Knowledge-Based Systems 77 (2015) 103–113

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A two-stage constructive method for the unweighted minimum string cover problem

Manuel Lozano [a], Francisco J. Rodriguez [a,*], Carlos García-Martínez [b]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Granada 18071, Spain
[b] Department of Computing and Numerical Analysis, University of Córdoba, Córdoba 14071, Spain

ARTICLE INFO

Article history:
Received 3 September 2014
Received in revised form 9 January 2015
Accepted 10 January 2015
Available online 17 January 2015

Keywords:
Unweighted minimum string cover problem
Constructive two-stage algorithm
Combinatorial optimization
Identifying parts within sets of DNA sequences
Dictionary generation

ABSTRACT

In this work, we propose a novel constructive method to deal with the unweighted minimum string cover problem. Given a set of strings S, this defiant optimization problem aims to find a minimum set of substrings M from S such that every string in S can be written as a concatenation of the strings in M. This problem has challenging real-world applications, especially in the field of computational biology.

The proposed constructive algorithm is composed of two stages that are executed iteratively. The objective of the first stage is to find frequent substrings in S to be included in M. The aim of the second stage is to simplify the set M to try to get a minimal set. Extensive computational experiments reveal that the proposed algorithm is highly effective for solving complex instances involving up to 100 000 strings in S as compared to the current state-of-the-art method.

1. Introduction

Given a set of strings S, a *cover* M of S is a set of substrings from S so that each string $s \in S$ can be built by concatenating strings in M. The *minimum string cover (MSC)* problem aims to find the cover M with minimal cost, given a cost for each substring in M. In the *unweighted MSC* (uMSC), the cost of each element $m \in M$ is 1, and thus the objective is to minimize the cardinality of the set M.

The uMSC problem was shown to be NP-complete by Neraud [11]. Several real-world applications of the uMSC problem have been described in the literature. In the first place, Bodlaender et al. [2] presented an application to the *computational biology* field to determine the 3-D structure of biopolymers from their sequences, though it appears under the name of *dictionary generation*. Secondly, Canzar et al. [3] considered the application of uMSC to the field of *data storage* to obtain a compact representation of a set of strings.

Moreover, recently Blakes et al. [1] described the application of uMSC as part of the problem that arises when identifying parts within a given set of *DNA sequences*. In this work, they focus on the second stage of this problem that addresses the generation of plans to construct the targets that minimize the concatenations needed. The first stage in which common building blocks are identified is equivalent to the uMSC problem.

Despite the potential applications described above, the studies on the uMSC problem have mostly addressed its theoretical aspects [11,2]. The only practical approach for solving the uMSC problem, as far as we know, was recently presented by Canzar et al. [3]. In this work, the authors propose a *branch and bound* method whose bounds are obtained through a repeated computation of the shortest path problem in a directed acyclic graph associated with the strings in S. This method requires the calculation of all the different decompositions into substrings of each string in S. Each node of the branch and bound tree represents a subset of those substrings that must be included into the solution and a subset of substrings that are forbidden. The experimental study performed shows for the first time that non-trivial uMSC instances can be solved in reasonable time. In particular, the instances used have up to 100 strings in S. Moreover, this experimental comparison includes an study on instances with costs different from 1 for each element in M (*weighted MSC*) and up to 620 strings in S. However, Blakes et al. [1] highlighted that the existing approaches so far cannot effectively scale up to lengths from a few hundred to several thousand base pairs required for most DNA libraries.

Previous works have also studied a variant of the MSC problem known as *l-cover* problem [9], where each substring in S must be produced by the concatenation of at most l substrings, where l is

* Corresponding author.
E-mail addresses: lozano@decsai.ugr.es (M. Lozano), fjrodriguez@decsai.ugr.es (F.J. Rodriguez), cgarcia@uco.es (C. García-Martínez).

http://dx.doi.org/10.1016/j.knosys.2015.01.003
0950-7051/

assumed to be constant. In particular, Ferdous et al. [4] developed several *ant colony optimization* approaches to deal with this problem. The search space is defined over the set of substrings obtained from every possible decomposition of the strings in S, like in the method proposed by Canzar et al. [3].

In this work, we propose a novel two-stage constructive method to tackle the uMSC problem. In the first stage, the proposal generates a cover set where the substrings included (*building blocks*) are generated in a constructive manner, as opposed to the previous methods that are based on decompositions. It starts from the basic symbols of the alphabet and iteratively adds new building blocks to the cover, which are constructed starting from the current ones according to the frequency with which they appear in S. The aim of the second stage is to simplify the best cover set found so far to try to obtain a minimal set.

The remainder of this paper is organized as follows. Section 2 gives a background of the basic notation for the uMSC problem and defines it formally. Section 3 details our two-stage constructive method for the tackled problem. Section 4 describes the empirical studies carried out in order to compare the results of our proposal with those of other approaches from the literature. Finally, Section 5 contains a summary of results and conclusions.

2. Notation and problem definition

In this section, we provide a description of the notation for the problem, based on the previous works by Canzar et al. [3] and Schwiegelshohn [13]. Furthermore, using this notation, we define formally the uMSC problem.

MSC problem *input* consists of a set of strings S over a symbolic alphabet Σ. Every string in S is denoted by s_i for $i \in \{1,\dots,|S|\}$, where $|S|$ is the cardinality of S. String s_i contains $|s_i|$ symbols, and the *position* p in the string s_i is represented by the *pair* (i,p) with $p \in \{1,\dots,|s_i|\}$. Additionally, let $\|S\| = \sum_{s_i \in S} |s_i|$ be the total number of symbols in S.

Example 2.1. Consider the set of strings S = {*a, aab, aba*} where the alphabet Σ={*a,b*}. Then, the cardinality of the set S is 3, $\|S\|$ is 7, s_1=*a*, s_2=*aab*, and s_3=*aba*. The position 2 in the string s_3, for example, occupied by the character *b* is represented by the pair *(3,2)*.

A *triplet* (i,p,q) with $q \in \{1,\dots,|s_i|\}$ and $p \leqslant q$ is an *occurrence* and represents the positions p to q of the string s_i. The *substring* that occurs at positions p to q in the string s_i is denoted by $s_i(p,q)$. The *set of all occurrences* in S is denoted by $C(S)$ and $T(S) \subset \Sigma^*$ is the *set of all substrings* in S, where Σ^* is the set of all strings that can be formed with the alphabet Σ. Note that every string $s_i \in S$ also belongs to $T(S)$ and hence $S \subseteq T(S)$. We use t_j with $j \in \{1,\dots,|T(S)|\}$ to denote a specific substring.

Example 2.2. Following the above example, the occurrence of the substring $s_3(2,3)$ = *ba* that starts at position 2 and ends at position 3 is represented by the triplet $(3,2,3)$. $C(S) = \{(1,1,1),(2,1,1),(2,1,2),(2,1,3),(2,2,2),(2,2,3),(2,3,3),(3,1,1),(3,1,2),(3,1,3),(3,2,2),(3,2,3),(3,3,3)\}$ and $T(S) = \{a, aa, aab, ab, b, aba, ba\}$. It is important to highlight the difference between $C(S)$ and $T(S)$. For example, let us consider the substring *ab* from $T(S)$ that has two occurrences in S. These occurrences appear in the strings s_2 and s_3 and are represented by two different triplets in $C(S)$ ($(2,2,3)$ and $(3,1,2)$, respectively).

Furthermore, we assign to every occurrence (i,p,q) the corresponding substring $s_i(p,q)$ and, therefore, we can define the set $C_j = \{(i,p,q) \in C(S) | s_i(p,q) = t_j\}$ that contains all the occurrences that are instances of the substring t_j.

Example 2.3. As we saw before, for the substring t_3=*ab*, the set $C_3 = \{(2,2,3),(3,1,2)\}$.

A set of occurrences $D_i \subset C(S)$ is a *factorization* of the string $s_i \in S$ if we can order the elements of $D_i = \{(i,p_0,q_0),(i,p_1,q_1),\dots,(i,p_n,q_n)\}$ such that $p_0 = 1$, $q_n = |s_i|$ and $p_{k+1} = q_k + 1$ for $k \in \{0,\dots,n-1\}$.

Example 2.4. The set D_2 conformed by the occurrences (2,1,1) and (2,2,3) is a factorization of the string s_2. This fact can be easily seen as the substring that appears in the occurrence (2,1,1) is *a* and that in (2,2,3) is *ab*, resulting their concatenation in the string s_2.

Then, $M \subseteq T(S)$ is a *cover* of S (or M covers S) if and only if the set of occurrences $D = \{(i,p,q) \in C(S) | s_i(p,q) \in M\}$ contains a factorization for every $s \in S$.

Example 2.5. M = {*a, ab*} is a cover of S since $D_1 = \{(1,1,1) = a\}$ is a factorization of s_1, $D_2 = \{(2,1,1) = a, (2,2,3) = ab\}$ of s_2, and $D_3 = \{(3,1,2) = ab, (3,3,3) = a\}$ of s_3.

Given a weight function $w : T(S) \to R_0^+$, the *minimum string cover* (MSC) consists of finding a set of strings $M \subseteq T(S)$ such that the two following conditions are met: (1) M covers S and (2) $\sum_{t \in M} w(t)$ is minimal. The most basic and also classic version of the problem uses unit weights for all substrings, i.e. $w(t) = 1$ for all $t \in T(S)$. In this case, the minimum string cover is a cover of S with minimal cardinality and the problem is called unweighted MSC (uMSC) problem.

3. Two-stage constructive method for the uMSC problem

As previously mentioned, we propose a *two-stage constructive* method (*DISFY* that stands for *DIScover and simpliFY*) to deal with the uMSC problem. The objective of the first stage is to find frequent substrings (basic *building blocks*) in the strings forming the set S. This discovering process is done constructively by starting with a cover set M that contains the elements of the alphabet Σ, which trivially can construct any string in S. From the current elements of M, new building blocks are built by performing concatenation operations. Those building blocks that appear more frequently in S are chosen to form part of the current cover M. In order to measure the frequency of appearance of a given substring m, DISFY uses the cardinality of the set of occurrences of that substring. In the second stage, a simplifying process of the best cover set found is done in order to minimize its cardinality. After the second stage, the algorithm continues again with the first stage, starting with the best cover set found so far.

The rest of the section is organized as follows. In Section 3.1, we provide an overview of the overall DISFY algorithm and, in Sections 3.2 and 3.3, we describe, in detail, the operation of the two stages that conform it. Finally, in Section 3.4, we illustrate the operation of the algorithm through the execution on a concrete example.

3.1. Overall algorithm

Fig. 1 shows the pseudocode of the main procedure of the proposed algorithm. DISFY maintains a current solution $M = \{m_1,\dots,m_c\}$ that is a cover of S, which is initialized to Σ at the beginning of the execution. In addition, our algorithm forms, for each substring $m_k \in M$, a specific set of occurrences F_k,

$$F_k = \{(i,p,q) \in C(S) | s_i(p,q) = m_k\},$$

which instantiate m_k in the strings of S. We should point out that the F_k sets allow a particular factorization for each $s \in S$ to be obtained. Therefore, F_k does not need to contain all the occurrences of m_k in S (C_k) but only those necessary for that particular factoriza-