Theoretical Computer Science 550 (2014) 51–58

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Compact q-gram profiling of compressed strings

Philip Bille, Patrick Hagge Cording *, Inge Li Gørtz

ARTICLE INFO

Article history:
Received 8 July 2013
Received in revised form 6 June 2014
Accepted 9 July 2014
Available online 17 July 2014
Communicated by G.F. Italiano

Keywords:
q-gram profile
Grammar compression
SLP

ABSTRACT

We consider the problem of computing the q-gram profile of a string T of size N compressed by a context-free grammar with n production rules. We present an algorithm that runs in $O(N-\alpha)$ expected time and uses $O(n+q+k_{T,q})$ space, where $N-\alpha \leq qn$ is the exact number of characters decompressed by the algorithm and $k_{T,q} \leq N-\alpha$ is the number of distinct q-grams in T. This simultaneously matches the current best known time bound and improves the best known space bound. Our space bound is asymptotically optimal in the sense that any algorithm storing the grammar and the q-gram profile must use $\Omega(n+q+k_{T,q})$ space. To achieve this we introduce the q-gram graph that space-efficiently captures the structure of a string with respect to its q-grams, and show how to construct it from a grammar.

1. Introduction

Given a string T, the q-gram profile of T is a data structure that can answer substring frequency queries for substrings of length q (q-grams) in $O(q)$ time. We study the problem of computing the q-gram profile from a string T of size N compressed by a context-free grammar with n production rules. We assume that the model of computation is the standard w-bit word RAM where each word is capable of storing a character of T, i.e., the characters of T are drawn from an alphabet $\{1,\ldots,2^w\}$, and hence $w \geq \log N$ [8]. The space complexities are measured by the number of words used.

The generalization of string algorithms to grammar-based compressed text is currently an active area of research. Grammar-based compression is studied because it offers a simple and strict setting and is capable of modelling many commonly used compression schemes, such as those in the Lempel–Ziv family [21,22], with little expansion [2,15]. The problem of computing the q-gram profile has its applications in bioinformatics, data mining, and machine learning [4,12,14]. All are fields where handling large amounts of data effectively is crucial. Also, the q-gram distance can be computed from the q-gram profiles of two strings and used for filtering in string matching [1,9,17–20].

Recently the first dedicated solution to computing the q-gram profile from a grammar-based compressed string was proposed by Goto et al. [7]. Their algorithm runs in $O(qn)$ expected time[1] and uses $O(qn)$ space. This was later improved by the same authors [6] to an algorithm that takes $O(N-\alpha)$ expected time and uses $O(N-\alpha)$ space, where N is the size of the uncompressed string, and α is a parameter depending on how well T is compressed with respect to its q-grams. $N-\alpha \leq \min(qn, N)$ is in fact the exact number of characters decompressed by the algorithm in order to compute the q-gram profile, meaning that the latter algorithm excels in avoiding decompressing the same character more than once.

We present a Las Vegas-type randomized algorithm that gives Theorem 1.

* Corresponding author.
E-mail addresses: phbi@dtu.dk (P. Bille), phaco@dtu.dk (P.H. Cording), inge@dtu.dk (I.L. Gørtz).

[1] The bound in [7] is stated as worst-case since they assume alphabets of size $O(N^c)$ for fast suffix sorting, where c is a constant. We make no such assumptions and without it hashing can be used to obtain the same bound in expectation.

http://dx.doi.org/10.1016/j.tcs.2014.07.010
0304-3975/

Theorem 1. *Let T be a string of size N compressed by a grammar of size n. The q-gram profile can be computed in $O(N-\alpha)$ expected time and $O(n+q+k_{T,q})$ space, where $k_{T,q} \leq N-\alpha$ is the number of distinct q-grams in T.*

Hence, our algorithm simultaneously matches the current best known time bound and improves the best known space bound. Our space bound is asymptotically optimal in the sense that any algorithm storing the grammar and the q-gram profile must use $\Omega(n+q+k_{T,q})$ space.

A straightforward approach to computing the q-gram profile is to first decompress the string and then use an algorithm for computing the profile from a string. For instance, we could construct a compact trie of the q-grams using an algorithm similar to a suffix tree construction algorithm as mentioned in [10], or use Rabin–Karp fingerprints to obtain a randomized algorithm [20]. However, both approaches are impractical because the time and space usage associated with a complete decompression of T is linear in its size $N = O(2^n)$. To achieve our bounds we introduce the q-gram graph, a data structure that space efficiently captures the structure of a string in terms of its q-grams, and show how to compute the graph from a grammar. We then transform the graph to a suffix tree containing the q-grams of T. Because our algorithm uses randomization to construct the q-gram graph, the answer to a query may be incorrect. However, as a final step of our algorithm, we show how to use the suffix tree to verify that the fingerprint function is collision free and thereby obtain Theorem 1.

2. Preliminaries and notation

2.1. Strings and suffix trees

Let T be a string of length $|T|$ consisting of characters from the alphabet Σ. We use $T[i:j]$, $0 \leq i \leq j < |T|$, to denote the substring starting in position i of T and ending in position j of T. We define $socc(s,T)$ to be the number of occurrences of the string s in T.

The suffix tree of T is a compact trie containing all suffixes of T. That is, it is a trie containing the strings $T[i:|T|-1]$ for $i = 0..|T|-1$. The suffix tree of T can be constructed in $O(|T|)$ time and uses $O(|T|)$ space [3]. The generalized suffix tree is the suffix tree for a set of strings. It can be constructed using time and space linear in the sum of the lengths of the strings in the set. The set of strings may be compactly represented as a common suffix tree (CS-tree). The CS-tree has the characters of the strings on its edges, and the strings start in the leaves and end in the root. If two strings have some suffix in common, the suffixes are merged to one path. In other words, the CS-tree is a trie of the reversed strings, and is not to be confused with the suffix tree. For CS-trees, the following is known.

Lemma 1. *(See Shibuya [16].) Given a set of strings represented by a CS-tree of size n and comprised of characters from an alphabet of size $O(n^c)$, where c is a constant, the generalized suffix tree of the set of strings can be constructed in $O(n)$ time using $O(n)$ space.*

For a node v in a suffix tree, the string depth $sd(v)$ is the sum of the lengths of the labels on the edges from the root to v. We use $parent(v)$ to get the parent of v, and $nca(v,u)$ is the nearest common ancestor of the nodes v and u.

2.2. Straight Line Programs

A Straight Line Program (SLP) is a context-free grammar in Chomsky normal form that derives a single string T of length N over the alphabet Σ. In other words, an SLP $\mathcal{S}$ is a set of n production rules of the form $X_i = X_l X_r$ or $X_i = a$, where a is a character from the alphabet Σ, and each rule is reachable from the start symbol X_n. Our algorithm assumes without loss of generality that the compressed string given as input is compressed by an SLP.

It is convenient to view an SLP as a directed acyclic graph (DAG) in which each node represents a production rule. Consequently, nodes in the DAG have exactly two outgoing edges. An example of an SLP is seen in Fig. 2(a). When a string is decompressed we get a derivation tree which corresponds to the depth-first traversal of the DAG.

We denote by t_{X_i} the string derived from production rule X_i, so $T = t_{X_n}$. For convenience we say that $|X_i|$ is the length of the string derived from X_i, and these values can be computed in linear time in a bottom–up fashion using the following recursion. For each $X_i = X_l X_r$ in $\mathcal{S}$,

$$|X_i| = \begin{cases} |X_l| + |X_r| & \text{if } X_i \text{ is a nonterminal,} \\ 1 & \text{otherwise.} \end{cases}$$

Finally, we denote by $occ(X_i)$ the number of times the production rule X_i occurs in the derivation tree. We can compute the occurrences using the following linear time and space algorithm due to Goto et al. [7]. Set $occ(X_n) = 1$ and $occ(X_i) = 0$ for $i = 1..n-1$. For each production rule of the form $X_i = X_l X_r$, in decreasing order of i, we set $occ(X_l) = occ(X_l) + occ(X_i)$ and similarly for $occ(X_r)$.