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

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*) *between two given sequences can be used as the similarity measurement of them. Several variants of the LCS problem have been proposed in the past. In 2021, we presented and solved the newly defined variant, the* constrained longest common subsequence with t-length substrings ($CLCS_t$) *problem. In this paper, we generalized the above variant, the* constrained longest common subsequence with at least t-length substrings ($CLCS_{t+}$) *problem. This new generalization is the combination of the* constrained longest common subsequence with at least t-length substrings (*CLCS*) *and the LCS with at least t-length substrings* (LCS_{t+}) *problem. We present three algorithms for solving it. Our first algorithm is the dynamic programming, with time complexity of* $O(mnr)$*, where* m*,* n*, and* r *are the lengths of the two target sequences and the constraint sequence, respectively. Then, our second algorithm is the row-wise algorithm, with time complexity of* $O(r \times \min\{mL + R, R \log L\} + m + n)$*, where* L *denotes the answer length, and* R *denotes the number of t-match pairs between the two target sequences. The time complexity of our third diagonal algorithm is* $O((m - L)(rL + R))$*. The experimental results show that the most efficient among these three proposed algorithms is the diagonal algorithm.*

Keywords: longest common subsequence (LCS), constrained LCS, LCS_{t+}, dynamic programming, diagonal

*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).

1 Introduction

In several fields, such as computational biology [13], pattern matching [8], plagiarism detection [4], and voice recognition [9], it is essential to measure the similarity of two or more given sequences. And the most popular measurement used in these fields is the LCS length since the LCS problem is a well-studied problem and many algorithms have been proposed for solving it [1, 6, 10, 11, 15] in the past decades.

The definition of *constrained longest common subsequence* (CLCS) problem is similar to the LCS problem but with an additional parameter, a constraint sequence P. The goal of CLCS is to find the LCS of two target sequences A and B which contains P as a subsequence in the answer. In 2003, Tsai [14] defined the CLCS problem and then proposed a *dynamic programming* (DP) algorithm, with $O(m^2n^2r)$ time and space, where $|A| = m$, $|B| = n$ and $|P| = r$, for solving it. Later in 2004, a DP-based method reducing both time and space complexities to $O(mnr)$ was proposed by Chin *et al.* [5]. In 2018, Hung *et al.* [7] presented a diagonal-based [11] algorithm with $O(rL(m - L))$ time and $O(mr)$ space, where L denotes the CLCS length.

In 2013, Benson *et al.* [3] defined a new variant of LCS, the *LCS with t-length substrings* (LCS_t), which is to find the LCS consisting of common substrings with t-length. We combined the definitions of the LCS_t problem and the CLCS problem, and then in 2021, defined and solved the *constrained longest common subsequence with t-length substrings* ($CLCS_t$) problem [17].

In this paper, again we combine the definitions of the *longest common subsequence problem with at least t-length substrings* (LCS_{t+}) which was proposed by Pavetić *et al.* [12] and *CLCS* problem to a new generalized variant, the *constrained* *longest common subsequence with at least t-length substrings* (CLCS$_{t+}$) problem. We then propose three algorithms to solve it in later sections.

Similar to CLCS$_t$ problem, the CLCS$_{t+}$ problem are given two target sequences A, B and a constraint sequence P. The answer is the LCS of A and B which contains P as a subsequence, and it is composed of common substrings of A and B with at least t-length.

We propose three algorithms for solving the CLCS$_{t+}$ problem. The time complexity of the first DP algorithm is O(mnr), and the second row-wise algorithm is 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 A and B. The third diagonal algorithm is theoretically and practically the most efficient with time complexity of O($(m-L)(rL+R)$).

The organization of the rest of this paper is given as follows. We introduce some preliminaries of the CLCS$_{t+}$ problem in Section 2. Then, we propose three algorithms for solving the CLCS$_{t+}$ problem in Sections 3 through 5. The experimental results are shown in Section 6. Finally, Section 7 concludes this paper.

2 Preliminaries

Definition 1. (CLCS$_{t+}$ problem) *Given two sequences $A=a_1a_2\cdots a_m$ and $B=b_1b_2\cdots b_n$, with a constraint sequence $P=p_1p_2p_3...p_r$ and a constant positive integer t, the CLCS$_{t+}$ problem is to find the LCS of A and B that consists of common substrings of A and B with at least t-length and it contains P as a subsequence.*

Suppose the answer consists of l segments of at least t-length substrings, then the length of the answer is between $l\times t$ and $l\times(2t-1)$. Thus, there exists a common subsequence $C = A_{i_1-t_1+1..i_1}\ A_{i_2-t_2+1..i_2}\ \cdots\ A_{i_l-t_l+1..i_l} = B_{j_1-t_1+1..j_1}\ B_{j_2-t_2+1..j_2}\ \cdots\ B_{j_l-t_l+1..j_l}$, where $i_g \le i_{g+1}-t_g$ and $j_g \le j_{g+1}-t_g$ for $1\le g\le l-1$, and C contains P as a subsequence. Note that each common substring is of length t_g for $1\le g\le l$, where $t\le t_g\le 2t-1$.

3 The Dynamic Programming Algorithm

Our DP method for solving the CLCS$_{t+}$ problem is given in Equation 1.

$$
M_{t+}(i,j,k) = \max\begin{cases} M_{t+}(i-1,j,k) & \text{if } i\ge 1,\\ M_{t+}(i,j-1,k) & \text{if } j\ge 1,\\ M_{t+}(i-t',j-t', k-h)+t' & \text{if } match_{t'}(i,j)=1 \text{ and } suf_{t'}(i,k)=h \text{ for } t\le t'\le 2t-1. \end{cases} \tag{1}
$$

with boundary condition:
for $0\le i\le m$, $0\le j\le n$, and $1\le k\le r$,
$M_{t+}(i,0,0)=M_{t+}(0,j,0)=0$,
$M_{t+}(i,0,k)=M_{t+}(0,j,k)=-\infty$.

Clearly, Equation 1 solves the CLCS$_{t+}$ problem with O($mnrt$) time and O(mnr) space.

For solving the LCS$_{t+}$ problem, Benson *et al.* [2] first proposed a straightforward DP algorithm with O(tmn) time. Then Ueki *et al.* [16] proposed an improved DP algorithm with O(mn) time. Here, we apply the concept of Ueki *et al.* [16] to improving our DP algorithm for CLCS$_{t+}$.

To improve the DP algorithm, we first present the monotonicity of $M_{t+}(i,j,k)$ in the CLCS$_{t+}$ problem.

Property 1. *For the CLCS$_{t+}$ problem with sequences A, B, and a constraint sequence P, we have the following:*

$M_{t+}(i,j,k)\ge M_{t+}(i-\delta,j-\delta,k-h')+\delta$, if $match_{t+\delta}(i,j)=1$, where $0\le\delta<t$ and $h'=suf_\delta(i,k)$.

Lemma 1. *For the CLCS$_{t+}$ problem with sequences A, B, and a constraint sequence P, we have the following:*

$$
M_{t+}(i-t',j-t',k-h')+t' \le \max\begin{cases} M_{t+}(i-t,j-t,k-h)+t,\\ M_{t+}(i-1,j-1,k-h_1)+1,\end{cases}
$$

where $t'\ge t$, (i,j) is a t'-match and $h'=suf_{t'}(i,k)$, $h=suf_t(i,k)$, $h_1=suf_1(i-1,k)$.

The proof of Lemma 1 is omitted here. Based on Lemma 1, we propose the improved DP method for the CLCS$_{t+}$ problem in Equation 2.

$$M_{t+}(i,j,k) = \max\begin{cases} M_{t+}(i-1,j,k) & \text{if } i \geq 1, \\ M_{t+}(i,j-1,k) & \text{if } j \geq 1, \\ M_{t+}(i-t,j-t, k-h)+t & \text{if } match_t(i,j)=1 \text{ and } suf_t(i,k)=h, \\ M_{t+}(i-1,j-1, k)+1 & \text{if } match_{t+1}(i,j)=1 \\ M_{t+}(i-1,j-1, k-1)+1 & \text{if } match_{t+1}(i,j)=1 \text{ and } k \geq 1 \text{ and } a_i = p_k. \end{cases} \tag{2}$$

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

It is clear that Equation 2 solves the CLCS$_{t+}$ problem with O(mnr) time and space.

Table 1 shows an example of Equation 2.

In Table 1a, the procedure is the same as the LCS$_{t+}$ problem, whose answer is `aac cca cta`.

In Table 1b, some cells that do not contain the constraint sequence $p_1 =$ `c` are set to $-\infty$. The result of $M_{t+}(8,7,1)$ is `aac ccac`, while $M_t(8,7,1)$ is `aac cca`, and both contain the constraint subsequence `c`.

In Table 1c, the values of some cells are smaller than the corresponding cells in $k = 1$ since the formers have to satisfy more constraint $P_{1..2} =$ `ct`. Nevertheless, the final result ($k = 2$) is still `aac cca cta`.

In Table 1d, with the whole constraint sequence $P_{1..3} =$ `ctt`, the CLCS$_{3+}$ solution is `act cta` with length 6.

4 The Row-wise Algorithm

Similar to the row-wise CLCS$_t$ algorithm, we use the match pair table to save the computation time and space for the CLCS$_{t+}$ problem. Therefore, we first give the definition of $d'_{i,s,k}$.

Definition 2. For two input sequences A and B, and a constraint sequence P, $d'_{i,s,k} = \min\{j| \, CLCS_{t+}(A_{1..i}, B_{1..j}, P_{1..k}) \geq s\}$. In other words, $d'_{i,s,k}$ records the lowest index j of B such that CLCS$_{t+}(A_{1..i}, B_{1..j}, P_{1..k}) \geq s$.

By Definition 2, the row-wise algorithm for the CLCS$_{t+}$ problem by updating $d'_{i,s,k}$ is given in Equation 3.

$$d'_{i,s,k} = \min\begin{cases} 0 & \text{if } s=0 \text{ and } k=0, \\ \infty & \text{if } s \geq 1 \text{ or } k \geq 1, \\ d'_{i-1,s,k} & \text{if } i \geq 1, \\ j & \text{if } match_t(i,j)=1 \text{ and } suf_t(i,k)=h \text{ and } d'_{i-t,s-t,k-h} \leq j-t, \\ j' & \text{if } match_{t+1}(i,j')=1 \text{ and } a_i = b_{j'} \neq p_k \text{ and } d'_{i-1,s-1,k} \leq j'-1, \\ j' & \text{if } match_{t+1}(i,j')=1 \text{ and } a_i = b_{j'} = p_k \text{ and } d'_{i-1,s-1,k-1} \leq j'-1. \end{cases} \tag{3}$$

With Equation 3, the row-wise algorithm for solving the CLCS$_{t+}$ problem is presented in Algorithm 1.

Table 2 shows an example of the row-wise CLCS$_{t+}$ algorithm with Equation 3.

In Table 2a, only t'-match pairs, $t' \geq t = 3$, are calculated for the CLCS$_{3+}$ length. That is, $d'_{i,s,0}$ is calculated to record the minimum column index of string B that the CLCS$_{3+}$ length is greater than or equal to s in row i (index of string A) for $k = 0$.

For example, to calculate $d'_{8,7,0}$ in row 8, we find there is a 3-match at $(8,7)$, but $d'_{8-t,7-t,0} = d'_{5,4,0} = 9 > 7-3 = 4$. So $(8,7)$ cannot be extended from the 3-match at $(5,9)$ in row 5. On the other hand, the 3-match at $(7,6)$ in row 7 can be extended to $(8,7)$ to become a 4-match, and $d'_{8-1,7-1,0} = d'_{7,6,0} = 6 \leq 7-1 = 6$. So we get $d'_{8,7,0} = 7$.

Table 2b is the same as Table 2a. That is, the first constraint character $p_1 =$ `c` is contained in the solution.

In Table 2c, some cells are set to ∞ since CLCS$_{t+}$ of solutions in such rows do not contain the constraint $P_{1..2} =$ `ct`. For example, to calculate $d'_{3,3,2}$ in row 3, the t-match `acc` at $(3,3)$ contains no common suffix of constraint $P_{1..2} =$ `ct`, so $suf_t(3,2) = 0$. Then $d'_{3-t,3-t,2-0} = d'_{0,0,2} = \infty > 3-3 = 0$, so we obtain $d'_{3,3,2} = \infty$.

As another example, to calculate $d'_{4,3,2}$ in row 4, the t-match `act` at $(4,8)$ contains common suffix `ct` with constraint `ct`, so $suf_t(4,2) = 2$. As $d'_{4-t,3-t,2-2} = d'_{1,0,0} = 0 \leq 8-3 = 5$, we get $d'_{4,3,2} = 8$.

In Table 2d, only one t-match `cta` at $(10,11)$ can update $d'_{10,6,3}$ in row 10 with constraint `ctt`. As $suf_t(10,3) = 1$ and $d'_{10-t,6-t,3-1} = d'_{7,3,2} = 8 \leq 11-3 = 8$. We obtain $d'_{10,6,3} = 11$, meaning $M_{t+}(10,11,3) = 6$.

Table 1: An example of the DP algorithm for CLCS_{3+} with A = aactccacta, B = aacccactcta, P = ctt and $t = 3$.

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

(a) $k = 0$

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

(b) $k = 1$

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

(c) $k = 2$

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

(d) $k = 3$

Table 2: An example for constructing $d'_{i,s,k}$ in the row-wise CLCS_{t+} algorithm with A = aactccacta, B = aacccactcta, P = ctt and $t = 3$.

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

(a) $k = 0$

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

(b) $k = 1$

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

(c) $k = 2$

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

(d) $k = 3$

Algorithm 1 Row-wise CLCS_{t+} algorithm

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

Output: length of $CLCS_{t+}(A, B, P)$

$d'_{i,0,0} \leftarrow 0$ for $0 \leq i \leq m$
$d'_{0,0,k} \leftarrow \infty$ for $1 \leq k \leq r$
for $k = 0 \rightarrow r$ **do**
  $d'_{0,s,k} \leftarrow \infty$ for $1 \leq s \leq n$
  for $i = 1 \rightarrow m$ **do**
    $h \leftarrow suf_t(i, k)$
    $s \leftarrow t$
    while $d'_{i,s-1,k} < \infty$ or $s \leq t$ **do**
      $y \leftarrow \infty$
      for $j = 1 \rightarrow n$ **do**
        if $match_t(i, j) = 1$ and $d'_{i-t,s-t,k-h} \leq j - t$ **then**
          $y \leftarrow \min\{y, j\}$ ▷ extend length t from a t-match
        else if $match_{t+1}(i, j) = 1$ **then** ▷ extend length 1 from a $(t + 1)$-match
          if $d'_{i-1,s-1,k} \leq j - 1$ **then**
            $y \leftarrow \min\{y, j\}$
          else if $a_i = p_k$ and $d'_{i-1,s-1,k-1} \leq j - 1$ **then**
            $y \leftarrow \min\{y, j\}$
      end for
      $d'_{i,s,k} \leftarrow \min\{d'_{i-1,s,k}, y\}$
      $s \leftarrow s + 1$
  end for
end for
return $\max\limits_{d'_{i,s,r} \leq m, 1 \leq i \leq m} \{s\}$

Theorem 1. *Algorithm 1 solves the $CLCS_{t+}$ problem in $O(r \times \min\{mL + R, R \log L\} + m + n)$ time. The space complexity is $O(r(L + n + R))$.*

5 The Diagonal Algorithm

To propose the DP formula of the diagonal CLCS_{t+} algorithm, we first define the next match $NextMatch_t(i, j)$ and the extended positions $Extend[i]$ as follows.

Definition 3. *For two sequences A and B, $NextMatch_t(i, j)$ denotes the smallest j' such that $j' \geq j + t$ and $A_{i-t+1..i} = B_{j'-t+1..j'}$. If there is no such j', then $NextMatch_t(i, j) = \infty$.*

Definition 4. *[7] For sequences A and B, $Extend[i]$ is a set consisting of position indexes of B which can be extended with one character from t-match pairs in $A_{i-t..i-1}$. That is, $Extend[i] = \{j | A_{i-t..i} = B_{j-t..j}$ and $t + 1 \leq j \leq n\}$.*

$$d'_{i,s,k} = \min \left\{ \begin{array}{ll} 0 & \text{if } s = 0 \text{ and } k = 0, \\ \hline \infty & \text{if } s \geq 1 \text{ or } k \geq 0, \\ \hline d'_{i-1,s,k} & \text{if } i \geq 1, \\ \hline NextMatch_t & \text{if } i \geq s \geq t \\ (i, d'_{i-t,s-t,k-h}) & \text{and } suf_t(i,k) = h, \\ \hline j & \text{if } i \geq t+1 \text{ and} \\ & j \in Extend[i] \\ & \text{and } a_i \neq p_k \text{ and} \\ & j \geq d'_{i-1,s-1,k} + 1, \\ \hline j' & \text{if } i \geq t+1 \text{ and} \\ & j' \in Extend[i] \\ & \text{and } a_i = p_k \text{ and} \\ & j' \geq d'_{i-1,s-1,k-1} + 1. \end{array} \right. \quad (4)$$

By Lemma 1, Definitions 3 and 4, the diagonal algorithm for solving the CLCS_{t+} problem is given in Equation 4.

Accordingly, the pseudocode of the diagonal algorithm for solving the CLCS_{t+} problem is presented in Algorithm 2.

Tables 3a to 3e show an example of the diagonal

Algorithm 2 The diagonal algorithm for CLCS_{t+}

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

Output: length of $CLCS_{t+}(A, B, P)$

$L \leftarrow 0$, $L_0 \leftarrow 0$ ▷ L: answer; L_0: length for $k = 0$
for $i = t \to m$ **do** ▷ round $i - t + 1$
  $d'_{i-t,0,0} \leftarrow 0$, $d'_{i-t,0,k} \leftarrow \infty$ for $1 \le k \le r$
  $i' \leftarrow i$
  for $i' = i \to m$ **do**
    $s \leftarrow i' - i + t$
    for $k = 0 \to r$ **do**
      $h \leftarrow suf_t(i', k)$
      $j \leftarrow NextMatch_t(i', d'_{i'-t,s-t,k-h})$ ▷ extend length t from a t-match
      if $s \ge t + 1$ **then** ▷ extend length 1 from a $(t + 1)$-match
        if $a_{i'} \ne p_k$ **then**
          $j \leftarrow \min\{j, j'\}$ for $j' \in \mathit{Extend}[i']$ and $j' \ge d'_{i'-1,s-1,k} + 1$
        else
          $j \leftarrow \min\{j, j'\}$ for $j' \in \mathit{Extend}[i']$ and $j' \ge d'_{i'-1,s-1,k-1} + 1$
      $d_{i',s,k} \leftarrow \min\{d_{i'-1,s,k}, j\}$
      if $d'_{i',s,k} = \infty$ **then** break
    end for
    if $d'_{i',s,0} \le n$ **then** $L_0 \leftarrow \max\{d'_{i',s,0}, L_0\}$
    if $d'_{i',s,r} \le n$ **then** $L \leftarrow \max\{d'_{i',s,r}, L\}$
    if $L_0 + t < i' - i$ **then** break ▷ L_0 controls the early break
  end for
  if $m - i \le L$ **then** return L
end for
return L

CLCS_{t+} algorithm with Equation 4.

For round 1 ($i = 3$) in Table 3a, we first calculate $d'_{3,3,0}$. As $NextMatch_t$ $(3, d'_{3-3,3-3,0})$ $= NextMatch_t(3, 0) = 3$, we get a 3-match at $(3, 3)$. So $d'_{3,3,0} = 3$. Then we calculate $d'_{3,3,1}$. $A_{1..3} =$ `aac` contain the constraint $p_1 =$ `c`, so $suf_t(3, 1) = 1$. As $NextMatch_t$ $(3, d'_{3-3,3-3,1-1}) = NextMatch_t(3, 0) = 3$, we get $d'_{3,3,1} = 3$. Because we cannot find the 3-match of $A_{4..6} =$ `tcc` in B after index 6, and we cannot extend one more character from $A_{1..3} =$ `aac`, round 1 stops.

For round 2 ($i = 4$) in Table 3b, we start by calculating $d'_{4,3,k}$. To calculate $d'_{4,3,2}$, $A_{2..4} =$ `act` contains the suffix `ct` of the constraint $P_{1..2} =$ `ct`, so $suf_t(4, 2) = 2$. We have $NextMatch_t(4, d'_{4-3,3-3,2-2}) = NextMatch_t(4, 0) = 8$. Then, $d'_{4,3,2} = \min\{d'_{3,3,2}, 8\} = 8$.

To calculate $d'_{5,4,0}$, we get $NextMatch_t(5, d'_{5-3,4-3,0}) = NextMatch_t(4, \infty) = \infty$, meaning that $d'_{5,4,0}$ cannot get a good value from a t-match. On the other hand, we find $9 \in \mathit{Extend}[5]$ as $A_{2..5} = B_{6..9} =$ `actc`, and $9 \ge d'_{5-1,4-1,0} + 1 = 4$. So $d'_{5,4,0} = \min\{\infty, 9\} = 9$. We can also get $d'_{5,4,1} = 9$ as $suf_t(5, 1) = 1$ and $d'_{5,4,2=9}$ as $suf_t(5, 2) = 2$.

$d'_{6,5,0} = \infty$ because there is no t-match in B and $\mathit{Extend}[6]$ is empty. $d'_{7,6,0} = 6$ as $NextMatch_t(7, d'_{4,3,0} = 3) = 6$. $d'_{8,7,0} = 8$ as $8 \in \mathit{Extend}[9]$. As $NextMatch_t(10,\ d'_{7,6,0} = 6) = 11$, we have $d'_{10,9,0} = 11$.

Rounds 3 and 4 can be computed similarly.

For round 5 ($i = 7$) in Table 3e, to calculate $d'_{10,6,3}$, we have that $A_{8..10} =$ `cta` contains the common suffix `t` of the constraint $P_{1..3} =$ `ctt`, and $NextMatch_t(10,\ d'_{10-3,6-3,3-1}) = NextMatch_t(10, 8) = 11$. So we get $d'_{10,6,3} = 11$, which is the only one satisfying the constraint P with $k = 3$. Thus, the final result is $\text{CLCS}_{3+}(A, B, P) = 6$.

Table 3: An example of the diagonal CLCS_{t+} algorithm with A = aactccacta, B = aacccactcta, P = ctt and $t = 3$.

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$i = 3$	$d'_{0,0,k}$	$d'_{1,1,k}$	$d'_{2,2,k}$	$d'_{3,3,k}$	$d'_{4,4,k}$	$d'_{5,5,k}$	$d'_{6,6,k}$	$d'_{7,7,k}$	$d'_{8,8,k}$	$d'_{9,9,k}$
$k = 0$	0	∞	∞	3	∞	∞	∞	∞	∞	∞
$k = 1$	∞	∞	∞	3	∞	∞	∞	∞	∞	∞
$k = 2$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$k = 3$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

(a) Round 1

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$i = 3$	$d'_{1,0,k}$	$d'_{2,1,k}$	$d'_{3,2,k}$	$d'_{4,3,k}$	$d'_{5,4,k}$	$d'_{6,5,k}$	$d'_{7,6,k}$	$d'_{8,7,k}$	$d'_{9,8,k}$	$d'_{10,9,k}$
$k = 0$	0	∞	∞	3	9	∞	6	7	8	11
$k = 1$	∞	∞	∞	3	9	∞	6	7	8	11
$k = 2$	∞	∞	∞	8	9	∞	∞	∞	8	11
$k = 3$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

(b) Round 2

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$i = 3$	$d'_{2,0,k}$	$d'_{3,1,k}$	$d'_{4,2,k}$	$d'_{5,3,k}$	$d'_{6,4,k}$	$d'_{7,5,k}$	$d'_{8,6,k}$	$d'_{9,7,k}$	$d'_{10,8,k}$	$d'_{10,9,k}$
$k = 0$	0	∞	∞	3	9	∞	6	7	8	11
$k = 1$	∞	∞	∞	3	9	∞	6	7	8	11
$k = 2$	∞	∞	∞	8	9	∞	∞	8	8	11
$k = 3$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

(c) Round 3

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$i = 3$	$d'_{3,0,k}$	$d'_{4,1,k}$	$d'_{5,2,k}$	$d'_{6,3,k}$	$d'_{7,4,k}$	$d'_{8,5,k}$	$d'_{9,6,k}$	$d'_{10,7,k}$	$d'_{10,8,k}$	$d'_{10,9,k}$
$k = 0$	0	∞	∞	3	9	∞	6	7	8	11
$k = 1$	∞	∞	∞	3	9	∞	6	7	8	11
$k = 2$	∞	∞	∞	8	9	∞	8	8	8	11
$k = 3$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

(d) Round 4

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$i = 3$	$d'_{4,0,k}$	$d'_{5,1,k}$	$d'_{6,2,k}$	$d'_{7,3,k}$	$d'_{8,4,k}$	$d'_{9,5,k}$	$d'_{10,6,k}$	$d'_{10,7,k}$	$d'_{10,8,k}$	$d'_{10,9,k}$
$k = 0$	0	∞	∞	3	7	8	6	7	8	11
$k = 1$	∞	∞	∞	3	7	8	6	7	8	11
$k = 2$	∞	∞	∞	8	9	8	8	8	8	11
$k = 3$	∞	∞	∞	∞	∞	∞	11	∞	∞	∞

(e) Round 5

Theorem 2. *Algorithm 2 solves the $CLCS_{t+}$ problem in $O((m-L)(rL+R))$ time. The space complexity is $O(rL+R)$.*

6 Experimental Results

Unlike the CLCS_t problem, the CLCS_{t+} algorithms need to consider each match pair's extension, which spends more time handling match pairs and makes CLCS_{t+} algorithms more complicated than CLCS_t.

The pseudorandom datasets used in our experiments are generated with $|A| = B \in \{500, 1000\}$, constraint ratio $\frac{|P|}{\min\{|A|,|B|\}} \in \{0.05, 0.2\}$, alphabet size $|\Sigma| \in \{4, 20\}$, substrings length $t \in \{2, 5\}$, and various similarities $SI \in \{0.1, 0.2, \ldots, 0.9\}$. The algorithms compared in our experiments are shown as follows:

- **DP+**: the improved DP algorithm.
- **R-Linear+**: the row-wise algorithm with linear scan. CLCS_{t+} problem.
- **R-Binary+**: the row-wise algorithm with binary search. CLCS_{t+} problem.
- **R-Hybrid+**: the row-wise algorithm by choosing linear or binary strategy based on the current number of match pairs.
- **DIA+**: the diagonal algorithm.

Figures 1, 2, 3, 4, and 5 show the execution time under the above algorithms with various parameters ($|A|$, $|B|$, $|P|$, t, $|\Sigma|$, *similarity*, *algo*).

In Figure 1, the diagonal method has the lowest time cost, and the row-wise algorithm spends more time than DP under high similarities. In addition, though the complexity of checking the extension of match pairs of the row-wise is lower than the DP+, the row-wise needs to scan and update the columns, which costs more time significantly when the similarities increase.

In Figure 2, the alphabet set size increases. Thus, the number of match pairs decreases under the same similarity as Figure 1, decreasing the row-wise method's time cost. Due to a decrease in match pairs, there is a significant drop in time of the row-wise algorithm than Figure 1. Moreover, the time required for the diagonal algorithm also decreases significantly.

As the substring length t increases, it decreases the number of match pairs and the answer length

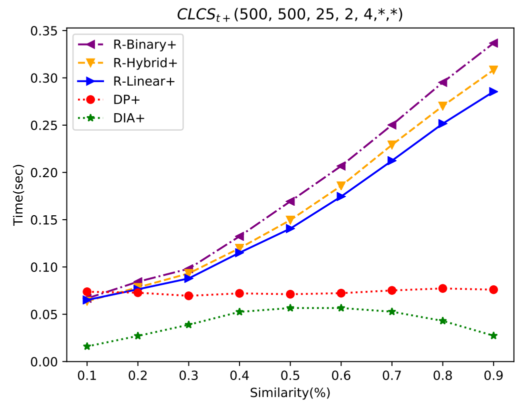

Figure 1: The average execution time for the CLCS_{t+} algorithms with $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 2$ and $|\Sigma| = 4$.

in Figure 3, so the row-wise and the diagonal algorithms spend less time than those in Figure 1. Furthermore, the execution time of the row-wise algorithm is less than the DP method due to less time spent scanning match extensions and column-match arrays, and the diagonal method still spends the least time.

In Figure 4, the size of input sequences ($|A| \times |B| \times |P|$) increases from $25 \times 500 \times 500$ to $50 \times 1000 \times 1000$. Hence, the execution time of all algorithms increases rapidly, especially for the row-wise method. On the other hand, the diagonal algorithm spends less time than other methods and also has much less space cost ($\text{O}(rL+R)$) than the DP method ($\text{O}(mnr)$) with similar time complexity.

Figure 5 increases the constraint length $|P|$ from 25 to 100. As the constraint length increases, there is a rapid growth of the execution time for the row-wise algorithm, which surpasses the DP and the diagonal method. Moreover, the diagonal method has better performance than other methods, significantly when similarities increase.

The above figures show the diagonal algorithm is the most efficient. Furthermore, The DP method has better time efficiency when $|\Sigma|$ and t are small, such as $|\Sigma| = 4$, $t = 2$. On the other hand, the DP needs much more space cost ($\text{O}(mnr)$) than the row-wise ($\text{O}(r(L+n+R))$) and the diagonal ($\text{O}(rL+R)$) methods. So the DP method is suitable when $|A|$, $|B|$, and $|P|$ are short and the memory resource is enough to experiment.

The row-wise may not be efficient for high con-

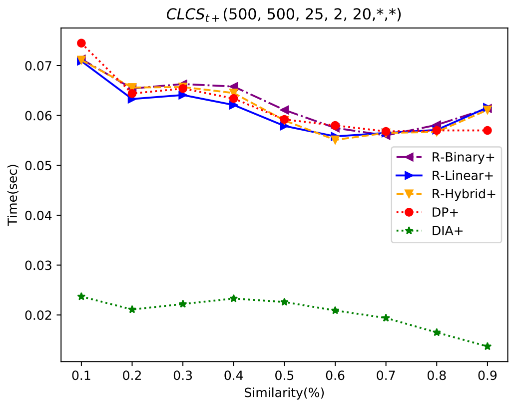

Figure 2: The average execution time for the CLCS_{t+} algorithms with $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 2$ and $|\Sigma| = 20$.

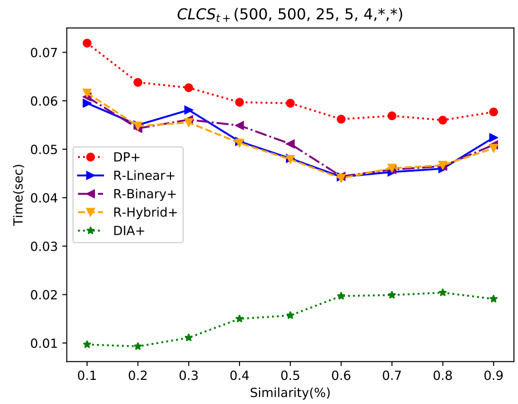

Figure 3: The average execution time for the CLCS_{t+} algorithms with $|A| = 500$, $|B| = 500$, $|P| = 25$, $t = 5$ and $|\Sigma| = 4$.

straint length ratios and low similarities. But the row-wise algorithm is still a good choice in the case of large values of $|\Sigma|$ (e.g., $|\Sigma| = 20$) or t (e.g., $t = 5$).

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 it. Experimental results show that the diagonal algorithm is the most powerful with various parameter combinations. On the other hand, the row-wise methods are efficient for short sequences and large alphabet set sizes. Finally, the DP algorithms are efficient for long sequences but spend much more space than other algorithms.

In the future, we may apply our methods to more applications, especially for biosequences measurement. The proposed CLCS_{t+} problem, in which each common substring is of length t or more (not exactly t) becomes more flexible and meaningful in biosequences measurement.

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] G. Benson, A. Levy, S. Maimoni, D. Noifeld, and B. R. Shalom, "LCSk a refined similarity measure," *Theoretical Computer Science*, Vol. 368, pp. 11–26, 2016.

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

[4] R. C. Campos and F. Z. Martínez, "Batch source-code plagiarism detection using an algorithm for the bounded longest common subsequence problem," *Proceedings of the 9th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, Mexico City, Mexico, pp. 1–4, IEEE, 2012.

[5] F. Y. L. Chin, A. D. Santis, A. L. Ferrara, N. L. Ho, and S. K. Kim, "A simple algorithm for the constrained sequence problems," *Information Processing Letters*, Vol. 90, pp. 175–179, 2004.

[6] K.-S. Huang, C.-B. Yang, K.-T. Tseng, Y.-H. Peng, and H.-Y. Ann, "Dynamic programming algorithms for the mosaic longest common subsequence problem," *Information Processing Letters*, Vol. 102, No. 2-3, pp. 99–103, 2007.

[7] S.-H. Hung, C.-B. Yang, and K.-S. Huang, "A

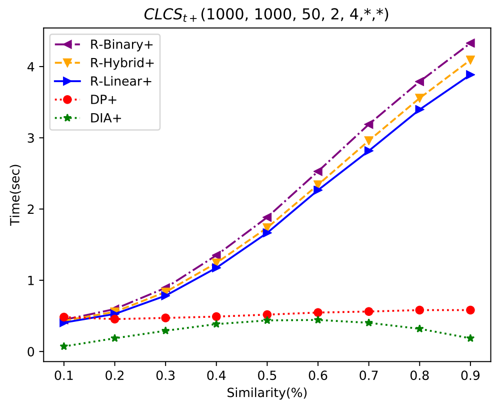

Figure 4: The average execution time for the CLCS_{t+} algorithms with $|A| = 1000$, $|B| = 1000$, $|P| = 50$, $t = 2$ and $|\Sigma| = 4$.

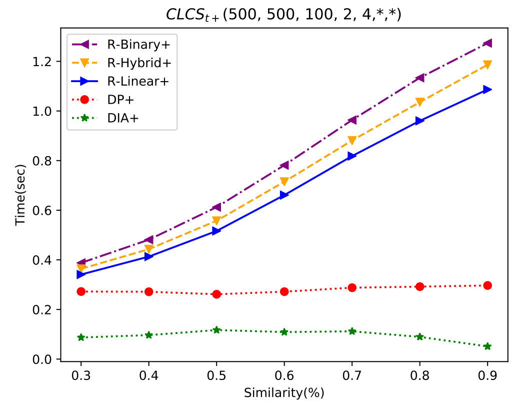

Figure 5: The average execution time for the CLCS_{t+} algorithms with $|A| = 500$, $|B| = 500$, $|P| = 100$, $t = 2$ and $|\Sigma| = 4$.

diagonal-based algorithm for the constrained longest common subsequence problem," *New Trends in Computer Technologies and Applications* (C.-Y. Chang, C.-C. Lin, and H.-H. Lin, eds.), Vol. 1013, pp. 425–432, Springer Singapore, 2019.

[8] M. Jalali, N. Mustapha, M. N. Sulaiman, and A. Mamat, "A recommender system approach for classifying user navigation patterns using longest common subsequence algorithm," *American Journal of Scientific Research*, Vol. 4, pp. 17–27, 2009.

[9] J. B. Kruskal, "An overview of sequence comparison: Time warps, string edits, and macromolecules," *SIAM Review*, Vol. 25, No. 2, pp. 201–237, 1983.

[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] 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.

[12] F. Pavetić, G. ŽŽužić, and M. Šikić, "*LCSk*++: Practical similarity metric for long strings," *CoRR, abs/1407.2407*, 2014.

[13] C. Y. Tang, C. L. Lu, M. D.-T. Chang, Y.-T. Tsai, Y.-J. Sun, K.-M. Chao, J.-M. Chang, Y.-H. Chiou, C.-M. Wu, H.-T. Chang, and W.-I. Chou, "Constrained multiple sequence alignment tool development and its application to RNase family alignment," *Journal of Bioinformatics and Computational Biology*, Vol. 1, No. 02, pp. 267–287, 2003.

[14] Y.-T. Tsai, "The constrained longest common subsequence problem," *Information Processing Letters*, Vol. 88, pp. 173–176, 2003.

[15] C.-T. Tseng, C.-B. Yang, and H.-Y. Ann, "Efficient algorithms for the longest common subsequence problem with sequential substring constraints," *Journal of Complexity*, Vol. 29(1), pp. 44–52, 2013.

[16] 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," *Proceedings of the 43rd International Conference on Current Trends in Theory and Practice of Computer Science*, Vol. 10139, Limerick, Ireland, pp. 364–374, 2017.

[17] K. Zhu, C.-B. Yang, and K.-T. Tseng, "Algorithms for the constrained longest common subsequence problem with t-length substrings," *Proceedings of of the National Computer Symposium 2021 (NCS 2021)*, Taichung, Taiwan, pp. 1348–1353, 2021.