國立臺灣大學電機資訊學院資訊工程學系

博士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Doctoral Dissertation

附加限制條件的最長共同子序列問題之演算法設計

Efficient Algorithms for the Constrained Longest Common Subsequence Problems

陳怡靜

Yi-Ching Chen

指導教授：趙坤茂 博士

Advisor: Dr. Kun-Mao Chao

中華民國九十九年六月

June, 2010

誌謝

2005 年 6 月 10 日，以一封寫給趙坤茂教授的 e-mail 為開端。

每當有人問起博士班指導教授時，我總是很驕傲地說：「是趙坤茂教授。」趙老師幽默風趣的言談、身體力行的研究態度總是能帶給我前進的力量。感謝老師在我遇到瓶頸、進退兩難時，不厭其煩地鼓勵我，並且給予我諸多寶貴的建議。也謝謝老師總是在關鍵時刻捎來幾句關心與祝福，讓我如同大力水手吃下菠菜罐頭後衝勁十足。滿懷的感謝難以訴諸於文字，也許數十年後，我會忘記寫過的論文細節，但是老師的恩情將是沒齒亦難忘。老師，謝謝您！

論文計畫審查與口試期間，承蒙陳健輝教授、陳俊良教授、劉邦鋒教授、曾宇鳳教授與蔡英德教授提出諸多寶貴意見，使得本論文得以更臻完善，並且感謝每位委員給予許多鼓勵與肯定。此外，要特別感謝陳健輝教授經常關心我的學業，林守德教授與林婉瑜老師傳授許多英文論文寫作技巧，在修課結束後依然關心我的研究進展；還要感謝國立暨南國際大學的杜迪榕教授、阮夙姿教授、黃光璿教授持續關心我在台北的生活與研究，並給予諸多勉勵。

在進入台大之前，聽到種種與資格考有關的恐怖傳言。感謝一起努力的伙伴：秉慧、冠宇、冠伶、國煒、啟維、啟堯、容任、思遠、銘康、義興，因為有你們的打氣與陪伴，我才能順利通過每科考試。我永遠記得從讀書會啟動到放榜前焦慮失眠與作惡夢的那些日子，苦澀卻甜美。

感謝在這五年間，相處時間遠超過家人的 ACB 夥伴：耀廷學長時常關心我們這些老毛頭的近況，並且給予我許多精神上的勉勵與實質上的餵食；效飛學長與弘倫學長總是耐著性子回答我研究上遇到的問題，並且慷慨分享求職經驗；建均經常舉辦研究室出遊舒緩壓力，同時經常帶給我們女性夥伴諸多生活上的「色彩」；一軒經常分享在美國求學與工作遇到的趣事與辛苦，並且在我到舊金山旅行時盛情招待；遇到電腦麻煩事時，芃安與陳琨總是熱心幫忙；機車在校外拋錨時，明江與冠宇多次出手相救；秉慧在我初到台北時給予諸多陪伴；正偉與峻偉在我口試前提供即時的幫助；秋芸陪著我耍幼稚、做白日夢；家榮與伍隆時常分享許多生活趣事與讀書心得；容任、帥朋、謦儀、蔚茵給予我許多鼓勵；安強為大家留下許多活動倩影。感謝所有 ACB 伙伴，烏來、淡水、平溪、花東綠島、澎湖等多次出遊，還有話三國、雨天訂便當都是我最珍貴的回憶。

另外,要感謝身邊所有朋友的陪伴與鼓勵,特別是佳衛幫忙解決所有使用 LaTex 的疑難雜症，吳瑞瑜學姊像姊姊般照顧我，晏禕、彥緯、林清池學長在我求職期間給予許多寶貴的建議，以及那些曾讓我擁有夢想的人們。

每當我走在台北的街頭看到各種年齡的辛勤工作者，總是會質疑自己憑什麼擁有現在的生活。一切的一切，都要感謝養育、支持我的爸媽，謝謝您們賜予我健康的身體，謝謝您們提供衣食無缺的成長環境，謝謝您們一直以來把彼此照顧得很好，讓我能夠專心於學業、無後顧之憂。同時要感謝我的後援團，五伯與二姑一直提供我求學過程溫暖的避風港，堂哥奕丞給予我很多溫暖的陪伴；回老家時，二伯與四伯總是給予我最熱情的款待；最後要謝謝爺爺奶奶賜予我眾多珍貴的家人。

2010 年 7 月 27 日，擁有 ACB 鑰匙的第 1822 天。雖然不捨，但我依然會瀟灑地離開，邁向另一個嶄新的旅程。

祝福所有關心我的朋友們平安快樂。

陳怡靜謹致
國立台灣大學資訊工程學系
中華民國九十九年七月

中文摘要

本篇論文探討數個最長共同子序列的變異問題，是由分子生物學和序列比對的實際應用與理論興趣發展而來。

在論文的第一部份，我們研究四個附加條件限制的最長共同子序列問題，目的是在兩條序列之共同子序列中，求得包含或排除一條附加限制字串為子序列或子字串之最長序列。我們研究這些問題的最佳化特性，並針對每個問題提出一個動態規劃演算法。理論分析顯示，我們提出的演算法之時間複雜度與兩條序列及一條附加限制字串的長度乘積成正比。此外，我們也考慮任意多條附加限制字串的情況。

為了使序列的相似度衡量更有彈性，在論文的第二部份，我們研究一個混合附加條件限制的問題，目的是在兩條序列之共同子序列中，求得包含一條附加限制字串為子序列且排除另一條附加限制字串為子序列之最長序列。我們提出一個動態規劃演算法來解決這個問題，演算法所需的時間與兩條序列及兩條附加限制字串的長度乘積成正比。另外，我們提出一個針對兩條序列配對位置做計算的快速演算法。

在論文最後一個部份，我們在最長共同子序列問題與一個附加條件限制的最長共同子序列問題上考慮一個被廣泛使用的資料壓縮技術，稱為區段長度編碼法。為了解決以區段長度編碼的序列之最長共同子序列問題，我們研究序列以區段分割動態規劃矩陣的特性，並利用簡化矩陣內部份計算來求得最長共同子序列的長度。最後，我們設計兩個演算法，在兩條以區段長度編碼的序列之共同子序列中，求得包含一條以區段長度編碼的附加限制字串為子序列之最長序列。

關鍵字：動態規劃、最長共同子序列、附加限制條件之最長共同子序列、區段長度編碼

Abstract

This dissertation studies several variants of the longest common subsequence (abbreviated LCS) problem. These variants arise from some applications and theoretical interests in molecular biology and sequence comparison.

In the first part of this dissertation, we study four constrained LCS (abbreviated CLCS) problems, each of which is to find a longest sequence that is a common subsequence of two sequences and either includes or excludes a constrained pattern as a subsequence or substring. We investigate the optimality principles of these problems and then derive a dynamic programming algorithm for each problem. The theoretical analyses show that the time complexity of each algorithm is proportional to the product of the lengths of the given sequences and constrained pattern. We also consider the case where the number of constrained patterns in each problem is arbitrary.

To make the similarity measurement of sequences more flexible, in the second part of this dissertation, we study the problem of finding a longest sequence that is a common subsequence of two sequences and not merely includes a constrained pattern as a subsequence but excludes the other constrained pattern as a subsequence. We give a dynamic programming algorithm whose time complexity is proportional to the product of the lengths of the given sequences and constrained patterns. We also present a fast algorithm which restricts the computation on the positions of matches between the sequences.

In the last part of this dissertation, we consider a common used data compression scheme called run-length encoding (abbreviated RLE) on the input sequences of the LCS problem and one of the CLCS problems. To solve the LCS problem of two RLE sequences, we investigate the properties of the partition, induced by the runs of two sequences, in the dynamic programming matrix for the LCS problem and exploit the sequences for computing the length of an LCS by utilizing the simplicity of some positions. Finally, we devise two algorithms for the problem of finding a longest sequence that is a common subsequence of two RLE sequences and includes a constrained RLE pattern as a subsequence.

Keywords: Dynamic Programming; Longest Common Subsequence; Constrained Longest Common Subsequence; Run-Length Encoding

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mystery of the life attracts the human begin for a long time. Since scientists found the encoded sequences of the nucleotides in the chromosomes of each cell, the domain of molecular biology has arisen. The four nucleotides (i.e., `A`, `T` or `U`, `C`, and `G`) are assembled together into DNA (deoxyribonucleic acid) or RNA (ribonucleic acid). Extracting the information of nucleic acid sequences can help scientists to comprehend the functional expressions of segments and the evolutionary relationships among species, but it is difficult to identify the sequences by human eyes. This obstacle can be overcome by modeling many biological issues into the classical problems in computer science such as sequence comparison.

Sequence alignment is a well-known sequence comparison problem, which is to align two sequences and the similarity between the sequences is measured by its score. Needleman and Wunsch [62] proposed a dynamic programming algorithm that performs a global alignment of two protein sequences. The multiple sequence alignment problem is to align simultaneously an arbitrary number of sequences and has been shown to be NP-complete [48, 78], i.e., there is no polynomial-time algorithm to find the mathematically optimal solution unless NP = P. In some applications, however, the suboptimal solution might be acceptable to develop efficient tools for reducing the time or space complexity. To reduce the computational cost, most multiple sequence alignment algorithms use heuristic approaches rather than global optimization.

In contrast to the global alignment problem, several methods were developed for the local alignment problem, such as the Smith-Waterman algorithm [69], which is to determine similar regions between two genetic sequences. Because the local alignment problem has wide implementations in sequence database searching, the problem was intensively investigated for the past two decades to develop efficient tools. Pearson and Lipman [63] proposed a heuristic approach to improving the Smith-Waterman algorithm and developed a tool named FASTA, which first identifies short matched regions (also called seeds), and then performs the Smith-Waterman algorithm on the seeds. The FASTA method reduces the time complexity of the Smith-Waterman algorithm, and meanwhile obtains good results in most cases. Altschul *et al.* [2] developed a popular tool named the basic local alignment search tool (abbreviated BLAST) whose running time is faster than the Smith-Waterman algorithm and FASTA method. With the seeds, the BLAST method extends seeds without gap existence by finding the high-scoring pairs of the aligned seeds. Another tool named gapped-BLAST [3] uses a new criterion for extending seeds and generates a gapped alignment for seed pairs with high scores. Because the gapped-BLAST method is a bit faster and more flexible in the practical application than BLAST, the tool is widely used up to now.

The longest common subsequence (abbreviated LCS) problem is another sequence comparison problem, which is a simple form of sequence alignment [25, 37]. The LCS problem measures the similarity of the sequences by only counting the identical aligned pairs. The problem was first proposed by Wagner and Fischer [77]. Since then, it has been intensively studied in the theoretical computer science and much ink has been spent on it [1, 4, 13, 18, 21, 22, 31, 36, 37, 42, 45, 51, 57, 60, 61, 66, 77]. In fact, the problem of an arbitrary number of sequences, even on a binary alphabet, has been shown to be NP-complete [55]. The problem has applications not only in molecular biology, but also in pattern comparison, and screen redisplay. On the side of molecular biology, several studies devoted to saving computational space [41, 67] due to the long lengths of DNA sequences. The LCS problem can be reduced to two well-known problems. One is to obtain a maximum-weight directed path on a two-dimensional grid that is converted from its dynamic programming table. The other is the longest increasing subsequence problem [68], which is to look for a numeric subsequence of a sequence that is strictly increasing. In addition, a similar problem to the LCS problem, named longest common substring problem, is defined for computing a maximum-length substring that appears in both two sequences. This problem can be solved in linear time by using a well-known data structure - generalized suffix tree [37].

Arising from several applications in molecular biology and pattern comparison, some constraints on sequence comparison are considered. From a sequence function point of view, for example, the Human and bovine pancreatic ribonuclease (RNase) sequences contain a conserved catalytic triad, `His-12(H)`, `Lys-41(K)`, `His-119(H)`, which is essential for RNA degradation. For aligning each residue of the conserved catalytic triad of such RNase sequences in the same column, Tang *et al.* [70] addressed the constrained multiple sequence alignment problem of finding an alignment such that each character of the additional constrained pattern should be aligned with the same character of every sequence. Given two sequences of length at most n and a constrained pattern of length d, Tang *et al.* [70] presented an algorithm with both time and space requiring $O(dn^4)$. Chin *et al.* [28] further investigated the problem and improved the time and space requirements both into $O(dn^2)$. Later, many studies devote to this issue [30, 38, 39, 40, 54, 64, 65, 73, 74].

Relying on some applications, it is advantageous to use a particular representation for text documents to be compared or transferred. For speeding up the comparing and transferring time of documents, two categories of data compression, i.e., lossless compression (also called reversal compression) and lossy compression (also called irreversal compression), have been developed. Lossless compression schemes, such as run-length encoding, Huffman coding, and arithmetic coding, exploit statistical redundancy in such a way as to represent the sender's data more concisely without error. Lossy compression schemes, such as fractal compression and vector quantization, scrap some of less-relevant information if some loss of precision is acceptable. Lossless compression schemes are generally used for compressing text files, executable programs, medical images because of their precise and concise properties. On the other hand, because lossy compression schemes are faster than the other ones, the coding schemes are commonly used for compressing graphic and musical documents.

Run-length encoding (RLE) is a simple coding scheme of lossless compression schemes, which compresses a sequence into several runs so that each run is a maximal-length substring with an identical character in the sequence. A sequence in RLE format is represented by an ordered sequence of the characters corresponding to the runs with their lengths. For example, if sequence $A = \texttt{aaaabbcccccaaaaaaaa}$, the RLE representation of A is $\texttt{a}^4\texttt{b}^2\texttt{c}^5\texttt{a}^8$. A well-known and relatively efficient application of RLE is the fax transmission because most fax documents are composed of a great portion of white space with a small portion of black space [20]. Many studies have devoted to the sequence comparison with RLE sequences, such as pattern matching [5, 9, 14, 27, 56], edit distance [15, 24], sequence alignment [44, 50, 52], LCS [11, 34, 53, 58].

In this dissertation, we study the following constrained longest common subsequence problems.

- Constrained LCSs (abbreviated CLCS). The input of the CLCS problem is two sequences X, Y and a constrained pattern P over a finite alphabet. Four variants of the CLCS problem are defined as follows.
    - The SEQ-IC-LCS problem [17, 26, 29, 46, 72]: find a longest sequence that is a common subsequence of X and Y and includes P as a subsequence.
    - The STR-IC-LCS problem: find a longest sequence that is a common subsequence of X and Y and includes P as a substring.
    - The SEQ-EC-LCS problem: find a longest sequence that is a common subsequence of X and Y and excludes P as a subsequence.
    - The STR-EC-LCS problem: find a longest sequence that is a common subsequence of X and Y and excludes P as a substring.

  The former two problems have applications on molecular biology and pattern comparison to take a common specific segment into consideration for similarity measurement between two sequences. The last two problems consider the theoretical opposite constraints to the former two problems. We also investigate the CLCS problems with an arbitrary number of constrained patterns.

- Hybrid Constrained LCSs (abbreviated HC-LCS). The problem considers the constraints of the SEQ-IC-LCS and SEQ-EC-LCS problems. Given two sequences X, Y and two constrained patterns P, Q over a finite alphabet, the HC-LCS problem is to find a longest sequence that is a common subsequence of X and Y and not merely includes P as a subsequence but excludes Q as a subsequence. We give the properties of the HC-LCS problem and solve it by using the dynamic programming technique. To speed up the computation time, we further employ a data structure based upon van Emde Boas trees and restrict the computation on the positions where the corresponding characters of each pair are identical.

- LCSs of Run-Length Encoded Sequences (abbreviated RLE-LCS). We first exploit the properties of RLE sequences and introduce an approach proposed by Ann *et al.* [11]. We further adapt Hunt-Szymanski strategy for improving Ann *et al.* approach for speeding up the computation to the pairs of runs whose corresponding characters are identical.

- Constrained LCSs of Run-Length Encoded Sequences (abbreviated RLE-CLCS). We consider the SEQ-IC-LCS problem of RLE sequences over a finite alphabet. We first present a simple algorithm, and then devise a fast algorithm for speeding up the computation to the runs where the corresponding characters of two sequences are identical by adapting the framework of our approach to the RLE-LCS problem.

Chapter 2

Longest Common Subsequences (LCSs)

In this chapter we first give some basic definitions. We then introduce the longest common subsequence problem and address three well-known approaches to the problem as our preliminary knowledge in the forthcoming chapters. In addition, some previous results are briefly mentioned.

Definition 2.1. *An alphabet Σ is a finite set of symbols. An element of Σ is called character. A sequence over Σ is a string of characters of Σ.*

Let $a_1a_2\dots a_m$ denote a sequence A of length m over Σ. Character $\sigma \in \Sigma$ is at position i in a sequence A if $\sigma = a_i$. Given sequences A, B and a character $\sigma \in \Sigma$, sequences AB and $A\sigma$ denote the sequences that are constructed by appending B and σ to A, respectively. In this dissertation, we use capital letters to denote sequences and lowercase letters to denote characters.

Definition 2.2. *A subsequence of a sequence A is obtained by deleting zero or more characters from A (not necessarily contiguous). A substring of a sequence A is a subsequence of successive characters within A.*

Definition 2.3. *A prefix of a sequence A is a substring that begins at the first position in A. A suffix of a sequence A is a substring that ends at the last position in A.*

Given a sequence $A = a_1 a_2 \ldots a_n$, we denote $A[i..j]$ as the substring $a_i a_{i+1} \ldots a_j$ of A if $1 \leq i \leq j \leq m$, and an empty string otherwise. For example, if $A = \texttt{believe}$, then $A[3..5] = \texttt{lie}$. A substring $A[1..i]$ for any $i \in \{1, 2, \ldots, n\}$ is a prefix of A, and a substring $A[j..n]$ for any $j \in \{1, 2, \ldots, n\}$ is a suffix of A.

Definition 2.4. *A common subsequence (abbreviated CS) of two sequences A and B is a subsequence that appears both in A and B. A longest common subsequence (abbreviated LCS) of A and B is a maximum-length CS of A and B.*

Wagner and Fischer [77] proposed the LCS problem in 1974, which is formally defined as follows.

Problem 2.1. ***(LCS)*** *[77] Given two sequences X and Y of lengths m and n, respectively, the LCS problem is to find a maximum-length subsequences that appears in both X and Y.*

The LCS problem has been intensively studied for more than three decades [1, 13, 18, 21, 31, 37, 41, 42, 45, 57, 66, 47]. In general, there may exist more than one LCS between two sequences. For example, if X = `ACTGCCTAGGC` and Y = `CGATCTGGAC`, "`ATCTGGC`" is an LCS of X and$Y$, and "`CTCTGGC`" and "`CGCTGGC`" are two another LCSs of X and Y.

A naive approach to the LCS problem is to enumerate all subsequences of X and then check every subsequence if it is a subsequence of Y. Nevertheless, the approach requires exponential running time because X contains 2^m subsequences. It causes the difficulty in the implementation for long sequences.

Dynamic programming (abbreviated DP) technique is a wildly used approach to solving the LCS problem by the optimality principles of the problem. In the following we introduce three well-known DP approaches to the LCS problem, which are the traditional DP algorithm [31], space-saving strategy [41] using the divide-and-conquer technique to solve the problem in linear space, and Hunt-Szymanski strategy [45] disregarding the computation of the positions at which X and Y mismatch each other.

2.1 Dynamic Programming

Lemma 2.1 decomposes the structure of an optimal solution based on the solutions to its smaller subproblems.

Lemma 2.1. *Let $\mathcal{Z}_{i,j}$ denote the set of all LCSs of $X[1..i]$ and $Y[1..j]$. If $Z = z_1z_2 \dots z_l \in \mathcal{Z}_{i,j}$, the following conditions hold:*

(1) *If* $x_i = y_j$*, then* $z_l = x_i = y_j$ *and* $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1}$.

(2) *If* $x_i \neq y_j$*, then* $z_l \neq x_i$ *implies* $Z[1..l] \in \mathcal{Z}_{i-1,j}$.

(3) *If* $x_i \neq y_j$*, then* $z_l \neq y_j$ *implies* $Z[1..l] \in \mathcal{Z}_{i,j-1}$.

Let $\mathcal{L}[i,j]$ denote the length of any sequence belonging to $\mathcal{Z}_{i,j}$. The value $\mathcal{L}[i,j]$ given to position (i,j) is determined by three neighboring positions, $(i-1,j-1)$, $(i-1,j)$, and $(i,j-1)$ (see Figure 2.1(a)), and by the characters x_i and y_j. By the optimality principles of the LCS problem shown in Lemma 2.1, the recurrence for computing $\mathcal{L}[i,j]$ with the initializations of $\mathcal{L}[0,0] = \mathcal{L}[i,0] = \mathcal{L}[0,j] = 0$, for $i \in \{1,2,\ldots,m\}$ and $j \in \{1,2,\ldots,n\}$, can be formulated as follows.

$$\mathcal{L}[i,j] = \max \begin{cases} \mathcal{L}[i-1,j-1]+1 & \text{if } x_i = y_j, \\ \max\{\mathcal{L}[i-1,j], \mathcal{L}[i,j-1]\} & \text{otherwise.} \end{cases} \tag{2.1}$$

Suppose that Z is a sequence belonging to $\mathcal{Z}_{m,n}$ and is initialized to be an empty sequence. The length of Z is given by $\mathcal{L}[m,n]$, which is computed in $O(mn)$ time and space. While computing each entry $\mathcal{L}[i,j]$, we can build a (backtracking) link to point out the position which $\mathcal{L}[i,j]$ results from. Sequence Z in reverse order can be constructed by tracing back through the links from $\mathcal{L}[m,n]$ to $\mathcal{L}[0,0]$ (see Figure 2.1(b) as an example). The stage of backtracking takes $O(m+n)$ steps. Consequently, the following theorem is stated.

Theorem 2.1. *The LCS problem can be solved in* $O(mn)$ *time and space.*

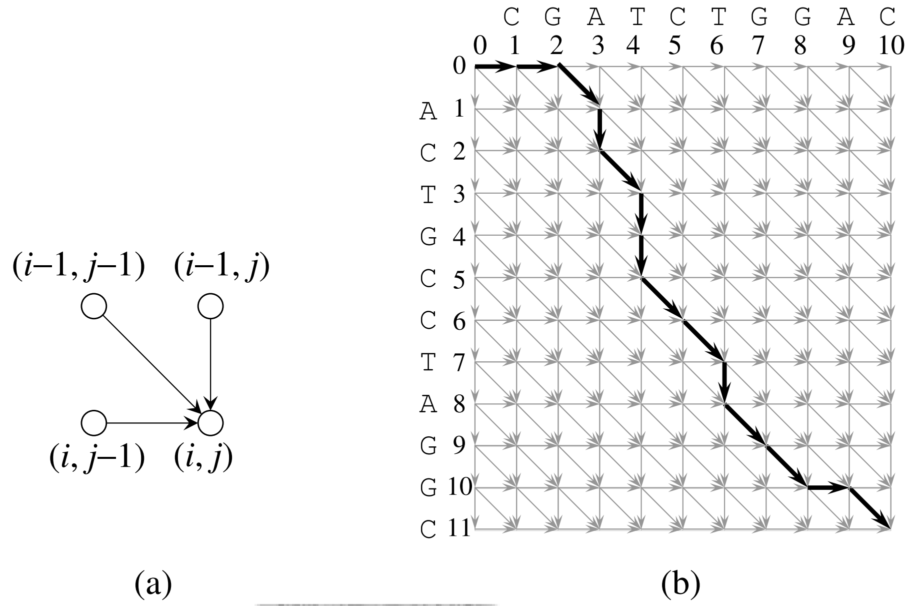

Figure 2.1: An LCS between two sequences. (a) Three neighboring positions of position (i, j). (b) A path in a directed acyclic graph over the LCS metric.

2.2 Space-Saving Strategy

For long sequences such as human genomic sequences of about 3×10^9 base pairs, space restriction is important to the implement of solving the LCS problem. Figure 2.2 illustrates the two linear-space approaches to computing the length of an LCS. Because computing an entry in each row of matrix $\mathcal{L}$ only needs the values in the preceding and present rows, one approach is to reduce the space to twice the number of entries in a row. On the other hand, Figure 2.1(a) shows that the value of any position (i, j) in $\mathcal{L}$ is only caused from positions $(i-1, j)$, $(i-1, j-1)$, or $(i, j-1)$; accordingly, the space requirement can be further reduced to the number of entries in a row plus one. Constructing an LCS requires the information of backtracking links; unfortunately,

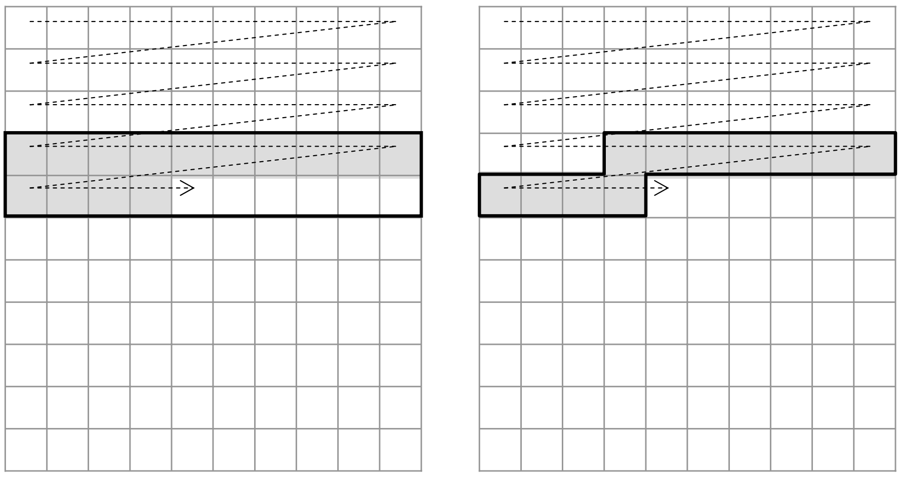

Figure 2.2: Computing the length of an LCS in linear time.

such information can not be achieved in linear space.

In 1975 Hirschberg [41] proposed a divide-and-conquer approach, named Algorithm LINEARLCS, that performs an LCS between two sequences in quadratic time and linear space. Let $LCS(X, Y)$ denote the length of an LCS of X and Y. Hirschberg derived the following lemma.

Lemma 2.2. *For any* $i \in \{0, 1, \ldots, m\}$, $LCS(X[1..m], Y[2..n]) =$ $\max_{0 \leq j \leq n} \{LCS(X[1, i], Y[1..j]) + LCS(X[i+1..m], Y[j+1..n])\}$.

The space-saving strategy adopts Lemma 2.2 to recursively divide a given problem into two smaller problems until it is a trivial problem. Figure 2.3 illustrates the idea of the strategy. The middle row of the given problem is chosen as a partition line. In the beginning of the LCS problem, the $\lfloor \frac{m}{2} \rfloor$-th row is chosen (see Figure 2.3(a)). The lengths of an LCS between $X[1..\lfloor \frac{m}{2} \rfloor]$ and $Y[1..j]$ and an LCS between $X[\lfloor \frac{m}{2} \rfloor + 1..m]$

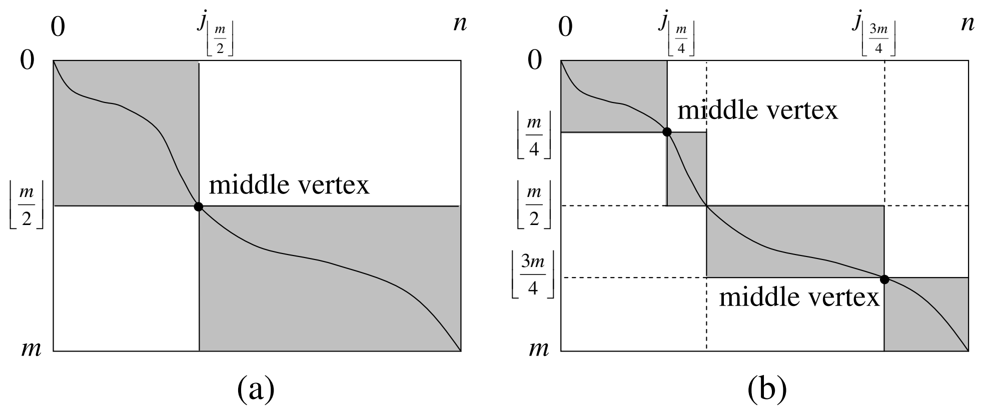
0
$j_{\lfloor \frac{m}{2} \rfloor}$
n
0
$\lfloor \frac{m}{2} \rfloor$
middle vertex
m
(a)
0
$j_{\lfloor \frac{m}{4} \rfloor}$
$j_{\lfloor \frac{3m}{4} \rfloor}$
n
0
$\lfloor \frac{m}{4} \rfloor$
middle vertex
$\lfloor \frac{m}{2} \rfloor$
$\lfloor \frac{3m}{4} \rfloor$
middle vertex
m
(b)

Figure 2.3: Deriving an LCS in linear space. (a) The partition at the first iteration. (b) The partitions at the second iteration.

and $Y[j+1..n]$ for all $j \in \{0, 1, \ldots, n\}$ are separately calculated by the former linear-space approaches. The j-coordinate of the middle vertex, denoted by $j_{\lfloor \frac{m}{2} \rfloor}$, is defined by the minimum j such that $LCS(X[1, i], Y[1..j]) + LCS(X[i+1..m], Y[j+1..n])$ is maximum. With the coordinates of the middle vertex $(\lfloor \frac{m}{2} \rfloor, j_{\lfloor \frac{m}{2} \rfloor})$, the problem can be divided into two subproblems, which are the LCS problems of $X[1..\lfloor \frac{m}{2} \rfloor]$ and $Y[1..j_{\lfloor \frac{m}{2} \rfloor}]$ and of $X[\lfloor \frac{m}{2} \rfloor + 1..m]$ and $Y[j_{\lfloor \frac{m}{2} \rfloor} + 1..n]$. Figure 2.3(b) illustrates the

```
Algorithm LinearLCS($m, n, X[1..m], Y[1..n]$)
1: if $m \leq 1$ then
2:     output an LCS of $X$ and $Y$;
3: else
4:     $i \leftarrow \lfloor \frac{m}{2} \rfloor$;
5:     Compute $LCS(X[1, i], Y[1..j])$ and $LCS(X[i+1..m], Y[j+1..n]$ for all $j \in \{0, 1, \ldots, n\}$;
6:     Find $\min j$ subject to $\max_{0 \leq j \leq n} \{LCS(X[1, i], Y[1..j]) + LCS(X[i+1..m], Y[j+1..n])\}$;
7:     LinearLCS($i, j, X[1..i], Y[1..j]$);
8:     LinearLCS($m-i, n-j, X[i+1..m], Y[j+1..n]$);
```

partitions of the two subproblems. Algorithm LinearLCS formally describes the recursive stages of the space-saving strategy.

A middle vertex in a given problem can be found in the time proportional to the number of entries in the corresponding DP matrix. Proceeding in this way, all middle vertices can be found in $mn + \frac{mn}{2} + \frac{mn}{4} + \ldots \leq 2mn$ time. In addition, the approach requires no more than $2n$ space, which can be reused during the computation. Therefore, the following theorem is stated.

Theorem 2.2. *Algorithm* LinearLCS *solves the LCS problem in $O(mn)$ time and $O(n)$ space.*

2.3 Hunt-Szymanski Strategy

Figure 2.4 gives an example of the DP matrix $\mathcal{L}$ for the LCS problem over sequences $X =$ "`ACTGCCTAGGC`" and $Y =$ "`CGATCTGGAC`", which is computed by Equation 2.1. The circled positions indicates the occurrences of matches between X and Y. During the computation of $\mathcal{L}$, the values might increase only when a match between X and Y is encountered. For speeding up the computation, Hunt and Szymanski [45] provided an approach to merely calculating the values of the positions where X and Y match each other.

Let T be a two-dimensional array of threshhold values where $T[i, l]$ denotes the smallest j such that there exists a CS of length l between $X[i..i]$ and $Y[1..j]$. For example, in Figure 2.4 $\mathcal{L}[6, 4] = 2$ and $\mathcal{L}[6, 5] = 3$ yields $T[6, 3] = 5$. The monotonicity

			C	G	A	T	C	T	G	G	A	C
		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
A	1	0	0	0	①	1	1	1	1	1	①	1
C	2	0	①	1	1	1	②	2	2	2	2	②
T	3	0	1	1	1	②	2	③	3	3	3	3
G	4	0	1	②	2	2	2	3	④	④	4	4
C	5	0	①	2	2	2	③	3	4	4	4	⑤
C	6	0	①	2	2	2	③	3	4	4	4	⑤
T	7	0	1	2	2	③	3	④	4	4	4	5
A	8	0	1	2	③	3	3	4	4	4	⑤	5
G	9	0	1	②	3	3	3	4	⑤	⑤	5	5
G	10	0	1	②	3	3	3	4	⑤	⑥	6	6
C	11	0	①	2	3	3	④	4	5	6	6	⑦

Figure 2.4: Matches between two sequences in the DP matrix.

of T is stated in the following lemma [45].

Lemma 2.3. *If* $T[i, l]$, *for some* $i \in \{1, 2, \ldots, m\}$ *and* $l > 0$, *are defined,* $T[i, l-1] < T[i, l]$ *and* $T[i, l-1] < T[1+1, l] \leq T[i, l]$.

With the property shown in Lemma 2.3, the following recurrence formula can compute $T[i+1, l]$ from $T[1, l-1]$ and $T[i, l]$.

$$T[i+1, l] = \max \begin{cases} \text{smallest } j \text{ subject to } X[i+1] = Y[j] \\ \qquad\qquad\qquad \text{and } T[i, l-1] \leq j \leq T[i, l], \\ T[i, l] \text{ if no such } j \text{ exists.} \end{cases} \tag{2.2}$$

To compute the array T efficiently, sequences X and Y are preprocessed for building a linked list for each index of X. The linked list for index i of X keeps the corresponding indices j, in decreasing order, such that $x_i = y_j$. The stage of building m linked lists takes $O(m + n)$ over a finite alphabet.

With the positions of matches between X and Y, the approach proceeds to the matches row by row. Let $THRESH$ be a one-dimensional array where $THRESH[l]$ at the end of iteration i (i.e., the i-th linked list is processed) denotes the smallest index j such that there exists a CS of length l between $X[1..i]$ and $Y[1..j]$. The initialization of $THRESH$ is $THRESH[0] = 0$ and $THRESH[l] = n + 1$ for all $l \in \{1, 2, \ldots, m\}$. At iteration i, for each j in the i-th linked list, the value l is queried to satisfy $THRESH[l - 1] < j \leq THRESH[l]$ and $THRESH[l]$ maintains the smaller value between j and $THRESH[l]$. Specifically, l can be obtained in $O(\log\log n)$ time by employing a data structure named van Emde Boas tree [75, 76], which allows the operations of inserting, deleting, and testing membership of elements in the set $\{1, 2, \ldots, n\}$. The data structure requires $O(n \log\log n)$ time for initialization, and each such operation can be performed in $O(\log\log n)$ time. Therefore, the stage of calculating the array $THRESH$ takes totally $O(r \log\log n)$ time, where r denotes the total number of ordered pairs of positions at which X and Y match each other. Finally, the largest l subject to $THRESH[l] \neq n+1$ is the length of an LCS between X and Y. Thus, the following theorem is stated.

Theorem 2.3. *The LCS problem over a finite alphabet can be solved in* $O((r + n) \times \log\log n)$ *time and in* $O(r + n)$ *space.*

2.4 Previous Results

In 1974, Wagner and Fischer [77] proposed the problem of LCS. Since then, the problem has been studied intensively for decades. The results for the problem are summarized in Table 2.1. Apart from the above mentioned approaches, in 1977 Hirschberg [42] depicted an efficient representation of the DP matrix $\mathcal{L}$, which use the contours of $\mathcal{L}$ (see Figure 2.5 as an example) to specify the entire matrix. The contours are described by dominant matches. By finding the dominant matches in $O(ln)$ time with the time $O(n \log s)$ for processing the longer input sequence Y, Hirschberg solved the problem in $O(lm + n \log s)$ time, where l denotes the length of an LCS between X and Y, and s denotes the number of distinct characters in Y.

In 1980, Masek and Paterson [57] solve the problem in $O(n \times \max\{1, \frac{m}{\log n}\})$ time by using a "Four Russians" approach [16]. Two years ago, Nakatsu *et al.* [61] presented a $O(n \times (m - l))$-time algorithm, which would be faster than any previous one if l is close to m (i.e., X and Y are similar).

Later, Hsu and Du [43] gave a $O(lm \times \log \frac{n}{l} + lm)$-time algorithm by employing efficient merging methods in the computations. Myers [59] demonstrated that the LCS problem is equivalent to the problem of finding a shortest/longest path in the corresponding edit graph, and proposed a $O(nE)$-time algorithm for the latter problem, where E denotes the edit distance between X and Y. Apostolico and Guerra [13] improved the strategy proposed by Hunt and Szymanski. They introduced an auxiliary data structure named Characteristic Trees to process the dominant matches

Table 2.1: Previous results for the LCS problem

Year	Author(s)	Time Complexity	Space Complexity
1974	Wagner and Fischer [77]	$O(mn)$	$O(mn)$
1975	Hirschberg [41]	$O(mn)$	$O(n)$
1977	Hunt and Szymanski [45]†	$O((r+n)\log\log n)$	$O(r+n)$
1977	Hirschberg [42]	$O(ln+n\log s)$	$O(ln)$
1980	Masek and Paterson [57]	$O(n\times\max\{1,\frac{m}{\log n}\})$	$O(\frac{n^2}{\log n})$
1982	Nakatsu *et al.* [61]‡	$O(n\times(m-l))$	$O(m^2)$
1984	Hsu and Du [43]	$O(lm\times\log\frac{n}{l}+lm)$	$O(lm)$
1986	Myers [59]	$O(E\times(m+n))$	$O(E\times(m+n))$
1987	Apostolico and Guerra [13]	$O(m\log n+\mu\log\frac{2mn}{\mu})$	$O(\mu+n)$
2007	Iliopoulos and Rahman [47]†	$O(r\log\log n+n)$	$O(r+n)$

† Over a finite alphabet

‡ Similar input sequences

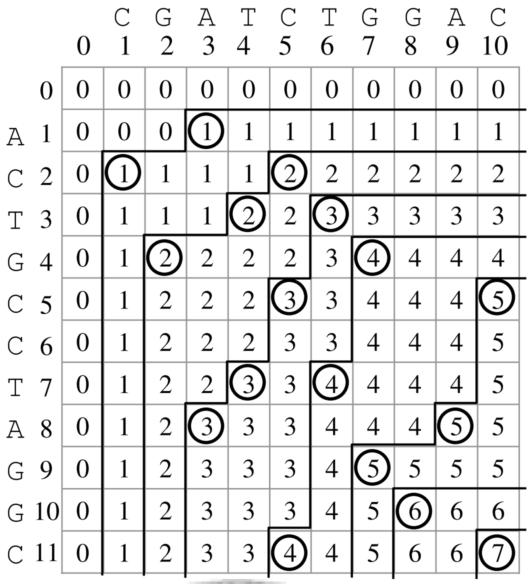
Figure 2.5: The contours of the DP matrix with dominant matches between two sequences.

between X and Y, and the improved algorithm requires $O(m \log n + \mu \log \frac{2mn}{\mu})$ time with $O(n \log s)$ preprocessing time, where μ denotes the number of dominant matches between X and Y, s denotes the number of distinct characters in Y

Recently, Iliopoulos and Rahman [47] proposed an $O(r \log\log n + n)$-time algorithm by employing the van Emde Boas tree [75, 76], where r denotes the number of matches between X and Y. It should be noted, however, that the preprocessing time of the van Emde Boas tree was not evaluated when analyzing the time complexity.

Chapter 3

Constrained LCSs

Functional and structural conservations of nucleic sequences play essential roles in the evolutionary molecular biology. With considerations of conserved segments, similarity analysis of nucleic sequences can provide some evidences for deduce the evolutionary relationships between species. On the other side, for two data (such as images and signals) including an essential message, similarity analysis of such data can provide some practical information for the applications if the continuity of the message is under consideration. For achieving these applications, the LCS problem with an inclusive constraint, named the inclusion-constrained longest common subsequence (abbreviated IC-LCS) problem, is considered. We address two IC-LCS problems as follows.

Problem 3.1. ***(SEQ-IC-LCS)*** *[72] Given two sequences X, Y and a constrained pattern P of lengths m, n, and d, respectively, the SEQ-IC-LCS problem is to find a* *longest sequence that is a CS of X and Y and includes P as a subsequence.*

Problem 3.2. ***(STR-IC-LCS)*** *Given two sequences X, Y and a constrained pattern P of lengths m, n, and d, respectively, the STR-IC-LCS problem is to find a longest sequence that is a CS of X and Y and includes P as a substring.*

For example, if $X =$ `AATGCCTAGGC`, $Y =$ `CGATCTGGAC`, and $P =$ `GTAC`, an LCS of X and Y is "`ATCTGGC`", and the outputs of the SEQ-IC-LCS and STR-IC-LCS problems are "`GCTAC`" and "`GTAC`", respectively.

Due to theoretical interests in the similarity measurement, we extend the definition of the IC-LCS problem to the LCS problem with an exclusive constraint, named the exclusion-constrained longest common subsequence (abbreviated EC-LCS) problem. We address two EC-LCS problems as follows.

Problem 3.3. ***(SEQ-EC-LCS)*** *Given two sequences X, Y and a constrained pattern P of lengths m, n, and d, respectively, the SEQ-EC-LCS problem is to find a longest sequence that is a CS of X and Y and excludes P as a subsequence.*

Problem 3.4. ***(STR-EC-LCS)*** *Given two sequences X, Y and a constrained pattern P of lengths m, n, and d, respectively, the STR-EC-LCS problem is to find a longest sequence that is a CS of X and Y and excludes P as a substring.*

For example, suppose that $X =$ `AATGCCTAGGC` and $Y =$ `CGATCTGGAC`. If $P =$ `TGC`, an output of the SEQ-EC-LCS problem is "`ATCTGG`". If $P =$ "`TG`", an output of the STR-EC-LCS problem is "`ATCGGC`".

Table 3.1: The CLCS problems

Problem	Input	Output	
SEQ-IC-LCS [72]	X, Y, and P	A longest sequence that is a common subsequence of X and Y and	includes P as a subsequence
STR-IC-LCS			includes P as a substring
SEQ-EC-LCS			excludes P as a subsequence
STR-EC-LCS			excludes P as a substring

The four variants of the LCS problem are summarized in Table 3.1. Throughout this chapter, the formats of the sequences and single constrained pattern are defined by $X = x_1x_2 \dots x_m$, $Y = y_1y_2 \dots y_n$, and $P = p_1p_2 \dots p_d$. The sections that follow use DP approaches to solve the four optimization problems. Section 3.1 introduces the previous results for the SEQ-IC-LCS problem. In Sections 3.2 to 3.4, we focus on the STR-IC-LCS, SEQ-EC-LCS, and STR-EC-LCS problems, respectively. Finally, we consider the four problems in the case of an arbitrary number of constrained patterns in Section 3.5.

3.1 Related Works on Problem SEQ-IC-LCS

The SEQ-IC-LCS problem was first addressed and solved in $O(m^2n^2d)$ time based on the DP technique by Tsai [72]. Later, Chin *et al.* [29] improved the time complexity of the problem from $O(m^2n^2d)$ to $O(mnd)$, and showed that this problem is equivalent to a special case of the constrained multiple sequence alignment problem [28, 70].

Meanwhile, Arslan and Eğecioğlu [17] also presented an improved algorithms with $O(mnd)$ time, and extend the definition of the problem such that the resulting LCS contains a subsequence whose edit distance from the constrained pattern is less than a given positive integer parameter.

Without loss of generality, assume that $m \leq n$. Recently, Iliopoulos and Rahman [46] proposed an algorithm by adapting Hunt-Szymanski strategy and employing a data structure named bounded heap, which is introduced in Section 4.2. The executing of the algorithm requires $O(dr \times \log\log n + n)$ time, where r is the total number of ordered pairs of positions at which X and Y match each other. It should be noted, however, that the preprocessing time of the bounded heap was not evaluated when the time complexity was analyzed.

Here we introduce an $O(mnd)$-time algorithm [29] for the problem as follows. Lemma 3.1 decomposes the structure of an optimal solution based on the solutions to its smaller subproblems.

Lemma 3.1. *Let $\mathcal{Z}_{i,j,k}$ denote the set of all longest sequences which are CSs of $X[1..i]$ and $Y[1..j]$ and include $P[1..k]$ as a subsequence. If $Z = z_1z_2\ldots z_l \in \mathcal{Z}_{i,j,k}$, the following conditions hold:*

(1) *If $x_i = y_j = p_k$ when $k > 0$, then $z_l = x_i = y_j = p_k$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k-1}$.*

(3) *If $x_i = y_j$, and ($x_i \neq p_k$ or $k = 0$), then $z_l = x_i = y_j$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.*

(4) *If $x_i \neq y_j$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j,k}$.*

(5) *If $x_i \neq y_j$, then $z_l \neq y_j$ implies $Z[1..l] \in \mathcal{Z}_{i,j-1,k}$.*

Let $\mathcal{L}[i, j, k]$ denote the length of a sequence that belongs to $\mathcal{Z}_{i,j,k}$. By the optimality principles of the SEQ-IC-LCS problem shown in Lemma 3.1, the following recursive formula can be derived, where $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, n\}$, and $k \in \{0, 1, \ldots, d\}$.

$$\mathcal{L}[i, j, k] = \begin{cases} 1 + \mathcal{L}[i-1, j-1, k-1]) & \text{if } k > 0 \text{ and } x_i = y_j = p_k, \\ 1 + \mathcal{L}[i-1, j-1, k] & \text{if } x_i = y_j \text{ and } (x_i \neq p_k \text{ or } k = 0), \\ \max\{\mathcal{L}[i-1, j, k], & \\ \qquad \mathcal{L}[i, j-1, k]\} & \text{if } x_i \neq y_j. \end{cases} \tag{3.1}$$

The boundary conditions of this recurrence are $\mathcal{L}[i, 0, 0] = \mathcal{L}[0, j, 0] = 0$ and $\mathcal{L}[0, j, k] = \mathcal{L}[i, 0, k] = -\infty$ for any $i \in \{0, 1, \ldots, m\}$, $j \in \{0, 1, \ldots, n\}$, and $k \in \{1, 2, \ldots, d\}$. Based on Equation (3.1), each entry in matrix $\mathcal{L}$ can be computed.

Suppose that Z is a sequence belonging to $\mathcal{Z}_{m,n,d}$ and is initially an empty sequence. The length of Z is given by $\mathcal{L}[m, n, d]$, which requires $O(mnd)$ computation time. In addition, sequence Z can be constructed by backtracking through the computation path from $\mathcal{L}[m, n, d]$ to $\mathcal{L}[0, 0, 0]$, and recovering the computation path of Z takes $O(m + n + d)$ steps. Consequently, the following theorem is stated.

Theorem 3.1. *The SEQ-IC-LCS problem can be solved in $O(mnd)$ time and space.*

3.2 Problem STR-IC-LCS

The STR-IC-LCS problem is to find a longest sequence that is a CS of two sequences X and Y and includes a constrained pattern P as a substring. Property 3.1 shows the characterization of the structure of a solution to the STR-IC-LCS problem.

Property 3.1. *If $Z[1..l]$ is a longest sequence which is a CS of $X[1..m]$ and $Y[1..n]$ and includes $P[1..d]$ as the substring $Z[l'-d+1..l']$ for some $l' \in \{d, d+1, \ldots, l\}$, then $Z[1..l]$ is a concatenation of the following two substrings for some $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$:*

1. *The prefix $Z[1..l']$ (i.e., Z_1): $Z[1..l']$ is a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ and includes $P[1..d]$ as the suffix $Z[l'-d+1..l']$, and*
2. *The suffix $Z[l'+1..l]$ (i.e., Z_2): $Z[l'+1..l]$ is an LCS of $X[i+1..m]$ and $Y[j+1..n]$.*

Figure 3.1 illustrates the idea of the problem decomposition shown in Property 3.1. In view of the idea, we solve the problems of computing a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ and includes $P[1..d]$ as the suffix $Z[l'-d+1..l']$, and of calculating an LCS of $X[i..m]$ and $Y[j..n]$, for all $i \in \{1, 2, \ldots, m\}$ and $j \in \{1, 2, \ldots, n\}$. The solutions to the two subproblems are subsequently merged to determine a longest concatenation. The former subproblem can be computed in quadratic time by employing the algorithm shown in Section 2.1. For solving the latter subproblem, Lemma 3.2 decomposes the structure of an optimal solution based on the solutions to its smaller

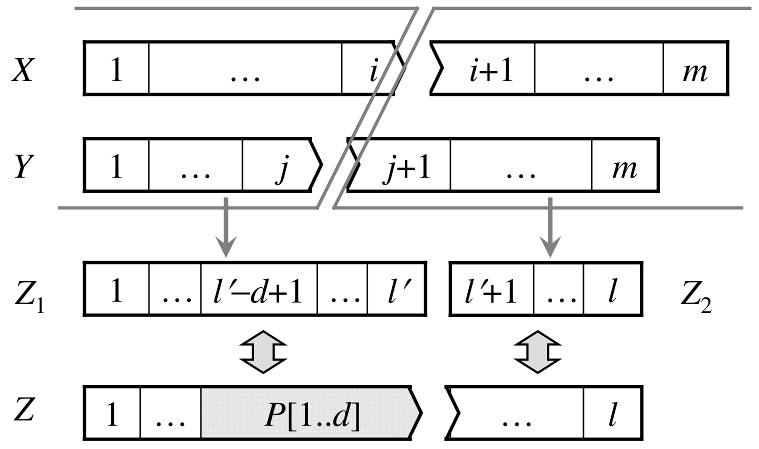

Figure 3.1: A problem decomposition of Problem STR-IC-LCS.

subproblems.

Lemma 3.2. *Let $\mathcal{Z}_{i,j,k}$ denote the set of all longest sequences which are CSs of $X[1..i]$ and $Y[1..j]$ and include $P[1..k]$ as a suffix. If $Z = z_1z_2\ldots z_l \in \mathcal{Z}_{i,j,k}$, the following conditions hold:*

(1) *If $x_i = y_j = p_k$ when $k > 0$, then $z_l = x_i = y_j = p_k$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k-1}$.*

(2) *If $x_i = y_j$ and $x_i \neq p_k$ when $k > 0$, then $z_l \neq x_i$ and $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$.*

(3) *If $x_i = y_j$ when $k = 0$, then $z_l = x_i = y_j$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.*

(4) *If $x_i \neq y_j$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j,k}$.*

(5) *If $x_i \neq y_j$, then $z_l \neq y_j$ implies $Z[1..l] \in \mathcal{Z}_{i,j-1,k}$.*

Proof. We prove this lemma case by case. (1) Since $P[1..k]$ is a suffix of $Z[1..l]$, we have $z_l = p_k$. If $z_l \neq x_i$, we could append $x_i = y_j = p_k$ to $Z[1..l-1]$ obtain a CS of length l, and $P[1..k]$ is also a suffix of the resulting sequence. Thus, $Z[1..l-1]$ is a CS of $X[1..i-1]$ and $Y[1..j-1]$ such that $P[1..k-1]$ is the suffix $Z[l-k+1..l-1]$. Assume by contradiction that there exists a CS $Z'[1..l]$ of $X[1..i-1]$ and $Y[1..j-1]$ such that $P[1..k-1]$ is the suffix $Z'[l-k+2..l]$. We could append $x_i = y_j = p_k$ to $Z'[1..l]$ for obtaining a CS of $X[1..i]$ and $Y[1..j]$ of length greater than l such that $P[1..k]$ is the suffix $Z'[l-k+2..l+1]$, which contradicts the hypothesis of $Z[1..l] \in \mathcal{Z}_{i,j,k}$.

(2) If $z_l = x_i$, then $z_l \neq p_k$ and $P[1..k]$ is not a suffix of $Z[1..l]$. Therefore, we can conclude that $z_l \neq x_i$ and $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$. Assume by contradiction that there exists a CS $Z'[1..l+1]$ of $X[1..i-1]$ and $Y[1..j-1]$ such that $P[1..k]$ is a suffix of $Z'[1..l+1]$. Obviously, $Z'[1..l+1]$ is also a CS of $X[1..i]$ and $Y[1..j]$ of length greater than l such that $P[1..k]$ is a suffix. This contradicts the hypothesis of $Z[1..l] \in \mathcal{Z}_{i,j,k}$.

(3) This case is a special case of the LCS problem for $x_i = y_j$. Thus, it is obvious that $z_l = x_i = y_j$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.

(4) Because $z_l \neq x_i$, $Z[1..l]$ is a CS of $X[1..i-1]$ and $Y[1..j]$ such that $P[1..k]$ is the suffix $Z[l-k+1..l]$. Similar to proof of Case (2), we have $Z[1..l] \in \mathcal{Z}_{i-1,j,k}$. Moreover, the proof of Case (5) is similar to the proof of this case. □

Let $\mathcal{L}[i,j,k]$ denote the length of a sequence that belongs to $\mathcal{Z}_{i,j,k}$. By the optimality principles of the STR-IC-LCS problem shown in Lemma 3.2, we have the following recurrence, where $i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,n\}$, and $k \in \{0,1,\ldots,d\}$.

$$\mathcal{L}[i,j,k] = \begin{cases} 1 + \mathcal{L}[i-1, j-1, k-1] & \text{if } k > 0 \text{ and } x_i = y_j = p_k, \\ \mathcal{L}[i-1, j-1, k] & \text{if } k > 0,\ x_i = y_j, \text{ and } x_i \neq p_k, \\ 1 + \mathcal{L}[i-1, j-1, k] & \text{if } k = 0 \text{ and } x_i = y_j, \\ \max\{\mathcal{L}[i-1, j, k], & \\ \qquad \mathcal{L}[i, j-1, k]\} & \text{if } x_i \neq y_j. \end{cases} \tag{3.2}$$

The boundary conditions of this recursive formula are $\mathcal{L}[i,0,0] = \mathcal{L}[0,j,0] = 0$ and $\mathcal{L}[0,j,k] = \mathcal{L}[i,0,k] = -\infty$ for any $i \in \{0,1,\ldots,m\}$, $j \in \{0,1,\ldots,n\}$, and $k \in \{1,2,\ldots,d\}$. Based on Equation (3.2), each entry in matrix $\mathcal{L}$ can be computed, and the computation time requires $O(mnd)$.

Let $S[i,j]$ denote the length of an LCS of $X[i..m]$ and $Y[j..n]$ for $i \in \{1,2,\ldots,m\}$ and $j \in \{1,2,\ldots,n\}$. If $i = m+1$ or $j = n+1$, we set $S[i,j] = 0$. All entries in matrix S can be computed by employing an $O(mn)$-time algorithm for the LCS problem.

Suppose that Z is a longest sequence which is a CS of X and Y and includes P as a substring, and is initially an empty sequence. We define a two-dimensional array $\mathcal{W}$ by $\mathcal{W}[i,j] = \mathcal{L}[i,j,d] + S[i+1,j+1]$ for any $i \in \{1,2,\ldots,m\}$ and $j \in \{1,2,\ldots,n\}$. According to Property 3.1, the length of Z is given by the maximum value of $\mathcal{W}$, which can be computed in $O(mn)$ time. Suppose that the maximum value in $\mathcal{W}$ is supplied from entry $\mathcal{W}[i^*,j^*]$ for some $i^* \in \{1,2,\ldots,m\}$ and $j^* \in \{1,2,\ldots,n\}$. Let Z_1 denote a sequence that belongs to $\mathcal{Z}_{i^*,j^*,d}$ and Z_2 denote an LCS of $X[i^*+1..m]$

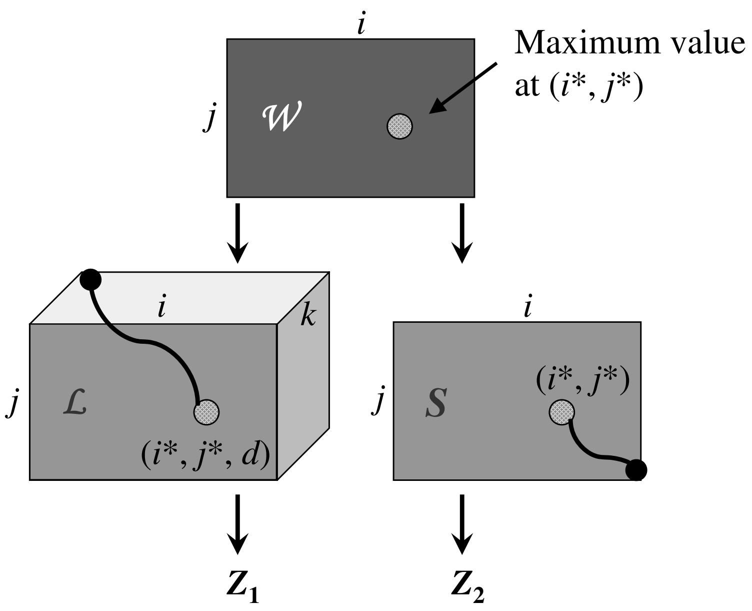

Figure 3.2: Deriving solutions to the two subproblems of Problem STR-IC-LCS.

and $Y[j^*+1..n]$. The idea of constructing Z_1 and Z_2 is illustrated in Figure 3.2. We construct Z_1 and Z_2 by backtracking through the computation paths from $\mathcal{L}[i^*, j^*, d]$ to $\mathcal{L}[0, 0, 0]$ and from $S[i^*+1, j^*+1]$ to $S[m+1, n+1]$, respectively. Finally, we obtain Z by concatenating Z_1 and Z_2. Recovering the computation paths of Z_1 and Z_2 take $O(m+n+d)$ and $O(m+n)$ steps, respectively. Consequently, the following theorem is stated.

Theorem 3.2. *The STR-IC-LCS problem can be solved in $O(mnd)$ time and space.*

3.3 Problem SEQ-EC-LCS

The SEQ-EC-LCS problem is to find a longest sequence that is a CS of two sequences X and Y and excludes a constrained pattern P as a subsequence. Lemma 3.3 decomposes the structure of an optimal solution based on the solutions to its smaller subproblems.

Lemma 3.3. *Let $\mathcal{Z}_{i,j,k}$ denote the set of all longest sequences which are CSs of $X[1..i]$ and $Y[1..j]$ and exclude $P[1..k]$ as a subsequence. If $Z = z_1 z_2 \dots z_l \in \mathcal{Z}_{i,j,k}$, the following conditions hold:*

(1) *If $x_i = y_j = p_k$ when $k = 1$, then $z_l \neq x_i$ and $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$.*

(2) *If $x_i = y_j = p_k$ when $k \geq 2$, then $z_l = x_i = y_j = p_k$ implies $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k-1}$.*

(3) *If $x_i = y_j = p_k$ when $k \geq 2$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$.*

(4) *If $x_i = y_j$ and ($x_i \neq p_k$ when $k > 0$, or $k = 0$), then $z_l = x_i = y_j$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.*

(5) *If $x_i \neq y_j$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j,k}$.*

(6) *If $x_i \neq y_j$, then $z_l \neq y_j$ implies $Z[1..l] \in \mathcal{Z}_{i,j-1,k}$.*

Proof. The proof is similar to Lemma 3.2. □

Let $\mathcal{L}[i,j,k]$ denote the length of a sequence that belongs to $\mathcal{Z}_{i,j,k}$. By the optimality principles of the SEQ-EC-LCS problem shown in Lemma 3.3, we derive the following recurrence, where $i \in \{1, 2, \dots, m\}$, $j \in \{1, 2, \dots, n\}$, and $k \in \{0, 1, \dots, d\}$.

$$\mathcal{L}[i,j,k] = \begin{cases} \mathcal{L}[i-1,j-1,k] & \text{if } k=1 \text{ and } x_i = y_j = p_k, \\ \max\{\mathcal{L}[i-1,j-1,k], & \\ \quad 1+\mathcal{L}[i-1,j-1,k-1]\} & \text{if } k>2 \text{ and } x_i = y_j = p_k, \\ 1+\mathcal{L}[i-1,j-1,k] & \text{if } x_i = y_j \text{ and} \\ & \quad (k=0, \text{ or } k>0 \text{ and } x_i \neq p_k), \\ \max\{\mathcal{L}[i-1,j,k], & \\ \quad \mathcal{L}[i,j-1,k]\} & \text{if } x_i \neq y_j. \end{cases} \tag{3.3}$$

The boundary conditions of this recursive formula are $\mathcal{L}[i,0,k] = \mathcal{L}[0,j,k] = 0$ for any $i \in \{0,1,\ldots,m\}$, $j \in \{0,1,\ldots,n\}$, and $k \in \{0,1,\ldots,d\}$. Based on Equation (3.3), each entry in matrix $\mathcal{L}$ can be computed.

Suppose that Z is a sequence that belongs to $\mathcal{Z}_{m,n,d}$ and is initially an empty sequence. The length of Z is given by $\mathcal{L}[m,n,d]$, which requires $O(mnd)$ computation time. Sequence Z can be constructed by backtracking through the computation path from $\mathcal{L}[m,n,d]$ to $\mathcal{L}[0,0,0]$. Recovering the computation path of Z takes $O(m+n+d)$ steps. Consequently, the following theorem is stated.

Theorem 3.3. *The SEQ-EC-LCS problem can be solved in $O(mnd)$ time and space.*

3.4 Problem STR-EC-LCS

The STR-EC-LCS problem is to find a longest sequence that is a CS of two sequences X and Y and excludes a constrained pattern P as a substring. Lemma 3.4 decomposes the structure of an optimal solution based on the solutions to its smaller subproblems.

Lemma 3.4. *Let $\mathcal{Z}_{i,j,k}$ denote the set of all longest sequences which are CSs of $X[1..i]$ and $Y[1..j]$ and exclude $P[1..k]$ as a substring. If $Z = z_1 z_2 \ldots z_l \in \mathcal{Z}_{i,j,k}$, the following conditions hold:*

(1) *If $x_i = y_j = p_k$ when $k = 1$, then $z_l \neq x_i$ and $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$.*

(2) *If $x_i = y_j = p_k$ when $k \geq 2$, then $z_l = x_i = y_j = p_k$ and $z_{l-1} = p_{k-1}$ implies $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k-1}$.*

(3) *If $x_i = y_j = p_k$ when $k \geq 2$, then $z_l = x_i = y_j = p_k$ and $z_{l-1} \neq p_{k-1}$ implies $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.*

(4) *If $x_i = y_j = p_k$ when $k \geq 2$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j-1,k}$.*

(5) *If $x_i = y_j$ and ($x_i \neq p_k$ when $k > 0$, or $k = 0$), then $z_l = x_i = y_j$ and $Z[1..l-1] \in \mathcal{Z}_{i-1,j-1,k}$.*

(6) *If $x_i \neq y_j$, then $z_l \neq x_i$ implies $Z[1..l] \in \mathcal{Z}_{i-1,j,k}$.*

(7) *If $x_i \neq y_j$, then $z_l \neq y_j$ implies $Z[1..l] \in \mathcal{Z}_{i,j-1,k}$.*

Proof. The proof is similar to Lemma 3.3. □

Let $\mathcal{L}[i,j,k]$ denote the length of a sequence that belongs to $\mathcal{Z}_{i,j,k}$. By the optimality principles of the STR-EC-LCS problem shown in Lemma 3.4, we derive the following recursive formula, where $i \in \{1,2,\ldots,m\}$, $j \in \{1,2,\ldots,n\}$, and $k \in \{0,1,\ldots,d\}$.

$$\mathcal{L}[i,j,k] = \begin{cases} \mathcal{L}[i-1,j-1,k] & \text{if } k=1 \text{ and } x_i = y_j = p_k, \\ \max\{1+\mathcal{L}[i-1,j-1,k-1]), & \\ \quad 1+\mathcal{L}[i-1,j-1,k]\} & \text{if } k \geq 2 \text{ and } x_i = y_j = p_k, \\ 1+\mathcal{L}[i-1,j-1,k] & \text{if } x_i = y_j \text{ and } (k=0, \text{ or} \\ & \quad k>0 \text{ and } x_i \neq p_k), \\ \max\{\mathcal{L}[i-1,j,k], & \\ \quad \mathcal{L}[i,j-1,k]\} & \text{if } x_i \neq y_j. \end{cases} \tag{3.4}$$

The boundary conditions of this recursive formula are $\mathcal{L}[i,0,k] = \mathcal{L}[0,j,k] = 0$ for any $i \in \{0,1,\ldots,m\}$, $j \in \{0,1,\ldots,n\}$, and $k \in \{0,1,\ldots,d\}$. Based on Equation (3.4), each entry in matrix $\mathcal{L}$ can be computed.

Suppose that Z is a sequence that belongs to $\mathcal{Z}_{m,n,d}$ and is initially an empty sequence. The length of Z is given by $\mathcal{L}[m,n,d]$, which requires $O(mnd)$ computation time. Moreover, sequence Z can be constructed by backtracking through the computation path from $\mathcal{L}[m,n,d]$ to $\mathcal{L}[0,0,0]$. Recovering the computation path of Z takes $O(m+n+d)$ steps. Consequently, the following theorem is stated.

Theorem 3.4. *The STR-EC-LCS problem can be solved in $O(mnd)$ time and space.*

3.5 Problem CLCS with an Arbitrary Number of Constrained Patterns

In this section, we consider the four variants of the CLCS problem in each of which the input is two sequences X, Y and w constrained patterns P_i of lengths m, n, and d_i for $i \in \{1, 2, \ldots, w\}$, respectively. Gotthilf *et al.* [35] has showed that the SEQ-IC-LCS problem with an arbitrary number of constrained patterns is NP-complete, and it does not have a polynomial-time approximation scheme (PTAS). In the following, we first introduce Gotthilf *et al.*'s proof for the SEQ-IC-LCS problem, and then demonstrate that the STR-IC-LCS and SEQ-EC-LCS problems with an arbitrary number of constrained patterns are also NP-complete. Furthermore, we give an exact algorithm for each problem.

3.5.1 Hardness

Problem SEQ-IC-LCS

The decision version of the SEQ-IC-LCS problem is in NP. Given two sequences X, Y and w constrained patterns P_i over Σ of lengths m, n, and d_i for $i \in \{1, 2, \ldots, w\}$, the problem is to determine if there is a CS of X and Y with length at least ℓ such that it includes each P_i as a subsequence. A proof certificate can simply be a ordered sequence of the characters in Σ, and verification can be clearly done in polynomial time by a deterministic Turing machine. It simply checks if the sequence is a CS X

and Y with length at least ℓ and if every P_i is a subsequence of the sequence.

A nondeterministic Turing machine can find such a sequence as follows. At each character of X, the machine either selects or refuses the character into a sequence, until all characters of X has be selected or refused. At the end it verifies in polynomial time that the sequence it has taken is a CS of Y with length at least ℓ and every P_i is a subsequence of the sequence.

Gotthilf *et al.* [35] showed the hardness of the problem by a reduction from the 3-SAT problem.

Lemma 3.5. *The 3-SAT problem is reducible to SEQ-IC-LCS problem in polynomial time.*

Proof. Let $F = \bigwedge_{i\in\{1,2,\ldots,l\}} C_i$ be a propositional formula with exact three literals in CNF and v_i, $i \in \{1, 2, \ldots, b\}$, be the variables in F. An instance of the SEQ-IC-LCS problem, containing two sequences X, Y and $b + l - 1$ constrained patterns P_1, P_2, ..., P_{b+l-1}, is constructed from F as follows.

The alphabet of the sequences in the SEQ-IC-LCS problem is the union of the set of clauses $\{C_i : 1 \leq i \leq l\}$ and a set of separators $\{s_i : 1 \leq i \leq b - 1\}$ for the clauses. For each variable v_i, a substring X_i is constructed by orderly concatenating all the clauses satisfied with $v_i = true$ with all the clauses satisfied with $v_i = false$. Subsequently, sequence X is set to be $X_1s_1X_2s_2 \ldots s_{b-1}X_b$, where substrings X_i and X_{i+1} are separated by a separator s_i for all $i \in \{1, 2, \ldots, b - 1\}$.

Similarly, for each variable v_i, a substring Y_i is constructed by orderly concatenating all the clauses satisfied with $v_i = false$ with all the clauses satisfied with $v_i = true$, and then sequence Y is set to be $Y_1 s_1 Y_2 s_2 \dots s_{b-1} Y_b$, where substrings Y_i and Y_{i+1} are separated by a separator s_i for all $i \in \{1, 2, \dots, b-1\}$. Furthermore, each element in $\{C_i : 1 \le i \le l\}$ and $\{s_i : 1 \le i \le b-1\}$ forms a constrained pattern of length one.

Claim. *F is satisfiable if and only if the SEQ-IC-LCS problem has a solution of length at least $b + l - 1$.*

Proof of Claim. For simplicity, assume that there exists at least one clause including either v_i or $\bar{v}_i$.

($\Rightarrow$) If F is satisfiable, let A be an assignment satisfying F, and let A_T and A_F denote the sets of variables assigned *true* and *false* in A, respectively. For every variable $v_i \in A_T$, the clauses satisfied by setting $v_i = ture$ are added from X_i and Y_i to a sequence Z. Also, for every variable $v_i \in A_F$, the clauses satisfied by setting $v_i = false$ are added from X_i and Y_i to Z. Besides, all separators s_i, $1 \in \{1, 2, \dots b-1\}$, are appropriately added to Z.

Because v_i is assigned to either *ture* or $false$, no internal crossings exists within X_i and Y_i, and no crossing exists over the separators. Obviously, all clauses are satisfied and all separators are added to Z; consequently, Z, a CS of X and Y with length at least $b + l - 1$, includes all constrained patterns as subsequences. Therefore, there exists a longest sequence that is a CS of X and Y with length at least $b + l - 1$ and includes every constrained pattern as a subsequence.

($\Leftarrow$) If there exists a longest sequence that is a CS of X and Y and includes every constrained pattern as a subsequence, C_i for all $i \in \{1, 2, \ldots, l\}$ and s_i for all $i \in \{1, 2, \ldots, b-1\}$ must be included in Z. Because every s_i, $i \in \{1, 2, \ldots, b-1\}$, is a separator between X_i and X_{i+1} (also between Y_i and Y_{i+1}), any clause appearing in X_i must be within Y_i, and vice versa. If some clause appears in X_i, there exists an assignment of v_i such that it satisfies the clause. Therefore, there exists an assignment of all variables satisfying F. $\square$

Example 3.1. *Suppose that* $F = (v_1 \vee v_2 \vee \bar{v_4}) \wedge (\bar{v_2} \vee \bar{v_3} \vee v_5) \wedge (\bar{v_1} \vee \bar{v_3} \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_5})$. *The construction of Lemma 3.5 yields an instance of the SEQ-IC-LCS problem as follows.*

$X = C_1C_4C_3s_1C_1C_2C_4s_2C_2C_3s_3C_3C_1s_4C_2C_4,$

$Y = C_3C_1C_4s_1C_2C_4C_1s_2C_2C_3s_3C_1C_3s_4C_4C_2,$

$P_1 = C_1,$

$P_2 = C_2,$

$P_3 = C_3,$

$P_4 = C_4,$

$P_5 = s_1,$

$P_6 = s_2,$

$P_7 = s_3,$ *and*

$P_8 = s_4.$

Consider the assignment with $\{v_1 = ture, v_2 = ture, v_3 = false, v_4 = true, v_5 = flase\}$. *There exists a CS* $C_1C_4s_1C_1s_2C_2C_3s_3C_3s_4C_4$ *between* X *and* Y *and includes*

$P_1, P_2, \ldots, P_8$ as subsequences. On the other hand, consider the solution of the SEQ-IC-LCS problem with $C_1C_4s_1C_2C_4s_2C_2C_3s_3C_3s_4C_2$. By setting $v_1 = ture$, $v_2 = false$, $v_3 = false$, $v_4 = true$, and $v_5 = ture$, F is satisfied.

Because the decision version of the SEQ-IC-LCS problem is in NP, and the 3-SAT problem can be reduced to it, we conclude the following theorem.

Theorem 3.5. *The SEQ-IC-LCS problem in case of an arbitrary number of constrained patterns is NP-complete.*

Problem STR-IC-LCS

The decision version of the STR-IC-LCS problem is defined as follows. Given two sequences X, Y and w constrained patterns P_i over Σ of lengths m, n, and d_i for $i \in \{1, 2, \ldots, w\}$, the problem is to determine if there is a CS of X and Y with length at least ℓ such that it includes each P_i as a substring. We can demonstrate that the problem is in NP by a verification similar to the SEQ-IC-LCS problem. In the following we show the hardness of the STR-IC-LCS problem by a reduction from the 3-SAT problem.

Lemma 3.6. *The 3-SAT problem is reducible to STR-IC-LCS problem in polynomial time.*

Proof. Let $F = \bigwedge_{i\in\{1,2,\ldots,l\}} C_w$ be a propositional formula with exact three literals in CNF and v_i, $i \in \{1, 2, \ldots, b\}$, be the variables in F. An instance of the STR-IC-LCS problem, containing two sequences X, Y and w constrained patterns P_1, P_2, $\ldots$, P_w, is constructed from F as follows.

The alphabet of the sequences in the STR-IC-LCS problem is the union of the set of clauses $\{C_i : 1 \leq i \leq w\}$ and a set of separators $\{s_i : 1 \leq i \leq b-1\}$ for the clauses. For each variable v_i, a substring X_i is constructed by concatenating twice all the clauses satisfied with $v_i = true$ with all the clauses satisfied with $v_i = false$ in order. Subsequently, sequence X is set to be $X_1 s_1^{2w} X_2 s_2^{2w} \ldots s_{b-1}^{2w} X_b$, where substrings X_i and X_{i+1} are separated by a unique character s_i of length $2w$, denoted by s_i^{2w}, for all $i \in \{1, 2, \ldots, b-1\}$.

Similarly, for each variable v_i, a substring Y_i is constructed by concatenating twice all the clauses satisfied with $v_i = false$ with all the clauses satisfied with $v_i = true$ in order, and then sequence Y is set to be $Y_1 s_1^{2w} Y_2 s_2^{2w} \ldots s_{b-1}^{2w} Y_b$, where substrings Y_i and Y_{i+1} are separated by a unique character s_i of length $2w$, denoted by s_i^{2w}, for all $i \in \{1, 2, \ldots, b-1\}$. Furthermore, each element in $\{C_i : 1 \leq i \leq w\}$ forms a constrained pattern which contains a unique character C_i of length two, denoted by C_i^2.

Claim. *F is satisfiable if and only if the STR-IC-LCS problem has a solution of length at least $2wb$.*

Proof of Claim. For simplicity, assume that there exists at least one clause including either v_i or $\bar{v}_i$.

($\Rightarrow$) If F is satisfiable, let A be an assignment satisfying F, and let A_T and A_F denote the sets of variables assigned $true$ and $false$ in A, respectively. For every variable $v_i \in A_T$, the clauses satisfied by setting $v_i = ture$ are added twice from X_i and Y_i to a sequence Z. Likewise, for every variable $v_i \in A_F$, the clauses satisfied by setting $v_i = false$ are added twice from X_i and Y_i to Z. Besides, the separators s_i^{2w}, $1 \in \{1, 2, \dots b-1\}$, are added to appropriate positions in Z.

Because any variable v_i is assigned to either $ture$ or $false$, no internal crossings exists within X_i and Y_i. In addition, no crossing exists over the separators because the number of clauses within X_i and Y_i are not more than $2w$. Obviously, all clauses are satisfied and all separators are added to Z; consequently, Z, a CS of X and Y of length at least $2wb$, includes all constrained patterns as substrings. Therefore, there exists a longest sequence which is a CS of X and Y with length at least $2wb$ and includes every constrained pattern as a substring.

($\Leftarrow$) If there exists a longest sequence that is a CS of X and Y and includes every constrained pattern as a substring, C_i^2 for all $i \in \{1, 2, \dots, w\}$ must be included in Z. Because there are no more than $2w$ elements separately within X_i and Y_i, every s_i^{2w}, $i \in \{1, 2, \dots, b-1\}$, is a separator between X_i and X_{i+1} (also between Y_i and Y_{i+1}). Hence, any clause appearing in X_i must be within Y_i, and vice versa. If some clause appears in X_i, there exists an assignment of v_i such that it satisfies the clause. Therefore, there exists an assignment of all variables satisfying F. $\square$

Example 3.2. *Suppose that* $F = (v_1 \vee v_2 \vee \bar{v_4}) \wedge (\bar{v_2} \vee \bar{v_3} \vee v_5) \wedge (\bar{v_1} \vee \bar{v_3} \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_5})$. *The construction of Theorem 3.6 yields an instance of the SEQ-IC-LCS problem as follows.*

$X = C_1^2C_4^2C_3^2s_1^8C_1^2C_2^2C_4^2s_2^8C_2^2C_3^2s_3^8C_3^2C_1^2s_4^8C_2^2C_4^2,$

$Y = C_3^2C_1^2C_4^2s_1^8C_2^2C_4^2C_1^2s_2^8C_2^2C_3^2s_3^8C_1^2C_3^2s_4^8C_4^2C_2^2,$

$P_1 = C_1^2,$

$P_2 = C_2^2,$

$P_3 = C_3^2,$ *and*

$P_4 = C_4^2.$

Consider the assignment with $\{v_1 = ture, v_2 = ture, v_3 = false, v_4 = true, v_5 = flase\}$. *There exists a CS* $C_1^2C_4^2s_1^8C_1^2s_2^8C_2^2C_3^2s_3^8C_3^2s_4^8C_3^2$ *between* X *and* Y *that includes* $P_1, P_2, \ldots, P_4$ *as substrings. On the other hand, consider the solution of the STR-IC-LCS problem with* $C_1^2C_4^2s_1^8C_2^2C_4^2s_2^8C_2^2C_3^2s_3^8C_1^2s_4^8C_2^2$. *By setting* $v_1 = ture$, $v_2 = false$, $v_3 = false$, $v_4 = false$, *and* $v_5 = ture$, *F is satisfied.*

Because the decision version of the STR-IC-LCS problem is in NP, and the 3-SAT problem can be reduced to it, we conclude the following theorem.

Theorem 3.6. *The STR-IC-LCS problem in case of an arbitrary number of constrained patterns is NP-complete.*

Problem SEQ-EC-LCS

The decision version of the SEQ-EC-LCS problem is defined as follows. Given two sequences X, Y and w constrained patterns P_i over Σ of lengths m, n, and d_i for $i \in \{1, 2, \ldots, w\}$, the problem is to determine if there is a CS of X and Y with length at least ℓ such that it excludes each P_i as a subsequence. We can demonstrate that the problem is in NP by a verification similar to the SEQ-IC-LCS problem. In the following we show the hardness of the SEQ-EC-LCS problem by a reduction from the 3-SAT problem.

Lemma 3.7. *The 3-SAT problem is reducible to SEQ-EC-LCS problem in polynomial time.*

Proof. Let $F = \bigwedge_{i\in\{1,2,\ldots,l\}} C_w$ be a propositional formula with exact three literals in CNF and v_i, $i \in \{1, 2, \ldots, b\}$, be the variables in F. An instance of the STR-EC-LCS problem, containing two sequences X, Y and w constrained patterns P_1, P_2, …, P_w, is constructed from F as follows.

The alphabet of the sequences in the SEQ-EC-LCS problem is the union of the set of clauses $\{C_i : 1 \le i \le w\}$ and a set of separators $\{s_i : 1 \le i \le b-1\}$ for the clauses. For each variable v_i, a substring X_i is constructed by orderly concatenating all the clauses satisfied with $v_i = true$ with all the clauses satisfied with $v_i = false$. Subsequently, sequence X is set to be $X_1 s_1^w X_2 s_2^w \ldots s_{b-1}^w X_b$, where substrings X_i and X_{i+1} are separated by a unique character s_i of length w, denoted by s_i^w, for all $i \in \{1, 2, \ldots, b-1\}$.

Similarly, for each variable v_i, a substring Y_i is constructed by orderly concatenating all the clauses satisfied with $v_i = false$ with all the clauses satisfied with $v_i = true$, and then sequence Y is set to be $Y_1 s_1^w Y_2 s_2^w \ldots s_{b-1}^w Y_b$, where substrings Y_i and Y_{i+1} are separated by a unique character s_i of length w, denoted by s_i^w, for all $i \in \{1, 2, \ldots, b-1\}$. Furthermore, each element in $\{C_i : 1 \le i \le w\}$ forms a constrained pattern which contains a unique character C_i of length two, denoted by C_i^2.

Claim. *F is satisfiable if and only if the SEQ-EC-LCS problem has a solution of length wb.*

Proof of Claim. For simplicity, assume that there exists at least one clause including either v_i or $\bar{v}_i$.

($\Rightarrow$) If F is satisfiable, let A be an assignment satisfying F, and let A_T and A_F denote the sets of variables assigned $true$ and $false$ in A, respectively. For every variable $v_i \in A_T$, if the clause satisfied by setting $v_i = ture$ has not appeared within Z so far, it would be added from X_i and Y_i to a sequence Z. Likewise, for every variable $v_i \in A_F$, the clause satisfied by setting $v_i = false$ is added from X_i and Y_i to Z if it has not been included by Z until now. In other words, any clause would be added into Z once. Besides, the separators s_i^w, $1 \in \{1, 2, \ldots b-1\}$, are added into appropriate positions in Z.

Because any variable v_i is assigned to either $ture$ or $false$, no internal crossings exists within X_i and Y_i. In addition, no crossing exists over the separators because the number of clauses within X_i and Y_i are not more than w. Each clause is added exactly once into Z and all separators are added to Z; consequently, Z, a longest sequence that is a CS between X and Y with length wb, excludes every constrained pattern as a subsequence.

($\Leftarrow$) Suppose that there exists a longest sequence Z that is a CS between X and Y with length $2w$ and excludes every constrained pattern as a subsequence.

Because there are not more than w elements separately within X_i and Y_i, every s_i^w, $1 \in \{1, 2, \ldots b-1\}$, are added into Z and every clause can not appear more than once within Z, sequence Z must include all clauses exactly once. Moreover, any clause appearing in X_i must be within Y_i, and vice versa. If some clause appears in X_i, there exists an assignment of v_i such that it satisfies the clause. Therefore, there exists an assignment of all variables satisfying F. $\square$

Example 3.3. *Suppose that* $F = (v_1 \vee v_2 \vee \bar{v_4}) \wedge (\bar{v_2} \vee \bar{v_3} \vee v_5) \wedge (\bar{v_1} \vee \bar{v_3} \vee v_4) \wedge (v_1 \vee \bar{v_2} \vee \bar{v_5})$. *The construction of Theorem 3.7 yields an instance of the SEQ-EC-LCS problem as follows.*

$X = C_1C_4C_3s_1^4C_1C_2C_4s_2^4C_2C_3s_3^4C_3C_1s_4^4C_2C_4,$

$Y = C_3C_1C_4s_1^4C_2C_4C_1s_2^4C_2C_3s_3^4C_1C_3s_4^4C_4C_2,$

$P_1 = C_1^2,$

$P_2 = C_2^2,$

$P_3 = C_3^2,$ *and*

$P_4 = C_4^2.$

Consider the assignment with $\{v_1 = ture, v_2 = ture, v_3 = false, v_4 = true, v_5 = flase\}$. *There exists a CS* $C_1C_4s_1^4s_2^4C_2C_3s_3^4s_4^4$ *between* X *and* Y *and excludes* $P_1, P_2, \ldots, P_4$ *as subsequences. On the other hand, consider the solution of the SEQ-EC-LCS problem with* $C_3s_1^4C_2C_4s_2^4s_3^4C_1s_4^4$. *By setting* $v_1 = flase$, $v_2 = false$, *and* $v_4 = false$, F *is satisfied. The two variables* v_3 *and* v_5 *may be set either to true or* $false$.

Because the decision version of the SEQ-EC-LCS problem is in NP, and the 3-SAT problem can be reduced to it, we conclude the following theorem.

Theorem 3.7. *The SEQ-EC-LCS problem in case of an arbitrary number of constrained patterns is NP-complete.*

3.5.2 Exact Algorithms

The SEQ-IC-LCS, SEQ-EC-LCS, and STR-EC-LCS problems with an arbitrary number of constrained patterns can be solved by the approaches similar to the problems with a single constrained pattern. Thus, we can immediately give an algorithm for each problem with time and space requirements being $O(mn \times \prod_{k=1}^{w} d_k)$.

It is impossible, unfortunately, to directly adopt the idea for solving the STR-IC-LCS problem with an arbitrary number of constrained patterns. Because the problem is complex, here we only investigate the STR-IC-LCS problem with two constrained patterns. Property 3.2 gives the characterization of the structure of a solution for the STR-IC-LCS problem with two constrained patterns.

Property 3.2. *If $Z[1..l]$ is a longest sequence which is a CS of $X[1..m]$ and $Y[1..n]$ and includes P_1 and P_2 as substrings, and assume that P_2 is the latter substring $Z[l'-d_2+1..l']$ (the case of P_1 being the latter substring is similar) for some $l' \in \{d_2, d_2+1, \ldots, l\}$, then $Z[1..l]$ is a concatenation of the following two substrings, for some $i \in \{1, 2, \ldots, m\}$ and $j \in \{1, 2, \ldots, n\}$:*

1. *The prefix $Z[1..l']$: $Z[1..l']$ is a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ and includes P_1 as a substring with P_2 as the suffix $Z[l'-d_2+1..l']$, and*

2. The suffix $Z[l'+1..l]$: $Z[l'+1..l]$ is an LCS of $X[i+1..m]$ and $Y[j+1..n]$.

Based on Property 3.2, we first solving the two subproblems separately of computing a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ such that P_1 is a substring and P_2 is a suffix, and of calculating an LCS of $X[i..m]$ and $Y[j..n]$. The solutions to the two subproblems are then merged to determine a longest concatenation.

The latter subproblem can be computed in quadratic time by employing the algorithm shown in Section 2.1. For solving the former subproblem, we need to consider the following two cases:

(a) P_2 overlaps P_1: We merge P_1 and P_2 into a new pattern (there are $\min\{d_1, d_2\}$ concatenations of length at most $d_1 + d_2 - 1$), and then compute $Z[1..l']$ by performing the algorithm for the STR-IC-LCS problem shown in Section 3.2. This case takes $\Sigma_{k=0}^{\min\{d_1,d_2\}-1} O(mn \times (\max\{d_1, d_2\} + k))$ $(= O(mnd_1d_2))$ time and $O(mn \times (d_1 + d_2 - 1))$ $(= O(mn \times \max\{d_1, d_2\}))$ space.

(b) P_2 does not overlap P_1: Because $Z[1..l']$ includes P_1 as a substring and P_2 as a suffix, we can decompose $Z[1..l']$ into two substrings, where one is a longest sequence which is a CS of $X[1..i']$ and $Y[1..j']$ such that P_1 is a substring, and the other is a longest sequence which is a CS of $X[i'+1..i]$ and $Y[j'+1..j]$ such that P_2 is a suffix for some $i' \in \{1, 2, \dots, i-1\}$ and $j' \in \{1, 2, \dots, j-1\}$.

An LCS of $X[1..i']$ and $Y[1..j']$ where P_1 is a substring, for all $i' \in \{1, 2, \dots, i-1\}$ and $j' \in \{1, 2, \dots, j-1\}$, can be computed by performing the algorithm for the STR-IC-LCS problem shown in Section 3.2. For all $i \in \{0, 1, \dots, m\}$ and

$j \in \{0, 1, \ldots, n\}$, the pair of the largest indices (i', j'), where P_2 is exactly an LCS of $X[i'+1..i]$ and $Y[j'+1..j]$, can be determined in $O(mnd_2)$ time. $Z[1..l']$ is the longest concatenation of the above two results, which can be obtained in $O(mn \times \max\{d_1, d_2\}))$ time and space.

The solution of the STR-IC-LCS problem is the longest concatenation of (1) a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ and includes P_1 as a substring with P_2 as a suffix (or P_2 is a substring and P_1 is a substring) and (2) an LCS of $X[i..m]$ and $Y[j..n]$. Computing the lengths of all concatenations takes $O(mnd_1d_2)$ time and $O(mn \times \max\{d_1, d_2\})$ space. Recovering an LCS by backtracking and concatenating takes $O(m+n+d_1+d_2)$ steps. Summarizing all stages, the STR-IC-LCS problem with two constrained patterns can be solved in $O(mnd_1d_2)$ time and $O(mn \times \max\{d_1, d_2\})$ space.

3.6 Discussion

In this chapter, we consider four variants of the LCS problem. We first introduce the previous results for the SEQ-IC-LCS problem, and then present three $O(mnd)$-time and $O(mnd)$-space algorithms for solving the STR-IC-LCS, SEQ-EC-LCS, and STR-EC-LCS problems, where m, n, and d are the lengths of two sequences and a constrained pattern, respectively. In fact, the space requirement can be further reduced to $O(d \times (m+n))$ by applying the space-saving strategy introduced in Section 2.2. We also consider the four problems with two sequences and an arbitrary number of constrained patterns, which are proved to be NP-complete except for the STR-EC-LCS problem. On the other hand, Gotthilf *et al.* [35] demonstrated that the SEC-IC-LCS problem with an arbitrary number of sequence and a single constrained pattern is NP-complete and can be solved by a linear-time approximation algorithm with the approximation factor $O(\frac{1}{m_{min}|\Sigma|})$, where m_{min} denotes the length of the shortest sequences and $|\Sigma|$ denotes the size of the alphabet.

Chapter 4

Hybrid Constrained LCSs

Due to applications and theoretical interests in molecular biology and sequence comparison, we have studied four variants for the LCS problem in Chapter 3. In this chapter, we merge the constraints on the SEQ-IC-LCS and SEQ-EC-LCS problems for making the similarity measurement more flexible. The new problem, named the hybrid constrained longest common subsequence (abbreviated HC-LCS) problem, are formally defined as follows.

Problem 4.1. ***(HC-LCS)*** *Given two sequences X, Y and two constrained patterns P, Q of lengths m, n, d, and e, respectively, the HC-LCS problem is to find a longest sequence that is a CS of X and Y and not merely includes P as a subsequence but excludes Q as a subsequence.*

Consider sequences $X = \texttt{BADBABD}$ and $Y = \texttt{ABCBDDA}$ as an example. "ABBD" is an LCS of X and Y. If $P = \texttt{DA}$ and $Q = \texttt{BA}$, "ADA" is a solution of the HC-LCS problem.

Obviously, the SEQ-IC-LCS and SEQ-EC-LCS problems are two simple forms of the HC-LCS problem.

Throughout this chapter, we define the formats of the sequences and constrained patterns as $X = x_1x_2 \dots x_m$, $Y = y_1y_2 \dots y_n$, $P = p_1p_2 \dots p_d$, and $Q = q_1q_2 \dots q_e$. In the following, we propose a DP algorithm for the HC-LCS problem. Later, we introduce a data structure named bounded heap, and show how to speed up the computation by adapting Hunt-Szymanski strategy and employing the data structure.

4.1 Dynamic Programming

Let $Z_{i,j,k,h}$ denote the set of all longest sequences which are CSs of $X[1..i]$ and $Y[1..j]$ and both include $P[1..k]$ as a subsequence and exclude $Q[1..h]$ as a subsequence. Lemmas 4.1 to 4.5 decompose the structure of an optimal solution based on the solutions to its smaller subproblems.

Lemma 4.1. *If $Z = z_1z_2 \dots z_l \in Z_{i,j,k,h}$ and $x_i = y_j = p_k = q_h$, then the following properties hold:*

(1) *$h = 1$ implies $Z[1..l]$ is an empty sequence.*

(2) *$h \geq 2$ and $z_l \neq x_i$ imply $Z[1..l] \in Z_{i-1,j-1,k,h}$.*

(3) *$h \geq 2$ and $z_l = x_i$ imply $Z[1..l-1] \in Z_{i-1,j-1,k-1,h-1}$.*

Proof. We prove this lemma case by case. (1) Since $p_k = q_h$ and $h = 1$, q_1 is a subsequence of $P[1..k]$. If $Z[1..l]$ is a nonempty sequence, then $Z[1..l]$ including

$P[1..k]$ as a subsequence implies that $Z[1..l]$ also includes q_1 as a subsequence. This is a contradiction.

(2) Since $z_l \neq x_i$, $Z[1..l]$ is a CS of $X[1..i-1]$ and $Y[1..j-1]$ such that $Z[1..l]$ includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. Assume by contradiction that there exists a CS $Z'[1..l+1]$ of $X[1..i-1]$ and $Y[1..j-1]$ where $Z'[1..l+1]$ includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. It follows that $Z'[1..l+1]$ is also a CS of $X[1..i]$ and $Y[1..j]$ such that $Z'[1..l+1]$ includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence, which is a contradiction.

(3) Since $z_l = x_i = y_j = p_k = q_h$, $Z[1..l-1]$ is a CS of $X[1..i-1]$ and $Y[1..j-1]$ such that $Z[1..l-1]$ both includes $P[1..k-1]$ as a subsequence and excludes $Q[1..h-1]$ as a subsequence. Assume by contradiction that there exists a CS $Z'[1..l]$ of $X[1..i-1]$ and $Y[1..j-1]$ such that $Z'[1..l]$ includes $P[1..k-1]$ as a subsequence and excludes $Q[1..h-1]$ as a subsequence. If we append $x_i = y_j = p_k = q_h$ to $Z'[1..l]$, then we obtain a CS of $X[1..i]$ and $Y[1..j]$ of length greater than l, which satisfies the constraints on $P[1..k]$ and $Q[1..h]$. This contradicts the hypothesis. □

Let $\mathcal{L}[i, j, k, h]$ denote the length of a sequence that belongs to $Z_{i,j,k,h}$. When $x_i = y_j = p_k = q_h$, we can derive that the following recurrence relation from Lemma 4.1.

$$\mathcal{L}[i,j,k,h] = \begin{cases} -\infty & \text{if } x_i = y_j = p_k = q_h \text{ and } h = 1, \\ \max\{\mathcal{L}[i-1,j-1,k,h], 1 + \mathcal{L}[i-1,j-1,k-1,h-1]\} & \\ & \text{if } x_i = y_j = p_k = q_h \text{ and } h \geq 2. \end{cases} \quad (4.1)$$

Lemma 4.2. *If $Z = z_1 z_2 \ldots z_l \in Z_{i,j,k,h}$, $x_i = y_j = p_k$, and ($h = 0$ or $x_i \neq q_h$), then $z_l = x_i$ and $Z[1..l-1] \in Z_{i-1,j-1,k-1,h}$.*

Proof. Since $x_i = y_j = p_k$ and $x_i \neq q_h$, we have $z_l = x_i = y_j = p_k$. Otherwise, we can obtain a CS of $X[1..i]$ and $Y[1..j]$ of length l by appending $x_i = y_j = p_k$ to $Z[1..l-1]$ such that it also satisfies the constraints for $P[1..k]$ and $Q[1..h]$. Therefore, $Z[1..l-1]$ is a CS of $X[1..i-1]$ and $Y[1..j-1]$ such that it includes $P[1..k-1]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. Similar to the proof in Lemma 4.1, we can show that $Z[1..l-1]$ belongs to $Z_{i-1,j-1,k-1,h}$. □

The following recurrence relation can be derived from Lemma 4.2 if $x_i = y_j = p_k$ and $x_i \neq q_h$.

$$\mathcal{L}[i,j,k,h] = 1 + \mathcal{L}[i-1,j-1,k-1,h] \quad \text{if } x_i = y_j = p_k \text{ and } (h = 0 \text{ or } x_i \neq q_h). \quad (4.2)$$

Lemma 4.3. *If $Z = z_1 z_2 \ldots z_l \in Z_{i,j,k,h}$, $x_i = y_j = q_h$, and ($k = 0$ or $x_i \neq p_k$), then the following properties hold:*

(1) *$h = 1$ implies $z_l \neq x_i$ and $Z[1..l] \in Z_{i-1,j-1,k,h}$.*

(2) *$h \geq 2$ and $z_l \neq x_i$ imply $Z[1..l] \in Z_{i-1,j-1,k,h}$.*

(3) *$h \geq 2$ and $z_l = x_i$ imply $Z[1..l-1] \in Z_{i-1,j-1,k,h-1}$.*

Proof. Because the proofs of Cases (2) and (3) are similar to Lemma 4.1, here we only prove Case (1) as follow. Since $x_i = y_j = q_h$, $x_i \neq p_k$, and $h = 1$, we have $z_l \neq x_i$. Otherwise, $Z[1..l]$ must contain $Q[1]$ as a subsequence. Thus, $Z[1..l]$ is a CS of $X[1..i-1]$ and $Y[1..j-1]$ such that it includes $P[1..k]$ as a subsequence and excludes $Q[1]$ as a subsequence. Similar to the proof in Lemma 4.1, we can show that $Z[1..l]$ belongs to $Z_{i-1,j-1,k,h}$. □

When $x_i = y_j = q_h$ and $x_i \neq p_k$, we can derive that the following recurrence relation from Lemma 4.3.

$$\mathcal{L}[i,j,k,h] = \begin{cases} \mathcal{L}[i-1,j-1,k,h] \\ \qquad\qquad \text{if } x_i = y_j = q_h,\ h = 1, \text{ and } (k = 0 \text{ or } x_i \neq p_k), \\ \max\{\mathcal{L}[i-1,j-1,k,h], 1 + \mathcal{L}[i-1,j-1,k,h-1]\} \\ \qquad\qquad \text{if } x_i = y_j = q_h,\ h \geq 2, \text{ and } (k = 0 \text{ or } x_i \neq p_k). \end{cases} \tag{4.3}$$

Lemma 4.4. *If $Z = z_1 z_2 \ldots z_l \in Z_{i,j,k,h}$, $x_i = y_j$, ($k = 0$ or $x_i \neq p_k$), and ($h = 0$ or $x_i \neq q_h$), then $z_l = x_i$ and $Z[1..l-1] \in Z_{i-1,j-1,k,h}$.*

Proof. Since $x_i = y_j$, $x_i \neq p_h$, and $x_i \neq q_h$, we have $z_l = x_i = y_j$. Otherwise, we can obtain a CS of $X[1..i]$ and $Y[1..j]$ of length l by appending $x_i = y_j$ to $Z[1..l-1]$ such that it also satisfies the constraints for $P[1..k]$ and $Q[1..h]$. Therefore, $Z[1..l-1]$ is a

CS of $X[1..i-1]$ and $Y[1..j-1]$ such that it includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. Similar to the proof in Lemma 4.1, we can show that $Z[1..l-1]$ belongs to $Z_{i-1,j-1,k,h}$. □

The following recurrence relation can be derived from Lemma 4.4 if $x_i = y_j$, $x_i \neq p_k$, and $x_i \neq q_h$.

$$\mathcal{L}[i,j,k,h] = 1 + \mathcal{L}[i-1,j-1,k,h] \quad \text{if } x_i = y_j,\ (k=0 \text{ or } x_i \neq p_k),\ \text{and } (h=0 \text{ or } x_i \neq q_h). \tag{4.4}$$

Lemma 4.5. *If $Z = z_1z_2\ldots z_l \in Z_{i,j,k,h}$ and $x_i \neq y_j$, then the following properties hold:*

(1) *$z_l \neq x_i$ implies that $Z[1..l] \in Z_{i-1,j,k,h}$.*

(2) *$z_l \neq y_j$ implies that $Z[1..l] \in Z_{i,j-1,k,h}$.*

Proof. Because the proofs of Case (2) is similar to Case (1), here we only prove Case (1) as follow. Since $z_l \neq x_i$, $Z[1..l]$ is a CS of $X[1..i-1]$ and $Y[1..j]$ such that it includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. Similar to the proof in Lemma 4.1, we can show that $Z[1..l-1]$ belongs to $Z_{i-1,j,k,h}$. □

The following recurrence relation can be derived from Lemma 4.5 if $x_i \neq y_j$.

$$\mathcal{L}[i,j,k,h] = \max\{\mathcal{L}[i-1,j,k,h], \mathcal{L}[i,j-1,k,h]\} \qquad \text{if } x_i \neq y_j. \tag{4.5}$$

By the optimality principles of the HC-LCS problem shown in Lemmas 4.1 to 4.5, each entry in matrix $\mathcal{L}$, for $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, n\}$, $k \in \{0, 1, \ldots, d\}$, and $h \in \{0, 1, \ldots, e\}$, can be computed by Equations (4.1) to (4.5). whose boundary conditions are $\mathcal{L}[i, 0, 0, h] = \mathcal{L}[0, j, 0, h] = 0$ and $\mathcal{L}[i, 0, k, h] = \mathcal{L}[0, j, k, h] = -\infty$ for $i \in \{0, 1, \ldots, m\}$, $j \in \{0, 1, \ldots, n\}$, $k \in \{1, 2, \ldots, d\}$, and $h \in \{0, 1, \ldots, e\}$.

Suppose that Z is a sequence that belongs to $\mathcal{Z}_{m,n,d,e}$, and is initially an empty sequence. The length of Z is given by $\mathcal{L}[m, n, d, e]$, which requires $O(mnde)$ computation time. Furthermore, sequence Z can be constructed by backtracking through a path from $\mathcal{L}[m, n, d, e]$ to $\mathcal{L}[0, 0, 0, 0]$. If $x_i = y_j$ and $\mathcal{L}[i, j, k, h]$ equals to $1 + \mathcal{L}(i - 1, j - 1, k - 1, h - 1)$, $1 + \mathcal{L}[i - 1, j - 1, k - 1, h]$, $1 + \mathcal{L}[i - 1, j - 1, k, h - 1]$, or $1 + \mathcal{L}[i - 1, j - 1, k, h]$, we append the character x_i to Z. Therefore, recovering an HC-LCS takes $O(m + n + d + e)$ steps, and the following theorem is stated.

Theorem 4.1. *The HC-LCS problem can be solved in $O(mnde)$ time and space.*

4.2 Bounded Heaps

Consider now the problem of querying an entry with the highest value (i.e., priority) among all those whose j-coordinates (i.e., key) on the DP matrix are below a given threshold. For this task, we employ a data structure *bounded heap* $\mathcal{H}$ that supports the following operations [23]:

- *Heap.Insert*($\mathcal{H}$; j, ℓ, *pos*): Insert the key j with priority ℓ and associated position *pos* into the bounded heap $\mathcal{H}$.

- *Heap.IncreaseLength*$(\mathcal{H}; j, \ell, pos)$: If the bounded heap $\mathcal{H}$ does not contain the key j, perform *Heap.Insert*$(\mathcal{H}; j, \ell, pos)$. Otherwise, set the priority of this key to $\max\{\ell, \ell'\}$, where ℓ' is its original priority.
- *Heap.Max*$(\mathcal{H}; j)$: Return the maximum priority of all items in $\mathcal{H}$ with key smaller than j and its associated position. If $\mathcal{H}$ does not contain any items with keys smaller than j, return null.

The bounded heap uses the van Emde Boas tree [75, 76] as a kernel structure. It follows that the initialization of the bounded heap takes $O(n \log\log n)$ time, and each of the above operations requires $O(\log\log n)$ time, where keys are drawn from the set $\{1, \ldots, n\}$ [23].

4.3 Speeding up the Computation

During the computation of $\mathcal{L}$ via Equations (4.1) to (4.5), the values in matrix $\mathcal{L}$ might increase only when a match between X and Y is encountered. To make the computation more efficiently, in this section we adapt Hunt-Szymanski strategy to restrict it on the positions of matches between X and Y.

For each position i of X, we use a linked list $\mathcal{M}[i]$ to record all corresponding positions j such that $x_i = y_j$, and keep this list in increasing order. For example, let $X =$ `BADBABD` and $Y =$ `ABCBDDA`, the desired lists are

$\mathcal{M}[1] = \langle 2, 4\rangle$,

$\mathcal{M}[2] = \langle 1, 7\rangle$,

$\mathcal{M}[3] = \langle 5, 6\rangle$,

$\mathcal{M}[4] = \mathcal{M}[1]$,

$\mathcal{M}[5] = \mathcal{M}[2]$,

$\mathcal{M}[6] = \mathcal{M}[1]$, and

$\mathcal{M}[7] = \mathcal{M}[3]$.

If visualizing matrix $\mathcal{L}$ as $(d + 1) \times (e + 1)$ two-dimensional submatrices rather than a four-dimensional matrix, we can convert the notation $\mathcal{L}[i, j, k, h]$ into $\mathcal{L}_{k,h}[i, j]$. Specifically, we exploit the monotonicity of every two-dimensional submatrix $\mathcal{L}_{k,h}$, which is stated in the following lemma.

Lemma 4.6. *The value of any position (i, j, k, h) in matrix $\mathcal{L}$ satisfies the property: $\mathcal{L}[i, j, k, h] \geq \mathcal{L}[i', j', k, h]$ for any $i' \leq i$ and $j' \leq j$.*

Proof. $\mathcal{L}[i, j, k, h]$ denotes the length of a longest sequence which is a CS of $X[1..i]$ and $Y[1..j]$ and not merely includes $P[1..k]$ as a subsequence but excludes $Q[1..h]$ as a subsequence. Similarly, $\mathcal{L}[i', j', k, h]$ denotes the length of a longest sequence that is a CS of $X[1..i']$ and $Y[1..j']$ and includes $P[1..k]$ as a subsequence and excludes $Q[1..h]$ as a subsequence. If $i' \leq i$ and $j' \leq j$, $X[1..i']$ and $Y[1..j']$ are substrings of $X[1..i]$ and $Y[1..j]$, respectively. Consequently, a longest sequence that is a CS of $X[1..i']$ and $Y[1..j']$ and satisfies the constraints for $P[1..k]$ and $Q[1..h]$ must be a CS of $X[1..i]$ and $Y[1..j]$ with the same constraints for $P[1..k]$ and $Q[1..h]$. In other

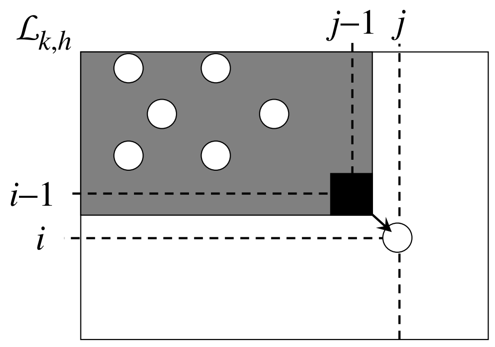

Figure 4.1: Obtaining $\mathcal{L}_{k,h}[i-1, j-1]$.

words, $\mathcal{L}[i, j, k, h] \geq \mathcal{L}[i', j', k, h]$. □

For the position (i, j) where a match occurs in a two-dimensional submatrix $\mathcal{L}_{k,h}$, the value of position $(i-1, j-1)$ would not be prepared for the entry $\mathcal{L}_{k,h}[i, j]$ if $x_{i-1} \neq y_{j-1}$. According to the monotonicity of each submatrix $\mathcal{L}_{k,h}$, $\mathcal{L}_{k,h}[i-1, j-1]$ can be derived from the maximum value within the submatrix in $\mathcal{L}_{k,h}$ composed by $X[1..i-1]$ and $Y[1..j-1]$ (the gray region in Figure 4.1) when x_{i-1} mismatches y_{j-1}. Without the computation of positions at which X mismatches Y, Equations (4.1) to (4.5) can be reformulated as follows, where $i \in \{1, 2, \ldots, m\}$, $j \in \mathcal{M}[i]$, $k \in \{0, 1, \ldots, d\}$, and $h \in \{0, 1, \ldots, e\}$.

$$\mathcal{L}_{k,h}[i,j] = \begin{cases} -\infty & \text{if } x_i = p_k = q_h \text{ and } h = 1, \\ \max\{\tau_1, 1+\tau_2\} & \text{if } x_i = p_k = q_h \text{ and } h \geq 2, \\ 1+\tau_3 & \text{if } x_i = p_k \text{ and } (h = 0 \text{ or } x_i \neq q_h), \\ \tau_1 & \text{if } x_i = q_h,\ h = 1, \text{ and } (k = 0 \text{ or } x_i \neq p_k), \\ \max\{\tau_1, 1+\tau_4\} & \text{if } x_i = q_h,\ h \geq 2, \text{ and } (k = 0 \text{ or } x_i \neq p_k), \\ 1+\tau_1 & \text{if } (k = 0 \text{ or } x_i \neq p_k) \text{ and } (h = 0 \text{ or } x_i \neq q_h), \end{cases} \tag{4.6}$$

where

$$\tau_1 = \max_{1 \leq i' < i, 1 \leq j' < j, j' \in \mathcal{M}[i']} \{\mathcal{L}_{k,h}[i', j']\}, \tag{4.7}$$

$$\tau_2 = \max_{1 \leq i' < i, 1 \leq j' < j, j' \in \mathcal{M}[i']} \{\mathcal{L}_{k-1,h-1}[i', j']\}, \tag{4.8}$$

$$\tau_3 = \max_{1 \leq i' < i, 1 \leq j' < j, j' \in \mathcal{M}[i']} \{\mathcal{L}_{k-1,h}[i', j']\}, \tag{4.9}$$

and

$$\tau_4 = \max_{1 \leq i' < i, 1 \leq j' < j, j' \in \mathcal{M}[i']} \{\mathcal{L}_{k,h-1}[i', j']\}. \tag{4.10}$$

The boundary conditions of this recursive formula are $\mathcal{L}_{0,h}[i,0] = \mathcal{L}_{0,h}[0,j] = 0$ and $\mathcal{L}_{k,h}[i,0] = \mathcal{L}_{k,h}[0,j] = -\infty$ for any $i \in \{0,1,\ldots,m\}$, $j \in \{0,1,\ldots,n\}$, $k \in$

```
**Algorithm** HC-LCS($X, Y, P, Q$)
 1: Build a linked list $M[i]$ for all $1 \le i \le m$;
 2: Initialize the bounded heap $\mathcal{H}_{k,h}$ for all $1 \le k \le p$ and $1 \le h \le q$;
 3: Perform $Insert(\mathcal{H}_{k,h}; 0, 0, (0, 0, k, h))$ if $k = 0$ and $Insert(\mathcal{H}_{k,h}; 0, -\infty, (0, 0, k, h))$ otherwise;
 4: for $k \leftarrow 0$ to $d$ do
 5:     for $h \leftarrow 0$ to $e$ do // compute each $\mathcal{L}_{k,h}$
 6:         for $i \leftarrow 1$ to $m$ do
 7:             Initialize the insert list $\mathcal{I}$ as an empty list;
 8:             for each $j$ in $M[i]$ do
 9:                 $\mathcal{T}_{k,h}(i,j) \leftarrow Heap.Max(\mathcal{H}_{k,h}; j)$; // the 2nd, 4th, and 5th cases of Equation (4.6)
10:                 $\mathcal{L}_{k,h}(i,j) \leftarrow \mathcal{T}_{k,h}(i,j)$;
11:                 if $x_i = p_k$ and $x_i = q_h$ and $h = 1$ then
12:                     $\mathcal{L}_{k,h}(i,j) \leftarrow -\infty$; // the 1st case of Equation (4.6)
13:                 else if $x_i = p_k$ and $x_i = q_h$ and $h \ge 2$ and $\mathcal{T}_{k,h}(i,j) \le 1 + \mathcal{T}_{k-1,h-1}(i,j)$ then
14:                     $\mathcal{L}_{k,h}(i,j) \leftarrow 1 + \mathcal{T}_{k-1,h-1}(i,j)$; // the 2nd case of Equation (4.6)
15:                 else if $x_i = p_k$ and ($x_i \ne q_h$ or $h = 0$) then
16:                     $\mathcal{L}_{k,h}(i,j) \leftarrow 1 + \mathcal{T}_{k-1,h}(i,j)$; // the 3rd case of Equation (4.6)
17:                 else if $x_i = q_h$ and $h \ge 2$ and ($x_i \ne p_k$ or $k = 0$) and $\mathcal{T}_{k,h}(i,j) \le 1 + \mathcal{T}_{k,h-1}(i,j)$ then
18:                     $\mathcal{L}_{k,h}(i,j) \leftarrow 1 + \mathcal{T}_{k,h-1}(i,j)$; // the 5th case of Equation (4.6)
19:                 else if ($x_i \ne p_k$ or $k = 0$) and ($x_i \ne q_h$ or $h = 0$) then
20:                     $\mathcal{L}_{k,h}(i,j) \leftarrow 1 + \mathcal{T}_{k,h}(i,j)$; // the 6th case of Equation (4.6)
21:                 $\mathcal{I} \leftarrow \mathcal{I} \cup (\mathcal{L}_{k,h}(i,j), (i,j,k,h))$;
22:             for each $(\mathcal{L}_{k,h}(i,j), (i,j,k,h))$ in $\mathcal{I}$ do
23:                 $IncreaseLength(\mathcal{H}_{k,h}; j, \mathcal{L}_{k,h}(i,j), (i,j,k,h))$;
24: **Output** $Heap.Max(\mathcal{H}_{p,q}; n+1)$;
```

$\{1, 2, \ldots, d\}$, and $h \in \{0, 1, \ldots, e\}$. Specifically, we employ the bounded heap to compute Equations (4.7) to (4.10) efficiently.

Algorithm HC-LCS formally describes the our algorithm adapting Hunt-Szymanski strategy and employing a bounded heap. We first sort X and Y, and then scan the two sorted sequences to build a linked list $\mathcal{M}[i]$, for each position i of X, as an increasing list of all corresponding positions j such that $x_i = y_j$. Note that the original four-dimensional matrix $\mathcal{L}$ is visualized as $(a+1)\times(b+1)$ two-dimensional matrices, i.e., $\mathcal{L}_{k,h}$ where $k \in \{0,1,\ldots,d\}$ and $h \in \{0,1,\ldots,e\}$. We maintain a bounded heap $\mathcal{H}_{k,h}$ for each $\mathcal{L}_{k,h}$ to implement Equations (4.7) to (4.10).

The matches between X and Y in each $\mathcal{L}_{k,h}$ is processed in row-major order. At iteration i (i.e., row i of $\mathcal{L}_{k,h}$ is processed), $\mathcal{H}_{k,h}$ has kept a position with its $\mathcal{L}_{k,h}$-value for each $j' \in \{1,2,\ldots,n\}$ such that the position provides the maximum value among all positions (i',j',k,h) where $l' \in \{1,2,\ldots,i-1\}$, and empty if no such position exists. For each j in $\mathcal{M}[i]$, we query the position that has the maximum value among all positions in $\mathcal{H}_{k,h}$ with key smaller than j, and then store the queried value in $\mathcal{T}_{k,h}[i,j]$. According to Equation (4.6), we compute $\mathcal{L}_{k,h}[i,j]$ by the relationships among x_i, p_k, and q_h, and then place the position (i,j,k,h) with $\mathcal{L}_{k,h}[i,j]$ into the insert list $\mathcal{I}$. After row i of $\mathcal{L}_{k,h}$ is processed, we update the data in $\mathcal{H}_{k,h}$ by performing *Heap.IncreaseLength* on each entry in $\mathcal{I}$. Finally, we query the position that has the maximum value among all positions in $\mathcal{H}_{d,e}$ with key smaller than $n+1$. The maximum value in $\mathcal{H}_{d,e}$ is the length of an HC-LCS, and we can obtain an HC-LCS by tracing back through the backtracking links from the position with the maximum length until $i=0$ or $j=0$.

Since sorting is invoked in building the m linked lists, building linked lists requires $O(n)$ time and $O(n)$ space over a finite alphabet [31]. The initialization of each $\mathcal{H}_{k,h}$ takes $O(n\log\log n)$ time. Let r denote the total number of matches between X and Y. We need to maintain an $\mathcal{H}_{k,h}$ and $\mathcal{I}$ for each $\mathcal{L}_{k,h}$. Because the operations *Heap.IncreaseLength* and *Heap.Max* are performed on $\mathcal{H}_{k,h}$ at most r times, computing a two-dimensional submatrix $\mathcal{L}_{k,h}$ takes $O(r \log\log n)$ time and $O(r+n)$ space. Consequently, the calculation of the $(d+1)\times(e+1)$ matrices requires in total $O(de \times r\log\log n)$ time and $O(de \times (r+n))$ space. Finally, delivering an HC-LCS takes $O(m+n+d+e)$ steps. In summary, we have the following corollary.

Theorem 4.2. *Given two sequences X, Y and two constrained patterns P, Q of lengths m, n, d, and e, respectively. Without loss of generality, assume that $m \leq n$. Let r denote the total number of matches between X and Y. The HC-LCS problem can be solved in $O((der+n)\times\log\log n))$ time and $O(de\times(r+n))$ space.*

4.4 Discussion

In this chapter, we studied the HC-LCS problem, which is a hybrid problem of the SEQ-IC-LCS and SEQ-EC-LCS problems. We presented a traditional DP algorithm for solving the HC-LCS problem in $O(mnde)$ time and space. Subsequently, Algorithm HC-LCS, an improved algorithm, was proposed for restricting the computation on the positions of matches between X and Y, which requires $O((der+n)\times\log\log n)$ time and $O(de\times(r+n))$ space over a finite alphabet. In the worse case, Algorithm HC-LCS requires $O(demn\log\log n)$ time. However, the latter outperforms the former algorithm if $r = o(n(\frac{m}{\log m}-1))$. If only the dominant matches between X and Y are considered during the computation, the latter algorithm may be improved by adapting the strategy proposed in [13].

Chapter 5

LCSs of Run-Length Encoded Sequences

The LCS problem has many applications in molecular biology, pattern comparison, and screen redisplay. Some sequences appearing in the real world may contain several segments of contiguous identical characters. For reducing the processing space and speeding up the processing time of similarity measurement, it is beneficial to use particular representations for such sequences, such as run-length encoding (abbreviated RLE) representation. RLE is a simple and widely used coding scheme, which compresses a sequence into several runs so that each run is a maximal-length substring of an identical character in the sequence. A sequence in RLE format is represented by an ordered sequence of the characters corresponding to the runs with their lengths. In this chapter, we consider the LCS problem of RLE sequences (abbreviated RLE-LCS), which is formally defined as follows.

Problem 5.1. ***(RLE-LCS)*** *Given two sequences X and Y of lengths m and n with M and N runs, respectively, the RLE-LCS problem is to find an LCS of X and Y.*

Throughout this chapter, the format of input sequences is defined as follows. Let $X = x_1x_2 \dots x_m$ and $Y = y_1y_2 \dots y_n$ denote two sequences over a finite alphabet Σ. Let $\widehat{X} = \alpha_1^{m_1}\alpha_2^{m_2} \dots \alpha_M^{m_M}$ denote the RLE representation of X, where α_i for $i \in \{1, 2, \dots, M\}$ is the unique character of the i-th run in X and m_i is the length of this run. We also let $\widehat{X}[i..j]$ denote the RLE substring of X from run i to run j if $1 \leq i \leq j \leq M$, and an empty string otherwise. Likewise, $\widehat{Y} = \beta_1^{n_1}\beta_2^{n_2} \dots \beta_N^{n_N}$ denotes the RLE representation of Y; $\widehat{Y}[i..j]$ denotes the RLE substring of Y from run i to run j if $1 \leq i \leq j \leq N$, and an empty string otherwise.

Section 5.1 introduces the previous results for the RLE-LCS problem. In Section 5.2, we devise a linear-time algorithm for computing the maxima within a sliding window upon a numerical sequence where the window size is dynamic. We further improve some previous results for the REL-LCS problem in Section 5.3, by adapting Hunt-Szymanski strategy and the approach to the sliding-window maxima problem with a dynamic window size.

5.1 Related Works

Some related results for the RLE-LCS problem are summarized in Table 5.1. Bunke and Csirik [24] and Arbell *et al.* [15] separately proposed an $O(mN + Mn)$-time algorithm, while Apostolico *et al.* [14] discovered an $O(MN \times \log(MN))$-time algorithm. Mitchell [58] reduced the RLE-LCS problem to the geometric shortest path problem, and gave an $O((M + N + R) \times \log(M + N + R))$-time algorithm solving the latter problem by using special convex distance functions, where R denotes the number of the pairs of runs at which the two sequences match each other. Recently, algorithms with time complexity of $O(Mn + mN - MN)$ [34], $O(\min\{Mn, mN\})$ [53], and $O(\min\{\rho_1, \rho_2\} + MN)$ [11] were presented, where ρ_1 and ρ_2 denote the numbers of entries on the bottom borders and right borders of the blocks whose corresponding runs match each other, respectively.

In the following we depict the properties of the RLE-LCS problem and introduce two approaches separately shown in [34] and [11]. Lemma 5.1 shows the characterization of an optimal solution to the RLE-LCS problem based on the solutions to its smaller subproblems [34].

Lemma 5.1. *Let $LCS(\widehat{X}, \widehat{Y})$ denote the length of an LCS of $\widehat{X}$ and $\widehat{Y}$. Given two distinct characters σ_1, σ_2 and two nonnegative integers s_1, s_2, the following properties hold:*

Table 5.1: Some results related to the RLE-LCS problem

Year	Author(s)	Time Complexity
1995	Bunke and Csirik [24]	$O(mN + Mn)$
1997	Mitchell [58]	$O((M + N + R) \times \log(M + N + R))$
1999	Apostolico *et al.* [14]	$O(MN \log(MN))$
2002	Arbell *et al.* [15]	$O(Mn + mN)$
2004	Freschi and Bogliolo [34]	$O(Mn + mN - MN)$
2008	Liu *et al.* [53]	$O(\min\{Mn, mN\})$
2008	Ann *et al.* [11]	$O(\min\{\rho_1, \rho_2\} + MN)$

(1) $LCS(\widehat{X}\sigma_1^{s_1}, \widehat{Y}\sigma_1^{s_2}) = LCS(\widehat{X}\sigma_1^{s_1-d}, \widehat{Y}\sigma_1^{s_2-d}) + d$, *where* $d = \min\{s_1, s_2\}$.

(2) $LCS(\widehat{X}\sigma_1^{s_1}, \widehat{Y}\sigma_2^{s_2}) = \max\{LCS(\widehat{X}\sigma_1^{s_1}, \widehat{Y}), LCS(\widehat{X}, \widehat{Y}\sigma_2^{s_2})\}$.

Bunke and Csirik [24] discovered the division of run-sized blocks in the DP matrix, which partitions the matrix into $M \times N$ *blocks* corresponding to pairs of runs in $\widehat{X}$ and $\widehat{Y}$. A block (i, j) is *match* if $\alpha_i = \beta_j$; otherwise, it is *mismatch*. We take Figure 5.1 as an example. The DP matrix over sequences "$\mathtt{a}^4\mathtt{b}^2\mathtt{c}^4\mathtt{a}^8$" and "$\mathtt{b}^2\mathtt{a}^6\mathtt{b}^3\mathtt{c}^4\mathtt{a}^3\mathtt{b}^6\mathtt{a}^4$" is split into 4×7 blocks, where match blocks are colored by gray and white blocks indicate mismatch blocks.

A four-dimensional matrix $\mathcal{B}$ is used for the bottom borders and right borders in blocks, where $\mathcal{B}_{i,j}[m_i, h]$ denotes the value of the h-th entry on the bottom border of

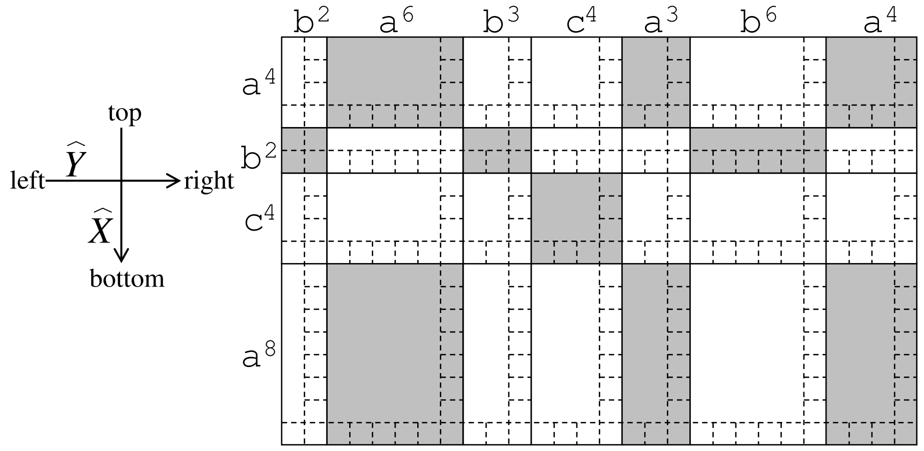

Figure 5.1: Match blocks (gray blocks) and mismatch blocks (white blocks).

the block (i, j), and $\mathcal{B}_{i,j}[v, n_j]$ denotes the value of the v-th entry on the right border of the block (i, j). Notice that $\mathcal{B}$ is physically mapped to the original two-dimensional DP matrix. According to Lemma 5.1, the following recurrence [34] is derived for computing each lattice $\mathcal{B}_{i,j}[v, h]$ for $v = m_i$ or $h = n_j$ with proper initializations.

$$\mathcal{B}_{i,j}[v,h] = \begin{cases} \mathcal{B}_{i-1,j}[m_{i-1}, h-v] + v & \text{if block } (i,j) \text{ is match and } v < h, \\ \mathcal{B}_{i,j-1}[v-h, n_{j-1}] + h & \text{if block } (i,j) \text{ is match and } v > h, \\ \mathcal{B}_{i-1,j-1}[m_{i-1}, n_{j-1}] + h & \text{if block } (i,j) \text{ is match and } v = h, \\ \max\{\mathcal{B}_{i-1,j}[m_{i-1}, h], & \\ \qquad \mathcal{B}_{i,j-1}[v, n_{j-1}]\} & \text{if block } (i,j) \text{ is mismatch.} \end{cases} \tag{5.1}$$

By Equation 5.1, the length of an LCS between $\widehat{X}$ and $\widehat{Y}$, given by $\mathcal{B}_{M,N}[m_M, n_N]$, can be calculated in $O(Mn + mN - MN)$ [34].

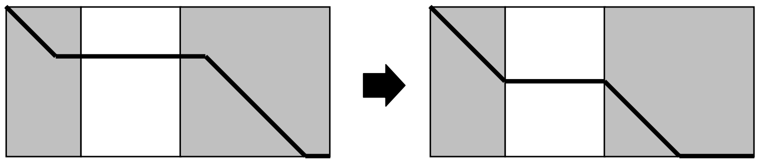
Figure 5.2: Converting an arbitrary subpath into a forced path.

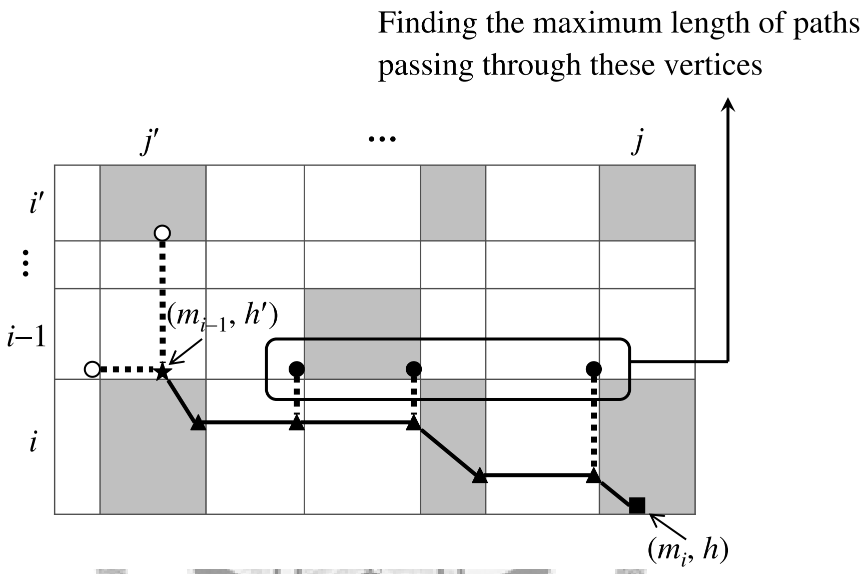
Finding the maximum length of paths passing through these vertices
j'
$\cdots$
j
i'
$\vdots$
$i-1$
i
(m_{i-1}, h')
(m_i, h)

Figure 5.3: Ann *et al.*'s approach [11].

In the original DP matrix, a monotonically nondecreasing path that begins at the top-left corner of a match block is defined as a *forced path* [14] if it traverses the match blocks by strictly diagonal moves and traverses the mismatch blocks by either strictly horizontal moves or strictly vertical moves. Figure 5.2 gives an example of forced paths, and illustrates that any subpath which begins at the top-left corner of a match block can be converted into a forced subpath [14].

By the property of path conversion, Ann *et al.* [11] presented an algorithm that reduces the computation on the bottom borders and right borders in all blocks to the bottom-right corners in all blocks and the bottom borders in the match blocks. Figure 5.3 illustrates the idea of Ann *et al.*'s approach. For any entry (m_i, h) of block (i, j), suppose that the forced path passing through it crosses row $i-1$ at entry (m_{i-1}, h') of block $(i-1, j')$. The value of entry (m_i, h) of block (i, j), denoted by $\mathcal{B}_{i,j}[m_i, h]$, is derived from the maximum length of two forced paths ending at the entry over the LCS matric function. One path is passing through entry (m_{i-1}, h') of block $(i-1, j')$, and the other is with the maximum length among such the forced paths passing through the corner of block $(i-1, j'')$ for all $j'' \in \{j'+1, j'+2, \ldots, j-1\}$. In addition, the value of entry (m_{i-1}, h') of block $(i-1, j')$ can be further derived from the maximum of the values given in the bottom-right corner of block $(i-1, j'-1)$ and entry $(m_{i'}, h')$ of block (i', j'), where i' denotes the index of the former run of $\widehat{X}[i]$ with the same character α_i in $\widehat{X}$. The length of an LCS between $\widehat{X}$ and $\widehat{Y}$ is given by $\mathcal{B}_{M,N}[m_M, n_N]$. Before the algorithm starts, the numbers of entries in the bottom borders and right borders of the match blocks can be separately calculated to determine which of the input sequences is considered as $\widehat{X}$. Because the computation on the bottom-right corners of all blocks and the bottom borders of the match blocks is required, the algorithm takes $O(MN + \min\{\rho_1, \rho_2\})$ time [11].

Notice that the increment of the values given in entries might occur when a pair of runs in $\widehat{X}$ and $\widehat{Y}$ match each other. With the monotonicity of the original DP matrix [45], therefore, in Section 5.3 we restrict the computation merely on the match blocks.

5.2 Sliding-Window Maxima with a Dynamic Window Size

Consider now the following problem. Given two numerical sequences $A = a_1a_2 \dots a_n$ and $S = s_1s_2 \dots s_n$ satisfying $1 \leq s_i \leq i$, $s_0 = 0$, and $s_{i-1} \leq s_i$ for all $i \in \{1, 2, \dots, n\}$, the sliding-window maxima problem with dynamic window sizes is to compute a numerical sequence $Z = z_1z_2 \dots z_n$ such that each z_i is the maximum number within a interval $[s_i, i]$, i.e., $z_i = \max\{x_j : s_i \leq j \leq i\}$. For this task, we employ a data structure *double-ended queue* (abbreviated deque) $\mathcal{Q}$ that supports the following operations [71]:

- *Queue.Insert*($\mathcal{Q}$; ℓ): Insert value ℓ at the rear of $\mathcal{Q}$.
- *Queue.DeleteFront*($\mathcal{Q}$): Delete the front element from $\mathcal{Q}$.
- *Queue.DeleteRear*($\mathcal{Q}$): Delete the rear element from $\mathcal{Q}$.
- *Queue.Max*($\mathcal{Q}$): Return the front element in $\mathcal{Q}$.

We devise an approach named Algorithm DSW-Max, which maintains $\mathcal{Q}$ as a decreasing list to solving the problem. Because the front element of $\mathcal{Q}$ is always the maximum value within the currently window-interval, the correctness of the algorithm can be easily demonstrated. The deque $\mathcal{Q}$ is accessed $O(n)$ times during the execution of the algorithm, and each operation requires $O(1)$ time in the worst case [71]. Consequently, the maximum value within the interval $[s_i, i]$ for all $i \in \{1, 2, \dots, n\}$ can be obtained in linear time.

```
**Algorithm** DSW-MAX$(A, S)$
1 **for** $i \leftarrow 1$ to $n$ **do**
2     **while** $\mathcal{Q}$ is nonempty **and** the front element of $\mathcal{Q}$ is out of $[s_i, i]$ **do**
3         $Queue.DeleteFront(\mathcal{Q})$;
4     **while** $\mathcal{Q}$ is nonempty **and** $x_i$ is larger than or equal to the rear element of $\mathcal{Q}$ **do**
5         $Queue.DeleteRear(\mathcal{Q})$;
6     $Queue.Insert(\mathcal{Q}; x_i)$;
7     $z_i = Queue.Max(\mathcal{Q})$;
```

Theorem 5.1. *The sliding-window maxima problem with dynamic window sizes can be solved in linear time.*

5.3 An Efficient Algorithm

Let $\widehat{\mathcal{L}}[i,j]$ denote the length of an LCS of $\widehat{X}[1..i]$ and $\widehat{Y}[1..j]$. In other words, $\widehat{\mathcal{L}}[i,j]$ records the value given by the bottom-right corner of the block (i,j). Because the $\widehat{\mathcal{L}}$-values might increase only when a match run between $\widehat{X}$ and $\widehat{Y}$ is encountered, in this section we modify Ann *et al.*'s approach [11] to restrict the computation on the bottom borders of the match blocks by adapting Hunt-Szymanski strategy. Algorithm RLE-LCS formally describes the modified approach.

First of all, a preprocessing for calculating all match runs between $\widehat{X}$ and $\widehat{Y}$ is performed as follows. For each run i, we use a linked list $\mathcal{M}[i]$ to record the positions of the runs j in $\widehat{Y}$ in increasing order such that $\alpha_i = \beta_j$. For example, let $\widehat{X} = \mathtt{a}^4\mathtt{b}^2\mathtt{c}^5\mathtt{a}^8$ and $\widehat{Y} = \mathtt{b}^2\mathtt{a}^6\mathtt{b}^5\mathtt{c}^6\mathtt{a}^3\mathtt{b}^6\mathtt{a}^5$, the desired lists are

$\mathcal{M}[1] = \langle 2, 5, 7 \rangle$,

$\mathcal{M}[2] = \langle 1, 3, 6 \rangle$,

$\mathcal{M}[3] = \langle 4 \rangle$, and

$\mathcal{M}[4] = \mathcal{M}[1]$.

For this task, we sort $\widehat{X}$ and $\widehat{Y}$, and then scan the two sorted sequences to build a linked list $\mathcal{M}[i]$, for each run i in $\widehat{X}$, as an increasing list of all corresponding runs j such that $\alpha_i = \beta_j$.

By adapting Hunt-Szymanski strategy, we employ a bounded heap $\mathcal{H}$ to make the computation efficiently, which is restricted on the bottom borders of the match blocks.

Algorithm RLE-LCS proceeds to the match runs between $\widehat{X}$ and $\widehat{Y}$ in row-major order. While row i of matrix $\widehat{\mathcal{L}}$ is processed, the bounded heap $\mathcal{H}$ has kept a position in $\widehat{\mathcal{L}}$ with its corresponding value for each key $j'' \in \{1, 2, \ldots, N\}$ (i.e., each column in $\widehat{\mathcal{L}}$) such that its $\widehat{\mathcal{L}}$-value is the maximum amongst the positions (i'', j'') for all $i'' \in \{1, 2, \ldots, i-1\}$, and empty if no such position exists. For each j in $\mathcal{M}[i]$, we perform $Heap.Max(\mathcal{H}; j)$ for obtaining the position (i^*, j^*) with the maximum value amongst all positions in $\mathcal{H}$ where $j^* < j$, and set $\mathcal{T}[i, j] = \widehat{\mathcal{L}}[i^*, j^*]$.

The maximum length of the paths passing through the top-left corners of the match block (i, j), denoted by $W_i[j]$, is the maximum value amongst all positions in the match blocks (i'', j''), where $i'' < i$ and $j'' < j$, plus the number of occurrences of α_i from the position with maximum value to the end of Y. For $i \in \{1, 2, \ldots, M\}$, $j \in$

```
Algorithm RLE-LCS($\widehat{X}, \widehat{Y}$)
 1 Do preprocessing on $F$ and $\mathcal{M}[i]$ for all $1 \le i \le M$;
 2 Initialize the bounded heap $\mathcal{H}$ and the matrices $\widehat{\mathcal{L}}$ and $\mathcal{B}$;
 3 Perform $Heap.Insert(\mathcal{H}; 0, 0, (0, 0))$;
 4 for $i \leftarrow 1$ to $M$ do
 5     Do preprocessings on $\omega_i$ and $\chi_i$;
 6     Initialize the insertion list $\mathcal{I}$ as an empty list;
 7     for each $j$ in $\mathcal{M}[i]$ do // process the match blocks
 8         $\mathcal{T}[i, j] \leftarrow Heap.Max(\mathcal{H}; j)$;
 9         Compute $W_i[j]$;
10         for $h \leftarrow 1$ to $n_j$ do
11             if the forced path does not cross row $i - 1$ then
12                 $\mathcal{B}_{i,j}[h] \leftarrow \max\{W_i[0], \ldots, W_i[j]\} - occ_i[j, h] + 1$;
13             else
14                 $i' \leftarrow F[i]$;
15                 $(j', h') \leftarrow \chi_i[j, h]$;
16                 $\tau_1 \leftarrow \max\{W_i[j' + 1], \ldots, W_i[j]\} - occ_i[j, h] + 1$; // implemented by Algorithm DSW-Max
17                 $\tau_2 \leftarrow \max\{\mathcal{T}[i - 1, j'], \mathcal{B}_{i',j'}[h']\} + m_i$;
18                 $\mathcal{B}_{i,j}[h] \leftarrow \max\{\tau_1, \tau_2\}$;
19         $\widehat{\mathcal{L}}[i, j] \leftarrow \mathcal{B}_{i,j}[n_j]$;
20         $\mathcal{I} \leftarrow I \cup (\widehat{\mathcal{L}}[i, j], (i, j))$;
21     for each $(\widehat{\mathcal{L}}[i, j], (i, j))$ in $\mathcal{I}$ do
22         Perform $Heap.IncreaseLength(\mathcal{H}; j, \widehat{\mathcal{L}}[i, j], (i, j))$;
23 Output $Heap.Max(\mathcal{H}; N + 1)$;
```

$\{1, 2, \ldots, N\}$, and $h \in \{1, 2, \ldots, n_j\}$, let $occ_i[j, h]$ denote the number of occurrences of α_i in the suffix $Y[\pi..m]$, where π is the position of the h-th entry of $\widehat{Y}[j]$ in Y. In other words, $W_i[j] = \mathcal{T}[i, j] + occ_{i^*}[j^*, n_{j^*}] - 1$.

If the forced path passing through entry (m_i, h) of block (i, j) does not cross row $i-1$, then $\mathcal{B}_{i,j}[h] = \max\{W_i[0], \ldots, W_i[j]\} - occ_i[j, h] + 1$, where $\max\{W_i[0], \ldots, W_i[j]\}$ for all j in $\mathcal{M}[i]$ can be obtained by scanning W_i once. On the other hand, for $i \in \{1, 2, \ldots, M\}$, $j \in \{1, 2, \ldots, N\}$, and $h \in \{1, 2, \ldots, n_j\}$, we let $\chi_i[j, h]$ denote the position of row $i-1$ in which the forced path passing through entry (m_i, h) of block (i, j) crosses, and suppose that the position locates at entry (m_{i-1}, h') of block $(i-1, j')$, i.e., $(j', h') = \chi_i[j, h]$. We define $\tau_1 = \max\{W_i[j'+1], \ldots, W_i[j]\} - occ_i[j, h]$, where $\max\{W_i[j'+1], \ldots, W_i[j]\}$ can be obtained by employing Algorithm DSW-Max, and $\tau_2 = \max\{\mathcal{T}[i-1, j'], \mathcal{B}_{i',j'}[h']\} + m_i$, where i' denotes the index of the former run of $\widehat{X}[i]$ with the same character α_i in $\widehat{X}$, which is computed and kept into $F[i]$ at the preprocessing stage. Finally, $\mathcal{B}_{i,j}[h]$ is determined by $\max\{\tau_1, \tau_2\}$.

When all elements on the bottom border of block (i, j) are computed, we set $\widehat{\mathcal{L}}[i, j] = \mathcal{B}_{i,j}[n_j]$ and place the item of the position (i, j) with its value $\widehat{\mathcal{L}}[i, j]$ into the insertion list $\mathcal{I}$. At the end of the iteration on row i, we perform the operation *Heap.IncreaseLength* for each entry in $\mathcal{I}$ on $\mathcal{H}$.

After all match blocks are processed, we query the position that has the maximum value amongst the positions in $\mathcal{H}$ with key smaller than $N+1$, whose value is the length of an LCS. We can obtain an LCS by tracing back via the backtracking links from the position with the maximum value until $i = 0$ or $j = 0$. Because Algorithm RLE-LCS is extended from the framework given by Ann *et al.* [11], its correctness follows.

Theorem 5.2. *Algorithm* RLE-LCS *computes the length of an LCS of* $\widehat{X}$ *and* $\widehat{Y}$ *in* $O(\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N)$ *time, using* $O(\min\{\rho_1, \rho_2\} + M + N)$ *space, where* R *denotes the number of the match blocks.*

Proof. Before the algorithm starts, the numbers of entries on the bottom borders and right borders of the match blocks can be calculated to determine which of the input sequences is considered as sequence X.

For efficiently implementing the algorithm in space, we use linked lists $\mathcal{M}[i]$ for all $i \in \{1, 2, \ldots, M\}$ to keep the positions of all match blocks. For each match block, we employ a one-dimensional array for its bottom border. In addition, the bounded heap $\mathcal{H}$ uses $O(N)$ space. Thus, Algorithm RLE-LCS uses $O(\min\{\rho_1, \rho_2\} + M + N)$ space.

Building the linked lists $\mathcal{M}$ in line 1 takes $O(M + N)$ time over a finite alphabet [31]. Also, line 1 for computing the array F for $\widehat{X}$ takes $O(M)$ time. The initializations on $\mathcal{H}$, $\widehat{\mathcal{L}}$, and $\mathcal{B}$ in line 2 take $O(N \log\log N)$, $O(R)$, and $O(\min\{\rho_1, \rho_2\})$ time, respectively. Computing the arrays *occ* and χ in line 5 totally takes $O(\min\{\rho_1, \rho_2\})$ time. The operations *Heap.Max* performed on $\mathcal{H}$ in line 8 and *Heap.IncreaseLength* performed on $\mathcal{H}$ in line 22 take $O(r \log\log N)$ time in total. With the values of $\mathcal{T}$ and *occ*, $W_i[j]$ in line 9 can be computed in $O(1)$ time. For each row i, the sliding-window maxima problem of a input sequence W_i with a given dynamic window size can be solved in time linear to the number of elements in W_i. Because $W_1, W_2, \ldots$, and W_M totally contain R elements, lines 12 and 16 for computing the maximal values in the required intervals takes $O(R)$ time. Each value in matrices $\mathcal{B}$ and $\widehat{\mathcal{L}}$ can be calculated in $O(1)$ time. Summarizing the time complexity of all stages, we obtain the desired time complexity. $\square$

5.4 Discussion

In this chapter we study the RLE-LCS problem for two sequences X and Y of lengths m and n with M and N runs, respectively. The proposed algorithm for the RLE-LCS problem takes $O(\min\{\rho_1, \rho_2\} + (R+N) \times \log\log N)$ time and $O(\min\{\rho_1, \rho_2\} + M + N)$ space. Comparing with Ann *et al.*'s algorithm [11] with $O(\min\{\rho_1, \rho_2\} + MN)$ time, our algorithm outperforms Ann *et al.*'s algorithm if $R = o(N(\frac{M}{\log\log N} - 1))$.

Chapter 6

Constrained LCSs of Run-Length Encoded Sequences

The SEQ-IC-LCS problem aries from the applications in molecular biology, and also has applications in pattern comparison. Some of such sequences appearing in the real world may contain several segments of contiguous identical characters. In order to reduce the processing space and speeding up the processing time of the SEQ-IC-LCS problem, we consider this problem of RLE sequences (abbreviated RLE-CLCS) in this chapter, which are formally defined as follows.

Problem 6.1. ***(RLE-CLCS)*** *Given two sequences X, Y and a constrained pattern P of lengths m, n, and d with M, N, and D runs, respectively, the RLE-CLCS problem is to find a longest sequence that is a CS of X and Y and includes P as a subsequences.*

Throughout this chapter, the format of input sequences is defined as follows. Let $X = x_1x_2\dots x_m$, $Y = y_1y_2\dots y_n$, and $P = p_1p_2\dots p_d$ denote two sequences and a constrained pattern over a finite alphabet Σ, respectively. Let $\widehat{X} = \alpha_1^{m_1}\alpha_2^{m_2}\dots\alpha_M^{m_M}$ denote the RLE representation of X, where α_i is the unique character of the i-th run in X and m_i is the length of this run. We also let $\widehat{X}[i..j]$ denote the RLE substring of X from run i to run j if $1 \le i \le j \le M$, and an empty string otherwise. Likewise, $\widehat{Y} = \beta_1^{n_1}\beta_2^{n_2}\dots\beta_N^{n_N}$ and $\widehat{P} = \gamma_1^{d_1}\gamma_2^{d_2}\dots\gamma_D^{d_D}$ denote the RLE representations of Y and P, and $\widehat{Y}[i..j]$ for $1 \le i \le j \le N$ and $\widehat{P}[i..j]$ for $1 \le i \le j \le D$ denote the RLE substrings of Y and P from run i to run j, respectively.

In the following, we show some properties of the RLE-CLCS problem for devising a simple algorithm, and then present the other algorithm for speeding up the computation.

6.1 A Simple Algorithm

Lemma 6.1 shows the characterization of an optimal solution to the RLE-CLCS problem based on the solutions to its smaller subproblems.

Lemma 6.1. *Let $CLCS(X, Y, P)$ denote the length of a longest sequence which is a CS of X and Y and includes P as a subsequence. Given two distinct symbols σ_1, σ_2 and three nonnegative integers s_1, s_2, s_3, the following properties hold:*

(1) *$CLCS(X\sigma_1^{s_1}, Y\sigma_1^{s_2}, P\sigma_1^{s_3}) = CLCS(X\sigma_1^{s_1-d}, Y\sigma_1^{s_2-d}, P\sigma_1^{s_3-d}) + d$, where $d =$*

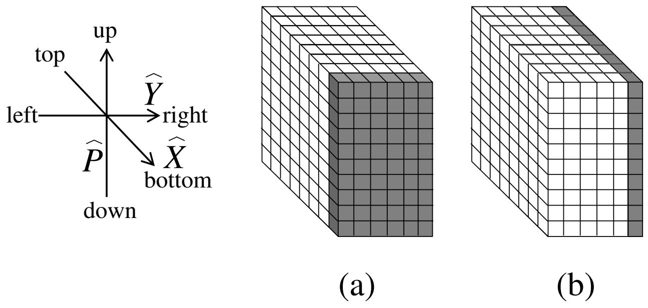

Figure 6.1: A cuboid corresponding a triple run. (a) The bottom border-page. (b) The right border-page.

$\min\{s_1, s_2, s_3\}$.

(2) $CLCS(X\sigma_1^{s_1}, Y\sigma_1^{s_2}, P\sigma_2^{s_3}) = CLCS(X\sigma_1^{s_1-d}, Y\sigma_1^{s_2-d}, P\sigma_2^{s_3}) + d$, *where* $d = \min\{s_1, s_2\}$.

(3) $CLCS(X\sigma_1^{s_1}, Y\sigma_2^{s_2}, P) = \max\{CLCS(X\sigma_1^{s_1}, Y, P), CLCS(X, Y\sigma_2^{s_2}, P)\}$.

Similar to the RLE-LCS problem, the runs of X, Y, and P induce the partition of the DP matrix for the CLCS problem into run-sized cuboids. We can visualize the three-dimensional DP matrix as $M \times N \times Q$ cuboids, and each cuboid corresponds to a triple of runs in $\widehat{X}$, $\widehat{Y}$, and $\widehat{P}$. A cuboid (i, j, k) is *fully match* if $\alpha_i = \beta_j = \gamma_k$ and is *partially match* if $\alpha_i = \beta_j$ and $\alpha_i \neq \gamma_k$. For the case of $\alpha_i \neq \beta_j$, the cuboid (i, j, k) is *mismatch*.

A six-dimensional matrix $\mathcal{C}$ is used for the bottom border-pages (see Figure 6.1(a) as an example) and right border-pages (see Figure 6.1(b) as an example) of cuboids, where $\mathcal{C}_{i,j,k}[m_i, h, l]$ holds the value of the h-th entry at level l in the bottom border-page of the cuboid (i, j, k) and $\mathcal{C}_{i,j,k}[v, n_j, l]$ holds the value of the v-th entry at level l in the right border-page of the cuboid (i, j, k). Notice that $\mathcal{C}$ is physically mapped to the original three-dimensional DP matrix, and we can employ two two-dimensional arrays for the bottom border-page and right border-page of each cuboid to implement the matrix $\mathcal{C}$. According to Lemma 6.1, the following recurrences can compute each entry $\mathcal{C}_{i,j,k}[v,h,l]$ for $v = m_i$ or $h = n_j$ with proper initialization.

1. Cuboid (i,j,k) is fully match:

$$\mathcal{C}_{i,j,k}[v,h,l] = \begin{cases} \mathcal{C}_{i-1,j,k}[m_{i-1}, h-v, l-v] + v & \text{if } l > v \text{ and } h > v, \\ \mathcal{C}_{i,j-1,k}[v-h, n_{j-1}, l-h] + h & \text{if } l > h \text{ and } v > h, \\ \mathcal{C}_{i-1,j-1,k}[m_{i-1}, n_{j-1}, l-h] + h & \text{if } l > h \text{ and } v = h, \\ \mathcal{C}_{i-1,j,k-1}[m_{i-1}, h-v, q_{k-1}] + v & \text{if } h > v \text{ and } v \geq l, \\ \mathcal{C}_{i,j-1,k-1}[v-h, n_{j-1}, q_{k-1}] + h & \text{if } v > h \text{ and } h \geq l, \\ \mathcal{C}_{i-1,j-1,k-1}[m_{i-1}, n_{j-1}, q_{k-1}] + h & \text{if } v = h \text{ and } h \geq l. \end{cases} \quad (6.1)$$

2. Cuboid (i,j,k) is partially match:

$$\mathcal{C}_{i,j,k}[v,h,l] = \begin{cases} \mathcal{C}_{i-1,j,k}[m_{i-1}, h-v, l] + v & \text{if } h > v, \\ \mathcal{C}_{i,j-1,k}[v-h, n_{j-1}, l] + h & \text{if } v > h, \\ \mathcal{C}_{i-1,j-1,k}[m_{i-1}, n_{j-1}, l] + h & \text{if } v = h, \end{cases} \quad (6.2)$$

3. Cuboid (i,j,k) is mismatch:

$$\mathcal{C}_{i,j,k}[v,h,l] = \max\{\mathcal{C}_{i-1,j,k}[m_{i-1}, h, l], \mathcal{C}_{i,j-1,k}[v, n_{j-1}, l]\}. \quad (6.3)$$

$\mathcal{C}_{M,N,D}[m_M, n_N, d_D]$ gives the length of a longest sequence that is a CS between $\widehat{X}$ and $\widehat{Y}$ and includes $\widehat{P}$ as a subsequence. Because there are $O(d \times (Mn + mN - MN))$ entries needed to compute and the computation of each entry requires $O(1)$ time, the following theorem is stated.

Theorem 6.1. *The RLE-CLCS problem can be solved in $O(d \times (Mn + mN - MN))$ time and space.*

6.2 A Faster Algorithm

Let $\widehat{\mathcal{L}}[i, j, k]$ denote the length of a longest sequence which is a CS of $\widehat{X}[1..i]$ and $\widehat{Y}[1..j]$ and includes $\widehat{P}[1..k]$ as a subsequence, i.e., $\widehat{\mathcal{L}}[i, j, k] = \mathcal{C}_{i,j,k}[m_i, n_j, d_k]$. Because the $\widehat{\mathcal{L}}$-values might increase only when a match run between $\widehat{X}$ and $\widehat{Y}$ is encountered, in this section we extend the idea of Algorithm RLE-LCS shown in Section 5.3 to the bottom border-pages of the fully match and partially match cuboids for speeding up the computation.

The algorithm proceeds to the match runs between $\widehat{X}$ and $\widehat{Y}$ at each position of P in row-major order. A linked list $\mathcal{M}[i]$, for each run i in $\widehat{X}$, is prepared for keeping an increasing list of the positions of the runs j in $\widehat{Y}$ such that $\alpha_i = \beta_j$. When row i at level l of layer k is processed, the bounded heap $\mathcal{H}_{k,l}$ has kept a position with the corresponding $\mathcal{C}$-value for each $j'' \in \{1, 2, \ldots, N\}$ such that the value is the maximum among the positions (i'', j'') for all $i'' \in \{1, 2, \ldots, i\}$, and empty if no such position exists. For each j in $\mathcal{M}[i]$, we perform the operation *Heap.Max*($\mathcal{H}$ to obtaining the position (i^*, j^*) for $j^* < j$ with the maximum value among all positions in $\mathcal{H}_{k,l}$, and then set $\mathcal{T}_{i,k}[j, l] = \mathcal{C}_{i^*,j^*,k}[m_{i^*}, n_{j^*}, l]$.

In the original matrix, we can conclude that a path beginning at $\mathcal{C}_{i,j,k}[1, 1, 1]$ of a cuboid (i, j, k) is a monotonically nondecreasing path and convert it into a path that consists of the following subpaths:

- a path that traverses the fully match cuboids by strictly three-dimensional diagonal moves (and strictly plane diagonal moves in some situation),
- a path that traverses the partially match cuboids by strictly plane diagonal moves, and
- a path that traverses the mismatch cuboids by either strictly plane horizontal moves or strictly plane vertical moves.

By the property of path conversion, we can determine $\mathcal{C}_{i,j,k}[v, h, l]$ by the position on row $i-1$ at layer k or on row i at layer $k-1$ where the path passing through entry (m_i, h, l) of cuboid (i, j, k) crosses. Figure 6.2 illustrates the idea of our approach to the RLE-CLCS problem. For any entry (m_i, h, l) of cuboid (i, j, k), suppose that the converted path passing through it crosses row $i-1$ at entry (m_{i-1}, h', l) of cuboid $(i-1, j', k)$. The value given by entry (m_i, h, l) of cuboid (i, j, k), denoted by $\mathcal{C}_{i,j,k}[m_i, h, l]$, is derived from the maximum length of two paths ending at the entry over the LCS matric function. One is passing through entry (m_{i-1}, h', l') of cuboid $(i-1, j', k)$,

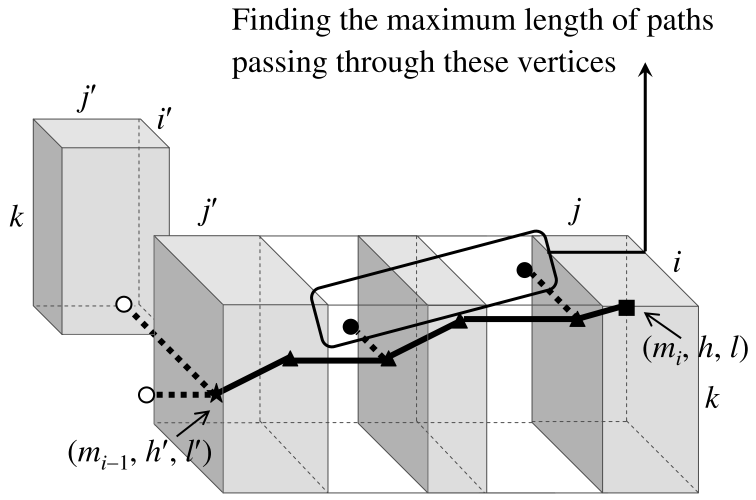
Finding the maximum length of paths
passing through these vertices
j'
i'
k
j'
j
i
(m_i, h, l)
k
(m_{i−1}, h', l')

Figure 6.2: Solving the RLE-CLCS problem.

and the other is with the maximum length amongst such the paths passing through entry $(m_{i-1}, n_{j''}, l'')$ of cuboid $(i-1, j'', k)$ for all $j'' \in \{j'+1, j'+2, \ldots, j-1\}$ and $l'' \in \{l'+1, l'+2, \ldots, l-1\}$. In addition, the value given by entry (m_{i-1}, h', l'') of cuboid $(i-1, j', k)$ can be further derived from the maximum of the values given by entry $(m_{i-1}, n_{j'-1}, l')$ of cuboid $(i-1, j'-1, k)$ and entry $(m_{i''}, h', l')$ of cuboid (i', j', k), where i' denotes the index of the former run of $\widehat{X}[i]$ with the same character α_i in $\widehat{X}$.

Let $W_{i,k}[j, l]$ denote the maximum possible length of all paths passing through entry (m_i, h, l) of a fully or partially match cuboid (i, j, k), which equals to $\mathcal{T}_{i,k}[j, l]$ plus the number of occurrences of α_i from the position with maximum value to the end of Y.

In order to obtain the maximum length amongst the paths passing through entry $(m_{i-1}, n_{j''}, l'')$ of cuboid $(i-1, j'', k)$ for all $j'' \in \{j'+1, j'+2, \ldots, j-1\}$ and $l'' \in$

$\{l'+1, l'+2, \ldots, l-1\}$, a list $\varphi_{i,j,k,h}$ is prepared for maintaining the W-values whose positions of the fully or partially match cuboids are walked across by the path ending at entry (m_i, h, d_k) of cuboid (i, j, k). Notice that every entry $W_{i,k}[j, l]$ is contained in a unique list. We can obtain the maximum value within all queried ranges of the list $\varphi_{i,j,k,k}$ by employing Algorithm DSW-Max.

Summarizing the above observations, we can now deduce the following lemmas that provide the basis for efficiently computing the entries in the bottom border-page of a fully or partially match cuboid. Let $occ_i[j, h]$ denote the number of occurrences of α_i in the suffix $Y[\pi..m]$, where π is the position of the h-th entry of $\widehat{Y}[j]$ in Y.

Lemma 6.2. *If the path passing through entry (m_i, h, l) of cuboid (i, j, k) crosses row $i-1$ on entry (m_{i-1}, h', l') of cuboid $(i-1, j', k)$ for some $1 \leq j' \leq j$, $\mathcal{C}_{i,j,k}[m_i, h, l]$ is given by $\max\{\tau_1 - occ_i[j, h] + 1, \tau_2 + m_i\}$, where τ_1 and τ_2 are defined as follows.*

- *τ_1 is the maximum value within the range $[j'+1, j]$ of the list $\varphi_{i,j,k,h}$.*
- *$\tau_2 = \max\{\mathcal{C}_{i',j',k}[m_{i'}, h', l'], \mathcal{T}_{i,k}[j', l']\}$, where i' denotes the index of the former run of the i-th run with the same character α_i in $\widehat{X}$.*

Lemma 6.3. *If the path passing through entry (m_i, h, l) of cuboid (i, j, k) crosses layer $k-1$ on entry (v', h', d_{k-1}) of cuboid $(i, j', k-1)$ for some $1 \leq j' \leq j$, $\mathcal{C}_{i,j,k}[m_i, h, l]$ is given by $\max\{\tau_1 - occ_i[j, h] + 1, \tau_2 + m_i\}$, where τ_1 is the maximum value within the range $[j'+1, j]$ of the list $\varphi_{i,j,k,h}$ and $\tau_2 = \mathcal{T}_{i,k}[j', 0]$.*

Lemma 6.4. *If the path passing through entry (m_i, h, l) of cuboid (i, j, k) does not cross row $i - 1$ nor layer $k - 1$, the path does not yield a legal CS passing through entry (m_i, h, l) of cuboid (i, j, k), i.e., $\mathcal{C}_{i,j,k}[m_i, h, l] = -\infty$.*

After $\mathcal{C}_{i,j,k}[m_i, n_j, l]$ is computed, we place the item of position (i, j, k, l) with its value $\mathcal{C}_{i,j,k}[m_i, n_j, l]$ into the insertion list $\mathcal{I}$. At the end of the iteration on row i at level l of layer k, we perform the operation *Heap.IncreaseLength* for each entry in $\mathcal{I}$ on $\mathcal{H}_{k,l}$. After all fully match and partially match cuboids are processed, we query the position that has the maximum value amongst all positions in $\mathcal{H}_{D,d_D}$ with key smaller than $N+1$. The maximum value is the length of a solution sequence. We can obtain the sequence by tracing back through the backtracking links from the position with the maximum value until $i = 0$ or $j = 0$. Theorem 6.2 summarized the main result.

Theorem 6.2. *There exists an $O(d \times (\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N))$-time algorithm, using $O(d \times \min\{\rho_1, \rho_2\} + N + M)$ space, for computing the length of a longest sequence which is a CS of $\widehat{X}$ and $\widehat{Y}$ and includes $\widehat{P}$ as a subsequence, where R denotes the number of the match runs between $\widehat{X}$ and $\widehat{Y}$, and ρ_1 and ρ_2 denote the numbers of entries in the bottom border-pages and right border-pages of the partially match cuboids at the first layer, respectively.*

Proof. Before the algorithm starts, the numbers of entries in the bottom borders and right borders of the match blocks can be calculated to determine which of the input sequences is considered as sequence X.

For efficiently implementing the algorithm in space, we use linked lists $\mathcal{M}[i]$ for all $i \in \{1, 2, \ldots, M\}$ to keep the positions of the match runs between $\widehat{X}$ and $\widehat{Y}$. For each fully or partially match cuboid, we employ a two-dimensional array for its bottom border-page. We also employ a bounded heap of size N, which can be repeatly used at each position of P. Thus, computing the length of a solution sequence uses $O(d \times \min\{\rho_1, \rho_2\} + N + M)$ space.

Building the linked lists $\mathcal{M}$ takes $O(M + N)$ time over a finite alphabet [31]. The initializations on all bounded heaps $\mathcal{H}_{k,l}$ and matrix $\mathcal{C}$ take $O(d \times (\min\{\rho_1, \rho_2\} + N \log\log N))$ time in total. Building all lists φ takes $O(d \times \min\{\rho_1, \rho_2\})$ time, and solving the dynamic sliding-window maxima problem with the input sequences φ takes $O(d \times R)$ time because there are $d \times R$ elements in φ. Matrix $\mathcal{C}$ can be computed by employing Algorithm RLE-LCS d times with proper preprocesses, and Lemmas 6.2 to 6.4 compute each required entry of the matrix $\mathcal{C}$ in constant time. Summarizing the time complexity of all stages, we conclude that the time complexity of solving the RLE-CLCS problem is $O(d \times (\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N))$. □

6.3 Discussion

In this chapter we consider the RLE-CLCS problem for two sequences X, Y and a constrained pattern P of lengths m, n, and d with M runs, N runs, and D runs, respectively. We first gave a simple algorithm which takes $O(d \times (Mn + mN))$ time and space. The second proposed algorithm takes $O(d \times (\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N))$

time and $O(d \times \min\{\rho_1, \rho_2\} + N + M)$ space by adapting Algorithm RLE-LCS.

In 2009, Ann *et al.* [12] gave an algorithm for the RLE-CLCS algorithm requiring $O(d \times (\min\{\rho_1, \rho_2\} + MN) + \lambda)$ time, where λ denotes the number of the entries of whole boundaries of the fully match cuboids. Comparing with Ann *et al.*'s algorithm, our latter algorithm outperforms Ann *et al.*'s algorithm if $R = o(\frac{MN}{\log\log N} + \frac{\lambda}{d\log\log N} - N)$.

Chapter 7

Concluding Remarks

In this chapter, we summarize the results reported in this dissertation, and then describe several further directions regarding the problems studied in this dissertation.

7.1 Summary and Contributions

This dissertation studied three research topics related to the LCS problem, which are the CLCS (constrained LCS) problem, the HC-LCS (hybrid constrained LCS) problem, and the LCS and CLCS problems of RLE sequences.

In Chapter 3, we studied four variants of the LCS problem, which are the SEQ-IC-LCS, STR-IC-LCS, SEQ-EC-LCS, and STR-EC-LCS problems. Table 7.1 shows our results as well as the previous works. Four $O(mnd)$-time and $O(mnd)$-space algorithms were separately presented for the four problems, where m, n, and d denote the lengths of two sequences and a constrained pattern, respectively. In fact, the

Table 7.1: Previous results and ours for the CLCS problem

Problem	Single Pattern	w patterns for $w \geq 2$	Hardness
SEQ-IC-LCS	$O(m^2n^2d)$ time and space [72]	$O(mn \times \prod_{k=1}^{w} d_k)$	NP-complete [35]
	$O(mnd)$ time and space[†][29]		
	$O(mnd)$ time and space [17]		
	$O(dr \times \log\log n + n)$ time and $O(d \times (r + n))$ space [46]		
STR-IC-LCS	$O(mnd)$[†]	$O(mnd_1d_2)$ if $w = 2$	NP-complete
SEQ-EC-LCS	$O(mnd)$[†]	$O(mn \times \prod_{k=1}^{w} d_k)$	NP-complete
STR-EC-LCS	$O(mnd)$[†]	$O(mn \times \prod_{k=1}^{w} d_k)$	

[†] The space can be reduced to $O(d \times (m + n))$ by the space-saving strategy.

space requirement can be further reduced to $O(d \times (m + n))$ by applying the space-saving strategy for the LCS problem introduced in Section 2.2. We also studied the four problems with an arbitrary number of constrained patterns, which were shown to be NP-complete except for the STR-EC-LCS problem, and presented the exact algorithms for the four problems. Moreover, it can be further demonstrated that the four problems are special cases of sequence alignments with linear-scoring functions.

In Chapter 4, we studied the HC-LCS problem, which is a hybrid problem of the SEQ-IC-LCS and SEQ-EC-LCS problems. Figure 7.2 shows the results for our

Table 7.2: Our results for the HC-LCS problem

Problem	Approach	Time Complexity	Space Complexity
HC-LCS	A DP algorithm	$O(mnde)$	$O(mnde)^{\dagger}$
	Speeding up the computation	$O((der+n) \times \log\log n)$	$O(de \times (r+n))$

† The space can be reduced to $O(de \times (m+n))$ by the space-saving strategy.

two algorithms. The former algorithm, using traditional DP techniques, requires $O(mnde)$ time and space, where m, n, d, and e denote the lengths of two sequences and two constrained patterns, respectively. Let r be the total number of ordered pairs of positions at which two sequences match each other. The latter algorithm, restricting the computation on the positions of matches between two sequences, takes $O(der \log\log n)$ time with $O(n \log\log n)$ time for initialization. In the worse case, the latter algorithm requires $O(demn \log\log n)$ time. However, the complexity of $O((der+n) \times \log\log n))$ are superior to $O(mnde)$ if $r = o(\frac{mn}{\log\log n} - \frac{n}{de})$. Consequently, the latter algorithm outperforms the former algorithm if $r = o(n(\frac{m}{\log m} - 1))$. In fact, the space requirement of the HC-LCS problem can be reduced to $O(de \times (m+n))$ by applying the space-saving strategy for the LCS problem introduced in Section 2.2.

In Chapter 5, we considered the RLE-LCS problem. Let two sequences be of lengths m and n with M and N runs, respectively. We first introduced Ann *et al.*'s algorithm [11] which takes $O(\min\{\rho_1, \rho_2\} + MN)$ time and devised a linear-time algorithm for computing the maxima within a sliding window upon a numerical sequence where the window size is dynamic. We then modified Ann *et al.*'s approach by adapting Hunt-Szymanski strategy and the approach to the sliding-window maxima problem with a dynamic window size. The new algorithm, taking $O(\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N)$ time, outperforms Ann *et al.*'s algorithm if $R = o(N(\frac{M}{\log\log N} - 1))$, where R denotes the number of runs at which the two sequences match each other, and ρ_1 and ρ_2 denote the numbers of entries on the bottom borders and right borders of the blocks whose corresponding runs match each other, respectively.

In Chapter 6, we studied the RLE-LCS problem. Let two sequences and a constrained pattern be of lengths m, n, and d with M runs, N runs, and D runs, respectively. We presented two algorithms for the RLE-CLCS problem. The former algorithm requires $O(d \times (Mn + mN))$ time and space. By adapting our approach to the RLE-LCS problem shown in Section 5.3 and employing a bounded heap, we delivered the latter algorithm for the RLE-CLCS problem, which requires $O(d \times (\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N))$ time and $O(d \times \min\{\rho_1, \rho_2\} + N + M)$ space. The latter algorithm outperforms Ann *et al.*'s algorithm [12] with $O(d \times (\min\{\rho_1, \rho_2\} + MN) + \lambda)$ time if $R = o(\frac{MN}{\log\log N} + \frac{\lambda}{d\log\log N} - N)$, where λ denotes the number of the entries of whole boundaries of the fully match cuboids. Table 7.3 summarizes our results as well as the previous works for the RLE-LCS and RLE-CLCS problems.

Table 7.3: Previous results and ours for the RLE-LCS and RLE-CLCS problems

Problem	Previous Results (Time Complexity)	Our Results (Time Complexity)
RLE-LCS	$O(Mn + mN)$ [24]	$O(\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N)$
	$O((M + N + R) \times \log(M + N + R))$ [58]	
	$O(MN \log(MN))$ [14]	
	$O(Mn + mN)$ [15]	
	$O(Mn + mN - MN)$ [34]	
	$O(\min\{Mn, mN\})$ [53]	
	$O(\min\{\rho_1, \rho_2\} + MN)$ [11]	
RLE-CLCS	$O(d \times (\min\{\rho_1, \rho_2\} + MN) + \lambda)$ [12]	$O(d \times (Mn + mN))$
		$O(d \times (\min\{\rho_1, \rho_2\} + (R + N) \times \log\log N))$

7.2 Further Work

Several directions related to the LCS problem are worthy of further study, such as the LCS problems of two-dimensional sequences, the alignment models of the four CLCS problems with a scoring scheme of affine gap penalties, and the CLCS problems of approximate pattern occurrences, which are stated as follows.

- LONGEST COMMON SUBSEQUENCES OF TWO-DIMENSIONAL SEQUENCES. The issues we mentioned in this dissertation are sequence comparisons with one-dimensional sequences, such as texts, numerical sequences, and musical sequences. For those two-dimensional sequences like images, it may not work by directly deducing an approach to comparing such sequences from the known approaches to comparing one-dimensional sequences. Many studies have devoted to two-dimensional pattern matching [6, 7, 8, 10, 19, 33, 49, 79]. It would be interesting to consider the LCS problem of two-dimensional sequences.

- CONSTRAINED SEQUENCE ALIGNMENTS WITH AFFINE GAP PENALTIES. The constrained sequence alignment problems, a alignment model of the SEQ-IC-LCS problem, arose from applications in molecular biology [70] in 2003. Since then, much ink has been spent on the problem [28, 30, 38, 39, 40, 54, 64, 65, 73, 74]. For aligning nucleic sequences, the scoring scheme of affine gap penalties is commonly used. The problem with affine gap penalties would become challenging issues in sequence comparison.

- Approximate Constrained Longest Common Subsequences. Arslan and Eğecioğlu [17] have investigated the approximate SEQ-IC-LCS problem of finding a longest sequence which is a common subsequence of two sequences and contains a subsequence whose edit distance from the constrained pattern is less than a given positive integer parameter. It would be interesting to consider such a criterion for other CLCS problems studied in this dissertation.

Bibliography

[1] A.V. Aho, D.S. Hirschberg, and J.D. Ullman. Bounds on the complexity of the longest common subsequence problem. *Journal of ACM*, 23:1–12, 1976.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[4] C.E.R. Alves, N. Cáceres, and S.W. Song. An all-substrings common subsequence algorithm. *Discrete Applied Mathematics*, 156(7):1025–1035, 2008.

[5] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *Proceeding of the 2nd IEEE Data Compression Conference* (*DCC'92*), pp. 279–288, 1992.

[6] A. Amir, G, Benson, and M. Farach. An alphabet independent approach to two dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994.

[7] A. Amir, A. Butman, M. Crochemore, G.M. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. In *Proceeding of the 14th Annual Symposium on Combinatorial Pattern Matching* (*CPM'03*), pp. 17–31, 2003.

[8] A. Amir and E. Chencinski. Faster Two Dimensional Scaled Matching. *Algorithmica*, 56(2):214–234, 2010.

[9] A. Amir, G.M. Landau, and D. Sokol. Inplace run-length 2d compressed search. *Theoretical Computer Science*, 290(3):1361–1383, 2003.

[10] A. Amir, D. Tsur, and O. Kapah. Faster two dimensional pattern matching with rotations. In *Proceeding of the 15th Annual Symposium on Combinatorial Pattern Matching* (*CPM'04*), pp. 409–419, 2004.

[11] 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 sequences. *Information Processing Letters*, 108(6):360–364, 2008.

[12] H.Y. Ann, C.B. Yang, C.T. Tseng, and C.Y. Hor. Fast algorithms for computing the constrained LCS of run-length encoded strings. In *Proceedings of the 2009 International Conference on Bioinformatics and Computational Biology* (*BIOCOMP'09*), vol. 2, pp. 646–649, 2009.

[13] A. Apostolico and C. Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:315–336, 1987.

[14] A. Apostolico, G.M. Landau, and S. Skiena. Matching for run-length encoded sequences. *Journal of Complexity*, 15(1):4–16, 1999.

[15] O. Arbell, G.M. Landau, and J.S.B. Mitchell. Edit distance of run-length encoded sequences. *Information Processing Letters*, 83(6):307–314, 2002.

[16] V.L. Arlazarov, E.A. Dinic, M.A. Kronrod, and I.A. Faradzev, On economic construction of the transitive closure of a directed graph. *Soviet mathematics - Doklady*, 11(5):1209–1210, 1970 (in English).

[17] A.N. Arslan and Ö. Eğecioğlu. Algorithms for the constrained longest common subsequence problems. *International Journal of Foundations of Computer Science*, 16(6):1099–1109, 2005.

[18] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings of the 7th International Symposium on Sequence Processing and Information Retrieval* (*SPIRE'00*), pp. 39–48, 2000.

[19] R. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977.

[20] D. Bodson, K.R. McConnell, and R. Schaphorst. *FAX: Digital Facsimile Technology and Applications*, Artech House, Norwood, MA, 1989.

[21] P. Bonizzoni, G.D. Vedova, R. Dondi, G. Fertin, R. Rizzi, and S. Vialette. Exemplar longest common subsequence. *IEEE Transactions on Computational Biology and Bioinformatics*, 4(4):535–543, 2007.

[22] E.A. Breimer, M.K. Goldberg, and D.T. Lim. A learning algorithm for the longest common subsequence problem. *Journal of Experimental Algorithmics*, 8(2.1), 2003.

[23] G.S. Brodal, K. Kaligosi, I. Katriel, and M. Kutz. Faster algorithms for computing longest common increasing subsequence. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching* (*CPM'06*), pp. 330–341, 2006.

[24] H. Bunke and J. Csirik. An improved algorithm for computing the edit distance of run-length coded sequences. *Information Processing Letters*, 54(2):93–96, 1995.

[25] K.M. Chao and L. Zhang. *Sequence Comparison: Theory and Methods*, Springer, 2009.

[26] Y.C. Chen and K.M. Chao. On the generalized constrained longest common subsequence problems. *Journal of Combinatorial Optimization*, accepted, 2009.

[27] K.Y. Chen, P.H. Hsu, and K.M. Chao. Hardness of comparing two run-length encoded strings. *Journal of Complexity*, accepted, 2010.

[28] F.Y.L. Chin, N.L. Ho, T.W. Lam, and P.W.H. Wong. Efficient constrained multiple sequence alignment with performance guarantee. *Journal of Bioinformatics and Computational Biology*, 3(1):1–18, 2005.

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

[30] Y.S. Chung, C.L. Lu, and C.Y. Tang. Constrained sequence alignment: a general model and the hardness results. *Discrete Applied Mathematics*, 155:2471–2486, 2007.

[31] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, 2nd edition, MIT Press and McGraw-Hill, 2001.

[32] M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6): 1654–1673, 2003.

[33] J.J. Fanm and K.Y. Su. The design of efficient algorithms for two-dimensional pattern matching, *IEEE Transactions on Knowledge and Data Engineering*, 7(2):318–327, 1995.

[34] V. Freschi and A. Bogliolo. Longest common subsequence between run-length-encoded sequences: a new algorithm with improved parallelism. *Information Processing Letters*, 90(4):167–173, 2004.

[35] Z. Gotthilf, D. Hermelin, and M. Lewenstein. Constrained LCS: hardness and approximation. In *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching* (*CPM'08*), pp. 255–262, 2008.

[36] R.I. Greenberg. Fast and simple computation of all longest common subsequences. *Technical Report*, Department of Mathematical and Computer Sciences, Loyola University, Chicago, 2002.

[37] D. Gusfield. *Algorithms on Sequences, Trees, and Sequences*, Cambridge University Press, 1997.

[38] D. He and A.N. Arslan. A space-efficient algorithm for the constrained pairwise sequence alignment problem. *Genome Informatics*, 16(2):237–246, 2005.

[39] D. He and A.N. Arslan. A parallel algorithm for the constrained multiple sequence alignment problem. In *Proceedings of the 5th IEEE Symposium on Bioinformatics and Bioengineering* (*BIBE'05*), pp. 258–262, 2005.

[40] D. He, A.N. Arslan, and A.C.H. Ling. A fast algorithm for the constrained multiple sequence alignment problem. *Acta Cybernetica*, 17(4):701–717, 2005.

[41] D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of ACM*, 18:341–343, 1975.

[42] D.S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of ACM*, 24:664–675, 1977.

[43] W.J. Hsu and M.W. Du. New algorithms for the LCS problem. *Journal of Computer and System Sciences*, 29:133–152, 1984.

[44] G.S. Huang, J.J. Liu, and Y.L. Wang. Sequence alignment algorithms for run-length-encoded sequences. In *Proceedings of the 14th Annual International Computing and Combinatorics* (*COCOON'08*), pp. 319–330, 2008.

[45] J.W. Hunt and T.G. Szymanski. A fast algorithm for computing longest common subsequence. *Communications of ACM*, 20(5):350–353, 1977.

[46] C.S. Iliopoulos and M.S. Rahman. New efficient algorithms for the LCS and constrained LCS problems. *Information Processing Letters*, 106:13–18, 2008.

[47] C.S. Iliopoulos. and M.S. Rahman. A new efficient algorithm for computing the longest common subsequence. *Theory of Computing Systems*, 355–371, 2008.

[48] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615-23, 2001.

[49] J. Kärkkäinen and E. Ukkonen. Two and higher dimensional pattern matching in optimal expected time. In *Proceedings of the 5th annual ACM-SIAM symposium on Discrete algorithms*, pp.715–723, 1994.

[50] J.W. Kim, A. Amir, G.M. Landau, and K. Park. Computing similarity of run-length encoded sequences with affine gap penalty. *Theoretical Computer Science*, 395(2–3):268–282, 2008.

[51] G.M. Landau, E.W. Myers, and M. Ziv-Ukelson. Two algorithms for LCS consecutive suffix alignment. *Journal of Computer and System Sciences*, 73(7):1095–1117, 2007.

[52] J.J. Liu, G.S. Huang, Y.L. Wang, and R.C.T. Lee. Edit distance for a run-length-encoded sequence and an uncompressed sequence. *Information Processing Letters*, 105(1):12–16, 2007.

[53] J.J. Liu, Y.L. Wang, and R.C.T. Lee. Finding a longest common subsequence between a run-length-encoded sequence and an uncompressed sequence. *Journal of Complexity*, 24(2):173–184, 2008.

[54] C.L. Lu and Y.P. Huang. A memory-efficient algorithm for multiple sequence alignment with constraints. *Bioinformatics*, 21:20–30, 2005.

[55] D. Maier. The complexity of some problems on subsequences and supersequence. *Journal of ACM*, 25:322–336, 1978.

[56] V. Mäkinen, E. Ukkonen, and G. Navarro. Approximate matching of run-length compressed sequences. *Algorithmica*, 35(4):347–369, 2003.

[57] W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31, 1980.

[58] J.S.B. Mitchell. A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded sequences. *Technical Report*, Department of Applied Mathematics, SUNY, Stony Brook, NY, 1997.

[59] E.W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[60] E.W. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, 1999.

[61] N. Nakatsu, Y. Kambayashi, and S. Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Informatica*, 18:171–179, 1982.

[62] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[63] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. In *Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, pp. 2444–2448, 1988.

[64] Z.S. Peng and H.F. Ting. Time and space efficient algorithms for constrained sequence alignment. In *Proceedings of the 9th International Conference on Implementation and Application of Automata (CIAA'04)*, pp. 237–246, 2004.

[65] Y.H. Peng, C.B. Yang, K.S. Huang, and K.T. Tseng. An algorithm and applications to sequence alignment with weighted constraints. *International Journal of Foundations of Computer Science*, 21(1):51–59, 2010.

[66] P.A. Pevzner. *Computational Molecular Biology: an Algorithmic Approach*, The MIT Press, Cambridge, MA, 2000.

[67] C. Rick. Simple and fast linear space computation of longest common subsequence. *Information Processing Letters*, 75:275–281, 2000.

[68] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.

[69] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[70] C.Y. Tang, C.L. Lu, M.D. 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*, 1(2):267–287, 2003.

[71] R.E. Tarjan. *Data Structure and Network Algorithms*, CBMS 44 (Society for Industrial and Applied Mathematics, Philadephia, PA, 1983.

[72] Y.T. Tsai. The constrained longest common subsequence problem. *Information Processing Letters*, 88:173–176, 2003.

[73] Y.T. Tsai, Y.P. Huang, C.T. Yu, and C.L. Lu. MuSiC: a tool for multiple sequence alignment with constraints. *Bioinformatics*, 20:2309–2311, 2004

[74] H.J. Tsai, C.Y. Lin, Y.C. Chung, and C.Y. Tang. An Efficient Parallel Algorithm for Constraint Multiple Sequence Alignment. In *Proceedings of the International Computer Symposium* (*ICS'06*), pp. 1261–1266, 2006.

[75] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

[76] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

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

[78] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337-348, 1994.

[79] R.F. Zhu and T. Takaoka. A technique for two-dimensional pattern matching. *Communications of the ACM*, 32(9):1110–1120, 1989.