Theoretical Computer Science 854 (2021) 44–51

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computing a longest common almost-increasing subsequence of two sequences

Toan Thang Ta, Yi-Kung Shieh, Chin Lung Lu *

Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan

ARTICLE INFO

Article history:
Received 20 September 2019
Received in revised form 5 May 2020
Accepted 8 November 2020
Available online 20 November 2020

Keywords:
Algorithm
Dynamic programming
Longest common almost-increasing subsequence

ABSTRACT

Given a positive constant c, a sequence $S = \langle s_1, s_2, \ldots, s_k \rangle$ of k numbers is said to be almost increasing if and only if $s_i > \max_{1 \le j < i} s_j - c$ for all $1 < i \le k$. A longest common almost-increasing subsequence (LCaIS) between two input sequences is a longest common subsequence that is also an almost increasing sequence. We found out that the existing algorithm proposed by Moosa et al. [1] to find an LCaIS of two sequences without repeated elements gives an incorrect result for some instances. In this work, we present a dynamic programming algorithm that can correctly compute an LCaIS between any two sequences with repeated elements in $O(nml)$ time and $O(nm)$ space, where n and m are the lengths of two input sequences and l is the length of the output LCaIS.

1. Introduction

Longest common subsequence (LCS) is a commonly used metric to measure the similarity of two sequences [2–5]. Given a sequence $S = \langle s_1, s_2, \ldots, s_k \rangle$ of k numbers, S is an *increasing* sequence if $s_i > s_{i-1}$ for all $1 < i \le k$. By applying the increasing condition to LCS, Yang et al. [6] designed a dynamic programming algorithm to find a longest common increasing subsequence (LCIS) between two comparable sequences in $O(nm)$ time and space, where n and m are the lengths of two input sequences. In fact, the LCIS problem can be applied to compute the alignment of two large-scale genomes [7,8] and is an extension of the well-known longest increasing subsequence (LIS) problem that can be solved in optimal time $O(n \log n)$ [9–11], where n is the length of the original sequence.

Recently, Elmasry [12] introduced the concept of an almost-increasing subsequence. That is, given a constant $c > 0$, a sequence $S = \langle s_1, s_2, \ldots, s_k \rangle$ is said to be *almost increasing* if and only if $s_i > \max_{1 \le j < i} s_j - c$ for all $1 < i \le k$. Elmasry used this concept to introduce the problem of finding a longest almost-increasing subsequence (LaIS) in a given sequence of length n, which can be considered as a relaxed version of LIS problem when the elements of the input sequence have a small amount of noise. In addition, Elmasry [12] designed an optimal algorithm that solves the LaIS problem in $O(n \log l)$ time, where l is the length of the output subsequence. Later, Moosa et al. [1] studied the problem of finding a longest common almost-increasing subsequence (LCaIS) between two sequences. Formally, given two sequences of numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_m \rangle$ and a constant $c > 0$, a *common almost-increasing subsequence* of A and B is a common sequence $C = \langle a_{i_1} = b_{j_1}, a_{i_2} = b_{j_2}, \ldots, a_{i_k} = b_{j_k} \rangle$, where $1 \le i_1 < i_2 < \ldots < i_k \le n$ and $1 \le j_1 < j_2 < \ldots < j_k \le m$, such that C itself is an almost increasing sequence. A *longest common almost-increasing subsequence* of A and B is a common almost-increasing

* Corresponding author.
E-mail addresses: toanthanghy@gmail.com (T.T. Ta), d9762814@oz.nthu.edu.tw (Y.-K. Shieh), cllu@cs.nthu.edu.tw (C.L. Lu).

https://doi.org/10.1016/j.tcs.2020.11.035
0304-3975/

subsequence having the maximum length. Note that the LCaIS problem considered by Moosa et al. [1] requires no repeated elements in each input sequence, i.e., $a_i \neq a_u$ and $b_j \neq b_v$ for all $i \neq u$ and $j \neq v$. In [1], Moosa et al. finally gave a dynamic programming algorithm that uses $O(n^2)$ space and runs in $O(n(n+c^2))$ time to find an LCaIS between two sequences of equal length n.

We have found, however, that the algorithm proposed by Moosa et al. [1] gives an incorrect result for the following simple instance. Consider two sequences $A = \langle 11, 7, 9, 8, 6\rangle$ and $B = \langle 9, 8, 11, 7, 6\rangle$ and a constant $c = 5$. Their algorithm outputs $\langle 7, 6\rangle$ of length 2 as an LCaIS, but it can be easily verified that the sequence $\langle 9, 8, 6\rangle$ of length 3 is a common almost-increasing subsequence of A and B and clearly $\langle 7, 6\rangle$ is not an LCaIS between A and B. The flaw in this algorithm is that when constructing common almost-increasing subsequences of $\langle 11, 7, 9, 8\rangle$ and $\langle 9, 8, 11, 7\rangle$, there are four maximal common almost-increasing sequences, namely $\langle 9\rangle$ and $\langle 11\rangle$ of length 1 as well as $\langle 9, 8\rangle$ and $\langle 11, 7\rangle$ of length 2. The algorithm only keeps the sequence of the smallest ending value among sequences having the same length (i.e., $\langle 9\rangle$ and $\langle 11, 7\rangle$ in this case) for later computation. Since the sequence $\langle 9, 8\rangle$ is removed, the algorithm cannot produce a common almost-increasing subsequence $\langle 9, 8, 6\rangle$ of length 3. The reason why removing the sequence of larger ending value when there exist more than one sequence of equal length was not strictly proved in [1]. This motivates us to devise a correct algorithm to solve the LCaIS problem in this work.

In this study, we present a dynamic programming algorithm for computing an LCaIS of two input sequences of lengths n and m and its time and space complexities are $O(nml)$ and $O(nm)$, respectively, where l is the length of the output LCaIS. Note that computing just the length of the LCaIS can be done using less space (i.e., $O(l\min(m, n))$) by our algorithm. It is worth mentioning that the LCaIS problem we consider in this work allows repeated elements in the input sequences, while the LCaIS problem studied by Moosa et al. [1] does not.

The rest of the paper is organized as follows. In Section 2, we provide some notations to facilitate the presentation of our algorithm. Next, we discuss the main algorithm and analyze its time and space complexities in Section 3. Finally, we give a brief conclusion in Section 4.

2. Preliminaries

For a sequence of n numbers $S = \langle s_1, s_2, \ldots, s_n\rangle$, we use $|S| = n$ to denote the *length* of S and $S_{i,j} = \langle s_i, s_{i+1}, \ldots, s_j\rangle$ to denote a substring of S from the i-th to the j-th element, where $1 \leq i \leq j \leq n$. We define the *representative* of S, written as $\gamma(S)$, to be the last maximum element in S. For example, if $S = \langle 5, 3, \mathbf{5}, 1\rangle$, then $\gamma(S)$ is s_3 even though both s_1 and s_3 are maxima in S. The representative of a sequence is written in bold style in all examples given in this work.

Let $A = \langle a_1, a_2, \ldots, a_n\rangle$ and $B = \langle b_1, b_2, \ldots, b_m\rangle$ be two sequences. We call a pair (i, j) as a *matched pair* if and only if $a_i = b_j$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. Otherwise, it is called a *mismatched pair*. For a matched pair (i, j), we utilize $\alpha(i, j)$ to indicate either a_i or b_j, i.e., $\alpha(i, j) = a_i = b_j$. With these two sequences A and B, we also create an $n \times m$ two-dimensional table T, called *matching table*, in which $T(i, j) =$ '$*$' if $a_i = b_j$; otherwise $T(i, j) = \varepsilon$, where ε denotes the empty character. Note that a star entry $T(i, j)$ corresponds to a matched pair (i, j) and we say that $\alpha(i, j)$ is the *value* of $T(i, j)$. Basically, a common almost increasing subsequence (CaIS) of length k between two sequences A and B corresponds to an *almost increasing path* $P = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_k, j_k)$ consisting of star entries in T with $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and $1 \leq j_1 < j_2 < \ldots < j_k \leq m$. Conversely, an almost increasing path $P = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_k, j_k)$ in T *yields* a unique CaIS of A and B, denoted as $\beta(P)$, by concatenating the values of star entries in P, i.e., $\beta(P) = \langle \alpha(i_1, j_1), \alpha(i_2, j_2), \ldots, \alpha(i_k, j_k)\rangle$. Therefore, our problem is equivalent to finding a longest almost increasing path in the matching table T of two input sequences A and B. Given an almost increasing path $P = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_k, j_k)$ in T, we call the star entry $T(i_h, j_h)$ in P as the *representative entry* (or *point*) of P if $\alpha(i_h, j_h)$ is the representative of $\beta(P)$ for some $1 \leq h \leq k$. We use $\omega(P)$ to denote the value of the representative point in P and $|P|$ to indicate its length, i.e., $\omega(P) = \alpha(i_h, j_h)$ and $|P| = k$. Actually, $\omega(P)$ is the maximum value of the sequence $\beta(P)$. For convenience, we use $T_{i,j}$ to denote a sub-table of T whose top-left entry is $T(1, 1)$ and bottom-right entry is $T(i, j)$. Note that $T_{i,j}$ is an empty table if $i = 0$ or $j = 0$. For each star entry $T(u, v)$ in $T_{i,j}$, we utilize $LP^i_j(u, v)$ to indicate a longest almost increasing path in $T_{i,j}$ such that $T(u, v)$ is its representative point. For example, consider $A = \langle 3, 2, 1, 3, 2\rangle$ and $B = \langle 1, 3, 3, 2\rangle$. When $c = 2$, we have $LP^5_4(4, 3) = T(1, 2), T(4, 3), T(5, 4)$ and $\beta(LP^5_4(4, 3)) = \langle \alpha(1, 2), \alpha(4, 3), \alpha(5, 4)\rangle = \langle 3, \mathbf{3}, 2\rangle$ as illustrated in Fig. 1. Actually, it can be verified that $P = T(3, 1), T(4, 3), T(5, 4)$ is another longest almost increasing path in $T_{5,4}$ such that $T(4, 3)$ is the representative point of P as well.

3. Our dynamic programming algorithm for solving the LCaIS problem

We use $L(i, j)$ to denote the set of $LP^i_j(u, v)$ for all star entries $T(u, v)$ in $T_{i,j}$, i.e., $L(i, j) = \{LP^i_j(u, v) : T(u, v) =$'$*$', $1 \leq u \leq i$ and $1 \leq v \leq j\}$. In fact, if we can compute $L(i, j)$ for all $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$, then the longest path in $L(n, m)$ will yield an LCaIS of A and B. By definition, $L(i, j) = \varnothing$ if $i = 0$ or $j = 0$. Now, suppose that $L(i-1, j-1)$, $L(i-1, j)$ and $L(i, j-1)$ are already known. Then we can construct $L(i, j)$ from these sets according to the approaches described in the following two cases.

Case 1: $T(i, j)$ is an empty entry. For each star entry $T(u, v)$ in $T_{i,j}$, we add the longest path among all paths in $L(i-1, j) \cup L(i, j-1)$ such that their representative points are $T(u, v)$ into $L(i, j)$.

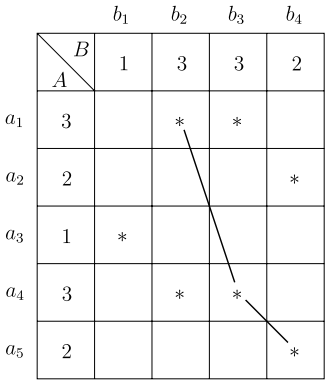

Fig. 1. Matching table T of $A = \langle 3, 2, 1, 3, 2\rangle$ and $B = \langle 1, 3, 3, 2\rangle$. When $c = 2$, we have $LP_4^5(4, 3) = T(1, 2), T(4, 3), T(5, 4)$ and its yielded CaIS $\langle \alpha(1, 2), \alpha(4, 3), \alpha(5, 4)\rangle = \langle 3, \mathbf{3}, 2\rangle$.

$i \backslash j$	3	4
3		$\beta(LP_4^3(1, 2)) = \langle \mathbf{3}, 2\rangle$ $\beta(LP_4^3(1, 3)) = \langle \mathbf{3}, 2\rangle$ $\beta(LP_4^3(2, 4)) = \langle \mathbf{2}\rangle$ $\beta(LP_4^3(3, 1)) = \langle \mathbf{1}\rangle$
4	$\beta(LP_3^4(1, 2)) = \langle \mathbf{3}\rangle$ $\beta(LP_3^4(1, 3)) = \langle \mathbf{3}\rangle$ $\beta(LP_3^4(3, 1)) = \langle \mathbf{1}\rangle$ $\beta(LP_3^4(4, 2)) = \langle 1, \mathbf{3}\rangle$ $\beta(LP_3^4(4, 3)) = \langle 3, \mathbf{3}\rangle$	$\beta(LP_4^4(1, 2)) = \langle \mathbf{3}, 2\rangle$ $\beta(LP_4^4(1, 3)) = \langle \mathbf{3}, 2\rangle$ $\beta(LP_4^4(2, 4)) = \langle \mathbf{2}\rangle$ $\beta(LP_4^4(3, 1)) = \langle \mathbf{1}\rangle$ $\beta(LP_4^4(4, 2)) = \langle 1, \mathbf{3}\rangle$ $\beta(LP_4^4(4, 3)) = \langle 3, \mathbf{3}\rangle$

Fig. 2. $L(4, 4)$ is obtained from $L(4, 3)$ and $L(3, 4)$, where $(4, 4)$ is a mismatched pair.

Case 2: $T(i, j)$ is a star entry. First, we construct $LP_j^i(i, j)$ by searching for the longest path P in $L(i-1, j-1)$ such that $\omega(P) \le \alpha(i, j)$ and then appending $T(i, j)$ to P. Second, we append $T(i, j)$ to every path P in $L(i-1, j-1)$ satisfying $\alpha(i, j) < \omega(P) < \alpha(i, j) + c$. Now, let $L'(i, j)$ denote the set of all paths obtained in the above two steps. Finally, for each star entry $T(u, v)$ in $T_{i,j}$, we add the longest path among all paths in $L(i-1, j) \cup L(i, j-1) \cup L'(i, j)$ such that their representative points are $T(u, v)$ into $L(i, j)$.

For example, consider two sequences A, B and constant c as given in Fig. 1. Then the computations of $L(4, 4)$ (respectively, $L(5, 4)$) is briefly illustrated in Fig. 2 (respectively, Fig. 3) using the method described in the aforementioned Case 1 (respectively, Case 2), where for each almost increasing path P, we only write down its yielded CaIS $\beta(P)$ for simplicity.

In the following, we discuss how to efficiently implement the basic idea mentioned above to solve the LCaIS problem.

Definition 1. Let P and Q be two almost increasing paths in a sub-table $T_{i,j}$. We say that P *dominates* Q if and only if $\omega(P) \le \omega(Q)$ and $|P| > |Q|$.

Below, we simplify $L(i, j)$ by removing those paths that are useless to find the optimal solution of the LCaIS problem. First, we remove all dominated paths in $L(i, j)$. Next, if the resulting $L(i, j)$ contains $k > 1$ paths $LP_j^i(u_1, v_1)$, $LP_j^i(u_2, v_2)$, ..., $LP_j^i(u_k, v_k)$ of the same length with $\alpha(u_1, v_1) \le \alpha(u_2, v_2) \le \ldots \le \alpha(u_k, v_k)$, then we retain $LP_j^i(u_1, v_1)$ and remove $k-1$ remaining paths from $L(i, j)$. For convenience, we call these $k-1$ removed paths as *redundant* paths in $L(i, j)$. Finally, we replace each $LP_j^i(u, v)$ in $L(i, j)$ by a 3-tuple $(u, v, |LP_j^i(u, v)|)$ and call the final $L(i, j)$ as a *simplified* $L(i, j)$. Fig. 4 shows an example of simplifying $L(4, 3)$ as shown in Fig. 3 according to the above discussion. It is not hard to see that a 3-tuple with the maximum length in the simplified $L(n, m)$ corresponds to a longest almost increasing path in the matching table T that yields an LCaIS of the input sequences A and B. It can also be observed that every two 3-tuples (u, v, k) and (u', v', k') in a simplified $L(i, j)$ satisfy the conditions $k \ne k'$ and $\alpha(u, v) < \alpha(u', v')$ if and only if $k < k'$. This means that the cardinal number of a simplified $L(i, j)$ is at most l, where l denotes the length of the output for the LCaIS problem.

In Procedure 1, we describe an algorithm to compute simplified $L'(i, j)$ directly from simplified $L(i-1, j-1)$.

For simplicity, in the rest of this section, we assume that $L(i, j)$, as well as $L'(i, j)$, is already simplified for any $1 \le i \le n$ and $1 \le j \le m$ when we mention it.

$i \backslash j$	3	4
4	$\beta(LP_3^4(1,2)) = \langle \mathbf{3} \rangle$ $\beta(LP_3^4(1,3)) = \langle \mathbf{3} \rangle$ $\beta(LP_3^4(3,1)) = \langle \mathbf{1} \rangle$ $\beta(LP_3^4(4,2)) = \langle 1, \mathbf{3} \rangle$ $\beta(LP_3^4(4,3)) = \langle 3, \mathbf{3} \rangle$	$\beta(LP_4^4(1,2)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^4(1,3)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^4(2,4)) = \langle \mathbf{2} \rangle$ $\beta(LP_4^4(3,1)) = \langle \mathbf{1} \rangle$ $\beta(LP_4^4(4,2)) = \langle 1, \mathbf{3} \rangle$ $\beta(LP_4^4(4,3)) = \langle 3, \mathbf{3} \rangle$
5	$\beta(LP_3^5(1,2)) = \langle \mathbf{3} \rangle$ $\beta(LP_3^5(1,3)) = \langle \mathbf{3} \rangle$ $\beta(LP_3^5(3,1)) = \langle \mathbf{1} \rangle$ $\beta(LP_3^5(4,2)) = \langle 1, \mathbf{3} \rangle$ $\beta(LP_3^5(4,3)) = \langle 3, \mathbf{3} \rangle$	**Step 1:** $\beta(LP_4^5(5,4)) = \langle 1, \mathbf{2} \rangle$ **Step 2:** $\beta(LP_4^5(1,2)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(1,3)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(4,2)) = \langle 1, \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(4,3)) = \langle 3, \mathbf{3}, 2 \rangle$ **Step 3:** $\beta(LP_4^5(1,2)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(1,3)) = \langle \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(2,4)) = \langle \mathbf{2} \rangle$ $\beta(LP_4^5(3,1)) = \langle \mathbf{1} \rangle$ $\beta(LP_4^5(4,2)) = \langle 1, \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(4,3)) = \langle 3, \mathbf{3}, 2 \rangle$ $\beta(LP_4^5(5,4)) = \langle 1, \mathbf{2} \rangle$

Fig. 3. Three steps for constructing $L(5,4)$, where $(5,4)$ is a matched pair.

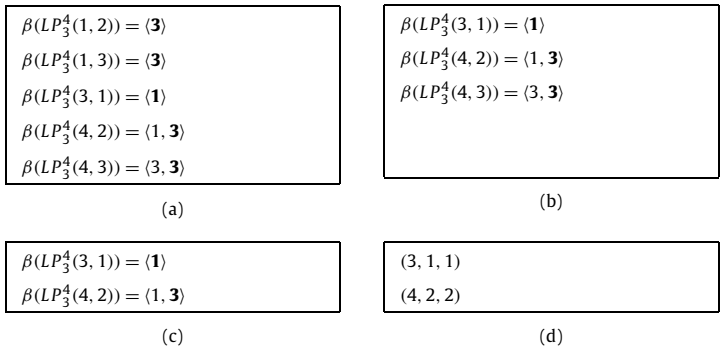

Fig. 4. (a) Original $L(4,3)$, (b) temporary $L(4,3)$ after removing dominated paths $LP_3^4(1,2)$ and $LP_3^4(1,3)$, (c) temporary $L(4,3)$ after removing a redundant path $LP_3^4(4,3)$ and (d) simplified $L(4,3)$.

Definition 2. Given any two sets of 3-tuples $L(i_1, j_1)$ and $L(i_2, j_2)$, where $1 \le i_1, i_2 \le n$ and $1 \le j_1, j_2 \le m$, the *merging* operation $\oplus$ is defined by $L(i_1, j_1) \oplus L(i_2, j_2)$ that equals to the set obtained by first uniting $L(i_1, j_1)$ and $L(i_2, j_2)$ and then removing all the dominated and redundant 3-tuples.

In the following lemma, we derive a simpler recursive formula for more efficiently computing $L(i, j)$. The intuition behind this lemma is that if $T(i, j)$ is a star entry, then every path in $L(i-1, j) \cup L(i, j-1)$ is either dominated by some path in $L(i-1, j-1) \cup L'(i, j)$ or redundant when compared with some path in $L(i-1, j-1) \cup L'(i, j)$.

Lemma 1. *If (i, j) is a matched pair, then $L(i, j) = L(i-1, j-1) \oplus L'(i, j)$; otherwise $L(i, j) = L(i, j-1) \oplus L(i-1, j)$. In other words, we have*

$$L(i, j) = \begin{cases} L(i-1, j-1) \oplus L'(i, j), & \text{if } T(i, j) = \text{'}*\text{'} \\ L(i, j-1) \oplus L(i-1, j), & \text{otherwise.} \end{cases}$$

Procedure 1 Append1($L(i-1, j-1), c$).

Input: Simplified $L(i-1, j-1)$ and constant $c > 0$.
Output: Simplified $L'(i, j)$.
1: $L'(i, j) = \varnothing$; $l_{max} = 0$;
2: **for each** (u, v, k) in $L(i-1, j-1)$ **do**
3: **if** $\alpha(u, v) \leq \alpha(i, j)$ and $l_{max} < k$ **then**
4: $l_{max} = k$;
5: **end if**
6: **if** $\alpha(i, j) < \alpha(u, v) < \alpha(i, j) + c$ **then**
7: $L'(i, j) = L'(i, j) \cup \{(u, v, k+1)\}$;
8: **end if**
9: **end for**
10: $L'(i, j) = L'(i, j) \cup \{(i, j, l_{max} + 1)\}$;
11: **return** $L'(i, j)$;

Proof. To prove the correctness of Lemma 1, it is sufficient to show that for any non-dominated path P in the sub-table $T_{i,j}$, there exists a 3-tuple (u, v, k) in $L(i, j)$ such that $\alpha(u, v) \leq \omega(P)$ and $k = |P|$, where $L(i, j)$ is computed by the formula stated in Lemma 1 and $L'(i, j)$ is obtained by Procedure 1. Below we prove it by induction on i and j.

Basis step ($i = 0$ or $j = 0$): The claim clearly holds since $L(i, j) = \varnothing$ when $i = 0$ or $j = 0$ and there is no path in the empty table $T_{i,j}$.

Induction step ($i > 0$ and $j > 0$): It can be verified that for each 3-tuple in $L'(i, j)$ obtained by Procedure 1, there exists a corresponding almost increasing path in $T_{i,j}$. Let $P = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_k, j_k)$ be a non-dominated path in $T_{i,j}$. It should be noted that a non-dominated path in $T_{i,j}$ can be a redundant path. If (i, j) is a matched pair, then we prove the claim according to two possible locations of the matched pair (i_k, j_k).

Case 1: $(i_k, j_k) = (i, j)$. Let $P' = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_{k-1}, j_{k-1})$. We show below that P' is actually a non-dominated path in $T_{i-1,j-1}$. Suppose that P' is dominated by another almost increasing path Q' in $T_{i-1,j-1}$, i.e., $\omega(Q') \leq \omega(P')$ and $|Q'| > |P'|$. Let R be the path obtained by appending star entry $T(i, j)$ into Q'. Then R is an almost increasing path in $T_{i,j}$ with $\omega(R) = \max\{\omega(Q'), \alpha(i, j)\} \leq \max\{\omega(P'), \alpha(i, j)\} = \omega(P)$ but $|R| = |Q'| + 1 > |P'| + 1 = k = |P|$. As a result, R dominates P in $T_{i,j}$, a contradiction. Hence, we can conclude that P' is a non-dominated path in $T_{i-1,j-1}$. By induction hypothesis, there exists a 3-tuple $(u, v, k-1)$ in $L(i-1, j-1)$ such that $\alpha(u, v) \leq \omega(P')$ and $k - 1 = |P'|$. This also indicates that $\alpha(i, j) > \omega(P') - c \geq \alpha(u, v) - c$.

Case 1.1: $\alpha(i, j) < \alpha(u, v)$. Then $(u, v, k) \in L'(i, j)$ by line 7 in Procedure 1 and the representative point of P' is identical to that of P, i.e., $\omega(P) = \omega(P') = \alpha(u, v)$. Since P is a non-dominated path in $T_{i,j}$, the 3-tuple (u, v, k) cannot be dominated by other tuples in $L'(i, j)$ or $L(i-1, j-1)$. Therefore, there is a tuple (x, y, k) in $L(i, j)$, which is obtained by performing $L(i-1, j-1) \oplus L'(i, j)$, such that $\alpha(x, y) \leq \alpha(u, v) = \omega(P)$ and $k = |P|$.

Case 1.2: $\alpha(i, j) \geq \alpha(u, v)$. Then (i, j, k) is a candidate tuple in $L'(i, j)$ according to lines 4 and 10 in Procedure 1. In this case, the representative point of P is clearly $T(i, j)$. Hence, the 3-tuple (i, j, k) actually corresponds to P in $T_{i,j}$. Since P is a non-dominated path in $T_{i,j}$, the 3-tuple (i, j, k) cannot be dominated by other tuples in $L'(i, j)$ or $L(i-1, j-1)$. Hence, there exists a tuple (x, y, k) in $L(i, j)$, which is obtained by performing $L(i-1, j-1) \oplus L'(i, j)$, such that $\alpha(x, y) \leq \alpha(i, j) = \omega(P)$ and $k = |P|$.

Case 2: $(i_k, j_k) \neq (i, j)$. Assume that P is a path in $T_{i-1,j}$. There are two possibilities for j_k.

Case 2.1: $j_k = j$. Let $Q = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_{k-1}, j_{k-1}), T(i, j)$. Since $\alpha(i_k, j_k) = b_j = \alpha(i, j)$, Q is an almost increasing path in $T_{i,j}$ with $|Q| = k = |P|$ and $\omega(Q) = \omega(P)$. As P is a non-dominated path in $T_{i,j}$, Q also is. Let $P' = T(i_1, j_1), T(i_2, j_2), \ldots, T(i_{k-1}, j_{k-1})$. Because Q is a non-dominated path in $T_{i,j}$, P' is a non-dominated path in $T_{i-1,j-1}$ using similar argument to that in Case 1. By induction hypothesis, there exists a 3-tuple $(u, v, k-1)$ in $L(i-1, j-1)$ such that $\alpha(u, v) \leq \omega(P')$ and $k - 1 = |P'|$. By the arguments in Cases 1.1 and 1.2, we can conclude that there is a tuple (x, y, k) in $L(i, j)$ such that $\alpha(x, y) \leq \omega(P)$ and $k = |P|$.

Case 2.2: $j_k < j$. Clearly, P is a non-dominated path in $T_{i-1,j-1}$. By induction hypothesis, there exists a 3-tuple (u, v, k) in $L(i-1, j-1)$ such that $\alpha(u, v) \leq \omega(P)$ and $k = |P|$. Since P is also a non-dominated path in $T_{i,j}$, the 3-tuple (u, v, k) cannot be dominated by other tuples in $L'(i, j)$ or $L(i-1, j-1)$. Thus, there is a tuple (x, y, k) in $L(i, j)$, which is obtained by performing $L(i-1, j-1) \oplus L'(i, j)$, such that $\alpha(x, y) \leq \alpha(u, v) \leq \omega(P)$ and $k = |P|$.

On the other hand, suppose that P is a path in $T_{i,j-1}$. Then using the arguments similar to Cases 2.1 and 2.2, we can prove that there exists a tuple (x, y, k) in $L(i, j)$ such that $\alpha(x, y) \leq \omega(P)$ and $k = |P|$.

If (i, j) is a mismatched pair, then P is a path either in $T_{i-1,j}$ or $T_{i,j-1}$. Assume that P is a path in $T_{i-1,j}$ (respectively, $T_{i,j-1}$). Then P is a non-dominated path in $T_{i-1,j}$ (respectively, $T_{i,j-1}$). By induction hypothesis, there exists a 3-tuple (u, v, k) in $L(i-1, j)$ (respectively, $L(i, j-1)$) such that $\alpha(u, v) \leq \omega(P)$ and $k = |P|$. Since P is a non-dominated path in $T_{i,j}$, the 3-tuple (u, v, k) cannot be dominated by other tuples in $L(i-1, j)$ or $L(i, j-1)$. Therefore, there exists a tuple (x, y, k) in $L(i, j)$, which is obtained by performing $L(i-1, j) \oplus L(i, j-1)$, such that $\alpha(x, y) \leq \alpha(u, v) \leq \omega(P)$ and $k = |P|$. □

Using the recursive formula in Lemma 1, we can compute $L(n,m)$, in which a 3-tuple (u', v', l_{max}) with $l_{max} = \max\{k : (u, v, k) \in L(n,m)\}$ represents a longest almost increasing path in the matching table T. To help recover this longest path, we build an $n \times m$ table *Prev* as follows. When computing $LP^i_j(i,j)$, we use entry $Prev(i,j)$ to store the location (x,y) in T such that $LP^{i-1}_{j-1}(x,y)$ is a longest path in $L(i-1, j-1)$ satisfying $\alpha(x,y) \le \alpha(i,j)$. That is, $LP^{i-1}_{j-1}(x,y)$ is the path selected to construct $LP^i_j(i,j)$ by appending it with $T(i,j)$. Based on Lemma 1, we describe Algorithm 1 below that can more efficiently compute an LCaIS between two input sequences.

Algorithm 1 Finding an LCaIS of two sequences.

Input: Two sequences $A = \langle a_1, a_2, \ldots, a_n\rangle$ and $B = \langle b_1, b_2, \ldots, b_m\rangle$ and constant $c > 0$.
Output: An LCaIS between A and B.

```
1: for i = 0 to n do
2:     L(i, 0) = ∅;
3: end for
4: for j = 0 to m do
5:     L(0, j) = ∅;
6: end for
7: for i = 1 to n do
8:     for j = 1 to m do
9:         if a_i = b_j then
10:            L(i, j) = Append2(L(i − 1, j − 1), c) ⊕ L(i − 1, j − 1);
11:        else
12:            L(i, j) = L(i − 1, j) ⊕ L(i, j − 1);
13:        end if
14:    end for
15: end for
16: Backtrack(L(n, m), Prev);
```

Procedure 2 Append2$(L(i-1, j-1), c)$.

Input: Set $L(i-1, j-1)$ and constant $c > 0$.
Output: $L'(i,j)$.

```
1: L'(i, j) = ∅; l_max = 0; (x, y) = (0, 0);
2: for each (u, v, k) in L(i − 1, j − 1) do
3:     if α(u, v) ≤ α(i, j) and l_max < k then
4:         l_max = k; (x, y) = (u, v);
5:     end if
6:     if α(i, j) < α(u, v) < α(i, j) + c then
7:         L'(i, j) = L'(i, j) ∪ {(u, v, k + 1)};
8:     end if
9: end for
10: L'(i, j) = L'(i, j) ∪ {(i, j, l_max + 1)};
11: Prev(i, j) = (x, y);
12: return L'(i, j);
```

Note that the information in the *Prev* table can only be used to construct a partial result of the LCaIS of the two input sequences. The reason is explained as follows. Suppose that (x_1, y_1, l_{max}) is a 3-tuple in $L(n,m)$ with $l_{max} = \max\{k : (u, v, k) \in L(n,m)\}$ and P_1 denotes its corresponding almost increasing path in $T(n,m)$. Clearly, $T(x_1, y_1)$ is the representative entry in P_1. Suppose further that $Prev(x_1, y_1) = (x_2, y_2)$. Then we can utilize the value of $Prev(x_1, y_1)$ to find another entry $T(x_2, y_2)$ in P_1 that occurs before (but not necessarily immediately before) $T(x_1, y_1)$. Let P'_1 be the subpath of P_1 from the position of $T(x_1, y_1)$ to the end of P_1, and $P_2 = P_1 - P'_1$ (meaning that P_2 is obtained from P_1 by removing its P'_1). It is not hard to see that $T(x_2, y_2)$ becomes the representative entry of P_2 and P'_1 is a kind of *special* almost increasing subpath whose representative is the first entry of P'_1. By continuing this argument, we actually can partition P_1 into several subpaths, say $P'_1, P'_2, \ldots, P'_k$, such that each P'_i is a special almost increasing subpath, where $1 \le i \le k$, and moreover $\langle T(x_k, y_k), T(x_{k-1}, y_{k-1}), \ldots, T(x_1, y_1)\rangle$, which are obtained through the *Prev* table, form a partial result of P_1 (i.e., a subsequence of $\beta(P_1)$). Therefore, to recover the complete result of P_1, as described in Procedure 3, we need to solve another problem called *specialized LCaIS problem*.

Specialized LCaIS problem
Input: Two sequences $A = \langle a_1, a_2, \ldots, a_n\rangle$ and $B = \langle b_1, b_2, \ldots, b_m\rangle$ with $a_1 = b_1$ and a constant $c > 0$.
Output: A longest almost increasing path in the matching table T for A and B such that $T(1,1)$ is its representative point.

Let $\delta(i,j)$ be the length of $LP^i_j(1,1)$ in sub-table $T_{i,j}$. According to Lemma 1, we can solve the specialized LCaIS problem by the following recursive formula. For all $2 \le i \le n$ and $2 \le j \le m$,

$$\delta(i,j) = \begin{cases} \delta(i-1, j-1) + 1, & \text{if } \alpha(1,1) - c < \alpha(i,j) < \alpha(1,1) \\ \max\{\delta(i-1, j), \delta(i, j-1)\}, & \text{otherwise.} \end{cases}$$

Procedure 3 Backtrack($L(n,m)$, $Prev$).

Input: Set $L(n,m)$ and table $Prev$.
Output: An LCaIS of two input sequences A and B.
1: Let $(top, left, l_{max}) \in L(n,m)$ with $l_{max} = \max\{k : (u,v,k) \in L(n,m)\}$;
2: $(bottom, right) = (n,m)$;
3: Let S be a specialized LCaIS of $A_{top,bottom}$ and $B_{left,right}$;
4: **while** $(top, left) \neq (0,0)$ **do**
5: $(bottom, right) = (top-1, left-1)$;
6: $(top, left) = Prev(top, left)$;
7: Let S' be a specialized LCaIS of $A_{top,bottom}$ and $B_{left,right}$;
8: $S = S' \cdot S$; /* $S' \cdot S$ denotes the concatenation of S' and S */
9: **end while**

The boundary condition of the above formula is that $\delta(i,1) = \delta(1,j) = 1$ for all $1 \le i \le n$ and $1 \le j \le m$. It is interesting that this formula is very similar to that for computing the LCS between two strings. Therefore, we can solve the specialized LCaIS problem in quadratic time and space using the method similar to that for solving the traditional LCS problem [2–5].

Clearly, the space used for each $L(i,j)$ in Algorithm 1 is $O(l)$, where l denotes the length of an LCaIS of the two input sequences. Since we can use two alternative rows to fill the entire table L and the size of the table $Prev$ is $O(nm)$, the space complexity of Algorithm 1 is $O(m) \times O(l) + O(nm) = O(nm)$. The time for constructing $L(i,j)$ is $O(l)$ regardless of whether (i,j) is a matched pair or not. Moreover, the backtracking procedure for obtaining the specialized LCaIS needs $O(nm)$ time. As a result, the total time complexity of Algorithm 1 is $O(nm) \times O(l) + O(nm) = O(nml)$.

Theorem 1. *Algorithm 1 solves the LCaIS problem in $O(nml)$ time and $O(nm)$ space, where n and m are the lengths of two input sequences and l is the length of their LCaIS.*

Note that if only the length of the LCaIS between two input sequences is computed, then it is not hard to see that our algorithm can do this job by using only $O(l\min(m,n))$ space.

4. Conclusion

In this work, we studied the LCaIS problem on two sequences even with repeated elements. We presented a correct dynamic programming algorithm to solve this LCaIS problem in $O(nml)$ time and $O(nm)$ space, where n and m are the lengths of the two input sequences and l is the length of their LCaIS. Our algorithm has overcome the limitation and shortcoming in the algorithm proposed by Moosa et al. [1], which does not allow repeated elements in its input sequences and can produce erroneous results for some instances. Recall that according to the definition of the LCaIS problem, the input parameter c is required to be positive (i.e., $c > 0$). However, if it is allowed to be zero (i.e., $c = 0$), then the LCaIS problem becomes the LCIS problem. From the point of this view, the LCIS problem can be considered as a special case of the LCaIS problem. In the matching table, moreover, the representative entry of an increasing path for the LCIS problem is always at the last position of the path, while the representative entry of an almost increasing path for the LCaIS problem can be at any position. Currently, the LCIS problem is already known to be solvable in $O(nm)$ time [6]. Therefore, it will be interesting in future to study how to design more time efficient algorithms to solve the LCaIS problem.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by Ministry of Science and Technology of Taiwan under grant MOST107-2221-E-007-066-MY2.

References

[1] J.M. Moosa, M.S. Rahman, F.T. Zohora, Computing a longest common subsequence that is almost increasing on sequences having no repeated elements, J. Discret. Algorithms 20 (2013) 12–20.
[2] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, 2007.
[3] D. Gusfield, Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology, Cambridge University Press, 1997.
[4] D.S. Hirschberg, Algorithms for the longest common subsequence problem, J. ACM 24 (1977) 664–675.
[5] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1974) 168–173.
[6] I.-H. Yang, C.-P. Huang, K.-M. Chao, A fast algorithm for computing a longest common increasing subsequence, Inf. Process. Lett. 93 (2005) 249–253.
[7] A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White, S.L. Salzberg, Alignment of whole genomes, Nucleic Acids Res. 27 (1999) 2369–2376.
[8] A.L. Delcher, A. Phillippy, J. Carlton, S.L. Salzberg, Fast algorithms for large-scale genome alignment and comparison, Nucleic Acids Res. 30 (2002) 2478–2483.

[9] D.E. Knuth, Permutations, matrices, and generalized young tableaux, Pac. J. Math. 34 (1970) 709–727.
[10] P. Ramanan, Tight Ω(n lg n) lower bound for finding a longest increasing subsequence, Int. J. Comput. Math. 65 (1997) 161–164.
[11] C. Schensted, Longest increasing and decreasing subsequences, Can. J. Math. 13 (1961) 179–191.
[12] A. Elmasry, The longest almost-increasing subsequence, Inf. Process. Lett. 110 (2010) 655–658.