Operations Research Letters 41 (2013) 644–649

Contents lists available at ScienceDirect

Operations Research Letters

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

A hybrid genetic algorithm for the repetition free longest common subsequence problem

Mauro Castelli [a,*], Stefano Beretta [b,c], Leonardo Vanneschi [a,b]

[a] ISEGI, Universidade Nova de Lisboa, 1070-312, Lisboa, Portugal

[b] DISCo, University of Milano Bicocca, 20126 Milano, Italy

[c] Inst. for Biomedical Technologies, National Research Council, 20090 Segrate, Italy

ARTICLE INFO

Article history:
Received 11 June 2013
Received in revised form
5 September 2013
Accepted 5 September 2013
Available online 12 September 2013

Keywords:
Repetition free longest common subsequence
Heuristics
Genetic algorithms
Estimation of distribution algorithms

ABSTRACT

Computing the longest common subsequence of two sequences is one of the most studied algorithmic problems. In this work we focus on a particular variant of the problem, called repetition free longest common subsequence (RF-LCS), which has been proved to be NP-hard. We propose a hybrid genetic algorithm, which combines standard genetic algorithms and estimation of distribution algorithms, to solve this problem. An experimental comparison with some well-known approximation algorithms shows the suitability of the proposed technique.

1. Introduction

One of the most important problem in the field of algorithms on strings is the computation of the *longest common subsequence* (LCS) of two sequences, over a given alphabet. Its importance is mostly due to its wide range of applications, covering many research fields. For example, several applications of different variants of the LCS problem can be found in bioinformatics, which are used to perform analysis on sequences of DNA or RNA. Other interesting applications of variants of the LCS problem can be found in [2]. More specifically, the LCS problem between two given strings S_1 and S_2 asks for finding a longest sequence S which is subsequence of both the input strings S_1 and S_2 (see Section 2 for a formal definition of the problem). Although the LCS problem can be solved in polynomial time (see [14]) using for example a dynamic programming approach (see [7]), its generalization to a set of sequences, which ask for finding the longest sequence that is subsequence of all the input sequences, is NP-hard (see [12]).

Different variants of the longest common subsequence problem have been proposed [4–6,17] to compare biological sequences, where, given two strings S_1 and S_2, the computed longest common subsequence is required to satisfy some constrains. An interesting variant of the LCS problem is the so- called *repetition-free longest common subsequence* (RF-LCS) which, given two sequences S_1 and S_2, asks for finding the longest sequence S that is subsequence of both S_1 and S_2 and such that it contains at most one occurrence of each symbol [1]. This variant is used to model the genome rearrangement with respect to the duplications of some genes. The goal of this study is to infer the (supposed) original sequence of genes, in which every gene has only one occurrence, starting from a set of sequences, each one (possibly) containing multiple copies of some genes [16]. This particular problem has been strongly investigated and the complexity results can be found in [8]. Finally, other recent studies focused on the so-called *doubly-constrained longest common subsequence* (DC-LCS), which generalizes the mathematical formulation of the RF-LCS (and other constrained variants of LCS problems). In particular, some complexity results on this problem can be found in [5], while the work proposed in [3] discusses the parameterized complexity of the RF-LCS problem.

In this paper we define a hybrid *genetic algorithm* (GA) to address the RF-LCS problem. The idea of the proposed technique is to use standard genetic algorithms and estimation of distribution algorithms. In particular a genetic algorithm is used to explore the search space, while an estimation of distribution algorithm provides a simple, efficient and theoretically sound method to guarantee that the constrains of the RF-LCS problem are satisfied.

The paper is organized as follows: Section 2 introduces the basic definitions that will be used in the rest of the paper; Section 3 introduces the genetic algorithms (GAs) describing their

* Corresponding author. Tel.: +351 213828628; fax: +351 213872140.
E-mail addresses: castelli.mauro@gmail.com, mcastelli@isegi.unl.pt (M. Castelli), beretta@disco.unimib.it (S. Beretta), lvanneschi@isegi.unl.pt (L. Vanneschi).

0167-6377/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.orl.2013.09.002

computational model, while Section 4 focuses on the estimation of distribution algorithms (EDAs), presenting their properties and the main differences with GAs. Section 5 presents the hybrid algorithm developed to solve the RF-LCS problem, while Section 6 describes the experimental settings, and discusses the obtained results comparing them with the ones obtained with other approximation algorithms.

2. Basic definitions

In this section we give some basic concepts and notations that will be used in the rest of the paper. Let S be a string over an alphabet Σ of size $|\Sigma|$. Given a string S, we denote by $S[i]$ the symbol occurring at position i in string S. Let also $S[i \ldots j]$ be the substring of S starting at position i and ending at position j. Given two sequences S and S' over a finite alphabet Σ, S' is a *subsequence* of S if S' can be obtained from S by removing some (possibly zero) characters. When S' is a subsequence of S, then S is a *supersequence* of S'. Given two sequences S_1 and S_2, the LCS problem asks for a longest possible sequence S that is a subsequence of both S_1 and S_2. In the rest of the paper we will focus on the following variant of the LCS problem.

Problem 1 (Repetition-free Longest Common Subsequence (RF-LCS)). **Input:** two strings S_1 and S_2 over an alphabet Σ. **Output:** a longest common subsequence S of S_1 and S_2 so that S contains at most one occurrence of each symbol $\sigma \in \Sigma$.

It has been proved in [1] that Problem 1 is APX-hard even when each symbol occurs at most twice in each of the input strings S_1 and S_2.

3. Genetic algorithm

Genetic Algorithms (GAs) [9,10] are a class of computational models that mimic the process of natural evolution. GAs are often viewed as function optimizers although the range of problems to which they have been applied is quite broad. Although different variants of GAs exist, most of the methods called "GAs" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. The chromosomes in a GA population typically take the form of bit strings. Each locus in the chromosome has two possible alleles: 0 and 1. Each chromosome (or individual) can be thought of as a point in the search space of candidate solutions. The GA processes populations of chromosomes, successively replacing one such population with another. GAs most often require a fitness function that assigns a score (fitness) to each chromosome in the current population. The fitness of a chromosome depends on how well that chromosome solves the problem at hand.

Genetic operators are used to create a new population of chromosomes starting from the existing one (population of parents), and the simplest form of the genetic algorithm involves three types of operators: selection, crossover (single point), and mutation. The selection operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce. The crossover operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. The crossover operator roughly mimics biological recombination between two single-chromosome (haploid) organisms. The mutation operator randomly flips some of the bits in a chromosome. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

Given a problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows: (1) start with a randomly generated population of nl-bit chromosomes (candidate solutions to the problem). (2) Calculate the fitness $f(x)$ of each chromosome x in the population. (3) Repeat the following steps until n offspring has been created: (3.1) select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement", meaning that the same chromosome can be selected more than once to become a parent. (3.2) With probability p_c (the "crossover probability"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (3.3) Mutate the two offspring at each locus with probability p_m (the "mutation probability"), and place the resulting chromosomes in the new population. (4) Replace the current population with the new population, and repeat from Step 2 until a certain number of generations have been performed (or other termination criteria are met). Each iteration of this process is called a generation.

This introduction to the basic GA algorithm is taken from [13] and the reader is referred to [13,9] for a complete introduction to GAs.

4. Estimation of distribution algorithm

As observed in [15], from an abstract point of view, the selected set of promising solutions can be viewed as a sample drawn from a probability distribution. Although the true probability distribution is unknown, there are algorithms that are able to estimate that probability distribution by using the selected set of solutions itself and use this estimate to generate new solutions. These algorithms are called *estimation of distribution algorithms* (EDAs) [11]. In EDAs better solutions are selected from an initially randomly generated population of solutions like in the simple GAs. Then, the true probability distribution of the selected set of solutions is estimated and new solutions are generated according to this estimate. The new solutions are then added into the original population, replacing some of the old ones. The process is repeated until the termination criteria are met.

EDAs therefore do the same as GAs except for that they replace genetic crossover and mutation operators by the following two steps: (1) a model (an estimate of the true distribution) of selected promising solutions is constructed and (2) new solutions are generated according to the constructed model. Hence, while in GAs (and other heuristics from evolutionary computation) the interrelations between the different variables representing the individuals are kept in mind implicitly (e.g. building block hypothesis [9]), in EDAs the interrelations are expressed explicitly through the probability distribution associated with the individuals selected at each iteration.

5. Method

In this section, the proposed algorithm used to address the RF-LCS problem is presented. Let S_1 and S_2 be the two sequences over an alphabet Σ that represent the input of the problem. We assume, without the loss of generality, that the two sequences have the same length l. A possible solution for the RF-LCS problem is codified by means of a binary string K that has the same length of the two sequences S_1 and S_2. More in detail, '1' in a particular position i (with $i \leq l$) indicates that the longest common subsequence (without repetition of characters) between S_1 and S_2 includes the ith character of S_1. On the other hand, '0' indicates that the common subsequence should not consider the ith character of S_1. Hence, an individual of the GA population is a string where each bit in position i, with $1 \leq i \leq l$, indicates whether or not the ith character