Approximation of Grammar-Based Compression via Recompression*

Artur Jeż\textsuperscript{1,2,**}

\textsuperscript{1} Max Planck Institute für Informatik, Campus E1 4, DE-66123 Saarbrücken, Germany
\textsuperscript{2} Institute of Computer Science, University of Wrocław
ul. Joliot-Curie 15, 50-383 Wrocław, Poland
aje@cs.uni.wroc.pl

Abstract. We present a simple linear-time algorithm constructing a context-free grammar of size $O(g \log(N/g))$ for the input string of size $N$, where $g$ 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 $\Sigma$ of the input string is a subset of $\{1, \ldots, N^c\}$ for some constant $c$. Algorithms with such an approximation guarantees and running time are known, the novelty of this paper is the particular simplicity of the algorithm as well as the analysis of the algorithm, which uses a general technique of recompression recently introduced by the author. Furthermore, contrary to the previous results, this work does not use the LZ representation of the input string in the construction, nor in the analysis.

Keywords: Grammar-based compression, Construction of the smallest grammar, SLP.

1 Introduction

Grammar Based Compression. This paper presents an alternative linear-time approximation algorithm for the construction of the smallest grammar (CFG) generating a given string $T$. There are three known algorithms with an approximation ratio $O(\log(N/g))$, where $N$ is the input-string length and $g$ is the size of the optimal grammar \cite{14,15}. The novelty of the proposed algorithm is its apparent simplicity (it uses only local replacement of strings) and an analysis that uses the recompression technique developed recently by the author. In particular, neither the algorithm, nor its analysis relate to the LZ-compression, which was the case for previously known algorithms.

In the grammar-based compression text is represented by a context-free grammar generating exactly one string. 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 CFGs make such

* The full version of this paper is available at \texttt{http://arxiv.org/abs/1301.5842}

** Supported by NCN grant number 2011/01/D/ST6/07164, 2011–2014
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 \[3,9,13,14,11\]. Lastly, there is a close connection between block-based compression methods and the grammar compression. To be more specific, it fairly easy to rewrite the LZW definition as a context free grammar (with just a multiplicative constant factor overhead), LZ77 can also be presented in this way, but this is much less obvious (and introduces a \(\log(N/\ell)\) blow-up, where \(\ell\) is the size of the LZ77 representation) \[14,1\].

The main drawback of the grammar-based compression is that producing the smallest CFG for a text is difficult: the decision problem is NP-hard \[16\] and the size of the grammar cannot be approximated within a constant factor \[1\]. Furthermore, the connection with addition chains makes any algorithm with an approximation guarantee \(o(\log N/\log\log N)\) unlikely \[1\].

Approximation. The two first algorithms with approximation ratio \(O(\log(N/g))\) were due to Rytter \[14\] and independently Charikar et al. \[1\]. They both applied the LZ77 compression to the input string and transformed the obtained LZ77 representation to a grammar. The main idea was to require that the derivation tree of the intermediate constructed grammar was balanced, the former algorithm assumed AVL-condition, while the latter imposed that for a rule \(X \rightarrow YZ\) the lengths of words generated by \(Y\) and \(Z\) are within a certain multiplicative constant factor from each other.

Sakamoto \[15\] proposed a different approach, based on RePair \[10\], a practically implemented and used algorithm for grammar-based compression. His algorithm iteratively replaced pairs of different letters and maximal blocks of letters (\(a^\ell\) is a maximal block if that cannot be extended by \(a\) to either side). A special pairing of the letters was devised, so that it is ‘synchronising’: for any two appearances of the same string \(w\) in the instance we can represent \(w\) as \(w = w_1w_2w_3\), where \(w_1, w_3 = O(1)\) and \(w_2\) in both appearances are compressed in the same way (though \(w_1\) and \(w_3\) in those appearances can be compressed differently). The analysis considered 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.

However, to the author’s best knowledge and understanding, the presented analysis \[15\] is incomplete, as the cost of nonterminals introduced for the representation of maximal blocks is not bounded in the paper; the bound that the author was able to obtain using there presented approach is \(O(\log(N/g)^2)\).

Proposed Approach: Recompression. In this paper another algorithm is proposed, using the general approach of recompression, developed by the author, based on iterative application of two replacement schemes performed on the text \(T\):

**pair compression of \(ab\)** For two different symbols (i.e. letters or nonterminals) \(a, b\) such that substring \(ab\) appears in \(T\) replace each of \(ab\) in \(T\) by a fresh nonterminal \(c\).

**a’s block compression** For each maximal block \(a^\ell\) appearing in \(T\), where \(a\) is a letter or a nonterminal, replace all \(a^\ell\)’s in \(T\) by a fresh nonterminal \(a_\ell\).