baabbba} = \text{ba}^2\text{b}^3\text{a}$, and a constraint string $P = \text{aabb} = \text{a}^2\text{b}^2$. Here, ϵ represents the empty string, and only state 4 is unacceptable. Note that a single match in the RLE format for X and Y includes a , a^2 , b , and b^2 .

		State	0(ϵ)	1(a)	2(a^2)	3(a^2b)
		Input	0(ϵ)	1(a)	2(a^2)	3(a^2b)
	a	1(a)	2(a^2)	2(a^2)	1(a)	
	a^2	2(a^2)	2(a^2)	2(a^2)	2(a^2)	
	b	0(ϵ)	0(ϵ)	3(a^2b)	4(a^2b^2)	
	b^2	0(ϵ)	0(ϵ)	4(a^2b^2)	4(a^2b^2)	

Table 7

The previous and next match tables for $X = \text{baaabba} = \text{ba}^3\text{b}^2\text{a}$ in the RLE format. Here, 0 denotes no previous match, and ∞ denotes no next match.

Run	1	2	3	4
X	b	a^3	b^2	a
Previous match				
a	0	0	2	2
b	0	1	1	3
Next match				
a	2	4	4	∞
b	3	3	∞	∞

Proof. In the state transition table, by Theorem 1, the construction of all $\delta(k, \sigma)$ entries (excluding duplicate characters) can be achieved in $O(r(m+n))$ time. Then, each $\delta(k, \sigma^u)$ for $u \geq 2$ can be computed in $O(1)$ time. The size of the table can be determined as follows. The distinct number of states k is equal to r . The distinct number of σ^u is equal to η for each $\sigma \in \Sigma$, where $\eta = \min\{\eta_1, \eta_2\}$, $\eta_1 = \max_{1 \leq i \leq M} \{m_i \mid \alpha_i = \sigma\}$, and $\eta_2 = \max_{1 \leq j \leq N} \{n_j \mid \beta_j = \sigma\}$. Thus, the distinct number of all σ^u is bounded by $m+n$. Consequently, the table size is bounded by $O(r(m+n))$. Hence, the theorem holds. \square

3.2. The DP algorithm with RLE

Definition 7 (DP cell). In the DP table, for each cell on the right or bottom boundaries ($\mu = m_i$ or $\nu = n_j$), let $Up(i, j, \mu, \nu, k)$ denote the nearest upper non-empty cell in the same column, and let $Left(i, j, \mu, \nu, k)$ denote the nearest left non-empty cell in the same row.

The exact conditions for identifying the nearest upper non-empty cells are described as follows.

$$Up(i, j, \mu, \nu, k) = \begin{cases} (i-1, j, m_{i-1}, \nu, k) & \text{if } \alpha_i = \beta_j, \mu = 1, \nu = n_j; \\ (i, j, \mu-1, \nu, k) & \text{if } \alpha_i = \beta_j, 2 \leq \mu \leq m_i, \nu = n_j; \\ (i-1, j, m_{i-1}, \nu, k) & \text{if } \alpha_i \neq \beta_j \text{ and } \nu = n_j; \\ (i', j, m_{i'}, \nu, k) & \text{if } \mu = m_i, 1 \leq \nu \leq n_j - 1, i' \text{ is the run index of } X \text{ for the previous match with } \beta_j; \\ (0, j, 1, \nu, k) & \text{otherwise.} \end{cases} \quad (2)$$

The first and second conditions are for the cells on the right boundary of a gray block; the third is for the cells on the right boundary of a white block; the fourth is for the cells on the bottom boundary (excluding the bottom-right corner) of a gray or white block; the fifth is for no upper non-empty cell.

For the fourth condition, a table of the previous and next matches, which is a common technique used in solving the LCS-like problems [6,27,33], can be constructed during the preprocessing stage. As an example shown in Table 7, the previous match table records the previous occurrence position of every character at each position. The construction of the table of previous and next matches requires $O(M|\Sigma|)$ time, where M is the number of runs in the input X , and Σ denotes the alphabet set. Once constructed, this allows each calculation of the fourth condition to be completed in $O(1)$ time.

Table 8

An example for solving STR-EC-LCS with two input strings $X = \text{baaabba}$, $Y = \text{baabbba}$, and a constraint string $P = \text{aabb}$, all in the RLE format. Here, each “*” denotes $-\infty$; i^j represents the j th element in the i th run. The answer is babbb for $k = 0$, babba or baaba for $k = 1$, baaa for $k = 2$, and baab for $k = 3$. The final answer is babba or baaba , with length 5.

$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1	$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1
X	b	a	a	b	b	b	a		b	a	a	b	b	b	a		
0^1	0	0	0	0	0	0	0	0	*	*	*	*	*	*	*	*	
1^1	b	0	1		1	1	1	1	*	*	*	*	*	*	*	*	
2^1	a	0			1								1				
2^2	a	0			1								1				
2^3	a	0	1	1	1								1	1			
3^1	b	0	1										3				
3^2	b	0	1		1	3	4	4						2	2		
4^1	a	0	1	1	1								4	4			

$k = 0 (\epsilon)$

$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1	$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1
X	b	a	a	b	b	b	a		b	a	a	b	b	b	a		
0^1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
1^1	b	*	*		*	*	*	*	*	*	*	*	*	*	*	*	
2^1	a	*			*								*				
2^2	a	*			3								3				
2^3	a	*	*	*	3								3	4			
3^1	b	*	*										3				
3^2	b	*	*		3	3	3	3					3	4			
4^1	a	*	*	*	3								3	4			

$k = 1 (\text{a})$

$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1	$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1
X	b	a	a	b	b	b	a		b	a	a	b	b	b	a		
0^1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
1^1	b	*	*		*	*	*	*	*	*	*	*	*	*	*	*	
2^1	a	*			*								*				
2^2	a	*			3								*				
2^3	a	*	*	*	3								3	4			
3^1	b	*	*										3				
3^2	b	*	*		3	3	3	3					3	4			
4^1	a	*	*	*	3								3	4			

$k = 2 (a^2)$

$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1	$X \backslash Y$	0^1	1^1	2^1	2^2	3^1	3^2	3^3	4^1
X	b	a	a	b	b	b	a		b	a	a	b	b	b	a		
0^1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
1^1	b	*	*		*	*	*	*	*	*	*	*	*	*	*	*	
2^1	a	*			*								*				
2^2	a	*			3								*				
2^3	a	*	*	*	3								3	4			
3^1	b	*	*										3				
3^2	b	*	*		3	3	3	3					3	4			
4^1	a	*	*	*	3								3	4			

$k = 3 (a^2b)$

Similarly, the precise criteria for determining the nearest left non-empty cells are outlined below.

$$\text{Left}(i, j, \mu, \nu, k) = \begin{cases} (i, j - 1, \mu, n_{j-1}, k) & \text{if } \alpha_i = \beta_j, \mu = m_i, \nu = 1; \\ (i, j, \mu, \nu - 1, k) & \text{if } \alpha_i = \beta_j, \mu = m_i, 2 \leq \nu \leq n_j; \\ (i, j - 1, \mu, n_{j-1}, k) & \text{if } \alpha_i \neq \beta_j, \mu = m_i; \\ (i, j', \mu, n_{j'}, k) & \text{if } 1 \leq \mu \leq m_i - 1, \nu = n_j, j' \text{ is the run index of } Y \text{ for the previous match with } \alpha_i; \\ (i, 0, \mu, 1, k) & \text{otherwise.} \end{cases} \quad (3)$$

For example, in Table 8, we have $\text{Up}(3, 3, 2, 2, 0) = (1, 3, 1, 2, 0)$ and $\text{Left}(3, 3, 2, 2, 0) = (3, 3, 2, 1, 0)$. Each calculation of $\text{Up}(\cdot)$ or $\text{Left}(\cdot)$ can be performed in $O(1)$ time.

The DP formula for solving the STR-EC-LCS problem with the RLE format is given in Equation (4). Notably, we only calculate the cells on the bottom boundary and the right boundary of each gray block, and the bottom-right corner of each white block. The STR-EC-LCS length can be obtained by $\max_{0 \leq k \leq r-1} \{L(M, N, m_M, n_N, k)\}$.

$$L(i, j, \mu, \nu, k) = \max \begin{cases} L(\text{Up}(i, j - 1, \mu - \nu, n_{j-1}, k')) + \nu, & \text{if } \alpha_i = \beta_j, \mu > \nu, \text{ and } k = \delta(k', \alpha_i^\nu); \\ L(\text{Left}(i, j - 1, \mu - \nu, n_{j-1}, k')) + \nu & \\ L(\text{Up}(i - 1, j, m_{i-1}, \nu - \mu, k') + \mu, & \text{if } \alpha_i = \beta_j, \mu < \nu, \text{ and } k = \delta(k', \alpha_i^\mu); \\ L(\text{Left}(i - 1, j, m_{i-1}, \nu - \mu, k') + \mu & \\ L(i - 1, j - 1, m_{i-1}, n_{j-1}, k') + \mu & \text{if } \alpha_i = \beta_j, \mu = \nu, \text{ and } k = \delta(k', \alpha_i^\mu); \\ L(\text{Up}(i, j, \mu, \nu, k)), & \text{for all cases.} \\ L(\text{Left}(i, j, \mu, \nu, k)) & \end{cases} \quad (4)$$

The boundary conditions are set as follows.

$$\begin{aligned} L(i, 0, 1, 1, 0) &= 0, \text{ for } 0 \leq i \leq m; \\ L(0, j, 1, 1, 0) &= 0, \text{ for } 0 \leq j \leq n; \\ L(i, 0, 1, 1, k) &= -\infty, \text{ for } 0 \leq i \leq m \text{ and } 1 \leq k \leq r - 1; \\ L(0, j, 1, 1, k) &= -\infty, \text{ for } 0 \leq j \leq n \text{ and } 1 \leq k \leq r - 1. \end{aligned}$$

Table 9

The inverse state transition table constructed from Table 6.

Input \ State	0(ϵ)	1(a)	2(a^2)	3(a^2b)
a	0, 3	1, 2		
a^2			0, 1, 2, 3	
b	0, 1			2
b^2	0, 1			

Here, we do not need to calculate $L(i, j, \mu, \nu, k)$ for $k = r$, because it would imply the inclusion of P as a substring in the answer, which is not allowed.

To calculate $L(i, j, \mu, \nu, k)$, we need to find which k' satisfies $k = \delta(k', \alpha_i^\mu)$. In the preprocessing stage, we can construct an inverse state transition table identifying which k' transitions to k . Table 9 illustrates an example. Suppose that $\alpha_i = \sigma$. Each cell corresponding to σ^u and state k contains the inverse state transition set $S_{k,\sigma^u} = \{k' | \delta(k', \sigma^u) = k\}$. This construction can be achieved in $O(r(m + n))$ time, which is the same as the time complexity for building the state transition table.

Table 8 presents an illustrative example of solving the STR-EC-LCS problem using Equation (4). The calculation of $L(i, j, \mu, \nu, k)$ considers five possible cases, explained as follows.

- Case 1: If $\alpha_i = \beta_j$, $\mu > \nu$, and $k = \delta(k', \alpha_i^\mu)$, then $L(i, j, \mu, \nu, k) = \max\{L(Up(i, j, \mu, \nu, k)), L(Left(i, j, \mu, \nu, k)), L(Up(i, j - 1, \mu - \nu, n_{j-1}, k')) + \nu, L(Left(i, j - 1, \mu - \nu, n_{j-1}, k')) + \nu\}$. The first two terms are involved in the last condition “for all cases”. If all ν matched characters ($\mu > \nu$) are included in the solution, then these ν newly appended characters will change the state from k' (possibly more than one) to k . The last two terms deal with this subcase. Thus, we retrieve the values on the boundary of a neighboring block (which must be white) from state k' to construct the solution. If there is no such k' (the solution cannot include all ν matched characters), it will be handled in Case 5. For example, to compute $L(3, 3, 2, 1, 3)$, we first find $k' = 2$ such that $\delta(k', b) = k = 3$ in Table 9. Then, we have $L(3, 3, 2, 1, 3) = \max\{L(1, 3, 1, 1, 3), L(3, 2, 2, 2, 3), L(2, 2, 3, 2, 2) + 1, L(3, 1, 1, 1, 2) + 1\} = \max\{-\infty, -\infty, 3 + 1, -\infty + 1\} = 4$.
- Case 2: $\alpha_i = \beta_j$, $\mu < \nu$, and $k = \delta(k', \alpha_i^\mu)$. This case is similar to Case 1. For example, to compute $L(2, 2, 1, 2, 1)$, we first find $k' = 0$ or 3 such that $\delta(k', a) = k = 1$ in Table 9. Then, we have $L(2, 2, 1, 2, 1) = \max\{L(1, 2, 1, 2, 1), L(2, 0, 1, 1, 1), L(0, 2, 1, 1, 0) + 1, L(1, 1, 1, 1, 0) + 1, L(0, 2, 1, 1, 3) + 1, L(1, 1, 1, 1, 3) + 1\} = \max\{-\infty, -\infty, 0 + 1, 1 + 1, -\infty + 1, -\infty + 1\} = 2$.
- Case 3: $\alpha_i = \beta_j$, $\mu = \nu$, and $k = \delta(k', \alpha_i^\mu)$. If all μ (where $\mu = \nu$) matched characters are part of the solution, appending these μ new characters will transit the state from k' (possibly more than one) to k . Consequently, we get the value at the bottom-right corner of $block(i - 1, j - 1)$ from k' to establish the solution. In the absence of such k' (the solution cannot include all of the ν matched characters), it will be addressed in Case 5. For example, to compute $L(3, 3, 2, 2, 0)$, we first find $k' = 0$ or 1 such that $\delta(k', b^2) = k = 0$ in Table 9. Then, we have $L(3, 3, 2, 2, 0) = \max\{L(1, 3, 1, 2, 0), L(3, 3, 2, 1, 0), L(2, 2, 3, 2, 0) + 2, L(2, 2, 3, 2, 1) + 2\} = \max\{1, 3, 1 + 2, 2 + 2\} = 4$.
- Case 4: If $\alpha_i \neq \beta_j$, then $L(i, j, \mu, \nu, k) = \max\{L(Up(i, j, \mu, \nu, k)), L(Left(i, j, \mu, \nu, k))\} = \max\{L(i - 1, j, m_{i-1}, n_j, k), L(i, j - 1, m_i, n_{j-1}, k)\}$. This case is covered under the last condition “for all cases”. In this case, no new character is included in the solution since it corresponds to a white block. Thus, the transition state k remains unchanged. In addition, it selects the maximum of its upper and left non-empty cells as its length value. As an example, $L(2, 3, 3, 3, 0) = \max\{L(Up(2, 3, 3, 3, 0), L(Left(2, 3, 3, 3, 0)\} = \max\{L(1, 3, 1, 3, 0), L(2, 2, 3, 2, 0)\} = \max\{1, 1\} = 1$.
- Case 5: If $\alpha_i = \beta_j$, but no k' that $k = \delta(k', \alpha_i^\mu)$, $u = \min\{\mu, \nu\}$, then $L(i, j, \mu, \nu, k) = \max\{L(Up(i, j, \mu, \nu, k)), L(Left(i, j, \mu, \nu, k))\}$. This case is also included in the last condition “for all cases”. Though it is a gray block, we may not include all of the u matched characters, as it may violate the constraint. For example, suppose $X = aaaaa$, $Y = aaaa$ and $P = aaa$. Then a^2 is a valid solution, while a^3 is not. In the case of a^3 , we cannot find k' that $k = \delta(k', \alpha_i^\mu)$. Consequently, we discard one matched character, and this process can be done recursively in the DP formula until we find such k' . This situation will eventually be reduced to Case 1, 2, or 3 after the recursion. The discarding process can be achieved by $L(Up(i, j, \mu, \nu, k))$ when (i, j, μ, ν, k) is on the right boundary, or $L(Left(i, j, \mu, \nu, k))$ when (i, j, μ, ν, k) is on the bottom boundary. For example, to compute $L(3, 3, 2, 3, 3)$, we cannot find k' that $k = 3 = \delta(k', b^2)$ in Table 9. It means that b^2 cannot be appended to the solution. In other words, we have to discard some of b^2 . Thus, $L(3, 3, 2, 3, 3) = \max\{L(Up(3, 3, 2, 3, 3)), L(Left(3, 3, 2, 3, 3))\} = \max\{L(3, 3, 1, 3, 3), L(3, 3, 2, 2, 3)\} = \max\{4, 4\} = 4$.

Our algorithm for solving the STR-EC-LCS problem with the RLE strings is formally presented in Algorithm 1.

Theorem 3. For two given strings X and Y , along with a constraint string P , all in the RLE format, Algorithm 1 solves the STR-EC-LCS problem in $O(r(Mn + mN))$ time, where there are M, N and R runs in X, Y and P , respectively, and $|X| = m$, $|Y| = n$ and $|P| = r$.

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

Input: Two RLE strings $X = RX_1RX_2\dots RX_M$ and $Y = RY_1RY_2\dots RY_N$, along with an RLE constraint string $P = RP_1, RP_2, \dots, RP_R$.

Output: Length of STR-EC-LCS(X, Y, P).

- 1: Construct the state transition table $\delta(\cdot)$ and its inverse state transition table.
- 2: Set the boundaries of the DP table, $L(\cdot)$.
- 3: **for** $i = 1$ to M **do**
- 4: **for** $j = 1$ to N **do**
- 5: **for** $k = 0$ to $r - 1$ **do**
- 6: **if** $\alpha_i \neq \beta_j$ **then** ▷ bottom-right corner of a white block (i, j)
- 7: $L(i, j, m_i, n_j, k) \leftarrow \max\{L(Up(i, j, m_i, n_j, k)), L(Left(i, j, m_i, n_j, k))\}$
- 8: **else** ▷ block(i, j) is gray.
- 9: **for** $\mu = 1$ to m_i **do** ▷ bottom boundary
- 10: Calculate $L(i, j, \mu, n_j, k)$ with Equation (4).
- 11: **for** $\nu = 1$ to n_j **do** ▷ right boundary
- 12: Calculate $L(i, j, m_i, \nu, k)$ with Equation (4).
- 13: **return** $\max_{0 \leq k \leq r-1} \{L(M, N, m_M, n_N, k)\}$

Table 10

The state transition table of the STR-IC-LCS problem with $P = abab$ and $\Sigma = \{a, b, c\}$. Here, ϵ denotes the empty string and only state 4 is acceptable.

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

Proof. For Line 1, by Theorem 2, the state transition table and its inverse can be built in $O(r(m + n))$ time. For Line 2, the boundary conditions of $L(\cdot)$ can be set in $O(r(m + n))$ time.

Each $Up(\cdot)$ in Equation (2) or $Left(\cdot)$ in Equation (3) can be determined in $O(1)$ time.

In Equation (4), $L(\cdot)$ of each cell can be computed in $O(1)$ time when $\alpha_i \neq \beta_j$.

Suppose that $\alpha_i = \beta_j = \sigma$. When $\mu > \nu = u$, computing $L(i, j, \mu, \nu, k)$ for $0 \leq k \leq r - 1$, involves transitioning each state $k' \in S_{k, \sigma^u} = \{k' \mid \delta(k', \sigma^u) = k\}$ to state k . The size of S_{k, σ^u} for a specific k may be more than one. However, one value of k' can transition to only one value of k . So, there are at most r distinct values for k' . In other words, $|S_{0, \sigma^u}| + |S_{1, \sigma^u}| + \dots + |S_{r-1, \sigma^u}| \leq r$. Consequently, computing $L(i, j, \mu, \nu, k)$ for these r cells, for all $0 \leq k \leq r - 1$, can be done in $O(r)$ time.

For a specific k , we only calculate the cells on the bottom boundary or the right boundary in a gray block, and the cell at the bottom-right corner in a white block. The number of computed cells is bounded by $O(Mm + mN)$. Therefore, the total time for calculating $L(\cdot)$ across the cells with distinct values of k is $O(r(Mm + mN))$.

In summary, the overall time complexity for Algorithm 1 is $O(r(Mm + mN))$. \square

4. The STR-IC-LCS, SEQ-IC-LCS and SEQ-EC-LCS problems

We can apply the concept of the state transition table to solve the STR-IC-LCS, SEQ-IC-LCS, and SEQ-EC-LCS problems by building a state transition table for each problem.

For the STR-IC-LCS problem, the definition of the state transition for the constraint string P is similar to Definitions 2 and 3. However, there are two differences: (1) The state of $k = r$ has to be added, where $r = |P|$. Specifically, we have to define the transition for $\delta(r, \sigma)$, where $\sigma \in \Sigma$. (2) State r is the only acceptable state; all other states are unacceptable. For example, Table 10 illustrates the state transition table for the STR-IC-LCS problem.

As observed, the state transition tables of STR-IC-LCS and STR-EC-LCS are nearly identical, differing primarily in the acceptable states. Thus, by Theorem 2, the time required for establishing the state transition table for STR-IC-LCS is $O(r(m + n))$.

With the above state transition table and the similar method of STR-EC-LCS, the STR-IC-LCS problem can be solved in $O(r(Mm + mN))$ time. However, Tseng et al. proposed a more efficient algorithm for the STR-IC-LCS problem, achieving a time complexity of $O(mn + mr + nr)$ time [33]. Thus, future work should focus on redesigning the state transition table for STR-IC-LCS to improve efficiency and reduce the required time.

For the SEQ-IC-LCS and SEQ-EC-LCS problems [12], we have to redefine the state of two strings as follows.

Definition 8 (State of two strings in SEQ-IC-LCS and SEQ-EC-LCS). Given a string $S = s_1s_2\dots s_t$ and a constraint string $P = p_1p_2\dots p_r$, if P is a subsequence of S , then we denote $\text{state}(S, P) = r$; otherwise, we denote $\text{state}(S, P) = k$ if $P_{1..k}$ is a subsequence of S , but $P_{1..k+1}$ is not a subsequence of S . Let $\delta'(k, \sigma)$ denote $\text{state}(S \oplus \sigma, P)$, where $k = \text{state}(S, P)$.

Table 11

The state transition table for SEQ-IC-LCS with $P = abab$ and $\Sigma = \{a, b, c\}$. Here, ϵ denotes the empty string and state 4 is the only acceptable state.

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

Table 12

The state transition table for SEQ-EC-LCS with $P = abab$ and $\Sigma = \{a, b, c\}$. Here, ϵ denotes the empty string. States 0, 1, 2, and 3 are acceptable, but state $r = 4$ is unacceptable.

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

Table 13

The comparison of this study and the two algorithms proposed by Ann et al. [5] for SEQ-IC-LCS with RLE. Here, m_i, n_j and r_k denote the lengths of RX_i, RY_j and RP_k , respectively.

	This study	First of Ann et al. [5]	Second of Ann et al. [5]
white	$O(r_k)$	$O(m_i n_j + m_i r_k + n_j r_k)$	$O(r_k)$
gray	$O(r_k \times (m_i + n_j))$	$O(m_i n_j + m_i r_k + n_j r_k)$	$O(r_k \times n_j)$
black	$O(r_k \times (m_i + n_j))$	$O(m_i n_j + m_i r_k + n_j r_k)$	$O(m_i n_j + m_i r_k + n_j r_k)$

With the above definition, $\delta'(\cdot)$ can be calculated as follows.

$$\delta'(k, \sigma) = \begin{cases} r, & \text{if } k = r \\ k + 1, & \text{if } 0 \leq k \leq r - 1 \text{ and } \sigma = p_{k+1} \\ k, & \text{if } 0 \leq k \leq r - 1 \text{ and } \sigma \neq p_{k+1} \end{cases} \quad (5)$$

For example, Table 11 shows an example of the state transition table for the SEQ-IC-LCS problem, in which state $r = 4$ is the only acceptable state. Table 12 shows an example of the state transition table for the SEQ-EC-LCS problem, in which states 0, 1, 2, and 3 are acceptable, but state $r = 4$ is unacceptable.

The state transition tables of SEQ-IC-LCS and SEQ-EC-LCS are the same; the only difference is the acceptable states. The state transition table $\delta'(\cdot)$ can be built with Equation (5) in $O(r(m + n))$ time.

It is worth noting that the main difference in the state transition between STR and SEQ lies in how the substring or subsequence in the constraint P is treated. For STR, the substring in P must be consecutive, but for SEQ, the subsequence in P can be interrupted by any other character that is not the next expected character in P . Consider Table 10. For STR, state 2 expects the next character to be a to extend the prefix match of P . We have $\delta(2, b) = 0$, since b interrupts this extension. Thus, the state resets (not always resets to 0).

However, in Table 11, when the next character is b, the state 2 remains unchanged, i.e. $\delta'(2, b) = 2$. This means that b is discarded for the purpose of extending the constraint, but it still contributes to the total length of the answer. The state remains at 2, still waiting for the next character a to extend the prefix match in P .

With the state transition tables of STR-IC-LCS, SEQ-IC-LCS and SEQ-EC-LCS in the RLE format, the DP stage can be performed similarly with Equation (4). Hence, these three CLCS problems with the RLE format can also be solved in $O(r(Mn + mN))$ time and space.

Here, we compare our algorithm of SEQ-IC-LCS and the two algorithms proposed by Ann et al. [5] in Table 13. For comparison, we divide the time complexity into three parts: white cuboids ($\alpha_i \neq \beta_j$), gray cuboids ($\alpha_i = \beta_j \neq \rho_k$) and black cuboids ($\alpha_i = \beta_j = \rho_k$).

Although our algorithm is not superior to the two algorithms of Ann et al., it remains competitive. Our algorithm needs to build the state transition table during the preprocessing stage. Then, our algorithm is much simpler to perform calculation in the DP stage with the state transition table. It is noteworthy that our algorithm employs a unified strategy to tackle all four variants of the CLCS problem.

5. Conclusion

In this paper, we solve the STR-EC-LCS problem with the RLE strings in $O(r(Mn + mN))$ time and space. In other previous algorithms without the RLE format [3,35], the computation of each $block(i, j)$ (matched or mismatched) requires $O(m_i \times n_j)$ time. Our algorithm reduces the required time to $O(m_i + n_j)$ for a matched block and to $O(1)$ for a mismatched block.

Our algorithm is versatile and can be extended to address the three other variants of the CLCS problem: STR-IC-LCS, SEQ-IC-LCS and SEQ-EC-LCS. For these three variants, we only need to build the state transition table with a similar way in the preprocessing stage. Subsequently, the same DP formula can be applied to solve all four variants of the CLCS problem. However, applying the state transition method to STR-IC-LCS may not be so efficient since it has already been solved in $O(mn)$ time [33,17]. To overcome this limitation, the state transition table may need to be constructed by considering substrings, rather than individual characters.

To solve the STR-EC-LCS problem more efficiently, one possible approach is to apply the RMQ technique. By doing so, we would only need to calculate the cells on either the bottom boundary or the right boundary, rather than both in the current algorithm. However, it would make the algorithm more sophisticated, even though it may be slightly faster practically. Additionally, it would increase the required time for calculating each cell in a matched block.

As for multiple constraints, Ann et al. [3] successfully demonstrated how to solve the STR-EC-LCS problem with plain sequences (not RLE) along with multiple plain constraints. They constructed the state transition table using the Aho-Corasick automaton [1] with multiple constraint strings. Similarly, we can build the state transition table for multiple constraints using the scheme described in Section 3.1. Once the state transition table is established, the DP formula in Equation (4) can be used to solve the problem with multiple RLE constraints. This method can be applied to all of the four problems solved in this paper.

CRediT authorship contribution statement

En-An Song: Writing – original draft, Methodology, Formal analysis, Conceptualization. **Chang-Biau Yang:** Writing – review & editing, Validation, Supervision, Formal analysis, Conceptualization. **Kuo-Tsung Tseng:** Writing – review & editing, Validation, Formal analysis.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Chang-Biau Yang reports financial support was provided by National Science and Technology Council of Taiwan. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

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

Data availability

No data was used for the research described in the article.

References

- [1] A.V. Aho, M.J. Corasick, Efficient string matching: an aid to bibliographic search, *Commun. ACM* 18 (6) (1975) 333–340.
- [2] I. Alsmadi, M. Nuser, String matching evaluation methods for DNA comparison, *Int. J. Adv. Sci. Technol.* 47 (1) (2012) 13–32.
- [3] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion, *J. Comb. Optim.* 28 (4) (2014) 800–813.
- [4] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, C.-Y. Hor, A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings, *Inf. Process. Lett.* 108 (6) (2008) 360–364.
- [5] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, C.-Y. Hor, Fast algorithms for computing the constrained LCS of run-length encoded strings, *Theor. Comput. Sci.* 432 (2012) 1–9.
- [6] A. Apostolico, C. Guerra, The longest common subsequence problem revisited, *Algorithmica* 2 (1987) 315–336.
- [7] A. Apostolico, G.M. Landau, S. Skiena, Matching for run-length encoded strings, *J. Complex.* 15 (1) (1999) 4–16.
- [8] O. Arbell, G.M. Landau, J.S. Mitchell, Edit distance of run-length encoded strings, *Inf. Process. Lett.* 83 (6) (2002) 307–314.
- [9] A.N. Arslan, Ö. Egecioğlu, Algorithms for the constrained longest common subsequence problems, *Int. J. Found. Comput. Sci.* 16 (06) (2005) 1099–1109.
- [10] D. Becerra, W. Soto, L. Nino, Y. Pinzón, An algorithm for constrained LCS, in: Proceedings of 2010 IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, IEEE, 2010, pp. 1–7.
- [11] H. Bunke, J. Csirik, An improved algorithm for computing the edit distance of run-length coded strings, *Inf. Process. Lett.* 54 (2) (1995) 93–96.
- [12] Y.C. Chen, K.M. Chao, On the generalized constrained longest common subsequence problems, *J. Comb. Optim.* 21 (3) (2011) 383–392.
- [13] F.Y. Chin, A. De Santis, A.L. Ferrara, N. Ho, S. Kim, A simple algorithm for the constrained sequence problems, *Inf. Process. Lett.* 90 (4) (2004) 175–179.
- [14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, MIT Press, 2009.
- [15] S. Deorowicz, Fast algorithm for the constrained longest common subsequence problem, *Theor. Appl. Inform.* 19 (2) (2007) 91–102.
- [16] S. Deorowicz, Bit-parallel algorithm for the constrained longest common subsequence problem, *Fundam. Inform.* 99 (4) (2010) 409–433.
- [17] S. Deorowicz, Quadratic-time algorithm for a string constrained LCS problem, *Inf. Process. Lett.* 112 (11) (2012) 423–426.
- [18] S. Deorowicz, S. Grabowski, Subcubic algorithms for the sequence excluded LCS problem, in: D.A. Gruca, T. Czachórski, S. Kozielski (Eds.), *Man-Machine Interactions 3*, Springer International Publishing, Cham, 2014, pp. 503–510.
- [19] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Commun. ACM* 18 (6) (1975) 341–343.

- [20] D.S. Hirschberg, Algorithms for the longest common subsequence problem, *J. ACM* 24 (4) (1977) 664–675.
- [21] W.-C. Ho, A fast algorithm for the constrained longest common subsequence problem with small alphabet, Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, 2017.
- [22] S.-H. Hung, C.-B. Yang, K.-S. Huang, A diagonal-based algorithm for the constrained longest common subsequence problem, in: Proc. of the 23rd International Computer Symposium, 2018 (ICS 2018), Yunlin, Taiwan, Dec. 20–22, in: New Trends in Computer Technologies and Applications, Communications in Computer and Information Science, vol. 1013, Springer, Singapore, 2018, pp. 425–432.
- [23] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, *Commun. ACM* 20 (5) (1977) 350–353.
- [24] C.S. Iliopoulos, M.S. Rahman, New efficient algorithms for the LCS and constrained LCS problems, *Inf. Process. Lett.* 106 (1) (2008) 13–18.
- [25] D.E. Knuth, J.H. Morris Jr, V.R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* 6 (2) (1977) 323–350.
- [26] K. Kuboi, Y. Fujishige, S. Inenaga, H. Bannai, M. Takeda, Faster STR-IC-LCS computation via RLE, in: Proceedings of 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017), Warsaw, Poland, 2017, pp. 20:1–20:12.
- [27] G.M. Landau, E. Myers, M. Ziv-Ukelson, Two algorithms for LCS consecutive suffix alignment, *J. Comput. Syst. Sci.* 73 (7) (2007) 1095–1117.
- [28] J.J. Liu, Y.-L. Wang, R.C.T. Lee, Finding a longest common subsequence between a run-length-encoded string and an uncompressed string, *J. Complex.* 24 (2) (2008) 173–184.
- [29] J. Mitchell, A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings, Technical Report, Department of Applied Mathematics, SUNY StonyBrook, NY, 1997.
- [30] N. Nakatsu, Y. Kambayashi, S. Yajima, A longest common subsequence algorithm suitable for similar text strings, *Acta Inform.* 18 (2) (1982) 171–179.
- [31] 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 (2010) 51–59.
- [32] Y.-T. Tsai, The constrained longest common subsequence problem, *Inf. Process. Lett.* 88 (4) (2003) 173–176.
- [33] C.-T. Tseng, C.-B. Yang, H.-Y. Ann, Efficient algorithms for the longest common subsequence problem with sequential substring constraints, *J. Complex.* 29 (1) (2013) 44–52.
- [34] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, *J. ACM* 21 (1) (1974) 168–173.
- [35] L. Wang, X. Wang, Y. Wu, D. Zhu, A dynamic programming solution to a generalized LCS problem, *Inf. Process. Lett.* 113 (19) (2013) 723–728.
- [36] K. Yamada, Y. Nakashima, S. Inenaga, H. Bannai, M. Takeda, Faster STR-EC-LCS computation, in: Proceedings of the 46th International Conference on Current Trends in Theory and Practice of Computer Science, Limassol, Cyprus, 2020, pp. 125–135.
- [37] Y. Yonemoto, Y. Nakashima, S. Inenaga, H. Bannai, Faster space-efficient STR-IC-LCS computation, *Theor. Comput. Sci.* 1003 (2024) 114607.