A space efficient solution to the frequent string mining problem for many databases

Adrian Kügel · Enno Ohlebusch

Received: 20 June 2008 / Accepted: 23 June 2008 / Published online: 9 July 2008
Springer Science+Business Media, LLC 2008

Abstract The frequent string mining problem is to find all substrings of a collection of string databases which satisfy database specific minimum and maximum frequency constraints. Our contribution improves the existing linear-time algorithm for this problem in such a way that the peak memory consumption is a constant factor of the size of the largest database of strings. We show how the results for each database can be stored implicitly in space proportional to the size of the database, making it possible to traverse the results in lexicographical order. Furthermore, we present a linear-time algorithm which calculates the intersection of the results of different databases. This algorithm is based on an algorithm to merge two suffix arrays, and our modification allows us to also calculate the LCP table of the resulting suffix array during the merging.

Keywords String mining · Enhanced suffix array

1 Introduction

In string mining problems, one is given \( m \) databases \( \mathcal{D}_1, \ldots, \mathcal{D}_m \) of strings and searches for the (unknown) strings that fulfill certain constraints, which are usually specified by the user. Here, we focus on the frequent string mining problem. In this problem, the constraints consist of \( m \) pairs of frequency thresholds
A space efficient solution to the frequent string mining problem

We would like to give a medical example as a motivation to study the problem. Suppose a genetic disease, e.g. *Huntington's disease*, is suspected of being caused by a defect on a certain locus of a certain chromosome, say on the short arm of chromosome 4. To find the cause of the disease, a possible approach would be to sequence that segment of the DNA molecules of many healthy individuals and ill persons. Then one database contains the DNA sequences of the healthy individuals, while the second database contains the DNA sequences of the ill individuals. Now, one searches for all strings (contiguous DNA subsequences) that occur frequently (or always) in one of the databases and not too often (or never) in the other database. If one finds, for example, that the string CAGCAGCAG...CAG, in which the codon CAG (coding for the amino acid glutamine) is tandemly repeated more than 36 times, occurs frequently (or always) in the database of ill persons but not too often (or never) in the database of healthy persons, then this gives a hypothesis for the cause of the disease.

Fischer et al. (2006) presented an algorithm that solves the frequent string mining problem in optimal time, that is, in time linear in the size of the input (the databases) and the output (the strings that satisfy the constraints). Although their algorithm is a breakthrough in string mining, it still has certain disadvantages. First, their method is based on the construction of the suffix array of a very long string, namely the concatenation of all strings in all databases (the strings are separated by special separator symbols). Thus, the space consumption of their algorithm is proportional to the space occupied by all databases. It turned out that this space consumption is the bottleneck of their algorithm (Fischer 2007, personal communication). Second, the method is not very flexible if one is interested in different combinations of databases. For example, if just one database is replaced with another database, then the whole procedure has to be restarted again: One must construct the suffix array of the concatenation of all strings in all databases, etc. Third, their result is based on the assumption that the number of databases is treated as a constant.

In this paper, we presented an algorithm without these disadvantages. First, its peak memory consumption is merely proportional to the size of the largest database. Second, it is flexible in the sense that one of several databases can be replaced without having to recalculate everything, that is, intermediate data can be stored on file and be reused. Third, our algorithm has optimal worst case running time, regardless of the number of databases.

2 Preliminaries

We will consider strings $\phi = \phi_1 \ldots \phi_n$ consisting of symbols $\phi_i$ from an ordered alphabet $\Sigma$ of constant size. The length of a string $\phi$ is the number of symbols it contains, and is denoted by $|\phi|$. A substring ranging from position $n$ to $m$ will be written as $\phi_{n..m}$. A substring $\phi_{n..}|\phi|$ is also called a suffix of $\phi$, and a substring $\phi_{1..m}$ is also called a prefix of $\phi$. 