The Most and Longest Common Increasing Interval Subsequence Problems

Guan-Jie Chen, Chang-Biau Yang and Kuo-Si Huang

***Abstract*—The longest common increasing subsequence (LCIS) problem is to find the common increasing subsequence with the maximum length between two given sequences. In this paper, we define two new problems derived from the LCIS problem, the most common increasing interval subsequence (MCIIS) problem and the longest common increasing interval subsequence (LCIIS) problem. These two problems extend the LCIS problem by replacing single values with intervals. The MCIIS problem aims to identify the maximum common intervals formed between two interval sequences, where these intervals can build an increasing interval sequence with the maximum quantity of intervals. The LCIIS problem seeks to identify the maximum common intervals formed between two interval sequences, where these intervals can construct an increasing interval sequence whose union length of intervals is maximized. We propose algorithms for solving the MCIIS and the LCIIS problems in $O(n^4)$ and $O(n^3)$ time, respectively, where n denotes the number of intervals in the input interval sequences.**

***Index Terms*—longest common increasing subsequence, most common increasing interval subsequence, longest common increasing interval subsequence, interval, interval sequence**

I. Introduction

A sequence, comprised of a series of data elements arranged in a specific order, is commonly found in a variety of applications. Sequences have been extensively studied in various fields. For instance, DNA sequences enable the determination of species characteristics and behaviors. When exploring evolutionary relationships between species, comparing the similarity of DNA sequences serves as a valuable tool, such as FASTA [16] and BLAST [2]. In addition to character sequences like DNA or protein sequences, numeric sequences play a crucial role in our lives, including time series [9, 19], stock prices [10], and data compression [6, 7, 11]. To extract additional insights from numeric sequences, many related problems have been proposed and investigated [5, 12, 15, 17, 20].

The *longest increasing subsequence* (LIS) problem [17] aims to find a strictly increasing subsequence with maximum length in a given sequence. For example, given a sequence $A = \langle 7, 1, 5, 8, 6, 9\rangle$, its LIS length is 4, where the LIS solution is $\langle 1, 5, 8, 9\rangle$ or $\langle 1, 5, 6, 9\rangle$.

The LIS problem was initially defined by Schensted [17] in 1961. For a given sequence of length n, Schensted [17] proposed an algorithm with a time complexity of $O(n \log n)$ by utilizing Young tableau and binary search techniques. Subsequently, Hunt and Szymanski [12] introduced an algorithm that leveraged matching pairs with the van Emde Boas tree [18], achieving an improved time complexity of $O(n \log \log n)$ when the input sequence is a permutation of {1, 2, ..., n}. Following this, Bespamyatnikh and Segal [4] devised a method to enumerate all LIS solutions with the same time complexity of $O(n \log \log n)$. Crochemore and Porat [8] further reduced the time complexity to $O(n \log \log l)$, where l represents the LIS length. Lastly, Alam and Rahman [1] proposed a divide-and-conquer algorithm with a time complexity of $O(n \log n)$.

The *longest common increasing subsequence* (LCIS) problem focuses on identifying a longest common subsequence that exhibits increasing values from two given sequences. For example, consider two sequences $A = \langle 2, 1, 5, 3, 8\rangle$ and $B = \langle 1, 3, 5, 4, 8, 2\rangle$. In this case, the LCIS solution can be either $\langle 1, 5, 8\rangle$ or $\langle 1, 3, 8\rangle$, both of length 3.

The LCIS problem was first proposed by Yang *et al.* [20] in 2005, who devised an algorithm that can find a solution in $O(mn)$ time, where m and n represent the lengths of the two input sequences. Subsequently, Kutz *et al.* [13] introduced an algorithm that achieves a solution in $O((m+nl) \log \log |\Sigma| +$ Sort) time, where l represents the LCIS length, $|\Sigma|$ denotes the alphabet size, and Sort represents the time required to sort each input sequence. Additionally, Lo *et al.* [15] designed a diagonal-based algorithm, which can find a solution in $O((n+l(m-l)) \log \log |\Sigma|)$ time.

The *most increasing interval subsequence* (MIIS) problem and the *longest increasing interval subsequence* (LIIS) problem, proposed by Chen and Yang [5], are variations of the LIS problem where each element in the original LIS problem is replaced by an interval. The MIIS problem aims to find an increasing interval subsequence that has the maximum quantity of intervals, while the LIIS problem focuses on identifying an increasing interval subsequence whose union length of intervals is maximized. For example, given an interval sequence $X = \langle[1, 3], [7, 8], [2, 3], [9, 11], [4, 4], [5, 6], [10, 10]\rangle$, the solution to the MIIS problem is $\langle[1, 3], [4, 4], [5, 6], [10, 10]\rangle$ or $\langle[2, 3], [4, 4], [5, 6], [10, 10]\rangle$ containing four intervals (comparison by full interval), and the solution to the LIIS problem is $\langle[1, 3], [7, 8], [9, 11]\rangle$ with a total length of 8 (comparison by full interval). The related studies on the MIIS

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

Guan-Jie Chen is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.

Chang-Biau Yang is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan. E-mail: cbyang@cse.nsysu.edu.tw. (Corresponding author)

Kuo-Si Huang is with Department of Business Computing, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. E-mail: huangks@nkust.edu.tw.

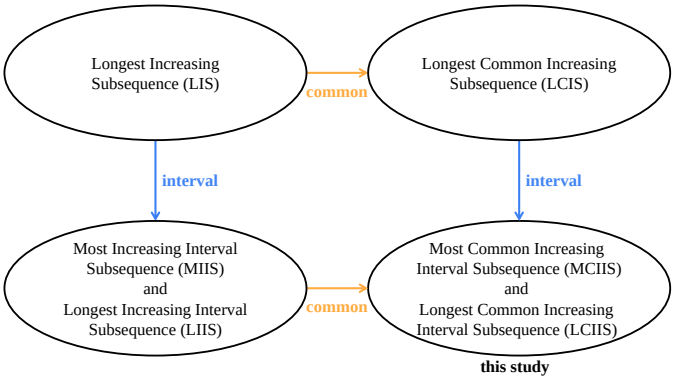

Fig. 1: Extension flow of the problems.

problem and the LIIS problem are summarized in Table I.

Similar to the relationship between the LIS problem [17] and the MIIS problem (as well as the LIIS problem) [5], this paper introduces the *most common increasing interval subsequence* (MCIIS) problem and the *longest common increasing interval subsequence* (LCIIS) problem as the interval versions of the LCIS problem [20]. Figure 1 shows the extension flow of the problems. Then, we devise algorithms for solving the MCIIS and LCIIS problems based on the dynamic programming (DP) approach [3, 14]. By leveraging specialized data structures and the properties of relationships between intervals, we can significantly reduce the required computational time and space. Our algorithms can solve the MCIIS problem in $O(n^4)$ time and space, and the LCIIS problem in $O(n^3)$ time and space. Additionally, through space reuse, we further reduce the space complexity to $O(n^3)$ for the MCIIS problem and $O(n^2)$ for the LCIIS problem.

The rest of this paper is organized as follows. In Section II, we review the preliminaries of the LCIS, MIIS and LIIS problems, and define the MCIIS and LCIIS problems. We present our algorithms for solving the proposed problems in Sections III and IV. Finally, in Section V, we present our conclusions and outline future research directions.

II. Preliminaries

The MCIIS problem and the LCIIS problem are variations of the LCIS problem. This chapter first introduces the LCIS problem, the MIIS problem and the LIIS problem, followed by the definitions of the MCIIS problem and the LCIIS problem.

A. Longest Common Increasing Subsequence

Given two sequences $A = \langle a_1, a_2, \ldots, a_m \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$, the longest common increasing subsequence (LCIS) problem aims to find an increasing subsequence $C = \langle c_1, c_2, \ldots, c_l \rangle$, where $c_1 < c_2 < \ldots < c_l$, and l is maximized. Each c_k (for $1 \leq k \leq l$) represents a common element $a_{i_k} = b_{j_k}$, satisfying that $1 \leq i_1 < i_2 < \ldots < i_l \leq m$ and $1 \leq j_1 < j_2 < \ldots < j_l \leq n$. For instance, $A = \langle 2, 1, 5, 3, 8 \rangle$ and $B = \langle 1, 3, 5, 4, 8, 2 \rangle$, the LCIS of A and B is $\langle 1, 5, 8 \rangle$ or $\langle 1, 3, 8 \rangle$, both of length 3.

B. Most Increasing Interval Subsequence

The most increasing interval subsequence (MIIS) problem, proposed by Chen and Yang [5], aims to find the increasing interval subsequence containing the maximum quantity of intervals for the input interval sequence. The quantity of MIIS is similar to the answer length in the LIS problem. The MIIS problem has two variations: comparison by full interval and comparison by a subinterval. Here, we introduce the MIIS variant with comparison by full interval.

Definition 1 (interval [5])**.** *$x = [x^s, x^e]$ denotes an interval x which starts from x^s and ends with x^e, where $x^s \leq x^e$.*

Definition 2 (comparison by full interval [5])**.** *Let $x = [x^s, x^e]$ and $y = [y^s, y^e]$ be two intervals. If $x^e < y^s$, the relation $x <_f y$ holds.*

Given an interval sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$, the MIIS problem (comparison by full interval) involves removing certain intervals from X, resulting in an interval sequence $C = \langle c_1, c_2, \ldots, c_q \rangle$, where $c_1 <_f c_2 <_f \ldots <_f c_q$, and the quantity q is maximized. For instance, $X = \langle [1,3], [7,8], [2,3], [9,11], [4,4], [5,6], [10,10] \rangle$, the MIIS answer is $\langle [1,3], [4,4], [5,6], [10,10] \rangle$ or $\langle [2,3], [4,4], [5,6], [10,10] \rangle$ which has four intervals.

Chen and Yang [5] devised an algorithm that can solve the MIIS problem in $O(n \log n)$ time.

C. Longest Increasing Interval Subsequence

The LIIS problem, also introduced by Chen and Yang [5], aims to find an increasing interval subsequence with the maximum union length of intervals. This problem has two variations: comparison by full interval and comparison by a subinterval. In this section, we introduce the LIIS variant with comparison by full interval.

Definition 3 (length of an interval [5])**.** *Let $x = [x^s, x^e]$ be an interval. The length of x, denoted as $|x|$, is defined as $|x| = x^e - x^s + 1$.*

Definition 4 (length of an interval sequence [5])**.** *An interval sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$, where each $x_i = [x_i^s, x_i^e]$ for $1 \leq i \leq n$, whose length is denoted as $|X|$. This length is calculated as the sum of the individual lengths of all intervals in X, where any overlapping part is counted only once.*

Given an interval sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$, the LIIS problem (comparison by full interval) aims to maximize $|C|$, where $C = \langle c_1, c_2, \ldots, c_q \rangle$ is an interval sequence derived from X such that $c_1 <_f c_2 <_f \ldots <_f c_q$, achieved by removing certain intervals from X. For example, given $X = \langle [1,3], [7,8], [2,3], [9,11], [4,4], [5,6], [10,10] \rangle$, the answer is $\langle [1,3], [7,8], [9,11] \rangle$ with interval sequence length 8.

Chen and Yang [5] designed an algorithm for solving the LIIS problem in $O(n \log n)$ time.

D. Most Common Increasing Interval Subsequence

Definition 5 (maximal common interval)**.** *Given two intervals $x = [x^s, x^e]$ and $y = [y^s, y^e]$, if there exists an interval $c =$*

TABLE I: A summary of the MIIS and the LIIS problems [5], as well as the proposed MCIIS and LCIIS problems. m and n: the numbers of intervals in the interval sequences X and Y, respectively, where $m \leq n$.

Year	Author(s)	Time complexity	Note
The most increasing interval subsequence (MIIS) problem			
2025	Chen and Yang [5]	$O(n \log n)$ $O(n \log^2 n)$	one interval sequence, greedy, binary search, domination
The longest increasing interval subsequence (LIIS) problem			
2025	Chen and Yang [5]	$O(n \log n)$	one interval sequence, greedy, binary search, domination
The most common increasing interval subsequence (MCIIS) problem			
2025	This paper	$O(n^4)$	two interval sequences, domination, dynamic programming
The longest common increasing interval subsequence (LCIIS) problem			
2025	This paper	$O(n^3)$	two interval sequences, domination, dynamic programming

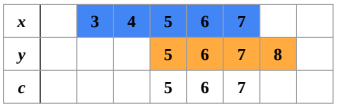

Fig. 2: An example of a maximal common interval.

$[c^{\mathrm{s}}, c^{\mathrm{e}}]$ *such that* $c^{\mathrm{s}} = \max(x^{\mathrm{s}}, y^{\mathrm{s}})$, $c^{\mathrm{e}} = \min(x^{\mathrm{e}}, y^{\mathrm{e}})$ *and* $c^{\mathrm{s}} \leq c^{\mathrm{e}}$, *then* c *is a maximal common interval of* x *and* y. *If* $c^{\mathrm{s}} > c^{\mathrm{e}}$, *then a maximal common interval does not exist.*

Figure 2 shows an example of a maximal common interval $c = [5, 7]$, constructed from $x = [3, 7]$ and $y = [5, 8]$, which is the intersection of x and y.

Definition 6 (interval comparison [5])**.** *Let* $x = [x^{\mathrm{s}}, x^{\mathrm{e}}]$ *and* $y = [y^{\mathrm{s}}, y^{\mathrm{e}}]$ *be two intervals. If* $x^{\mathrm{s}} < y^{\mathrm{s}}$ *and* $x^{\mathrm{e}} < y^{\mathrm{e}}$, *the relation* $x < y$ *holds.*

Definition 7 (increasing interval sequence)**.** *Given an interval sequence* $C = \langle c_1, c_2, \ldots, c_q \rangle$. *If* $c_1 < c_2 < \ldots < c_q$, *then* C *is an increasing interval sequence.*

By Definition 7, given an increasing interval sequence $C = \langle c_1, c_2, \ldots, c_q \rangle$, where $c_k = [c_k^{\mathrm{s}}, c_k^{\mathrm{e}}]$ for $1 \leq k \leq q$, C also satisfies $c_1^{\mathrm{s}} < c_2^{\mathrm{s}} < \ldots < c_q^{\mathrm{s}}$ and $c_1^{\mathrm{e}} < c_2^{\mathrm{e}} < \ldots < c_q^{\mathrm{e}}$.

Definition 8 (MCIIS problem)**.** *Given two interval sequences* $X = \langle x_1, x_2, \ldots, x_m \rangle$ *and* $Y = \langle y_1, y_2, \ldots, y_n \rangle$, *the* most common increasing interval subsequence (MCIIS) *problem is to find an increasing interval subsequence* $C = \langle c_1, c_2, \ldots, c_q \rangle$ *that achieves the maximum quantity* q. *Each* c_k, $1 \leq k \leq q$, *represents the maximal common interval between* x_{i_k} *and* y_{j_k}, *under the condition that* $1 \leq i_1 < i_2 < \ldots < i_q \leq m$ *and* $1 \leq j_1 < j_2 < \ldots < j_q \leq n$.

Table II presents an aligned example of the MCIIS problem with two given interval sequences $X = \langle [1, 6], [12, 14], [4, 12], [10, 11] \rangle$ and $Y = \langle [3, 11], [6, 9], [8, 17], [5, 7], [3, 14] \rangle$. The bottom row displays the maximal common intervals found between X and Y. The MCIIS answer is $\langle [3, 6], [5, 7], [10, 11] \rangle$, consisting of 3 intervals.

TABLE II: An aligned example of the MCIIS problem, where $X = \langle [1, 6], [12, 14], [4, 12], [10, 11] \rangle$ and $Y = \langle [3, 11], [6, 9], [8, 17], [5, 7], [3, 14] \rangle$. The MCIIS answer is $\langle [3, 6], [5, 7], [10, 11] \rangle$ with three intervals.

X	[1, 6]	[12, 14]	-	-	[4, 12]	[10, 11]
Y	[3, 11]	-	[6, 9]	[8, 17]	[5, 7]	[3, 14]
max. common interval	[3, 6]	-	-	-	[5, 7]	[10, 11]

In Table II, intervals $x_1 = [1, 6]$ and $y_1 = [3, 11]$ yield the maximal common interval $[3, 6]$ ($\max(1, 3) = 3$, $\min(6, 11) = 6$, for $[3, 6] = [1, 6] \cap [3, 11]$). Similarly, $x_3 = [4, 12]$ and $y_4 = [5, 7]$ result in the maximal common interval $[5, 7]$ ($\max(4, 5) = 5$, $\min(12, 7) = 7$, for $[5, 7] = [4, 12] \cap [5, 7]$), and $x_4 = [10, 11]$ and $y_5 = [3, 14]$ produce the maximal common interval $[10, 11]$ ($\max(10, 3) = 10$, $\min(11, 14) = 11$, for $[10, 11] = [10, 11] \cap [3, 14]$). Furthermore, $\langle [3, 6], [5, 7], [10, 11] \rangle$ is an increasing interval sequence. Among the common increasing interval subsequences found between X and Y, the maximum quantity of intervals is 3. Therefore, $\langle [3, 6], [5, 7], [10, 11] \rangle$ is the MCIIS answer of this example. Note that there may exist multiple MCIIS answers with the maximum quantity of intervals.

E. Longest Common Increasing Interval Subsequence

Definition 9 (LCIIS problem)**.** *Given two interval sequences* $X = \langle x_1, x_2, \ldots, x_m \rangle$ *and* $Y = \langle y_1, y_2, \ldots, y_n \rangle$, *the* longest common increasing interval subsequence (LCIIS) *problem is to find an increasing interval subsequence* $C = \langle c_1, c_2, \ldots, c_q \rangle$ *such that* $l = |\cup \{c_k \in C\}|$ *is maximized. Each* c_k, $1 \leq k \leq q$, *represents the maximal common interval between* x_{i_k} *and* y_{j_k}, *where* $1 \leq i_1 < i_2 < \ldots < i_q \leq m$ *and* $1 \leq j_1 < j_2 < \ldots < j_q \leq n$.

TABLE III: An aligned example of the LCIIS problem, where $X = \langle [1,6],\ [12,14],\ [4,12],\ [10,11] \rangle$ and $Y = \langle [3,11],\ [6,9],\ [8,17],\ [5,7],\ [3,14] \rangle$. The LCIIS answer is $\langle [3,6],\ [4,12] \rangle$.

X	[1, 6]	[12, 14]	-	-	-	[4, 12]	[10, 11]
Y	[3, 11]	-	[6, 9]	[8, 17]	[5, 7]	[3, 14]	-
max. common interval	[3, 6]	-	-	-	-	[4, 12]	-
interval length	4	-	-	-	-	9	-

Table III presents an aligned example of the LCIIS problem with the given interval sequences $X = \langle [1,6],\ [12,14],\ [4,12],\ [10,11] \rangle$ and $Y = \langle [3,11],\ [6,9],\ [8,17],\ [5,7],\ [3,14] \rangle$. The bottom row displays the length of the maximal common intervals found between X and Y. The LCIIS answer is $\langle [3,6],\ [4,12] \rangle$, with a total length of 10 for $[3,12] = [3,6] \cup [4,12]$. Note that overlapping portions are counted only once. The length of this sequence is 10, $(12 - 3 + 1) = 10$ for $[3,12] = [3,6] \cup [4,12]$, or $(4 + 9 - 3) = 10$, where 3 is the length of overlapping part from 4 to 6.

III. Most Common Increasing Interval Subsequence

The MCIIS problem is an extension of the LCIS problem [20]. In the LCIS problem, we seek the longest strictly increasing subsequence that is common to both input sequences. In the MCIIS problem, each element in the sequences is replaced by an interval. As a result, the common parts found are no longer specific values but rather the overlapped segments of two intervals, each from its respective interval sequence.

The primary objective of the MCIIS problem is to identify the maximal common intervals between two interval sequences, where these intervals can generate the increasing interval sequence with the maximum quantity of intervals. In the following, we employ the dynamic programming (DP) approach to track each potential in a cell to evolve into the solution at any given point.

Definition 10 (domination of intervals [5])**.** *Given two intervals $x = [x^{\mathrm{s}}, x^{\mathrm{e}}]$ and $y = [y^{\mathrm{s}}, y^{\mathrm{e}}]$, if $x^{\mathrm{s}} \le y^{\mathrm{s}}$ and $x^{\mathrm{e}} \le y^{\mathrm{e}}$, we state that x dominates y.*

Table IV illustrates an example of finding the MCIIS for two given interval sequences $X = \langle [1,6],\ [12,14],\ [4,12],\ [10,11] \rangle$ and $Y = \langle [3,11],\ [6,9],\ [8,17],\ [5,7],\ [3,14] \rangle$. The entries highlighted in red and bold represent the maximal common intervals found. Each entry is represented as a 3-tuple $\langle q, c^{\mathrm{s}}_{k'}, c^{\mathrm{e}}_{k'} \rangle$, denoting $\langle q$ (quantity), the starting value of the maximal common interval $c_{k'}$ corresponding to this q, the ending value of the maximal common interval $c_{k'}$ corresponding to this $q\rangle$ in detail. In each cell, we check whether an entry can extend from the 3-tuple entries in the upper-left cell, and inherit 3-tuple entries from its upper and its left cells.

Definition 11 (domination of 3-tuple entries)**.** *Given two 3-tuple entries $\langle q_1, c^{\mathrm{s}}_1, c^{\mathrm{e}}_1 \rangle$ and $\langle q_2, c^{\mathrm{s}}_2, c^{\mathrm{e}}_2 \rangle$, where $q_1 = q_2 = q'$. If $c^{\mathrm{s}}_1 \le c^{\mathrm{s}}_2$ and $c^{\mathrm{e}}_1 \le c^{\mathrm{e}}_2$, $\langle q_1, c^{\mathrm{s}}_1, wesaythatc^{\mathrm{e}}_1 \rangle$ dominates $\langle q_2, c^{\mathrm{s}}_2, c^{\mathrm{e}}_2 \rangle$.*

In the scenario where several entries have the same quantity q', a 3-tuple entry that dominates other entries must have equal or better developmental potential. Consider an interval $[c^{\mathrm{s}}_3, c^{\mathrm{e}}_3]$ being generated. If $[c^{\mathrm{s}}_3, c^{\mathrm{e}}_3]$ can extend $\langle q', c^{\mathrm{s}}_2, c^{\mathrm{e}}_2 \rangle$ to become $\langle q' + 1, c^{\mathrm{s}}_3, c^{\mathrm{e}}_3 \rangle$ (i.e., $c^{\mathrm{s}}_2 < c^{\mathrm{s}}_3$ and $c^{\mathrm{e}}_2 < c^{\mathrm{e}}_3$), then $[c^{\mathrm{s}}_3, c^{\mathrm{e}}_3]$ must also be able to extend $\langle q', c^{\mathrm{s}}_1, c^{\mathrm{e}}_1 \rangle$ to $\langle q' + 1, c^{\mathrm{s}}_3, c^{\mathrm{e}}_3 \rangle$, because $c^{\mathrm{s}}_1 \le c^{\mathrm{s}}_2 < c^{\mathrm{s}}_3$ and $c^{\mathrm{e}}_1 \le c^{\mathrm{e}}_2 < c^{\mathrm{e}}_3$.

In row $x_1 = [1,6]$ of Table IV, x_1 overlaps with $y_1 = [3,11]$, resulting in the maximal common interval $[3,6]$. We record $\langle 1,3,6 \rangle$ in the cell, where 1 represents the quantity of the increasing interval sequence ending with $[3,6]$. Continuing from this, it is observed that a maximal common interval $[6,6]$ is discovered between $x_1 = [1,6]$ and $y_2 = [6,9]$. In cases where the quantities are equal ($q = 1$), the dominance principle favors $[3,6]$ over $[6,6]$. Consequently, only $\langle 1,3,6 \rangle$ is retained for further computational purposes. The maximal common interval $[5,6]$ generated by $x_1 = [1,6]$ and $y_4 = [5,7]$ will also be discarded as it is dominated by $[3,6]$.

In row $x_2 = [12,14]$, x_2 and $y_3 = [8,17]$ can generate the maximal common interval $[12,14]$. Since $[12,14]$ can extend the interval with quantity 1 from the upper-left cell, $[3,6]$, it results in $\langle 2,12,14 \rangle$. This indicates that there is an increasing interval sequence ending with $[12,14]$ with a quantity of 2. By inheriting $\langle 1,3,6 \rangle$ from the upper and the left cells, we obtain the final result for this cell.

In row $x_3 = [4,12]$, x_3 and $y_1 = [3,11]$ can generate $\langle 1,4,11 \rangle$, but $\langle 1,4,11 \rangle$ is dominated by $\langle 1,3,6 \rangle$. x_3 and $y_2 = [6,9]$ can extend $\langle 2,6,9 \rangle$ from $\langle 1,3,6 \rangle$ in its upper-left cell. Then x_3 and $y_3 = [8,17]$ can extend from the upper-left cell's $\langle 1,3,6 \rangle$ to $\langle 2,8,12 \rangle$. By inheriting from the upper and the left cells, we additionally have $\langle 1,3,6 \rangle$, $\langle 2,6,9 \rangle$ and $\langle 2,12,14 \rangle$. However, since the quantity $q = 2$ for $[6,9]$ dominates $[8,12]$ and $[12,14]$, only $\langle 1,3,6 \rangle$ and $\langle 2,6,9 \rangle$ will be retained.

After processing as described above, the maximum quantity q of the MCIIS problem is recorded in the most bottom-right cell of Table IV. In this example, the maximum value of q is 3, indicating that the maximal common intervals found between X and Y can form an increasing interval sequence with a maximum quantity of 3. The MCIIS answer $\langle [3,6],\ [5,7],\ [10,11] \rangle$ can be obtained through backtracking.

It is noteworthy that when $q = 2$, both $\langle 2,5,7 \rangle$ and $\langle 2,4,12 \rangle$ coexist, as $[5,7]$ and $[4,12]$ could potentially evolve into better solutions. For instance, if a subsequent maximal common interval $[6,9]$ is identified, it can extend from $[5,7]$ but not from $[4,12]$. Similarly, if another maximal common interval $[5,14]$ is discovered, it can extend from $[4,12]$ but not from $[5,7]$. Consequently, both 3-tuple entries must be retained to accurately represent the possible intervals and their potential for further optimization.

A. The Algorithms

For solving the MCIIS problem, Algorithm 1 presents the main procedure, which includes the Extend$(\cdot)$ function for extending entries from the upper-left cell and the Dominate$(\cdot)$ function responsible for eliminating dominated

TABLE IV: An example of finding MCIIS, where $X = \langle [1,6], [12,14], [4,12], [10,11] \rangle$ and $Y = \langle [3,11], [6,9], [8,17], [5,7], [3,14] \rangle$. The MCIIS answer is $\langle [3,6], [5,7], [10,11] \rangle$, consisting of 3 intervals.

X \ Y		y_1 [3, 11]	y_2 [6, 9]	y_3 [8, 17]	y_4 [5, 7]	y_5 [3, 14]
x_1	[1, 6]	**⟨1, 3, 6⟩**	~~**⟨1, 6, 6⟩**~~ ⟨1, 3, 6⟩	⟨1, 3, 6⟩	~~**⟨1, 5, 6⟩**~~ ⟨1, 3, 6⟩	**⟨1, 3, 6⟩**
x_2	[12, 14]	⟨1, 3, 6⟩	⟨1, 3, 6⟩	⟨1, 3, 6⟩ **⟨2, 12, 14⟩**	⟨1, 3, 6⟩ ⟨2, 12, 14⟩	⟨1, 3, 6⟩ **⟨2, 12, 14⟩**
x_3	[4, 12]	⟨1, 3, 6⟩ ~~**⟨1, 4, 11⟩**~~	⟨1, 3, 6⟩ **⟨2, 6, 9⟩**	⟨1, 3, 6⟩ ⟨2, 6, 9⟩ ~~**⟨2, 8, 12⟩**~~ ~~⟨2, 12, 14⟩~~	⟨1, 3, 6⟩ **⟨2, 5, 7⟩** ~~⟨2, 6, 9⟩~~ ~~⟨2, 12, 14⟩~~	⟨1, 3, 6⟩ ⟨2, 5, 7⟩ **⟨2, 4, 12⟩** ~~⟨2, 12, 14⟩~~
x_4	[10, 11]	⟨1, 3, 6⟩ ~~**⟨1, 10, 11⟩**~~	⟨1, 3, 6⟩ ⟨2, 6, 9⟩	⟨1, 3, 6⟩ ⟨2, 6, 9⟩ **⟨3, 10, 11⟩**	⟨1, 3, 6⟩ ⟨2, 5, 7⟩ ~~⟨2, 6, 9⟩~~ ⟨3, 10, 11⟩	⟨1, 3, 6⟩ ⟨2, 5, 7⟩ ⟨2, 4, 12⟩ **⟨3, 10, 11⟩**

entries. Functions EXTEND$(\cdot)$ and DOMINATE$(\cdot)$ are respectively demonstrated in Algorithms 2 and 3.

In Algorithm 1, $D_{i,j}$ records the ordered dominant set within the cell at row i and column j during the DP execution. Line 1 is the initialization step. Since we assume that all starting and ending values of intervals are positive integers, the maximal common intervals $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ shall also satisfy c^{s} and c^{e} being positive integers. By setting $D_{0,0}$, $D_{i,0}$ and $D_{0,j}$ to $\{\langle 0,0,0\rangle\}$ as the boundary conditions in the first row and the first column, it ensures that the maximal common interval can at least extend from $\langle 0,0,0\rangle$ during the execution of EXTEND$(\cdot)$ in Line 8.

Lines 2 to 8 demonstrate the core of the MCIIS algorithm. Line 4 executes the inheritance from $D_{i-1,j}$ and $D_{i,j-1}$. The maximal common interval generated by the two input interval sequences is calculated in Lines 5 and 6 (refer to Definition 5). Lines 7 and 8 illustrate how $D_{i,j}$ is generated when x_i and y_j have a maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$. First, $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ searches the upper-left cell by iterating through $D_{i-1,j-1}$ to find the 3-tuple with the maximum extendable quantity. Subsequently, the 3-tuple constructed from $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ undergoes domination with the 3-tuple entries in $D_{i,j}$ (where $D_{i,j}$ has already inherited entries from $D_{i-1,j}$ and $D_{i,j-1}$).

Algorithm 2 demonstrates how the EXTEND$(\cdot)$ function operates. Firstly, we initialize the pointer p to the first 3-tuple entry in the set $D_{i-1,j-1}$ that we are going to iterate through. For an entry $p = \langle p.\text{quantity}, p.\text{start}, p.\text{end}\rangle$, p.quantity refers to the quantity of this common increasing interval sequence, p.start and p.end refer to the starting value and the ending value of the maximal common interval, respectively.

During each iteration, p moves to the next 3-tuple entry in $D_{i-1,j-1}$ until it reaches the last one. When p points to a new entry, the input maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ is compared with the 3-tuple pointed by p. If an extension is feasible (i.e., if the condition in Line 3 is satisfied), then q is updated to p.quantity $+ 1$. Upon executing EXTEND$(\cdot)$, it signifies the existence of a maximal common interval between x_i and y_j, ensuring the creation of a 3-tuple entry $\langle q, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle$ with quantity $q \geq 1$. Hence, we initialize q as 0, guaranteeing that encountering $\langle 0,0,0\rangle$ in the set can be extended to $\langle 1, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle$.

After iterating through all 3-tuple entries, q is updated to its feasible maximum value. The condition $q \leq p$.quantity in Line 3 ensures that the calculation proceeds only when it is possible to improve the result. This optimization avoids unnecessary computations and helps improve the efficiency and accuracy of the algorithm.

Algorithm 1 The algorithm for solving the MCIIS problem.

Input: Two interval sequences $X = \langle x_1, x_2, \ldots, x_m\rangle$ and $Y = \langle y_1, y_2, \ldots, y_n\rangle$, where $x_i = [x_i^{\mathrm{s}}, x_i^{\mathrm{e}}]$ and $y_j = [y_j^{\mathrm{s}}, y_j^{\mathrm{e}}]$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

Output: The maximum quantity q of common intervals included in the answer of MCIIS.

1: $D_{0,0} \leftarrow D_{i,0} \leftarrow D_{0,j} \leftarrow \{\langle 0,0,0\rangle\}$ for $1 \leq i \leq m$, $1 \leq j \leq n$ ▷ initialization
2: **for** $i \leftarrow 1$ to m **do**
3: **for** $j \leftarrow 1$ to n **do**
4: $D_{i,j} \leftarrow$ DOMINATE$(D_{i-1,j}, D_{i,j-1})$
5: $c^{\mathrm{s}} \leftarrow \max(x_i^{\mathrm{s}}, y_j^{\mathrm{s}})$
6: $c^{\mathrm{e}} \leftarrow \min(x_i^{\mathrm{e}}, y_j^{\mathrm{e}})$
7: **if** $c^{\mathrm{s}} \leq c^{\mathrm{e}}$ **then** ▷ maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ exists
8: $D_{i,j} \leftarrow$ DOMINATE(EXTEND($[c^{\mathrm{s}}, c^{\mathrm{e}}]$, $D_{i-1,j-1}$), $D_{i,j}$)
9: **return** $\max\{q | \langle q, \cdot, \cdot\rangle \in D_{m,n}\}$

Algorithm 2 The algorithm of EXTEND$([c^{\mathrm{s}}, c^{\mathrm{e}}], D_{i-1,j-1})$, which extends increasing interval sequence from the upper-left cell.

Input: A maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$, as well as the dominant set $D' = D_{i-1,j-1}$.

Output: A set containing a 3-tuple entry $\langle q, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle$.

1: $q \leftarrow 0$ ▷ initialization
2: **for** $p \leftarrow D'$.begin() to D'.end() **do** ▷ pointer p iterates over 3-tuples in D'
3: **if** $q \leq p$.quantity and $c^{\mathrm{s}} > p$.start and $c^{\mathrm{e}} > p$.end **then**
4: $q \leftarrow p$.quantity $+ 1$
5: **return** $\{\langle q, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle\}$

Algorithm 3 shows the flow of the DOMINATE(D_1', D_2') function. Initially, an empty set D'' is created to store 3-tuple entries, and pointers p_1 and p_2 are initialized to the first 3-tuple entries in D_1' and D_2', respectively. The entries in D'' are sorted in ascending order of q. Intervals with the same q indicate non-dominance and can coexist. The insertion begins with intervals having the maximum starting value (or minimum ending value) and proceeds until all intervals with the same q are processed before advancing to larger q values.

Lines 2 to 15 demonstrate the operation when both sets have not been fully traversed. When p_1 and p_2 point to entries with the same q, if one entry dominates the other, the dominated entry is discarded by advancing its pointer to the next entry in its set. If there is no domination between p_1 and p_2, the

one with a larger starting value and smaller ending value is appended at the end of D'', and then its pointer is moved to the next. When p_1 and p_2 point to entries with different q, the one with the smaller q is appened to D'', and its pointer is then moved to the next.

Lines 16 to 19 demonstrate that when one of D'_1 and D'_2 has been fully traversed, the remaining entries in the set that has not been fully traversed yet are sequentially appended to D''. Since the entries in D'_1 and D'_2 are also ordered according to the rules of D'', when fetching entries, we start with the most enclosed ones. When q values are the same, only the entry is appended to D'' if its interval entirely contains another, ensuring that D'' is mutually non-dominated and sorted according to the rules. If q values are different, it means that the pointer associated with the larger q has already traversed the entries with the smaller q. To increase q, either the entries with the smaller q have been dominated or successfully appended to D''. In both cases, the other pointer can still traverse the remaining entries with smaller q into D'', maintaining the integrity of data structure of D''.

Algorithm 3 The algorithm of DOMINATE(D'_1, D'_2), which merges D'_1 and D'_2 and removes the dominated entries.

Input: Two dominant sets D'_1 and D'_2 containing 3-tuple entries.
Output: A dominant set D'' containing 3-tuple entries.

1: $D'' \leftarrow \emptyset$, $p_1 \leftarrow D'_1$.begin(), $p_2 \leftarrow D'_2$.begin() ▷ initialization
2: **while** $p_1 \neq D'_1$.end() and $p_2 \neq D'_2$.end() **do**
3: **if** p_1.quantity $= p_2$.quantity **then**
4: **if** p_1.start $> p_2$.start and p_1.end $< p_2$.end **then**
5: D''.append(p_1), $p_1 \leftarrow p_1$.next()
6: **else if** p_1.start $< p_2$.start and p_1.end $> p_2$.end **then**
7: D''.append(p_2), $p_2 \leftarrow p_2$.next()
8: **else if** p_1.start $\leq p_2$.start and p_1.end $\leq p_2$.end **then** ▷ p_1 dominates p_2
9: $p_2 \leftarrow p_2$.next() ▷ discard p_2
10: **else** ▷ p_2 dominates p_1
11: $p_1 \leftarrow p_1$.next() ▷ discard p_1
12: **else if** p_1.quantity $< p_2$.quantity **then**
13: D''.append(p_1), $p_1 \leftarrow p_1$.next()
14: **else**
15: D''.append(p_2), $p_2 \leftarrow p_2$.next()
16: **while** $p_1 \neq D'_1$.end() **do**
17: D''.append(p_1), $p_1 \leftarrow p_1$.next()
18: **while** $p_2 \neq D'_2$.end() **do**
19: D''.append(p_2), $p_2 \leftarrow p_2$.next()
20: **return** D''

B. Complexity Analysis

Our algorithm solves the MCIIS problem in $O(n^4)$ time and space, where $m \leq n$. The EXTEND$(\cdot)$ operation traverses all entries within a cell in $O(n^2)$ time. The DOMINATE(D'_1, D'_2) operation merges two sets D'_1 and D'_2, each containing $O(n^2)$ entries, where pointers in these sets are moving in one direction until traversal is complete, also requiring $O(n^2)$ time.

x		3	4	5	6	7		
y				5	6	7	8	
z		3	4	5	6	7	8	

Fig. 3: An example of the interval combination.

Theorem 1. *Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, where $m \leq n$. In each cell, the number of 3-tuples in the dominant set is at most $m \times n$.*

Proof. The starting and ending values of a maximal common interval are identified by comparing the starting values x_i^{s} and y_j^{s} and ending values x_i^{e} and y_j^{e} of x_i and y_j, respectively, where $1 \leq i \leq m$ and $1 \leq j \leq n$. Therefore, X and Y can generate at most $m \times n$ maximal common intervals. Furthermore, each maximal common interval is extended only once, from its upper-left cell by extending the feasible 3-tuple with the largest q. Hence, there will be at most $m \times n$ 3-tuples in each cell. □

IV. LONGEST COMMON INCREASING INTERVAL SUBSEQUENCE

The primary objective of the LCIIS problem is to identify the maximal common intervals formed between two interval sequences, where these intervals can generate the increasing interval sequence with the maximum length of their union. Here, we still rely on dynamic programming (DP) approach to maintain the feasible entries corresponding to the LCIIS.

Definition 12 (domination among increasing interval sequences [5])**.** *Given two increasing interval sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, where $x_i = [x_i^{\mathrm{s}}, x_i^{\mathrm{e}}]$ and $y_j = [y_j^{\mathrm{s}}, y_j^{\mathrm{e}}]$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. X is considered to dominate Y if both conditions $|X| \geq |Y|$ and $|X| - |Y| \geq x_m^{\mathrm{e}} - y_n^{\mathrm{e}}$ hold.*

Definition 13 (combination of two contiguous intervals)**.** *Given two intervals $x = [x^{\mathrm{s}}, x^{\mathrm{e}}]$ and $y = [y^{\mathrm{s}}, y^{\mathrm{e}}]$ such that $x < y$, if $y^{\mathrm{s}} - x^{\mathrm{e}} \leq 1$, the two intervals can combine into one interval $z = [z^{\mathrm{s}}, z^{\mathrm{e}}]$, where $z^{\mathrm{s}} = x^{\mathrm{s}}$ and $z^{\mathrm{e}} = y^{\mathrm{e}}$.*

Figure 3 shows an example of combining two contiguous intervals, $x = [3, 7]$ and $y = [5, 8]$. The resulting interval, $z = [3, 8]$, is the union of x and y.

When calculating the LCIIS length, overlapping segments between intervals are only counted once. Therefore, when intervals are contiguous, we can combine the two intervals into a single interval with Definition 13, thereby maintaining the correctness of the result while reducing space usage. The proof will be presented in Theorem 2.

For solving the LCIIS problem, we will first introduce the 3-tuple algorithm, followed by the 2-tuple algorithm. For a 3-tuple entry $\langle l, c^{\mathrm{s}}, c^{\mathrm{e}} \rangle$ in the 3-tuple algorithm, l denotes the length of a corresponding LCIIS, c^{s} and c^{e} denote the starting

TABLE V: An example of finding LCIIS by using the 3-tuple entry $\langle l, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle$, where $X = \langle[1,6],\ [12,14],\ [4,12],\ [10,11]\rangle$ and $Y = \langle[3,11],\ [6,9],\ [8,17],\ [5,7],\ [3,14]\rangle$. The answer is $\langle[3,6],\ [4,12]\rangle$, with a total length of 10.

X \ Y		y_1 [3, 11]	y_2 [6, 9]	y_3 [8, 17]	y_4 [5, 7]	y_5 [3, 14]
x_1	[1, 6]	**⟨4, 3, 6⟩**	~~⟨1, 6, 6⟩~~ ⟨4, 3, 6⟩	⟨4, 3, 6⟩	~~⟨2, 5, 6⟩~~ ⟨4, 3, 6⟩	**⟨4, 3, 6⟩**
x_2	[12, 14]	⟨4, 3, 6⟩	⟨4, 3, 6⟩	⟨4, 3, 6⟩ **⟨7, 12, 14⟩**	⟨4, 3, 6⟩ ⟨7, 12, 14⟩	⟨4, 3, 6⟩ **⟨7, 12, 14⟩**
x_3	[4, 12]	⟨4, 3, 6⟩ **⟨8, 4, 11⟩**	~~⟨4, 3, 6⟩~~ **⟨7, 3, 9⟩** ⟨8, 4, 11⟩	~~⟨4, 3, 6⟩~~ ⟨7, 3, 9⟩ ~~⟨7, 12, 14⟩~~ ~~⟨8, 4, 11⟩~~ **⟨9, 8, 12⟩**	~~⟨4, 3, 6⟩~~ ~~⟨5, 3, 7⟩~~ ⟨7, 3, 9⟩ ~~⟨7, 12, 14⟩~~ ⟨9, 8, 12⟩	~~⟨4, 3, 6⟩~~ ~~⟨7, 3, 9⟩~~ ~~⟨7, 12, 14⟩~~ ~~⟨9, 8, 12⟩~~ **⟨10, 3, 12⟩**
x_4	[10, 11]	~~⟨2, 10, 11⟩~~ ⟨4, 3, 6⟩ ⟨8, 4, 11⟩	~~⟨4, 3, 6⟩~~ ⟨7, 3, 9⟩ ⟨8, 4, 11⟩	~~⟨7, 3, 9⟩~~ ~~⟨8, 4, 11⟩~~ **⟨9, 3, 11⟩** ~~⟨9, 8, 12⟩~~	~~⟨7, 3, 9⟩~~ ⟨9, 3, 11⟩ ~~⟨9, 8, 12⟩~~	~~⟨9, 3, 11⟩~~ ⟨10, 3, 12⟩

and ending values of the last maximal common interval in this common increasing interval subsequence determined by the l, respectively. In each cell, we verify its ability to extend entries in the upper-left cell and inherit data from both its upper and left cells.

Table V shows an example with two given interval sequences $X = \langle[1,6],\ [12,14],\ [4,12],\ [10,11]\rangle$ and $Y = \langle[3,11],\ [6,9],\ [8,17],\ [5,7],\ [3,14]\rangle$. In row $x_1 = [1,6]$, x_1 and $y_1 = [3,11]$ construct the maximal common interval $[3,6] = [1,6] \cap [3,11]$ with length 4. We record $\langle 4,3,6\rangle$ in this cell, where $l = 4$ represents the length of the increasing interval sequence ending with $[3,6]$. Additionally, $x_1 = [1,6]$ and $y_2 = [6,9]$ generate the maximal common interval $[6,6] = [1,6] \cap [6,9]$ with length 1, but $\langle 1,6,6\rangle$ is dominated by $\langle 4,3,6\rangle$ (since $4-1 \geq 6-6$). Similarly, x_1 and $y_4 = [5,7]$ generate $\langle 2,5,6\rangle$ which is also dominated by $\langle 4,3,6\rangle$.

In row $x_2 = [12,14]$, x_2 and $y_3 = [8,17]$ generate the maximal common interval $[12,14] = [12,14] \cap [8,17]$ with length 3, which can extend $\langle 4,3,6\rangle$ in the upper-left cell to become $\langle 7,12,14\rangle$ ($4+3=7$ for union $[3,6] \cup [12,14]$). Because $[3,6]$ and $[12,14]$ are not contiguous, $[3,6]$ may potentially develop further to dominate the result $\langle 7,12,14\rangle$. Therefore, $\langle 4,3,6\rangle$ must be retained.

In row $x_3 = [4,12]$, x_3 and $y_1 = [3,11]$ generate the maximal common interval $[4,11] = [4,12] \cap [3,11]$ with length 8, and $\langle 8,4,11\rangle$ can coexist with $\langle 4,3,6\rangle$. Additionally, x_3 and $y_2 = [6,9]$ intersect the maximal common interval $[6,9] = [4,12] \cap [6,9]$ with length 4, and it is feasible by extending from $\langle 4,3,6\rangle$ in the upper-left cell. Since $[3,6]$ and $[6,9]$ are contiguous, we combine these two intervals into a single interval $[3,9]$, with length of 7. After inheriting $\langle 4,3,6\rangle$ and $\langle 8,4,11\rangle$ from the upper and the left cells, respectively, we observe that $\langle 4,3,6\rangle$ is dominated by $\langle 7,3,9\rangle$.

Theorem 2. *If two intervals can be combined (i.e., they are contiguous), we only retain the 3-tuple entry that represents the resulting interval after combining, and discard the two original intervals that were combined, following the principle of domination.*

Proof. Given two contiguous intervals $c_1 = [c_1^{\mathrm{s}}, c_1^{\mathrm{e}}]$ and $c_2 = [c_2^{\mathrm{s}}, c_2^{\mathrm{e}}]$ such that $c_1 < c_2$, the interval $c_3 = [c_1^{\mathrm{s}}, c_2^{\mathrm{e}}]$ denotes the interval after combining c_1 and c_2. If there exists an incoming interval $\hat{c} = [\hat{c}^{\mathrm{s}}, \hat{c}^{\mathrm{e}}]$ that can be extended from c_1, five cases arise:

Case 1: $\hat{c}^{\mathrm{s}} - c_1^{\mathrm{e}} \leq 1$ and $\hat{c}^{\mathrm{e}} \leq c_2^{\mathrm{e}}$ (c_1 and $\hat{c}$ are contiguous) Suppose the 3-tuple created by c_1 is kept, $\hat{c}$ will generate a 3-tuple $\langle \hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, \hat{c}^{\mathrm{e}}\rangle$, but this 3-tuple will finally be dominated by $\langle c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, c_2^{\mathrm{e}}\rangle$ created by c_3 (according to the domination rule $(c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1) - (\hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1) = c_2^{\mathrm{e}} - \hat{c}^{\mathrm{e}}$).

Case 2: $\hat{c}^{\mathrm{s}} - c_1^{\mathrm{e}} > 1$ and $\hat{c}^{\mathrm{e}} \leq c_2^{\mathrm{e}}$ (c_1 and $\hat{c}$ are not contiguous) Suppose the 3-tuple created by c_1 is kept, $\hat{c}$ will generate a 3-tuple $\langle(\hat{c}^{\mathrm{e}} - \hat{c}^{\mathrm{s}} + 1) + (c_1^{\mathrm{e}} - c_1^{\mathrm{s}} + 1), \hat{c}^{\mathrm{s}}, \hat{c}^{\mathrm{e}}\rangle$, but this 3-tuple will finally be dominated by $\langle c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, c_2^{\mathrm{e}}\rangle$ created by c_3 (according to the domination rule $(c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1) - ((\hat{c}^{\mathrm{e}} - \hat{c}^{\mathrm{s}} + 1) + (c_1^{\mathrm{e}} - c_1^{\mathrm{s}} + 1)) > c_2^{\mathrm{e}} - \hat{c}^{\mathrm{e}}$).

Case 3: $\hat{c}^{\mathrm{s}} - c_1^{\mathrm{e}} \leq 1$ and $\hat{c}^{\mathrm{e}} > c_2^{\mathrm{e}}$ (c_1 and $\hat{c}$ are contiguous) Suppose the 3-tuple created by c_1 is kept, $\hat{c}$ will generate a 3-tuple $\langle \hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, \hat{c}^{\mathrm{e}}\rangle$ which will dominate $\langle c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, c_2^{\mathrm{e}}\rangle$ created by c_3 (according to the domination rule $(\hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1) - (c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1) = \hat{c}^{\mathrm{e}} - c_2^{\mathrm{e}}$). In this case, if the 3-tuple of c_1 does not initially exist, $\hat{c}$ can still extend the 3-tuple of c_3 to the correct result (since $c_3 < \hat{c}$).

Case 4: $\hat{c}^{\mathrm{s}} - c_1^{\mathrm{e}} > 1$ and $\hat{c}^{\mathrm{s}} - c_2^{\mathrm{e}} \leq 1$ and $\hat{c}^{\mathrm{e}} > c_2^{\mathrm{e}}$ (c_2 and $\hat{c}$ are contiguous)
Similar to Case 3, $\hat{c}$ will generate the 3-tuple $\langle \hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, \hat{c}^{\mathrm{e}}\rangle$ if the 3-tuples of c_1 and c_2 exist. However, even without the presence of these two 3-tuples, we can still extend the 3-tuple of c_3, $\langle c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, c_1^{\mathrm{s}}, c_2^{\mathrm{e}}\rangle$, to arrive at the correct answer.

Case 5: $\hat{c}^{\mathrm{s}} - c_2^{\mathrm{e}} > 1$ and $\hat{c}^{\mathrm{e}} > c_2^{\mathrm{e}}$ ($\hat{c}$ is not contiguous with any other intervals) If the 3-tuples of c_1 and c_2 exist, $\hat{c}$ will generate $\langle(\hat{c}^{\mathrm{e}} - \hat{c}^{\mathrm{s}} + 1) + (c_2^{\mathrm{e}} - c_1^{\mathrm{s}} + 1), \hat{c}^{\mathrm{s}}, \hat{c}^{\mathrm{e}}\rangle$, which is composed of c_1, c_2 and $\hat{c}$. With the existence of the 3-tuple of c_3, $\hat{c}$ can reach the correct answer by extending the 3-tuple of c_3 without relying on c_1 and c_2.

No matter which case occurs, whether the 3-tuples created by c_1 and c_2 exist or not does not affect the development of the LCIIS answer. □

Next, $x_3 = [4,12]$ and $y_3 = [8,17]$ share the maximal common interval $[8,12] = [4,12] \cap [8,17]$ with length 5, which can extend $\langle 4,3,6\rangle$ in the upper-left cell to $\langle 9,8,12\rangle$. After inheriting the entries from the upper and the left cells and performing the domination, only $\langle 7,3,9\rangle$ and $\langle 9,8,12\rangle$ are retained. Then, x_3 and $y_5 = [3,14]$ construct the maximal common interval $[4,12] = [4,12] \cap [3,14]$, which is contiguous with $[3,6]$ in the upper-left cell. Combining these two intervals, we obtain $\langle 10,3,12\rangle$. After inheriting and performing domination, only $\langle 10,3,12\rangle$ is retained.

Following the operations described above, the result of the LCIIS problem will be recorded in the most bottom-right cell of Table V. In this example, the unique entry $\langle 10,3,12\rangle$ in the bottom rightmost cell holds the maximum length $l = 10$ for all 3-tuple entries, indicating that the maximal common intervals found between X and Y can form an increasing interval

sequence with a maximum length of 10. The answer $\langle[3,6],[4,12]\rangle$ can be obtained through the backtracking technique.

According to Tables IV and V, note that the answers to the MCIIS and LCIIS problems with the same input interval sequences may differ.

As mentioned above, we observed that, the starting values of intervals were not used except the extension operation. The ability of extension can be determined by deducing the original starting value from the length l and its ending value c^{e}. Hence, we can streamline the original 3-tuple entry into a more space-efficient 2-tuple entry.

Definition 14 (domination of 2-tuple entries)**.** *Given two 2-tuple entries $\langle l_1, c_1^{\mathrm{e}}\rangle$ and $\langle l_2, c_2^{\mathrm{e}}\rangle$, if $l_1 \geq l_2$ and $l_1 - l_2 \geq c_1^{\mathrm{e}} - c_2^{\mathrm{e}}$, we say that $\langle l_1, c_1^{\mathrm{e}}\rangle$ dominates $\langle l_2, c_2^{\mathrm{e}}\rangle$.*

Theorem 3. *Simplifying from 3-tuple entries to 2-tuple entries does not affect the correctness of the answers.*

Proof. When employing 2-tuple entries, since c^{s} is not recorded, if the found maximal common intervals are not contiguous, we cannot determine the starting value of the last interval in the increasing interval sequence, which may lead to extending an entry that could not be extended. However, due to the existence of domination, these entries that cannot be extended will all be dominated.

Given two maximal common intervals $c_1 = [c_1^{\mathrm{s}}, c_1^{\mathrm{e}}]$ and $c_2 = [c_2^{\mathrm{s}}, c_2^{\mathrm{e}}]$ that are not contiguous, where $c_1 < c_2$, we can construct 2-tuple entries $\langle |c_1|, c_1^{\mathrm{e}}\rangle$ and $\langle |c_1| + |c_2|, c_2^{\mathrm{e}}\rangle$. While $\langle |c_1| + |c_2|, c_2^{\mathrm{e}}\rangle$ cannot yield c_2^{s}, when a maximal common interval $\hat{c} = [\hat{c}^{\mathrm{s}}, \hat{c}^{\mathrm{e}}]$ that can extend these two 2-tuple entries is encountered, two cases are considered as follows:

Case 1: $\hat{c}^{\mathrm{s}} > c_2^{\mathrm{s}}$

The extension of $\hat{c}$ in the 2-tuple entries follows the same approach as in the 3-tuple entries.

Case 2: $(c_2^{\mathrm{e}} - (|c_1| + |c_2|) + 1) < \hat{c}^{\mathrm{s}} \leq c_2^{\mathrm{s}}$

In this case, $\hat{c}$ should not extend c_2, but due to the absence of c_2^{s} in the 2-tuple entry, $\hat{c}$ still extends this 2-tuple entry, resulting in $\langle \hat{c}^{\mathrm{e}} - (c_2^{\mathrm{e}} - (|c_1| + |c_2|) + 1) + 1, \hat{c}^{\mathrm{e}}\rangle$ (due to combination). For the 2-tuple entry resulting from $\hat{c}$ extending c_1, if c_1 and $\hat{c}$ are contiguous, it results in $\langle \hat{c}^{\mathrm{e}} - c_1^{\mathrm{s}} + 1, \hat{c}^{\mathrm{e}}\rangle$; if they are not contiguous, it results in $\langle |c_1| + (\hat{c}^{\mathrm{e}} - \hat{c}^{\mathrm{s}} + 1), \hat{c}^{\mathrm{e}}\rangle$. Regardless of whether c_1 and $\hat{c}$ are contiguous, the 2-tuple entry resulting from $\hat{c}$ extending c_1 will always dominate the 2-tuple entry resulting from $\hat{c}$ extending c_2. Therefore, incorrect extensions will be discarded and they do not affect the final result. □

Table VI shows the example of LCIIS finding by the 2-tuple entry $\langle l, c^{\mathrm{e}}\rangle$ for the same input interval sequences that by the 3-tuple entry $\langle l, c^{\mathrm{s}}, c^{\mathrm{e}}\rangle$ in Table V. The LCIIS algorithms using 2-tuple entries will be introduced in Section IV-A.

A. The Algorithms

Similar to the algorithms for the MCIIS problem, Algorithm 4 presents the main procedure, incorporating the EXTEND$(\cdot)$ function to extend entries from the upper-left cell, and the

TABLE VI: An example of finding LCIIS by using the 2-tuple entry $\langle l, c^{\mathrm{e}}\rangle$, where $X = \langle[1,6],[12,14],[4,12],[10,11]\rangle$ and $Y = \langle[3,11],[6,9],[8,17],[5,7],[3,14]\rangle$. The answer is $\langle[3,6],[4,12]\rangle$, with a total length of 10.

X \ Y		y_1 [3, 11]	y_2 [6, 9]	y_3 [8, 17]	y_4 [5, 7]	y_5 [3, 14]
x_1	[1, 6]	**⟨4, 6⟩**	~~**⟨1, 6⟩**~~ ⟨4, 6⟩	⟨4, 6⟩	~~**⟨2, 6⟩**~~ ⟨4, 6⟩	**⟨4, 6⟩**
x_2	[12, 14]	⟨4, 6⟩	⟨4, 6⟩	⟨4, 6⟩ **⟨7, 14⟩**	⟨4, 6⟩ ⟨7, 14⟩	⟨4, 6⟩ **⟨7, 14⟩**
x_3	[4, 12]	⟨4, 6⟩ **⟨8, 11⟩**	~~⟨4, 6⟩~~ **⟨7, 9⟩** ⟨8, 11⟩	~~⟨4, 6⟩~~ ⟨7, 9⟩ ~~⟨7, 14⟩~~ ~~⟨8, 11⟩~~ **⟨9, 12⟩**	~~⟨4, 6⟩~~ **~~⟨5, 7⟩~~** ⟨7, 9⟩ ~~⟨7, 14⟩~~ ⟨9, 12⟩	~~⟨4, 6⟩~~ ~~⟨7, 9⟩~~ ~~⟨7, 14⟩~~ ~~⟨9, 12⟩~~ **⟨10, 12⟩**
x_4	[10, 11]	~~**⟨2, 11⟩**~~ ⟨4, 6⟩ ⟨8, 11⟩	~~⟨4, 6⟩~~ ⟨7, 9⟩ ⟨8, 11⟩	~~⟨7, 9⟩~~ ~~⟨8, 11⟩~~ **⟨9, 11⟩** ~~⟨9, 12⟩~~	~~⟨7, 9⟩~~ ⟨9, 11⟩ ~~⟨9, 12⟩~~	**~~⟨9, 11⟩~~** ⟨10, 12⟩

DOMINATE$(\cdot)$ function to eliminate dominated entries. Functions EXTEND$(\cdot)$ and DOMINATE$(\cdot)$ are separately outlined in Algorithms 5 and 6. Unlike MCIIS algorithms, LCIIS algorithms utilize the 2-tuple entry $\langle l, c^{\mathrm{e}}\rangle$ to store necessary information.

In Algorithm 4, $D_{i,j}$ records the ordered dominant set within the cell at row i and column j during the DP execution. Line 1 denotes the initialization step. Since the identified maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ may have $c^{\mathrm{s}} = 1$, initializing $D_{0,0}$, $D_{i,0}$, and $D_{0,j}$ as $\langle 0, 0\rangle$ could incorrectly indicate that the interval $[0, 0]$ is contiguous with $[c^{\mathrm{s}}, c^{\mathrm{e}}]$. Therefore, we initialize $D_{0,0}$, $D_{i,0}$, and $D_{0,j}$ as $\langle 0, -1\rangle$ to avoid this issue. This initialization ensures that the maximal common interval can extend from $\langle 0, -1\rangle$ during the execution of EXTEND$(\cdot)$ in Line 8, thereby eliminating the need for special case handling in the first row and first column.

Lines 2 to 8 form the core of the algorithm. First, each cell inherits data from its upper and its left cells. Then, it determines if a maximal common interval exists. If it does, the algorithm identifies the maximum l that can be extended from the 2-tuples in its upper-left cell. The operation is almost identical to Algorithm 1, except that the 2-tuple entries are used here.

Algorithm 5 demonstrates the operation of EXTEND$(\cdot)$. The pointer p points to the first 2-tuple in the set $D_{i-1,j-1}$ and iterates to the end. p.length refers to the value stored in the first attribute of this 2-tuple (the length of this common increasing interval sequence) and p.end refers to the value stored in the second attribute of this 2-tuple (the ending value of a maximal common interval). The variable l records the length achieved after extension in each iteration, while l' keeps track of the maximum l that can be extended across all iterations.

In each iteration, the algorithm first checks if the identified maximal common interval can be extended by the interval pointed to by p (Line 3). If the extension is possible, it further determines whether the two intervals are contiguous. If they

Algorithm 4 The algorithm for solving the LCIIS problem.

Input: Two interval sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$, where $x_i = [x_i^{\mathrm{s}}, x_i^{\mathrm{e}}]$ and $y_j = [y_j^{\mathrm{s}}, y_j^{\mathrm{e}}]$ for $1 \le i \le m$ and $1 \le j \le n$.
Output: The maximum length l in the LCIIS answer.

1: $D_{0,0} \leftarrow D_{i,0} \leftarrow D_{0,j} \leftarrow \{\langle 0, -1 \rangle\}$ for $1 \le i \le m$, $1 \le j \le n$ ▷ initialization
2: **for** $i \leftarrow 1$ to m **do**
3: **for** $j \leftarrow 1$ to n **do**
4: $D_{i,j} \leftarrow \textsc{Dominate}(D_{i-1,j}, D_{i,j-1})$
5: $c^{\mathrm{s}} \leftarrow \max(x_i^{\mathrm{s}}, y_j^{\mathrm{s}})$
6: $c^{\mathrm{e}} \leftarrow \min(x_i^{\mathrm{e}}, y_j^{\mathrm{e}})$
7: **if** $c^{\mathrm{s}} \le c^{\mathrm{e}}$ **then** ▷ maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$ exists
8: $D_{i,j} \leftarrow \textsc{Dominate}(\textsc{Extend}([c^{\mathrm{s}}, c^{\mathrm{e}}], D_{i-1,j-1}), D_{i,j})$
9: **return** $\max\{l | \langle l, \cdot \rangle \in D_{m,n}\}$

are, l records the combined interval length; if not, it calculates the lengths of the two separate intervals and sums them. Due to the existence of $\langle 0, -1 \rangle$ in the set (see Algorithm 4), it guarantees that when a maximal common interval exists, it always produces $\langle c^{\mathrm{e}} - c^{\mathrm{s}} + 1, c^{\mathrm{e}} \rangle$. Finally, l' is updated to the maximum l encountered among the data that have been iterated through.

Algorithm 5 The algorithm of $\textsc{Extend}([c^{\mathrm{s}}, c^{\mathrm{e}}], D_{i-1,j-1})$, which extends increasing interval sequence from the upper-left cell.

Input: A maximal common interval $[c^{\mathrm{s}}, c^{\mathrm{e}}]$, as well as the dominant set $D' = D_{i-1,j-1}$.
Output: A set containing a 2-tuple entry $\langle l, c^{\mathrm{e}} \rangle$.

1: $l \leftarrow 0$, $l' \leftarrow 0$ ▷ initialization
2: **for** $p \leftarrow D'$.begin() to D'.end() **do** ▷ pointer p iterates over 2-tuple entries in D'
3: **if** $c^{\mathrm{s}} > (p.\text{end} - p.\text{length} + 1)$ and $c^{\mathrm{e}} > p.\text{end}$ **then**
4: **if** $c^{\mathrm{s}} - p.\text{end} \le 1$ **then** ▷ two intervals are contiguous
5: $l \leftarrow c^{\mathrm{e}} - (p.\text{end} - p.\text{length} + 1) + 1$
6: **else**
7: $l \leftarrow p.\text{length} + (c^{\mathrm{e}} - c^{\mathrm{s}} + 1)$
8: **if** $l > l'$ **then**
9: $l' \leftarrow l$
10: **return** $\{\langle l', c^{\mathrm{e}} \rangle\}$

Algorithm 6 demonstrates the process of $\textsc{Dominate}(D_1', D_2')$. First, an empty set D'' is used to store 2-tuples, with pointers p_1 and p_2 pointing to the first 2-tuple entries in D_1' and D_2', respectively. D_1' and D_2' are both dominant sets. Each entry of D_1' and D_2' is inserted into D'' in ascending order of l if it is not dominated by the entry pointed by the other pointer.

Lines 2 to 15 illustrate the process when neither set has been completely examined. If the 2-tuple entries pointed to by p_1 and p_2 have the same l, the one with the larger ending value shall be eliminated, and its pointer moves to the next one in its set. If the 2-tuple entries pointed to by p_1 and p_2 have different l, the algorithm checks whether the 2-tuple entry with the smaller l might be dominated by the other entry. If it does not, the 2-tuple entry with the smaller l is appended to D''. Regardless of domination, the pointer to the 2-tuple entry with the smaller l moves to the next one. Lines 16 to 19 illustrate that once either D_1' or D_2' has been completely traversed, the remaining 2-tuple entries from the set that has not yet been fully traversed are sequentially appended to D''.

When creating a new D'', since the 2-tuples in D_1' and D_2' are sorted in ascending order of l, only when there is no domination relationship between the two 2-tuple entries, the one with the smaller l will be put to D''. Consequently, the 2-tuple entries in D'' will also maintain the ascending order of l.

Algorithm 6 The algorithm of $\textsc{Dominate}(D_1', D_2')$, which merges D_1' and D_2' and removes the dominated entries.

Input: Two dominant sets D_1' and D_2' containing 2-tuple entries.
Output: A dominant set D'' containing 2-tuple entries.

1: $D'' \leftarrow \emptyset$, $p_1 \leftarrow D_1'$.begin(), $p_2 \leftarrow D_2'$.begin() ▷ initialization
2: **while** $p_1 \ne D_1'$.end() and $p_2 \ne D_2'$.end() **do**
3: **if** p_1.length $= p_2$.length **then**
4: **if** p_1.end $\le p_2$.end **then** ▷ p_1 dominates p_2
5: $p_2 \leftarrow p_2$.next() ▷ discard p_2
6: **else** ▷ p_2 dominates p_1
7: $p_1 \leftarrow p_1$.next() ▷ discard p_1
8: **else if** p_1.length $< p_2$.length **then**
9: **if** p_2.length $- p_1$.length $< p_2$.end $- p_1$.end **then**
10: D''.append(p_1)
11: $p_1 \leftarrow p_1$.next()
12: **else**
13: **if** p_1.length $- p_2$.length $< p_1$.end $- p_2$.end **then**
14: D''.append(p_2)
15: $p_2 \leftarrow p_2$.next()
16: **while** $p_1 \ne D_1'$.end() **do**
17: D''.append(p_1), $p_1 \leftarrow p_1$.next()
18: **while** $p_2 \ne D_2'$.end() **do**
19: D''.append(p_2), $p_2 \leftarrow p_2$.next()
20: **return** D''

B. Complexity Analysis

Our algorithm solves the LCIIS problem in $\mathrm{O}(n^3)$ time and space, where each cell contains $\mathrm{O}(n)$ entries, given that $m \le n$.

Theorem 4. *Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$. Each cell contains at most $m + n$ 2-tuple entries.*

Proof. The ending value of a maximal common interval is determined by comparing the ending values x_i^{e} and y_j^{e} of x_i

and y_j, respectively, where $1 \leq i \leq m$ and $1 \leq j \leq n$. Thus, there are at most $m + n$ distinct ending values for maximal common intervals. Additionally, for 2-tuple entries with the same ending value, only the one with the longest length is retained (refer to Definition 12). □

The EXTEND(c^s, c^e, D') function scans through all entries within D' and takes $O(n)$ time. Function DOMINATE(D'_1, D'_2) merges two sets D'_1 and D'_2, each holding $O(n)$ entries sorted by length l and the respective pointer moves in one direction until the end of the traversal, which also takes $O(n)$ time. Therefore, the overall time complexity is given as follows.

Theorem 5. *The time complexity of Algorithm 4 is $O(n^3)$, where n is the maximum number of intervals in the two input sequences.*

V. CONCLUSION

In this paper, we introduce two novel problems, the most common increasing interval subsequence (MCIIS) problem and the longest common increasing interval subsequence (LCIIS) problem. These variations are derived from the longest common increasing subsequence (LCIS) problem by replacing values with intervals. We also propose efficient algorithms for solving these two problems.

In this paper, we attempt to constrain the number of entries within a cell in the MCIIS problem to $O(n)$ by utilizing the relationship between starting and ending values. This approach aimed to reduce the overall time complexity to $O(n^3)$, but ultimately, it was unsuccessful. In the future, we may aim to find lower complexity of MCIIS and LCIIS, leveraging the properties of intervals to develop efficient algorithms.

REFERENCES

[1] M. R. Alam and M. S. Rahman, "A divide and conquer approach and a work-optimal parallel algorithm for the LIS problem," *Information Processing Letters*, Vol. 113, No. 13, pp. 470–476, 2013.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, Vol. 215, No. 3, pp. 403–410, 1990.

[3] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, Vol. 60, No. 6, pp. 503–515, 1954.

[4] S. Bespamyatnikh and M. Segal, "Enumerating longest increasing subsequences and patience sorting," *Information Processing Letters*, Vol. 76, No. 1-2, pp. 7–11, 2000.

[5] G.-T. Chen and C.-B. Yang, "Efficient algorithms for the increasing interval subsequence problems," *Journal of Information Science and Engineering*, Vol. 41, No. 3, pp. 525–541, 2025.

[6] X. Chen, S. Kwong, and M. Li, "A compression algorithm for DNA sequences and its applications in genome comparison," *Proceedings of the fourth annual international conference on Computational molecular biology*, Lyon, France, p. 107, 2000.

[7] X. Chen, M. Li, B. Ma, and J. Tromp, "DNACompress: fast and effective DNA sequence compression," *Bioinformatics*, Vol. 18, No. 12, pp. 1696–1698, 2002.

[8] M. Crochemore and E. Porat, "Fast computation of a longest increasing subsequence and application," *Information and Computation*, Vol. 208, No. 9, pp. 1054–1059, 2010.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," *ACM Sigmod Record*, Vol. 23, No. 2, pp. 419–429, 1994.

[10] T.-c. Fu, F.-l. Chung, R. Luk, and C.-m. Ng, "Stock time series pattern matching: Template-based vs. rule-based approaches," *Engineering Applications of Artificial Intelligence*, Vol. 20, No. 3, pp. 347–364, 2007.

[11] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: genetic sequences," *Information processing & management*, Vol. 30, No. 6, pp. 875–886, 1994.

[12] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, Vol. 20, pp. 350–353, 1977.

[13] M. Kutz, G. S. Brodal, K. Kaligosi, and I. Katriel, "Faster algorithms for computing longest common increasing subsequences," *Journal of Discrete Algorithms*, Vol. 9, No. 4, pp. 314–325, 2011.

[14] R. Lee, R. Chand, S. Tseng, and Y. T. Tsai, *Introduction to the Design and Analysis of Algorithms: A Strategic Approach*. McGraw-Hill, 2006.

[15] S.-F. Lo, K.-T. Tseng, C.-B. Yang, and K.-S. Huang, "A diagonal-based algorithm for the longest common increasing subsequence problem," *Theoretical Computer Science*, Vol. 815, pp. 69–78, 2020.

[16] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences*, Vol. 85, No. 8, pp. 2444–2448, 1988.

[17] C. Schensted, "Longest increasing and decreasing subsequences," *Canadian Journal of Mathematics*, Vol. 13, pp. 179–191, 1961.

[18] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, Vol. 10, No. 1, pp. 99–127, 1976.

[19] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. Kaeli, "Subsequence matching on structured time series data," *Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data*, Baltimore, Maryland, USA, pp. 682–693, 2005.

[20] I.-H. Yang, C.-P. Huang, and K.-M. Chao, "A fast algorithm for computing a longest common increasing subsequence," *Information Processing Letters*, Vol. 93, No. 5, pp. 249–253, 2005.