Information and Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Approximation of smallest linear tree grammar

Artur Jeż [a,*,1], Markus Lohrey [b,1]

[a] Institute of Computer Science, University of Wrocław, ul. Joliot-Curie 15, PL50383 Wrocław, Poland
[b] University of Siegen, Department of Electrical Engineering and Computer Science, DE57068 Siegen, Germany

ARTICLE INFO

Article history:
Received 24 August 2014
Available online xxxx

Keywords:
Grammar-based compression
Tree compression
Tree grammars

ABSTRACT

A simple linear-time algorithm for constructing a linear context-free tree grammar of size $\mathcal{O}(rg + rg\log(n/rg))$ for a given input tree T of size n is presented, where g is the size of a minimal linear context-free tree grammar for T, and r is the maximal rank of symbols in T (which is a constant in many applications). This is the first example of a grammar-based tree compression algorithm with a good, i.e. logarithmic in terms of the size of the input tree, approximation ratio. The analysis of the algorithm uses an extension of the recompression technique from strings to trees.

1. Introduction

Grammar-based compression has emerged to an active field in string compression during the last decade. The idea is to represent a given string s by a small context-free grammar that generates only s; such a grammar is also called a *straight-line program*, briefly SLP. For instance, the word $(ab)^{1024}$ can be represented by the SLP with the productions $A_0 \to ab$ and $A_i \to A_{i-1}A_{i-1}$ for $1 \le i \le 10$ (A_{10} is the start symbol). The size of this grammar is much smaller than the size (length) of the string $(ab)^{1024}$. In general, an SLP of size n (the size of an SLP is usually defined as the total length of all right-hand sides of productions) can produce a string of length $2^{\Omega(n)}$. Hence, an SLP can be seen as the succinct representation of the generated word. The principle task of grammar-based string compression is to construct, from a given input string s, a small SLP that generates s. Unfortunately, finding a minimal (with respect to size) SLP for a given input string is not achievable in polynomial time, unless **P** = **NP** [1] (recently the same result was shown also in case of a constant-size alphabet [2]). Therefore, one can concentrate either on heuristic grammar-based compressors [3–5], or compressors whose output SLP is guaranteed to be not much larger than a size-minimal SLP for the input string [6–10]. In this paper we are interested mostly in the latter approach. Formally, in [6] the approximation ratio for a grammar-based compressor $\mathcal{G}$ is defined as the function $\alpha_{\mathcal{G}}$ with

$$\alpha_{\mathcal{G}}(n) = \max \frac{\text{size of the SLP produced by } \mathcal{G} \text{ with input } x}{\text{size of a minimal SLP for } x},$$

where the maximum is taken over all strings of length n (over an arbitrary alphabet). The above statement means that unless **P** = **NP** there is no polynomial time grammar-based compressor with the approximation ratio 1. Using approximation

* Corresponding author.
E-mail addresses: aje@cs.uni.wroc.pl (A. Jeż), lohrey@eti.uni-siegen.de (M. Lohrey).

[1] The first author was supported by the National Science Centre, Poland project number 2014/15/B/ST6/00615. The second author was supported by the German Research Foundation, project number LO 748/10-1.

http://dx.doi.org/10.1016/j.ic.2016.09.007
0890-5401/

lower bounds for computing vertex covers, it is shown in [6] that unless **P** = **NP** there is no polynomial time grammar-based compressor, whose approximation ratio is less than the constant 8569/8568.

Apart from this complexity theoretic bound, the authors of [6] prove lower and upper bounds on the approximation ratios of well-known grammar-based string compressors (LZ78, BISECTION, SEQUENTIAL, RePair, etc.). The currently best known approximation ratio of a polynomial time grammar-based string compressor is of the form $\mathcal{O}(\log(n/g))$, where g is the size of a smallest SLP for the input string. Actually, there are several compressors achieving this approximation ratio [6–10] and each of them works in linear time (a property that a reasonable compressor should have).

At this point, the reader might ask, what makes grammar-based compression so attractive. There are actually several arguments in favour of grammar-based compression:

- The output of a grammar-based compressor is a clean and simple object, which may simplify the analysis of a compressor or the analysis of algorithms that work on compressed data; see [11] for a survey.
- There are grammar-based compressors which achieve very good compression ratios. For example RePair [4] performs very well in practice and was for instance used for the compression of web graphs [12].
- The idea of grammar-based string compression can be generalised to other data types as long as suitable grammar formalisms are known for them. See for instance the recent work on grammar-based graph compression [13].

The last point is the most important one for this work. In[14], grammar-based compression was generalised from strings to trees.[2] For this, context-free tree grammars were used. Context free tree grammars that produce only a single tree are also known as straight-line context-free tree grammars (SLCF tree grammars). Several papers deal with algorithmic problems on trees that are succinctly represented by SLCF tree grammars [15–20]. In [21], RePair was generalised from strings to trees, and the resulting algorithm TreeRePair achieves excellent results on real XML trees. Other grammar-based tree compressors were developed in [22,23], but none of these compressors has a good approximation ratio. For instance, in [21] a series of trees is constructed, where the n-th tree t_n has size $\Theta(n)$, there exists an SLCF tree grammar for t_n of size $\mathcal{O}(\log n)$, but the grammar produced by TreeRePair for t_n has size $\Omega(n)$ (and similar examples can be constructed for the compressors in [22,14]).

In this paper, we give the first example of a grammar-based tree compressor TtoG (for "tree to grammar") with an approximation ratio of $\mathcal{O}(\log(n/g))$ assuming the maximal rank r of symbols is bounded and where g denotes the size of the smallest grammar generating the given tree; otherwise the approximation ratio becomes $\mathcal{O}(r + r\log(n/gr))$. Our algorithm TtoG is based on the work [7] of the first author, where another grammar-based string compressor with an approximation ratio of $\mathcal{O}(\log(n/g))$ is presented (here g denotes the size of the smallest grammar for the input string). The remarkable fact about this latter compressor is that in contrast to [6,8–10] it does not use the LZ77 factorization of a string (which makes the compressors from [6,8–10] not suitable for a generalization to trees, since LZ77 ignores the tree structure and no good analogue of LZ77 for trees is known), but is based on the *recompression technique*. This technique was introduced in [24] and successfully applied for a variety of algorithmic problems for SLP-compressed strings [24,25] and word equations [26–28]. The basic idea is to compress a string using two operations:

- block compressions: replace every maximal substring of the form a^ℓ for a letter a by a new symbol a_ℓ;
- pair compression: for a given partition $\Sigma_\ell \uplus \Sigma_r$ replace every substring $ab \in \Sigma_\ell\Sigma_r$ by a new symbol c.

It can be shown that the composition of block compression followed by pair compression (for a suitably chosen partition of the input letters) reduces the length of the string by a constant factor. Hence, the iteration of block compression followed by pair compression yields a string of length one after a logarithmic number of phases. By reversing a single compression step, one obtains a grammar rule for the introduced letter and thus reversing all such steps yields an SLP for the initial string. The term "recompression" refers to the fact, that for a given SLP $\mathbb{G}$, block compression and pair compression can be simulated on $\mathbb{G}$. More precisely, one can compute from $\mathbb{G}$ a new SLP $\mathbb{G}'$, which is not much larger than $\mathbb{G}$ such that $\mathbb{G}'$ produces the result of block compression (respectively, pair compression) applied to the string produced by $\mathbb{G}$. In [7], the recompression technique is used to bound the approximation ratio of the above compression algorithm based on block and pair compression.

In this work we generalise the recompression technique from strings to trees. The operations of block compression and pair compression can be directly applied to chains of unary nodes (nodes having only a single child) in a tree. But clearly, these two operations alone cannot reduce the size of the initial tree by a constant factor. Hence we need a third compression operation that we call *leaf compression*. It merges all children of a node that are leaves into the node. The new label of the node determines the old label, the sequence of labels of the children that are leaves, and their positions in the sequence of all children of the node. Then, one can show that a single phase, consisting of block compression (that we call chain compression), followed by pair compression (that we call unary pair compression), followed by leaf compression reduces the size of the initial tree by a constant factor. As for strings, we obtain an SLCF tree grammar for the input

[2] A tree in this paper is always a rooted ordered tree over a ranked alphabet, i.e., every node is labelled with a symbol and the rank of this symbol is equal to the number of children of the node.

Please cite this article in press as: A. Jeż, M. Lohrey, Approximation of smallest linear tree grammar, Inf. Comput. (2016), http://dx.doi.org/10.1016/j.ic.2016.09.007