ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

# New tabulation and sparse dynamic programming based techniques for sequence similarity problems

Szymon Grabowski

*Lodz University of Technology, Institute of Applied Computer Science, Al. Politechniki 11, 90–924 Łódź, Poland*

ARTICLE INFO

*Article history:*
Received 5 October 2014
Received in revised form 19 July 2015
Accepted 27 October 2015
Available online xxxx

*Keywords:*
Sequence similarity
Longest common subsequence
Sparse dynamic programming
Tabulation

ABSTRACT

Calculating the length $\ell$ of a longest common subsequence (LCS) of two strings, $A$ of length $n$ and $B$ of length $m$, is a classic research topic, with many known worst-case oriented results. We present three algorithms for LCS length calculation with respectively $O(mn \lg\lg n/\lg^2 n)$, $O(mn/\lg^2 n + r)$ and $O(n + r)$ time complexity, where the second one works for $r = o(mn/(\lg n \lg\lg n))$, and the third one for $r = \Theta(mn/\lg^k n)$, for a real constant $1 \le k \le 3$, and $\ell = O(n/(\lg^{k-1} n(\lg\lg n)^2))$, where $r$ is the number of matches in the dynamic programming matrix. We also describe conditions for a given problem sufficient to apply our techniques, with several concrete examples presented, namely the edit distance, the longest common transposition-invariant subsequence (LCTS) and the merged longest common subsequence (MerLCS) problems.



## 1. Introduction

Measuring the similarity of sequences is an old research topic and many actual measures are known in the string matching literature. One classic example concerns the computation of a longest common subsequence (LCS) in which a subsequence that is common to all sequences and has the maximal possible length is looked for. A simple dynamic programming (DP) solution works in $O(mn)$ time for two sequences of length $n$ and $m$, respectively, but faster algorithms are known. The LCS problem has many applications in diverse areas, like version control systems, comparison of DNA strings, structural alignment of RNA sequences. Other related problems comprise calculating the edit (Levenshtein) distance between two sequences, the longest common transposition-invariant subsequence, or LCS with constraints in which the longest common subsequence of two sequences must contain, or exclude, some other sequence.

Let us focus first on the LCS problem, for two sequences $A$ and $B$. It is defined as follows. Given two sequences, $A = a_1 \ldots a_n = A[1 \ldots n]$ and $B = b_1 \ldots b_m = B[1 \ldots m]$, over an alphabet $\Sigma$ of size $\sigma$, find a longest subsequence $\langle a_{i_1}, a_{i_2}, \ldots, a_{i_\ell}\rangle$ of $A$ such that $a_{i_1} = b_{j_1}, a_{i_2} = b_{j_2}, \ldots, a_{i_\ell} = b_{j_\ell}$, where $1 \le i_1 < i_2 < \cdots < i_\ell \le n$ and $1 \le j_1 < j_2 < \cdots < j_\ell \le m$. The found sequence may not be unique. W.l.o.g. we assume $n \ge m$. To avoid uninteresting complications, we also assume that $m = \Omega(\lg^2 n)$. Additionally, we assume that $\sigma = O(m)$. The case of a general alphabet, however, can be handled with standard means, i.e., we can initially map the sequences $A$ and $B$ onto an alphabet of size $\sigma' = O(m)$, in $O(n \lg \sigma')$ time, using a balanced binary search tree. We do not comprise this tentative preprocessing step in further complexity considerations.

Often, a simplified version of the LCS problem is considered, when one is interested in telling only the length of a longest common subsequence (LLCS).

In this paper we present three techniques for finding the LCS length, one (Section 3) based on tabulation and improving the result of Bille and Farach-Colton [3] by a factor of $\lg\lg n$, second (Section 4) combining tabulation and sparse dynamic

*E-mail address:* sgrabow@kis.p.lodz.pl.

http://dx.doi.org/10.1016/j.dam.2015.10.040
0166-218X/

**Table 1**
Summary of main results for finding $\ell$, the length of a longest common subsequence (LLCS) for two sequences of length $m$ and $n$, respectively, where $m \leq n$, over an integer alphabet of size $\sigma$. The number of pairs of symbols shared by the two input sequences is denoted with $r$, $0 \leq r \leq mn$. The number of dominant matches is denoted with $D$, $D \leq r$. All the algorithms require (at most) linear extra space in addition to the space for storing the input sequences. "Stand. DP" and "Stand. 4R" are abbreviations for the standard dynamic programming and the standard Four Russians (tabulation) algorithm.

| Algorithm | Time complexity | Conditions | Ref. |
|---|---|---|---|
| Stand. DP | $O(mn)$ | – | folklore |
| Stand. 4R | $O(n + mn/\lg^2 n)$ | $\sigma = O(1)$ and $m \geq \lg n$ | [15] |
| Sparse DP | $O(n\sigma + D \lg\lg(\min(D, mn/D)))$ | – | [8] |
| BFC | $O(n + mn(\lg\lg n)^2/\lg^2 n)$ | $m \geq \lg n \lg\lg n$ | [3] |
| Our-1 | $O(mn \lg\lg n/\lg^2 n)$ | $m \geq \lg^2 n$ | Section 3 |
| Our-2 | $O(mn/\lg^2 n + r)$ | $m \geq \lg^2 n$ and $r = o(mn/(\lg n \lg\lg n))$ | Section 4 |
| Our-3 | $O(n + r)$ | $m \geq \lg^2 n$ and $r = \Theta(mn/\lg^k n)$, for real $k \in [1, 3]$, and $\ell = O(n/(\lg^{k-1} n(\lg\lg n)^2))$ | Section 5 |

programming and being slightly faster if the number of matches is appropriately limited, and the last (Section 5) improving the previous result if also the LCS length is conveniently limited. In Section 6 we show the conditions necessary to apply these algorithmic techniques. Some other, LCS-related, problems fulfill these conditions, so we immediately obtain new results for these problems as well. The summary of our results for the LLCS-finding problem is presented in Table 1. The other results listed there are briefly discussed in Sections 2 and 3.

Throughout the paper, we assume the word-RAM model of computation with machine word size $w \geq \lg n$. All used logarithms are base 2.

A preliminary version of this article appeared in Proc. PSC 2014 [10].

## 2. Related work

A standard solution to the LCS problem is based on dynamic programming, and it is to fill a matrix $M$ of size $(n+1) \times (m+1)$, where $n+1$ is the number of columns, $m+1$ the number of rows, and each cell value depends on a pair of compared symbols from $A$ and $B$ (that is, only if they match or not), and its (at most) three already computed neighbor cells. Each computed $M[i, j]$ cell, $1 \leq i \leq n$, $1 \leq j \leq m$, stores the value of $LLCS(A[1 \ldots i], B[1 \ldots j])$. A well-known property describes adjacent cells: $M(i, j) - M(i-1, j) \in \{0, 1\}$ and $M(i, j) - M(i, j-1) \in \{0, 1\}$ for all valid $i, j$.

Despite almost 40 years of research, surprisingly little can be said about the worst-case time complexity of LCS. It is known that in the very restrictive model of unconstrained alphabet and comparisons with equal/unequal answers only, the lower bound is $\Omega(mn)$ [21], which is reached by a trivial DP algorithm. If the input alphabet is of constant size, the known lower bound is simply $\Omega(n)$, but if total order between alphabet symbols exists and $\leq$-comparisons are allowed, then the lower bound grows to $\Omega(n \lg n)$ [11]. In other words, the gap between the proven lower bounds and the best worst-case algorithm is huge.

A simple idea proposed in 1977 by Hunt and Szymanski [13] has become a milestone in LCS research, and the departure point for theoretically better algorithms (e.g., [8]). The Hunt–Szymanski (HS) algorithm is essentially based on dynamic programming, but it visits only the matching cells of the matrix, typically a small fraction of the entire set of cells. This kind of selective scan over the DP matrix is called *sparse dynamic programming* (SDP). We note that the number of all matches in $M$, denoted with the symbol $r$, can be found in $O(n)$ time, and after this (negligible) preprocessing we can decide if the HS approach is promising to given data. More precisely, the HS algorithm works in $O(n + r \lg m)$ or even $O(n + r \lg\lg m)$ time. Note that in the worst case, i.e., for $r = \Theta(mn)$, this complexity is however superquadratic.

The Hunt–Szymanski concept was an inspiration for a number of subsequent algorithms for LCS calculation, and the best of them, the algorithm of Eppstein et al. [8], achieves $O(D \lg\lg(\min(D, mn/D)))$ worst-case time (plus $O(n\sigma)$ preprocessing), where $D \leq r$ is the number of so-called dominant matches in $M$ (a match $(i, j)$ is called dominant iff $M[i, j] = M[i-1, j]+1 = M[i, j-1] + 1$). Note that this complexity is $O(mn)$ for any value of $D$. A more recent algorithm, by Sakai [19], is an improvement if the alphabet is very small (in particular, constant), as its time complexity is $O(m\sigma + \min(D\sigma, \ell(m-q)) + n)$, where $\ell = LLCS(A, B)$ and $q = LLCS(A[1 \ldots m], B)$.

A different approach is to divide the dynamic matrix into small blocks, such that the number of essentially different blocks is small enough to be precomputed before the main processing phase. In this way, the block may be processed in constant time each, making use of a built lookup table (LUT). This "Four Russians" technique was first used to the LCS problem by Masek and Paterson [15], for a constant alphabet, and refined by Bille and Farach-Colton [3] to work with an arbitrary alphabet. The obtained time complexities were $O(mn/\lg^2 n)$ and $O(mn(\lg\lg n)^2/\lg^2 n)$, respectively, with linear space.

A related, but different approach, is to use bit-parallelism to compute several cells of the dynamic programming matrix at a time. There are a few such variants (see [14] and references therein), all of them working in $O(\lceil m/w \rceil n)$ worst-case time, after $O(\sigma \lceil m/w \rceil + m)$-time and $O(\sigma m)$-space preprocessing, where $w$ is the machine word size.

Please cite this article in press as: S. Grabowski, New tabulation and sparse dynamic programming based techniques for sequence similarity problems, Discrete Applied Mathematics (2015), http://dx.doi.org/10.1016/j.dam.2015.10.040