Theoretical Computer Science 638 (2016) 11–26

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

LCSk: A refined similarity measure ☆

G. Benson [a,1], A. Levy [b,*], S. Maimoni [b], D. Noifeld [b], B.R. Shalom [b]

[a] Department of Computer Science, Boston University, Boston, MA 02215, United States
[b] Department of Software Engineering, Shenkar College, Ramat-Gan 52526, Israel

ARTICLE INFO

Article history:
Received 26 March 2015
Received in revised form 24 September 2015
Accepted 20 November 2015
Available online 28 November 2015

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

ABSTRACT

In this paper we define a new similarity measure: LCSk, aiming at finding the maximal number of k length substrings matching in both input strings while preserving their order of appearance, for which the traditional LCS is a special case, where $k = 1$. We examine this generalization in both theory and practice. We first describe its basic solution and give an experimental evidence in real data for its ability to differentiate between sequences that are considered similar according to the LCS measure. We then examine extensions of the LCSk definition to LCS in at least k-length substrings ($LCS_{\geq}k$) and 2-dimensional LCSk and also define complementary EDk and $ED_{\geq}k$ distances.

1. Introduction

The *Longest Common Subsequence* problem, whose first famous dynamic programming solution appeared in 1974 [23], is one of the classical problems in computer science. The LCS problem is motivated by the need to measure similarity of sequences and strings, and it has been very well studied (for a survey, see [8]). The well-known dynamic programming solution [13] requires running time of $O(n^2)$, for two input strings of length n.

The LCS problem has been investigated on more general structures such as trees and matrices [5], run-length encoded strings [6], weighted sequences [4,9] and more. Many variants of the LCS problem were studied as well, among which LCS alignment [19,18,17], constrained LCS [22,10], restricted LCS [12] and LCS approximation [16].[2]

LCS has been also used to measure similarity for biological sequence comparison. Its strength lies in its simplicity which has allowed development of an extremely fast, bit-parallel computation which uses the bits in a computer word to represent adjacent cells in a row of the LCS scoring matrix and computer logic operations to calculate the scores from one row to the next [3,11,14]. For example, in a recent experiment, 25,000,000 bit-parallel LCS computations (sequence length = 63) took approximately 7 seconds on a typical desktop computer [7] or about 60 times faster than a standard algorithm. This speed makes the LCS attractive for sequence comparison performed on high-sequencing data. The disadvantage of the LCS is that it is a crude measure of similarity because consecutive matching letters in the LCS can have different spacings in the two sequences, i.e., there is no penalty for insertion and deletion. Indeed, as is well-known, there is a strong relation between the total number of insertions and deletions and the LCS. However, there are no limitations on the "distribution" of such

☆ A preliminary version of this paper appeared in SISAP 2013.

* Corresponding author.
E-mail addresses: gbenson@bu.edu (G. Benson), avivitlevy@shenkar.ac.il (A. Levy), rivash@shenkar.ac.il (B.R. Shalom).

[1] This work was partially funded by NSF grants IIS-1017621 and IIS-1423022.

[2] We alternately use the terms *string* and *sequence* throughout the paper. Nevertheless, a *subsequence* is obtained from a sequence by deleting symbols at any index we want, while a *substring* is a consecutive part of the string.

http://dx.doi.org/10.1016/j.tcs.2015.11.026
0304-3975/

	1	2	3	4	5	6	7	8
A =	T	G	C	**G**	**T**	G	**T**	**G**
B =	**G**	**T**	T	G	**T**	**G**	C	C

Fig. 1. An LCS2 example.

insertions and deletions. Consider for example the sequences: $A = (GTG)^{n/3}$ and $B = (TCC)^{n/3}$, for which the LCS is quite large – of size $n/3$, but there are no two matched symbols consecutive in both sequences. Any common subsequence of this size "puts together" separated elements implying a rather "artificial" similarity.

A possible direction is to use affine gap penalties [15], i.e., to include penalties for opening new gaps besides penalties for individual insertions and deletions that merely extend the gaps. However, gap penalties are not flexible enough to force subsequent matches on the one hand (i.e., no gaps at all between matches) while minimizing the total number of gaps on the other hand (to maximize similarity). We propose a definition of LCS that is more accurate because it enables forcing letters to be adjacent in both sequences. In our problem, the common subsequence is required to consist of k or at least k length substrings.

The LCSk problem A formal definition appears in Definition 1.

Definition 1. *The Longest Common Subsequence in k Length Substrings Problem (LCSk)*:
Input: Two sequences $A = a_1a_2 \ldots a_n$, $B = b_1b_2 \ldots b_n$ over alphabet Σ.
Output: The maximal ℓ s.t. there are ℓ substrings, $a_{i_1} \ldots a_{i_{1+k-1}}, \ldots, a_{i_\ell} \ldots a_{i_\ell+k-1}$, identical to $b_{j_1} \ldots b_{j_{1+k-1}}, \ldots, b_{j_\ell} \ldots b_{j_\ell+k-1}$ where $\{a_{i_f}\}$ and $\{b_{j_f}\}$ are in increasing order for $1 \le f \le \ell$ and where two k length substrings in the same sequence, do not overlap.

We demonstrate LCSk using the sequences of Fig. 1. A possible common subsequence for ($k = 2$) is $G\ T\ T\ G$ obtained from a_4, a_5, a_7, a_8 and b_1, b_2, b_5, b_6. Though $a_6 = b_4$, and such a match preserves the order of the common subsequence, it cannot be added to the common subsequence in pairs, since it is a match of a single symbol. For $k = 3$, one of the possible solutions is $T\ G\ C$ achieved by matching a_1, a_2, a_3, with b_5, b_6, b_7.

The LCSk problem is a generalization of the LCS problem. However, using the LCS algorithm on the input sequences, then backtracking the dynamic programming table to mark the symbols participating in the common subsequence and checking whether those symbols appear in consecutive k length substrings in both input sequences (deleting them if not), indeed guarantees a common subsequence in k length substrings but not necessarily of optimal length. Hence, an algorithm designed for LCSk is required. The matchings of k consecutive symbols in the LCSk problem is called a *k matching* throughout this paper.

The LCS at least k problem ($LCS_{\ge}k$) LCSk measures similarity while limiting the fragmentation of the matched subsequence. The demand of matching substrings of length exactly k can be relaxed without decreasing the proximity demand by setting the length of the matched substrings to be at least k, achieving an even tighter similarity measure since longer common subsequence can be reported without refraining from the demand that the matches will be adjacent. Consider, for example, the sequences of Fig. 1. For $k = 2$ there is the solution of the 2 matchings GT and TG. However, the second 2 matching can be extended to a 3 matching GTG without overlapping GT, thus, a longer common subsequence consisting of substrings of length at least 2 can be obtained.

The length of the common substrings is limited by $2k - 1$ since a longer common substring contains two substrings each of length k or more. Thus, for substrings longer than $2k - 1$ a larger k can be used in the definition of the problem. Since in the $LCS_{\ge}k$ problem, not all matches are of the same length, it makes more sense to maximize the total length of the common substrings of length at least k, rather than maximizing the number of matches.

Definition 2. *The Longest Common Subsequence in at least k Length Substrings Problem ($LCS_{\ge}k$)*:
Input: Two sequences $A = a_1a_2 \ldots a_n$, $B = b_1b_2 \ldots b_n$ over alphabet Σ.
Output: The maximal ℓ s.t. there are substrings with total length ℓ, $a_{i_p} \ldots a_{i_p+k+f}$ is identical to $b_{j_p} \ldots b_{j_p+k+f}$ for $-1 \le f \le k-2$ where $\{a_{i_p}\}$ and $\{b_{j_p}\}$ are in increasing order and every two substrings in the same sequence, do not overlap.

Paper organization The paper is organized as follows. The basic solution for the LCSk problem is given in Section 2. Section 3 describes experiments comparing LCS and LCSk on real data. The relaxed problem of $LSC_{\ge}k$ is discussed in Section 4. Section 5 refers to the complementary edit distance measures, EDk and $ED_{\ge}k$. In Section 6 we discuss a two-dimensional extension of the LSCk. Section 7 concludes the paper.

2. Solving the LCSk problem

We first note that, as other LCS variants, LCSk can be solved using a dynamic programming algorithm. Every table cell $[i, j]$ keeps a diagonal counter, named *dcount*, counting the length of the longest match between the suffixes of $A[1 \ldots i]$