A Diagonal-Based Algorithm for the Constrained Longest Common Subsequence Problem

Siang-Huai Hung[1], Chang-Biau Yang[1](✉), and Kuo-Si Huang[2]

[1] National Sun Yat-sen University, Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

[2] National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract. Given two sequences A and B of lengths m and n, respectively, and another constrained sequence C with length r, the *constrained longest common subsequence* (CLCS) problem is to find the *longest common subsequence* (LCS) of A and B with the constraint that C is contained as a subsequence in the answer. Based on the diagonal concept for finding the LCS length, proposed by Nakatsu *et al.*, this paper proposes an algorithm for obtaining the CLCS length efficiently in $O(rL(m-L))$ time and $O(mr)$ space, where L denotes the CLCS length. According to the experimental result, the proposed algorithm outperforms the previously CLCS algorithms.

Keywords: Longest common subsequence · Constrained LCS · Diagonal-based strategy

1 Introduction

The goal of the *longest common subsequence* (LCS) problem is to calculate the identity or similarity of two sequences. It can be applied to several fields, such as bioinformatics, file plagiarism, and voice recognition [7].

A *subsequence* is gotten by deleting zero or more symbols from the original sequence or string. Given two sequences (or strings) $A = a_1a_2a_3 \cdots a_m$ and $B = b_1b_2b_3 \cdots b_n$, $|A| = m \leq n = |B|$, the LCS problem is to find the common subsequence of A and B whose length is the longest. A lot of algorithms have been proposed for solving the LCS problem [6,8]. Several variations of the LCS problem have also been proposed, such as the *constrained longest common subsequence* (CLCS) problem [3,5,11] and the *merged longest common subsequence* (MLCS) problem [12].

Given two sequences $A = a_1a_2a_3 \cdots a_m$, $B = b_1b_2b_3 \cdots b_n$, $|A| = m \leq n = |B|$, and a constrained sequence $C = c_1c_2c_3 \cdots c_r$, $|C| = r$, the *constrained longest*

This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 104-2221-E-110-018-MY3.

common subsequence (CLCS) problem, proposed by Tsai [11], is to find the LCS of A and B such that C is contained as a subsequence in the answer. Tsai [11] presented a *dynamic programming* (DP) algorithm with $\mathrm{O}(m^2n^2r)$ time and space. Peng also presented a DP algorithm with $\mathrm{O}(mnr)$ time and space [9]. Chin *et al.* proved that the CLCS problem is equivalent to a special case of the *constrained multiple sequence alignment* (CMSA) problem and presented a DP algorithm with $\mathrm{O}(mnr)$ time and space [3].

Arslan and Eğecioğlu [2] improved the algorithm complexity of Tsai from $\mathrm{O}(m^2n^2r)$ to $\mathrm{O}(mnr)$ by modifying the recurrence formula. Deorowicz [4] rewrote the DP formula of Chin *et al.* and applied the approach proposed by Hunt and Szymanski [6], then proposed an algorithm in $\mathrm{O}(r(mL'+R)+n)$ time, where $L' = |LCS(A,B)|$ and R is the total number of match pairs between A and B. Deorowicz's algorithm is more efficient when the alphabet size is large.

In 2010, Peng *et al.* [10] proposed an algorithm in $\mathrm{O}(mnr)$ time and $\mathrm{O}(nr)$ space for the weighted CPSA (WCPSA) problem. In the WCPSA problem, some constraints can be ignored by adopting proper constraint weights. In 2012, Ann *et al.* [1] proposed an efficient algorithm for the run-length encoded (RLE) sequences, which are often used to represent the secondary structure of proteins, in $\mathrm{O}(mnR_C + mNr + Mnr)$ time, where M, N and R_C are the numbers of runs of A, B and C, respectively. In their algorithm, the *range minimum querying* (RMQ) technique is invoked. Ho *et al.* [5] observed that the values of most corresponding CLCS lattice cells are identical in two consecutive layers when the alphabet size is small. They proposed an algorithm to avoid redundant computation by clarifying whether the lattice cells need to be calculated or not.

The organization of this paper is as follows. Section 2 introduces some preliminaries of the CLCS problem. In Sect. 3, we will propose a new algorithm, based on the diagonal scheme, for solving the CLCS problem. The diagonal scheme is inspired by the LCS algorithm proposed by Nakatsu *et al.* [8]. In Sect. 4, we show the experimental results of our algorithm and compare its efficiency to some previously published algorithms. Finally, the conclusion is given in Sect. 5.

2 Preliminaries

A sequence (string) $S = s_1s_2s_3\cdots s_{|S|}$ is composed of characters over a finite alphabet Σ. The sequence notations are described as follows.

- s_i: the ith character of sequence S.
- $|S|$: the length of sequence S.
- $i..j$: the range from indexes i to j.
- $S_{i..j}$: the substring of sequence S from indexes i to j. We set that $S_{i..j} = \emptyset$ if $j < i$.

2.1 The Longest Common Subsequence Problem

A *subsequence* is gotten by deleting zero or more character from the original sequence. Given two sequences $A = a_1a_2a_3\cdots a_m$ and $B = b_1b_2b_3\cdots b_n$, where

$m \leq n$, the LCS problem is to find the longest common subsequence of A and B. In 1974, Wagner and Fischer [13] proposed an algorithm with O(mn) time and space to solve the LCS problem by the *dynamic programming* (DP) approach.

Nakatsu *et al.* [8] proposed an algorithm to solve the LCS problem in $O(n(m-L'))$ time, where $L' = |LCS(A,B)|$. Evidently, if L' is close to m, the time complexity tends to be linear. That is, the algorithm is very efficient when the two sequences are very similar. For the prefix $A_{1..i}$ of A, let $j = d_{i,s}$ denote the smallest index j of B such that $|LCS(A_{1..i}, B_{1..j})| = s$. Nakatsu *et al.* [8] used $d_{i,s}$ to solve the LCS problem as follows.

$$d_{i,s} = \begin{cases} \infty & \text{if } i = 0, \\ \min\{j, d_{i-1,s}\} & \text{if there exists a smallest index} j \text{ such that} \\ & a_i = b_j \text{ and } j > d_{i-1,s-1} \text{ for } \ s \geq 2, \\ d_{i-1,s} & \text{ if there is no such } j. \end{cases} \tag{1}$$

2.2 The Constrained Longest Common Subsequence Problem

Given two sequences A, B, and a constrained sequence $C = c_1 c_2 c_3 \cdots c_r$, the *constrained longest common subsequence* (CLCS) problem is to find the LCS of A and B such that C is also a subsequence in this LCS. Let the solution of the *constrained longest common subsequence* (CLCS) problem be denoted as $CLCS(A,B,C)$. For example, suppose that A = `ccdbbcbdcd`, B = `dccbbcdbcb`, and C = `db`. We have $LCS(A,B)$ = `ccbbcbc` and $|LCS(A,B)| = 7$. When C is considered, $CLCS(A,B,C)$ = `dbbcdc` and $|CLCS(A,B,C)| = 6$. It is clear that $|LCS(A,B)| \geq |CLCS(A,B,C)|$.

3 The Diagonal-Based Algorithm

The proposed CLCS algorithm is inspired by the LCS algorithm of Nakatsu *et al.* [8]. The time and space complexities of our algorithm are $O(rL(m-L))$ and $O(mr)$, respectively, where $L = |CLCS(A,B,C)|$, $|A| = m$, $|B| = n$, $m \leq n$, and $|C| = r$. Accordingly, it performs well for the short CLCS since L is small, or for similar sequences since L is close to m.

Definition 1. *Let $D_{i,l}$, $i,l \geq 1$, denote a set of 3-tuple elements $\langle i', j, h \rangle$, $i' \leq i$ such that the constrained common subsequence of length l can be obtained, where $a_{i'} = b_j$, i' and j are the match indexes of A and B, the suffix $C_{r-h+1..r}$ (with length h) has not been included yet in the solution.*

By Definition 1, a 3-tuple element $\langle i', j, h \rangle \in D_{i,l}$ means the prefix $C_{1..k}$ has been included in the solution, $h = r - k$, that is, $|CLCS(A_{1..i'}, B_{1..j}, C_{1..r-h})| \geq l$. With the same feasible solution length, we prefer smaller values of j and h, because it requires a shorter prefix of B and includes more constraints in C. Accordingly, we define the *domination* concept as follows.

Definition 2. *For any two 3-tuple elements $\langle i_1, j_1, h_1 \rangle$ and $\langle i_2, j_2, h_2 \rangle$, where $\langle i_1, j_1, h_1 \rangle \neq \langle i_2, j_2, h_2 \rangle$, we say that $\langle i_1, j_1, h_1 \rangle$ dominates $\langle i_2, j_2, h_2 \rangle$ if ($j_1 \leq j_2$ and $h_1 \leq h_2$) or ($i_1 < i_2$ and $j_1 = j_2$ and $h_1 = h_2$).*

We use one array *limitB* and one matrix $NextMatch_B$ [12] to improve the efficiency for finding $|CLCS(A, B, C)|$. $limitB[k]$ records the largest index j such that $c_k = b_j$ and and there is no feasible solution for $c_k = b_{j'}$, $j' > j$. On the boundaries, we set $limitB[0] = 0$. For example, suppose that $B =$ `dccbbcdbcb`, $limitB = \{0, 7, 10\}$ for $C =$ `db`, and $limitB = \{0, 7, 8, 9\}$ for $C =$ `dbc`. $limitB[k]$ can be built by Eq. 2.

$$limitB[k] = \begin{cases} 0 & \text{if } k = 0, \\ \max\{j | c_k = b_j\} & \text{if } k = r, \\ \max\{j | c_k = b_j, j < limitB[k+1]\} & \text{if } 1 \le k \le r-1. \end{cases} \tag{2}$$

For each $\langle i', j', h' \rangle \in D_{i-1,l-1}$, we extend it to $\langle i, j, h \rangle \in D_{i,l}$ if it can get a longer CLCS candidate. We denote this operation as $\text{Extension}(\langle i', j', h' \rangle) = \langle i, j, h \rangle$. Some properties are described as follows.

(1) Naturally, $i \ge i' + 1$. If $i > m$, then $D_{i,l} = \emptyset$.
(2) $j = NextMatch_B[a_i][j']$. If $j > n$, then $\langle i, j, h \rangle$ is not valid and it cannot be appended into $D_{i,l}$.
(3) If $a_i = b_j = c_{r-h'+1}$, we update $h = h' - 1$; otherwise, $h = h'$.
(4) $\langle i, j, h \rangle$ is valid if $j \le limitB[r - h + 1]$. Otherwise, it is not valid and it cannot be appended into $D_{i,l}$.

After gathering $D_{i,l}$ from the extension of $D_{i-1,l-1}$, we combine $D_{i-1,l}$ and current $D_{i,l}$ together, and then perform the domination operation on this composite set to obtain a new $D_{i,l}$, where every pair of 3-tuple elements are not dominated by each other. This procedure is denoted as $D_{i,l} = \text{Domination}(\text{Extension}(D_{i-1,l-1}) \cup D_{i-1,l})$. Note that $D_{i,0} = \{\langle 0, 0, r \rangle\}$ for initialization. Table 1 shows an example of $D_{i,l}$ for this procedure.

Lemma 1. *If there exists* $\langle i', j, h \rangle \in D_{i,l}$, $1 \le i' \le i$ *and* $l \ge 1$, *we get* $|CLCS(A_{1..i'}, B_{1..j}, C_{1..r-h})| \ge l$, *where* $D_{i,l} = \text{Domination}(\text{Extension}(D_{i-1,l-1}) \cup D_{i-1,l})$ *and* $D_{0,0} = D_{i,0} = \{\langle 0, 0, r \rangle\}$.

Proof. By the properties of Extension, each $\langle i', j, h \rangle \in D_{i,l}$ should be valid. If $\text{Extension}(D_{i-1,l-1} = \langle i'', j', h' \rangle) = \langle i', j, h \rangle$, $i' \ge i'' + 1$, j is the next match index and $j > j'$, then the length of a feasible solution in $D_{i,l}$ is the length in $D_{i-1,l-1}$ plus one. So $|CLCS(A_{1..i'}, B_{1..j}, C_{1..r-h})| \ge l$.

Theorem 1. *If* $\langle i', j, 0 \rangle \in D_{i,l}$ *and there is no other* $\langle i'', j'', 0 \rangle \in D_{i,l}$, $i, l \ge 1$, $i' \ne i''$ *and* $j \ne j''$, $|CLCS(A_{1..i'}, B_{1..j}, C)| = l$, *where* $D_{i,l} = \text{Domination}(\text{Extension}(D_{i-1,l-1}) \cup D_{i-1,l})$ *and* $D_{0,0} = D_{i,0} = \{\langle 0, 0, r \rangle\}$.

Proof. By Lemma 1, $|CLCS(A_{1..i'}, B_{1..j}, C)| \ge l$. By Definition 1, we know $h = 0$ means the constrained sequence C has been included in the feasible solution. Definition 2 ensures $\langle i', j, 0 \rangle$ is better than $\langle i'', j'', 0 \rangle$. We conclude $|CLCS(A_{1..i'}, B_{1..j}, C)| = l$.

Algorithm 1. Computing the length of CLCS.

```
Input: Sequences $A = a_1a_2a_3 \dots a_m$, $B = b_1b_2b_3 \dots b_n$ and $C = c_1c_2c_3 \dots c_r$
Output: Length of $CLCS(A, B, C)$
 1: Construct the arrays of $limitB$ and $NextMatch_B$
 2: $L \leftarrow 0$                                              ▷ $L = |CLCS(A, B, C)|$
 3: if any $limitB[k] = 0$, $1 \leq k \leq r$ then return 0
 4: for $t = 1 \rightarrow m$ do                                  ▷ round $t$
 5:     $D_{t-1,0} \leftarrow \{\langle 0, 0, r \rangle\}$, $l \leftarrow 1$
 6:     for $i = t \rightarrow m$ do
 7:         for each $\langle i', j', h' \rangle \in D_{i-1,l-1}$ do
 8:             $j \leftarrow NextMatch_B[a_i][j']$
 9:             $k \leftarrow r - h' + 1$
10:             if $j \leq n$ and $h' = 0$ then
11:                 Insert $\langle i, j, 0 \rangle$ into $D_{i,l}$
12:             else if $j \leq limitB[k]$ then
13:                 Insert $\langle i, j, h' - 1 \rangle$ into $D_{i,l}$ if $a_i = c_k$
14:                 Insert $\langle i, j, h' \rangle$ into $D_{i,l}$ if $a_i \neq c_k$
15:         $D_{i,l} \leftarrow$ DOMINATION($D_{i,l} \cup D_{i-1,l}$)
16:         if $D_{i,l} = \emptyset$ then
17:             break
18:         Find the largest $L$ such that $\langle i'', j'', 0 \rangle \in D_{i,L}$
19:         $l \leftarrow l + 1$
20:     if $((m - t) \leq L)$ then
21:         return $L$
22: for $l = L \rightarrow 0$ do
23:     if $\langle i'', j'', 0 \rangle \in D_{i,l}$ then return $l$
24: return 0
```

Function 1. Domination.

```
 1: function DOMINATION(set $D$)
 2:     Initialize: $list[0..r] \leftarrow \emptyset$
 3:     for each $\langle i, j, h \rangle \in D$ do                  ▷ $list[h]$ stores one $\langle i, j, h \rangle$
 4:         if $list[h] = \emptyset$ then
 5:             Insert $\langle i, j, h \rangle$ into $list[h]$
 6:         else if $\langle i, j, h \rangle$ dominates $\langle i', j', h \rangle \in list[h]$ then
 7:             Remove $\langle i', j', h \rangle$ from $list[h]$ and insert $\langle i, j, h \rangle$ into $list[h]$
 8:     Set $D \leftarrow \emptyset$
 9:     $p \leftarrow 0$, $q \leftarrow 1$
10:     while $p \leq r$ and $q \leq r$ do                      ▷ $list[p]$ compares with $list[q]$
11:         if $\langle i, j, p \rangle \in list[p]$ dominates $\langle i', j', q \rangle \in list[q]$ then
12:             Remove $\langle i', j', q \rangle$ from $list[q]$
13:             $q \leftarrow q + 1$
14:         else
15:             $p \leftarrow q$, $q \leftarrow q + 1$
16:     Insert each $\langle i, j, h \rangle \in list[h]$ into $D$
17:     return $D$
```

Table 1. An example for the construction of $D_{i,l}$ with A = `ccdbbcbdcd`, B = `dccbbcdbcb` and C = `db`, where a strikethrough symbol means that the 3-tuple element is dominated and thus removed.

Round t	Length(l)							
	0	1	2	3	4	5	6	7
1	$D_{0,0}$	$D_{1,1}$	$D_{2,2}$	$D_{3,3}$	$D_{4,4}$	$D_{5,5}$		
	$\langle 0,0,2\rangle$	$\langle 1,2,2\rangle$	$\langle 2,3,2\rangle$	$\langle 3,7,1\rangle$	$\langle 4,8,0\rangle$	$\langle 5,10,0\rangle$		
2	$D_{1,0}$	$D_{2,1}$	$D_{3,2}$	$D_{4,3}$	$D_{5,4}$	$D_{6,5}$	$D_{7,6}$	
	$\langle 0,0,2\rangle$	$\langle 1,2,2\rangle$	$\langle 3,7,1\rangle$	$\langle 4,8,0\rangle$	$\langle 4,8,0\rangle$	$\langle 6,9,0\rangle$	$\langle 7,10,0\rangle$	
		~~$\langle 2,2,2\rangle$~~	$\langle 2,3,2\rangle$	$\langle 3,7,1\rangle$	~~$\langle 5,8,0\rangle$~~	~~$\langle 5,10,0\rangle$~~		
				$\langle 4,4,2\rangle$	~~$\langle 5,10,0\rangle$~~	$\langle 6,6,2\rangle$		
					$\langle 5,5,2\rangle$			
3	$D_{2,0}$	$D_{3,1}$	$D_{4,2}$	$D_{5,3}$	$D_{6,4}$	$D_{7,5}$	$D_{8,6}$	$D_{9,7}$
	$\langle 0,0,2\rangle$	$\langle 3,1,1\rangle$	$\langle 4,4,0\rangle$	~~$\langle 4,8,0\rangle$~~	~~$\langle 4,8,0\rangle$~~	~~$\langle 6,9,0\rangle$~~	$\langle 7,10,0\rangle$	$\langle 9,9,1\rangle$
		~~$\langle 1,2,2\rangle$~~	~~$\langle 3,7,1\rangle$~~	$\langle 5,5,0\rangle$	$\langle 6,6,0\rangle$	$\langle 7,8,0\rangle$	$\langle 8,7,1\rangle$	
			$\langle 2,3,2\rangle$	~~$\langle 3,7,1\rangle$~~	$\langle 5,5,2\rangle$	$\langle 6,6,2\rangle$		
				$\langle 4,4,2\rangle$	~~$\langle 6,6,2\rangle$~~			
				~~$\langle 5,4,2\rangle$~~				
4	$D_{3,0}$	$D_{4,1}$	$D_{5,2}$	$D_{6,3}$	$D_{7,4}$	$D_{8,5}$	$D_{9,6}$	$D_{10,7}$
	$\langle 0,0,2\rangle$	$\langle 3,1,1\rangle$	$\langle 4,4,0\rangle$	$\langle 5,5,0\rangle$	$\langle 6,6,0\rangle$	~~$\langle 7,8,0\rangle$~~	~~$\langle 7,10,0\rangle$~~	$\langle 9,9,1\rangle$
		~~$\langle 4,4,2\rangle$~~	~~$\langle 5,4,0\rangle$~~	~~$\langle 6,6,0\rangle$~~	~~$\langle 7,8,0\rangle$~~	$\langle 8,7,0\rangle$	$\langle 9,9,0\rangle$	
			$\langle 2,3,2\rangle$	$\langle 4,4,2\rangle$	$\langle 5,5,2\rangle$	$\langle 6,6,2\rangle$	$\langle 8,7,1\rangle$	
				~~$\langle 6,6,2\rangle$~~	~~$\langle 7,5,2\rangle$~~	~~$\langle 8,7,1\rangle$~~		

Theorem 2. *Algorithm 1 solves the CLCS problem in $O(rL(m-L))$ time and $O(mr)$ space.*

Proof. The algorithm terminates when it finds $|CLCS(A,B,C)| = L$. The number of rounds (t) is no more than $m-L$. The inner loop is executed at most L times. $|D_{i,l}| \le r$, because of the domination property. It means that operation EXTENSION is performed at most r times in each cell. Hence, the time complexity of Algorithm 1 is $O(rL(m-L))$. Since $|D_{i,l}| \le r$, the space required in each cell is $O(r)$. The space in different rounds can be reused. Thus, the space complexity of the algorithm is $O(Lr) = O(mr)$.

4 Experimental Results

This section presents the experimental results of the proposed algorithm compared with the algorithms of Chin *et al.* [3], Arslan and Eğecioğlu [2], Deorowicz [4], and Ho *et al.* [5]. The experimental environment is a computer running 64-bit Windows 7 OS with 3.30 GHz CPU (Intel Core i5-4590) and 8 GB RAM. These algorithms are implemented in C++ by Code::Blocks 13.12 with the GNU compiler collection GCC 4.8.1. In the experiment, the *similarity* of two sequences A and B is defined in Eq. 3.

$$similarity(A, B) = \frac{LCS(A, B)}{\min\{|A|, |B|\}}. \tag{3}$$

The program is executed 100 times for each parameter combination: $|C| \in \{2, 4\}$, $|\Sigma| = 256$, and $|A| = |B| = 1000$. Figure 1 shows that our algorithm has better performance than other algorithms obviously. Furthermore, our algorithm needs fewer execution time when the similarities more than 70% or less than 20%. With higher similarity, the number of match pairs between A and B increases, hence our algorithm will find the feasible solution faster. On the other hand, with lower similarity, the number of match pairs between A and B decreases, our algorithm needs to compute fewer match pairs, it reduces the computation time.

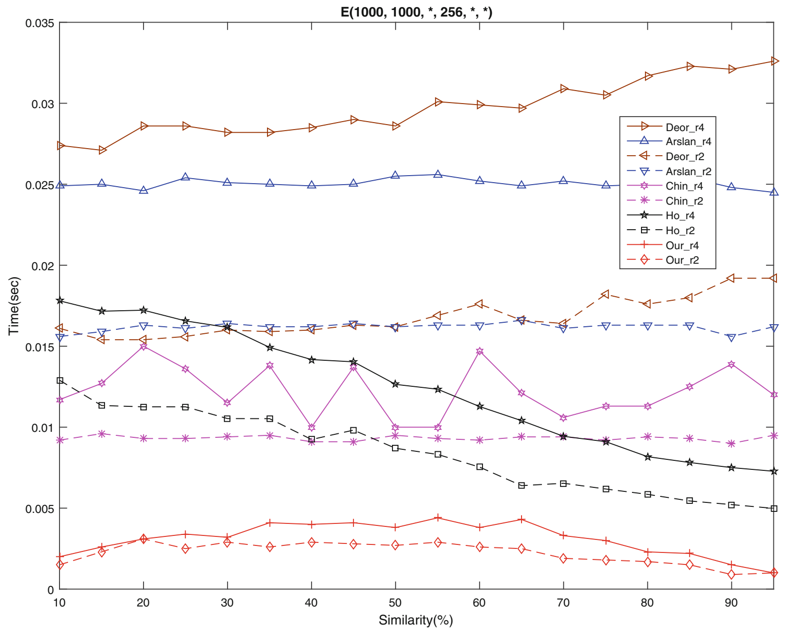

Fig. 1. The execution time (in seconds) for various similarities from 10% to 95% with $|A| = 1000$, $|B| = 1000$, $|C| \in \{2, 4\}$, and $|\Sigma| = 256$. `Chin_r4` represents the algorithm of Chin *et al.* with $|C| = 4$ and `Our_r2` represents our algorithm with $|C| = 2$.

5 Conclusion

This paper proposes a diagonal-based algorithm for solving the constrained longest common subsequence (CLCS) problem in $O(rL(m - L))$ time and in $O(mr)$ space, where m, r, and L denote the lengths of input sequences A, C, and CLCS length, respectively. As the experimental results show, our algorithm requires less execution time while the number of match pairs of A and B is small or their similarity is higher than 70%. The proposed algorithm uses $O(mr)$ space to store the 3-tuple elements for finding $|CLCS(A, B, C)|$. In the future, it is worthy of discussing the properties of the 3-tuple elements between neighboring cells for reducing redundant calculation.

References

1. Ann, H.Y., Yang, C.B., Tseng, C.T., Hor, C.Y.: Fast algorithms for computing the constrained LCS of run-length encoded strings. Theor. Comput. Sci. **432**, 1–9 (2012)
2. Arslan, A.N., Eğecioğlu, Ö.: Algorithms for the constrained longest common subsequence problems. Int. J. Found. Comput. Sci. **16**(06), 1099–1109 (2005)
3. Chin, F.Y.L., Santis, A.D., Ferrara, A.L., Ho, N.L., Kim, S.K.: A simple algorithm for the constrained sequence problems. Inform. Process. Lett. **90**(4), 175–179 (2004)
4. Deorowicz, S.: Fast algorithm for the constrained longest common subsequence problem. Theor. Appl. Inform. **19**(2), 91–102 (2007)
5. Ho, W.C., Huang, K.S., Yang, C.B.: A fast algorithm for the constrained longest common subsequence problem with small alphabet. In: Proceedings of the 34th Workshop on Combinatorial Mathematics and Computation Theory, Taichung, Taiwan, pp. 13–25 (2017)
6. Hunt, J.W., Szymanski, T.G.: A fast algorithm for computing longest common subsequences. Commun. ACM **20**(5), 350–353 (1977)
7. Kruskal, J.B.: An overview of sequence comparison: time warps, string edits, and macromolecules. SIAM Rev. **25**(2), 201–237 (1983)
8. Nakatsu, N., Kambayashi, Y., Yajima, S.: A longest common subsequence algorithm suitable for similar text strings. Acta Inform. **18**, 171–179 (1982)
9. Peng, C.L.: An approach for solving the constrained longest common subsequence problem. Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan (2003)
10. Peng, Y.H., Yang, C.B., Huang, K.S., Tseng, K.T.: An algorithm and applications to sequence alignment with weighted constraints. Int. J. Found. Comput. Sci. **21**, 51–59 (2010)
11. Tsai, Y.T.: The constrained longest common subsequence problem. Inform. Process. Lett. **88**, 173–176 (2003)
12. Tseng, K.T., Chan, D.S., Yang, C.B., Lo, S.F.: Efficient merged longest common subsequence algorithms for similar sequences. Theor. Comput. Sci. **708**, 75–90 (2018)
13. Wagner, R., Fischer, M.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)