Journal of Discrete Algorithms 6 (2008) 570–581

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Semi-local longest common subsequences in subquadratic time

Alexander Tiskin

Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom

A R T I C L E   I N F O

Article history:
Available online 20 August 2008

Keywords:
String algorithms
Longest common subsequence
Semi-local string comparison

A B S T R A C T

For two strings a, b of lengths m, n, respectively, the longest common subsequence (LCS) problem consists in comparing a and b by computing the length of their LCS. In this paper, we define a generalisation, called "the all semi-local LCS problem", where each string is compared against all substrings of the other string, and all prefixes of each string are compared against all suffixes of the other string. An explicit representation of the output lengths is of size $\Theta((m+n)^2)$. We show that the output can be represented implicitly by a geometric data structure of size $O(m+n)$, allowing efficient queries of the individual output lengths. The currently best all string–substring LCS algorithm by Alves et al., based on previous work by Schmidt, can be adapted to produce the output in this form. We also develop the first all semi-local LCS algorithm, running in time $o(mn)$ when m and n are reasonably close. Compared to a number of previous results, our approach presents an improvement in algorithm functionality, output representation efficiency, and/or running time.

1. Introduction

Given two strings a, b of lengths m, n, respectively, the longest common subsequence (LCS) problem consists in comparing a and b by computing the length of their LCS. In this paper, we define a generalisation, called "the all semi-local LCS problem", where each string is compared against all substrings of the other string, and all prefixes of each string are compared against all suffixes of the other string. The all semi-local LCS problem arises naturally in the context of LCS computations on substrings. It is closely related to local sequence alignment (see e.g. [10,12]) and to approximate string matching (see e.g. [9,18]).

A standard approach to string comparison is representing the problem as an alignment dag (directed acyclic graph) of size $\Theta(mn)$ on an $m \times n$ grid of nodes. The basic LCS problem, as well as its many generalisations, can be solved by dynamic programming on this dag in time $O(mn)$ (see e.g. [9,10,12,18]). It is well known (see e.g. [2,16] and references therein) that all essential information in the alignment dag can in fact be represented by a data structure of size $O(m+n)$. In this paper, we expose a rather surprising (and to the best of our knowledge, previously unnoticed) connection between this linear-size representation of the string comparison dag, and a standard computational geometry problem known as dominance counting.

If the output lengths of the all semi-local LCS problem are represented explicitly, the total size of the output is $\Theta((m+n)^2)$, corresponding to m^2+n^2 possible substrings and $2mn$ possible prefix–suffix pairs. To reduce the storage requirements, we allow the output lengths to be represented implicitly by a smaller data structure that allows efficient retrieval of individual output values. Using previously known linear-size representations of the string comparison dag, retrieval of an individual output length typically requires scanning of at least a constant fraction of the representing data structure, and therefore takes time $O(m+n)$. By exploiting the geometry connection, we show that the output lengths can be rep-

E-mail address: tiskin@dcs.warwick.ac.uk.

1570-8667/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2008.07.001

resented by a set of $m+n$ grid points. Individual output lengths can be obtained from this representation by dominance counting queries. This leads to a data structure of size $O(m+n)$, that allows to query an individual output length in time $O(\frac{\log(m+n)}{\log\log(m+n)})$, using a recent result by JáJá et al. [11]. The described approach presents a substantial improvement in query efficiency over previous approaches.

It has long been known [8,17] that the (global) LCS problem can be solved in subquadratic[1] time $O(\frac{mn}{\log(m+n)})$ when m and n are reasonably close. Alves et al. [2], based on previous work by Schmidt [20], proposed an all string–substring (i.e. restricted semi-local) LCS algorithm that runs in time $O(mn)$. In this paper, we propose the first all semi-local LCS algorithm, which runs in subquadratic time $O(\frac{mn}{\log^{0.5}(m+n)})$ when m and n are reasonably close. This improves on [2] simultaneously in algorithm functionality, output representation efficiency, and running time.

A preliminary version of this paper appeared as [21].

2. Previous work

Although our generic definition of the all semi-local LCS problem is new, several algorithms dealing with similar problems involving multiple substring comparison have been proposed before. The standard dynamic programming approach can be regarded as comparing all prefixes of each string against all prefixes of the other string. Papers [2,7,13–16,20] present several variations on the theme of comparing substrings (prefixes, suffixes) of two strings. In [13,15], the two input strings are revealed character by character. Every new character can be either appended or prepended to the input string. Therefore, the computation is performed essentially on substrings of subsequent inputs. In [16], multiple strings sharing a common substring are compared against a common target string. A common feature in many of these algorithms is the use of linear-sized string comparison dag representation, and a suitable merging procedure that "stitches together" the representations of neighbouring dag blocks to obtain a representation for the blocks' union. As a consequence, such algorithms could be adapted to work with our new, potentially more efficient geometric representation, without any increase in asymptotic time or memory requirements.

3. Semi-local longest common subsequences

We consider strings of characters from a fixed finite alphabet, denoting string concatenation by juxtaposition. Given a string, we distinguish between its contiguous *substrings*, and not necessarily contiguous *subsequences*. Special cases of a substring are *a prefix* and *a suffix* of a string. For two strings $a=\alpha_1\alpha_2\ldots\alpha_m$ and $b=\beta_1\beta_2\ldots\beta_n$ of lengths m, n, respectively, the *longest common subsequence* (*LCS*) *problem* consists in computing the length of the longest string that is a subsequence both of a and b.

We define a generalisation of the LCS problem, which we call the *all semi-local LCS problem*. It consists in computing the LCS lengths on substrings of a and b as follows:

- the *all string–substring LCS problem*: a against every substring of b;
- the *all prefix–suffix LCS problem*: every prefix of a against every suffix of b;
- symmetrically, the *all substring–string LCS problem* and the *all suffix–prefix LCS problem*, defined as above but with the roles of a and b exchanged.

It turns out that by considering this combination of problems rather than each problem separately, the algorithms can be greatly simplified.

A traditional distinction, especially in computational biology, is between global (full string against full string) and local (all substrings against all substrings) comparison. Our problem lies in between, hence the term "semi-local". Many string comparison algorithms output either a single optimal comparison score across all local comparisons, or a number of local comparison scores that are "sufficiently close" to the globally optimal. In contrast with this approach, we require to output all the locally optimal comparison scores.

In addition to standard integer indices $\ldots,-2,-1,0,1,2,\ldots$, we use *odd half-integer*[2] indices $\ldots,-\frac{5}{2},-\frac{3}{2},-\frac{1}{2},\frac{1}{2},\frac{3}{2},\frac{5}{2},\ldots$. For two numbers i, j, we write $i \trianglelefteq j$ if $j-i\in\{0,1\}$, and $i \lhd j$ if $j-i=1$. We denote

$$[i:j]=\{i,i+1,\ldots,j-1,j\},$$
$$\langle i:j\rangle=\left\{i+\tfrac{1}{2},i+\tfrac{3}{2},\ldots,j-\tfrac{3}{2},j-\tfrac{1}{2}\right\}.$$

To denote infinite intervals of integers and odd half-integers, we will use $-\infty$ for i and $+\infty$ for j where appropriate.

We will make extensive use of finite and infinite matrices, with integer elements and integer or odd half-integer indices. A *permutation matrix* is a (0, 1)-matrix containing exactly one nonzero in every row and every column. An *identity matrix*

[1] The term "subquadratic" throughout this paper refers to the case $m=n$.

[2] It would be possible to reformulate all our results using only integers. However, using odd half-integers helps to make the exposition simpler and more elegant.