The Diagonal Method for the Merged Longest Common Increasing Subsequence Problem

Chien-Ting Lee, Chang-Biau Yang and Kuo-Si Huang

***Abstract*—In the merged longest common increasing subsequence (MLCIS) problem, we are given a pair of sequences A and B along with a target sequence T. Its goal aims to identify the longest common subsequence that is increasing in both the merged sequence $E(A, B)$ and the target sequence T. Here, $E(A, B)$ represents a new sequence constructed from arbitrarily merging A and B while maintaining their original order. This problem represents a generalized variant by combining the merged longest common subsequence (merged LCS) problem and the longest increasing subsequence (LIS) problem. In this paper, we propose the diagonal algorithm for solving the MLCIS problem, with time complexity O($L(r-L)(m+n)$), where m, n, r, and L denote the lengths of sequences A, B, T, and MLCIS answer, respectively. The diagonal algorithm is very efficient when L is either very small or L is close to r.**

***Index Terms*—longest increasing subsequence, merged longest common subsequence, merged longest common increasing subsequence, dynamic programming, diagonal**

I. Introduction

The concept of *longest common subsequence* (LCS) [5, 17] can be used to measure the similarity of two given sequences, such as bioinformatics, speech recognition and plagiarism detection [10]. Several variants of the LCS problem were proposed, such as *longest common increasing subsequence* (LCIS) [18], and *merged longest common subsequence* (MLCS) [6], and *multiple longest common subsequence* [11].

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

In 1974, Wagner and Fischer [17] proposed a dynamic programming (DP) algorithm for solving the LCS problem, with time complexity O(mn). Then, in 1977, the algorithm of Hunt and Szymanski [7] is based on the dominant matches of A and B, and the time complexity is O($(R+m)\log m$), where R is the total number of match pairs in A and B. Nakatsu *et al.* [12] introduced a diagonal method in O($L(m-L)$) time, where L denotes the LCS length. The algorithm has better efficiency when L is small or close to m,

In 2005, Yang *et al.* [18] first introduced the *longest common increasing subsequence* (LCIS) problem. The LCIS problem attempts to find the common subsequence of A and B, that is increasing and has the maximum length. In 2008, Huang *et al.* [6] first defined the *merged longest common subsequence* (MLCS) problem. The MLCS problem is to find the LCS between the merged sequence $E(A, B)$ and the target sequence T. $E(A, B)$ denotes a sequence obtained from arbitrarily merging two subsequences in A and B with preserving their original orders. The list of related algorithms for solving the MLCS problem [4, 6, 13, 14, 16] and LCIS problem [1–3, 10, 15, 18] can be found in the Master's thesis of Lee [8].

In our previous study [9], we have defined the *merged longest common increasing subsequence* (MLCIS) problem, and then propose a dynamic programming algorithm for solving it. Given two sequences A and B, along with a target sequence T, the MLCIS problem is to identify the common subsequence of $E(A, B)$ and T with the maximal length such that the answer is increasing in both $E(A, B)$ and T. Here, $E(A, B)$ represents an arbitrary sequence obtained by merging A and B while maintaining the original orders. For example, consider $A = \langle \underline{2}, \underline{5}, \underline{4}, \underline{8}\rangle$, $B = \langle 7, 4, 1, 8, 7\rangle$ and $T = \langle 2, 7, 4, 5, 9, 7, 8\rangle$. $E(A, B)$ could be $\langle \underline{2}, \underline{5}, \underline{4}, \underline{8}, 7, 4, 1, 8, 7\rangle$, $\langle \underline{2}, 7, 4, \underline{5}, \underline{4}, 1, 8, 7, \underline{8}\rangle$, or others. The MLCIS answer is $\langle \underline{2}, 4, \underline{5}, 7, \underline{8}\rangle$ with length 5, where $\langle a_1 = 2, b_2 = 4, a_2 = 5, b_5 = 7, a_4 = 8\rangle$.

This paper solves the MLCIS problem in O($L(r-L)(m+n)$) time based on the diagonal approach, where m, n, r, and L denote the lengths of sequences A, B, T, and MLCIS answer, respectively. The experimental results show that in the DP approach, the larger the $|A| \times |B|$ is, the more time is spent. In the diagonal method, the efficiency is better when either the MLCIS length L is very small or L is close to r.

This paper is arranged as follows. Section II gives the preliminaries of the MLCIS problem and the diagonal-based approach. Section III presents the data structures for the proposed algorithm. In Section IV, we compare the performance of the DP and diagonal algorithms. At the end, conclusions and future work are given in Section V.

II. The Diagonal Algorithm

Our diagonal algorithm for the MLCIS problem [9] is inspired by the algorithm proposed by Nakatsu *et al.* [12],

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

Chien-Ting Lee 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.

which was originally designed for solving the LCS problem. The diagonal method is more efficient in solving the LCS problem when the input sequences are highly similar. For solving the MLCIS problem by the diagonal method, some preliminaries are introduced.

Definition 1 (MLCIS [9]). *Given a pair of sequences $A = \langle a_1, a_2, \cdots, a_m\rangle$ and $B = \langle b_1, b_2, \cdots, b_n\rangle$, along with a target sequence $T = \langle t_1, t_2, \cdots, t_r\rangle$, the* merged longest common increasing subsequence (*MLCIS*) *problem is to find the longest common increasing subsequence of $E(A, B)$ and T, where $E(A, B)$ is a sequence obtained by merging subsequences of A and B arbitrarily with preserving their original orders in A and B individually.*

Definition 2 (3-tuple (i, j, v)). *Given two sequences $A = \langle a_1, a_2, \cdots, a_m\rangle$ and $B = \langle b_1, b_2, \cdots, b_n\rangle$, along with a target sequence $T = \langle t_1, t_2, \cdots, t_r\rangle$, we use a 3-tuple (i, j, v) to represent a potential MLCIS solution of $A_{1..i}$ and $B_{1..j}$ ending at value v.*

Definition 3 (domination). *Given two 3-tuples $(i_1, j_1, v_1) \neq (i_2, j_2, v_2)$, if $i_1 \leq i_2$, $j_1 \leq j_2$ and $v_1 \leq v_2$, it is said that (i_1, j_1, v_1) dominates (i_2, j_2, v_2).*

Definition 4 (dominating set). *A set is a dominating set if none of its elements dominates any other element.*

Definition 5 ($(i, j, v) \in S_{k,q}$). *Let $S_{k,q}$ be a dominating set, $k, q \geq 0$, and each 3-tuple $(i, j, v) \in S_{k,q}$ represents $|MLCIS(A_{1..i}, B_{1..j}, T_{1..k})| = q$, where $v = \max\{a_i, b_j\}$ and the MLCIS answer ends at v. In addition, $S_{k,q} = \{(0, 0, 0)\}$ for $q = 0$ as initialization.*

To solve the MLCIS problem, we need two auxiliary functions, EXTEND$(\cdot)$ and DOMINATE$(\cdot)$, where EXTEND$(\cdot)$ performs solution extension for each 3-tuple in a set, and DOMINATE$(\cdot)$ maintains a dominating set by removing dominated 3-tuples.

Definition 6 (EXTEND$(\cdot)$). *Given t_k, we can extend each 3-tuple $(i_1, j_1, v_1) \in S_{k-1,q-1}$ to (i_2, j_1, t_k) if $a_{i_2} = t_k$, $i_1 < i_2$, and $v_1 < t_k$, and to (i_1, j_2, t_k) if $b_{j_2} = t_k$, $j_1 < j_2$ and $v_1 < t_k$. If no such a_{i_2} or b_{j_2} exists, the extension result is set to null. Such feasible (i_2, j_1, t_k) and (i_1, j_2, t_k) are possible elements of $S_{k,q}$. This extension is denoted by* EXTEND$(S_{k-1,q-1}, t_k)$.

Definition 7 (DOMINATE$(\cdot)$). *Given a set S, the domination function, denoted by* DOMINATE(S), *is to remove all dominated 3-tuples in S such that S becomes a dominating set.*

With EXTEND$(\cdot)$ and DOMINATE$(\cdot)$, Theorem 1 is our main operation for generating $S_{k,q}$.

Theorem 1. $S_{k,q}$ = DOMINATE($S_{k-1,q} \cup$ EXTEND($S_{k-1,q-1}$, t_k))

We explain the details by the example shown in Table I, where $A = \langle 2, 5, 4, 8\rangle$, $B = \langle 7, 4, 1, 8, 7\rangle$, and $T = \langle 2, 7, 4, 5, 9, 7, 8\rangle$.

In Round 1, we initialize $S_{0,0} = \{(0, 0, 0)\}$. By Theorem 1, $S_{1,1}$ = DOMINATE($S_{0,1} \cup$ EXTEND$(S_{0,0}, t_1)$). By Definition 5, we have $S_{0,1} = \emptyset$. For EXTEND$(S_{0,0}, t_1)$, we can find $a_1 = t_1 = 2$, and there is no b_j equal to $t_1 = 2$ in B, resulting in $(0, 0, 0)$ being only extended to $(1, 0, 2)$. Thus, $S_{1,1} = \{(1, 0, 2)\}$. Next, $S_{2,2}$ = DOMINATE($S_{1,2} \cup$ EXTEND$(S_{1,1}, t_2)$), where $S_{1,2} = \emptyset$. We find that $b_1 = t_2 = 7$, resulting in $S_{2,2} = \{(1, 1, 7)\}$. Note that, $(1, 1, 7)$ cannot be further extended, thus $S_{3,3} = \emptyset$.

In Round 2, we repeat the above steps to get $S_{2,1} = \{(0, 1, 7), (1, 0, 2)\}$, $S_{3,2} = \{(1, 1, 7), (1, 2, 4), (3, 0, 4)\}$ and $S_{4,3} = \{(2, 2, 5)\}$.

In Round 3, $S_{3,1}$ = DOMINATE($S_{2,1} \cup$EXTEND$(S_{2,0}, t_3)$). We can find $a_3 = b_2 = t_3 = 4$, so $S_{2,0} = (0, 0, 0)$ can be extended to $(3, 0, 4)$ and $(0, 2, 4)$. $S_{2,1} \cup$ EXTEND$(S_{2,0}, 4) = \{(0, 1, 7), (0, 2, 4), (1, 0, 2), (3, 0, 4)\}$. $(3, 0, 4)$ is dominated by $(1, 0, 2)$. Finally, $S_{3,1} = \{(0, 1, 7), (0, 2, 4), (1, 0, 2)\}$. Next, $S_{4,2}$ = DOMINATE($S_{3,2} \cup$ EXTEND$(S_{3,1}, t_4)$). $(0, 2, 4)$ and $(1, 0, 2)$ can be extended to $(2, 2, 5)$. $(2, 2, 5)$ is dominated by $(2, 0, 5)$, so $D_{4,2} = \{(1, 1, 7), (1, 2, 4), (2, 0, 5), (3, 0, 4)$ $\}$. Then we can get $S_{5,3} = \{(2, 2, 5)\}$, $S_{6,4} = \{(2, 5, 7)\}$ and $S_{7,5} = \{(4, 5, 8)\}$ in order. In the end, according to $S_{7,5} = \{(4, 5, 8)\}$, the MLCIS length is 5.

III. THE DATA STURTURES

The 3-tuples in a set $S_{k,q}$ form two doubly linked lists, with each 3-tuple being present in both lists. One list, named the i-list, is sorted increasingly by the i values, and the other, named the j-list, is sorted increasingly by the j values. Each node has four pointers: $next\_i$ and $prev\_i$, used in the i-list, $next\_j$ and $prev\_j$, used in the j-list.

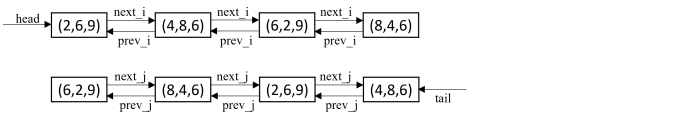
The result obtained from EXTEND$(S_{k-1,q-1}, t_k)$ (extension of $S_{k-1,q-1}$ with t_k) is also an i-list. We employ a linear merge scheme to merge the extended list and $S_{k-1,q}$, and then remove the dominated elements.

We explain the details of the domination operation DOMINATE$(\cdot)$ by the example shown in Figure 1. Suppose that we have EXTEND$(S_{k-1,q-1}) = \{(0, 7, 7), (1, 4, 7), (4, 1, 7), (7, 0, 7)\}$, denoted by the i-list H, and $S_{k-1,q} = \{(2, 6, 9), (4, 8, 6), (6, 2, 9), (8, 4, 6)\}$. The i-list of $S_{k-1,q}$ is denoted by $S' = \{(2, 6, 9), (4, 8, 6), (6, 2, 9), (8, 4, 6)\}$, used for checking the insertion possibility of elements in H. The j-list of $S_{k-1,q}$ is denoted by $S'' = \{(6, 2, 9), (8, 4, 6), (2, 6, 9), (4, 8, 6)\}$, used for domination check if the elements in H are inserted.

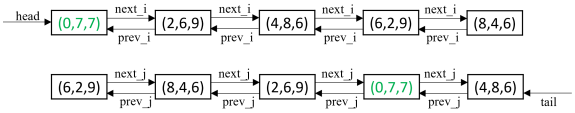
We aim to insert the elements (3-tuples) in H sequentially into $S_{k-1,q}$. We first check whether $(0, 7, 7)$ in H can be inserted to S'. The i-value of $(0, 7, 7)$ is 0. We check only the elements in S' with i-values less than or equal to 0, since only these elements may dominate $(0, 7, 7)$. Consequently, $(0, 7, 7)$ can be inserted into S'. Next, we check if any element in S'' is dominated by $(0, 7, 7)$. We need to check only the elements with j-values in S'' greater than or equal to 7, since only these elements may be dominated by $(0, 7, 7)$. So we only check $(4, 8, 6)$, which is not dominated by $(0, 7, 7)$. Finally, $(0, 7, 7)$ is inserted into both lists S' and S''.

TABLE I: The construction of $S_{k,q}$ in the diagonal MLCIS algorithm with $A = \langle 2, 5, 4, 8\rangle$, $B = \langle 7, 4, 1, 8, 7\rangle$, and $T = \langle 2, 7, 4, 5, 9, 7, 8\rangle$.

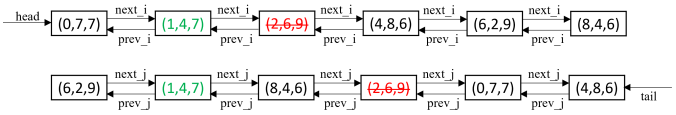
Round f \ q (length)	0	1	2	3	4	5
1	$S_{0,0}$	$S_{1,1}$	$S_{2,2}$	$S_{3,3}$		
	$(0,0,0)$	$(1,0,2)$	$(1,1,7)$			
2	$S_{1,0}$	$S_{2,1}$	$S_{3,2}$	$S_{4,3}$	$S_{5,4}$	
	$(0,0,0)$	$(0,1,7)$	$(1,1,7)$	$(2,2,5)$		
		$(1,0,2)$	$(1,2,4)$			
			$(3,0,4)$			
3	$S_{2,0}$	$S_{3,1}$	$S_{4,2}$	$S_{5,3}$	$S_{6,4}$	$S_{7,5}$
	$(0,0,0)$	$(0,1,7)$	$(1,1,7)$	$(2,2,5)$	$(2,5,7)$	$(4,5,8)$
		$(0,2,4)$	$(1,2,4)$			
		$(1,0,2)$	$(2,0,5)$			
		~~$(3,0,4)$~~	~~$(2,2,5)$~~			
			$(3,0,4)$			



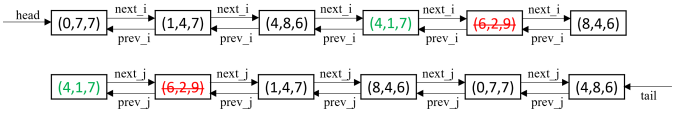
(a) Initialization



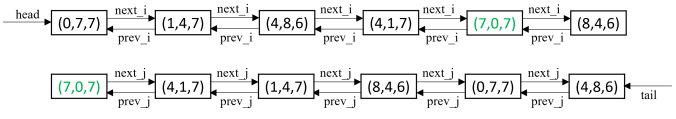
(b) Insertion of $(0, 7, 7)$.



(c) Insertion of $(1, 4, 7)$.



(d) Insertion of $(4, 1, 7)$.



(e) Insertion of $(7, 0, 7)$.

Fig. 1: An example of DOMINATE$(\cdot)$, where EXTEND $(S_{k-1,q-1}) = \{(0,7,7),(1,4,7),(4,1,7),(7,0,7)\}$ and $S_{k-1,q} = \{(2,6,9),(4,8,6),(6,2,9),(8,4,6)\}$. Green: a newly inserted element; red: a deleted element.

The next element in H is $(1, 4, 7)$. We check only the elements in S' with i-values less than or equal to 1. Note that we need not check the elements that have been checked for $(0, 7, 7)$. Because the j-values or v-values of the checked elements is greater than 7, they will be also greater than 4 or greater than 7, respectively. Thus, the checked elements of $(0, 7, 7)$ cannot dominate $(1, 4, 7)$. As a result, $(1, 4, 7)$ can be inserted into S'. Next, we check the elements in S'' dominated by $(1, 4, 7)$. Similarly, we need not check the elements that have been checked for $(0, 7, 7)$. For example, we need not check $(4, 8, 6)$, since $(4, 8, 6)$ is not dominated by $(0, 7, 7)$, resulting in that $(4, 8, 6)$ is not dominated by $(1, 4, 7)$ due to the v-value. In a word, we check the elements in S'' with j-values for $7 > j \geq 4$. In S'', $(2, 6, 9)$ is dominated and $(8, 4, 6)$ is not. Finally, $(1, 4, 7)$ is inserted into both lists S' and S'', while $(2, 6, 9)$ is removed from both lists S' and S''.

For the next element $(4, 1, 7)$, we check the elements in S' with i-values for $4 \geq i > 1$. As a result, $(4, 1, 7)$ can be inserted into S'. Then, we check the elements in S'' with j-values for $4 > j \geq 1$. In S'', $(6, 2, 9)$ is dominated. Finally, $(4, 1, 7)$ is inserted into both lists S' and S'', while $(6, 2, 9)$ is removed from both lists S' and S''.

For the next element $(7, 0, 7)$, we check the elements in S' with i-values for $7 \geq i > 4$. $(7, 0, 7)$ can be inserted into S'. Then, we check the elements in S'' with j-values for $1 > j \geq 0$. In S'', no element is dominated by $(7, 0, 7)$. Finally, $(7, 0, 7)$ is inserted into both lists S' and S''.

In a short summary, the insertion check for a new element of H is done with the i-list increasingly, while the domination check for a new element of H is performed with the j-list decreasingly. Thus, the domination merge of H and $S_{k-1,q}$ is handled with a linear merge scheme. $S_{k,q}$ is set to the merged result. It is clear that the linear merge scheme can be done in linear time.

The detailed analysis was presented in the Master's thesis of

Lee [8]. In summary, Algorithm **??** solves the MLCIS problem in $O(L(r-L)(m+n))$ time and $O((|\Sigma|+L)(m+n))$ space.

IV. Experimental Results

In this section, we compare the execution time of the dynamic programming approach and diagonal method. These two algorithms are implemented using Visual Studio 2022. We conduct experiments on a computer equipped with a 64-bit Windows 10 operating system, a CPU clocked at 2.9GHz (Intel Core i7-10700), and 16GB of memory.

With slight modification, these two algorithms can also solve the *merged longest common weakly increasing subsequence* (MLCWIS) problem, where the answer is weakly increasing (non-decreasing), rather than strictly increasing. The problem is defined in Definition 8. For solving the MLCWIS problem, we can slightly modify the MLCIS algorithm, by replacing $v < t_k$ with $v \leq t_k$ in Function **??**.

Definition 8 (MLCWIS)**.** *Given a pair of sequences $A = \langle a_1, a_2, \cdots, a_m\rangle$ and $B = \langle b_1, b_2, \cdots, b_n\rangle$, along with a target sequence $T = \langle t_1, t_2, \cdots, t_r\rangle$, the* merged longest common weakly increasing subsequence (*MLCWIS*) *problem is to find the longest common non-decreasing (weakly increasing) subsequence of $E(A, B)$ and T, where $E(A, B)$ is the merged sequence by merging subsequences of A and B arbitrarily with preserving their original orders in A and B individually.*

In the experiments, we test the relationship between execution time and the lengths of MLCIS and MLCWIS with pseudo-random datasets. The pseudo-random datasets are generated with combinations of various parameters $(|A|, |B|, |T|, |\Sigma|, L)$, where $|A| = \lambda \times 1000$, $|B| = (1 - \lambda) \times 1000$, $|T| = 1000$, $|\Sigma| \in \{4, 64, 256, 1000\}$, L is the answer length and $\lambda \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The maximum length of the answer in the MLCIS problem is $|\Sigma|$, and the maximum length of the answer in the MLCWIS problem is $|T|$.

In order to get more accurate execution time, we perform 100 experiments to calculate the average. We use a 5-tuple $(|A|, |B|, |T|, |\Sigma|, L)$ to represent different parameters settings in the figures. For example, in $(*, *, 1000, 4, *)$, the first "$*$", second "$*$", and third "$*$" represent all possible lengths of A, B, and L, respectively; $|T| = 1000$, $|\Sigma| = 4$.

For the MLCIS and MLCWIS problems, Figures 2 and 3 show the average execution time by using the DP algorithm [9] and the diagonal algorithm, respectively. As we can see in Figure 2, the larger $|A| \times |B|$ is, the more time is spent. In Figures 3b and 3c, the maximum execution time occurs at $L = 500$, because each dominating set has almost the maximum number of elements at length $L = 500$. This is also consistent with the time complexity $O(L(r - L)(m + n))$, whose maximum value occurs at about $L = \frac{r}{2} = 500$. In Figure 4, we compare the DP algorithm and the diagonal algorithm for different answer lengths. The execution time of the diagonal algorithm is much less than that of the DP algorithm.

V. Conclusion

In this paper, we propose the diagonal algorithm to solve the MLCIS problem in $O(L(r - L)(m + n))$ time, where m, n, r, and L denote the lengths of sequences A, B, T, and MLCIS, respectively. Experimental results demonstrate that the diagonal algorithm outperforms the dynamic programming algorithm. The diagonal algorithm exhibits high efficiency when L is either very small or close to r.

In the future, we may try to find other properties of the problem to reduce the number of elements in $S_{k,q}$ and then reduce the time complexity. Furthermore, we may study other related problems. For example, given a sequence T, the *split longest common subsequence* (SLCS) problem aims to maximize $LCS(A, B)$ by splitting T into two sequences A and B in order. Similarly, given a sequence T, the *split longest common increasing subsequence* (SLCIS) problem aims to split T into two sequences A and B in order such that $LCIS(A, B)$ is maximized. These are all interesting questions.

References

[1] G. S. Brodal, K. Kaligosi, I. Katriel, and M. Kutz, "Faster algorithms for computing longest common increasing subsequences," *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, Barcelona, Spain, pp. 330–341, 2006.

[2] D. Cai, D. Zhu, L. Wang, and X. Wang, "A simple linear space algorithm for computing a longest common increasing subsequence," *IAENG International Journal of Computer Science*, Vol. 45, No. 3, pp. 472–477, 2018.

[3] W.-T. Chan, Y. Zhang, S. P. Y. Fung, D. Ye, and H. Zhu, "Efficient algorithms for finding a longest common increasing subsequence," *Journal of Combinatorial Optimization*, Vol. 13, No. 3, pp. 277–288, 2007.

[4] S. Deorowicz and A. Danek, "Bit-parallel algorithms for the merged longest common subsequence problem," *International Journal of Foundations of Computer Science*, Vol. 24, pp. 1281–1298, 2013.

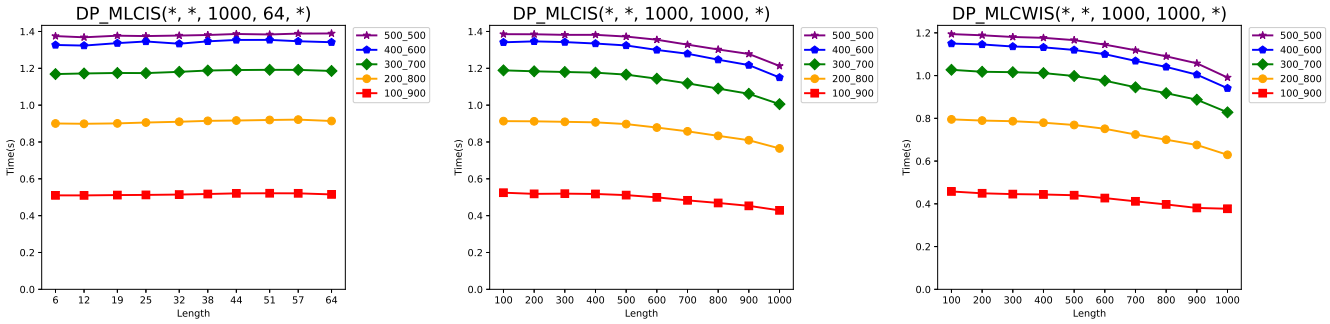
[5] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, Vol. 18, No. 6, pp. 341–343, 1975.

[6] K. S. Huang, C. B. Yang, K. T. Tseng, H. Y. Ann, and Y. H. Peng, "Efficient algorithms for finding interleaving relationship between sequences," *Information Processing Letters*, Vol. 105, pp. 188–193, 2008.

[7] 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.

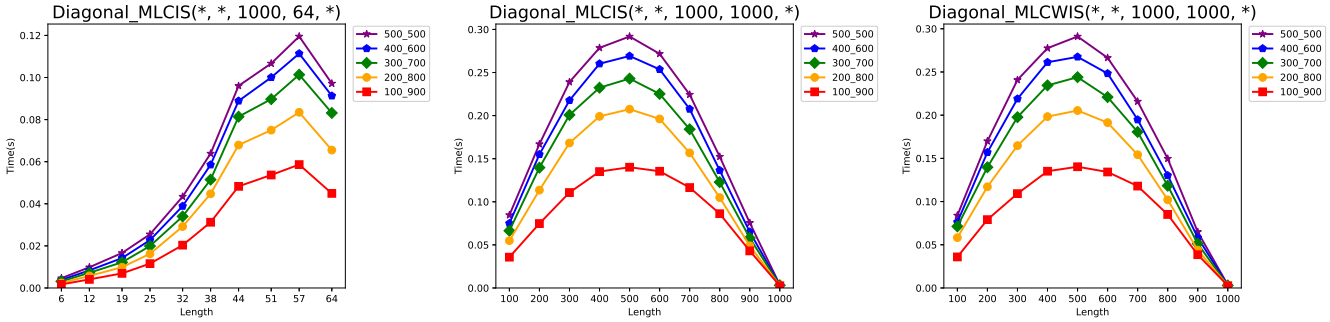
[8] C.-T. Lee, "The merged longest common increasing subsequence problem," *Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-sen University*, Kaohsiung, Taiwan, 2023.

[9] C.-T. Lee, C.-B. Yang, and K.-S. Huang, "The merged longest common increasing subsequence problem," *Proceedings of the 16th Asian Conference on Intelligent*



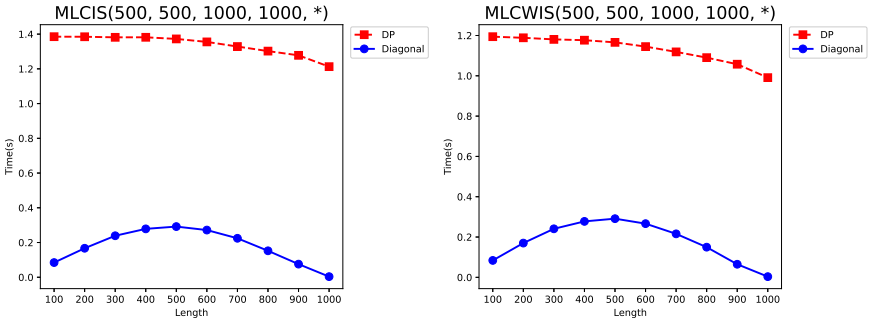
(a) MLCIS, $|\Sigma| = 64$. (b) MLCIS, $|\Sigma| = 1000$. (c) MLCWIS, $|\Sigma| = 1000$.

Fig. 2: The average execution time of the DP algorithm [9] for different answer lengths, where $|T| = 1000$.



(a) MLCIS, $|\Sigma| = 64$. (b) MLCIS, $|\Sigma| = 1000$. (c) MLCWIS, $|\Sigma| = 1000$.

Fig. 3: The average execution time of the diagonal algorithm for different answer lengths, where $|T| = 1000$.



(a) MLCIS (b) MLCWIS

Fig. 4: The average execution time of the DP algorithm [9] and diagonal algorithm on the MLCIS and MLCWIS problems for different answer lengths, where $|A| = 500$, $|B| = 500$, $|T| = 1000$ and $|\Sigma| = 1000$.

Information and Database Systems, Ras Al Khaimah, UAE, April, 2024.

[10] 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.

[11] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM (JACM)*, Vol. 25, No. 2, pp. 322–336, 1978.

[12] N. Nakatsu, Y. Kambayashi, and S. Yajima, "A longest common subsequence algorithm suitable for similar text strings," *Acta Informatica*, Vol. 18, pp. 171–179, 1982.

[13] Y. H. Peng, C. B. Yang, K. S. Huang, C. T. Tseng, and C. Y. Hor, "Efficient sparse dynamic programming for the merged LCS problem with block constraints," Vol. 6, pp. 1935–1947, 2010.

[14] A. M. Rahman and M. S. Rahman, "Effective sparse dynamic programming algorithms for merged and block merged LCS problems," *Journal of Computers*, Vol. 9, No. 8, pp. 1743–1754, 2014.

[15] Y. Sakai, "A linear space algorithm for computing a longest common increasing subsequence," *Information*

Processing Letters, Vol. 99, No. 5, pp. 203–207, 2006.

[16] K.-T. Tseng, D.-S. Chan, C.-B. Yang, and S.-F. Lo, "Efficient merged longest common subsequence algorithms for similar sequences," *Theoretical Computer Science*, Vol. 708, pp. 75–90, 2018.

[17] R. Wagner and M. Fischer, "The string-to-string correction problem," *Journal of the ACM*, Vol. 21, No. 1, pp. 168–173, 1974.

[18] 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.