The Merged Longest Common Increasing Subsequence Problem

Chien-Ting Lee[1], Chang-Biau Yang[1(✉)], and Kuo-Si Huang[2]

[1] Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw

[2] Department of Business Computing, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan
huangks@nkust.edu.tw

Abstract. In this paper, we first define the merged longest common increasing subsequence (MLCIS) problem, a composite variant combining the longest common subsequence (LCS) problem and longest increasing subsequence (LIS) problem. Given a pair of sequences A and B, along with a target sequence T, the goal of the MLCIS problem is to find the subsequence with the maximal length that is both common and increasing in both $E(A, B)$ and T. Here, $E(A, B)$ denotes any new sequence obtained by arbitrarily merging A and B while preserving their original orders. We propose a dynamic programming algorithm for solving the MLCIS problem. The time complexity of our algorithm is $O(mnr)$, where m, n and r represent the lengths of sequences A, B and T, respectively.

Keywords: longest increasing subsequence · merged longest common subsequence · merged longest common increasing subsequence · dynamic programming

1 Introduction

The well-known *longest common subsequence* (LCS) problem has been widely studied in the past decades [5,15]. Measuring the identity or the similarity of two sequences is often used in real-life situations such as bioinformatics, speech recognition and plagiarism detection [8]. Recognizing the diverse applications of LCS, researchers have explored several variants, including *multiple longest common subsequence* [9], *longest common increasing subsequence* (LCIS) [16], and *merged longest common subsequence* (MLCS) [6] problems.

A dynamic programming (DP) algorithm with both time and space complexities $O(mn)$ was proposed by Wagner and Fischer [15] to solve the LCS problem

This research work was partially supported by National Science and Technology Council of Taiwan under contract NSTC 112-2221-E-110-026-MY2.

N. T. Nguyen et al. (Eds.): ACIIDS 2024, CCIS 2144, pp. 202–212, 2024.
https://doi.org/10.1007/978-981-97-5937-8_17

in 1974. Based on the dominant matches of A and B, Hunt and Szymanski proposed an LCS algorithm in 1977 [7], with time complexity $O((R+m)\log m)$, where R denotes the total number of match pairs of positions in A and B. This algorithm demonstrates high efficiency when R is small. Nakatsu *et al.* [10] introduced a diagonal method with time complexity is $O(n(m-L))$, where L represents the LCS length in the answer. When L is close to m, the algorithm improves efficiency.

The *longest common increasing subsequence* (LCIS) was first defined by Yang *et al.* in 2005 [16]. They devised a dynamic programming (DP) algorithm in $O(mn)$ time and space. Subsequently, Sakai [13] improved Yang *et al.*'s algorithm [16] by reducing the space complexity from $O(mn)$ to $O(m+n)$ using the divide-and-conquer concept of Hirschberg [5]. Brodal *et al.* [1] proposed an algorithm in $O((n+mL)\log\log|\Sigma| + \textsc{Sort}(n))$ time and $O(n)$ space, where L denotes the LCIS length and Σ denotes the alphabet set of A and B. Chan *et al.* [3] presented a faster algorithm in $O(\min\{R\log L, nL+R\}\log\log n+\textsc{Sort}(n))$ time for that the number of match pairs R is relatively small. In 2018, Cai *et al.* [2] proposed an algorithm based on a recursive formula, achieving a time complexity of $O(mn)$ and a space complexity of $O(m+n)$. In 2020, Lo *et al.* [8] designed an algorithm in $O(n+L(m-L)\log\log|\Sigma|)$ time and $O(n)$ space. The algorithm exhibits high efficiency when L is either very small or close to m.

Huang *et al.* [6] first defined the *merged longest common subsequence* (MLCS) problem in 2008. The MLCS problem aims to determine the LCS between the merged sequence $E(A,B)$ and the target sequence T, where $E(A,B)$ is any sequence obtained by merging the sequences A and B in an arbitrary manner while maintaining their original orders. They presented a DP algorithm with time complexity $O(mnr)$, where $r=|T|$. An algorithm with time complexity $O(Lmr)$ was proposed by Peng *et al.* in 2010 [11]. In 2013, Deorowicz and Danek [4] proposed a bit-parallel algorithm in $O(\lceil r/w\rceil mn\log w)$ time, where w represents the word size of a computer. In 2014, Rahman and Rahman [12] presented an algorithm with $O((Pn+Qm)\log\log m)$ time, where P denotes the number of matching pairs between A and T, and Q denotes the number of matching pairs between B and T. They utilized a bounded heap to reduce the time complexity. In 2018, Tseng *et al.* [14] designed an algorithm in $O(n|\Sigma|+(r-L+1)Lm)$ time based on the diagonal approach.

The previous algorithms for solving the LCIS and the MLCS problems are summarized in Tables 1 and 2, respectively.

In this paper, we first define the *merged longest common increasing subsequence* (MLCIS) problem, and then propose an efficient algorithms to address it. Given a pair of sequences $A=\langle a_1, a_2, a_3, \cdots, a_m\rangle$ and $B=\langle b_1, b_2, b_3, \cdots, b_n\rangle$, along with a target sequence $T=\langle t_1, t_2, t_3, \cdots, t_r\rangle$, the MLCIS problem aims to find the common subsequence of $E(A,B)$ and T with the maximal length such that the common subsequence is increasing in both $E(A,B)$ and T. Here, $E(A,B)$ denotes any sequence obtained by arbitrarily merging A and B while keeping their original orders. For example, suppose that $A=\langle 2,5,4,8\rangle$, $B=\langle 7,4,1,8,7\rangle$ and $T=\langle 2,7,4,5,9,7,8\rangle$. The MLCIS answer is $\langle 2,4,5,7,8\rangle$

Table 1. The time and space complexities of the previous LCIS algorithms for two input sequences A and B. $m = |A|$, $n = |B|$; L: LCIS length; R: number of dominant matches of A and B; Σ: alphabet set [8].

LCIS				
Year	Author(s)	Time	Space	Notes
2005	Yang *et al.* [16]	$O(mn)$	$O(mn)$	Dynamic programming
2006	Sakai [13]	$O(mn)$	$O(m+n)$	Divide-and-conquer
2006	Brodal *et al.* [1]	$O((n+mL)\log\log\lvert\Sigma\rvert + \textsc{Sort}(n))$	$O(n)$	Divide-and-conquer, bounded heap, van Emde Boas tree
2007	Chan *et al.* [3]	$O(\min\{R\log L, nL+R\}\log\log n + \textsc{Sort}(n))$	$O(R)$	Match pair, binary search, van Emde Boas tree
2018	Cai *et al.* [2]	$O(mn)$	$O(m+n)$	Divide-and-conquer
2020	Lo *et al.* [8]	$O((n+L(m-L))\log\log\lvert\Sigma\rvert)$	$O(n)$	Diagonal, van Emde Boas tree

Table 2. The time and space complexities of the previous MLCS algorithms for two input sequences A and B as well as the target sequence T. $m = |A|$, $n = |B|$, $r = |T|$; L: MLCS length; R: number of dominant matches of A and B; w: word size of the computer; P: number of matching pairs between A and T; Q: number of matching pairs between B and T; Σ: alphabet set [14].

MLCS				
Year	Author(s)	Time	Space	Notes
2008	Huang *et al.* [6]	$O(mnr)$	$O(mn)$	Dynamic programming
2010	Peng *et al.* [11]	$O(Lmr)$	$O(n+Lm)$	Dynamic programming
2013	Deorowicz and Danek [4]	$O(\lceil r/w\rceil mn\log w)$	$\Theta(\lceil r/w\rceil mn)$	Bit-parallel
2014	Rahman and Rahman [12]	$O((Pn+Qm)\log\log m)$	$\Theta(\max\{nr, m\})$	Bounded heap
2018	Tseng *et al.* [14]	$O(n\lvert\Sigma\rvert + (r-L+1)Lm)$	$O(n\lvert\Sigma\rvert + Lm)$	Diagonal

with length 5, where $\langle a_1 = 2, b_2 = 4, a_2 = 5, b_5 = 7, a_4 = 8\rangle$. We propose a dynamic algorithms in $O(mnr)$ time for effectively solving the MLCIS problem.

This paper is organized as follows. We introduce the background knowledge in Sect. 2. Next, in Sect. 3, we propose a DP algorithm for solving the MLCIS problem. Section 4 provides a detailed example to illustrate our algorithm. At the end, conclusions and potential future work are given in Sect. 5.

2 Preliminaries

A subsequence of a sequence is formed from the original sequence by removing zero or some elements without destroying the relative order of the remaining

elements in the original sequence. Given two sequences $A = \langle a_1, a_2, a_3, \cdots, a_m \rangle$ and $B = \langle b_1, b_2, b_3, \cdots, b_n \rangle$, the LCS refers to the common subsequence of both A and B with the maximal length. Without loss of generality, it is assumed that $m \leq n$. As an illustration, $A = \langle \mathtt{a}, \mathtt{t}, \mathtt{g}, \mathtt{g}, \mathtt{t}, \mathtt{c} \rangle$ and $B = \langle \mathtt{c}, \mathtt{a}, \mathtt{g}, \mathtt{a}, \mathtt{c}, \mathtt{t} \rangle$, the LCS length of A and B is 3, which may be $\langle \mathtt{a}, \mathtt{g}, \mathtt{t} \rangle$ or $\langle \mathtt{a}, \mathtt{g}, \mathtt{c} \rangle$. Note that there may exist more than one LCS of the same length.

Suppose two numeric sequences $A = \langle a_1, a_2, \cdots, a_m \rangle$ and $B = \langle b_1, b_2, \cdots, b_n \rangle$ are given, where $m \leq n$. A subsequence $S = \langle s_1, s_2, \cdots, s_l \rangle$ is a *common increasing subsequence* (CIS) of A and B, where $a_{i_1} = b_{j_1} = s_1$, $a_{i_2} = b_{j_2} = s_2$, $\cdots$, $a_{i_l} = b_{j_l} = s_l$, $i_1 < i_2 < \cdots < i_l$, $j_1 < j_2 < \cdots < j_l$, and $s_1 < s_2 < \cdots < s_l$. The *longest common increasing subsequence* (LCIS) of A and B is any common increasing subsequence with the maximum length. For example, given $A = \langle 3, 1, 9, 6, 3, 4 \rangle$, $B = \langle 6, 1, 9, 3, 6, 7, 2, 4 \rangle$, the LCIS answer is $\langle 1, 3, 4 \rangle$ with length 3.

Yang *et al.* [16] proposed a dynamic programming algorithm in both $O(mn)$ time and space for solving the LCIS problem. Inspired by the diagonal concept of LCS algorithm proposed by Nakatsu *et al.* [10], Lo *et al.* [8] proposed a diagonal method for solving the LCIS problem, with time complexity $O((n + L(m - L)) \log \log |\Sigma|)$. That is, the algorithm is highly efficient when the two sequences are either highly dissimilar or extremely similar.

Given two sequences $A = \langle a_1, a_2, a_3, \cdots, a_m \rangle$ and $B = \langle b_1, b_2, b_3, \cdots, b_n \rangle$, along with a target sequence $T = \langle t_1, t_2, t_3, \cdots, t_r \rangle$, the *merged LCS* (MLCS) problem [6] is represented as MLCS(A, B, T). The MLCS problem is to find the longest common subsequence of $E(A, B)$ and T, where, $E(A, B)$ denotes a sequence built by merging A and B while preserving their original orders. Let $E(A, B) = \langle e_1, e_2, e_3, \cdots, e_{m+n} \rangle$. We define $C = c_{l_1} c_{l_2} c_{l_3} \cdots c_{l_m} = A$ as a subsequence of $E(A, B)$, where $1 \leq l_1 < l_2 < \cdots < l_m \leq m + n$. The remaining sequence, $E(A, B) \setminus C$, is precisely equal to B. Without loss of generality, we assume that $m \leq n$.

For example, suppose that $A = \langle \mathtt{g}, \mathtt{a}, \mathtt{t} \rangle$, $B = \langle \mathtt{a}, \mathtt{t}, \mathtt{g}, \mathtt{a} \rangle$ and $T = \langle \mathtt{a}, \mathtt{g}, \mathtt{c}, \mathtt{a}, \mathtt{t}, \mathtt{a} \rangle$. The MLCS answer is $\langle \mathtt{a}, \mathtt{g}, \mathtt{a}, \mathtt{t}, \mathtt{a} \rangle$ with length 5.

A DP algorithm was proposed by Huang *et al.* [6] with time and space complexities of $O(mnr)$ and $O(mn)$, respectively. Let $G(i, j, k)$ denote the length of MLCS$(A_{1..i}, B_{1..j}, T_{1..k})$. Their DP formula for MLCS is presented in Eq. 1.

$$G(i,j,k) = \max \begin{cases} G(i-1,j,k-1)+1 \text{ if } a_i = t_k; \\ G(i,j-1,k-1)+1 \text{ if } b_j = t_k; \\ \max \begin{cases} G(i-1,j,k) \\ G(i,j-1,k) \\ G(i,j,k-1) \end{cases} \text{ if } a_i \neq t_k \text{ or } b_j \neq t_k. \end{cases} \tag{1}$$

Tseng *et al.* [14] proposed a diagonal method to solve the MLCS problem in $O(n|\Sigma| + (r - L + 1)Lm)$ time. Their algorithm is highly efficient when L is either very small or close to r.

3 The Proposed Dynamic Programming Algorithm

The *merged longest common increasing subsequence* (MLCIS) problem is a generalized extension of the LCIS problem combined with the MLCS problem.

Definition 1 (MLCIS). *Given a pair of sequences* $A = \langle a_1, a_2, \cdots, a_m \rangle$ *and* $B = \langle b_1, b_2, \cdots, b_n \rangle$*, along with a target sequence* $T = \langle t_1, t_2, \cdots, t_r \rangle$*, the* merged longest common increasing subsequence (*MLCIS*) *problem is to find the longest common increasing subsequence of* $E(A, B)$ *and* T*, where* $E(A, B)$ *is a sequence obtained by merging* A *and* B *arbitrarily with preserving their original orders in* A *and* B *individually.*

Note that $E(A, B)$ is not a function; it merely describes an arbitrarily merging operation. Thus, there are many possible results for obtaining $E(A, B)$. For example, consider two sequences $A = \langle 2, 5, 4, 8 \rangle$, $B = \langle 7, 4, 1, 8, 7 \rangle$ and a target sequence $T = \langle 2, 7, 4, 5, 9, 7, 8 \rangle$. $E(A, B)$ could be $\langle 2, 5, 4, 8, 7, 4, 1, 8, 7 \rangle$, $\langle 2, 7, 4, 5, 4, 1, 8, 7, 8 \rangle$, or others. We get the MLCIS answer $\langle 2, 4, 5, 7, 8 \rangle$, whose length is 5. More precisely, in the answer $\langle a_1 = 2, b_2 = 4, a_2 = 5, b_5 = 7, a_4 = 8 \rangle$, we observe that $\langle a_1 = 2, a_2 = 5, a_4 = 8 \rangle$ is from A and $\langle b_2 = 4, b_5 = 7 \rangle$ comes from B.

In the DP algorithm for MLCIS, two tables α and β are employed to represent the MLCIS length ending at elements of A and B, respectively. To establish the boundary conditions, we add dummy elements a_0, b_0, and t_0 in front of sequences A, B, and T, respectively. Here, $a_0 = b_0 = t_0 = \epsilon$.

Definition 2 ($\alpha(i, j, k) = l, \beta(i, j, k) = l$). *Let* $\alpha(i, j, k) = l$ *and* $\beta(i, j, k) = l$*, where* l *represents the length of the MLCIS answer. These functions denote the MLCIS answer formed by sequences* $A_{0..i}$*,* $B_{0..j}$ *and* $T_{0..k}$*, where* a_i *and* b_j *are the ending elements in* $\alpha(\ )$ *and* $\beta(\ )$*, respectively.*

To calculate the MLCIS length, when t_k in T matches a_i in A ($t_k = a_i$), we refer to the maximum MLCIS length preceding a_i with an ending value less than t_k. Similarly, if t_k in T matches b_j in B ($t_k = b_j$), we refer to the maximum MLCIS length before b_j whose ending value is less than t_k.

By Definition 2, we get the following theorem.

Theorem 1. *In the* α *table, it holds that* $\alpha(i, j-1, k) \leq \alpha(i, j, k)$*, for* $0 \leq i \leq m$ *and* $1 \leq j \leq n$*. Similarly, in the* β *table, the inequality* $\beta(i-1, j, k) \leq \beta(i, j, k)$ *holds for* $1 \leq i \leq m$ *and* $0 \leq j \leq n$*.*

Proof. $\alpha(i, j-1, k)$ is the maximum MLCIS length built from $A_{0..i}$, $B_{0..j-1}$ and $T_{0..k}$, ending at a_i. The sequence merged by $A_{0..i}$ and $B_{0..j-1}$ is a subsequence of the one merged by $A_{0..i}$ and $B_{0..j}$. Therefore, the length of $\alpha(i, j-1, k)$ that can form an MLCIS ending at a_i is less than or equal to $\alpha(i, j, k)$.

The same result can be obtained for β.

Based on Theorem 1, when $t_k = a_i$, the calculation of $\alpha(i, j, k)$ only requires referencing the maximum from $\alpha(0, j, k-1)$ through $\alpha(i-1, j, k-1)$ and the

maximum from $\beta(i-1,0,k-1)$ to $\beta(i-1,j,k-1)$. When $t_k = b_j$, the calculation of $\beta(i,j,k)$ needs only to refer to the maximum from $\alpha(0,j-1,k-1)$ through $\alpha(i,j-1,k-1)$ and the maximum from $\beta(i,0,k-1)$ to $\beta(i,j-1,k-1)$. Figures 1 and 2 show the examples of the reference area for $\alpha(i,j,k)$ and $\beta(i,j,k)$, respectively.

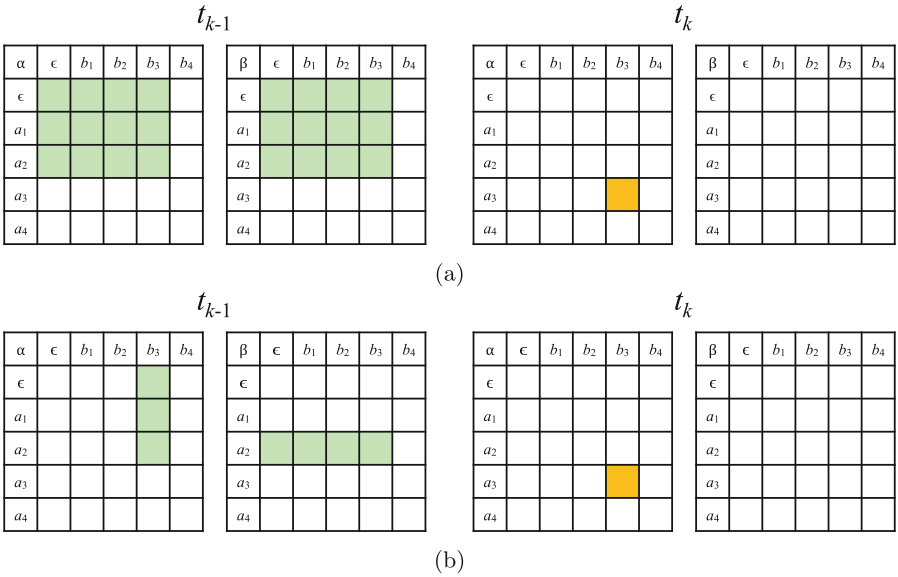

Fig. 1. The reference area for the calculation of $\alpha(3,3,k)$. Yellow block: $\alpha(3,3,k)$ for a match of $a_3 = t_k$; green blocks: the reference area for $\alpha(3,3,k)$. (a) The original reference area. (b) The shrunk reference area based on Theorem 1 (Color figure online).

When a match occurs in t_k ($a_i = t_k$ or $b_j = t_k$), to avoid duplicate calculation of α and β, and to gather the currently properly maximal values in α and β (the elements in A and B less than t_k), we introduce two additional tables, $\alpha^\star$ and $\beta^\star$, defined in Eqs. 2 to 3, respectively.

$$\alpha^\star(i,j,k-1) = \max\{\alpha(i',j,k-1) | a_{i'} < t_k, 0 \le i' \le i\} \tag{2}$$

$$\beta^\star(i,j,k-1) = \max\{\beta(i,j',k-1) | b_{j'} < t_k, 0 \le j' \le j\} \tag{3}$$

To reduce the required time for the calculation of $\alpha^\star$ and $\beta^\star$, we rewrite the recurrence formulas in Eqs. 4 to 7.

$$\alpha^\star(i,j,k-1) = \begin{cases} 0 & \text{if } i = 0; \\ \alpha^\star(i-1,j,k-1) & \text{if } a_i \ge t_k; \\ \max\{\alpha^\star(i-1,j,k-1), \alpha(i,j,k-1)\} & \text{if } a_i < t_k. \end{cases} \tag{4}$$

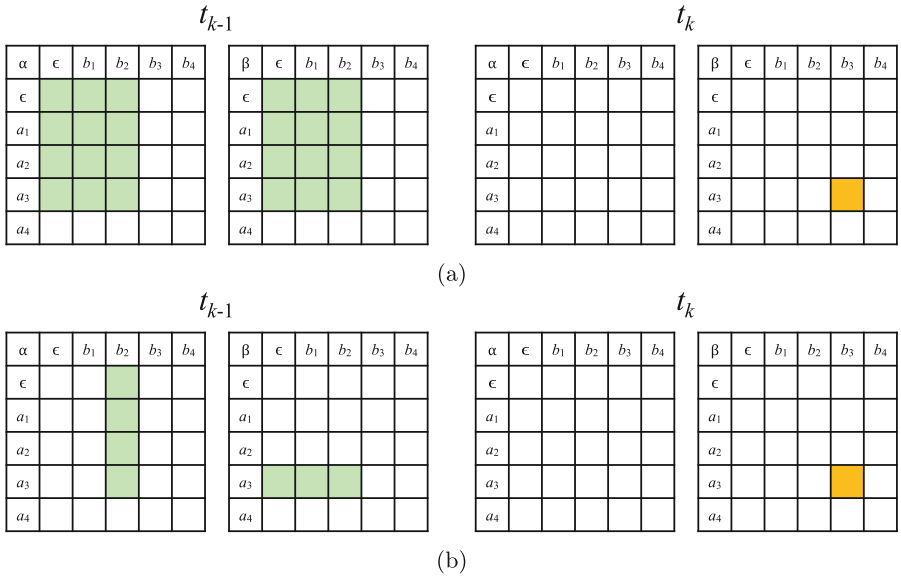

Fig. 2. The reference area for the calculation of $\beta(3,3,k)$. Yellow block: $\beta(3,3,k)$ for a match of $b_3 = t_k$; green blocks: the reference area for $\beta(3,3,k)$. (a) The original reference area. (b) The shrunk reference area based on Theorem 1 (Color figure online).

$$\beta^{\star}(i,j,k-1) = \begin{cases} 0 & \text{if } j = 0; \\ \beta^{\star}(i,j-1,k-1) & \text{if } b_j \geq t_k; \\ \max\{\beta^{\star}(i,j-1,k-1), \beta(i,j,k-1)\} & \text{if } b_j < t_k. \end{cases} \tag{5}$$

$$\alpha(i,j,k) = \begin{cases} 0 & \text{if } i = 0 \text{ or } k = 0; \\ \max\{\alpha^{\star}(i-1,j,k-1), \beta^{\star}(i-1,j,k-1)\} + 1 & \text{if } t_k = a_i; \\ \alpha(i,j,k-1) & \text{if } t_k \neq a_i. \end{cases} \tag{6}$$

$$\beta(i,j,k) = \begin{cases} 0 & \text{if } j = 0 \text{ or } k = 0; \\ \max\{\alpha^{\star}(i,j-1,k-1), \beta^{\star}(i,j-1,k-1)\} + 1 & \text{if } t_k = b_j; \\ \beta(i,j,k-1) & \text{if } t_k \neq b_j. \end{cases} \tag{7}$$

Finally, the MLCIS length can be obtained by Eq. 8.

$$|\text{MLCIS}| = \max\{\alpha(i,n,r), \beta(m,j,r) | 1 \leq i \leq m, 1 \leq j \leq n\}. \tag{8}$$

If the content of the MLCIS solution is desired to be output, a tracing back technique can be applied by recording the incremental path of the MLCIS solution. Based on these recurrence formulas, the algorithm for solving the MLCIS problem is formally presented in Algorithm 1.

Theorem 2. *Algorithm 1 solves the MLCIS problem in $O(mnr)$ time and $O(mn)$ space.*

Algorithm 1. The computation of the MLCIS length with DP (based on Eqs. 4 through 8)

Input: Two sequences $A = \langle a_1, a_2, \cdots, a_m \rangle$, $B = \langle b_1, b_2, \cdots, b_n \rangle$, and a target sequence $T = \langle t_1, t_2, \cdots, t_r \rangle$.
Output: The length of MLCIS(A, B, T)

```
1: α(i,j,k), β(i,j,k) ← 0, for 0 ≤ i ≤ m, 0 ≤ j ≤ n, k = 0,
2: α*(i,j,k), β*(i,j,k) ← 0, for 0 ≤ i ≤ m, 0 ≤ j ≤ n, k = 0,
3: for k = 1 to r do
4:     for i = 0 to m do
5:         for j = 0 to n do
6:             if i = 0 then
7:                 α*(i,j,k−1) ← 0
8:             else if a_i < t_k then
9:                 α*(i,j,k−1) ← max{α*(i−1,j,k−1), α(i,j,k−1)}
10:            else
11:                α*(i,j,k−1) ← α*(i−1,j,k−1)
12:            if j = 0 then
13:                β*(i,j,k−1) ← 0
14:            else if b_j < t_k then
15:                β*(i,j,k−1) ← max{β*(i,j−1,k−1), β(i,j,k−1)}
16:            else
17:                β*(i,j,k−1) ← β*(i,j−1,k−1)
18:            if a_i = t_k then
19:                α(i,j,k) ← max{α*(i−1,j,k−1), β*(i−1,j,k−1)} + 1
20:            else
21:                α(i,j,k) ← α(i,j,k−1)
22:            if b_j = t_k then
23:                β(i,j,k) ← max{α*(i,j−1,k−1), β*(i,j−1,k−1)} + 1
24:            else
25:                β(i,j,k) ← β(i,j,k−1)
26: return max{α(i,n,r), β(m,j,r) | 1 ≤ i ≤ m, 1 ≤ j ≤ n}
```

Proof. The outer loop in Line 3 is executed exactly r times, the middle loop in Line 4 is executed exactly $m+1$ times, and the inner loop in Line 5 is executed exactly $n+1$ times. The computation of each $\alpha(i,j,k)$, $\beta(i,j,k)$, $\alpha^\star(i,j,k)$ and $\beta^\star(i,j,k)$ requires constant time. So the time complexity is $O(mnr)$.

Even though $\alpha(i,j,k)$, $\beta(i,j,k)$, $\alpha^\star(i,j,k)$ and $\beta^\star(i,j,k)$ are 3-dimensional tables, they can be reduced to 2-dimensional tables. This is due to the reuse of each set of indices for different k independently. Thus, each of them needs $O(mn)$ space. In other words, the MLCIS problem can be solved in $O(mn)$ space.

4 A Complete Example

Figure 3 illustrates an example to explain the proposed DP algorithm, where $A = \langle 2, 5, 4, 8 \rangle$, $B = \langle 7, 4, 1, 8, 7 \rangle$, and $T = \langle 2, 7, 4, 5, 9, 7, 8 \rangle$. For the initialization

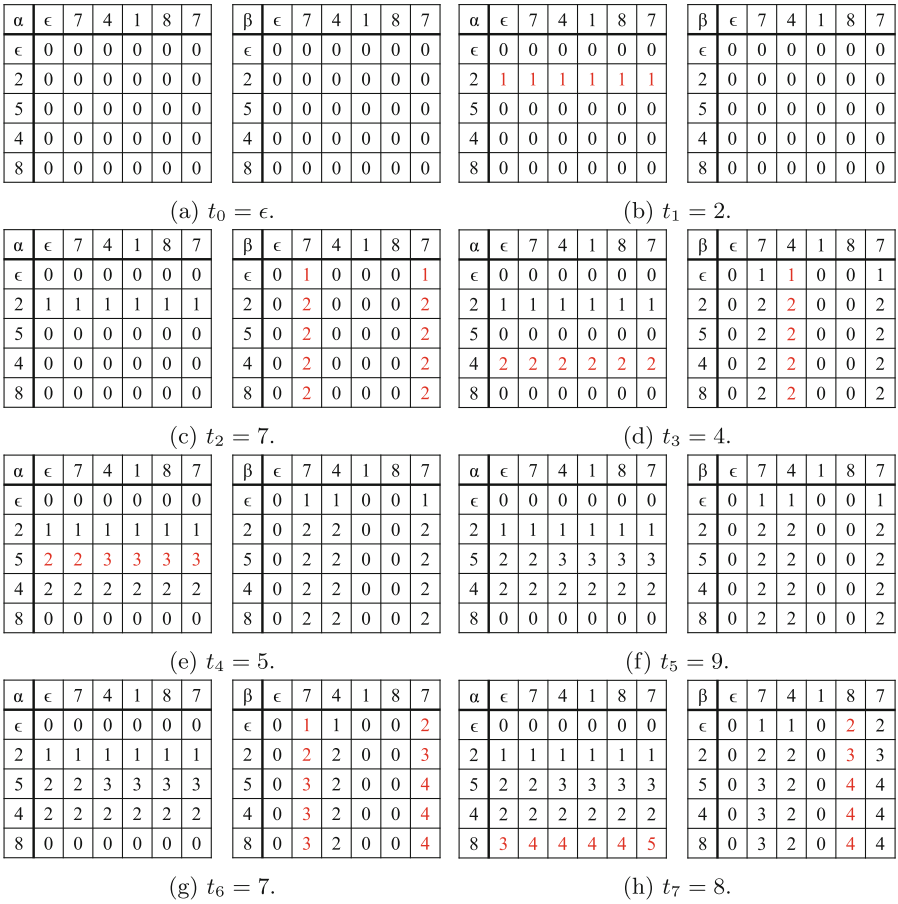

Fig. 3. An example of our DP algorithm for MLCIS, where $A = \langle 2, 5, 4, 8\rangle$, $B = \langle 7, 4, 1, 8, 7\rangle$ and $T = \langle 2, 7, 4, 5, 9, 7, 8\rangle$. Here, ϵ denotes a dummy element. The MLCIS answer is $\langle 2, 4, 5, 7, 8\rangle$, with length 5.

step with $t_0 = \epsilon$ and $k = 0$, where ϵ denotes a dummy element, each cell in the tables α and β is set to 0 according to Definitions 2.

Figure 3(a) shows the initialization of $\alpha(i, j, k)$ and $\beta(i, j, k)$ for $k = 0$ ($t_k = t_0 = \epsilon$). In Fig. 3(b), for considering $t_1 = 2$, since $a_1 = 2$, only $\alpha(1, *, 1)$ is updated by Eq. 6. For example, to update $\alpha(1, 3, 1)$, we utilize the values of $\alpha^\star(0, 3, 0)$ and $\beta^\star(0, 3, 0)$, which are the maximal lengths of MLCIS($\emptyset, B_{0..3}, \emptyset$) with ending elements smaller than $t_1 = 2$. So, $\alpha(1, 3, 1) = \max\{\alpha^\star(0, 3, 0), \beta^\star(0, 3, 0)\} + 1 = 1$. Subsequently, we update other cells in $\alpha(1, *, 1)$ using Eq. 6.

When considering $t_2 = 7$, since $b_1 = b_5 = 7$, we only update $\beta(*, 1, 2)$ and $\beta(*, 5, 2)$ by Eq. 7. For example, to update $\beta(3, 5, 2)$, we rely on the values of $\alpha^\star(3, 4, 1)$ and $\beta^\star(3, 4, 1)$, representing the maximal lengths of MLCIS($A_{0..3}, B_{0..4}, T_{0..1}$) with ending elements smaller than $t_2 = 7$. So

$\beta(3,5,2) = \max\{\alpha^\star(3,4,1), \beta^\star(3,4,1)\} + 1 = 2$. Consequently, we update other cells in $\beta(*,1,2)$ and $\beta(*,5,2)$ by Eq. 7.

For $t_3 = 4$, we have that $a_3 = b_2 = 4$. The update process focuses on $\alpha(3,*,3)$ by Eq. 6 and $\beta(*,2,3)$ by Eq. 7. As an illustration, the calculation of $\alpha(3,2,3)$ can be achieved by $\alpha(3,2,3) = \max\{\alpha^\star(2,2,2), \beta^\star(2,2,2)\} + 1 = 2$. As another example, $\beta(4,2,3) = \max\{\alpha^\star(4,1,2), \beta^\star(4,1,2)\} + 1 = 2$. These steps are repeated until $t_7 = 8$. In the end, the MLCIS length is determined from the maximum value in the tables α and β. Thus, the MLCIS length is 5.

5 Conclusion

In this paper, we begin by defining the MLCIS problem and subsequently propose a DP algorithm for solving it. The time complexity of the proposed algorithm is $O(mnr)$, where m, n and r denote the lengths of sequences A, B and T, respectively.

The diagonal concept is a robust approach for tackling the LCS problem, particularly effective when dealing with highly similar sequences. Currently, we are working on the development of a diagonal method for the MLCIS problem.

Moreover, our proposed algorithm can be adapted to address various LCS-related problems, such as longest common almost increasing subsequence. In addition, we may explore other related problems. For example, given a sequence T, the *split longest common subsequence* (SLCS) problem may be defined to maximize $LCS(A,B)$ by strategically splitting T into two sequences, A and B. Similarly, given a sequence T, the *split longest common increasing subsequence* (SLCIS) problem optimizes the split of T into two sequences, A and B, in order such that $LCIS(A,B)$ is maximized. These questions may broaden the scope of our research.

References

1. Brodal, G.S., Kaligosi, K., Katriel, I., Kutz, M.: Faster algorithms for computing longest common increasing subsequences. In: Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM), Barcelona, Spain, pp. 330–341 (2006)
2. Cai, D., Zhu, D., Wang, L., Wang, X.: A simple linear space algorithm for computing a longest common increasing subsequence. IAENG Int. J. Comput. Sci. **45**(3), 472–477 (2018)
3. Chan, W.T., Zhang, Y., Fung, S.P.Y., Ye, D., Zhu, H.: Efficient algorithms for finding a longest common increasing subsequence. J. Comb. Optim. **13**(3), 277–288 (2007)
4. Deorowicz, S., Danek, A.: Bit-parallel algorithms for the merged longest common subsequence problem. Int. J. Found. Comput. Sci. **24**, 1281–1298 (2013)
5. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM **18**(6), 341–343 (1975)
6. Huang, K.S., Yang, C.B., Tseng, K.T., Ann, H.Y., Peng, Y.H.: Efficient algorithms for finding interleaving relationship between sequences. Inf. Process. Lett. **105**, 188–193 (2008)

7. Hunt, J.W., Szymanski, T.G.: A fast algorithm for computing longest common subsequences. Commun. ACM **20**, 350–353 (1977)
8. Lo, S.F., Tseng, K.T., Yang, C.B., Huang, K.S.: A diagonal-based algorithm for the longest common increasing subsequence problem. Theoret. Comput. Sci. **815**, 69–78 (2020)
9. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM (JACM) **25**(2), 322–336 (1978)
10. Nakatsu, N., Kambayashi, Y., Yajima, S.: A longest common subsequence algorithm suitable for similar text strings. Acta Informatica **18**, 171–179 (1982)
11. Peng, Y.H., Yang, C.B., Huang, K.S., Tseng, C.T., Hor, C.Y.: Efficient sparse dynamic programming for the merged LCS problem with block constraints **6**, 1935–1947 (2010)
12. Rahman, A.M., Rahman, M.S.: Effective sparse dynamic programming algorithms for merged and block merged LCS problems. J. Comput. **9**(8), 1743–1754 (2014)
13. Sakai, Y.: A linear space algorithm for computing a longest common increasing subsequence. Inf. Process. Lett. **99**(5), 203–207 (2006)
14. Tseng, K.T., Chan, D.S., Yang, C.B., Lo, S.F.: Efficient merged longest common subsequence algorithms for similar sequences. Theoret. Comput. Sci. **708**, 75–90 (2018)
15. Wagner, R., Fischer, M.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)
16. Yang, I.H., Huang, C.P., Chao, K.M.: A fast algorithm for computing a longest common increasing subsequence. Inf. Process. Lett. **93**(5), 249–253 (2005)