Algorithms for the Constrained Longest Common Subsequence Problem with t-length Substrings *

Kexin Zhu[a], Chang-Biau Yang[a]†and Kuo-Tsung Tseng[b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[b]Department of Shipping and Transportation Management
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

The longest common subsequence (*LCS*) *problem and its variants, such as the constrained LCS* (*CLCS*) *problem, and the LCS with t-length substrings* (LCS_t) *problem, are well-studied problems as the similarity measurement for strings or biosequences. In this paper, we first define the new variant, which is the* constrained longest common subsequence with t-length substrings ($CLCS_t$) *problem. Then, we propose three algorithms for solving the problem. The time complexity of the dynamic programming algorithm is $O(mnr)$, where m, n, and r are the lengths of the two target sequences and the constraint sequence, respectively. The row-wise algorithm is our second method, with $O(r \times \min\{mL+R, R\log L\}+m+n)$ time, where L denotes the answer length, and R denotes the number of t-match pairs between the two target sequences. Our third method is the diagonal algorithm with $O(rL(m-L)+R)$ time. As experimental results show, the diagonal algorithm is the most efficient.*

Keywords: longest common subsequence (LCS), constrained LCS, LCS_t, dynamic programming, diagonal

1 Introduction

The goal of the LCS problem is to measure the similarity of two or more given sequences by the LCS length. In the past decades, many algorithms have been proposed for solving the LCS problem [1–5].

Given two sequences A and B with a constraint sequence P, the *constrained longest common subsequence* (CLCS) problem aims to find the LCS of A and B containing P as a subsequence of the answer. The CLCS problem was first introduced and solved by Tsai [6], who presented a *dynamic programming* (DP) algorithm with $\text{O}(m^2n^2r)$ time and space, where $|A| = m$, $|B| = n$ and $|P| = r$. Chin *et al.* [7] proposed a DP-based method reducing both time and space complexities to $\text{O}(mnr)$. In 2018, Hung *et al.* [8] presented a diagonal-based [4] algorithm with $\text{O}(rL(m-L))$ time and $\text{O}(mr)$ space, where L denotes the CLCS length.

In 2013, Benson *et al.* [9] first defined a new variant of LCS, the *LCS with t-length substrings* (LCS_t), aiming to find the LCS answer consisting of common t-length substrings. They proposed a DP algorithm to solve the LCS_t problem, with $\text{O}(mnt)$ time and $\text{O}(mt)$ space. In 2014, Deorowicz and Grabowski [10] proposed an improved algorithm, reducing the time complexity to $\text{O}(mn)$. In 2020, Huang *et al.* [11] proposed a diagonal-based algorithm with $\text{O}(n(m-L))$ time and $\text{O}(n)$ space.

The summary of the above CLCS and LCS_t algorithms, including our algorithms, is presented in Table 1.

In this paper, we first define the *constrained LCS with t-length substrings* (CLCS_t) problem, a new variant combined from the CLCS problem and the LCS_t problem. In the CLCS_t problem, we are given two sequences A and B with a constraint sequence P. The answer is the LCS of A and B with containing P as a subsequence, and it is composed of common t-length substrings of A and B. We propose three algorithms for solving the CLCS_t problem. The first is a DP algorithm with $\text{O}(mnr)$ time, the second is a row-wise algorithm with $\text{O}(r \times \min\{mL+R, R\log L\}+m+n)$

*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).

Table 1: The time and space complexities of the CLCS and LCS_t algorithms. t: length of each common substring; L: length of the answer; R: number of t-match pairs between A and B; $r = |P| \leq m = |A| \leq n = |B|$; M, N, R': numbers of runs in the two target sequences and constraint sequence, respectively, with the run-length encoded format.

CLCS

Year	Author(s)	Time complexity	Space complexity	Note
2003	Tsai [6]	$O(m^2n^2r^2)$	$O(m^2n^2)$	All components
2004	Chin *et al.* [7]	$O(mnr)$	$O(mnr)$	DP
2012	Ann *et al.* [12]	$O(nmR' + nMr + Nmr)$		Run-length encoded
2018	Hung *et al.* [8]	$O(rL(m-L))$	$O(mr)$	Diagonal
2021	Ours	$O(mnr)$ $O(r \times \min\{mL + R, R \log L\} + m + n)$ $O(rL(m-L) + R)$	$O(mnr)$ $O(r(L + m + n))$ $O(rL + R)$	DP CLCS_t Row-wise CLCS_t Diagonal CLCS_t

LCS_t

Year	Author(s)	Time complexity	Space complexity	Note
2013	Benson *et al.* [9]	$O(mnt)$	$O(mt)$	DP
2014	Deorowicz and Grabowski [10]	DP: $O(mn)$ Sparse: $O(n + R \log L)$ Dense: $O(mn/t + m(t \log n)^{2/3})$ Dense-vEB: $O(mn \log\log m/t)$ DP-4R: $O(mn/\log m)$	DP: $O(mt)$ Sparse: $O(n + \min\{R, mL\})$ Dense: $O(m)$ Dense-vEB: $O(m \log\log m)$ DP-4R: $O(n + nt/\log n)$	Sparse DP, van Emde Boas tree
2018	Pavetić *et al.* [13]	$O(m + n + R + \min$ $(R \log L, R + mL))$	$O(m + n)$	Match pair
2020	Huang *et al.* [11]	$O(n(m-L))$ $O(n + L(m-L) \log n)$ $O(n + L(m-L) + R)$	$O(n)$	Diagonal

time, where L denotes the answer length, and R denotes the number of t-match pairs between A and B, and the third is a diagonal algorithm with $O(rL(m-L)+R)$ time. Theoretically and practically, the diagonal algorithm is the most efficient.

The organization of this paper is given as follows. Section 2 introduces some preliminaries of the CLCS_t problem. Then, in Sections 3 through 5, we propose three algorithms for solving the CLCS_t problem. Section 6 shows the experimental results. Finally, the conclusion is given in Section 7.

2 Preliminaries

A sequence $S = s_1 s_2 \cdots s_m$ consists of characters over a finite alphabet set Σ. $S_{i..j}$ represents the substring of S from positions i to j. $S_{i..j} = \emptyset$ if $i > j$. A subsequence of S is obtained by deleting an arbitrary number, or zero, of characters from S with the order-preserving.

Definition 1. (CLCS_t problem) *Given two sequences $A = a_1 a_2 \cdots a_m$ and $B = b_1 b_2 \cdots b_n$, with a constraint sequence $P = p_1 p_2 p_3 ... p_r$ and a constant positive integer t, the CLCS_t problem is to find the LCS with common t-length substrings of A and B that contains P as a subsequence.*

Suppose the answer length is $l \times t$, then there exists a common subsequence $C = A_{i_1-t+1..i_1} A_{i_2-t+1..i_2} \cdots A_{i_l-t+1..i_l} = B_{j_1-t+1..j_1} B_{j_2-t+1..j_2} \cdots B_{j_l-t+1..j_l}$, where $i_g \leq i_{g+1} - t$ and $j_g \leq j_{g+1} - t$ for $1 \leq g \leq l-1$, and C contains P as a subsequence. Note that each common substring is of length t.

3 The Dynamic Programming Algorithm

Definition 2. [9] *For sequences A and B, there is a t-match pair at (i, j), denoted as $match_t(i, j) = 1$, if and only if $A_{i-t+1..i} = B_{j-t+1..j}$.*

Definition 3. *For a sequence A and a constraint sequence P, let $suf_t(i, k)$ denote the length of the maximum suffix of $P_{1..k}$ contained in $A_{i-t+1..i}$ as a subsequence. That is, if $suf_t(i, k) = h$, then*

$P_{k-h+1..k}$ is a subsequence of $A_{i-t+1..i}$, but $P_{k-h..k}$ is not.

Definition 4. *Let $M_t(i,j,k)$ denote the CLCS$_t$ length for $A_{1..i}$ and $B_{1..j}$ with a constraint sequence $P_{1..k}$ contained as a subsequence in the CLCS$_t$ answer.*

Note that $suf_t(i,k) = 0$ if p_k is not in $A_{i-t+1..i}$. With the above definitions, the DP method for CLCS$_t$ is presented in Equation 1. The idea of Equation 1 is simple. Since $M_t(i,j,k)$ could only come from the vertical, horizontal or diagonal direction, it solves the problem by considering these 3 directions and finding the maximum. Moreover, Table 2 shows an example of this algorithm.

$$M_t(i,j,k) = \max \begin{cases} M_t(i-1,j,k) & \text{if } i \geq 1, \\ M_t(i,j-1,k) & \text{if } j \geq 1, \\ M_t(i-t,j-t, & \text{if } match_t(i,j) = 1 \\ k-h)+t & \text{and } suf_t(i,k) = h, \end{cases} \tag{1}$$

with boundary condition:
$M_t(i,0,0) = M_t(0,j,0) = 0$,
$M_t(i,0,k) = M_t(0,j,k) = -\infty$,
for $0 \leq i \leq m$, $0 \leq j \leq n$, and $1 \leq k \leq r$.

Table 2: An example of the DP algorithm for CLCS$_t$ with $A =$ `aactccacta`, $B =$ `aacccactcta`, $P =$ `ctt` and $t = 3$. '-' denotes $-\infty$.

		0	1 a	2 a	3 c	4 c	5 c	6 a	7 c	8 t	9 c	10 t	11 a
0		0	0	0	0	0	0	0	0	0	0	0	0
1	a	0	0	0	0	0	0	0	0	0	0	0	0
2	a	0	0	0	0	0	0	0	0	0	0	0	0
3	c	0	0	0	3	3	3	3	3	3	3	3	3
4	t	0	0	0	3	3	3	3	3	3	3	3	3
5	c	0	0	0	3	3	3	3	3	3	3	3	3
6	c	0	0	0	3	3	3	3	3	3	3	3	3
7	a	0	0	0	3	3	3	6	6	6	6	6	6
8	c	0	0	0	3	3	3	6	6	6	6	6	6
9	t	0	0	0	3	3	3	6	6	6	6	6	6
10	a	0	0	0	3	3	3	6	6	6	6	6	9

(a) $k = 0$

		0	1 a	2 a	3 c	4 c	5 c	6 a	7 c	8 t	9 c	10 t	11 a
0		-	-	-	-	-	-	-	-	-	-	-	-
1	a	-	-	-	-	-	-	-	-	-	-	-	-
2	a	-	-	-	-	-	-	-	-	-	-	-	-
3	c	-	-	-	3	3	3	3	3	3	3	3	3
4	t	-	-	-	3	3	3	3	3	3	3	3	3
5	c	-	-	-	3	3	3	3	3	3	3	3	3
6	c	-	-	-	3	3	3	3	3	3	3	3	3
7	a	-	-	-	3	3	3	6	6	6	6	6	6
8	c	-	-	-	3	3	3	6	6	6	6	6	6
9	t	-	-	-	3	3	3	6	6	6	6	6	6
10	a	-	-	-	3	3	3	6	6	6	6	6	9

(b) $k = 1$

		0	1 a	2 a	3 c	4 c	5 c	6 a	7 c	8 t	9 c	10 t	11 a
0		-	-	-	-	-	-	-	-	-	-	-	-
1	a	-	-	-	-	-	-	-	-	-	-	-	-
2	a	-	-	-	-	-	-	-	-	-	-	-	-
3	c	-	-	-	-	-	-	-	-	-	-	-	-
4	t	-	-	-	-	-	-	-	3	3	3	3	3
5	c	-	-	-	-	-	-	-	3	3	3	3	3
6	c	-	-	-	-	-	-	-	3	3	3	3	3
7	a	-	-	-	-	-	-	-	3	3	3	3	3
8	c	-	-	-	-	-	-	-	3	3	3	3	3
9	t	-	-	-	-	-	-	-	6	6	6	6	6
10	a	-	-	-	-	-	-	-	6	6	6	6	9

(c) $k = 2$

		0	1 a	2 a	3 c	4 c	5 c	6 a	7 c	8 t	9 c	10 t	11 a
0		-	-	-	-	-	-	-	-	-	-	-	-
1	a	-	-	-	-	-	-	-	-	-	-	-	-
2	a	-	-	-	-	-	-	-	-	-	-	-	-
3	c	-	-	-	-	-	-	-	-	-	-	-	-
4	t	-	-	-	-	-	-	-	-	-	-	-	-
5	c	-	-	-	-	-	-	-	-	-	-	-	-
6	c	-	-	-	-	-	-	-	-	-	-	-	-
7	a	-	-	-	-	-	-	-	-	-	-	-	-
8	c	-	-	-	-	-	-	-	-	-	-	-	-
9	t	-	-	-	-	-	-	-	-	-	-	-	-
10	a	-	-	-	-	-	-	-	-	-	-	-	6

(d) $k = 3$

The time and space complexities of the DP algorithm are both O(mnr) time. With the monotonicity of $M_t(i,j,k)$, as given as follows, we can easily show the correctness of Equation 1.

Property 1. *Each of the following holds.*

(1) $M_t(i,j,k_1) \geq M_t(i,j,k_2)$ *if* $k_1 < k_2$.

(2) $M_t(i_1,j_1,k) \leq M_t(i_2,j_2,k)$ *if* $i_1 \leq i_2$ *and* $j_1 \leq j_2$.

(3) $M_t(i_1,j_1,k) < M_t(i_2,j_2,k)$ *if* $i_1 \leq i_2 - t$, $j_1 \leq j_2 - t$, $match_t(i_2, j_2) = 1$ *and* $M_t(i_2,j_2,k) \neq -\infty$.

(4) $M_t(i_1,j_1,k-h) < M_t(i_2,j_2,k)$ *if* $i_1 \leq i_2 - t$, $j_1 \leq j_2 - t$, $match_t(i_2,j_2) = 1$, $M_t(i_2,j_2,k) \neq -\infty$ *and* $h = suf_t(i_2,k)$.

(5) $M_t(i_1,j_1,k-h)+t \leq M_t(i_2,j_2,k)$ *if* $i_1 \leq i_2 - t$, $j_1 \leq j_2 - t$, $match_t(i_2,j_2) = 1$, $M_t(i_2,j_2,k) \neq -\infty$ *and* $h = suf_t(i_2,k)$.

4 The Row-wise Algorithm

In the DP algorithm, we observe that the computation of most cells in $M_t(i,j,k)$ is not needed because the CLCS$_t$ length depends on only the t-match pairs. Accordingly, we propose the row-wise method for solving the CLCS$_t$ problem, which calculates the answer length only at t-match pairs with the row by row manner (increasing values of index i of A).

For implementing the row-wise concept, we formally define $d_{i,s,k}$ as follows.

Definition 5. *For two input sequences A and B with a constraint sequence P, let $d_{i,s,k} = \min\{j | M_t(i,j,k) \geq s\}$. In other words, $d_{i,s,k}$ records the smallest index j of B for $M_t(i,j,k) \geq s$.*

Then, the row-wise algorithm for calculating $d_{i,s,k}$ is given as follows.

$$d_{i,s,k} = \min \begin{cases} 0 & \text{if } s = 0 \text{ and } k = 0, \\ \infty & \text{if } s \geq 1 \text{ or } k \geq 1, \\ d_{i-1,s,k} & \text{if } i \geq 1, \\ j & \text{if } match_t(i,j) = 1 \text{ and } suf_t(i,k) \\ & = h \text{ and } d_{i-t,s-t,k-h} \leq j - t. \end{cases} \tag{2}$$

According to Equation 2, the pseudocode of the row-wise algorithm for solving the CLCS$_t$ problem is presented in Algorithm 1. Table 3 shows an example of the row-wise CLCS$_t$ algorithm with Equation 2. The first two items in Equation 2 are boundry conditions and the third item is for the cases without matching, while the last item is for matching cases.

Theorem 1. *Algorithm 1 solves the CLCS$_t$ problem with $O(r \times \min\{mL+R, R\log L\} + m+n)$ time and $O(r(L+m+n))$ space, where $r = |P| \leq m = |A| \leq n = |B|$, L denotes the CLCS$_t$ length and R denotes the number of t-match pairs between A and B.*

Algorithm 1 Row-wise algorithm for CLCS_t

Input: two sequences $A = a_1a_2\dots a_m$ and $B = b_1b_2\dots b_n$, a constraint sequence $P = p_1p_2\dots p_r$, and a positive integer t for the substring length in the solution, where $r = |P| \le m = |A| \le n = |B|$.
Output: length of $CLCS_t(A, B, P)$

```
1: d_{i,0,0} ← 0 for 0 ≤ i ≤ m
2: d_{0,0,k} ← ∞ for 1 ≤ k ≤ r
3: for k = 0 → r do
4:     d_{0,s,k} ← ∞ for 1 ≤ s ≤ n
5:     for i = 1 → m do
6:         h ← suf_t(i, k)
7:         s ← 0
8:         while d_{i,s−t,k} < ∞ or s ≤ t do
9:             s ← s + t
10:            d_{i,s,k} ← d_{i−1,s,k}
11:            for j = 1 → n do
12:                if match_t(i, j) = 1 then
13:                    if d_{i−t,s−t,k−h} ≤ j − t then
14:                        d_{i,j,k} ← min{d_{i,j,k}, j}
15:                        break
16: return max_{d_{i,s,r} ≤ n, 1 ≤ i ≤ m} {s}
```

5 The Diagonal Algorithm

Our diagonal algorithm for solving the CLCS_t problem is inspired by the diagonal LCS algorithm of Nakatsu *et al.* [4]. The diagonal algorithm also calculates the CLCS_t length only at t-match pairs. But, the calculation of the diagonal algorithm is done along the diagonal direction (not row by row). Thus, the diagonal algorithm is very efficient in dealing with highly similar sequences.

To solve the CLCS_t problem with the diagonal algorithm, we first define $NextMatch_t(i, j)$.

Definition 6. *For two sequences A and B, $NextMatch_t(i, j)$ denotes the smallest j' such that $j' \ge j + t$ and $A_{i-t+1..i} = B_{j'-t+1..j'}$. If there is no such j', then $NextMatch_t(i, j) = \infty$.*

The diagonal algorithm for calculating $d_{i,s,k}$ is given as follows.

$$d_{i,s,k} = \min \begin{cases} 0 & \text{if } s = 0 \text{ and } k = 0, \\ \infty & \text{if } s \ge 1 \text{ or } k \ge 1, \\ d_{i-1,s,k} & \text{if } i \ge 1, \\ NextMatch_t(i, & \text{if } i \ge s \ge t \\ d_{i-t,s-t,k-h}) & \text{and } suf_t(i, k) = h. \end{cases} \tag{3}$$

Accordingly, the pseudocode of the diagonal algorithm for solving the CLCS_t problem is presented in Algorithm 2. Table 4 shows an example of the diagonal CLCS_t algorithm with Equation 3.

Table 3: An example of $d_{i,s,k}$ calculated by the row-wise method with A = `aactccacta`, B = `aacccactcta`, P = `ctt` and $t = 3$.

i	Length s	0	3	6	9	12
0		0	∞	∞	∞	∞
1	a	0	∞	∞	∞	∞
2	a	0	∞	∞	∞	∞
3	c	0	3	∞	∞	∞
4	t	0	3	∞	∞	∞
5	c	0	3	∞	∞	∞
6	c	0	3	∞	∞	∞
7	a	0	3	6	∞	∞
8	c	0	3	6	∞	∞
9	t	0	3	6	∞	∞
10	a	0	3	6	11	∞

(a) $k = 0$

i	Length s	0	3	6	9	12
0		∞	∞	∞	∞	∞
1	a	∞	∞	∞	∞	∞
2	a	∞	∞	∞	∞	∞
3	c	∞	3	∞	∞	∞
4	t	∞	3	∞	∞	∞
5	c	∞	3	∞	∞	∞
6	c	∞	3	∞	∞	∞
7	a	∞	3	6	∞	∞
8	c	∞	3	6	∞	∞
9	t	∞	3	6	∞	∞
10	a	∞	3	6	11	∞

(b) $k = 1$

i	Length s	0	3	6	9	12
0		∞	∞	∞	∞	∞
1	a	∞	∞	∞	∞	∞
2	a	∞	∞	∞	∞	∞
3	c	∞	∞	∞	∞	∞
4	t	∞	8	∞	∞	∞
5	c	∞	8	∞	∞	∞
6	c	∞	8	∞	∞	∞
7	a	∞	8	∞	∞	∞
8	c	∞	8	∞	∞	∞
9	t	∞	8	8	∞	∞
10	a	∞	8	8	11	∞

(c) $k = 2$

i	Length s	0	3	6	9	12
0		∞	∞	∞	∞	∞
1	a	∞	∞	∞	∞	∞
2	a	∞	∞	∞	∞	∞
3	c	∞	∞	∞	∞	∞
4	t	∞	∞	∞	∞	∞
5	c	∞	∞	∞	∞	∞
6	c	∞	∞	∞	∞	∞
7	a	∞	∞	∞	∞	∞
8	c	∞	∞	∞	∞	∞
9	t	∞	∞	∞	∞	∞
10	a	∞	11	11	∞	∞

(d) $k = 3$

Table 4: An example of the diagonal algorithm for the CLCS_t problem with A = `aactccacta`, B = `aacccactcta`, P = `ctt` and $t = 3$.

Length s	**0**	**3**	**6**	**9**
$i = 3$	$d_{0,0,k}$	$d_{3,3,k}$	$d_{6,6,k}$	$d_{9,9,k}$
$k = 0$	0	3	∞	∞
$k = 1$	∞	3	∞	∞
$k = 2$	∞	∞	∞	∞
$k = 3$	∞	∞	∞	∞

(a) Round 1

Length s	**0**	**3**	**6**	**9**
$i = 4$	$d_{1,0,k}$	$d_{4,3,k}$	$d_{7,6,k}$	$d_{10,9,k}$
$k = 0$	0	3	6	11
$k = 1$	∞	3	6	11
$k = 2$	∞	8	∞	11
$k = 3$	∞	∞	∞	∞

(b) Round 2

Length s	**0**	**3**	**6**	**9**
$i = 6$	$d_{3,0,k}$	$d_{6,3,c}$	$d_{9,6,k}$	$d_{12,9,k}$
$k = 0$	0	3	6	11
$k = 1$	∞	3	6	11
$k = 2$	∞	8	8	11
$k = 3$	∞	∞	∞	∞

(c) Round 4

Length s	**0**	**3**	**6**	**9**
$i = 7$	$d_{4,0,k}$	$d_{7,3,k}$	$d_{10,6,k}$	$d_{13,9,k}$
$k = 0$	0	3	6	11
$k = 1$	∞	3	6	11
$k = 2$	∞	8	8	11
$k = 3$	∞	∞	11	∞

(d) Round 5

Theorem 2. *Algorithm 2 solves the $CLCS_t$ problem in $O(rL(m-L)+R)$ time, and its space complexity is $O(rL+R)$.*

6 Experimental Results

In the experiments, we compare the execution time of the DP, row-wise, and diagonal CLCS_t algorithms. The execution time is obtained from the average of ten testing cases for each parameter combination. The pseudorandom datasets used in our experiments are generated with $|A| = 500$, $|B| = 500$, constraint ratio $\frac{|P|}{\min\{|A|,|B|\}} = 0.05$, alphabet size $|\Sigma| \in \{4, 20\}$, substrings length $t \in \{2, 5\}$, and various similarities $SI \in \{0.1, 0.2, \dots, 0.9\}$.

The following algorithms are compared in our

Algorithm 2 The diagonal algorithm for CLCS$_t$

Input: two sequences $A = a_1a_2 \dots a_m$ and $B = b_1b_2 \dots b_n$, a constraint sequence $P = p_1p_2 \dots p_r$, and a positive integer t for the substring length in the solution, where $r = |P| \leq m = |A| \leq n = |B|$.
Output: length of $CLCS_t(A, B, P)$

```
1: L ← 0
2: for i = t → m do                                   ▷ round i − t + 1
3:     d_{i−t,0,0} ← 0
4:     d_{i−t,0,k} ← ∞ for 1 ≤ k ≤ r
5:     i' ← i
6:     while i' ≤ m do
7:         s ← i' − i + t
8:         for k = 0 → r do
9:             h ← suf_t(i', k)
10:            j ← NextMatch_t(i', d_{i'−t,s−t,k−h})
11:            d_{i',s,k} ← min{d_{i'−t,s,k}, j}
12:            if d_{i',s,k} = ∞ then break
13:        end for
14:        if d_{i',s,0} = ∞ then break
15:        if d_{i',s,r} ≤ n then L ← max{d_{i',s,r}, L}
16:        i' ← i' + t
17:    end while
18:    if m − i ≤ L then return L
19: end for
20: return L
```

experiments:

- **DP**: our DP algorithm for the CLCS$_t$ problem;
- **R-Linear**: our row-wise algorithm with the linear scan for the CLCS$_t$ problem.
- **R-Binary**: our row-wise algorithm with the binary search for the CLCS$_t$ problem.
- **R-Hybrid**: our row-wise algorithm choosing the linear or binary strategy based on the current number of t-match pairs for the CLCS$_t$ problem.
- **DIA**: our diagonal algorithm for the CLCS$_t$ problem.

Figures 1, 2 and 3 show the average execution time of the above algorithms. We use the 7-tuple ($|A|$, $|B|$, $|P|$, t, $|\Sigma|$, *similarity*, *algo*) to represent the parameters in the title of each line chart. For example, in Figure 1, $(500, 500, 25, 2, 4, *, *)$ means that $|A| = |B| = 500$, $|P| = 25$, $t = 2$, $|\Sigma| = 4$. Then the first wildcard '*' means that all similarities are tested in the experiment, and the second wildcard '*' indicates that all algorithms are executed.

As the experimental results show, the efficiency of the diagonal algorithm is always superior to the others with various parameter combinations. When $t \times |\Sigma|$ is relatively small (Figure 1), the

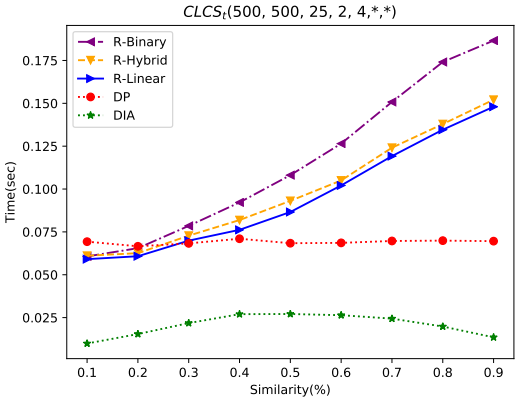

Figure 1: The average execution time for $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 2$ and $|\Sigma| = 4$.

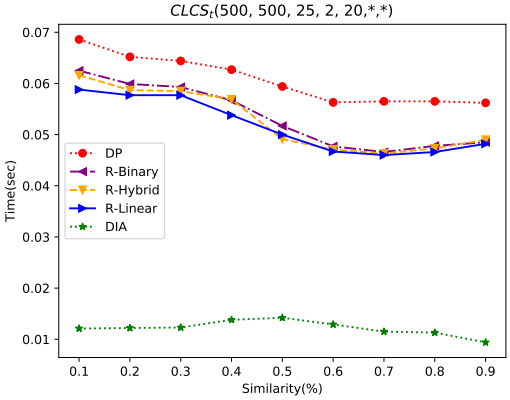

Figure 2: The average execution time for $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 2$ and $|\Sigma| = 20$.

row-wise algorithms are worse than DP if the similarity is high. The reason is that the number of t-match pairs becomes more in such situation. Thus, the row-wise algorithms are slightly better than DP when $t \times |\Sigma|$ is relatively large (Figures 2 and 3), since the number of t-match pairs is relatively small.

7 Conclusion

In this paper, we define the CLCS$_t$ problem, a new variant of the CLCS problem. Then, we propose the DP, row-wise, and diagonal algorithms for solving the problem. As the experimental results show, the diagonal algorithm is the most efficient with various parameter combinations.

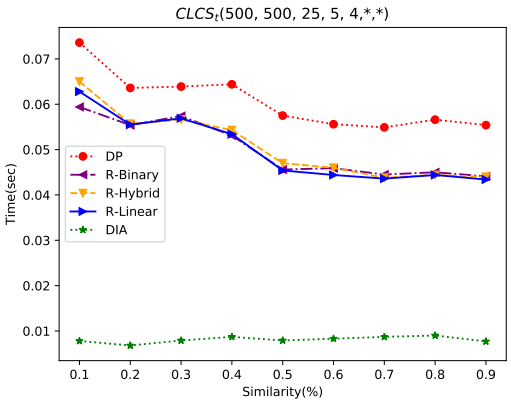

Figure 3: The average execution time for $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 5$ and $|\Sigma| = 4$.

In the future, we may apply our methods to more applications, especially for biosequences measurement. We may also generalize the problem to the CLCS$_{t+}$ problem, in which each common substring is of length t or more (not exactly t). This new variant becomes more complicated since the length of each common substring becomes more flexible.

References

[1] Hsing-Yen Ann, Chang-Biau Yang, Chiou-Ting Tseng, and Chiou-Yi Hor. A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings. *Information Processing Letters*, 108:360–364, 2008.

[2] Kuo-Si Huang, Chang-Biau Yang, Kuo-Tsung Tseng, Yung-Hsing Peng, and Hsing-Yen Ann. Dynamic programming algorithms for the mosaic longest common subsequence problem. *Information Processing Letters*, 102(2-3):99–103, 2007.

[3] Shou-Fu Lo, Kuo-Tsung Tseng, Chang-Biau Yang, and Kuo-Si Huang. A diagonal-based algorithm for the longest common increasing subsequence problem. *Theoretical Computer Science*, 815:69–78, 2020.

[4] Narao Nakatsu, Yahiko Kambayashi, and Shuzo Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982.

[5] Chiou-Ting Tseng, Chang-Biau Yang, and Hsing-Yen Ann. Efficient algorithms for the longest common subsequence problem with sequential substring constraints. *Journal of Complexity*, 29(1):44–52, 2013.

[6] Yin-Te Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88:173–176, 2003.

[7] Francis Y. L. Chin, Alfredo De Santis, Anna Lisa Ferrara, N. L. Ho, and S. K. Kim. A simple algorithm for the constrained sequence problems. *Information Processing Letters*, 90:175–179, 2004.

[8] Siang-Huai Hung, Chang-Biau Yang, and Kuo-Si Huang. A diagonal-based algorithm for the constrained longest common subsequence problem. In Chuan-Yu Chang, Chien-Chou Lin, and Horng-Horng Lin, editors, *New Trends in Computer Technologies and Applications*, volume 1013, pages 425–432. Springer Singapore, 2019.

[9] G. Benson, A. Levy, and B. Riva Shalom. Longest common subsequence in k length substrings. In *Proceedings of the 6th International Conference on Similarity Search and Applications*, volume 8199, pages 257–265, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

[11] Kuo-Tsung Tseng Guo-Fong Huang, Chang-Biau Yang and Kuo-Si Huang. Diagonal algorithms for the longest common subsequence problems with t-length and at least t-length substrings. In *Proceedings of the 37th Workshop on Combinatorial Mathematics and Computation Theory*, pages 119–127, Kaohsiung, Taiwan, 2020.

[12] Hsing-Yen Ann, Chang-Biau Yang, Chiou-Ting Tseng, and Chiou-Yi Hor. Fast algorithms for computing the constrained LCS of run-length encoded strings. *Theoretical Computer Science*, 432:1–9, 2012.

[13] Filip Pavetić, Ivan Katanić, Gustav Matula, Goran Žužić, and Mile Šikić. Fast and simple algorithms for computing both LCSk and LCSk+. *CoRR, abs/1705.07279*, 2018.