Computers & Operations Research 39 (2012) 512–520

Contents lists available at ScienceDirect

Computers & Operations Research

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

An improved algorithm for the longest common subsequence problem

Sayyed Rasoul Mousavi*, Farzaneh Tabataba

Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran

ARTICLE INFO

Available online 2 April 2011

Keywords:
Longest common subsequence
LCS
Beam search
Heuristic function
Algorithms
Bioinformatics

ABSTRACT

The Longest Common Subsequence problem seeks a longest subsequence of every member of a given set of strings. It has applications, among others, in data compression, FPGA circuit minimization, and bioinformatics. The problem is NP-hard for more than two input strings, and the existing exact solutions are impractical for large input sizes. Therefore, several approximation and (meta) heuristic algorithms have been proposed which aim at finding good, but not necessarily optimal, solutions to the problem. In this paper, we propose a new algorithm based on the constructive beam search method. We have devised a novel heuristic, inspired by the probability theory, intended for domains where the input strings are assumed to be independent. Special data structures and dynamic programming methods are developed to reduce the time complexity of the algorithm. The proposed algorithm is compared with the state-of-the-art over several standard benchmarks including random and real biological sequences. Extensive experimental results show that the proposed algorithm outperforms the state-of-the-art by giving higher quality solutions with less computation time for most of the experimental cases.

1. Introduction

The Longest Common Subsequence (LCS) problem asks for a longest string that is a subsequence of every member of a given set of strings. A subsequence of a given string is a string that can be obtained by deleting zero or more characters from the given string. Among various applications of this problem are file comparison [1], text editing [2], data compression [3], query optimization in databases [4], clustering Web users [5], and circuit minimization in field programmable gate arrays (FPGAs) [6]. In addition, LCS is used in molecular biology to compare DNA or RNA sequences and to determine homology in macromolecules [2,7–9].

For two input strings, LCS can be efficiently solved to optimality using dynamic programming in $O(l_1.l_2)$, where l_1 and l_2 are the lengths of the input strings. However, the problem is NP-hard for an arbitrary number of strings [10,11]. Various optimal (exact) algorithms have been proposed for this problem. One approach was to use dynamic programming. In [12,13], dynamic programming algorithms were proposed to solve the problem in $O(l^n)$, where n is the number of the input strings and l is the length of the longest one. These algorithms were improved in [14,15] to reduce the complexity to $O(l^{n-1})$, which is still exponential in the number of strings. Further algorithms based on dynamic programming may be found in the survey by Berghot et al. [16]. Another approach to tackle the LCS problem was based on traversing a search tree. Hsu and Du proposed in [17] an enumeration algorithm based on backtracking. The idea was further enhanced by Easton et al., who adopted a selection heuristic and two new types of branch and bound pruning [18]. The resulting algorithm, called *Specialized Branching* (*SB*), was compared with the previous state-of-the-art algorithms with positive results. In contrary to the above-mentioned dynamic programming algorithms, which are exponential in the number of strings, SB is exponential in the length of the longest common subsequence (LLCS). Among other works on LCS is [19] where an integer programming formulation was proposed whose complexity was still $O(l^n)$.

The above-mentioned optimal algorithms for LCS are impractical for large input sizes, hence the use of non-optimal solutions are inevitable. Until 1994, no heuristic method was introduced for the LCS problem [20]. The first non-optimal algorithm was *Long Run* (LR) which was an approximation algorithm with an approximation ratio of $|\sum|$ [20,21]. This algorithm simply constructs a string, as its output, using only a single character in $\sum$, which is not of interest in practice. Another approximation algorithm called *Expansion* was introduced in [22] which provided the same approximation ratio of $|\sum|$, without the single-character restriction of Long Run. The complexity of Expansion was $O(nl^4 \lg l)$, which was further improved in [23] using minimum-spanning-trees. Huang et al. [24] devised two more approximation

* Corresponding author.
E-mail addresses: srm@cc.iut.ac.ir (S.R. Mousavi), f.tabataba@ec.iut.ac.ir (F. Tabataba).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.02.026

algorithms called *Enhanced Long Run* (ELR) and *Best Next for Maximal Available Symbols* (BNMAS), which were of $O(|\sum|nl)$ and $O(|\sum|^2nl+|\sum|^3l)$ complexities, respectively, still with the approximation ratio of $|\sum|$. They showed that their algorithms were quite successful in practice. In [25], the authors showed that BNMAS was considerably faster than Expansion, especially when $|\sum|$ is small and/or n is large and that it outperformed Expansion in most of the test cases.

In addition to the above-mentioned approximation algorithms, heuristic algorithms, which do not normally guarantee an approximation ratio, were also proposed for the LCS problem. The *Best-Next heuristic* was proposed in [26,27] as a simple heuristic algorithm which is run in $O(|\sum|nl)$, and it was shown to be of superior results compared to some of the above-mentioned approximation algorithms, such as LR, for practical datasets. Guenoche and Vitte [28] proposed a linear-time *dynamic programming heuristic* (DPH), which was further modified by Guenoche [29]. Easton and Singireddy [30] introduced, based on the large-neighborhood search paradigm, a new algorithm called *time horizon specialized branching heuristic* (THSB), which was shown to be superior to DPH. More recently, Shyu and Tsai [25] used ant colony optimization (ACO) to solve LCS. They compared their algorithm with expansion and BNMAS algorithms by implementing and testing them over random and biological datasets obtained from NCBI [31]. According to the experimental results, ACO dominates both of the other algorithms in terms of quality and is faster than Expansion. Finally, Blum et al. proposed a constructive Beam Search algorithm, called *BS*, for the LCS problem [32]. In their algorithm, two different greedy functions were used to evaluate and compare candidate solutions. Their BS algorithm is an extension of a predecessor beam search introduced by Blum and Blesa [33]. In order to compare their BS algorithm with previous leading algorithms in the literature, Blum et al. used two types of parameter settings; one called *low time* aimed at producing quick solutions and the other called *high quality* intended for high quality solutions but at extra computation cost. They compared BS with Expansion, Best-Next, G&V, THSB and ACO algorithms over three benchmarks previously introduced in [33,30,25]. Extensive experimental results showed that Blum et al.'s BS algorithm outperforms, on average, its predecessors in terms of both quality and computation time, concluding that it is the current state-of-the-art.

In this paper, we provide an improved beam search algorithm called IBS-LCS for the LCS problem, which, on average, improves over the state-of-the-art, with respect to both quality and computation time. It has been inspired by the Blum et al.'s beam search algorithm but has the following distinguishing characteristics. First, a novel probability-based heuristic function is used as opposed to the heuristic functions used in BS and the other heuristic algorithms in the literature. We believe that our proposed heuristic function performs better than the existing ones in domains where the given strings are expected to be independent. Second, in contrary to the BS algorithm, it does not use upper bounds for pruning the search tree. Third, BS checks, at each level of the search tree, whether each new candidate solution is dominated by an existing candidate solution. To do so, it compares each new candidate solution with every existing one until it is found to be dominated by some of them or compared by all. However, we use a pre specified number of 'best' solutions, at each level of the search tree, as potential dominators for the other candidate solutions. Instead, the time saved by avoiding the extra comparisons and calculation of upper bounds is invested into larger values of beam size. As the consequence of the above-mentioned modifications, IBS-LCS outperforms the state-of-the-art not only with respect to quality but also with respect to run time, for most standard benchmarks. More specifically, IBS-LCS outperforms the state-of-the-art, on average, over 4 out of the 5 benchmark datasets used in [32]. The only benchmark for which IBS-LCS is not suggested is composed of strings which are highly similar.

The rest of the paper is organized as follows. Section 2 provides basic notations and definitions used in the rest of the paper. In Section 3, we present our proposed algorithm. The new heuristic function is developed in Section 4 followed by a brief analysis of the time complexity of the algorithm. Section 5 reports the experimental results, and Section 6 concludes the paper.

2. Basic notations and definitions

Let s be a string of length m. We use s^k, where k is an integer between 1 and m inclusive, to denote the kth character of s. Let s_1 and s_2 be two strings, $A_1=\{i|i\in\mathbf{N},i\le|s_1|\}$, and $A_2=\{i|i\in\mathbf{N},i\le|s_2|\}$, where $\mathbf{N}$ is the set of integers greater than zero. We say that s_1 is a subsequence of s_2, and write $s_1\prec s_2$, if there is an injective function g from A_1 to A_2 such that: (1) $\forall k\in A_1,\ s_1^k=s_2^{g(k)}$ and (2) $\forall k,k'\in A_1,\ k<k'\Rightarrow g(k)<g(k')$. We call such a function g a *map of* s_1 to s_2. Note that such a map is not necessarily unique. We define the cost of a map g of s_1 to s_2 as the integer $g(s_1^k)$, where $k=|s_1|$. By an optimal map of s_1 to s_2, we mean a minimum cost map of s_1 to s_2, if any. The null string, i.e. the string of zero length, is considered to be a subsequence of any string.

Let x be a string and S be a nonempty set of strings. We write $x\prec S$ if $\forall s_i\in S,\ x\prec s_i$. The Longest Common Subsequence (LCS[1]) problem is then defined as to obtain a string x of maximum length such that $x\prec S$. By an input string, we mean a string in S. The alphabet over which the input strings are defined is denoted by $\sum$; we assume $|\sum|>1$. We use n to denote the number of input strings; that is, $n=|S|$. Since LCS can be efficiently solved for $n=2$, we assume $n>2$. We further assume that $S=\{s_1,\ldots,s_n\}$; that is, the input strings are denoted by the small letter s indexed from 1 to n. We use m_i to refer to $|s_i|$ and assume $m_i>0$, $i=1,\ldots,n$. In the case all the input strings are of the same length, we use m to denote their length; otherwise, m denotes $\max\{m_i,\ i=1,\ldots,n\}$. We use (possibly indexed) x to denote a candidate solution. A candidate solution x is called feasible if $x\prec S$; it is otherwise called infeasible. A feasible candidate solution x is optimal if there exists no other feasible solution of a length greater than $|x|$.

Let x be a feasible candidate solution. We use $p_i(x)$ to denote the cost of the optimal map of x to s_i. Then $q_i(x)$ is defined as $m_i-p_i(x)$. By $r_i(x)$, we mean the string obtained by deleting the first $p_i(x)$ characters from s_i (see Fig. 1), and $R(x)$ is defined as the set $\{r_i(x),\ i=1,\ldots,n\}$. By a random string in this paper, we mean a string each of whose characters obtained by uniformly-randomly selecting one of the characters in $\sum$. Finally, we use $Pr(.)$ to denote the statistical probability function. Although there are two types of beam search, namely constructive and perturbative (Local Beam Search [34]), we use beam search in this paper to refer to the former.

3. The proposed algorithm

The beam search algorithm, in its standard form, is a deterministic, yet heuristic, tree search. It is similar to the breath-first search algorithm except that it does not keep all the leaves but only β of them, where $\beta>0$ is called the beam size. It turns to a pure constructive greedy heuristic in the case $\beta=1$; it also turns to the breath-first search if β is large enough to keep all the

[1] It is also referred to as k-LCS, where $k=|S|$, in the literature.