Theoretical Computer Science 687 (2017) 79–92

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

# A space efficient algorithm for the longest common subsequence in $k$-length substrings

Daxin Zhu[a], Lei Wang[b], Tinran Wang[c], Xiaodong Wang[d,*]

[a] Quanzhou Normal University, Quanzhou, China
[b] Facebook, 1 Hacker Way, Menlo Park, CA 94052, USA
[c] School of Mathematical Sciences, Peking University, Beijing, China
[d] Fujian University of Technology, Fuzhou, China

A R T I C L E  I N F O

*Article history:*
Received 9 February 2017
Received in revised form 17 May 2017
Accepted 19 May 2017
Available online 26 May 2017
Communicated by R. Giancarlo

*Keywords:*
Longest common subsequence
Similarity of strings
Edit distance
Dynamic programming

A B S T R A C T

Two space efficient algorithms to solve the $LCSk$ problem and $LCS_{\geq}k$ problem are presented in this paper. The algorithms improve the time and space complexities of the algorithms of Benson et al. [4]. The space cost of the first algorithm to solve the $LCSk$ problem is reduced from $O(n^2)$ to $O(kn)$, if the size of the two input sequences are both $n$. The time and space costs of the second algorithm to solve the $LCS_{\geq}k$ problem are both improved. The time cost is reduced from $O(kn^2)$ to $O(n^2)$, and the space cost is reduced from $O(n^2)$ to $O(kn)$. In the case of $k = O(1)$, the two algorithms are both linear space algorithms.

## 1. Introduction

The longest common subsequence (LCS) problem is a classic problem in computer science [8,17]. Given two sequences $A$ and $B$, the longest common subsequence (LCS) problem is to find a subsequence of $A$ and $B$ whose length is the longest among all common subsequences of the two given sequences. The problem has numerous applications in many apparently unrelated fields ranging from file comparison, pattern matching and computational biology [11]. The LCS problem has many variants, such as LCS alignment [13–15], constrained LCS [2,5,16,18,19], weighted LCS [1], restricted LCS [7] and LCS approximation [12].

In past years, many related sequence similarity problems, often motivated by computational biology, have also been studied. One of them, proposed very recently by Benson et al. [3,4], is the longest common subsequence in $k$-length substrings problem, in which the common subsequence is required to consist of $k$ or at least $k$ length substrings.

The $LCSk$ problem can be characterized as follows.

**Definition 1.** Given two sequences $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_m$, and an integer $k$, the $LCSk$ problem is to find the maximal length $l$ such that there are $l$ substrings, $a_{i_1} \cdots a_{i_1+k-1}, \cdots, a_{i_l} \cdots a_{i_l+k-1}$, identical to $b_{j_1} \cdots b_{j_1+k-1}, \cdots, b_{j_l} \cdots b_{j_l+k-1}$ where $\{a_{i_t}\}$ and $\{b_{j_t}\}$ are in increasing order for $1 \leq t \leq l$ and any two $k$-length substrings in the same sequence, do not overlap.

* Corresponding author.
*E-mail address:* wangxd139@139.com (X. Wang).

http://dx.doi.org/10.1016/j.tcs.2017.05.015
0304-3975/© 2017 Elsevier B.V. All rights reserved.

A similar problem is the LCS at least $k$ problem ($LCS_{\geq}k$). In this problem, the demand of matching substrings of length exactly $k$ is relaxed to the length of the matched substrings to be at least $k$. The length of the common substrings is further limited by $2k-1$ since a longer common substring contains two substrings each of length $k$ or more.

The $LCS_{\geq}k$ problem can be defined as follows.

**Definition 2.** Given two sequences $A = a_1a_2 \cdots a_n$ and $B = b_1b_2 \cdots b_m$, and an integer $k$, the $LCS_{\geq}k$ problem is to find substrings with maximal total length such that $a_{i_p} \cdots a_{i_p+k+t}$ is identical to $b_{j_p} \cdots b_{j_p+k+t}$ for $-1 \leq t \leq k-2$ where $\{a_{i_t}\}$ and $\{b_{j_t}\}$ are in increasing order for $1 \leq t \leq l$ and any two substrings in the same sequence, do not overlap.

In the case of $n = m$, Benson et al. [3] presented a dynamic programming algorithm to solve the $LCSk$ problem using $O(kn^2)$ time and $O(n^2)$ space. In the case of $k = O(1)$, the time complexity of the algorithm becomes $O(n^2)$. For unbounded $k$, the time complexity was further improved from $O(kn^2)$ to $O(n^2)$ [4,6]. If only the length of $LCSk$ has to be computed, the space cost of their algorithm can be reduced to $O(kn)$, but if an $LCSk$ has to be constructed, the whole table is needed in their algorithm, implying $O(n^2)$ space requirement. Applying the sparse dynamic programming paradigm of Hunt and Szymanski [10], Deorowicz et al. [6] presented an $O(n + r \log l)$ time and $O(r)$ space algorithm, where $r$ is the number of matches, $l \leq n/k$ is the solution length. If the number of matches in the dynamic programming matrix is large, an $O(n^2/k + n(k \log n)^{2/3})$ and $O(nl)$ space algorithm was also presented in [6] by using the observation that matches forming a longest common subsequence must be separated with gaps of size at least $k$. Its variant based on the van Emde Boas tree was also briefly discussed. Finally, a tabulation-based algorithm was presented, using $O(n^2/\log n)$ time and $O(n^2/\log n)$ space.

For the $LCS_{\geq}k$ problem, Benson et al. [4] presented a first dynamic programming algorithm to solve the problem using $O(kn^2)$ time and $O(n^2)$ space. If only the length of $LCS_{\geq}k$ has to be computed, the space cost of their algorithm can be reduced to $O(kn)$, but if an $LCS_{\geq}k$ has to be constructed, their algorithm requires $O(n^2)$ space.

Very recently, Ueki et al. presented a similar dynamic programming algorithm to solve the problem [16]. The algorithm was described very concisely in their paper since the main topic in their paper is to find the LCS in at least $k$ length order-isomorphic substrings problem. Their algorithm composes of two parts. In the first part, the longest common suffix problem for the same input strings is solved. Then, in the second part of the algorithm, a dynamic programming formula can be established by utilizing the solution array $L$ obtained in the first part. If the sizes of the two input strings are $m$ and $n$ respectively, their algorithm requires $O(mn)$ time and $O(mn)$ space since three tables of size $(m+1)(n+1)$ are used. If only the length of an $LCS_{\geq}k$ is required, the space complexity can be easily reduced to $O(kn)$. The algorithm presented in this paper for the same problem requires also $O(mn)$ time, but uses only $O(kn)$ space.

In this paper, we focus on the space efficient algorithms to solve the $LCSk$ problem and $LCS_{\geq}k$ problem. We present two new algorithms to solve the problems. The first algorithm is a dynamic programming algorithm to solve the $LCSk$ problem, using $O(mn)$ time and $O(kn)$ space, if the sizes of the input sequences are $n$ and $m$ respectively. In the case of $k = O(1)$, the algorithm is a linear space algorithm.

The second algorithm is an improved algorithm of Benson et al. to solve the $LCS_{\geq}k$ problem. The time complexity is reduced from $O(kmn)$ to $O(mn)$, and the space complexity is reduced from $O(mn)$ to $O(kn)$. In the case of $k = O(1)$, the algorithm is also a linear space algorithm.

The organization of the paper is as follows.

In the following 3 sections, we describe our improved algorithms of Benson et al to solve the $LCSk$ and $LCS_{\geq}k$ problems. In Section 2, we present an $O(kn)$ space algorithm for solving the $LCSk$ problem. In Section 3, the time and space costs of the algorithm of Benson et al to solve the $LCS_{\geq}k$ problem are reduced to $O(mn)$ and $O(kn)$. Some concluding remarks are placed in Section 4.

## 2. An $O(kn)$ space algorithm for solving the $LCSk$ problem

### 2.1. The description of the algorithm

As stated in [8], $LCSk$ can be solved by using a dynamic programming algorithm. Let $d(i, j)$ denote the length of the longest match between the prefixes of $A[1:i] = a_1a_2 \cdots a_i$ and $B[1:j] = b_1b_2 \cdots b_j$. Then, $d(i, j)$ can be computed recursively as follows.

$$d(i,j) = \begin{cases} 1 + d(i-1, j-1) & \text{if } a_i = b_j, \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Let $f(i, j)$ denote the number of $k$ matchings in the longest common subsequence, consisting of $k$ matchings in the prefixes $A[1:i]$ and $B[1:j]$. Then, $f(i, j)$ can be computed recursively as follows.

$$f(i,j) = \max \begin{cases} f(i-1, j) \\ f(i, j-1) \\ f(i-k, j-k) + \delta(d(i,j)) \end{cases} \tag{2}$$