ISBN/978-957-437-895-1(PDF)

A Fast Algorithm for the Longest Common Palindromic Subsequence Problem[*]

Ting-Wei Liang[a], Chang-Biau Yang[a][†]and Kuo-Si Huang[b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[†]E-mail address: cbyang@cse.nsysu.edu.tw

[b]Department of Business Computing
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

Given two input sequences A and B, the LCPS problem is to find the common subsequence of A and B such that the answer is a palindrome and its length is the maximum. In 2014, Chowdhury et al. first defined the LCPS problem, and proposed a dynamic programming algorithm with time complexity $O(m^2n^2)$, where m and n denote the lengths of A and B, respectively. In this paper, we propose a diagonal algorithm, inspired by the diagonal LCS algorithm proposed by Nakatsu et al. to solve the LCPS problem with $O(L(m-L)R\log n)$ time and $O(RL)$ space, where R denotes the number of match pairs, and L denotes the LCPS length of A and B. As experimental results show, our algorithm is faster than the previous algorithms practically.

1 Introduction

The *longest common subsequence* (LCS) problem has been studied extensively for years in stringology. It can be used to compare the similarity between two strings. The LCS problem has been widely applied to bioinformatics, string comparison, speech recognition, and many other fields [2, 4, 5].

A *palindrome* is a sequence which is the same by reading from either its forward or backward direction, that is, $A = a_1a_2\cdots a_m = a_ma_{m-1}\cdots a_1$. For example, `abfba` and `abffba` are palindromes. In bioinformatics, palindrome sequences appear extensively in genomes. Palindrome sequences can be found in the DNA of plasmid, virus and bacteria, and they also present in cancer cells. In addition, many DNA binding sites of proteins, transcription factors and terminators are also palindrome sequences [9, 11, 15]. Thus, some researchers paid attention to palindrome, such as counting distinct palindromic strings in a sequence [10] and the palindromic length problem (factorizing a sequence into palindromes) [1, 8].

Given two sequences $A = a_1a_2\cdots a_m$ and $B = b_1b_2\cdots b_n$, the *longest common palindromic subsequence* (LCPS) problem is to find the common palindromic subsequence of A and B with the maximal length. In other words, the found answer must be a palindrome. For example, suppose $A =$ `cbccbaabb` and $B =$ `bbccabbca`. Then we have LCS$(A, B) =$ `bccabb` and LCPS$(A, B) =$ `bbabb`.

The LCPS problem was first proposed by Chowdhury *et al.* [7] in 2014. They presented a *dynamic programming* (DP) algorithm with $O(m^2n^2)$ time and space, and an algorithm with $O(R^2\log^2 n\log\log n)$ time and $O(R^2)$ space by mapping the problem to computational geometry, where R is the number of total match pairs between A and B. In 2018, Inenaga and Hyyrö [12] presented a DP algorithm with $O(n+R^2|\Sigma|)$ time and $O(R^2)$ space, and proved that the *four strings longest common palindromic subsequence* (4-LCS) problem can be reduced to the LCPS problem. In the same year, Bae and Lee [3] claimed that they presented a DP algorithm with $O(n+R^2)$ time and $O(R^2)$ space. However, we find that the algorithm of Bae and Lee is not correct. Some of counterexamples have been presented in the master's thesis of Liang [13]. The time complexities and the space

[*]This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 108-2221-E-110-031.
[†]Corresponding author.

Table 1: The time complexities and space complexities of the LCPS algorithms. A and B are the two input sequences; $|A| = m$, $|B| = n$ and $m \leq n$; Σ: alphabet set; R: number of total match pairs between A and B; L: length of LCPS(A, B).

Authors	Year	Time complexity	Space complexity	Note
Chowdhury *et al.* [7]	2014	$O(m^2n^2)$ or $O(R^2 \log^2 n \log\log n)$	$O(m^2n^2)$ or $O(R^2)$	DP
Inenaga and Hyyrö [12]	2018	$O(n + R^2\|\Sigma\|)$	$O(R^2)$	Match pair, Rectangle
Bae and Lee [3]	2018	$O(n + R^2)$	$O(R^2)$	Incorrect, Match pair, Dominant contour
This paper	2020	$O(L(m - L)R \log n)$	$O(RL)$	Diagonal, Match pair, Domination

complexities of the previous LCPS algorithms are summarized in Table 1. The definition and NP-hardness proof of the 2-dimensional LCS problem, another variant of the LCS problem, was proposed by Chan *et al.* [6].

The organization of this paper is as follows. Section 2 presents some of the preliminaries for this paper. In Section 3, we propose the diagonal algorithm for solving the LCPS problem with $O(L(m-L)R \log n)$ time and $O(RL)$ space, where R denotes the number of match pairs between input sequences A and B, and L denotes the LCPS length. In Section 4, we implement our algorithm and some previous algorithms. Then, as the experimental results on pseudorandom datasets, our algorithm is superior to the previous algorithms practically. Finally, the conclusion is given in Section 5.

2 Preliminaries

A sequence (string) $A = a_1a_2a_3 \cdots a_m$ is composed of characters over a finite alphabet Σ. $\bar{A}$ denotes the reverse sequence of A, where $\bar{A} = a_ma_{m-1} \cdots a_1$. $A_{i..j}$ represents the substring of A from indexes i to index j. $A_{i..j} = \emptyset$ if $j < i$.

2.1 Multi-dimensional Maxima Finding by Kung *et al.* [16]

In computational geometry, a point p in a set S of n points is said to be *maximal* or *non-dominated* if there is no other point $q \in S$ whose coordinate values are all greater than or equal to the corresponding coordinate values of p. The set of all the maximal points in S is called the *maxima set* of S, and the *maxima finding problem* is to discard the dominated points and get the maxima set.

In 1975, Kung *et al.* [16] proposed the algorithms for the d-dimensional maxima finding problem, where $d = 2$ or 3, and $d \geq 4$. They showed that the algorithm can finish in $O(n \log n)$ time when $d = 2$ or 3, or $O(n(\log n)^{d-2})$ time when $d \geq 4$. Here, we introduce their algorithm with $O(n \log n)$ time for $d = 3$. Suppose that the input set S consists of n points $s_1, s_2 \cdots s_n$ with their 3-dimensional coordinates (values of x, y and z). Let ${s_i}^*$ denote the projection point of s_i onto the yz plane. M stores the 2-dimensional maxima set of the ${s_i}^*$. For the 3-dimensional maxima finding problem can be solved in $O(n \log n)$ time with the following steps.

Step 1: Sort s_i in S by the x-coordinate, so that $x(s_1) \geq x(s_2) \geq \cdots \geq x(s_n)$

Step 2: Set $i \leftarrow 1$ and $M \leftarrow \phi$

Step 3: If ${s_i}^*$ is maximal in M, $M \leftarrow \textsc{maxima}(M \cup \{{s_i}^*\})$, and output s_i as one of the maximal points.

Step 4: If $i \neq n$, then $i \leftarrow i+1$ and go to Step 3.

Kung *et al.* [16] maintain the set M by an AVL tree. The time complexity of their algorithm is $O(n \log n)$.

2.2 The LCPS Algorithm by Chowdhury *et al.* [7]

In 2014, Chowdhury *et al.* [7] first defined the *longest common palindromic subsequence* (LCPS) problem. Given two sequences A and B, and let $D((p, q), (r, s))$ denote the length of LCPS$(A_{p..q}, B_{r..s})$, where $1 \leq p \leq q \leq m = |A|$ and $1 \leq r \leq s \leq n = |B|$. They proposed the dynamic pro-

gramming algorithm for solving the LCPS problem, as shown in Equation (1), with $O(m^2n^2)$ time.
$D((p,q),(r,s)) =$

$$\begin{cases} 0 & \text{if } p > q \text{ or } r > s; \\ 1 & \text{if } ((p = q \text{ and } r \leq s) \\ & \text{or } (p \leq q \text{ and } r = s)), \\ & \text{and } a_p = a_q = b_r = b_s; \\ 2 + D((p+1, q-1), & \text{if } p < q,\ r < s, \\ \quad (r+1, s-1)) & \text{and } a_p = a_q = b_r = b_s; \\ \max \begin{cases} D((p+1,q),(r,s)) \\ D((p,q),(r+1,s)) \\ D((p,q-1),(r,s)) \\ D((p,q),(r,s-1)) \end{cases} & \text{otherwise.} \end{cases} \tag{1}$$

3 Our Diagonal Algorithm

In this paper, we propose an algorithm for solving the LCPS problem with $O(L(m-L)R\log n)$ time and $O(RL)$ space, where $|\Sigma|$ denotes the alphabet size; m, n, and L denote the lengths of input sequences A, B, and LCPS(A, B), respectively; R denotes the number of match pairs between A and B. Our algorithm is based on the diagonal concept, inspired by the LCS algorithm proposed by Nakatsu *et al.*[14]. Note that our LCPS algorithm is more efficient when L is close to m.

In our LCPS algorithm, two main operations are involved, including extension and minima-finding. We first define a dominating set $D_{i,s}$ with minima. A 3-tuple element $\langle x,y,z\rangle \in D_{i,s}$ means that there exists $|\text{CPS}(A_{1..i} + A_{m-x+1..m}, B_{1..y} + B_{n-z+1..n})| = l$, where $l = 2s$ or $l = 2s-1$. That is, there exists a *common palindromic subsequence* (CPS) of length l in $A' = a_1a_2\cdots a_ia_{m-x+1}\cdots a_{m-1}a_m$ and $B' = b_1b_2\cdots b_yb_{n-z+1}\cdots n_{n-1}b_n$ (not necessarily longest), which is constructed by s match characters, and $a_{m-x+1} = b_y = b_{n-z+1}$. Note that it is possible for either $a_i = a_{m-x+1}$ or $a_i \neq a_{m-x+1}$. For example, $A =$ `cbccbaabb` and $B =$ `bbccabbca`, in Table 2, $\langle 2,2,4\rangle \in D_{5,2}$ means that there exists $|\text{CPS}(A_{1..5} + A_{8..9}, B_{1..2} + B_{6..9})| = |\text{CPS}(a_1\cdots a_5a_8a_9,\ b_1b_2b_6\cdots b_9)| = 4$ with 2 match characters `bb`.

Definition 1 (Domination). *For a pair of 3-tuples $k = \langle x,y,z\rangle$ and $k' = \langle x',y',z'\rangle$ in set $D_{i,s}$, we say that k dominates k' if $x \leq x'$, $y \leq y'$ and $z \leq z'$.*

To form a CPS solution, the 3-tuple element $\langle x,y,z\rangle \in D_{i,s}$ represents the numbers of characters which have been used in sequences $\bar{A}$, B, and $\bar{B}$ are x, y and z, respectively. Thus, a 3-tuple is better or *dominant* if all integers in the 3-tuple are not larger than the corresponding integers in all other 3-tuples. For example, suppose that $k = \langle 6,3,6\rangle$ and $k' = \langle 7,4,6\rangle$. Then we say that k dominates k'. According to the problem introduced in Section 2.1, $D_{i,s}$ can be recognized as a *3-dimensional minima* set.

Table 2: The construction of $D_{i,s}$ in our LCPS algorithm with $A =$ `cbccbaabb` and $B =$ `bbccabbca`. Here, s denotes the number of match characters, and the CPS length may be $l = 2s$ or $2s-1$.

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

Now, we illustrate our algorithm with an example. The construction process of $D_{i,s}$ is shown in Table 2 for $A =$ `cbccbaabb` and $B =$ `bbccabbca`. Each $D_{i,s}$ is initialized to contain $\langle 0,0,0\rangle$ for $s = 0$.

In the first round ($r = 1$), we start with the first character $a_1 =$ `c` of sequence A, and find the first match `c` in $\bar{A}$, and get $a_4 =$ `c`, where 6 characters are used in $\bar{A}$. Then we find the first matches `c` in B and $\bar{B}$, and we obtain $b_3 =$ `c` and $b_8 =$ `c`, respectively, where 3 and 2 characters are used in B and $\bar{B}$, respectively. Thus, we get $D_{1,1} = \{\langle 6,3,2\rangle\}$, which is extended from $\langle 0,0,0\rangle$ in $D_{0,0}$. Next, we extend $\langle 6,3,2\rangle$ for $D_{2,2}$. We extend the character $a_2 =$ `b` in A, and find the first match `b` in $\bar{A}$, B and $\bar{B}$, which are a_2, b_6, and b_7. Now, 8, 6 and 3 characters have been used in $\bar{A}$, B and $\bar{B}$, respectively. So we have $D_{2,2} = \{\langle 8,6,3\rangle\}$. Then, extend $D_{2,2}$ with the character $a_3 =$ `c`. We cannot extend $\langle 8,6,3\rangle$ anymore since no appropriate match character `c` can be found in the remaining substring of A and B. So the first round stops, and we have CPS length 3 (`cbc`) with 2 match characters. The length is odd because A and $\bar{A}$ have the same match position at $a_2 =$ `b` in sequences A and $\bar{A}$.

In the second round ($r = 2$), we start with the second character $a_2 =$ `b` of sequence A, and find the first matches in $\bar{A}$, B and $\bar{B}$. We get a_9, b_1 and b_7, respectively, where 1, 1 and 3 characters are used in $\bar{A}$, B and $\bar{B}$, respectively. So we have $D_{2,1} = D_{1,1} \cup \{\langle 1,1,3\rangle\} = \{\langle 6,3,2\rangle, \langle 1,1,3\rangle\}$. Next, extend $D_{2,1}$ with character $a_3 = \texttt{c}$. We get that $\langle 6,3,2\rangle$ is extended to $\langle 7,4,6\rangle$ and $\langle 1,1,3\rangle$ is extended to $\langle 6,3,6\rangle$. We denote the process as that $D_{3,2} = \textsc{Extend}(D_{2,1}, 3) = \textsc{Extend}(\{\langle 6,3,2\rangle,\ \langle 1,1,3\rangle\}, 3) = \{\langle 7,4,6\rangle, \langle 6,3,6\rangle\}$. Here, $\langle 6,3,6\rangle$ dominates $\langle 7,4,6\rangle$ according to Definition 1. By adding $D_{2,2}$ and removing the dominated 3-tuple $\langle 7,4,6\rangle$, we get $D_{3,2} = \{\langle 8,6,3\rangle,\ \langle 6,3,6\rangle\}$. We denote the process as that $D_{3,2} \leftarrow \textsc{Dominate}(D_{2,2} \cup D_{3,2})$. The second round stops since no extension for $D_{3,2}$ can be achieved with $a_4 = \texttt{c}$. And we have CPS length 4 (`bccb`) with 2 match characters, which is obtained by tracing back from $\langle 6,3,6\rangle$ to $\langle 1,1,3\rangle$.

In the third round ($r = 3$), we start with the third character $a_3 = \texttt{c}$ of sequence A. $\textsc{Extend}(D_{2,0}, 3) = \{\langle 6,3,2\rangle\}$, and it is dominated by $\langle 6,3,2\rangle$ from $D_{2,1}$. Thus, $D_{3,1} = \textsc{Dominate}(\textsc{Extend}(D_{2,0}, 3) \cup D_{2,1})) = \{\langle 6,3,2\rangle, \langle 1,1,3\rangle\}$. Next, we extend $D_{3,1}$ with $a_4 = \texttt{c}$. We get $D_{4,2} = \textsc{Dominate}(\textsc{Extend}(D_{3,1}, 4) \cup D_{3,2})) = \{\langle 8,6,3\rangle, \langle 6,3,6\rangle\}$. The third round stops since no extension from $D_{4,2}$ with $a_5 = \texttt{b}$ can be obtained. And we have CPS length 4 (`bccb`) with 2 match characters, which is obtained by tracing back from $\langle 6,3,6\rangle$ to $\langle 1,1,3\rangle$.

In the fourth round ($r = 4$), we start with the fourth character $a_4 = \texttt{c}$ of sequence A. $\textsc{Extend}(D_{3,0}, 4) = \{\langle 6,3,2\rangle\}$, and it is dominated by $\langle 6,3,2\rangle$ from $D_{3,1}$. Thus, $D_{4,1} = \{\langle 6,3,2\rangle, \langle 1,1,3\rangle\}$. Next, we extend $D_{4,1}$ with $a_5 = \texttt{b}$. $\textsc{Extend}(D_{4,1}, 5) = \{\langle 2,2,4\rangle\}$. $\langle 2,2,4\rangle$ dominates $\langle 6,3,6\rangle$ in $D_{4,2}$. In other words, $D_{5,2} = \textsc{Dominate}(\textsc{Extend}(D_{4,1}, 5) \cup D_{4,2})) = \{\langle 8,6,3\rangle, \langle 2,2,4\rangle\}$. Finally, extending $D_{5,2}$ with $a_6 = \texttt{a}$, we get $D_{6,3} = \textsc{Extend}(D_{5,2}, 6) = \{\langle 3,5,5\rangle\}$. The fourth round stops since no extension can be found further. Now, we have CPS length 5 (`bbabb`) with 3 match characters. The length is odd because $\langle 3,5,5\rangle$ in $D_{6,3}$ matches the same position at b_5 in sequences B and $\bar{B}$. Moreover, the whole algorithm ends since we cannot find any CPS longer than 5 in the next round (only 5 characters of A are remained to extend in round 5). Finally we, get the answer of LCPS length is 5 (`bbabb`), which can be got by tracing back from $\langle 3,5,5\rangle$ through $\langle 2,2,4\rangle$ to $\langle 1,1,3\rangle$.

In Table 2, we show how to calculate each $D_{i,s}$ by using the diagonal concept. The main operations involved in the algorithm is $\textsc{Extend}$ and $\textsc{Dominate}$. In summary, we obtain $D_{i,s}$ by $\textsc{Dominate}(D_{i-1,s} \cup \textsc{Extend}(D_{i-1,s-1}, i))$. The formal algorithm is omitted here.

To analyze the time complexity of our algorithm, we first observe that the number of distinct 3-tuples of one column in Table 2 is bounded, as described in the following theorem.

Theorem 1. *$|D_{s,s} \cup D_{s+1,s} \cup D_{s+2,s} \cup \cdots| = O(Rn/|\Sigma|)$ in average, for some fixed value of s.*

Proof. Omitted here. □

For any two 3-tuples in each $D_{i,s}$, their x and y values cannot be equal at the same time. That is, $|D_{i,s}| \le R$. The extension of $D_{i,s}$ requires $O(|D_{i,s}|) = O(R)$ time, where R is $O(mn) = O(n^2)$ in the worst case. The operation $\textsc{Dominate}$ can be implemented by the 3-dimensional minima finding algorithm, described in Section 2.1, with $O(|D| \log |D|)$ time. Thus, the time required for $\textsc{Dominate}(D_{i,s})$ is $O(R \log R) = O(R \log n)$.

The algorithm terminates when round $r \ge m - L$. Each round performs the extension at most $\lceil L/2 \rceil$ times. Hence, the LCPS problem can be solved in $O(L(m-L)R \log n)$ time. The space complexity of the algorithm is $O(RL)$.

4 Experimental Results

In the experiments, we compare the execution time of our algorithm, the algorithm proposed by Chowdhury *et al.* [7], and the algorithm proposed by Inenaga and Hyyrö [12]. Each experiment is performed 100 times to get the average execution time. These algorithms are implemented in Java by Eclipse 4.6.3, and they are tested on a computer with 64-bit Windows 10 OS, CPU clock rate of 3.00GHz (Intel(R) Core(TM) i5-7400 CPU) and 16 GB of RAM.

We use a 5-tuple $(|A|, |B|, |\Sigma|, algo, similarity)$ to represent the parameters in each performance chart. For example, $(200, 200, 2, *, *)$ means that $|A| = |B| = 200$, $|\Sigma| = 2$, the first "$*$" is a wildcard representing all possible algorithms, and the second "$*$" is a wildcard representing all possible similarities. We omit the 5th parameter *similarity* when the performance chart does not consider the similarities of the data set.

Out experiment is to test the algorithms for various values of $|\Sigma| \in \{2, 4, 20\}$, with the input lengths n ranging from 10 to 200 if $n \ge |\Sigma|$. The test data A and B are generated with a fully random manner and they are picked up if the input sequences involve all characters $c \in \Sigma$. The results of the experiment are shown in Fig. 1.

The algorithm of Chowdhury *et al.* solves the LCPS problem with the dynamic programming method. Their computational amount is larger than the number R of match pairs. Fig. 1 shows that the algorithm of Chowdhury *et al.* takes more time than other algorithms. The algorithm of Inenaga and Hyyrö and our algorithm are both related to the number of match pairs. When $|\Sigma|$ is getting larger, the number of match pairs become smaller. Thus, the two algorithms are efficient when $|\Sigma|$ is large. The algorithm of Inenaga and Hyyrö extends all possible rectangles that may form a palindrome, whose size is $O(R^2)$. Our algorithm extends only the current optimal 3-tuples. The calculation of our algorithm is practically less than the algorithm of Inenaga and Hyyrö. Thus, our algorithm has better performance than the other algorithms in almost all cases.

Our algorithm has better performance than the other algorithms. In our algorithm, each $D_{i,s}$ only keeps the minima and dominant 3-tuples. And the number of 3-tuples in most $D_{i,s}$ is much smaller than R in the simulations. Thus, the amount of calculations of our algorithm is much smaller than the theoretical bound of our algorithm, which is $L(m-L)R$ where R is $mn/|\Sigma|$ in average.

5 Conclusion

This paper proposes a diagonal algorithm for solving the longest common palindromic subsequence (LCPS) problem with $O(L(m-L)R \log n)$ time and $O(RL)$ space in the worst case, where m, n, and L denote the lengths of A, B, and LCPS(A, B), respectively, and R denotes the number of match pairs between A and B. In the future, the theoretical time complexity may be improved by reducing the domination and the upper bound of $|D_{i,s}|$.

Furthermore, the *cyclic longest common palindromic subsequence* problem is a variant of the LCPS problem. The cyclic version of the LCPS problem can separate the problem into subproblems by cutting the two cyclic sequences with $O(mn)$ different ways. And each subproblem can be solved by the LCPS algorithms. In the cyclic problem, shifting a cutting position is similar to appending a new character to the sequence. Therefore, the online version of the LCPS problem

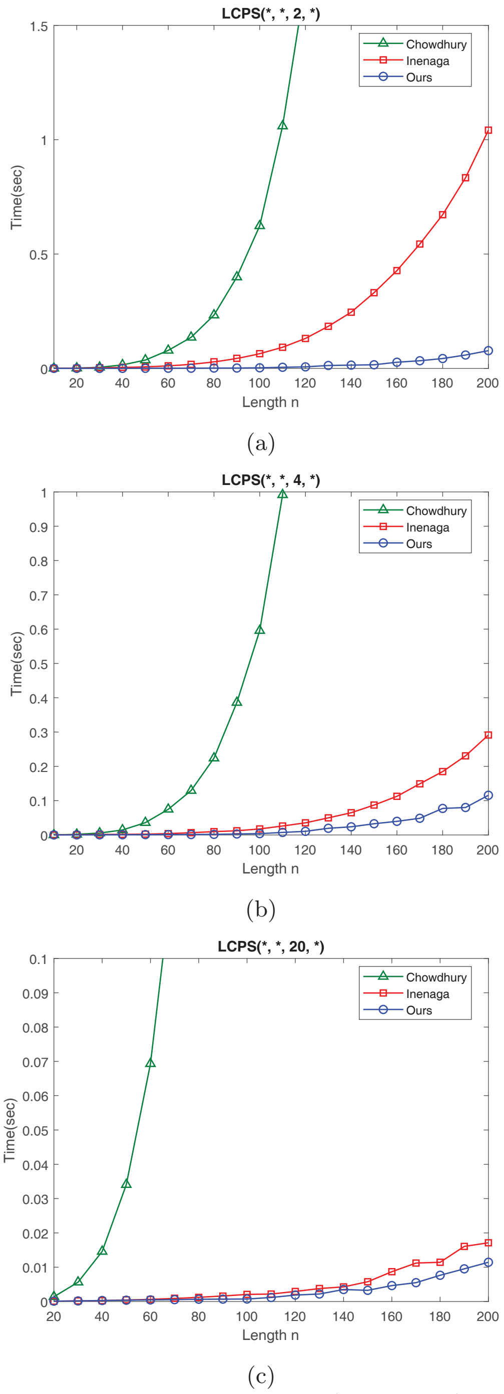

Figure 1: The execution time (in seconds) of the three algorithms for various $|\Sigma|$ with input lengths ranging from 10 to 200, where $m = n$. (a) $|\Sigma| = 2$. (b) $|\Sigma| = 4$. (c) $|\Sigma| = 20$.

deserves the further study.

References

[1] A. Alatabbi, C. Iliopoulos, and M. Rahman, "Maximal palindromic factorization," *Proceedings of the Prague Stringology Conference*, Prague, Czech, pp. 70–77, Jan. 2013.

[2] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, New York, USA, pp. 39–44, 2003.

[3] S. W. Bae and I. Lee, "On finding a longest common palindromic subsequence," *Theoretical Computer Science*, Vol. 710, pp. 29–34, 2018.

[4] B. S. Baker and R. Giancarlo, "Longest common subsequence from fragments via sparse dynamic programming," *Proceedings of the 6th Annual European Symposium on Algorithms*, Venice, Italy, pp. 79–90, 1998.

[5] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," *Proceedings of the Seventh International Symposium on String Processing and Information Retrieval*, A Coruna, Spain, pp. 39–48, Sept. 2000.

[6] H. T. Chan, H. T. Chiu, C. B. Yang, and Y. H. Peng, "The generalized definitions of the two-dimensional largest common substructure problems," *Algorithmica*, Vol. 82, pp. 2039–2062, 2020.

[7] S. R. Chowdhury, M. M. Hasan, S. Iqbal, and M. S. Rahman, "Computing a longest common palindromic subsequence," *Fundamenta Informaticae*, Vol. 129, No. 4, pp. 329–340, Oct. 2014.

[8] G. Fici, T. Gagie, J. Krkkinen, and D. Kempa, "A subquadratic algorithm for minimum palindromic factorization," *Journal of Discrete Algorithms*, Vol. 28, pp. 41–48, 2014.

[9] A. Fuglsang, "The relationship between palindrome avoidance and intragenic codon usage variations: A Monte Carlo study," *Biochemical and Biophysical Research Communications*, Vol. 316, pp. 755–762, May 2004.

[10] R. Groult, lise Prieur, and G. Richomme, "Counting distinct palindromes in a word in linear time," *Information Processing Letters*, Vol. 110, No. 20, pp. 908–912, 2010.

[11] M. Hoffmann and J. Rychlewski, "Searching for palindromic sequences in primary structure of proteins," *Computational Methods in Science and Technology*, Vol. 5, pp. 21–24, Jan. 1999.

[12] S. Inenaga and H. Hyyrö, "A hardness result and new algorithm for the longest common palindromic subsequence problem," *Information Processing Letters*, Vol. 129, pp. 11–15, 2018.

[13] T.-W. Liang, "A diagonal-based algorithm for the longest common palindromic subsequence problem," Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, 2019.

[14] N. Nakatsu, Y. Kambayashi, and S. Yajima, "A longest common subsequence algorithm suitable for similar text strings," *Acta Informatica*, Vol. 18, No. 2, pp. 171–179, Nov. 1982.

[15] A. H. Porto and V. C. Barbosa, "Finding approximate palindromes in strings," *Pattern Recognition*, Vol. 35, No. 11, pp. 2581 – 2591, 2002.

[16] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM*, Vol. 22, pp. 469–476, Oct. 1975.