The Computer Journal Advance Access published March 5, 2014

doi:10.1093/comjnl/bxu012

Constrained Longest Common Subsequences with Run-Length-Encoded Strings

JIA-JIE LIU[1,*], YUE-LI WANG[2] AND YU-SHAN CHIU[1]

[1]*Department of Information Management, Shih Hsin University, 1 Lane 17 Sec.1, Mu-Cha Rd., Taipei 10607, Taiwan*
[2]*Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan*
[*]*Corresponding author: jjliu@cc.shu.edu.tw*

Given two strings X and Y and a constraining string P, a string Z is called a constrained longest common subsequence of X and Y with respect to P if Z is the longest common subsequence of X and Y such that P is a subsequence of Z. In this paper, we propose an $O(r \times \min\{mN, nM\})$-time algorithm for solving this problem, where m, n and r are the lengths of X, Y and P, respectively, and M and N are the number of runs of the run-length-encoded strings of X and Y, respectively.

Keywords: longest common subsequence; constrained longest common subsequence; run-length-encoding; string compression

Received 12 June 2013; revised 23 January 2014
Handling editor: Fionn Murtagh

1. INTRODUCTION

Let $T = t_1 t_2 \cdots t_{|T|}$ be a string over a finite alphabet set Σ, where $|T|$ denotes the length of T and $t_i \in \Sigma$ for integer i with $1 \leqslant i \leqslant |T|$. Substring $t_i t_{i+1} \cdots t_j$ of T with $1 \leqslant i \leqslant j \leqslant |T|$ is represented by $T_{i,j}$. For simplicity, we also use T_j to represent $T_{i,j}$ when $i = 1$. Measuring the similarity or difference between two strings is fundamental to many applications. For this purpose, many measures are defined, and the longest common subsequence (abbreviated as LCS) is possibly the most popular one. A *subsequence* of a string is obtained by deleting zero or some (not necessarily consecutive) characters from this string. A *common subsequence* of strings X and Y is a subsequence in both X and Y, where $|X| = m$ and $|Y| = n$. An *LCS* of X and Y is a common subsequence with the maximum length. The *LCS problem* is to find an LCS between X and Y.

In [1], Tsai introduced the constrained longest common subsequence problem (the CLCS problem for short) which is described as follows: Given two strings X and Y and a constraining string P, a string Z is called a CLCS of X and Y with respect to P if Z is a LCS of X and Y containing P as a subsequence. The *CLCS problem* is to find a CLCS for X and Y with respect to P. In [1], Tsai gave an $O(m^2n^2r)$-time algorithm for solving the CLCS problem, where m, n and r are the lengths of X, Y and P, respectively. In [2], Chin *et al.* (and independently, Arslan *et al.* [3]) proposed an $O(mnr)$-time algorithm for solving this problem. Iliopoulos and Rahman [4] proposed an algorithm for solving the CLCS problem in $O(m + n + rq \log\log(m + n))$ time, where q is the total number of ordered pairs of positions at which X and Y match.

Run-length-encoding strings are a simple technique to compress strings. It divides a string into several *runs* and each run consists of maximal consecutive identical letters. A *run-length-encoded string* (RLE string for short) X is represented by $r_1^{\ell_1} r_2^{\ell_2} \cdots r_M^{\ell_M}$, where r_j for $1 \leqslant j \leqslant M$ is the repeated character of run j and ℓ_j is its corresponding run-length. For example, the RLE string of string $bdcccaaaaaa$ is $b^1d^1c^3a^6$. The reader is referred to [5] for the details of RLE strings.

The string alignment problem with RLE strings has been widely studied [6–17]. Ann *et al.* [7] proposed an $O(r(mN + nM))$-time algorithm for solving the CLCS problem with RLE strings, where M and N are the numbers of runs in X and Y, respectively. In this paper, we propose an $O(r \times \min\{mN, nM\})$-time algorithm for solving the problem.

The remaining part of this paper is organized as follows. In some preliminaries are introduced in Section 2. In this section, we also explain the reason why our algorithm is better than

the algorithm in [7]. In Section 3, we introduce the recurrence formula introduced in [2] for solving the CLCS problem. Some properties of CLCS with RLE strings are introduced in Section 4. We also introduce our $O(r \times \min\{mN, nM\})$-time algorithm in this section. Finally, concluding remarks and open problems are given in Section 5.

2. PRELIMINARIES

In this section, we introduce some terms which are used in the rest of this paper. We also use examples to illustrate the algorithms proposed in [2, 7]. Note that the algorithm proposed in [7] is with RLE strings while the algorithm proposed in [2] is used to solve the CLCS problem when strings are not encoded. After that, we use an example to illustrate our result so that the reader can understand what we want to improve the algorithm in [7].

We assume that $|X| = m, |Y| = n$ and $|P| = r$ and the numbers of runs in X, Y and P are M, N and R, respectively. We also assume that $X = r_1^{l_1} r_2^{l_2} \cdots r_M^{l_M}$, where r_i for $1 \leqslant i \leqslant M$ is the repeated character of run i and l_i is its run-length.

Let $\ell_{i,j,k}$ stand for the length of a CLCS of X_i and Y_j with constraining string P_k. By using the recurrence formula defined in [2], $\ell_{i,j,k}$ can be computed as follows:

$$\ell_{i,j,k} = \begin{cases} \ell_{i-1,j-1,k-1} + 1 & \text{if } k > 0 \text{ and } x_i = y_j = p_k, \\ \ell_{i-1,j-1,k} + 1 & \text{if } x_i = y_j, \text{ and either} \\ & \quad k = 0 \text{ or } x_i \neq p_k, \\ \max\{\ell_{i-1,j,k}, \ell_{i,j-1,k}\} & \text{if } x_i \neq y_j, \end{cases} \tag{1}$$

where $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$ and $0 \leqslant k \leqslant r$ with boundary conditions $\ell_{i,0,0} = \ell_{0,j,0} = 0$ and $\ell_{0,j,k} = \ell_{i,0,k} = -\infty$ for $0 \leqslant i \leqslant m$, $0 \leqslant j \leqslant n$ and $0 \leqslant k \leqslant r$.

We call the above formula *the standard formula* of an CLCS algorithm. The values of $\ell_{i,j,k}$ form a 3D dynamic programming (DP) lattice. We use (i, j, k) to denote an element in a 3D DP lattice. Let $X_{a,b}, Y_{c,d}$ and $P_{e,f}$ be a run in X, Y and P, respectively. All elements (i, j, k) in a 3D DP lattice with $a \leqslant i \leqslant b, c \leqslant j \leqslant d$ and $e \leqslant k \leqslant f$ form a *cuboid* [7]. Thus, there are $N \times M \times (R+1)$ cuboids in a 3D DP lattice. A horizontal slice of a 3D DP lattice containing all of the values $\ell_{i,j,k}$ for a fixed k, $0 \leqslant k \leqslant r$, is called the kth CLCS *table*. Note that, for simplicity, we neglect x_0 and y_0. However, the zeroth CLCS table is necessary in our algorithm.

EXAMPLE 2.1. In Fig. 1, $X = ddaaadddd$, $Y = adaadddd$ and $P = ddd$. We can find that $M = 3$, $N = 4$ and $R = 1$. Thus, there are $M \times N \times (R+1) = 24$ cuboids. Since $p_0 = \epsilon$ and $r = 3$, there are four CLCS Tables, i.e. the zeroth–third CLCS tables (see Fig. 2). For brevity, we use '$-$' to represent '$-\infty$' in Fig. 2. The zeroth CLCS table contains 12 cuboids and the thickness of each cuboid is 1 since $p_0 = \epsilon$. The thickness of the other 12 cuboids is 3 since $p_1 = p_2 = p_3 = d$.

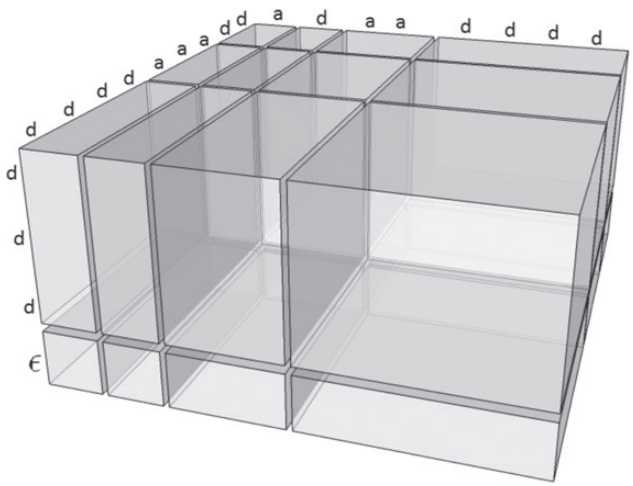

FIGURE 1. A 3D DP lattice.

In [7], Ann *et al.* classified cuboids into the following classes: (a) *fully matched cuboids* if $x_i = y_j = p_k$, (b) *partially matched cuboid* if $x_i = y_j \neq p_k$, and *mismatched cuboids* if $x_i \neq y_j$, where (i, j, k) is an element in the cuboid. The *prism* of a cuboid contains the elements (i, j, k) with i and j maximal in the cuboid (see Fig. 3(a)). The *face* of a cuboid contains the elements (i, j, k) with i maximal in the cuboid (see Fig. 3(b)). The *surface* of a cuboid contains the elements (i, j, k) in which one of i, j and k is zero or maximal in the cuboid (see Fig. 3(c)). They also showed that

(i) for a mismatched cuboid, we need only to compute the elements in its prism,
(ii) for a partially matched cuboid, we need only to compute the elements in its face and
(iii) for a fully matched cuboid, we need only to compute the elements in its surface.

EXAMPLE 2.2. Figure 4 depicts the values which are computed by the algorithm proposed in [7]. Note that an empty element in the CLCS table means that it is not necessary to compute the value of the element. The upper-left blocks in the first–third CLCS tables form a mismatched block since $x_i = d$ and $y_j = a$. The lower-right blocks in the first–third CLCS tables form a fully matched block since $x_i = y_j = p_k = d$. The block in the second row and the third column in the first–third CLCS tables form a partially matched block since $x_i = y_j = a$ and $p_k = d$.

From Example 2.2, we can find that the most time-consuming step in the algorithm proposed in [7] is to compute the values of the surfaces of matched cuboids. Thus, the time complexity of their algorithm is $O(r(mN + nM))$. In our algorithm, we need only to compute the values in the faces of matched cuboids as

the 0th CLCS table

		$p_0=\epsilon$		Y						
				a	d	a	a	d	d	d
		d		0	1	1	1	1	1	1
		d		0	1	1	1	2	2	2
		a		1	1	2	2	2	2	2
		a		1	1	2	3	3	3	3
X		a		1	1	2	3	3	3	3
		d		1	2	2	3	4	4	4
		d		1	2	2	3	4	5	5
		d		1	2	2	3	4	5	6
		d		1	2	2	3	4	5	6

the 1st CLCS table

		$p_1=d$		Y						
				a	d	a	a	d	d	d
		d		–	1	1	1	1	1	1
		d		–	1	1	1	2	2	2
		a		–	1	2	2	2	2	2
		a		–	1	2	3	3	3	3
X		a		–	1	2	3	3	3	3
		d		–	2	2	3	4	4	4
		d		–	2	2	3	4	5	5
		d		–	2	2	3	4	5	6
		d		–	2	2	3	4	5	6

the 2nd CLCS table

		$p_2=d$		Y						
				a	d	a	a	d	d	d
		d		–	–	–	–	–	–	–
		d		–	–	–	–	2	2	2
		a		–	–	–	–	2	2	2
		a		–	–	–	–	2	2	2
X		a		–	–	–	–	2	2	2
		d		–	–	–	–	4	4	4
		d		–	–	–	–	4	5	5
		d		–	–	–	–	4	5	6
		d		–	–	–	–	4	5	6

the 3rd CLCS table

		$p_3=d$		Y						
				a	d	a	a	d	d	d
		d		–	–	–	–	–	–	–
		d		–	–	–	–	–	–	–
		a		–	–	–	–	–	–	–
		a		–	–	–	–	–	–	–
X		a		–	–	–	–	–	–	–
		d		–	–	–	–	–	3	3
		d		–	–	–	–	–	5	5
		d		–	–	–	–	–	5	6
		d		–	–	–	–	–	5	6

FIGURE 2. An illustration for Equation (1) with $X = ddaaadddd$, $Y = adaaddd$ and $P = ddd$.

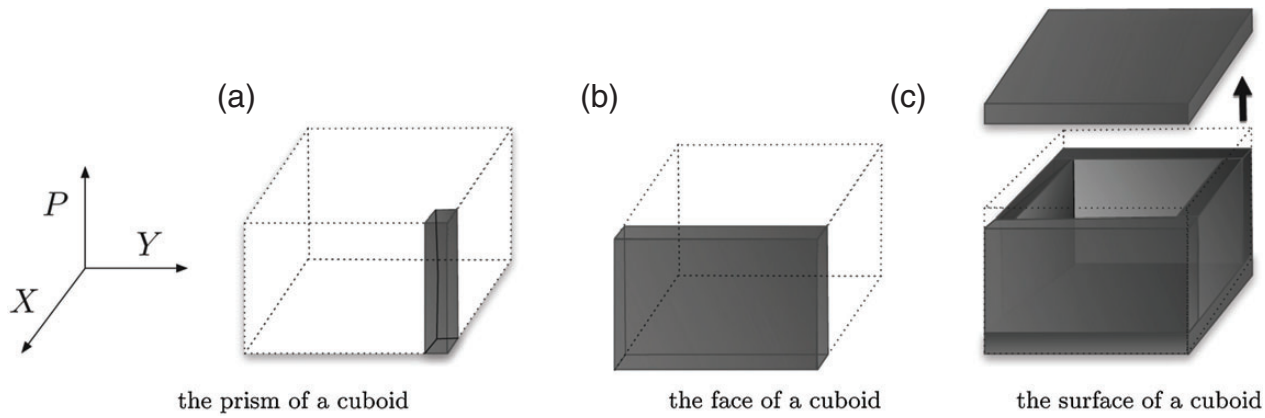

FIGURE 3. Prisms, faces and surfaces. (**a**) The prism of a cuboid. (**b**) The face of a cuboid. (**c**) The surface of a cuboid.

well as partially matched cuboids. Figure 5 depicts the values in each cuboid which are computed by our algorithm.

3. A RECURRENCE FORMULA FOR COMPUTING CLCS

In our algorithm, we use the inverted index technique to compute $\ell_{i,j,k}$. Since our algorithm only computes $\ell_{i,j,k}$ in the face of each cuboid, we shall derive a recurrence formula for them. Before introducing the recurrence formula, we need the following terms and properties which will be used in our algorithm.

Let T be a string and $\tau \in \Sigma$ a symbol in T. The position of the ith τ in T for $1 \leqslant i \leqslant |T|$ is denoted by $T_\tau(i)$, where $|T|$ denotes the length of T. In particular, let $T_\tau(0) = 0$. The inverse function of T_τ is denoted by T_τ^{-1}. That is, if $T_\tau(i) = j$, then $T_\tau^{-1}(j) = i$. Let $\mathrm{pre}^u(T_\tau(i))$ denote the position just preceding the $(i-u+1)$th τ in T for $1 \leqslant u \leqslant i$. When $u = 1$, $\mathrm{pre}^u(T_\tau(i))$ is simply written as $\mathrm{pre}(T_\tau(i))$. Note that $\mathrm{pre}^u(T_\tau(i)) = \mathrm{pre}^1(T_\tau(i-u+1)) = \mathrm{pre}(T_\tau(i-u+1))$. In particular, let $\mathrm{pre}^0(T_\tau(i)) = T_\tau(i)$. For example, if $u = 1$, then

the 0th CLCS table

$p_0 = \epsilon$		Y						
		a	d	a	a	d	d	d
	d							
	d	0	1		1	2	2	2
	a							
	a							
X	a	1	1	2	3			3
	d							
	d							
	d							
	d	1	2		3	4	5	6

the 1st CLCS table

$p_1 = d$		Y						
		a	d	a	a	d	d	d
	d		1			1	1	1
	d	–	1		1	2	2	2
	a	–						
	a	–						
X	a	–	1	2	3			3
	d		2			4	4	4
	d		2			4	5	5
	d		2			4	5	6
	d	–	2		3	4	5	6

the 2nd CLCS table

$p_2 = d$		Y						
		a	d	a	a	d	d	d
	d		–			–	–	–
	d	–	–		–	2	2	2
	a	–						
	a	–						
X	a	–	–	–	–			2
	d		–			4	4	4
	d		–			4		5
	d		–			4		6
	d	–	–		–	4	5	6

the 3rd CLCS table

$p_3 = d$		Y						
		a	d	a	a	d	d	d
	d		–			–	–	–
	d	–	–	–	–	–	–	–
	a	–						
	a	–						
X	a	–	–	–	–		–	–
	d		–			–	3	3
	d		–			–	5	5
	d		–			–	5	6
	d	–	–		–	–	5	6

FIGURE 4. The elements computed by the algorithm in [7].

the 0th CLCS table

$p_0 = \epsilon$		Y						
		a	d	a	a	d	d	d
	d							
	d	0	1		1	2	2	2
	a							
	a							
X	a	1	1	2	3			3
	d							
	d							
	d							
	d	1	2		3	4	5	6

the 1st CLCS table

$p_1 = d$		Y						
		a	d	a	a	d	d	d
	d							
	d	–	1		1	2	2	2
	a							
	a							
X	a	–	1	2	3			3
	d							
	d							
	d							
	d	–	2		3	4	5	6

the 2nd CLCS table

$p_2 = d$		Y						
		a	d	a	a	d	d	d
	d							
	d	–	–		–	2	2	2
	a							
	a							
X	a	–	–	–	–			2
	d							
	d							
	d							
	d	–	–		–	4	5	6

the 3rd CLCS table

$p_3 = d$		Y						
		a	d	a	a	d	d	d
	d							
	d	–	–	–	–	–	–	–
	a							
	a							
X	a	–	–	–	–		–	–
	d							
	d							
	d							
	d	–	–		–	–	5	6

FIGURE 5. The elements computed by our algorithm.

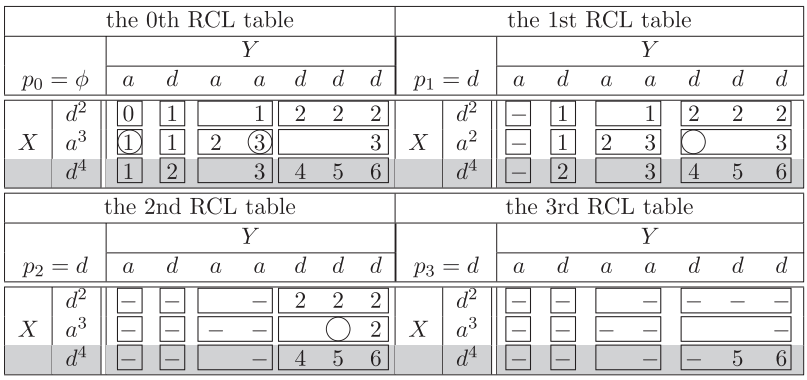

FIGURE 6. An illustration of the 3D DP lattice with X in RLE form.

$\text{pre}^1(T_\tau(i))$ is the position just preceding position j in T, namely position $j-1$, where $j = T_\tau(i)$. For t_i in T, the largest position $j < i$ (respectively, the smallest position $j > i$) with $t_j \neq t_i$ is denoted by $\delta_T(i)$ (respectively, $\Delta_T(i)$). If T_j is ended with s τ's, i.e. $t_j = \tau$ and $s = j - \delta_T(j)$, then T_j is also represented as $T_j \| \tau^s$. Note that $T_j \| \tau^0$ implies that $t_j \neq \tau$. For brevity, $T_{|T|} \| \tau^s$ is simply written as $T \| \tau^s$.

EXAMPLE 3.1. We use an example to illustrate above terms. Suppose that $T = t_1 t_2 \cdots t_8 = \underset{1\,2\,3\,4\,5\,6\,7\,8}{baaaccaa}$. We can find that $T_a(1) = 2$, $T_a(2) = 3$, $T_a(3) = 4$, $T_a(4) = 7$ and $T_a(5) = 8$ while $T_a^{-1}(2) = 1$, $T_a^{-1}(3) = 2$, $T_a^{-1}(4) = 3$, $T_a^{-1}(7) = 4$ and $T_a^{-1}(8) = 5$. Furthermore, $\text{pre}^0(T_a(4)) = 7$, $\text{pre}^1(T_a(4)) = 6$, $\text{pre}^2(T_a(4)) = \text{pre}(T_a(4-2+1)) = \text{pre}(T_a(3)) = 3$ and so on. In addition, $\delta_T(2) = \delta_T(3) = \delta_T(4) = 1$ and $\delta_T(5) = 4$. The representation $T \| a^2$ means that T is ended with two a's.

Let $lp(r_i)$ stand for the position of the last character of r_i in the corresponding uncompressed string. That is, $lp(r_i) = l_1 + l_2 + \cdots + l_i$. Let $\sigma_X(r_i)$ denote the largest number $j < i$ such that $r_j = r_i$. If no such r_j exists, then $\sigma_X(r_i) = 0$. Note that, when X is represented in RLE form, every cuboid in a 3D DP lattice becomes a face. Accordingly, we also call them *fully matched faces*, *partially matched faces* and *mismatched faces*. A horizontal slice containing all $\ell_{lp(r_i),j,k}$ for a fixed k, $0 \leqslant k \leqslant r$, is called the *kth RCL table*.

EXAMPLE 3.2. The CLCS tables in Fig. 5 can be represented by the RCL tables as shown in Fig. 6. Since only elements $\ell_{lp(r_i),j,k}$ are contained in a RCL table, we use their runs in X as indices.

In the following, we introduce some lemmas to compute the values in the $lp(r_i)$th row of a CLCS table by only using the values in the $lp(r_j)$th row with $j \leqslant i$.

LEMMA 3.1 [7]. *For $0 < i \leqslant M$, $0 < j \leqslant n$ and $0 \leqslant k \leqslant r$, if $r_i \neq y_j$, then $\ell_{lp(r_i),j,k} = \max\{\ell_{lp(r_i),\delta_Y(j),k}, \ell_{lp(r_{i'}),j,k}\}$, where $i' = \sigma_X(r_i)$.*

PROPOSITION 3.1. *Assume that $Z \| \tau^u$ is a CLCS of $X_{lp(r_i)}$, Y_j and P_k with $0 < u \leqslant l_i$. If $r_i = \tau$, then $\ell_{lp(r_i),j,k} = \ell_{lp(r_i)-1,j,k} = \cdots = \ell_{lp(r_i)-l_i+u,j,k}$.*

Proof. If $r_i = \tau$ and $0 < u \leqslant l_i$, then we can use the first u τ's in $r_i^{l_i}$ to form Z. This implies that $\ell_{lp(r_i),j,k} = \ell_{lp(r_i)-1,j,k} = \cdots = \ell_{lp(r_i)-l_i+u,j,k}$. □

LEMMA 3.2. *Assume that $y_j = \tau$, $v = Y_\tau^{-1}(j)$ and $w = \min\{v, l_i\}$. If $r_i = y_j \neq p_k$, then $\ell_{lp(r_i),j,k} = \max_{1 \leqslant u \leqslant w}\{\ell_{lp(r_{i-1}),\text{pre}^u(Y_\tau(v)),k} + u\}$.*

Proof. Assume that $Z \| \tau^u$ with $0 < u \leqslant v$ is a CLCS of $X_{lp(r_i)}$, Y_j and P_k. If $u \geqslant \ell_i$, then we can assume that the last ℓ_i τ's of Y_j and those τ's in $r_i^{l_i}$ are used to construct Z. By the non-decreasing property of the values in each row of a CLCS table, the length of Z, i.e. $\ell_{lp(r_i),j,k}$, is equal to $\ell_{lp(r_{i-1}),\text{pre}^{\ell_i}(Y_\tau(v)),k} + \ell_i$. If $1 < u < \ell_i$, then, by Proposition 3.1 and the second formula in Equation (1), we can have the following derivation.

$$\begin{aligned}
\ell_{lp(r_i),j,k} &= \ell_{lp(r_i)-l_i+u,j,k} \\
&= \ell_{lp(r_i)-l_i+u-1,\text{pre}^1(Y_\tau(v)),k} + 1 \\
&= \ell_{lp(r_i)-l_i+u-2,\text{pre}^2(Y_\tau(v)),k} + 2 \\
&\vdots \\
&= \ell_{lp(r_{i-1}),\text{pre}^u(Y_\tau(v)),k} + u.
\end{aligned}$$

As a consequence, by examining all possible values of $\ell_{lp(r_{i-1}),\text{pre}^u(Y_\tau(v)),k} + u$ for $1 \leqslant u \leqslant w = \min\{v, \ell_i\}$, the maximum value among them is the value of $\ell_{lp(r_i),j,k}$ (see Fig. 7 for an illustration). Thus, $\ell_{lp(r_i),j,k} = \max_{1 \leqslant u \leqslant w}\{\ell_{lp(r_{i-1}),\text{pre}^u(Y_\tau(v)),k} + u\}$ and the lemma follows. □

LEMMA 3.3. *Assume that $y_j = \tau$, $v = Y_\tau^{-1}(j)$ and $w = \min\{v, l_i\}$. If $r_i = y_j = p_k$, then*

$$\ell_{lp(r_i),j,k} = \max_{1 \leqslant u \leqslant w}\{\ell_{lp(r_{i-1}),\text{pre}^u(Y_\tau(v)),d_u} + u\}, \text{ where}$$

$$d_u = \max\{k - u, \delta_P(k)\}.$$

		Y									Y						
$p_0=\phi$		a	d	a	a	d	d	d	$p_1=d$		a	d	a	a	d	d	d
	d	0	1	1	1	1	1	1		d	–	1	1	1	1	1	1
	d	0	1	1	1	2	2	2		d	–	1	1	1	2	2	2
	a	1	1	2	2	2	2	2		a	–	1	2	2	2	2	2
	a	1	1	2	3	3	3	3		a	–	1	2	3	3	3	3
X	a	1	1	2	3	3	3	3	X	a	–	1	2	3	3	3	3
	d	1	2	2	3	4	4	4		d	–	2	2	3	4	4	4
	d	1	2	2	3	4	5	5		d	–	2	2	3	4	5	5
	d	1	2	2	3	4	5	6		d	–	2	2	3	4	5	6
	d	1	2	2	3	4	5	6		d	–	2	2	3	4	5	6

FIGURE 7. Computing $\ell_{lp(r_2),4,1}$ for illustrating Lemma 3.2.

Proof. Let $\alpha = k - \delta_P(k)$. Assume that $Z\|\tau^u$ with $u \leqslant v$ is a CLCS of $X_{lp(r_i)}$, Y_j and P_k. According to the relation between u and l_i, we consider the following two cases.

Case 1. $l_i \geqslant u$.

In this case, by Proposition 3.1, $\ell_{lp(r_i),j,k} = \ell_{lp(r_i)-l_i+u,j,k}$. If $\alpha \geqslant u$, then, by applying the first formula in Equation (1) u times, we can obtain

$$\ell_{lp(r_i)-l_i+u,j,k} = \ell_{lp(r_{i-1}),\mathrm{pre}^u(Y_\tau(v)),k-u} + u.$$

For the case where $\alpha < u$, we can apply the first formula in Equation (1) α times. By Lemma 3.2 and the non-decreasing property of the values in each row of a CLCS table, we can obtain

$$\begin{aligned}\ell_{lp(r_i)-l_i+u,j,k} &= \ell_{lp(r_i)-l_i+u-\alpha,\mathrm{pre}^\alpha(Y_\tau(v)),k-\alpha} + \alpha \\ &= \ell_{lp(r_i)-l_i+u-\alpha-(u-\alpha),\mathrm{pre}^{\alpha+u-\alpha}(Y_\tau(v)),k-\alpha} \\ &\quad + \alpha + (u-\alpha) \\ &= \ell_{lp(r_{i-1}),\mathrm{pre}^u(Y_\tau(v)),\delta_P(k)} + u.\end{aligned}$$

Thus, in this case, $\ell_{lp(r_i),j,k} = \ell_{lp(r_{i-1}),\mathrm{pre}^u(Y_\tau(v)),d_u} + u$ where $d_u = k - u$ if $\alpha \geqslant u$; otherwise, $d_u = \delta_P(k)$.

Case 2. $u > l_i$.

In this case, we also consider the possible relation between l_i and α. If $l_i > \alpha$, we can apply the first formula in Equation (1) α times. This yields $\ell_{lp(r_i),j,k} = \ell_{lp(r_i)-\alpha,\mathrm{pre}^\alpha(Y_\tau(v)),k-\alpha} + \alpha$. By Lemma 3.2, we can have the following derivation:

$$\begin{aligned}&\ell_{lp(r_i)-\alpha,\mathrm{pre}^\alpha(Y_\tau(v)),k-\alpha} + \alpha \\ &\quad = \ell_{lp(r_i)-\alpha-(l_i-\alpha),\mathrm{pre}^{\alpha+(l_i-\alpha)}(Y_\tau(v)),k-\alpha} + \alpha + (l_i-\alpha) \\ &\quad = \ell_{lp(r_{i-1}),\mathrm{pre}^{l_i}(Y_\tau(v)),\delta_P(k)} + l_i.\end{aligned}$$

For the case where $\alpha \geqslant l_i$, after applying the first formula in Equation (1) l_i times, this results in $\ell_{lp(r_i),j,k} = \ell_{lp(r_{i-1}),\mathrm{pre}^{l_i}(Y_\tau(v)),k-l_i} + l_i$.

From above cases, by examining the possible values of u from 1 to w, we can obtain that $\ell_{lp(r_i),j,k} = \max_{1\leqslant u\leqslant w}\{\ell_{lp(r_{i-1}),\mathrm{pre}^u(Y_\tau(v)),d_u} + u\}$, where $d_u = \max\{k-u, \delta_P(k)\}$. This concludes the proof of this lemma. □

We use Fig. 8 as an example to illustrate Lemma 3.3. By using Lemma 3.3, the values in the boxes in Fig. 8 are used to compute $\ell_{9,7,3}$. Note that, in computing $\ell_{9,7,3}$, $i = 3$, $j = 7$ and $k = 3$. That is, $lp(r_3) = 9$, $y_7 = d$ and $v = Y_d^{-1}(7) = 4$. According to Lemma 3.3, $\ell_{9,7,3}$ can be computed as follows:

$$\begin{aligned}\ell_{9,7,3} &= \ell_{lp(r_3),7,3} \\ &= \max_{1\leqslant u\leqslant w}\{\ell_{lp(r_2),\mathrm{pre}^u(Y_d(v)),d_u} + u\} \\ &= \max_{1\leqslant u\leqslant 4}\{\ell_{5,\mathrm{pre}^u(Y_d(v)),d_u} + u\} \\ &= \max\{\ell_{5,6,2}+1, \ell_{5,5,1}+2, \ell_{5,4,0}+3, \ell_{5,1,0}+4\} \\ &= \max\{2+1, 3+2, 3+3, 1+4\} \\ &= 6.\end{aligned}$$

Note that, in the above derivation, $d_u = k - u = 3 - u$ for $1 \leqslant u \leqslant 3$ and $d_4 = \max\{k-u, \delta_P(k)\} = \max\{3-4, 0\} = 0$.

For brevity, let $\mathcal{R}_{i,j,k} = \ell_{lp(r_i),j,k}$ for $i = 1, 2, \ldots, M$. Combining Lemmas 3.1–3.3, we can obtain a recurrence formula to compute $\mathcal{R}_{i,j,k}$ as follows (see Fig. 8 for an illustration).

THEOREM 3.1. *Assume that* $X = r_1^{l_1} r_2^{l_2} \cdots r_M^{l_M}$, $y_j = \tau$, $v = Y_\tau^{-1}(j)$, $w = \min\{v, l_i\}$ *and* $i' = \sigma_X(r_i)$. *Then*

$$\mathcal{R}_{i,j,k} = \begin{cases} \max\{\mathcal{R}_{i,\delta_Y(j),k}, \mathcal{R}_{i',j,k}\} & \text{if } r_i \neq y_j, \\ \max\limits_{1\leqslant u\leqslant w}\{\mathcal{R}_{i-1,\mathrm{pre}^u(Y_\tau(v)),k} + u\} & \text{if } r_i = y_j \neq p_k, \\ \max\limits_{1\leqslant u\leqslant w}\{\mathcal{R}_{i-1,\mathrm{pre}^u(Y_\tau(v)),d_u} + u\} & \text{if } r_i = y_j = p_k, \end{cases} \tag{2}$$

with boundary conditions $\mathcal{R}_{i,0,0} = \mathcal{R}_{0,j,0} = 0$ *and* $\mathcal{R}_{0,j,k} = \mathcal{R}_{i,0,k} = -\infty$ *where* $0 \leqslant i \leqslant M$, $0 \leqslant j \leqslant n$, $1 \leqslant k \leqslant r$, $d_u = \max\{k-u, \delta_P(k)\}$.

By using a similar technique as in [7], when $r_i \neq y_j$, we need only to compute those $\mathcal{R}_{i,j,k}$'s with $j = \Delta_Y(j) - 1$ (see Fig. 6 as an example). However, if some $\mathcal{R}_{i-1,\mathrm{pre}^u(Y_\tau(v)),k}$ is not computed when computing $\mathcal{R}_{i,j,k}$ by Equation (2) for the case where $r_i = y_j$, then we can compute $\max\{\mathcal{R}_{i-1,j',k}, \mathcal{R}_{i',\mathrm{pre}^u(Y_\tau(v)),k}\}$ instead, where $i' = \sigma_X(r_{i-1})$ and $j' = \delta_Y(\mathrm{pre}^u(Y_\tau(v)))$.

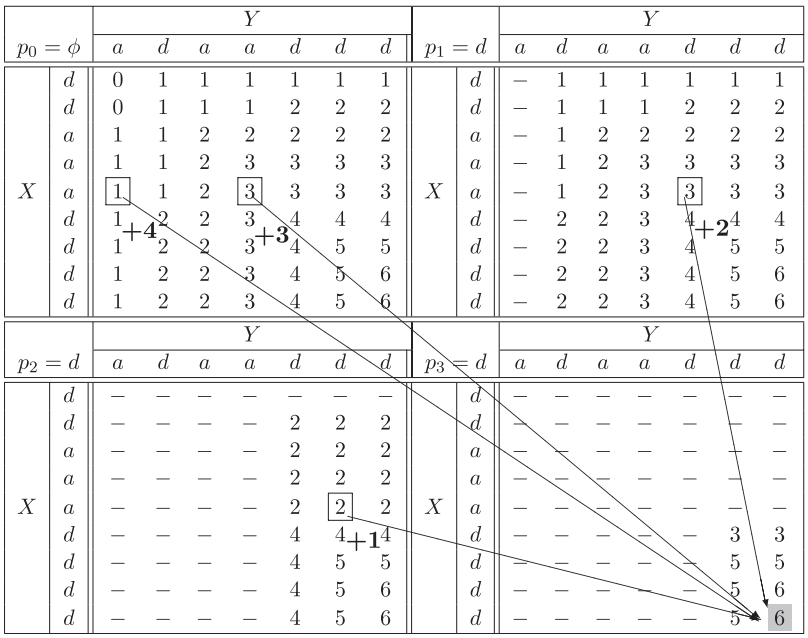

FIGURE 8. Computing $\ell_{lp(r_3),7,3}$ for illustrating Lemma 3.3.

4. AN EFFICIENT WAY TO COMPUTE THE LENGTH OF A CLCS

In Equation (2), the most time-consuming step occurs at the case that $r_i = y_j$. It takes $O(l_i)$ time for computing $\mathcal{R}_{i,j,k}$ so that the total time complexity becomes $O((l_1+l_2+\cdots+l_M)nr-\xi) = O(mnr-\xi)$, where ξ is the number of mismatched elements between r_i and y_j, for $1 \leqslant i \leqslant M$ and $1 \leqslant j \leqslant n$, with $j \neq \Delta_Y(j)-1$. In this section, we shall show how to compute $\mathcal{R}_{i,j,k}$ efficiently so that the total time complexity becomes $O(\min\{Mn, Nm\})$.

In the rest of this section, we assume that $r_i = y_j = \tau$, $v = Y_\tau^{-1}(j)$, $w = \min\{v, l_i\}$ and $\alpha = k - \delta_P(k)$ unless otherwise stated. We also call α the *thickness* of the kth RCL table. The total number of τ's in Y is denoted by μ_τ. An element (i, j, k) in an RCL table is called a *critical element* if $\mathcal{R}_{i,j,k}$ is computed by using the second or third formula in Equation (2). For consistency, the second formula in Equation (2) is rewritten as $\mathcal{R}_{i,j,k} = \max_{1\leqslant u\leqslant w}\{\mathcal{R}_{i-1,\mathrm{pre}^u(Y_\tau(v)),d_u} + u\}$ with $d_u = k$ so as to have the same description as the last formula in Equation (2). For brevity, define

$$\mathcal{R}_x(v) = \begin{cases} \mathcal{R}_{i-1,\mathrm{pre}^1(Y_\tau(x)),d_{v-x+1}} + l_i & \text{if } 1 \leqslant x \leqslant v-w, \\ \mathcal{R}_{i-1,\mathrm{pre}^1(Y_\tau(x)),d_{v-x+1}} & \\ \quad +v-x+1 & \text{if } v-w+1 \leqslant x \leqslant v, \end{cases}$$

with respect to critical element $(i, Y_\tau(v), k)$, where $d_{v-x+1} = k$ for $1 \leqslant x \leqslant v$ when $r_i = y_j \neq p_k$. When context is clear, $\mathcal{R}_x(v)$ is simply written as $\mathcal{R}_x$. Element $(i-1, \mathrm{pre}^x(Y_\tau(v)), d_{v-x+1})$ and $\mathcal{R}_x$ for $1 \leqslant x \leqslant v$ are called an *originating element* and an *originating value*, respectively, of critical element (i, j, k). For example, see Fig. 6. Assume that, for critical element $(1, 6, 2)$, $v = Y_\tau^{-1}(j) = Y_\tau^{-1}(7) = 3$, $w = \min\{v, l_i\} = \min\{3, l_1\} = 2$ and $\alpha = k - \delta_P(k) = 2 - \delta_P(2) = 2$. The originating elements of element $(1, 6, 2)$ are $(0, 1, 0)$, $(0, 4, 0)$ and $(0, 5, 1)$ whose originating values are $\mathcal{R}_1 = 0+2 = 2$, $\mathcal{R}_2 = 0+2 = 2$ and $\mathcal{R}_3 = 0+1 = 1$, respectively. For critical element $(1, 7, 2)$, $v = Y_\tau^{-1}(j) = Y_\tau^{-1}(7) = 4$, $w = \min\{v, l_i\} = \min\{4, l_1\} = 2$ and $\alpha = k - \delta_P(k) = 2 - \delta_P(2) = 2$. The originating elements of element $(1, 7, 2)$ are $(0, 1, 0)$, $(0, 4, 0)$, $(0, 5, 0)$ and $(0, 6, 1)$ whose originating values are $\mathcal{R}_1 = 0+2 = 2$, $\mathcal{R}_2 = 0+2 = 2$, $\mathcal{R}_3 = 0+2 = 2$, and $\mathcal{R}_4 = 0+1 = 1$, respectively.

PROPOSITION 4.1. *For $2 \leqslant v \leqslant \mu_\tau$, $\mathcal{R}_x(v) = \mathcal{R}_x(v-1)$ for $1 \leqslant x \leqslant v-w$, while $\mathcal{R}_x(v) = \mathcal{R}_x(v-1)+1$ for $v-w+1 \leqslant x \leqslant v-1$.*

Since $\mathcal{R}_x$ for $1 \leqslant x \leqslant v-w$ is always $\leq \mathcal{R}_{v-w+1}$, Theorem 3.1 can be rewritten as follows:

THEOREM 4.1. *Assume that $X = r_1^{l_1}r_2^{l_2}\cdots r_M^{l_M}$, $y_j = \tau$, $v = Y_\tau^{-1}(j)$, $w = \min\{v, l_i\}$ and $i' = \sigma_X(r_i)$. Then*

$$\mathcal{R}_{i,j,k} = \begin{cases} \max\{\mathcal{R}_{i,\delta_Y(j),k}, \mathcal{R}_{i',j,k}\} & \text{if } r_i \neq y_j, \\ \max_{1\leqslant x\leqslant v}\{\mathcal{R}_x\} & \text{otherwise}, \end{cases} \tag{3}$$

with boundary conditions $\mathcal{R}_{i,0,0} = \mathcal{R}_{0,j,0} = 0$ and $\mathcal{R}_{0,j,k} = \mathcal{R}_{i,0,k} = -\infty$ where $0 \leqslant i \leqslant M$, $0 \leqslant j \leqslant n$, and $1 \leqslant k \leqslant r$.

Note that if $r_i = y_j \neq p_k$, then all $\mathcal{R}_x$ for $1 \leqslant x \leqslant v$ are in the kth RCL table. However, when $r_i = y_j = p_k$, all $\mathcal{R}_x$ for $1 \leqslant x \leqslant v$ may not be in the same table. We call the set containing all $\mathcal{R}_x$ with $d_u = k$ the *base set* of (i, j, k), when $r_i = y_j \neq p_k$. For $r_i = y_j = p_k$, the set containing all $\mathcal{R}_x$ with $d_u = \delta_P(k)$ is also called the *base set* of (i, j, k) while the set containing all other $\mathcal{R}_x$ is called the *stripe set* of (i, j, k). Accordingly, the kth (respectively, $\delta_P(k)$th) RCL table is called the *base table* of (i, j, k) when $r_i = y_j \neq p_k$ (respectively, $r_i = y_j = p_k$). For the previous example, see Fig. 6. Assume that, for critical element (3, 7, 3), $v = Y_\tau^{-1}(j) = Y_\tau^{-1}(7) = 4$, $w = \min\{v, l_i\} = \min\{4, l_3\} = 4$ and $\alpha = k - \delta_P(k) = 3 - \delta_P(3) = 3$. The originating elements of element (3, 7, 3) are (2, 1, 0), (2, 4, 0), (2, 5, 1) and (2, 6, 2) whose originating values are $\mathcal{R}_1 = 1+4 = 5$, $\mathcal{R}_2 = 3+3 = 6$, $\mathcal{R}_3 = 3+2 = 5$, and $\mathcal{R}_4 = 2+1 = 3$, respectively (see the square boxes in Fig. 8). The base and stripe sets of element (3, 7, 3) are $\{\mathcal{R}_1, \mathcal{R}_2\}$ and $\{\mathcal{R}_3, \mathcal{R}_4\}$, respectively. The base table of (3, 7, 3) is the zeroth RCL table. Similarly, for critical element (3, 6, 3), $v = Y_\tau^{-1}(6) = Y_\tau^{-1}(6) = 3$, $w = \min\{v, l_i\} = \min\{3, l_3\} = 3$ and $\alpha = k - \delta_P(k) = 3 - \delta_P(3) = 3$. Its base and stripe sets are $\{\mathcal{R}_1\} = \{\mathcal{R}_{2,1,0}\}$ and $\{\mathcal{R}_2, \mathcal{R}_3\} = \{\mathcal{R}_{2,4,1}, \mathcal{R}_{2,5,2}\}$, respectively.

LEMMA 4.1. *Assume that $r_i = y_j = p_k = \tau$, $v = Y_\tau^{-1}(j)$, $w = \min\{v, l_i\}$ and $\alpha = k - \delta_P(k)$. If $\alpha = 1$, then the stripe set of (i, j, k) is an empty set. If $v < \alpha$, then the base set of (i, j, k) is an empty set and all values in the stripe set are $-\infty$.*

Proof. If $\alpha = 1$, then, by the third formula of Equation (2), $d_u = \max\{k - u, \delta_P(k)\}$ for $1 \leqslant u \leqslant w$, where $w = \min\{v, l_i\}$ and $\delta_P(k) = k - 1$. This means that $d_u = \delta_P(k)$ for $1 \leqslant u \leqslant w$. By the definition of base sets, all originating elements of (i, j, k) are in its base table, and the stripe set of (i, j, k) is an empty set. For the case where $v < \alpha$, it is obvious that no subsequence of Y_j is the same as P_k and $\mathcal{R}_{i,j,k} = -\infty$. This further implies that the base set of (i, j, k) is empty and all values in the stripe set are $-\infty$. □

Assume that $a_1, a_2, \ldots, a_n$ are a list of numbers. An index is called the *ω-index* of a_i, denoted by ω_i, for $1 \leqslant i \leqslant n$ if $\omega_i = \max\{j | a_j = \max\{a_i, a_{i+1}, \ldots, a_n\}, i \leqslant j \leqslant n\}$, i.e. $\omega_i \geqslant i$ is the largest index such that $a_{\omega_i} = \max\{a_i, a_{i+1}, \ldots, a_n\}$. Let $\{\omega_1, \omega_2, \ldots, \omega_n\}$ be the set of ω-indices of $a_1, a_2, \ldots, a_n$. Merging the same ω-indices as an interval, which is represented by their starting and ending indices, results in a set of intervals $\{[f_1, h_1], [f_2, h_2], \ldots, [f_\lambda, h_\lambda]\}$ in which $[f_i, h_i]$ is called an *ω-interval*. Note that $f_i \leqslant h_i$ for $1 \leqslant i \leqslant \lambda$, and $f_{i+1} = h_i + 1$ for $1 \leqslant i \leqslant \lambda - 1$. For example, let (3, 7, 4, 6, 2, 1, 6, 4) be a list of numbers. Their corresponding ω-indices are 2, 2, 7, 7, 7, 7, 7, 8 and $\{[1, 2], [3, 7], [8, 8]\}$ is the set of their ω-intervals.

PROPOSITION 4.2. *If $\{[f_1, h_1], [f_2, h_2], \ldots, [f_\lambda, h_\lambda]\}$ is the set of ω-intervals of numbers $a_1, a_2, \ldots, a_n$, then $a_{h_1} > a_{h_2} > \cdots > a_{h_\lambda}$.*

Let $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_\beta\}$ and $\{\mathcal{R}_{\beta+1}, \mathcal{R}_{\beta+2}, \ldots, \mathcal{R}_v\}$ be the base set and stripe set, respectively, of critical element (i, j, k). Let $\{[f_1, h_1], [f_2, h_2], \ldots, [f_{b(v)}, h_{b(v)}]\}$ and $\{[f'_1, h'_1], [f'_2, h'_2], \ldots, [f'_{s(v)}, h'_{s(v)}]\}$ be the sets of ω-intervals of the base set and stripe set, respectively, of critical element (i, j, k). That is, there are $b(v)$ and $s(v)$ ω-intervals in the base set and stripe set, respectively, of critical element (i, j, k). Let $a(v) = b(v) + s(v)$. For simplicity, we use $[f_{b(v)+x}, h_{b(v)+x}]$ to represent $[f'_x, h'_x]$ for $1 \leqslant x \leqslant s(v)$. In particular, an ω-interval $[f_t, h_t]$ is called a *base ω-interval* (*b-interval* for short) if $1 \leqslant t \leqslant b(v)$ and is called a *stripe ω-interval* (*s-interval* for short) if $b(v) + 1 \leqslant t \leqslant a(v)$. Moreover, an ω-interval $[f_t, h_t]$ is called a *critical ω-interval* if $\mathcal{R}_{i,j,k} = \mathcal{R}_{h_t}$, and, in this case, h_t is called the *critical ω-index* of (i, j, k). Collecting all b-intervals (respectively, s-intervals) of critical element (i, j, k) forms the *b-interval set* (respectively, *s-interval set*) of (i, j, k). Let $B_i(v, k)$ and $S_i(v, k)$ stand for the b-interval and s-interval, respectively, sets of $(i, Y_\tau(v), k)$. We use critical element (3, 7, 3) in Fig. 6 to illustrate the above terms. Since $\delta_P(3) = 0$, the 0th RCL table is the base table. Thus $\{\mathcal{R}_1, \mathcal{R}_2\} = \{5, 6\}$ and $\{\mathcal{R}_3, \mathcal{R}_4\} = \{5, 3\}$ are the base and stripe, respectively, sets. Furthermore, $\omega_1 = \omega_2 = 2$ for the base set and $\omega_3 = 3$ and $\omega_4 = 4$ for the stripe set, where w_x stands for the ω-index of $\mathcal{R}_x$ for $1 \leqslant x \leqslant 4$. Consequently, there are three ω-intervals of element (3, 7, 3) which are [1, 2], [3, 3] and [4, 4] in which $B_3(4, 3) = \{[1, 2]\}$ and $S_3(4, 3) = \{[3, 3], [4, 4]\}$. Thus, $b(v) = 1$, $s(v) = 2$ and $a(v) = 3$. Furthermore, since $\mathcal{R}_{3,7,3} = \mathcal{R}_2 = 6$, [1, 2] is the critical interval and $h_1 = 2$ is the critical ω-index of (3, 7, 3).

By using the concept of critical ω-indices, Theorem 4.1 can be rewritten as Theorem 4.2.

THEOREM 4.2. *For critical element (i, j, k) with $y_j = \tau$, $v = Y_\tau^{-1}(j)$, $i' = \sigma_X(r_i)$, and critical ω-index h_t,*

$$\mathcal{R}_{i,j,k} = \begin{cases} \max\{\mathcal{R}_{i,\delta_Y(j),k}, \mathcal{R}_{i',j,k}\} & \text{if } r_i \neq y_j, \\ \mathcal{R}_{h_t} & \text{otherwise}, \end{cases}$$

with boundary conditions $\mathcal{R}_{i,0,0} = \mathcal{R}_{0,j,0} = 0$ and $\mathcal{R}_{0,j,k} = \mathcal{R}_{i,0,k} = -\infty$ where $0 \leqslant i \leqslant M$, $0 \leqslant j \leqslant n$, and $1 \leqslant k \leqslant r$.

By inspection on Theorem 4.2, the time complexity on computing $\mathcal{R}_{i,j,k}$ depends on the time for computing $\mathcal{R}_{h_t}$. Clearly, we can compute $\mathcal{R}_{h_t}$ for each critical element and it takes $O(l_i)$ time. In the following, we show that the critical ω-index of element (i, j, k) can be found in $O(1)$ amortized time. Lemmas 4.2 and 4.3 describe how to find $B_i(v, k)$ for the cases $r_i = y_j \neq p_k$ and $r_i = y_j = p_k$ with $k > \delta_P(k)$, respectively.

LEMMA 4.2. *For the case where $r_i = y_j \neq p_k$ with $j = Y_\tau(v)$,*

$$B_i(v, k) = \begin{cases} \{[1, 1]\} & \text{if } v = 1, \\ \{[f_1, h_1], [f_2, h_2], \ldots, \\ \quad [f_t, h_t], [f_{t+1}, v]\} & \text{if } v > 1 \text{ and } t \text{ exists}, \\ \{[f_1, v]\} & \text{otherwise}, \end{cases}$$

(a)

critical elements	$B_3(v,0)$
$(3, Y_d(1), 0)$	$[1,1]$
$(3, Y_d(2), 0)$	$[1,2]$
$(3, Y_d(3), 0)$	$[1,2][3,3]$
$(3, Y_d(4), 0)$	$[1,2][3,3][4,4]$

$r_i = y_j \neq p_k$

(b)

(v,k)	$B_3(v,k)$
$(1,1),(2,2),(3,3)$	$[1,1]$
$(2,1),(3,2),(4,3)$	$[1,2]$
$(3,1),(4,2)$	$[1,2][3,3]$
$(4,1)$	$[1,2][3,3][4,4]$

$r_i = y_j = p_k$ with $k > \delta_P(k)$

FIGURE 9. Illustrations of b-interval sets. (a) $r_i = y_j \neq p_k$, (b) $r_i = y_j = p_k$ with $k > \delta_P(k)$.

where $S_i(v-1,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$ *and* t *is the largest index in* $[1,\lambda]$ *such that* $\mathcal{R}_{h_t}(v) > \mathcal{R}_v(v)$.

Proof. Clearly, $B_i(1,k) = \{[1,1]\}$ when $r_i = y_j \neq p_k$ and $v = 1$. Assume that $B_i(v-1,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$ for some $2 \leqslant v < \mu_\tau$. By Proposition 4.1, $\mathcal{R}_x(v-1) = \mathcal{R}_x(v)$ for $1 \leqslant x \leqslant v-w$ while $\mathcal{R}_x(v-1)+1 = \mathcal{R}_x(v)$ for $v-w+1 \leqslant x \leqslant v-1$. Furthermore, by Proposition 4.2, $\mathcal{R}_{h_x} > \mathcal{R}_{h_{x+1}}$ for $1 \leqslant x \leqslant \lambda$. Let t be the largest index in $[1,\lambda]$ such that $\mathcal{R}_{h_t}(v) > \mathcal{R}_v(v)$ is satisfied. Thus, $\mathcal{R}_{h_t}(v) > \mathcal{R}_v(v) \geqslant \mathcal{R}_{h_t+1}(v) > \mathcal{R}_{h_t+2}(v) > \cdots > \mathcal{R}_{v-1}(v)$. This results in $B_i(v,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_t,h_t],[f_{t+1},v]\}$. If no such t exists, then it is clear that $B_i(v,k) = \{[1,v]\}$. This completes the proof. □

LEMMA 4.3. *For the case where* $r_i = y_j = p_k$ *with* $k > \delta_P(k)$ *and* $j = Y_\tau(v)$,

$$B_i(v,k) = \begin{cases} \emptyset & \text{if } v < \alpha, \\ \{[1,1]\} & \text{if } v = \alpha, \\ \{[f_1,h_1],[f_2,h_2],\ldots, & \\ \quad [f_t,h_t],[f_{t+1},v-\alpha+1]\} & \text{if } v > 1 \text{ and } t \text{ exists}, \\ \{[1,v-\alpha+1]\} & \text{otherwise}, \end{cases}$$

where $B_i(v-1,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$ *and* t *is the largest index in* $[1,\lambda]$ *such that* $\mathcal{R}_{h_t}(v) > \mathcal{R}_{v-\alpha+1}(v)$.

Proof. By Lemma 4.1, $B_i(v,k) = \emptyset$ when $v < \alpha$. For the other cases, by using a similar argument as in Lemma 4.2, the lemma follows. □

Figure 9(a) and (b) is used to illustrate Lemmas 4.2 and 4.3, respectively.

LEMMA 4.4. *If* $B_i(v,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$, *then* $h_1 \geqslant v-w+1$.

Proof. By the non-decreasing property of the values in each row of a CLCS table and $\mathcal{R}_x = \mathcal{R}_{i-1,\text{pre}^1(Y_\tau(x)),d_{v-x+1}} + l_i$ for $1 \leqslant x \leqslant v-w$, this lemma follows directly. □

TABLE 1. $S_3(v,k)$.

	k		
v	1	2	3
1	$\emptyset$	[1, 1]	[1, 1]
2	$\emptyset$	[2, 2]	[2, 2]
3	$\emptyset$	[3, 3]	[2, 2][3, 3]
4	$\emptyset$	[4, 4]	[3, 3][4, 4]

COROLLARY 4.1. *For the case where* $r_i = y_j \neq p_k$ *or* $r_i = y_j = p_k$ *with* $k = \delta_P(k)+1$, *the critical* ω*-index of critical element* $(i, Y_\tau(v), k)$ *is* h_1, *where* $B_i(v,k) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$.

Lemma 4.5 describes how to find $S_i(v,k)$ for the case where $r_i = y_j = p_k$ with $k > \delta_P(k)$.

LEMMA 4.5. *For the case where* $r_i = y_j = p_k$ *with* $k > \delta_P(k)$ *and* $j = Y_\tau(v)$,

$$S_i(v,k) = \begin{cases} \emptyset & \text{if } \alpha = 1, \\ \{[v,v]\} & \text{if } \alpha = 2 \text{ or } v < \alpha, \\ \{[f_1,h_1],[f_2,h_2],\ldots, & \\ \quad [f_t,h_t],[f_{t+1},v]\} & \text{if } v \geqslant \alpha > 2 \text{ and } t \text{ exists}, \\ \{[f_1,v]\} & \text{otherwise}, \end{cases}$$

where $S_i(v-1,k-1) = \{[f_1,h_1],[f_2,h_2],\ldots,[f_\lambda,h_\lambda]\}$ *and* t *is the largest index in* $[1,\lambda]$ *such that* $\mathcal{R}_{h_t}(v) > \mathcal{R}_v(v)$.

Proof. By Lemma 4.1, $S_i(v,k) = \emptyset$ when $\alpha = 1$. Since all originating values of (i,j,k) are $-\infty$ when $v < \alpha$, we can obtain all its originating values through $\{[v,v]\}$. For the other cases, by using a similar argument as in Lemma 4.2, the lemma follows. □

Table 1 is used to illustrate Lemma 4.5 for finding $S_3(v,k)$.

LEMMA 4.6. *For critical element* $(i, Y_\tau(v), k)$, *if both* $b(v)$ *and* $s(v)$ *are* >0 *and* $\mathcal{R}_{h_1} < \mathcal{R}_{h_{b(v)+1}}$, *then* $h_{b(v)+1}$ *is its critical* ω*-index*; *otherwise*, h_1 *is its critical* ω*-index.*

Proof. By Proposition 4.2, $\mathcal{R}_{h_1} > \mathcal{R}_{h_2} > \cdots > \mathcal{R}_{h_{b(v)}}$ and $\mathcal{R}_{h_{b(v)+1}} > \mathcal{R}_{h_{b(v)+2}} > \cdots > \mathcal{R}_{h_{a(v)}}$. Clearly, if either $b(v)$ or $s(v)$ is equal to 0, then h_1 is the critical ω-index of $(i, Y_\tau(v), k)$. For the case where both $b(v)$ and $s(v)$ are >0, $\mathcal{R}_{i,j,k} = \max\{\mathcal{R}_{h_1}, \mathcal{R}_{h_2}, \ldots, \mathcal{R}_{h_{a(v)}}\} = \max\{\mathcal{R}_{h_1}, \mathcal{R}_{h_{b(v)+1}}\}$. Thus, if $\mathcal{R}_{h_1} > \mathcal{R}_{h_{b(v)+1}}$, then h_1 is the critical ω-index of $(i, Y_\tau(v), k)$; otherwise, $h_{b(v)+1}$ is the critical ω-index. This completes the proof. □

LEMMA 4.7. *The b- and s-interval sets of critical element (i, j, k) can be constructed in $O(1)$ amortized time.*

Proof. We prove only that constructing $B_i(v, k)$ for $1 \leqslant v \leqslant \mu_\tau$ can be done in $O(1)$ amortized time. The case for constructing $S_i(v, k)$ can be handled similarly.

Let $B_i(v-1, k) = \{[f_1, h_1], [f_2, h_2], \ldots, [f_\lambda, h_\lambda]\}$. Note that, initially, $B_i(1, k) = \{[1, 1]\}$ when $r_i = y_j \neq p_k$ or $B_i(\alpha, k) = \{[1, 1]\}$ when $r_i = y_j = p_k$ and $k > \delta_P(k)$. Let Φ be the potential function of $B_i(v, k)$ which represents the number of intervals in $B_i(v, k)$. It is obvious that $\Phi(B_i(1, k)) = 1$. Since the number of intervals in $B_i(v, k)$ is never empty for $v \geqslant 1$, $\Phi(B_i(v, k)) \geqslant \Phi(B_i(1, k))$. To find $B_i(v, k)$ from $B_i(v-1, k)$, either a comparison or merging a subset of intervals to be a new interval will be applied. If only a comparison is applied, i.e. $\mathcal{R}_{h_\lambda}(v) > \mathcal{R}_v(v)$, then a new interval $[v, v]$ is appended to $B_i(v-1, k)$ and we say that its cost is 1, denoted by $c_v = 1$. That is, $\Phi(B_i(v, k)) - \Phi(B_i(v-1, k)) = 1$. If merging a subset of m intervals to obtain $B_i(v, k)$ is applied, then we say that its cost is $c_v = m + 1$, i.e. m comparisons plus one merging. This results in $\Phi(B_i(v, k)) - \Phi(B_i(v-1, k)) = -m + 1$. For the former case, the amortized cost $\widehat{c_v}$ on constructing $B_i(v, k)$ is $\widehat{c_v} = c_v + \Phi(B_i(v, k)) - \Phi(B_i(v-1, k)) = 1 + 1 = 2$. For the latter case, the amortized cost on constructing $B_i(v, k)$ is $\widehat{c_v} = c_v + \Phi(B_i(v, k)) - \Phi(B_i(v-1, k)) = m + 1 - m + 1 = 2$. It can be seen that the amortized costs of both operations are $O(1)$. This completes the proof. □

Now we are at a position to describe our algorithm as follows.

Algorithm A

Input: An RLE string X and uncompressed strings Y and P in which $|Y| = n$, $|P| = r$, and X has M runs.
Output: The length of a CLCS of X, Y, and P.

```
begin
    Step 1. /* Initialization */
    R_{i,0,0} = R_{0,j,0} = 0 and R_{0,j,k} = R_{i,0,k} = -∞ for
    0 ⩽ i ⩽ M, 0 ⩽ j ⩽ n, and 1 ⩽ k ⩽ r;
    Step 2. /* computing R_{i,j,k} */
    Compute R_{i,j,k} for 1 ⩽ i ⩽ M, 1 ⩽ j ⩽ n, and 1 ⩽ k ⩽ r
    by using Theorem 4.2.
    Step 3. Output R_{M,n,r};
end
```

We summarize our result as the following theorem.

THEOREM 4.3. *Given strings X and Y, and a constraining string P, the CLCS of X and Y with respect to P can be found in $O(NMr + r \min\{mN, nM\})$ time.*

Proof. Assume that the RCL tables are constructed with X in RLE format. By using the formulas in Lemmas 4.2, 4.3 and 4.5, the values in the kth RCL table for $k = 0, 1, \ldots, r$ can be computed. By Lemmas 4.4, 4.6 and 4.7, every $\mathcal{R}_{i,j,k}$ for critical element (i, j, k) can be computed in $O(1)$ amortized time. Let f_1 be the number of mismatched faces and f_2 be the number of elements in partially and fully matched faces. The total time to find the length of a CLCS is $O(rf_1 + f_2)$. If we construct the RCL tables with Y in RLE format and f_3 is the number of elements in partially and fully matched faces, then the total time to find the length of a CLCS is $O(rf_1 + f_3)$. Therefore, the CLCS problem can be solved in $O(rf_1 + \min\{f_2, f_3\})$ time. Note that, in the worst case, $f_1 = NM$ and $\min\{f_2, f_3\} = r \min\{mN, nM\}$. □

5. CONCLUDING REMARKS

In this paper and in [7], we solve only the case where one of X and Y is in RLE form. Our algorithm for solving the CLCS problem can be done in $O(NMr + r \min\{mN, nM\})$ time which improves the best-known result in [7]. The reason is that the algorithm in [7] still needs to compute all $\mathcal{R}_{i,j,k}$ in the surface of fully matched cuboids. Note that, in describing the time complexity, the term NMr is the maximal number of prisms in a 3D DP lattice. It occurs only when all cuboids are mismatched cuboids. A challenge related to this problem is to consider the case where both X and Y are in RLE form, or even more difficult, all three strings are in RLE form.

FUNDING

This work was supported in part by the National Science Council of Republic of China under contracts NSC 98-2221-E-128-003-, NSC 99-2221-E-011-106-, NSC 100-2221-E-011-067-MY3 and NSC 101-2221-E-011-038-MY3.

REFERENCES

[1] Tsai, Y.T. (2003) The constrained common sequence problem. *Inf. Process. Lett.*, **88**, 173–176.

[2] Chin, F.Y.L., De Santis, A., Ferrara, A.L., Ho, N.L. and Kim, S.K. (2004) A simple algorithm for the constrained sequence problem. *Inf. Process. Lett.*, **90**, 175–179.

[3] Arslan, A.N. and Eğecioğlu, Ö. (2005) Algorithms for the constrained longest common subsequence problems. *Int. J. Found. Comput. Sci.*, **16**, 1099–1109.

[4] Iliopoulos, C.S. and Rahman, M.S. (2008) New efficient algorithms for the LCS and constrained LCS problems. *Inf. Process. Lett.*, **106**, 13–18.

[5] Sayoood, K. and Fow, E. (eds) (2000) *Introduction to Data Compression* (2nd edn). Morgan Kaufmann, Los Altos, CA.

[6] Ahsan, S.B., Aziz S.P. and Rahman, M.S. (2012) Longest Common Subsequence Problem for Run-Length-Encoded Strings. *International Conference on Computer and Information Technology (ICCIT)*, Chittagong, December 22–24, pp. 36–41.

[7] Ann, H.Y., Yang, C.B., Tseng, C.T. and Hor C.Y. (2012) Fast algorithms for computing the constrained LCS of run-length-encoded strings. *Theor. Comput. Sci.*, **432**, 1–9.

[8] Apostolico, A., Landau, G.M. and Skiena, S. (1999) Matching for run-length encoded strings. *J. Complexity*, **15**, 4–16.

[9] Arbell, O., Landau, G.M. and Mitchell, J.S.B. (2002) Edit distance of run-length encoded strings. *Inf. Process. Lett.*, **83**, 307–314.

[10] Bunke, H. and Csirik, J. (1995) An improved algorithm for computing the edit distance of run length coded strings. *Inf. Process. Lett.*, **54**, 93–96.

[11] Chen, K.Y., Hsu, P.H. and Chao, K.M. (2010) Hardness of comparing two run-length encoded strings. *J. Complexity*, **26**, 364–374.

[12] Chen, K.Y. and Chao, K.M. (2013) A fully compressed algorithm for computing the edit distance of run-length encoded strings. *Algorithmica*, **65**, 354–370.

[13] Freschi, V. and Bogliolo, A. (2004) Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Inf. Process. Lett.*, **90**, 167–173.

[14] Liu, J.J., Wang, Y.L. and Lee, R.C.T. (2008) Finding a longest common subsequence between a run-length-encoding string and an uncompressed string. *J. Complexity*, **24**, 173–184.

[15] Sakai, Y. (2012) *Computing the Longest Common Subsequence of two Run-Length Encoded Strings*, Algorithms and Computation: Lecture Notes in Computer Science 7676, pp. 197–206. Springer Berlin Heidelberg.

[16] Liu, J.J., Huang, G.S., Wang, Y.L. and Lee, R.C.T. (2007) Edit distance for a run-length-encoded string and an uncompressed string. *Inf. Process. Lett.*, **105**, 12–16.

[17] Ahsan, S.B., Moosa, T.M., Rahman, M.S. and Shahriyar, S. (2011) Computing a longest common subsequence of two strings when one of them is run length encoded. *INFOCOMP J. Comput. Sci.*, **10**, 48–55.