Theoretical Computer Science 616 (2016) 141–150

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A *really* simple approximation of smallest grammar

Artur Jeż

Institute of Computer Science, University of Wrocław, 50-383 Wrocław, Poland

ARTICLE INFO

Article history:
Received 27 August 2015
Received in revised form 17 December 2015
Accepted 21 December 2015
Available online 30 December 2015
Communicated by M. Crochemore

Keywords:
Grammar-based compression
Construction of the smallest grammar
SLP
Compression
LZ77

ABSTRACT

In this paper we present a *really* simple linear-time algorithm constructing a context-free grammar of size $4g\log_{3/2}(N/g)$ for the input string, where N is the size of the input string and g the size of the optimal grammar generating this string. The algorithm works for arbitrary size alphabets, but the running time is linear assuming that the alphabet Σ of the input string can be identified with numbers from $\{1,\ldots,N^c\}$ for some constant c. Algorithms with such an approximation guarantee and running time are known, however all of them were non-trivial and their analyses were involved. The here presented algorithm computes the LZ77 factorisation and transforms it in phases to a grammar. In each phase it maintains an LZ77-like factorisation of the word with at most ℓ factors as well as additional $\mathcal{O}(\ell)$ letters, where ℓ was the size of the original LZ77 factorisation. In one phase in a greedy way (by a left-to-right sweep and a help of the factorisation) we choose a set of pairs of consecutive letters to be replaced with new symbols, i.e. nonterminals of the constructed grammar. We choose at least 2/3 of the letters in the word and there are $\mathcal{O}(\ell)$ many different pairs among them. Hence there are $\mathcal{O}(\log N)$ phases, each of them introduces $\mathcal{O}(\ell)$ nonterminals to a grammar. A more precise analysis yields a bound $\ell + 4\ell\log(N/\ell)$. As $\ell \leq g$, this yields the desired bound $g + 4g\log(N/g)$.

1. Introduction

1.1. Grammar based compression

In the grammar-based compression text is represented by a context-free grammar (CFG) generating exactly one string. Such an approach was first considered by Rubin [23], though he did not mention CFGs explicitly. In general, the idea behind this approach is that a CFG can compactly represent the structure of the text, even if this structure is not apparent. Furthermore, the natural hierarchical definition of the context-free grammars makes such a representation suitable for algorithms, in which case the string operations can be performed on the compressed representation, without the need of the explicit decompression [6,11,15,22,7,2].

The grammar-based compression was introduced with practical purposes in mind and the paradigm was used in several implementations [16,20]: intuitively, in many cases large data have relatively simple inductive definition, which results in a grammar representation of small size. On the other hand, it turned out that grammar compression is useful in more theoretical considerations: for instance, unveiling the repetitive structure of the text can be used to estimate text similarity [17,4,19]. Another group of applications was made possible due to the aforementioned algorithms that operate directly on the compressed representations: In this approach "regular" data is compressed and then processed in this compressed form,

E-mail address: aje@cs.uni.wroc.pl.

http://dx.doi.org/10.1016/j.tcs.2015.12.032
0304-3975/

often yielding more efficient algorithms. A recent survey by Lohrey [18] gives a comprehensive description of several areas of theoretical computer science where such an approach was successfully applied, such as word equations, computations in groups, computational topology and others.

The main drawback of the grammar-based compression is that producing the smallest CFG for a text is *intractable*: given a string w and number k it is NP-hard to decide whether there exists a CFG of size k that generates w [26]. Furthermore, the size of the smallest grammar for the input string cannot be approximated within factor 8569/8568 [2], assuming NP $\neq$ P.

1.2. Previous approximation algorithms

The first two algorithms with an approximation ratio $\mathcal{O}(\log(N/g))$ were developed simultaneously by Rytter [24] and Charikar et al. [2]. They followed a similar approach, we first present Rytter's one as it is a bit easier to explain.

Rytter's algorithm [24] applies the LZ77 compression to the input string and then transforms the obtained LZ77 representation to an $\mathcal{O}(\ell \log(N/\ell))$ size grammar, where ℓ is the size of the LZ77 representation. It is easy to show that $\ell \leq g$ and as $f(x) = x\log(N/x)$ is increasing, the bound $\mathcal{O}(g\log(N/g))$ on the size of the grammar follows (and so a bound $\mathcal{O}(\log(N/g))$ on the approximation ratio). The crucial part of the construction is the requirement that the derivation tree of the intermediate constructed grammar satisfies the AVL condition. While enforcing this requirement is in fact easier than in the case of the AVL search trees (as the internal nodes do not store any data), it remains involved. Note that the final grammar for the input text is also AVL-balanced, which makes it suitable for later processing.

Charikar et al. [2] followed a similar path, with a different condition imposed on the grammar: it is required that the derivation tree is length-balanced, i.e. for a rule $X \to YZ$ the lengths of words generated by Y and Z are within a constant factor from each other. For such trees efficient implementation of merging, splitting and other operations were given (i.e. constructed from scratch) by the authors and so the same running time as in the case of the AVL grammars was obtained. Since all the operations are defined from scratch, the obtained algorithm is also quite involved and the analysis is even more non-trivial.

Sakamoto [25] proposed a different algorithm, based on RePair [16], which is one of the practically implemented and used algorithms for grammar-based compression. His algorithm iteratively replaces pairs of different letters and maximal repetitions of letters (a^ℓ is a *maximal repetition* if it cannot be extended by a to either side). A special pairing of the letters was devised, so that it is 'synchronising': if u has 2 disjoint occurrences in w, then those two occurrences can be represented as $u_1u'u_2$, where $|u_1|, |u_2| = \mathcal{O}(1)$, such that both occurrences of u' in w are paired and replaced in the same way. The analysis was based on considering the LZ77 representation of the text and proving that due to 'synchronisation' the factors of LZ77 are compressed very similarly as the text to which they refer. Constructing such a pairing is involved and the analysis non-trivial.

Recently, the author proposed another algorithm [10]. Similarly to the Sakamoto's algorithm it iteratively applied two local replacement rules (replacing pairs of different letters and replacing maximal repetitions of letters). Though the choice of pairs to be replaced was simpler, still the construction was involved. The main feature of the algorithm was its analysis based on the recompression technique, which allowed avoiding the connection of SLPs and LZ77 compression. This made it possible to generalise this approach also to grammars generating trees [12]. On the downside, the analysis is quite complex.

1.3. Contribution of this paper

We present a very simple algorithm together with a straightforward and natural analysis. It chooses the pairs to be replaced in the word during a left-to-right sweep and additionally using the information given by an LZ77 factorisation. We require that any pair that is chosen to be replaced is either inside a factor of length at least 2 or consists of two factors of length 1 and that the factor of length at least 2 is paired in the same way as its definition. To this end we modify the LZ77 factorisation during the sweep. After the choice, the pairs are replaced and the new word inherits the factorisation from the original word. This procedure is repeated until a word of length 1 is obtained. This is indeed a grammar construction: when the pair ab is replaced by c we create a rule $c \to ab$. The approximation ratio of our algorithm is at most $1 + 4\log_{3/2}(N/g)$.

Note on computational model The presented algorithm runs in linear time, assuming that we can compute the LZ77 factorisation in linear time. This can be done assuming that the letters of the input words can be sorted in linear time, which follows from a standard assumption that Σ can be identified with a continuous subset of natural numbers of size $\mathcal{O}(N^c)$ for some constant c and the RadixSort can be performed on it. Note that such an assumption is needed for all currently known linear-time algorithms that attain the $\mathcal{O}(\log(N/g))$ approximation guarantee.

2. Notions

Sizes By N we denote the size of the input word, by ℓ, g the sizes of the LZ77 factorisation and smallest grammar for the input string, both notions are defined in detail below.

LZ77-like factorisation An LZ77-like factorisation (called simply *factorisation* in the rest of the paper) of a word w is a representation $w = f_1 f_2 \cdots f_\ell$, where each f_i is either a single letter (called *free letter* in the following) or $f_i = w[j..j + |f_i| - 1]$