The Merged Longest Common Subsequence Problems with t-length Substrings and at Least t-length Substrings*

Tu-Cheng Wu[a], Chang-Biau Yang[a†] and Kuo-Si Huang[b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[b]Department of Business Computing
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

This paper proposes two variants of the merged longest common subsequence (MLCS) problem, the MLCS problem with t-length substrings ($MLCS_t$) and the MLCS problem with at least t-length substrings ($MLCS_{t+}$). The dynamic programming approaches can solve both of them in $O(mnr)$ time, where m, n and r are the lengths of the two main sequences A and B and the target sequence P, respectively. In addition, the diagonal concept can be used to solve the $MLCS_t$ and $MLCS_{t+}$ problems in $O(R+(r-L+1)Lm)$ and $O(R+(r-L+1)(Lm+R))$ time respectively, where R is the number of total match pairs between A and P, B and P; and L is the length of the answer.

Keywords: longest common subsequence, merged longest common subsequence, dynamic programming, diagonal method, t-length substring.

1 Introduction

The longest common subsequence (LCS) problem [1] can be used for measuring sequence similarity and it has been extensively studied for several decades since 1970. It has wide applications, such as the comparisons of DNA sequences and protein sequences. With the rapid development of bioinformatics in recent years, many variants of the LCS problem have been proposed. Given two sequences $A = a_1a_2 \cdots a_m$ and $B = b_1b_2 \cdots b_n$, the LCS problem is to find the common subsequence of both A and B with the maximal length. A subsequence can be obtained by deleting zero or more characters at arbitrary positions of a given sequence.

The well-known dynamic programming (DP) algorithm with $O(mn)$ time was proposed by Wagner and Fischer [1] in 1974. After that, lots of improvements and variants of the LCS problem were also proposed, such as the *merged longest common subsequence* (MLCS) problem [2–7], the *longest common subsequence problem with t-length substrings* (LCS_t) [8–12], and the *longest common subsequence problem with at least t-length substrings* (LCS_{t+}) [10–14].

This paper integrates the properties in the MLCS, LCS_t, and LCS_{t+} problems to propose two new variants of the MLCS problem, called the *merged longest common subsequence problem with t-length substrings* ($MLCS_t$) and the *merged longest common subsequence problem with at least t-length substrings* ($MLCS_{t+}$). We propose the DP algorithms for solving the two problems in $O(mnr)$ time. In addition, the diagonal $MLCS_t$ and $MLCS_{t+}$ algorithms are developed respectively with $O(R+(r-L+1)Lm)$ time, and with $O(R+(r-L+1)(Lm+R))$ time. Finally, some experimental results are provided to illustrate the efficiencies of our algorithms.

2 The $MLCS_t$ and $MLCS_{t+}$ Problems and Their DP Algorithms

Let A denote a sequence of characters $a_1a_2 \cdots a_m$. Then, a_i denotes the ith character of A, and $A_{i..j}$ denotes the substring of A from positions i to j, i.e. $A_{i..j} = a_ia_{i+1} \cdots a_j$. In addition, the length or size of A is represented as $|A|$. Given two sequences A, B and a target se-

*This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 109-2221-E-110-040-MY2.

†Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

quence P, the merged LCS problem with t-length substrings, denoted as $MLCS_t(A, B, P)$, is to find the LCS of P and $E(A, B)$, where $E(A, B)$ denotes the merging operation for merging t-length substrings of A and B. In addition, the merged LCS problem with at least t-length substrings, denoted as $MLCS_{t+}(A, B, P)$, is to find the LCS of P and $E(A, B)$, where $E(A, B)$ denotes the merging operation for merging substrings of length t or more in A and B. Note that the characters in the solution of both $MLCS_t(A, B, P)$ and $MLCS_{t+}(A, B, P)$ retain the original order in A and B individually.

The dynamic programming approach can be applied for solving the MLCS$_t$ and the MLCS$_{t+}$ problems. Let $L_t(i, j, k)$ and $L_{t+}(i, j, k)$ denote the lengths of $MLCS_t(A_{1..i}, B_{1..j}, P_{1..k})$ and $MLCS_{t+}(A_{1..i}, B_{1..j}, P_{1..k})$, respectively. The DP formulas for solving the MLCS$_t$ and MLCS$_{t+}$ problems are given in Equations 1 and 2.

$$L_t(i,j,k) = \max \begin{cases} L_t(i-t, j, k-t) + t, & \text{if } A_{i-t+1..i} = P_{k-t+1..k}, \\ L_t(i, j-t, k-t) + t, & \text{if } B_{j-t+1..j} = P_{k-t+1..k}, \\ L_t(i-1, j, k) & \\ L_t(i, j-1, k) & \\ L_t(i, j, k-1) & \\ 0 & \text{if } i \le t-1, j \le t-1 \text{ or } k \le t-1. \end{cases} \tag{1}$$

$$L_{t+}(i,j,k) = \max \begin{cases} L_{t+}(i-t, j, k-t) + t, & \text{if } A_{i-t+1..i} = P_{k-t+1..k}, \\ L_{t+}(i-t-1, j, k-t-1) + t + 1, & \text{if } A_{i-t..i} = P_{k-t..k}, \\ \vdots & \vdots \\ L_{t+}(i-t', j, k-t') + t', & \text{if } A_{i-t'+1..i} = P_{k-t'+1..k} \text{ and } t \le t' \le i, \\ L_{t+}(i, j-t, k-t) + t, & \text{if } B_{j-t+1..j} = P_{k-t+1..k}, \\ L_{t+}(i, j-t-1, k-t-1) + t + 1, & \text{if } B_{j-t..j} = P_{k-t..k}, \\ \vdots & \vdots \\ L_{t+}(i, j-t'', k-t'') + t'', & \text{if } B_{j-t''+1..j} = P_{k-t''+1..k} \text{ and } t \le t'' \le j, \\ L_{t+}(i-1, j, k) & \\ L_{t+}(i, j-1, k) & \\ L_{t+}(i, j, k-1) & \\ 0 & \text{if } i \le t-1,\ j \le t-1 \text{ or } k \le t-1. \end{cases} \tag{2}$$

The preprocessing concept in the LCS$_{t+}$ algorithm of Ueki *et al.* [14] can be applied to the improvement of the DP algorithm for the MLCS$_{t+}$ problem. Based on the auxiliary tables for storing the lengths of matches between A and P, and B and P, respectively, Equation 2 can be revised and improved. Accordingly, the time complexity can be reduced. Hence, we can solve the MLCS$_t$ and MLCS$_{t+}$ problems with the DP approach in O(mnr) time, where m, n, and r are the lengths of A, B, and P, respectively.

Table 1: An example for building $D_{k,s}$ in our diagonal $MLCS_t$ algorithm with $t = 2$ for $A =$ `agcag`, $B =$ `gcgga`, and $P =$ `agcggcag`.

Round i \ Length s	**0**	**2**	**4**	**6**
1	$D_{0,0}$	$D_{2,2}$	$D_{4,4}$	$D_{6,6}$
	$\langle 0,0\rangle$	$\langle 2,0\rangle$	$\langle 2,3\rangle$	
2	$D_{1,0}$	$D_{3,2}$	$D_{5,4}$	$D_{7,6}$
	$\langle 0,0\rangle$	$\langle 0,2\rangle$	$\langle 0,4\rangle$	$\langle 4,3\rangle$
		$\langle 2,0\rangle$	$\langle 2,3\rangle$	~~$\langle 4,4\rangle$~~
		~~$\langle 3,0\rangle$~~	~~$\langle 2,4\rangle$~~	

3 The Diagonal Algorithms

3.1 The Diagonal MLCS$_t$ Algorithm

The diagonal concept for the LCS problem was proposed by Nakatsu*et al.* in 1982 [15]. It is especially efficient with highly similar sequences. In 2018, Tseng *et al.* [7] applied the diagonal concept to solving the MLCS problem. Here, we have to treat the match pairs of characters in the MLCS algorithm as the match pairs of t-length substrings in the MLCS$_t$ algorithm. With appropriate modifications, we give the definition of the domination and the minimum dominating set $D_{k,s}$ for solving the MLCS$_t$ problem.

Definition 1 (Domination [7])**.** *For any two 2-tuples $\langle i_1, j_1\rangle$ and $\langle i_2, j_2\rangle$, $\langle i_1, j_1\rangle \neq \langle i_2, j_2\rangle$, we say that $\langle i_1, j_1\rangle$ dominates $\langle i_2, j_2\rangle$ if $i_1 \le i_2$ and $j_1 \le j_2$.*

Let $MLCS_t(A_{1..i}, B_{1..j}, P_{1..k})$ represent the length of the MLCS$_t$ solution for $A_{1..i}$, $B_{1..j}$ and $P_{1..k}$). The following definition of the dominating set is modified from that of Tseng *et al.* [7].

Definition 2 ($D_{k,s}$ for MLCS$_t$)**.** *$D_{k,s}$, for $k, s \ge 0$ is a dominating set such that any two 2-tuple elements in $D_{k,s}$ do not dominate each other. For each $\langle i, j\rangle \in D_{k,s}$, the $MLCS_t$ length is $MLCS_t(A_{1..i}, B_{1..j}, P_{1..k}) = s$.*

We present an example to explain the building process of $D_{k,s}$ with $t = 2$ for $A =$ `agcag`, $B =$ `gcgga`, and $P =$ `agcggcag`, as shown in Table 1. The MLCS$_t$ answer of this example is $P_{1..2}P_{3..4}P_{6..7} = A_{1..2}B_{2..3}A_{3..4} =$ `agcgca`, with length 6.

For solving the MLCS$_t$ problem efficiently, we build the $Match_A$ and $Match_B$ sets for the extension of $D_{k,s}$.

Definition 3 ($Match_A[\cdot]$ and $Match_B[\cdot]$)**.** *The ordered sets of $Match_A[k]$ and $Match_B[k]$ collect all ending indexes of A and B with the increasing order that match $P_{k-t+1..k}$ with t-length strings. In other words, $Match_A[k] = \{i \mid P_{k-t+1..k} = A_{i-t+1..i}, t \le i \le |A|\}$ and $Match_B[k] = \{j \mid P_{k-t+1..k} = B_{j-t+1..j}, t \le j \le |B|\}$.*

In [7], the $D_{k,s}$ is progressively built from the extension and domination processes. After clarifying the relationship between the match pairs of characters and the match pairs of t-length substrings, we get Fact 1 by Definitions 1, and 2 for the MLCS$_t$ problem. Here, we briefly introduce our diagonal MLCS$_t$ algorithm by Lemma 1 and Theorem 1. Hence the MLCS$_t$ problem can be solved by Theorem 1 in $O(R+(r-L+1)Lm)$ time, where R is the total number of the t-matches of P and A and of P and B.

Fact 1. *$MLCS_t(\cdot)$ is a monotonically increasing function. It means that $MLCS_t(A_{1..i}, B_{1..j}, P_{1..k}) \le MLCS_t(A_{1..x}, B_{1..y}, P_{1..z})$ if $i \le x$, $j \le y$ and $k \le z$.*

Lemma 1. *For a 2-tuple $\langle i, j\rangle \in D_{k,s}$, there exists $\langle x, y\rangle \in D_{k-t,s-t}$ that $\langle x, y\rangle$ dominates $\langle i, j\rangle$, where $k, s \ge t$.*

Theorem 1. *$D_{k,s} = \textsc{Dominate}(D_{k-1,s} \cup \textsc{Extend}(D_{k-t,s-t}))$, where $t \le k, s \le r$.*

3.2 The Diagonal MLCS$_{t+}$ Algorithm

For the diagonal algorithm of the MLCS$_t$ problem, we can regard each t-length substring as a condensed character. However we cannot use this concept directly to solve the MLCS$_{t+}$ problem. In the MLCS$_{t+}$ problem, the 2-tuple elements of dominating set $D_{k,s}$ may be extended from every $D_{k-g,s-g}$ for $t \le g \le 2t-1$. For example, assume that $A =$ `acctggatc`, $B =$ `gacggacc`, $P =$ `accaggtg` and $t = 2$. The $D_{k,s}$ table is shown in Table 2. There is an element in $D_{8,7} = \{\langle 5, 5\rangle\}$, which means that $MLCS_{t+}(A_{1..5}, B_{1..5}, P_{1..8}) = 7$. $\langle 5, 5\rangle$ is extended from $\langle 3, 5\rangle \in D_{6,5}$ with $P_{7..8} = A_{4..5}$ of 2-length, $\langle 3, 5\rangle \in D_{6,5}$ is extended from $\langle 3, 0\rangle \in D_{4,3}$ with $P_{5..6} = B_{4..5}$ of 2-length, and $\langle 3, 0\rangle \in D_{4,3}$ is extended from $\langle 0, 0\rangle \in D_{0,0}$ with $P_{1..3} = A_{1..3}$ of 3-length. We get the answer $P_{1..3}P_{5..6}P_{7..8} = A_{1..3}B_{4..5}A_{4..5} =$ `accggtg`. The solution may be extended with common substring of length $g = 2$ or $g = 3$ for $t = 2$.

Let $MLCS_{t+}(A_{1..i}, B_{1..j}, P_{1..k})$ represent the answer length of the MLCS$_{t+}$ problem for $A_{1..i}$, $B_{1..j}$ and $P_{1..k}$. For efficiently handling the extensions from multiple sources, we define the match pair sets, $MA_{k,s}$ and $MB_{k,s}$. And the $MA_{k,s}$ and $MB_{k,s}$ can induce Fact 2.

Definition 4 ($MA_{k,s}$ and $MB_{k,s}$ for MLCS$_{t+}$)**.** *For $k, s \ge 0$, $MA_{k,s} = \{\langle i, j\rangle \mid i \in Match_A[k]$ and $j = \min\{j', j''\}$, where $\langle i', j'\rangle \in D_{k-t,s-t}$ with the largest $i' \le i - t$, and $\langle i-1, j''\rangle \in MA_{k-1,s-1}\}$. Note that the $MB_{k,s}$ can be similarly defined for B and $Match_B[k]$.*

Fact 2. *$MLCS_{t+}(A_{1..i}, B_{1..j}, P_{1..k}) \ge s$ for each $\langle i, j\rangle \in MA_{k,s}$ or $\langle i, j\rangle \in MB_{k,s}$, where $k, s \ge 0$.*

For solving the MLCS$_{t+}$ problem, we can get the $D_{k,s}$ by applying domination on $MA_{k,s} \cup MB_{k,s} \cup D_{k-1,s}$. Table 2 shows the examples of $D_{k,s}$, $MA_{k,s}$ and $MB_{k,s}$ for the MLCS$_{t+}$ problem with $t = 2$, $A =$ `acctggatc`, $B =$ `gacggacc` and $P =$ `accaggtg`. The $MLCS_{t+}$ answer of this example is $P_{1..3}P_{5..6}P_{7..8} = A_{1..3}B_{4..5}A_{4..5} =$ `accggtg`, with length 7.

By Definitions 1, and 2, we can obtain Fact 3 for the MLCS$_{t+}$ problem. Then, we briefly introduce our diagonal MLCS$_{t+}$ algorithm by Lemma 2 and Theorem 2. Hence the MLCS$_t$+ length can be found by Theorem 2 in $O(R+(r-L+1)(Lm+R))$ time, where R is the number of the total matches of P and A and of P and B.

Fact 3. *$MLCS_{t+}(\cdot)$ is a monotonically increasing function. It means $MLCS_{t+}(A_{1..i}, B_{1..j}, P_{1..k}) \le MLCS_{t+}(A_{1..x}, B_{1..y}, P_{1..z})$ if $i \le x$, $j \le y$ and $k \le z$.*

Lemma 2. *For a 2-tuple $\langle i, j\rangle \in D_{k,s}$, there exists $\langle x, y\rangle \in D_{k-t,s-t}$ or $\langle x', y'\rangle \in D_{k-1,s-1}$ that $\langle x, y\rangle$ or $\langle x', y'\rangle$ dominates $\langle i, j\rangle$, where $k, s \ge t$.*

Theorem 2. *$D_{k,s} = \textsc{Dominate}(D_{k-1,s} \cup MA_{k,s} \cup MB_{k,s})$, where $t \le k, s \le r$.*

4 Experimental Results

We perform our algorithms on pseudorandom sequences of various lengths. These algorithms are implemented by Code::Blocks 16.01 C++ software, and they are tested on several computers with 64-bit Windows 10 OS, CPU clock rate of 3.00GHZ (Intel(R) Core(TM) i5-9500 CPU) and 8 GB RAM. Each experiment is performed 100

Table 2: An example of our $MLCS_{t+}$ algorithm with $t = 2$ for $A =$ `acctggatc`, $B =$ `gacggacc` and $P =$ `accaggtg`. (a) $D_{k,s}$, (b) $MA_{k,s}$, (c) $MB_{k,s}$.

(a) $D_{k,s}$

Round i \ **Length** s	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	$D_{0,0}$	$D_{1,1}$	$D_{2,2}$	$D_{3,3}$	$D_{4,4}$	$D_{5,5}$		
	$\langle 0,0\rangle$		$\langle 0,3\rangle$ $\langle 2,0\rangle$	$\langle 0,8\rangle$ $\langle 3,0\rangle$				
2	$D_{1,0}$	$D_{2,1}$	$D_{3,2}$	$D_{4,3}$	$D_{5,4}$	$D_{6,5}$	$D_{7,6}$	$D_{8,7}$
	$\langle 0,0\rangle$		$\langle 0,3\rangle$ ~~$\langle 0,8\rangle$~~ $\langle 2,0\rangle$ ~~$\langle 3,0\rangle$~~	$\langle 0,8\rangle$ $\langle 3,0\rangle$		$\langle 3,5\rangle$ $\langle 6,0\rangle$		$\langle 5,5\rangle$

(b) $MA_{k,s}$

Round i \ **Length** s	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	$MA_{0,0}$	$MA_{1,1}$	$MA_{2,2}$	$MA_{3,3}$	$MA_{4,4}$	$MA_{5,5}$		
	$\langle 0,0\rangle$		$\langle 2,0\rangle$	$\langle 3,0\rangle$				
2	$MA_{1,0}$	$MA_{2,1}$	$MA_{3,2}$	$MA_{4,3}$	$MA_{5,4}$	$MA_{6,5}$	$MA_{7,6}$	$MA_{8,7}$
	$\langle 0,0\rangle$		$\langle 3,0\rangle$			$\langle 6,0\rangle$ ~~$\langle 6,8\rangle$~~		$\langle 5,5\rangle$

(c) $MB_{k,s}$

Round i \ **Length** s	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	$MB_{0,0}$	$MB_{1,1}$	$MB_{2,2}$	$MB_{3,3}$	$MB_{4,4}$	$MB_{5,5}$		
	$\langle 0,0\rangle$		$\langle 0,3\rangle$ $\langle 0,7\rangle$	$\langle 0,8\rangle$				
2	$MB_{1,0}$	$MB_{2,1}$	$MB_{3,2}$	$MB_{4,3}$	$MB_{5,4}$	$MB_{6,5}$	$MB_{7,6}$	$MB_{8,7}$
	$\langle 0,0\rangle$		$\langle 0,8\rangle$			$\langle 3,5\rangle$		

times to get the average time. For the DP algorithms with string length 5000, because the execution time is too enormous, we perform only 5 times to get the average time.

The similarity of input sequences for the MLCS_t and MLCS_{t+} problem is defined as $\rho = \frac{\text{solution length}}{\min(|A|+|B|,|P|)}$. For example, assume $|A| = |B| = 1000$, $|P| = 2000$ and $t = 3$. The similarity $\rho = 90\%$ means the MLCS_t length is 1800. The pseudorandom datasets are generated with $|P| \in \{1000, 2000, 5000\}$, ratios $\gamma = \frac{|A|}{|P|} = \frac{m}{r} = \{0.1, 0.2, 0.5\}$, alphabet size $\sigma = |\Sigma| \in \{4,\ 20,\ 64,\ 1000\}$, $t \in \{2,\ 3,\ 5\}$, and $\rho \in \{10\%, 20\%, \cdots, 100\%\}$, where $|A| = m$, $|B| = n$, $|P| = r$ and $m + n = r$.

Suppose that ρ is the target similarity of the datasets with MLCS_t (or MLCS_{t+}), and the testing data are randomly generated as follows.

1: Randomly generate the target sequence P and then randomly place each substring of length t (or at least length t) without overlapping into A or B in order.

2: Compute the MLCS_t (or MLCS_{t+}) length of A, B and P, and calculate their similarity λ.

3: If $\lambda \in [\rho-\epsilon, \rho+\epsilon]$, where $\epsilon = 2\%$, then output A, B and P. Otherwise, randomly replace some characters in P and go back to Step 2.

In our experiments, we compare the performances of our MLCS_t algorithms, DP and Diagonal. The average execution time charts of MLCS_t algorithms are denoted by the 5-tuple $(|P|, \gamma, \sigma, t, algorithm)$ for representing various parameters. For example, in Figure 1(a), $(1000, 0.5, 4, 2, *)$ means $|P| = 1000$, $\gamma = 0.5$, alphabet size $\sigma = 4$, $t = 2$ and the wildcard character $*$ indicates all algorithms. The results of $(5000, 0.5, 4, 2, *)$ are shown in Figure 1(b). When the length $|P| = 1000$, the diagonal algorithm is more efficient than the DP algorithm. When the length $|P| = 5000$, the gap becomes wider.

As shown in Figure 2, for the diagonal algorithm with the same lengths of P, A and B, the execution time is decreased when t is large or the alphabet size σ is large, because the number of match pairs is reduced.

The average execution time charts of MLCS_{t+} algorithms are represented by the same 5-tuple $(|P|, \gamma, \sigma, t, algorithm)$. In Figure 3, we can find that the diagonal algorithm is still more efficient than the DP algorithm with the same parameters. In addition, the diagonal algorithm still takes less time when t is large or when the alphabet size σ is large, due to fewer match pairs, as shown in Figure 4.

From the experimental results, we can see that the average execution time of MLCS_{t+} algorithms is more than MLCS_t algorithms in both DP and diagonal algorithms. The reason is that more match pairs need to be calculated in the MLCS_{t+} problem. We can also find that the diagonal algorithm has better time efficiency than the DP algorithm for various parameter values. In the diagonal algorithm, we can see that the execution time is decreased when t or σ is large.

The diagonal algorithm has better time efficiency when L is small or large, because L or $(r-L+1)$ in the time complexity will tend to a constant. Therefore, the experimental result shows a wave situation of rising and then falling.

5 Conclusion

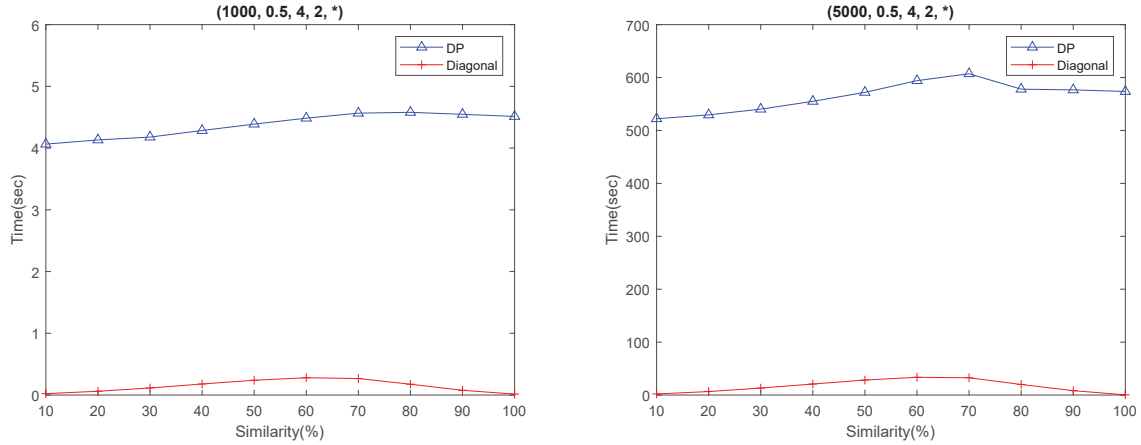
This paper defines two new variants of the MLCS problem, including MLCS_t and MLCS_{t+}. Beside the $O(mnr)$-time dynamic programming algorithms, we propose the diagonal algorithms to solve the MLCS_t and MLCS_{t+} problems in $O(R + (r - L + 1)Lm)$ and $O(R + (r - L + 1)(Lm + R))$ time, respectively, where R is the number of total match pairs between A and P, B and P; and L is the length of the answer. We also perform pseudorandom experiments to compare the efficiency between the DP and the diagonal algorithms. The experimental results show that the diagonal algorithm is faster when L is close to r or L is very small.

References

[1] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21(1), pp. 168–173, 1974.

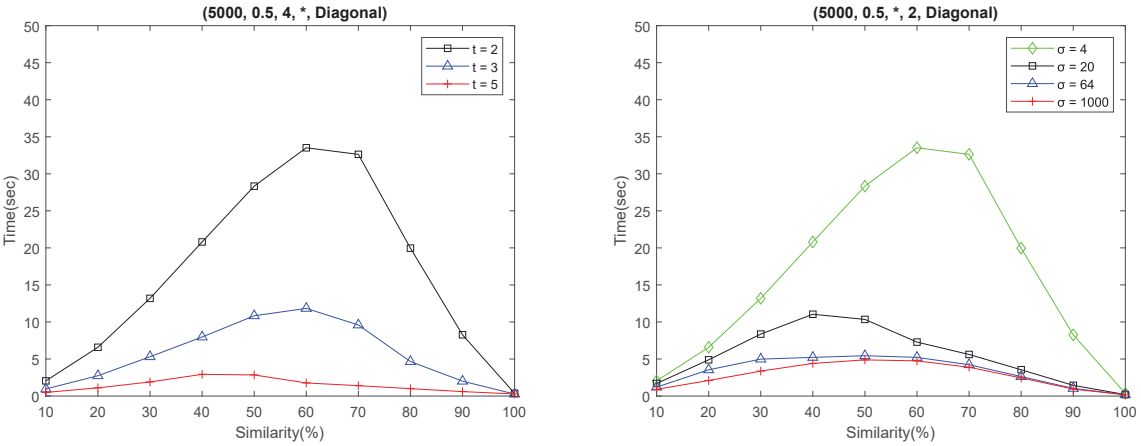
[2] K. S. Huang, C. B. Yang, K. T. Tseng, H. Y. Ann, and Y. H. Peng, "Efficient algorithms for finding interleaving relationship between sequences," *Information Processing Letters*, vol. 105, pp. 188–193, 2008.

[3] Y. H. Peng, C. B. Yang, C. T. Tseng, and K. S. Huang, "An algorithm and applications to sequence alignment with weighted constraint," *International Journal of Foundations of Computer Science*, vol. 21, pp. 51–59, 2010.



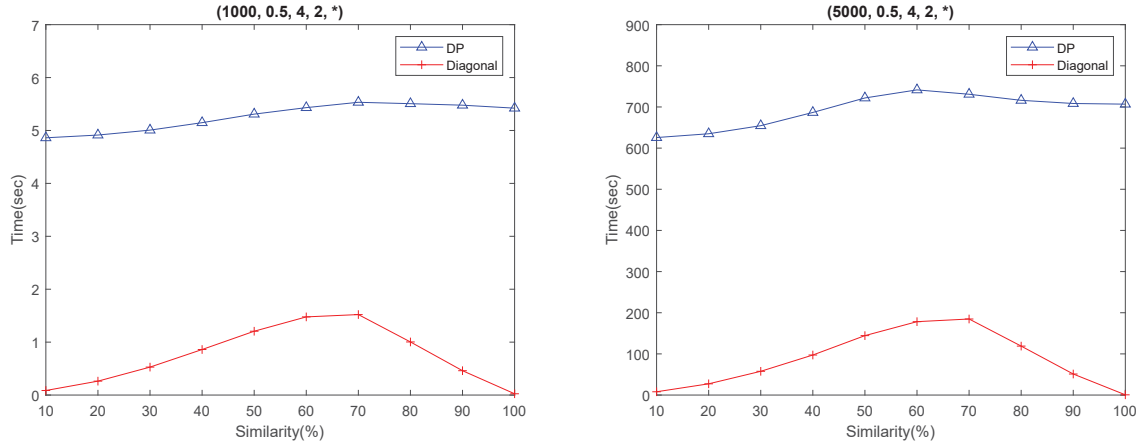
(a) $|P| = 1000$. (b) $|P| = 5000$.

Figure 1: The average execution time of two MLCS$_t$ algorithms with $\gamma = 0.5$, $\sigma = 4$ and $t = 2$.



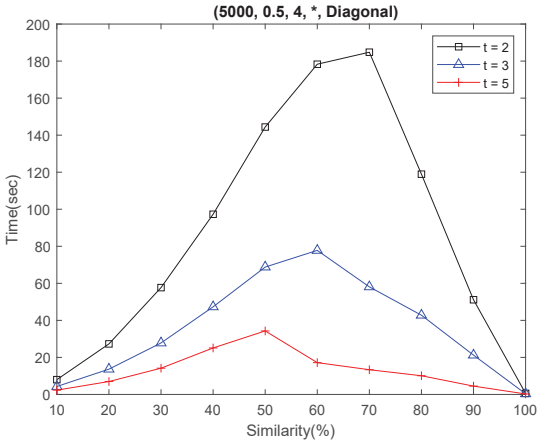
(a) Various t values with $\sigma = 4$. (b) Various alphabet sizes σ with $t = 2$.

Figure 2: The average execution time for the diagonal MLCS$_t$ algorithm with $|P| = 5000$ and $\gamma = 0.5$.

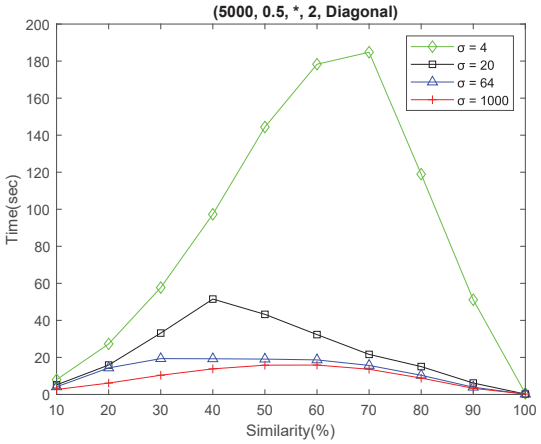


(a) $|P| = 1000$. (b) $|P| = 5000$.

Figure 3: The average execution time of two MLCS$_{t+}$ algorithms with $\gamma = 0.5$, $\sigma = 4$ and $t = 2$.



(a) Various t values with $\sigma = 4$.



(b) Various alphabet sizes σ with $t = 2$.

Figure 4: The average execution time for the diagonal MLCS$_{t+}$ algorithm with $|P| = 5000$ and $\gamma = 0.5$.

[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] A. M. Rahman and M. S. Rahman, "Effective sparse dynamic programming algorithms for merged and block merged LCS problems," *Journal of Computers*, vol. 9(8), pp. 1743–1754, 2014.

[6] S. Grabowski, "New tabulation and sparse dynamic programming based techniques for sequence similarity problems," *Discrete Applied Mathematics*, vol. 201, pp. 96–103, 2016.

[7] 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.

[8] G. Benson, A. Levy, and B. R. Shalom, "Longest common subsequence in k length substrings," in *Proceedings of the 6th International Conference on Similarity Search and Applications*, vol. 8199, 2013, pp. 257–265.

[9] S. Deorowicz and S. Grabowski, "Efficient algorithms for the longest common subsequence in k-length substrings," *Information Processing Letters*, vol. 114, pp. 634–638, 2014.

[10] G. Benson, A. Levy, S. Maimoni, D. Noifeld, and B. R. Shalom, "LCSk: A refined similarity measure," *Theoretical Computer Science*, vol. 638, pp. 11–26, 2016.

[11] D. Zhu, L. Wang, T. Wang, and X. Wang, "A space efficient algorithm for the longest common subsequence in k-length substrings," *Theoretical Computer Science*, vol. 687, pp. 79–92, 2017.

[12] F. Pavetić, I. Katanić, G. Matula, G. Žužić, and M. Šikić, "Fast and simple algorithms for computing both LCS_k and LCS_{k+}," *CoRR, abs/1705.07279*, 2018.

[13] F. Pavetić, G. Žužić, and M. Šikić, "$LCSk{+}{+}$: Practical similarity metric for long strings," *CoRR, abs/1407.2407*, 2014.

[14] Y. Ueki, Diptarama, M. Kurihara, Y. Matsuoka, K. Narisawa, R. Yoshinaka, H. Bannai, S. Inenaga, and A. Shinohara, "Longest common subsequence in at least k length order-isomorphic substrings," in *Proceedings of the 43rd International Conference on Current Trends in Theory and Practice of Computer Science*, vol. 10139, Limerick, Ireland, 2017, pp. 364–374.

[15] 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.