Theoretical Computer Science 432 (2012) 1–9

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Fast algorithms for computing the constrained LCS of run-length encoded strings[☆]

Hsing-Yen Ann, Chang-Biau Yang[*], Chiou-Ting Tseng, Chiou-Yi Hor

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

ARTICLE INFO

Article history:
Received 29 January 2011
Received in revised form 30 October 2011
Accepted 16 January 2012
Communicated by 'A. Apostolico'

Keywords:
Design of algorithms
Longest common subsequence
Run-length encoding
Constrained LCS

ABSTRACT

The constrained LCS (CLCS) problem, a recent variant of the longest common subsequence (LCS) problem, has gained much attention. Given two sequences X and Y of lengths n and m, respectively, and the constrained sequence P of length r, previous research shows that the CLCS problem can be solved by either an $O(nmr)$-time algorithm based upon dynamic programming (DP) techniques or an $O(r\mathcal{R}\log\log(n+m))$-time Hunt–Szymanski-like algorithm, where $\mathcal{R}$ is the total number of ordered pairs of positions at which the two strings match. In this paper, we investigate the case that X, Y and P are all in run-length encoded (RLE) format, where the numbers of runs are N, M and R, respectively. We first show that when the sequences are encoded, the CLCS problem can be solved by a simple algorithm in $O(nmR + nMr + Nmr)$ time without decompressing the sequences. Then, we propose a more efficient algorithm with $O(NMr + r \times \min\{q_1, q_2\} + q_3)$ time, where q_1 and q_2 denote the numbers of elements in the south and east faces of the matched blocks on the first layer, respectively, and q_3 denotes the number of face elements of all fully matched cuboids in the DP lattice.

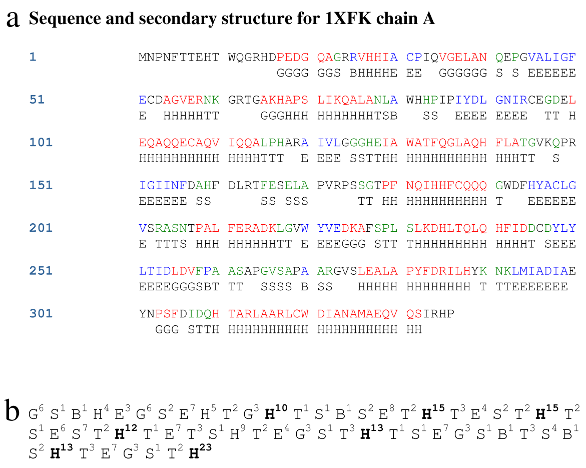
1. Introduction

The *longest common subsequence* (LCS) problem is a well-known measurement for computing the similarity of two strings. Given a sequence or string, a *subsequence* is formed by deleting zero or more elements arbitrarily. The LCS problem measures the length of the longest subsequence which is contained in the both given sequences. For example, let $X = badbcacd$ and $Y = adacbdbc$ be the two input strings. The LCS of X and Y is $adacd$. It can be widely applied in diverse areas, such as file comparison, pattern matching and computational biology. For example, if two DNA sequences are given, the longer the LCS is, the more similar the two DNA sequences are. The most referred algorithm to solve the LCS problem was proposed by Wagner and Fischer [24], which employs the *dynamic programming* (DP) technique. Other advanced algorithms have been proposed. Hirschberg reduced the space requirement from quadratic to linear space [13]. Hunt and Szymanski considered the smaller number of matching elements rather than the whole DP lattice [14]. Yang and Lee solved this problem with parallelism [25]. When the number of input sequences increases to k, $k \geq 3$, the problem is defined as the *multiple-sequence longest common subsequence* (k-LCS) problem. Since the k-LCS problem has been proved as NP-hard [17], Shyu and Tsai solved this problem by using ant colony optimization [21], and Blum et al. employed beam search to solve this problem heuristically [7].

[☆] A preliminary version of this paper was presented at the International Conference on Bioinformatics and Computational Biology, Las Vegas, USA, July, 2009.

[*] Corresponding author. Tel.: +886 7 5252000; fax: +886 7 5254301.
E-mail address: cbyang@cse.nsysu.edu.tw (C.-B. Yang).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.01.038



a **Sequence and secondary structure for 1XFK chain A**

```
1          MNPNFTTEHT WQGRHDPEDG QAGRRVHHIA CPIQVGELAN QEPGVALIGF
                            GGGG GGS BHHHHE EE  GGGGGG S S EEEEEE

51         ECDAGVERNK GRTGAKHAPS LIKQALANLA WHHPIPIYDL GNIRCEGDEL
           E  HHHHHTT     GGGHHH HHHHHHHTSB   SS  EEEE EEEE  TT H

101        EQAQQECAQV IQQALPHARA IVLGGGHEIA WATFQGLAQH FLATGVKQPR
           HHHHHHHHHH HHHHTTT  E EEE SSTTHH HHHHHHHHHH HHHTT  S

151        IGIINFDAHF DLRTFESELA PVRPSSGTPF NQIHHFCQQQ GWDFHYACLG
           EEEEEE SS      SS SSS      TT HH HHHHHHHHHH T   EEEEEE

201        VSRASNTPAL FERADKLGVW YVEDKAFSPL SLKDHLTQLQ HFIDDCDYLY
           E TTTS HHH HHHHHHTT E EEEGGG STT THHHHHHHHH HHHHT SEEE

251        LTIDLDVFPA ASAPGVSAPA ARGVSLEALA PYFDRILHYK NKLMIADIAE
           EEEEGGGSBT TT  SSSS B SS   HHHHH HHHHHHHH T TTEEEEEEE

301        YNPSFDIDQH TARLAARLCW DIANAMAEQV QSIRHP
             GGG STTH HHHHHHHHHH HHHHHHHHHH HH
```

b $G^6\ S^1\ B^1\ H^4\ E^3\ G^6\ S^2\ E^7\ H^5\ T^2\ G^3\ \mathbf{H}^{\mathbf{10}}\ T^1\ S^1\ B^1\ S^2\ E^8\ T^2\ \mathbf{H}^{\mathbf{15}}\ T^3\ E^4\ S^2\ T^2\ \mathbf{H}^{\mathbf{15}}\ T^2$
$S^1\ E^6\ S^7\ T^2\ \mathbf{H}^{\mathbf{12}}\ T^1\ E^7\ T^3\ S^1\ H^9\ T^2\ E^4\ G^3\ S^1\ T^3\ \mathbf{H}^{\mathbf{13}}\ T^1\ S^1\ E^7\ G^3\ S^1\ B^1\ T^3\ S^4\ B^1$
$S^2\ \mathbf{H}^{\mathbf{13}}\ T^3\ E^7\ G^3\ S^1\ T^2\ \mathbf{H}^{\mathbf{23}}$

Fig. 1. An example of the RLE format in bioinformatics. (a) The sequence and secondary structure of protein 1XFK chain A that is downloaded from PDB [6]. (b) The compressed string in RLE format.

1.1. *Constrained LCS (CLCS)*

Recently, the *constrained LCS* (CLCS) problem, a variant of the LCS problem, has gained much attention. Given two input sequences X, Y and a constrained sequence P, the CLCS problem is to find a common subsequence Z of X and Y such that P is a subsequence of Z and the length of Z is maximized. Tsai [22] first addressed the CLCS problem and proposed an algorithm to solve it in $O(n^2m^2r)$ time, where n, m and r denotes the lengths of X, Y and P, respectively. Two improved algorithms [4,10] based on the DP technique were presented independently for solving this problem with $O(nmr)$ time and space complexity. Iliopoulos and Rahman [15] proposed a Hunt–Szymanski-like algorithm [14] and employed a special data structure, van Emde Boas tree [23], to solve this problem in $O(r\mathcal{R}\log\log(n+m))$ time, where $\mathcal{R}$ is the total number of ordered pairs of positions at which the two strings match. The Hunt–Szymanski-like algorithm is very efficient when $\mathcal{R}$ is small. However, it should be noted that $\mathcal{R} = O(nmr)$ in the worst case. A more generalized version of the CLCS problem was presented by Peng et al. [19]. In the generalized version, some of the constraints might be ignored according to the constraint weights.

1.2. *Run-length encoding (RLE)*

Run-length encoding (RLE) is a well-known and simple method for compressing strings [20]. It divides a string into a sequence of *runs* where each run is a maximal repetitive substring of an identical symbol and it can be represented as a pair, the symbol and the length. For example, a string *aaaddccccbbbbbb* is encoded as $a^3d^2c^4b^6$ in the RLE format. Since the secondary structure of a protein usually repeats in a small region, the protein secondary structure sequence can be highly compressed [11] in the RLE format. For example, Fig. 1(a) shows the sequence and secondary structure of protein 1XFK chain A of length 346 that is downloaded from *Protein Data Bank* (PDB) [6]. The compressed sequence of the secondary structure in RLE format is shown in Fig. 1(b). It is mentionable that there are seven runs containing ten or more symbols and the length of the longest run is 23.

1.3. *The LCS problem of RLE strings*

Some researchers proposed the algorithms to solve the LCS problem for RLE strings efficiently without decompressing the strings [2,8,12,16,18]. Bunke and Csirik [8] first illustrated the concept that splits the DP lattice into *blocks* and they proposed an algorithm with $O(nM + Nm)$ time for solving this problem, where N and M represent the numbers of runs in the two given strings, respectively. Apostolico et al. [2] proposed an algorithm with $O(NM \log NM)$ time by maintaining the collection of *forced paths*. Liu et al. [16] proposed the concept of processing the elements of the DP lattice run by run rather than row by row, and the time complexity of their algorithm is $O(\min\{nM, Nm\})$. Ann et al. [1] proposed an efficient DP algorithm which combines the *forced path* of Apostolico et al. [2], *run-by-run processing* of Liu et al. [16] and optimal *range minimum query* (RMQ) algorithm of Bender and Farach-Colton [5]. Their algorithm outperforms the previous works [2,8,16] and requires $O(NM+min\{p_1, p_2\})$ time, where p_1 and p_2 denote the numbers of elements in the bottom and right boundaries of the matched blocks, respectively.

1.4. *Our results*

In this paper, we study how to compute the CLCS of the strings which are in RLE format. Similar to the concept of Section 1.3, our goal is to develop the algorithms which depend on the parameters such as the numbers of runs rather than the lengths of the uncompressed strings. We first propose a simple algorithm, which extends the concepts of dividing the DP lattice into blocks [8,12] and requires $O(nmR + nMr + Nmr)$ time. Then we propose a more efficient algorithm by invoking the RMQ technique. Its time complexity is $O(NMr + r \times \min\{q_1, q_2\} + q_3)$, where q_1 and q_2 denote the numbers of elements in the south and east faces of the matched blocks on the first layer, respectively, and q_3 denotes the number of face elements of all fully matched cuboids in the DP lattice.

The rest of this paper is organized as follows. In Section 2, we present the preliminaries about solving the CLCS problem of RLE strings. In Section 3, we first propose a simple algorithm to solve this problem, and then a more efficient algorithm is presented. And finally, the conclusion goes in Section 4.

2. Preliminaries

2.1. Notations

Let X, Y denote the two input strings and P denote the constrained sequence, where $X = x_1x_2 \cdots x_n$, $Y = y_1y_2 \cdots y_m$ and $P = p_1p_2 \cdots p_r$ over a finite alphabet Σ. The lengths of X, Y and P are denoted as n, m and r, respectively. If a string is of zero length, it is denoted by an empty string ϕ. The string X is run-length-encoded into N runs, $RX_1, RX_2, \ldots, RX_N$, where the lengths of the runs are denoted by $n_1, n_2, \ldots, n_N$, respectively. Similarly, the RLE strings of Y and P are encoded as $RY_1RY_2 \cdots RY_M$ and $RP_1RP_2 \cdots RP_R$, whose lengths are denoted by $m_1, m_2, \ldots, m_M$ and $r_1, r_2, \ldots, r_R$, respectively. A substring $x_ix_{i+1} \cdots x_j$ of X is denoted as $X_{i..j}$. A prefix $X_{1..i}$ of X is simply denoted as X_i. The consecutive runs $RX_i, RX_{i+1}, \ldots, RX_j$ is denoted as $RX_{i..j}$, which is also a substring of X. We denote the longest common subsequence Z of X and Y as $LCS(X, Y)$, whose length is denoted as $\mathcal{L}(X, Y)$. We also denote the CLCS of X and Y with respect to P as $CLCS(X, Y, P)$, whose length is denoted as $\mathcal{CL}(X, Y, P)$. For example, suppose that $X = badbcacd$, $Y = adacbdbc$ and $P = bbc$. Then, $LCS(X, Y) = adacd$, $\mathcal{L}(X, Y) = 5$, $CLCS(X, Y, P) = bdbc$, and $\mathcal{CL}(X, Y, P) = 4$.

The symbol of a given run RX_i is denoted by $\sigma(RX_i)$. For convenient algorithm description, let $\sigma(RX_0) = \sigma(RY_0) = \sigma(RP_0) = x_0 = y_0 = p_0 = \epsilon$, where ϵ is a dummy character that does not appear in X, Y, or P. A pointer of the prefix of string X which ends at the i'th element of the ith run is denoted by the couple $X(i; i')$. For easy representation, let $X(i; 0) = X(i-1; n_{i-1})$. That is, $X(i; 0)$ and $X(i-1; n_{i-1})$ point to the same position with different notations.

2.2. The properties of the CLCS problem

The traditional LCS problem (without any constraint) was solved with a DP algorithm on a 2-dimensional (2D) *lattice* by Wagner and Fischer [24]. The time and space complexities of their algorithm are both $O(nm)$. The algorithm is illustrated in Lemma 1.

Lemma 1 (*LCS*). *(See [24].) Given two strings, X and Y, and two distinct symbols a and b, the following conditions hold:*

1. $\mathcal{L}(X, \phi) = 0$.
2. $\mathcal{L}(\phi, Y) = 0$.
3. $\mathcal{L}(Xa, Ya) = \mathcal{L}(X, Y) + 1$.
4. $\mathcal{L}(Xa, Yb) = \max\{\mathcal{L}(Xa, Y), \mathcal{L}(X, Yb)\}$.

For example, it is easy to verify that $\mathcal{L}(badbca, ada) = \mathcal{L}(badbc, ad) + 1$ and $\mathcal{L}(badbca, adacb) = \max\{\mathcal{L}(badbca, adac), \mathcal{L}(badbc, adacb)\}$. The two well-known DP algorithms [4,10] for the CLCS problem, which are extended from Wagner and Fischer's DP algorithm, require a 3-dimensional (3D) DP lattice. The properties of the CLCS problem are presented in Lemma 2.

Lemma 2 (*Constrained LCS*). *(See [10].) Given two input strings X and Y, a constrained sequence P, and two distinct symbols a and b, the following conditions hold:*

1. $\mathcal{CL}(\phi, Y, P) = -\infty$ *if* $P \neq \phi$.
2. $\mathcal{CL}(X, \phi, P) = -\infty$ *if* $P \neq \phi$.
3. $\mathcal{CL}(X, Y, \phi) = \mathcal{L}(X, Y)$.
4. $\mathcal{CL}(Xa, Ya, Pa) = \mathcal{CL}(X, Y, P) + 1$.
5. $\mathcal{CL}(Xa, Ya, Pb) = \mathcal{CL}(X, Y, Pb) + 1$.
6. $\mathcal{CL}(Xa, Yb, P) = \max\{\mathcal{CL}(Xa, Y, P), \mathcal{CL}(X, Yb, P)\}$.

We use some examples to illustrate the above lemma. For Case 4, $\mathcal{CL}(badb, adacbdb, bb) = \mathcal{CL}(bad, adacbd, b) + 1$. For Case 5, $\mathcal{CL}(badbc, adacbdbc, bb) = \mathcal{CL}(badb, adacbdb, bb) + 1$. For Case 6, $\mathcal{CL}(ba, adacb, b) = \max\{\mathcal{CL}(ba, adac, b), \mathcal{CL}(b, adacb, b)\}$.

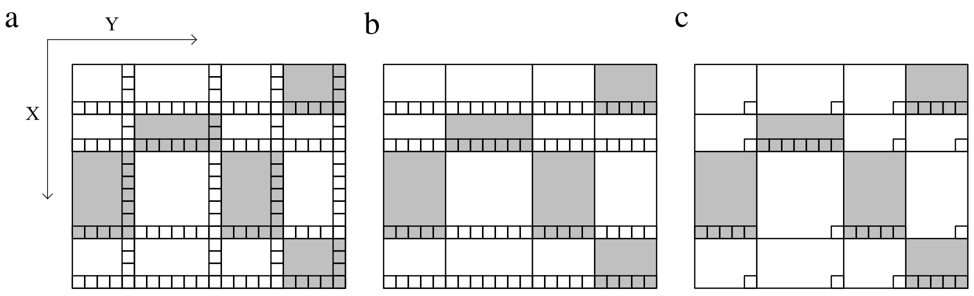


Fig. 2. (See [1].) The algorithms for solving the LCS problem on RLE strings by using the DP technique. (a) Bunke and Csirik [8]. (b) Liu et al. [16]. (c) Ann et al. [1].

2.3. The properties of the LCS problem for RLE strings

The following lemma illustrates the properties of the LCS problem for RLE strings.

Lemma 3 (*LCS for RLE*)**.** *(See [1,12].) Given two strings, X and Y, and two distinct symbols a and b, the following conditions hold:*

1. $\mathcal{L}(Xa^s, Ya^s) = \mathcal{L}(X, Y) + s$.
2. $\mathcal{L}(Xa^{s_1}, Ya^{s_2}) = \mathcal{L}(Xa^{s_1-s}, Ya^{s_2-s}) + s$, *where* $s = \min\{s_1, s_2\}$.
3. $\mathcal{L}(Xa^s, Yb^t) = \max\{\mathcal{L}(Xa^s, Y), \mathcal{L}(X, Yb^t)\}$.
4. *(Merged light blocks.)* $\mathcal{L}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}) = \max\{\mathcal{L}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Y), \mathcal{L}(X, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j})\}$, *where* $a_{i'} \neq b_{j'}$ *for each* $i' \in [1, i]$ *and* $j' \in [1, j]$.

We illustrate the above lemma with some examples as follows. For Case 1, $\mathcal{L}(a^3b^6c^4a^5, b^3a^8c^4b^8a^5) = \mathcal{L}(a^3b^6c^4, b^3a^8c^4b^8) + 5$. For Case 2, $\mathcal{L}(a^3b^6c^4a^{12}, b^3a^8c^4b^8a^5) = \mathcal{L}(a^3b^6c^4a^7, b^3a^8c^4b^8) + 5$. For Case 3, $\mathcal{L}(a^3b^6c^4a^{12}, b^3a^8c^4b^8) = \max\{\mathcal{L}(a^3b^6c^4a^{12}, b^3a^8c^4), \mathcal{L}(a^3b^6c^4\ , b^3a^8c^4b^8)\}$. For Case 4, $\mathcal{L}(a^3b^6c^4a^{12}, b^3a^8c^4b^8d^3) = \max\{\mathcal{L}(a^3b^6c^4a^{12}, b^3a^8c^4), \mathcal{L}(a^3b^6, b^3a^8c^4b^8d^3)\}$.

According to the facts shown in Lemma 3, the concept that splits the original DP lattice into *blocks* was proposed [8], where each block corresponds to a run pair of X and Y. If the symbol of run RX_i is identical to the symbol of run RY_j, we say that the block is a *matched* (gray) block; otherwise it is a *mismatched* (light) block. Fig. 2 shows examples for dividing the DP lattice.

Bunke and Csirik's algorithm [8] solves the LCS problem for RLE strings by the following recurrence formula, for any $0 \le i \le N$, $0 \le i' \le n_i$, $0 \le j \le M$ and $0 \le j' \le m_j$,

$$\mathcal{L}((i; i'), (j; j')) = \begin{cases} \mathcal{L}((i; i'-s), (j; j'-s)) + s, \ \text{where } s = \min\{i', j'\}, \\ \quad \text{if } \sigma(RX_i) = \sigma(RY_j), \\ \max\{\mathcal{L}((i; i'), (j-1; m_{j-1})), \mathcal{L}((i-1; n_{i-1}), (j; j'))\} \\ \quad \text{if } \sigma(RX_i) \neq \sigma(RY_j), \end{cases} \tag{1}$$

with boundary conditions $\mathcal{L}((i; i'), (0; 0)) = \mathcal{L}((0; 0), (j; j')) = 0$.

By applying the concept of block splitting, $\mathcal{L}(X, Y)$ can be computed according to the elements on the boundaries of the blocks, rather than the whole DP lattice. In other words, the value of each element inside the blocks is never evaluated, as shown in Fig. 2(a). Liu et al. proposed the concept to process the elements of the DP lattice run by run rather than row by row [16], and the number of computed elements is reduced, as shown in Fig. 2(b). Furthermore, Ann et al. [1] proposed an improved algorithm by extending the run-by-run concept [16] and employing an optimal RMQ algorithm [5] to find the values of the elements on each *forced path* [1,2]. It is mentionable that the optimal RMQ method applied in Ann et al.'s algorithm evaluates each required element in $O(1)$ time. The number of elements to be computed in Ann et al.'s algorithm is shown in Fig. 2(c).

3. Our algorithms

Fig. 3 shows an example of the CLCS problem, where $X = a^3b^6c^4a^{12}$, $Y = b^3a^8c^4a^5b^8c^4a^4$ and $P = ab$. In the lowest (first) layer, L_0, the operations on the DP lattice are the same as those for the traditional LCS problem (without any constraint). However, things change in the higher layers. In each higher layer, if the current symbols in X, Y and P are identical, the value of an element (a position on the DP lattice) depends on a corresponding element on the next lower layer. Otherwise, the value only depends on some elements on the same layer. Note that each element in the 3D DP lattice represents a 3-tuple (X_i, Y_j, P_k), where $0 \le i \le n$, $0 \le j \le m$ and $0 \le k \le r$. Therefore, the constrained LCS length $\mathcal{CL}(X_i, Y_j, P_k)$ can be denoted as $\mathcal{CL}(v)$ for an element v. For example, the values of elements u_0, u_1 and u_2 depend on the values of elements v_0, v_1 and v_2, respectively. That is, $\mathcal{CL}(u_0) = \mathcal{CL}(v_0) + 1$, $\mathcal{CL}(u_1) = \mathcal{CL}(v_1) + 1$ and $\mathcal{CL}(u_2) = \mathcal{CL}(v_2) + 1$.

In the following, we first propose a simple algorithm, which extends the concept of the previous works [8,12]. Then we will propose an improved algorithm, which employs the RMQ algorithm.

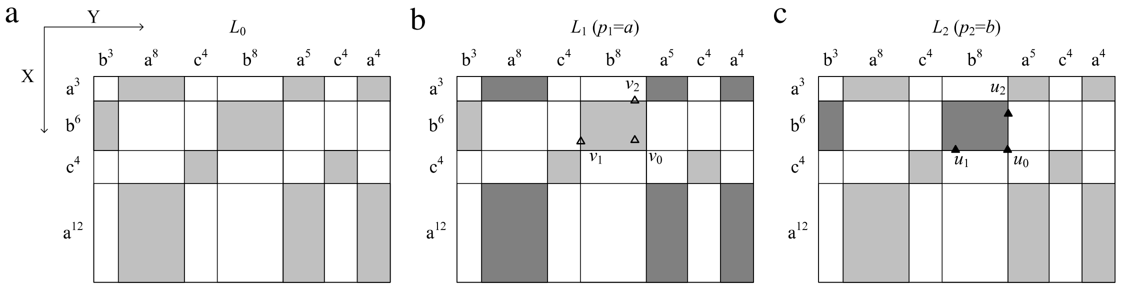


Fig. 3. An example of the DP lattice for solving the CLCS problem. (a) The lowest (first) layer, L_0. (b) *Layer*L_1, where $p_1 = a$. (c) Layer L_2, where $p_2 = b$.

3.1. A simple algorithm: algorithm CLCS1

Some additional properties of the CLCS problem for RLE strings can be easily deduced by combining Lemmas 2 and 3 as follows.

Corollary 1 (*Constrained LCS of RLE Strings*)**.** *Given two input strings X and Y, a constrained sequence P, and two distinct symbols a and b, the following conditions hold:*

1. $\mathcal{CL}(Xa^{s_1}, Ya^{s_2}, Pa^{s_3}) = \mathcal{CL}(Xa^{s_1-s}, Ya^{s_2-s}, Pa^{s_3-s}) + s$, *where* $s = \min\{s_1, s_2, s_3\}$.
2. $\mathcal{CL}(Xa^{s_1}, Ya^{s_2}, Pb^{t}) = \mathcal{CL}(Xa^{s_1-s}, Ya^{s_2-s}, Pb^{t}) + s$, *where* $s = \min\{s_1, s_2\}$.
3. $\mathcal{CL}(Xa^{s}, Yb^{t}, P) = \max\{\mathcal{CL}(Xa^{s}, Y, P), \mathcal{CL}(X, Yb^{t}, P)\}$.
4. *(Merged light blocks.)* $\mathcal{CL}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}, P) = \max\{\mathcal{CL}(Xa_1^{s_1}a_2^{s_2}\cdots a_i^{s_i}, Y, P), \mathcal{CL}(X, Yb_1^{t_1}b_2^{t_2}\cdots b_j^{t_j}, P)\}$, *where* $a_{i'} \neq b_{j'}$ *for each* $i' \in [1, i]$ *and* $j' \in [1, j]$.

According to the facts shown in Corollary 1, we propose a simple algorithm by directly extending the concept of block splitting proposed by Bunke and Csirik [8]. The recurrence formula of Algorithm CLCS1 is given as follows.

$$\mathcal{CL}((i;i'),(j;j'),(k;k')) = \begin{cases} \mathcal{CL}((i;i'-s),(j;j'-s),(k;k'-s)) + s, \text{ where } s = \min\{i',j',k'\}, \\ \quad \text{if } \sigma(RX_i) = \sigma(RY_j) = \sigma(RP_k), \\ \mathcal{CL}((i;i'-s),(j;j'-s),(k;k')) + s, \text{ where } s = \min\{i',j'\}, \\ \quad \text{if } \sigma(RX_i) = \sigma(RY_j) \neq \sigma(RP_k), \\ \max\{\mathcal{CL}((i;i'),(j-1;m_{j-1}),(k;k')), \mathcal{CL}((i-1;n_{i-1}),(j;j'),(k;k'))\}, \\ \quad \text{if } \sigma(RX_i) \neq \sigma(RY_j) \end{cases} \tag{2}$$

for $1 \le i \le N$, $1 \le j \le M$, $1 \le k \le R$, and that at least one of $i' = n_i$, $j' = m_j$ and $k' = r_k$ holds, and with boundary conditions,
$\mathcal{CL}((i;i'),(0;0),(0;0)) = \mathcal{L}((0;0),(j;j'),(0;0)) = 0$
and
$\mathcal{CL}((i;i'),(0;0),(k;k')) = \mathcal{L}((0;0),(j;j'),(k;k')) = -\infty$,
for $0 \le i \le N$, $0 \le i' \le n_i$, $0 \le j \le M$, $0 \le j' \le m_j$, $1 \le k \le R$, and $0 \le k' \le r_k$.

Note that the the DP lattice to be divided for the traditional LCS problem [8] is in 2D space, but the DP lattice for solving the CLCS problem is in 3D space. In our new algorithm, a 3D DP lattice is divided into $N \times M \times R$ *cuboids*, where each cuboid corresponds to a tuple of runs of X, Y and P. For example, cuboid $C_{i,j,k}$ is denoted as the tuple of RX_i, RY_j and RP_k. We call $C_{i,j,k}$ a *fully matched cuboid* (*black cuboid*) if $\sigma(RX_i) = \sigma(RY_j) = \sigma(RP_k)$, and a *partially matched cuboid* (*gray cuboid*) if $\sigma(RX_i) = \sigma(RY_j) \neq \sigma(RP_k)$. Otherwise, $C_{i,j,k}$ is called a *mismatched cuboid* (*light cuboid*). Meanwhile, the boundary of a cuboid is called a *face* in 3D space.

Fig. 3 shows an example of the CLCS problem of RLE strings, where $X = a^3b^6c^4a^{12}$, $Y = b^3a^8c^4a^5b^8c^4a^4$ and $P = ab$. If only one layer of the DP lattice is considered, the *black blocks*, *gray blocks* and *light blocks* represent the slices of the fully matched cuboids, partially matched cuboids and mismatched cuboids, respectively. The behaviors of the gray blocks and light blocks in Fig. 3 are the same as the matched blocks and mismatched blocks in Bunke and Csirik's algorithm [8]. That is, the elements in these blocks only depend on the other elements on the same layer. However, the elements in the black blocks depend on the lower layers.

For a convenient description of the calculation in our algorithm, we show the directions of the six faces of one cuboid, *north*, *south*, *east*, *west*, *top* and *bottom* in Fig. 4(b), and the three axes X, Y and P in 4(c). In Eq. (2), only the elements on the south, east and top faces of each cuboid have to be evaluated. The combination of these three faces with the faces of three neighboring cuboids form a hollow cuboid with thickness 1, as shown in Fig. 4(a). It is clear that under the *random access machine* (RAM) model, we can easily apply the technique proposed by Bunke and Csirik [8] to prevent the allocation and initialization of the elements which are never evaluated. Fig. 5 shows the elements that need be calculated in these three kinds of cuboids. The time complexity of Algorithm CLCS1 is shown in the following theorem.

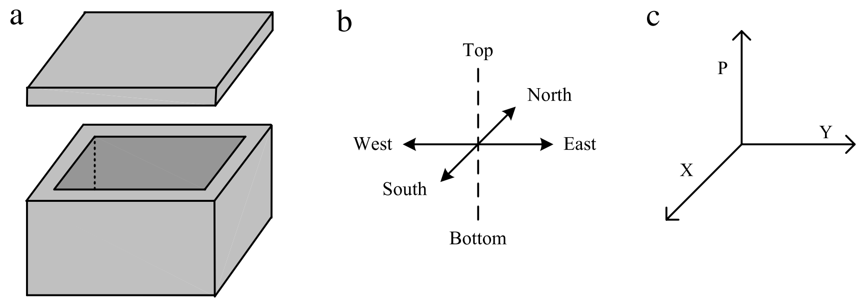


Fig. 4. The 3D view of one cuboid. (a) The elements that should be calculated in each hollow cuboid. (b) The directions of the six faces of one cuboid. (c) The directions of the three axes.

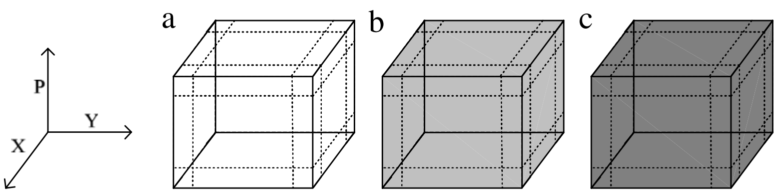


Fig. 5. Time complexity analysis of Algorithm CLCS1. (a) The elements calculated in a mismatched cuboid, which form a hollow cuboid with thickness 1. (b) The elements calculated in a partially matched cuboid, which form a hollow cuboid with thickness 1. (c) The elements calculated in a fully matched cuboid, which form a hollow cuboid with thickness 1.

Theorem 1. *Given two input sequences X, Y and a constrained sequence P with lengths n, m and r, respectively, which are encoded in RLE format into N, M and R runs, respectively, Algorithm CLCS1 computes the length of CLCS of X and Y with respect to P in $O(nmR + nMr + Nmr)$ time.*

Proof. In Eq. (2), for Case 1, at lease one of $i' - s, j' - s$ and $k' - s$ has value of 0, which means that a certain face element of the previous run in X, Y or P is used. Similarly, in the other two cases, only face elements are used for calculation. Thus, the calculation of the elements on the south, east and top faces is enough for solving the CLCS problem. The total number of these face elements is $O(nmR + nMr + Nmr)$. Besides, each calculation requires only constant time. This leads to the correctness of the theorem. □

3.2. An improved algorithm: algorithm CLCS2

By observing the 3D DP lattice, the following facts can be found.

Lemma 4 (*Two Kinds of Paths*)**.** *Given a layer of the DP lattice for the CLCS problem of RLE strings, the following conditions hold:*

1. *If the symbol of the current run RX_i in X is different from the current symbol in P, the blocks corresponding to the current run RX_i are either partially matched or mismatched.*
2. *If the symbol of the current run RX_i is identical to the current symbol in P, the blocks corresponding to the current run RX_i are either fully matched or mismatched.*

Fig. 6(a) illustrates an example for Case 1 of the above lemma, where the current symbol of P is b. The blocks corresponding to $RX_4 = a^{12}$ are either partially matched or mismatched. When the property of *merged light blocks* (Case 4 of Corollary 1) is applied, the destination u_1 of the big leap from u_0 is also on the boundary of a partially matched block, but never a fully matched block. Furthermore, the calculation of the south boundary elements of RX_4 never depends on the lower layer. That is, only the elements on the same layer are used for the calculation. In this situation, the computation job is completely the same as a 2D lattice. Thus, the optimal RMQ technique used in Ann et al.'s work [1] for the traditional LCS problem of RLE strings can be applied here.

We use the example shown in Fig. 6(a) to explain the idea of the RMQ technique used in Ann et al.'s work [1]. Suppose element v_0 is to be calculated. Conceptually, in the figure, each element is on either the east or south boundary. Here, we also use v_0 to denote the value of itself since there is no ambiguity. Only v_1 is needed to calculate v_0, that is $v_0 = v_1 + d_1$. To calculate v_1, two values v_2 and v_6 are needed, that is $v_1 = \max\{v_2, v_6\}$. Similarly, $v_2 = v_3 + d_2$, $v_3 = \max\{v_4, v_7\}$, $v_4 = \max\{v_5, v_8\}$ and $v_5 = u_0 + d_3$. The elements should be considered in the calculation of an element form a *forced path* [2]. For example, the forced path of v_o consists of v_1, v_2, v_3, v_4, v_5 and u_0.

By combining the above equations and removing some variables, we can get a clearer equation $v_0 = \max\{v_6 + d_1, v_7 + (d_1+d_2), v_8+(d_1+d_2), u_0+(d_1+d_2+d_3)\}$. u_0 can be further calculated by $\max\{u_1, u_2\}$. As one can see, each of $\{u_2, v_6, v_7, v_8\}$ is at the southeast corner of a certain block, and u_1 locates at the south boundary of a partially matched block. In summary,

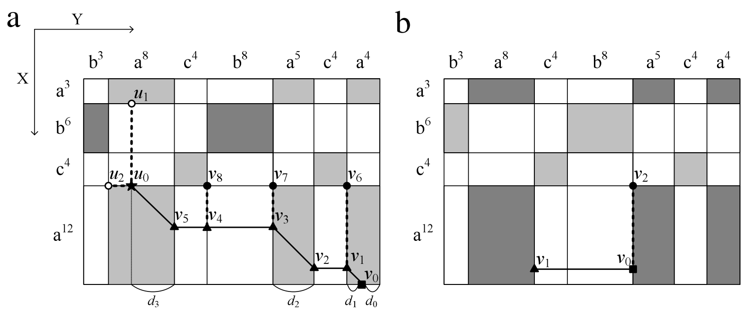


Fig. 6. The calculation of the elements on the boundaries. (a) The calculation of an element on the south boundary of a gray block, where the current symbol of *P* is *b*. (b) The calculation of the boundary element of a black block, where the current symbol of *P* is *a*.

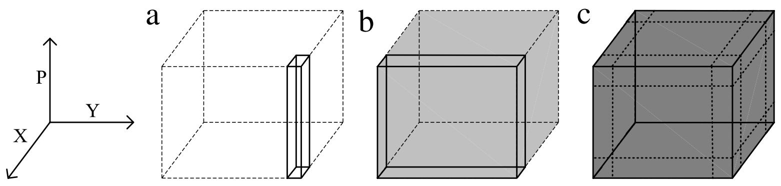


Fig. 7. Time complexity analysis of Algorithm CLCS2. (a) The elements calculated in a mismatched cuboid, which form a square prism with base 1 × 1. (b) The elements calculated in a partially matched cuboid, which form the south face of the cuboid with thickness 1 on the south. (c) The elements calculated in a fully matched cuboid, which form a hollow cuboid with thickness 1.

the elements needed to be calculated in the DP lattice for Case 1 of Lemma 4 include the south boundaries of all partially matched blocks and the southeast corners of the other blocks.

Let $C(v)$ denote the number of occurrences of the matched symbol in the east side of v. Then, $v_0 = \max\{v_6 + C(v_6) - d_0, v_7 + C(v_7) - d_0, v_8 + C(v_8) - d_0, u_0 + C(u_0) - d_0\} = \max\{v_6 + C(v_6), v_7 + C(v_7), v_8 + C(v_8), u_0 + C(u_0)\} - d_0$. One can see that different v_0 in the south boundary will have different value of d_0 and involve different range of elements on the boundaries of the north neighbor blocks. Since the elements on the boundaries of the north neighbor blocks have been calculated completely before they are used in the south boundary, the RMQ technique can be applied here. Thus, the above formula can be further reduced as $v_0 = \max\{ \text{RMQ}\{v_6 + C(v_6), v_7 + C(v_7), v_8 + C(v_8)\}, \max\{u_1, u_2\} + C(u_0)\} - d_0$. The calculation of each element requires only $O(1)$ time with the RMQ technique. Readers interested in formal description of the RMQ technique used in Ann et al.'s work can refer to their article [1].

However, as shown in Figs. 3 and 6(b), an element needs refer to the lower layer if it is located in a fully matched block (or cuboid). Such a calculation involves two layers, so it is difficult to directly apply the RMQ technique for computing the face elements in the fully matched cuboids. Nevertheless, we can still apply the property of *merged light blocks* (Case 4 of Corollary 1). This means that, to solve this problem, we have to evaluate all face elements of the fully matched cuboids. Fig. 7 shows the elements that need to be calculated in these three kinds of cuboids.

The formal description of Algorithm CLCS2 is given as follows.

Case 1 (southeast pillar of a light cuboid) $\mathcal{CL}((i; n_i), (j; m_j), (k; k')) = \max\{\mathcal{CL}((i-1; n_{i-1}), (j; m_j), (k; k')), \mathcal{CL}((i; n_i), (j-1; m_{j-1}), (k; k'))\}$, if $\sigma(RX_i) \neq \sigma(RY_j)$.

Case 2 (south face of a gray cuboid) Calculate $\mathcal{CL}((i; n_i), (j; j'), (k; k'))$ with the RMQ technique, if $\sigma(RX_i) = \sigma(RY_j) \neq \sigma(RP_k)$.

Case 3 (black cuboid) If $\sigma(RX_i) = \sigma(RY_j) = \sigma(RP_k)$, calculate $\mathcal{CL}((i; i'), (j; j'), (k; k'))$, where $i' = 1$ or $i' = n_i$ or $j' = 1$ or $j' = m_j$ or $k' = 1$ or $k' = r_k$.

Case 3.1 (south, east and top faces) $\mathcal{CL}((i; i'), (j; j'), (k; k')) = \mathcal{CL}((i; i'-s), (j; j'-s), (k; k'-s)) + s$, where $s = \min\{i', j', k'\} - 1$, if $i' = n_i$ or $j' = m_j$ or $k' = r_k$, but $i' \neq 1, j' \neq 1$ and $k' \neq 1$.

Case 3.2 (northwest pillar) $\mathcal{CL}((i; 1), (j; 1), (k; k')) = \mathcal{CL}((i-1; n_{i-1}), (j-1; m_{j-1}), (k; k'-1)) + 1$.

Case 3.3 (bottom face) $\mathcal{CL}((i; i'), (j; j'), (k; 1)) = \mathcal{CL}((i; i'-1), (j; j'-1), (k-1; r_{k-1})) + 1$, where $i', j' \geq 2$. $((i; i'-1), (j; j'-1), (k-1; r_{k-1}))$ locates at the top face of a gray cuboid and its value can be obtained by the RMQ method used in Case 2.

Case 3.4 (north face) $\mathcal{CL}((i; 1), (j; j'), (k; k')) = \mathcal{CL}((i-1; n_{i-1}), (j; j'-1), (k; k'-1)) + 1 = \max\{\mathcal{CL}((i-1; n_{i-1}), (j-1; m_{j-1}), (k; k'-1)), \mathcal{CL}((\hat{i}; n_{\hat{i}}), (j; j'-1), (k; k'-1))\} + 1$, where $j' \geq 2$, $0 \leq \hat{i} < i$, $\sigma(RX_{\hat{i}}) = \sigma(RP_k)$ and $\sigma(RP_k) \notin \{\sigma(RX_{\hat{i}+1}), \sigma(RX_{\hat{i}+2}), \ldots, \sigma(RX_{i-1})\}$.

Note that $((\hat{i}; n_{\hat{i}}), (j; j'-1), (k; k'-1))$ is located at the south face of a black cuboid when $k' \geq 2$, or the south face of a gray cuboid when $k' = 1$.

Case 3.5 (west face) $\mathcal{CL}((i; i'), (j; 1), (k; k')) = \mathcal{CL}((i; i'-1), (j-1; m_{j-1}), (k; k'-1)) + 1 = \max\{\mathcal{CL}((i-1; n_{i-1}), (j-1; m_{j-1}), (k; k'-1)), \mathcal{CL}((i; i'-1), (\hat{j}; m_{\hat{j}}), (k; k'-1))\} + 1$, where $i' \geq 2$, $0 \leq \hat{j} < j$,

$\sigma(RY_{\hat{j}}) = \sigma(RP_k)$ and $\sigma(RP_k) \notin \{\sigma(RY_{\hat{j}+1}), \sigma(RY_{\hat{j}+2}), \ldots, \sigma(RY_{j-1})\}$.

Note that $((i; i'-1), (\hat{j}; m_{\hat{j}}), (k; k'-1))$ is located at the east face of a black cuboid when $k' \geq 2$. It is located at the east face of a gray cuboid when $k' = 1$ and its value can be obtained by the RMQ method used in Case 2.

Case 4 (boundary conditions) $\mathcal{CL}((i; i'), (0; 0), (0; 0)) = \mathcal{L}((0; 0), (j; j'), (0; 0)) = 0$ and
$\mathcal{CL}((i; i'), (0; 0), (k; k')) = \mathcal{L}((0; 0), (j; j'), (k; k')) = -\infty$,
for $0 \leq i \leq N$, $0 \leq i' \leq n_i$, $0 \leq j \leq M$, $0 \leq j' \leq m_j$, $1 \leq k \leq R$, and $0 \leq k' \leq r_k$.

In Case 3.1, it computes the south, east and top faces, which will refer to the elements on the north, west or bottom faces. Cases 3.3, 3.4 and 3.5 calculate the elements on the bottom, north and west faces, respectively. In Case 3.3, $((i; i'-1), (j; j'-1), (k-1; r_{k-1}))$ is on the top face of run RP_{k-1}, where $\sigma(RX_i) = \sigma(RY_j) \neq \sigma(RP_{k-1})$. This top face is also a gray block on a 2D XY-plane, thus $\mathcal{CL}((i; i'-1), (j; j'-1), (k-1; r_{k-1}))$ can be calculated by the RMQ technique. Note that the calculation in Case 2 is to get the values of the elements on the south face of a gray cuboid. Obviously, the same RMQ technique can be applied to both Case 2 and Case 3.3. It is similar in Case 3.5 when $k' = 1$. The time complexity of Algorithm CLCS2 is given in the following theorem.

Theorem 2. *Given two input sequences X, Y and a constrained sequence P with lengths n, m and r, respectively, which are encoded in RLE format into N, M and R runs, respectively, Algorithm CLCS2 computes the length of the CLCS of X and Y with respect to P in $O(NMr + r \times \min\{q_1, q_2\} + q_3)$ time, where q_1 and q_2 denote the numbers of elements in the south and east faces of the partially matched blocks on the first layer L_0, respectively, and q_3 denotes the number of face elements of all fully matched cuboids in the DP lattice.*

Proof. In every case, the calculation for obtaining the value of one element requires $O(1)$ time. Thus, we can focus on the number of elements calculated in each case. For Case 1, only one element is calculated in a light block on an XY-plane. There are $O(NM)$ light blocks on a 2D XY-plane, and the height is r, so $O(NMr)$ elements are calculated. Case 2 computes the elements of the south face on the 3D view (boundary of a gray block on a 2D plane). It is clear that there are q_1 such elements. If the number of elements on the east face, denoted by q_2, is less than q_1, we can exchange the roles of X and Y to make the south face have fewer elements. Thus, the calculation in Case 2 requires $O(r \times \min\{q_1, q_2\})$ time. In Case 3.4, a matched table that can be built in advance helps us to find the previous match in string X with $O(1)$ time. Case 3.5 is similar to Case 3.4. In Cases 3.3 and 3.5, the RMQ method is invoked once to calculate one element, which requires $O(1)$ time. Case 3 computes all elements of the six faces of all fully matched cuboids, and thus it requires $O(q_3)$ time. The preprocessing time for constructing an RMQ data structure of n elements is $O(n)$. So, the total time required for RMQ construction do not exceed the total time required for the element calculation. Finally, we get the time complexity in this theorem. □

It is clear that the number of calculated elements in Algorithm CLCS2 is less than that in Algorithm CLCS1. One can also observe this fact by comparing Figs. 5 and 7.

4. Conclusion

In this paper, we study how to solve the CLCS problem more efficiently if the input strings and the constrained sequence are compressed in RLE format. For this purpose, we propose two algorithms for solving the CLCS problem without decompressing the encoded strings. Our algorithms calculate only the subset of the elements in the original 3D DP lattice. We first propose a simple algorithm, which requires $O(nmR + nMr + Nmr)$ time. Then we propose an improved algorithms by applying the RMQ technique used in Ann et al.'s work [1]. The time complexity of this algorithm is $O(NMr + r \times \min\{q_1, q_2\} + q_3)$ time, where q_1 and q_2 denote the numbers of elements in the south and east faces of the matched blocks on the first layer, respectively, and q_3 denotes the number of face elements of all fully matched cuboids in the DP lattice. The improvement can be obviously seen by comparing Figs. 5 and 7.

In the future, our work will be to find more properties of the cuboids, which may be useful to avoid the unnecessary calculation of the cuboid faces. There are still various kinds of constraints that may be applied to the LCS problem, such as the regular expression constraint [3], string-inclusion constraint and sequence-exclusion constraint [9]. We are also interested in how our concepts are applied to these variants.

Acknowledgement

This research work was partially supported by the National Science Council of Taiwan under contract NSC-97-2221-E-110-064.

References

[1] 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, Information Processing Letters 108 (2008) 360–364.
[2] A. Apostolico, G.M. Landau, S. Skiena, Matching for run-length encoded strings, Journal of Complexity 15 (1) (1999) 4–16.
[3] A.N. Arslan, Regular expression constrained sequence alignment, Journal of Discrete Algorithms 5 (4) (2007) 647–661.

[4] A.N. Arslan, Ö. Eğecioğlu, Algorithms for the constrained longest common subsequence problems, International Journal of Foundations Computer Science 16 (6) (2005) 1099–1109.
[5] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, 2000, pp. 88–94.
[6] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissing, I.N. Shindyalov, P.E. Bourne, The protein data bank, Nucleic Acids Research 28 (2000) 235–242.
[7] C. Blum, M.J. Blesa, M. López-Ibáñez, Beam search for the longest common subsequence problem, Computers and Operations Research 36 (12) (2009) 3178–3186.
[8] H. Bunke, J. Csirik, An improved algorithm for computing the edit distance of run-length coded strings, Information Processing Letters 54 (2) (1995) 93–96.
[9] Y.C. Chen, K.M. Chao, On the generalized constrained longest common subsequence problems, Journal of Combinatorial Optimization 21 (3) (2011) 383–392.
[10] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Information Processing Letters 90 (4) (2004) 175–179.
[11] M.Y. Eltabakh, W.K. Hon, W.A.R. Shah, J.S. Vitter, The SBC-tree: An index for run-length compressed sequences, in: Proceedings of the 11th International Conference on Extending Database Technology, Nantes, France, 2008, pp. 523–534.
[12] V. Freschi, A. Bogliolo, Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism, Information Processing Letters 90 (2004) 167–173.
[13] D.S. Hirschberg, Algorithms for the longest common subsequence problem, Journal of the ACM 24 (4) (1977) 664–675.
[14] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, Communications of the ACM 20 (5) (1977) 350–353.
[15] C.S. Iliopoulos, M.S. Rahman, New efficient algorithms for the LCS and constrained LCS problems, Information Processing Letters 106 (1) (2008) 13–18.
[16] 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, Journal of Complexity 24 (2) (2008) 173–184.
[17] D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the ACM 25 (1978) 322–336.
[18] J.S.B. 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 Stony Brook, 1997.
[19] Y.-H. Peng, C.-B. Yang, K.-S. Huang, K.-T. Tseng, An algorithm and applications to sequence alignment with weighted constraints, International Journal of Foundations of Computer Science 21 (1) (2010) 51–59.
[20] K. Sayoood, E.F. (Eds.), Introduction to Data Compression, second ed., Morgan Kaufmann, Los Altos, CA, 2000.
[21] S.J. Shyu, C.-Y. Tsai, Finding the longest common subsequence for multiple biological sequences by ant colony optimization, Computers and Operations Research 36 (1) (2009) 73–91.
[22] Y.T. Tsai, The constrained longest common subsequence problem, Information Processing Letters 88 (4) (2003) 173–176.
[23] P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, Information Processing Letters 6 (3) (1977) 80–82.
[24] R. Wagner, M. Fischer, The string-to-string correction problem, Journal of the ACM 21 (1) (1974) 168–173.
[25] C.B. Yang, R.C.T. Lee, Systolic algorithm for the longest common subsequence problem, Journal of the Chinese Institute of Engineers 10 (6) (1987) 691–699.