An Efficient Merged Longest Common Subsequence Algorithm for Similar Sequences [*]

Kuo-Tsung Tseng[a], De-Sheng Chan[b] and Chang-Biau Yang[b†]

[a] Department of Shipping and Transportation Management
National Kaohsiung Marine University, Kaohsiung, Taiwan
tsengkt@nkmu.edu.tw

[b]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan

Abstract

Given a pair of merging sequences A and B, and a target sequence T, the merged longest common subsequence (MLCS) problem is to find out a longest common subsequence (LCS) between sequences $E(A,B)$ and T, where $E(A,B)$ is obtained from merging two subsequences of A and B. In this paper, we propose an algorithm for solving the MLCS problem in $O(L(r-L+1)m)$ time, where r and L denote the lengths of T and MLCS, respectively, and m denotes the minimum length of A and B. From the time complexity, it is clear that our algorithm is extremely efficient when T and $E(A,B)$ are very similar. Experimental results show that our algorithm is faster than other previously published MLCS algorithms for sequences with high similarity.

1 Introduction

To measure the similarity of two sequences is essential in many applications, such as computational biology, pattern matching, plagiarism detection, voice recognition, and so on. The most well-known method to measure the similarity of two sequences in computer science is to ultilize the longest common subsequence (LCS) algorithms, which have been proposed by several researchers [1–3, 5, 8].

The *longest common subsequence* (LCS) problem is defined as follows. Given two sequences $A = a_1a_2a_3\ldots a_m$ and $B = b_1b_2b_3\ldots b_n$, the problem is to find a longest sequence which is a subsequence of both A and B, denoted by $LCS(A,B)$ for short. Here, a subsequence of a sequence S can be obtained by deleting an arbitrary number of characters at arbitrary positions of S. For example, given $A = accgt$ and $B = tagct$. Then the answer of $LCS(A,B)$ is agt or act. Note that the number of the longest common subsequences may be more than one.

The LCS problem has been extensively studied for more than 40 years. In 1977, Hunt and Syzmanski proposed an algorithm, with $O((R+m)\log m)$ time, that considers only the matching characters between the two sequences [7], where R denotes the total number of pairs of matching characters between the two sequences. The algorithm performs badly with similar sequences and its time complexity becomes $O(m^2 \log m)$ in the worst case time. Nakatsu and Yajima proposed an $O(n(m-L))$-time algorithm, which is more efficient for similar sequences [9], where L denotes the length of $LCS(A,B)$.

The *merged LCS* (MLCS) problem, a variant of the LCS problems, is to find the longest common subsequence of a merged sequence $E(A,B)$ and a target sequence T. A merged sequence $E(A,B)$ is a sequence merged by subsequences of A and B. The MLCS problem was defined by Huang *et al.* in 2008 and they proposed an $O(mnr)$-time algorithm to solve this problem with the dynamic programming method [6], where m, n and r denote the lengths of sequences A, B and T, respectively. In 2010, Peng *et al.* proposed an algorithm for solving the MLCS problem in $O(Lnr)$ time [10], where L denotes the length of the answer $MLCS(A,B,T)$. Their algorithm performs efficiently if the length of $MLCS(A,B,T)$ is

[*]This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 104-2221-E-110 -018 -MY3.

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

Table 1: The time complexities of the MLCS alogrithms. Here, w denotes the word size of a computer, L denotes the length of the MLCS answer, R and P denote the total numbers of matching pairs in A, T and B, T respectively.

Authors	**MLCS alogrithm**
Huang *et al.* [6]	$O(mnr)$
Peng *et al.* [10]	$O(Lnr)$
Deorowicz and Danek [4]	$O(\lceil r/w \rceil mn \log w)$
Rahman and Rahman [11]	$O((Rr + Pm) \log\log r)$
This paper	$O(L(r - L + 1)m)$

short. Based on Huang's [6] algorithm, Deorowicz and Danek proposed the bit-parallel algorithm for solving MLCS problem in $O(\lceil r/w \rceil mn \log w)$ time [4], which is about 10 times faster than Huang's algorithm, where w denotes the word size of a computer. In 2014, Rahman *et al.* proposed an $O((Rr + Pm) \log\log r)$-time algorithm, which uses the *BoundedHeap* data structure to reduce the time complexity [11], where R and P denote the total numbers of matching pairs in A, T and B, T respectively. The time complexities of these algorithms are shown in Table 1.

From the time complexities of Peng's and Rahman's algorithms, we can see that their algorithms are efficient for the MLCS problem if the merged sequence and the target sequence are dissimilar, which means that L, R and P are small. On the contrary, if they are similar, these algorithms become inefficient. Therefore, in this paper, we propose a new algorithm focusing on the MLCS problem of similar sequences. The time complexity of our new MLCS algorithm is $O(L(r - L + 1)m)$.

The rest of this paper is organized as follows. In Section 2, we introduce some preliminaries and definition of the MLCS problem. In Section 3, we present our MLCS algorithm. Then we do experiments on random sequences, and compare the execution times with previously published algorithms of MLCS in Section 4. Finally, we give our conclusions and possible future works in Section 5.

2 Preliminaries

Let a string $S = s_1 s_2 s_3 \dots s_{|S|}$ be a sequence of characters over a finite alphabet Σ. The notations used here are listed as follows:

- $|S|$: the length of sequence S.
- s_i: the ith character of S.
- $S_{i..j}$: the substring of S from position i to position j, where $S_{i..j} = \emptyset$ if $j < i$.

Given a pair of merging sequences $A = a_1 a_2 a_3 \dots a_m$ and $B = b_1 b_2 b_3 \dots b_n$, and a target sequence $T = t_1 t_2 t_3 \dots t_r$, the *merged LCS* problem is to find the LCS of the merged sequence $E(A, B)$ and the target sequence T [6]. Let $E(A, B) = e_1 e_2 e_3 \dots e_{m+n}$. Suppose e_i and e_j are got from $a_{i'}$ and $a_{j'}$, respectively. $i < j$ if and only if $i' < j'$. It has the similar property for B. Here, it is assumed $m \leq n$. For example, consider sequences $A = a_1 a_2 a_3 = acg$, $B = b_1 b_2 b_3 b_4 = ccca$ and a target sequence $T = t_1 t_2 t_3 t_4 t_5 t_6 = actcgc$. $MLCS(A, B, T) = accgc$ is the answer of the MLCS problem.

Let $H(i, j, k)$ denote the length of $MLCS(A_{1..i}, B_{1..j}, T_{1..k})$. The dynamic programming formula [6] for solving the MLCS problem is given as follows.

$H(i, j, k) =$

$$\max \begin{cases} H(i-1, j, k-1) + 1 & \text{if } a_i = t_k \\ H(i, j-1, k-1) + 1 & \text{if } b_j = t_k \\ \max \begin{cases} H(i-1, j, k) \\ H(i, j-1, k) \\ H(i, j, k-1) \end{cases} & \text{if } a_i \neq t_k \text{ or } b_j \neq t_k \end{cases} \quad (1)$$

with the boundary conditions:
$H(i, 0, k) =$ length of $LCS(A_{1..i}, T_{1..k})$ and
$H(0, j, k) =$ length of $LCS(B_{1..j}, T_{1..k})$.

It is easy to see that the time complexity of Huang's algorithm is data independent. In other words, the time complexities are the same when the given data are in the worst case and the best case.

3 Our Merged LCS algorithm

In this section, we first give several definitions, facts, lemmas and one theorem. Then, with the theorem we propose an algorithm for solving the MLCS problem, whose time complexity is $O(L(r-L+1)m)$. The algorithm is efficient when the given sequences are extremely similar (L is close to r), or dissimilar (L is small).

Our MLCS algorithm is based on the concept of Nakatsu's LCS algorithm [9], which is designed for sequences with high similarity. Let $MLCS(A_{1..i}, B_{1..j}, T_{1..k})$ denote the length of the answer for more clarity. We first give the definition of the *dominating set* $D_{k,s}$ as follows.

Definition 1. *For any two 2-tuples of $\langle i_1, j_1\rangle$ 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 $j_1 \le j_2$.*

Definition 2. *Let $D_{k,s}$, where $k, s \ge 0$, denote the dominating set $\{\langle i_1, j_1\rangle |\ MLCS(A_{1..i_1},\ B_{1..j_1},\ T_{1..k}) = s$, and for any $\langle i_2, j_2\rangle \neq \langle i_1, j_1\rangle$, $i_2 \le i_1$ and $j_2 \le j_1$, $MLCS(A_{1..i_2},\ B_{1..j_2}, T_{1..k}) \le s-1\}$. In other words, any two distinct elements in $D_{k,s}$ do not dominate each other.*

By Definition 2 and the definition of the MLCS problem, there are some facts presented as follows.

Fact 1. *$D_{k,s} = \emptyset$ if $k < s$, $k < 0$ or $s < 0$.*

Fact 2. *$MLCS(\ A_{1..i},\ B_{1..j},\ T_{1..k}) \le MLCS(\ A_{1..x},\ B_{1..j},\ T_{1..k})$ if and only if $i \le x$.*

Fact 3. *$MLCS(\ A_{1..i},\ B_{1..j},\ T_{1..k}) \le MLCS(\ A_{1..i},\ B_{1..y},\ T_{1..k})$ if and only if $j \le y$.*

Fact 4. *$MLCS(\ A_{1..i},\ B_{1..j},\ T_{1..k-1}) \le MLCS(\ A_{1..i},\ B_{1..j},\ T_{1..k})$.*

By Definition 1 and the above facts, we have the following lemmas.

Lemma 1. *If $MLCS(A_{1..i}, B_{1..j}, T_{1..k}) = s$, there exists $\langle x, y\rangle$, where $x \le i$ and $y \le j$, such that $MLCS(A_{1..x}, B_{1..y}, T_{1..k}) = s - 1$, where $k, s \ge 1$.*

Proof. By the definition of the MLCS problem, $LCS(E(A_{1..i}, B_{1..j}), T_{1..k})$ has a common subsequence with length s. Let the merged sequence $E(A_{1..i}, B_{1..j}) = e_1e_2e_3 \dots\ e_{i+j}$. Here, let $\langle e_{x_1}, t_{y_1}\rangle < \langle e_{x_2}, t_{y_2}\rangle$ denote that $e_{x_1} < e_{x_2}$ and $t_{y_1} < t_{y_2}$. There must exist s matching pairs $\langle e_{x_1}, t_{y_1}\rangle < \langle e_{x_2}, t_{y_2}\rangle < \cdots < \langle e_{x_s}, t_{y_s}\rangle$ to be a common subsequence with length s, where $1 \le x_1 < x_2 < \cdots < x_s \le (i+j)$ and $1 \le y_1 < y_2 < \cdots < y_s \le k$. Therefore, the common subsequence with length $s - 1$ can be obtained by deleting the last matching pair $\langle e_{x_s}, t_{y_s}\rangle$. Thus, the lemma holds. □

Lemma 2. *For $\langle i_1, j_1\rangle \in D_{k,s-1}$ and $\langle i_2, j_2\rangle \in D_{k,s}$, $k, s \ge 1$, it is true that $i_1 < i_2$ or $j_1 < j_2$.*

Proof. Assume that there exists $\langle i_2, j_2\rangle \in D_{k,s}$ where $i_2 \le i_1$ and $j_2 \le j_1$. By Lemma 1, there exists $\langle i_3, j_3\rangle \in D_{k,s-1}$ such that $i_3 \le i_2$ and $j_3 \le j_2$. Thus, we get $i_3 \le i_1$ and $j_3 \le j_1$, which implies that $\langle i_3, j_3\rangle$ dominates $\langle i_1, j_1\rangle$ in $D_{k,s-1}$, which is a contradition of Definition 2. Therefore, the assumption is not true and the lemma holds. □

Lemma 3. *For $\langle i_1, j_1\rangle \in D_{k-1,s}$ and $\langle i_2, j_2\rangle \in D_{k,s}$, $k, s \ge 1$, it is true that $i_2 \le i_1$ or $j_2 \le j_1$.*

Proof. Assume that there exists $\langle i_1, j_1\rangle \in D_{k-1,s}$ where $i_1 < i_2$ and $j_1 < j_2$. By Fact 4, we have

$$MLCS(A_{1..i_1}, B_{1..j_1}, T_{1..k-1}) = s \le MLCS(A_{1..i_1}, B_{1..j_1}, T_{1..k}),$$

Then, we have

$$MLCS(A_{1..i_1}, B_{1..j_1}, T_{1..k}) = s + \lambda,\ \lambda \ge 0.$$

Case 1: If $\lambda = 0$, then $\langle i_1, j_1\rangle$ can be put into $D_{k,s}$. Thus, $\langle i_1, j_1\rangle$ dominates $\langle i_2, j_2\rangle$ in $D_{k,s}$.
Case 2: If $\lambda \ge 1$, then by applying Lemma 1 λ times, we finally find $\langle x, y\rangle$ that $x \le i_1 < i_2$, $y \le j_1 < j_2$, and $MLCS(A_{1..x}, B_{1..y}, T_{1..k}) = s$. It implies $\langle x, y\rangle$ dominates $\langle i_2, j_2\rangle$ in $D_{k,s}$.

Thus, both cases contradict Definition 2. Therefore, the assumption is not true and the lemma holds. □

For solving the MLCS problem efficiently, we first define two functions EXTEND and DOMINATE. Let $\langle i_1, j_1\rangle \in D_{k-1,s-1}$. Note that $MLCS(A_{1..i_1}, B_{1..j_1}, T_{1..k-1}) = s - 1$. By observing one more character in T (i.e. t_k), EXTEND$(D_{k-1,s-1})$ is defined to extend an element $\langle i_1, j_1\rangle \in D_{k-1,s-1}$ to another 2-tuples $\langle i_2, j_1\rangle$ and $\langle i_1, j_2\rangle$, $i_1 < i_2 \le m$ and $j_1 < j_2 \le n$, such that i_2 and j_2 are the smallest indexes for $a_{i_2} = t_k$ and $b_{j_2} = t_k$, respectively. After this extension, we get $MLCS(A_{1..i_2}, B_{1..j_1},\ T_{1..k}) \ge s$ and $MLCS(A_{1..i_1}, B_{1..j_2},\ T_{1..k}) \ge s$. In other words, we extend one more character in T to find one more common character in A or B, and thus the length of the solution is extended from $s - 1$ to s or more. If both $\langle i_2, j_1\rangle$ and $\langle i_1, j_2\rangle$ do not exist, the extension result is defined to be empty.

For example, suppose $A = acg$, $B = ccca$, $T = actcgc$. We have $MLCS(\emptyset,\ B_{1..4},\ T_{1..1}) = 1$ and $MLCS(A_{1..1}, \emptyset, T_{1..1}) = 1$, thus $D_{1,1} = \{\langle 0, 4\rangle, \langle 1, 0\rangle\}$. Next, $\langle 0, 4\rangle$ is extended to $\langle 2, 4\rangle$ by finding the index of the first matching character of c (t_2) in A after a_0. Similarly, $\langle 1, 0\rangle$ is extended to $\langle 2, 0\rangle$ and $\langle 1, 1\rangle$ from A and B, respectively, by finding the index of the first matching character of c (t_2) after a_1 and b_0. Thus, we get an extended set $W =$ EXTEND$(D_{1,1}) = \{\langle 1, 1\rangle, \langle 2, 0\rangle, \langle 2, 4\rangle\}$.

DOMINATE is defined to remove all 2-tuples those are dominated by others in a set. In other words, the input set for DOMINATE may not be a dominating set, while its output is a dominating set. For example, suppose $W = \{\langle 1, 1\rangle, \langle 2, 0\rangle, \langle 2, 4\rangle\}$. We get DOMINATE$(W) = \{\langle 1, 1\rangle, \langle 2, 0\rangle\}$, since $\langle 2, 4\rangle$ is dominated by $\langle 2, 0\rangle$.

With the definitions of EXTEND and DOMINATE, we have the main theorem for solving the MLCS problem.

Theorem 1. *$D_{k,s} =$ DOMINATE (EXTEND $(D_{k-1,s-1}) \cup D_{k-1,s})$, $1 \le k, s \le r$.*

Proof. For each $\langle x', y' \rangle \in \text{EXTEND}(D_{k-1,s-1}) \cup D_{k-1,s}$, our proof focuses on that if $MLCS(A_{1..x'}, B_{1..y'}, T_{1..k}) > s$, $\langle x', y' \rangle$ will be dominated by another 2-tuple in $D_{k,s}$.

- Consider the case for $\text{EXTEND}(D_{k-1,s-1})$.
  Let $\langle i_1, j_1 \rangle \in D_{k-1,s-1}$. And suppose $\text{EXTEND}(\{\langle i_1, j_1 \rangle\}) = \{\langle x, j_1 \rangle, \langle i_1, y \rangle\}$. By Facts 2, 3 and 4, we have

  Case 1: $MLCS(A_{1..x}, B_{1..j_1}, T_{1..k}) = s + \lambda$, $\lambda \geq 0$,

  Case 2: $MLCS(A_{1..i_1}, B_{1..y}, T_{1..k}) = s + \mu$, $\mu \geq 0$.

  Case (a): If $\lambda = 0$, then $MLCS(A_{1..x}, B_{1..j_1}, T_{1..k}) = s$.

  Case (b): If $\lambda > 0$, then by applying Lemma 1 λ times, we find $\langle x_1, y_1 \rangle$, $x_1 \leq x$ and $y_1 \leq j_1$, such that
  $$MLCS(A_{1..x_1}, B_{1..y_1}, T_{1..k}) = s.$$
  It implies $\langle x_1, y_1 \rangle$ dominates $\langle x, j_1 \rangle$. $\langle x, j_1 \rangle$ will be removed after the DOMINATE function is applied. Thus, $\langle x, j_1 \rangle$ cannot be in $D_{k,s}$. The proof for case 2 is similar.

- Consider the case for $D_{k-1,s}$.
  Suppose that there exists $\langle i_2, j_2 \rangle \in D_{k-1,s}$ such that
  $$MLCS(A_{1..i_2}, B_{1..j_2}, T_{1..k}) = s + \lambda,\ \lambda \geq 1.$$
  Then by applying Lemma 1 λ times, we find $\langle x_2, y_2 \rangle$, $x_2 \leq i_2$ and $y_2 \leq j_2$, such that
  $$MLCS(A_{1..x_2}, B_{1..y_2}, T_{1..k}) = s.$$
  It implies $\langle x_2, y_2 \rangle$ dominates $\langle i_2, j_2 \rangle$. $\langle i_2, j_2 \rangle$ will be removed after the DOMINATE function is applied. Thus, $\langle i_2, j_2 \rangle$ cannot be in $D_{k,s}$.

According to the above, for each $\langle x', y' \rangle \in \text{EXTEND}(D_{k-1,s-1}) \cup D_{k-1,s}$, if $MLCS(A_{1..x'}, B_{1..y'}, T_{1..k}) > s$, $\langle x', y' \rangle$ is dominated by another 2-tuple in $D_{k,s}$. And after the DOMINATE function is applied, all dominated elements will be removed. Therefore, the theorem holds. ☐

Algorithm 1 Computing the MLCS length.

```
Input: Sequences $A$, $B$ and $T$
Output: Length of $MLCS(A, B, T)$
 1: Construct the arrays of $next_A$ and $next_B$
 2: $L \leftarrow 0$
 3: for $i = 1 \rightarrow r$ do
 4:     Set $D_{i-1,0} = \{\langle 0, 0 \rangle\}$
 5:     for $s = 1 \rightarrow r - i + 1$ do
 6:         $k \leftarrow i + s - 1$
 7:         $W \leftarrow \text{EXTEND}(D_{k-1,s-1})$
 8:         $D_{k,s} \leftarrow W \cup D_{k-1,s}$
 9:         $D_{k,s} \leftarrow \text{DOMINATE}(D_{k,s})$
10:         if $D_{k,s} = \emptyset$ then
11:             break
12:         if $s > L$ then
13:             $L \leftarrow s$
14:     if $i > r - L$ then
15:         break
16: return $L$
```

Table 2: The construction of $D_{k,s}$ in our MLCS algorithm with $A = acg$, $B = ccca$ and $T = actcgc$.

	s (length)					
	0	1	2	3	4	5
$i = 1$	$D_{0,0}$	$D_{1,1}$	$D_{2,2}$			
	$\langle 0, 0 \rangle$	$\langle 0, 4 \rangle$ $\langle 1, 0 \rangle$	$\langle 1, 1 \rangle$ $\langle 2, 0 \rangle$ ~~$\langle 2, 4 \rangle$~~			
$i = 2$	$D_{1,0}$	$D_{2,1}$	$D_{3,2}$	$D_{4,3}$	$D_{5,4}$	$D_{6,5}$
	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$ ~~$\langle 0, 4 \rangle$~~ $\langle 1, 0 \rangle$ ~~$\langle 2, 0 \rangle$~~	$\langle 1, 1 \rangle$ $\langle 2, 0 \rangle$	$\langle 1, 2 \rangle$ $\langle 2, 1 \rangle$ ~~$\langle 2, 1 \rangle$~~	$\langle 3, 1 \rangle$ ~~$\langle 3, 2 \rangle$~~	$\langle 3, 2 \rangle$

With Theorem 1, the MLCS problem can be solved by calculating $D_{k,s}$. The pseudo code of our MLCS algorithm is shown in Algorithm 1. Table 2 demonstrates an example for the construction of $D_{k,s}$ in our MLCS algorithm.

The main loop (containing lines 3 through 15) constructs $D_{k,s}$ with the row-major scheme shown in Table 2. In the ith round (in line 3), it scans T sequentially starting from t_i to construct sets $D_{i,1}$, $D_{i+1,2}$, $D_{i+2,3}$, $\cdots$, $D_{i+s-1,s}$ until the set cannot be constructed, where $s = MLCS(A_{1..m}, B_{1..n}, T_{1..(i+s-1)})$, the maximum length that can be obtained in this round. Line 7 extends $D_{k-1,s-1}$ with one more character in T (t_k) to the potential content of $D_{k,s}$. Line 9 uses DOMINATE to remove unnecessary (dominated) 2-tuples. Line 10 decides whether the termination condition is satisfied or not in this round. In lines 12-13, L stores the maximum value of s (the maximum length in this round). We quit our MLCS algorithm when $i > r - L$. The reason is that the maximum length of the ith round is $r - i + 1$. Thus, after round $i+1$, the maximum length will be no more than the current L. Eventually, the final L represents the length of $MLCS(A_{1..m}, B_{1..n}, T_{1..r})$.

Table 2 shows an example that $A = acg$, $B = ccca$ and $T = actcgc$. In the first round ($i = 1$), we first set $D_{0,0} = \{\langle 0, 0 \rangle\}$, and then $D_{1,1}$ is constructed from $\text{DOMINATE}(\text{EXTEND}(D_{0,0}) \cup D_{0,1})$ according to Theorem 1. $D_{0,1}$ is an empty set by Fact 1. Thus, $D_{1,1}=\{\langle 0, 4 \rangle, \langle 1, 0 \rangle\}$. And then $D_{2,2}$ can be constructed from $\text{DOMINATE}(\text{EXTEND}(D_{1,1}) \cup D_{1,2})$. Note that $\langle 2, 4 \rangle$ will be removed, since $\langle 2, 4 \rangle$ is dominated by $\langle 2, 0 \rangle$. Thus, $D_{2,2}=\{\langle 1, 1 \rangle, \langle 2, 0 \rangle\}$. $D_{3,3}$ is an empty set, because we cannot extend any 2-tuple from set $D_{2,2}$. Accordingly, the first round stops. In the second round ($i = 2$), set $D_{2,1}$ can be constructed from $D_{1,0}$ and $D_{1,1}$. Repeating the above steps, we get sets $D_{2,1}$, $D_{3,2}$, $D_{4,3}$, $D_{5,4}$ and $D_{6,5}$ in this round. Our MLCS algorithm terminates since the optimal length is got in the second round, which is $r - i + 1$. Therefore, the length of $MLCS(A_{1..m}, B_{1..n}, T_{1..r}) = 5$.

Now, we go into the details of the two functions, Extend and Dominate. Extend considers only the matching characters, such as $a_i = t_k$ and $b_j = t_k$. To find out the matching pairs efficiently, we build a symbol table with Peng's algorithm [10]. In the symbol table, the function $next_A(\alpha, k) = k'$ denotes the next position of character α in A after position k. $k' = \infty$ if no such k can be found. It takes $O(m|\Sigma|)$ time to construct the symbol table of A in the preprocessing stage, where $|\Sigma|$ denotes the number of distinct symbols in A. It requires only constant time to get the answer of $next_A(\alpha, k)$. Similarly, the symbol tables of B and T can also be constructed.

The Extend function is shown in Function 1. We use $next_A$ and $next_B$ to find the index of the first matching character t_k after a_{PrevA} and b_{PrevB}, respectively. If we extend successfully $\langle PrevA, PrevB\rangle$ to another 2-tuples, then it is true that $PosA \leq m$ and $PosB \leq n$.

Function 1 The extension of $D_{k-1,s-1}$ to $D_{k,s}$.

```
Input: $D_{k-1,s-1}$
Output: $W$
 1: function Extend($D_{k-1,s-1}$)
 2:     for each $\langle PrevA, PrevB\rangle \in D_{k-1,s-1}$ do
 3:         $PosA \leftarrow next_A(t_k, PrevA)$
 4:         $PosB \leftarrow next_B(t_k, PrevB)$
 5:         if $PosA \leq m$ then
 6:             Add $\langle PosA, PrevB\rangle$ into $W$
 7:         if $PosB \leq n$ then
 8:             Add $\langle PrevA, PosB\rangle$ into $W$
 9:     Return $W$
10: end function
```

Dominate is used to remove the dominated 2-tuples in a set. To remove dominated 2-tuples efficiently, we apply Peng's algorithm [10] by using the bucket sort, whose time complexity is $O(|Q| + min(Z_i, Z_j))$ time, where $|Q|$ denotes the number of 2-tuples $\langle i, j\rangle \in Q$, and Z_i, Z_j denote the maximum values of integer i and j of all 2-tuples, respectively. That is, 2-dimensional minima finding algorithm can be done in linear time if $|Q|$ is finite and each 2-tuple in Q is at a non-negative integer point with boundaries.

4 Experimental Results

In this section, to demonstrate the time efficiencies of various MLCS algorithms, we perform some simulation experiments on random sequences with high similarities. These algorithms were implemented by Visual Studio C++ 2013 software, and they were tested on a computer with 64-bit Windows 7 OS, CPU clock rate of 3.4GHZ (Intel Xeon CPU E3-1231 v3) and 16 GB of RAM.

The experiments focus on the relationship of execution time and similarities between merged sequences and target sequences. Both Peng's and Rahman's algorithm are time-efficient for dissimilar sequences. However, the performances of these two algorithms are worse than Huang's DP-based algorithm if the given sequences have high similarity. Thus, we do not involve the former two algorithms in our experiments.

Our experiments include randomly generated sequences A, B and T of lengths 500, 500 and 1000, respectively. Each experiment was repeated 50 times to get the average execution time.

In Table 3, the first column shows the similarity of $E(A, B)$ and T. For example, when the similarity is 98%, the MLCS length is about 980, where $|A| = |B| = 500$ and $|T| = 1000$. This table indicates that our MLCS algorithm is always better than the DP-based algorithm and it becomes the most efficient algorithm for the sequences with very high similarity.

5 Conclusion

In this paper, we propose an $O(L(r-L+1)m)$-time merged LCS algorithm for similar sequences, where r, m and L denote the length of sequences T, A and the MLCS length, respectively.

We compare our algorithm with previously published MLCS algorithms on random sequences. The experimental results show that our algorithm is better than Huang's DP-base algorithm [6]. Our algorithm is faster than all previous MLCS algorithms when the given sequences are very similar. However, for the case that the length of sequence T is much longer than the length sum of A and B, our algorithm does not perform well. The reason is that the extension of A and B is done along the progress of T. One possible improvement is to do the extension along the progress of A or B. It could be done with the similar algorithms.

The computation of a dominating set may be improved with some special scheme. And thus, the analysis of time complexity will be more precise. We have

Table 3: The average execution time of various MLCS algorithms. Here, **DP** denotes Huang's DP MLCS algorithm [6], and **Bit** is Deorowicz's bit-parallel MLCS algorithm [4], and **Ours** means the algorithm presented in this paper.

Similarity	**DP**	**Bit**	**Ours**
	$\|\Sigma\| = \mathbf{4}$		
95%	1.411	**0.086**	0.160
96%	1.415	**0.086**	0.123
97%	1.414	0.086	**0.080**
98%	1.414	0.086	**0.049**
99%	1.413	0.086	**0.026**
100%	1.415	0.086	**0.003**
	$\|\Sigma\| = \mathbf{20}$		
90%	1.268	**0.087**	0.116
91%	1.267	**0.087**	0.100
92%	1.267	**0.087**	**0.087**
93%	1.266	0.086	**0.073**
94%	1.267	0.086	**0.060**
95%	1.267	0.086	**0.050**
100%	1.267	0.086	**0.001**
	$\|\Sigma\| = \mathbf{256}$		
80%	1.111	**0.081**	0.123
85%	1.112	**0.081**	0.084
90%	1.113	0.081	**0.050**
95%	1.112	0.081	**0.022**
100%	1.110	0.080	**0.001**
	$\|\Sigma\| = \mathbf{1000}$		
10%	1.094	0.081	**0.044**
15%	1.095	0.081	**0.065**
20%	1.096	**0.081**	0.087
80%	1.102	**0.081**	0.102
85%	1.103	0.082	**0.074**
90%	1.102	0.081	**0.044**
95%	1.099	0.081	**0.021**
100%	1.098	0.081	**0.003**

also applied the algorithm to the blocked merged LCS problem, with slight modification. However, we do not present the algorithm due to the page limitation.

References

[1] H.-Y. Ann, C.-B. Yang, C.-T. Tseng, and C.-Y. Hor, "A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings," *Information Processing Letters*, Vol. 108, pp. 360–364, 2008.

[2] A. Apostolico, "Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings," *Information Processing Letters*, Vol. 23, pp. 63–69, 1986.

[3] M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon, and J. F. Reid, "A fast and practical bit-vector algorithm for the longest common subsequence problem," *Information Processing Letters*, Vol. 80, pp. 279–285, 2001.

[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] J. Guo and F. Hwang, "An almost-linear time and linear space algorithm for the longest common subsequence problem," *Information Processing Letters*, Vol. 94, pp. 131–135, 2005.

[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] J. Liu, G. Huang, Y. Wang, and R. Lee, "Edit distance for a run-length-encoded string and an uncompressed string," *Information Processing Letters*, Vol. 105, pp. 12–16, 2007.

[9] Y. K. Narao Nakatsu and S. Yajima, "A longest common subsequence algorithm suitable for similar text strings," *ACTA Informatica*, Vol. 18, pp. 171–179, 1982.

[10] 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," *International Journal of Innovative Computing, Information and Control*, Vol. 6, pp. 1935–1947, 2010.

[11] 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.