Application of Lempel-Ziv factorization to the approximation of grammar-based compression

Wojciech Rytter *

Key-words: LZ-compression, minimal grammar, AVL-tree, AVL-grammar

Abstract

We introduce new type of context-free grammars, AVL-grammars, and show their applicability to grammar-based compression. Using this type of grammars we present $O(n \cdot \log|\Sigma|)$ time and $O(\log n)$-ratio approximation of minimal grammar-based compression of a given string of length n over an alphabet Σ and $O(k \cdot \log n)$ time transformation of $LZ77$ encoding of size k into a grammar-based encoding of size $O(k \cdot \log n)$. A preliminary version of this paper has been presented in [18], independently of [3], where grammar-based approximation has been attacked with different construction and a more complicated type of grammars (α-balanced grammars for $\alpha \leq 1 - \frac{1}{2}\sqrt{2}$). The AVL-grammar is a very natural and simple tool for grammar based compression, it is a straightforward extension of the classical AVL-tree.

1 Introduction

Text compression based on context free grammars, or equivalently, on straight-line programs, has recently attracted much attention, see [1, 10, 11] and [13, 14, 17, 19]. The grammars give a more structured type of compression and are more convenient for example in compressed pattern-matching, see [19]. In a grammar-based compression a single text w of length n is generated by a context-free grammar G. Assume we deal with grammars generating single words. Computing exact size of the minimal grammar-based compression is known to be NP-complete.

In the paper, using ideas similar to *unwinding* from [5] and *balanced grammars* from [9], we show a logarithmic relation between LZ-factorizations and minimal grammars. Re-

*Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097, Warszawa, Poland, and Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK. Email: `rytter@csc.liv.ac.uk`.
Research supported by grant EPSRC GR/N09077.

cently, approximation ratios of several grammar-based compression have been investigated by Lehman and Shelat in [14]. In this paper we propose a new grammar-based compression algorithm based on Lempel-Ziv factorization (denoted here by LZ), which is a version of $LZ77$-encoding [15]. For a string w of length n denote by $LZ(w)$ the Lempel-Ziv factorization of w. We show:

1. For each string w and its grammar-based compression G $|LZ(w)| = O(|G|)$;

1. Given $LZ(w)$, a grammar-based compression G' for w can be efficiently constructed with $|G'| = O(\log |w| \cdot |LZ(w)|)$.

This gives $\log n$-ratio approximation of minimal grammar-based compression, since LZ-factorization can be computed efficiently [4]. The grammar-based type of compression is more convenient than LZ-compression, especially in compressed and fully compressed pattern-matching. For simplicity assume that the grammars are in Chomsky normal form. The size of the grammar G, denoted by $|G|$, is the number of productions (rules), or equivalently the number of nonterminals of a grammar G in Chomsky normal form. Grammar compression is essentially equivalent to straight-line programs. A *grammar* (*straight-line program*) is a sequence of assignment statements:

$$X_1 = expr_1;\ X_2 = expr_2; \ldots;\ X_m = expr_m,$$

where X_i are nonterminals and $expr_i$ is a single (terminal) symbol, or $expr_i = X_j \cdot X_k$, for some $j, k < i$, where $\cdot$ denotes the concatenation of X_j and X_k. For each nonterminal X_i, denote by $val(X_i)$ the value of X_i, it is the string described by X_i. The string described by the whole straight-line program is $val(X_m)$. The size of the straight-line program is m.

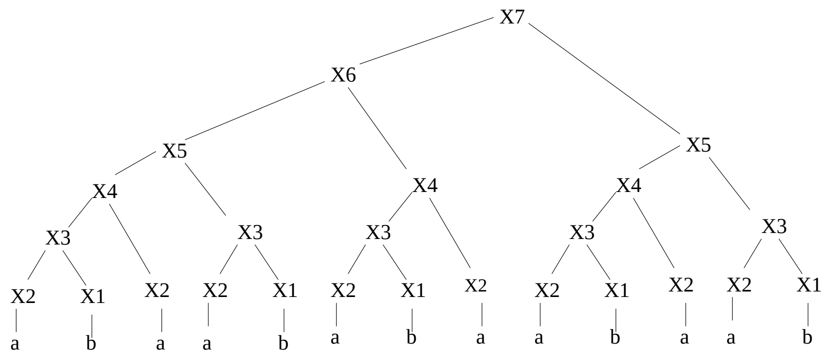

Figure 1: $Tree(G_7)$: the parse-tree of G_7. It is a binary tree: we assume that the nonterminals generating single terminal symbols are identified with these symbols. We have: $val(G_7) = Fib_7 = abaababaabaab$.

The problem of finding the smallest size grammar (or equivalently, straight line program) generating a given text is NP-complete. We consider the following problem (**approximation of grammar-based compression**):

Instance: given a text w of length n,

Question: construct in polynomial time a grammar G such that $val(G) = w$ and the *ratio* between $|G|$ and the size of the minimal grammar for w is *small.*

Example 1. Let us consider the following grammar G_7 which describes the $7th$ *Fibonacci word* $Fib_7 = abaababaabaab$. We have $|G_7| = 7$. This is the smallest size grammar in Chomsky normal form for Fib_7. However the general test for grammar minimality is computationally hard.

$$\begin{array}{llll} X_7 = X_6 \cdot X_5; & X_6 = X_5 \cdot X_4; & X_5 = X_4 \cdot X_3 & X_4 = X_3 \cdot X_2; \\ X_3 = X_2 \cdot X_1 & X_2 = a; & X_1 = b; & \end{array}$$

If A is a nonterminal of a grammar G then we sometimes identify A with the grammar G with the starting nonterminal replaced by A, all useless unreachable nonterminals being removed. In the parse tree for a grammar with the starting nonterminal A we can also sometimes informally identify A with the root of the parse tree.

2 LZ-factorizations and grammar-based factorizations

We consider a similar version of the LZ77 compression algorithm without *self-referencing* as one used in [5] (where it is called LZ1). Intuitively, LZ algorithm compresses the input word because it is able to discover some repeated subwords, see [4]. The Lempel-Ziv code defines a natural factorization of the encoded word into subwords which correspond to intervals in the code. The subwords are called *factors*. Assume that Σ is an underlying alphabet and let w be a string over Σ. The LZ-factorization of w is given by a decomposition: $w = f_1 \cdot f_2 \cdots f_k$, where $f_1 = w[1]$ and for each $1 \le i \le k$, f_i is the longest prefix of $f_i \dots f_k$ which occurs in $f_1 \dots f_{i-1}$. We can identify each f_i with an interval $[p, q]$, such that $f_i = w[p \dots q]$ and $q \le |f_1 \dots f_{i-1}|$. We identify LZ-factorization with $LZ(w)$. Its size if the number of factors. For a grammar G generating w we define the parse-tree $Tree(G)$ of w as a derivation tree of w, in this tree we identify (conceptually) terminal symbols with their parents, in this way every internal node has exactly two sons, see Figure 1. Define the partial parse-tree, denoted $PTree(G)$ as a maximal subtree of $Tree(G)$ such that for each internal node there is no node to the left having the same label. We define also the grammar factorization, denoted by G-factorization, of w, as a sequence of subwords generated by consecutive bottom nonterminals of $PTree(G)$, these nonterminals are enclosed by rectangles in Figure 2. Alternatively we can define G-factorization as follows: w is scanned from left to right, each time taking as next G-factor the longest unscanned prefix which is generated by a single nonterminal which has already occurred to the left or a single letter if there is no such nonterminal. The factors of LZ- and G-factorizations are called LZ-factors and G-factors, respectively.

Example 2. The LZ-factorization of the 7-th Fibonacci word Fib_7 is given by:

$$abaababaabaab \;=\; f_1\ f_2\ f_3\ f_4\ f_5\ f_6 \;=\; a\ b\ a\ aba\ baaba\ ab$$

The G_7-factorization is: $g_1\ g_2\ g_3\ g4\ g_5\ g_6 \;=\; a\ b\ a\ ab\ aba\ abaab$

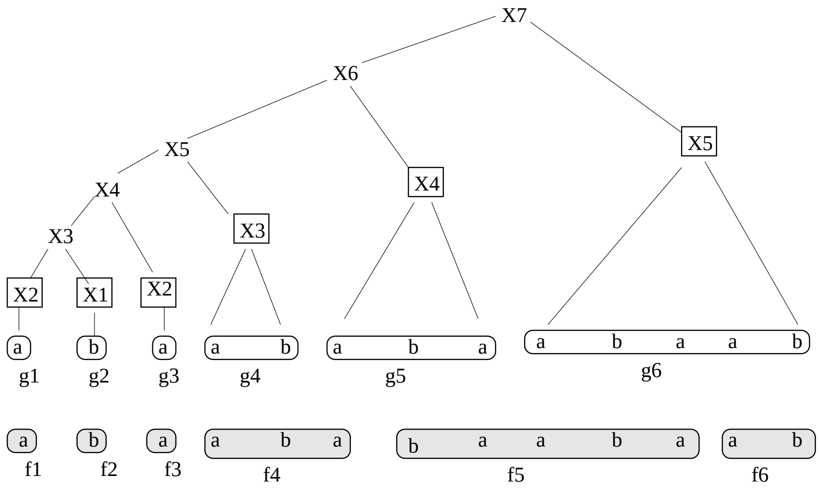

Figure 2: $PTree(G_7)$, LZ-factorization (shaded one) is shown below G_7 factorization of Fib_7. The number of LZ-factors does not exceed the number of grammar-based factors.

Theorem 1 *For each string w and its grammar-based compression G $|LZ(w)| \leq |G|$.*

Proof: Let G be a context free grammar in Chomsky normal form generating a single string w. Let $f_1 f_2 \ldots f_k$ be LZ-factorization and $g_1 g_2 \ldots g_r$ be the G-factorization of w.

Claim 1 $|G| \geq g$.
Proof:(of the claim) The nonterminals corresponding to G-factors do not need to be distinct, however all internal nodes of the tree $PTree(G)$ have different nonterminal labels, so there are at least $g-1$ internal nodes in this tree which correspond to nonterminals. Additionally there should be at least one nonterminal which production is of the type $A \to a$. Altogether there are at least g different nonterminals. $\square$

Claim 2 The number of LZ-factors is not greater than the number of G-factors.
Proof:(of the claim) We prove by induction on i that for each $i \leq \min(k, r)$ we have:

$$|g_1 g_2 \ldots g_i| \leq |f_1 f_2 \ldots f_i|.$$

If $|g_1 g_2 \ldots g_i| = |f_1 f_2 \ldots f_i|$ then $|f_{i+1}| \geq |g_{i+1}|$ because LZ-factorization is *greedy*, g_{i+1} is a prefix of the $f_{i+1} \ldots f_k$ which occurs in the subword $f_1 f_2 \ldots f_i$, so f_{i+1} is not shorter than g_{i+1}. Similar argument works for the case when $|g_1 g_2 \ldots g_i| < |f_1 f_2 \ldots f_i|$. In this case g_{i+1} can be already contained in $f_1 f_2 \ldots f_i$ or the suffix of g_{i+1} which is not contained in $f_1 f_2 \ldots f_i$ will be included in f_{i+1}. In all cases $|g_1 g_2 \ldots g_i g_{i+1}| \leq |f_1 f_2 \ldots f_i f_{i+1}|$. $\square$

Hence if $r \leq k$ then $|g_1 g_2 \ldots g_r| \leq |f_1 f_2 \ldots f_r|$ and $f_1 f_2 \ldots f_r = w$, since $g_1 g_2 \ldots g_r = w$. Consequently $k \leq r$. This completes the proof. $\square$

3 AVL-grammars

We introduce new type of grammars: AVL-grammars. They correspond naturally to AVL-trees. The first use of of a different type balanced grammars has appeared in [9]. AVL-trees are usually used in the context of binary search trees, here we use them in the context of storing in the leaves the consecutive symbols of the input string w. The basic operation is the concatenation of sequences of leaves of two trees. We use the standard AVL-trees, for each node v the balance of v, denoted $bal(v)$ is the difference between the height of the left and right subtrees of the subtree of T rooted at v. T is *AVL-balanced* iff $|bal(v)| \leq 1$ for each node v. We say that a grammar G is *AVL-balanced* if $Tree(G)$ is *AVL-balanced*. Denote by $height(G)$ the height of $Tree(G)$ and by $height(A)$ the height of the parse tree with the root labeled by a nonterminal A. The following fact is a consequence of a similar fact for AVL-trees, see [12].

Lemma 1 *If the grammar G is AVL-balanced then $height(G) = O(\log n)$.*

In case of *AVL-balanced* grammars in each nonterminal A additional information about the balance of A is kept: $bal(A)$ is the balance of the node corresponding to A in the tree $Tree(G)$. We do not define the balance of nodes corresponding to terminal symbols, they are identified with their fathers: nonterminals generating single symbols. Such nonterminals are leaves of $Tree(G)$, for each such nonterminal B we define $bal(B) = 0$.

Example 4. Let us consider $G = G_7$ and look at the tree in Figure 1. Only nonterminal nodes are considered. $bal(X_1) = bal(X_2) = bal(X_3) = 0$ and $bal(X_4) = \ldots bal(X_7) = +1$. Hence the grammar G_7 for the 7th Fibonacci word is *AVL-balanced*.

Lemma 2 *Assume A, B are two nonterminals of AVL-balanced grammars. Then we can construct in $O(|height(A) - height(B)|)$ time a AVL-balanced grammar $G = Concat(A, B)$, where $val(G) = val(A) \cdot val(B)$, by adding only $O(|height(A) - height(B)|)$ nonterminals.*

Proof: We refer to the 3rd volume of Knuth's book, page.474, see [12], for more detailed description of the *concatenation algorithm* for two AVL-*balanced* trees $T1$, $T2$ with roots A and B. Our AVL-trees contain keys (symbols) only in leaves, so to concatenate two trees we do not need to delete the root of one of them (implying a costly restructuring), see [12]. Assume that $height(T1) \geq height(T2)$, other case is symmetric. We follow the rightmost branch of $T1$, the heights of nodes decrease each time at most by 2. Then we stop at a node v such that $height(v) - height(T2) \in \{1, 0\}$. We create a new node v', its father is the father of v and its sons are v, $root(T2)$, see Figure 3.

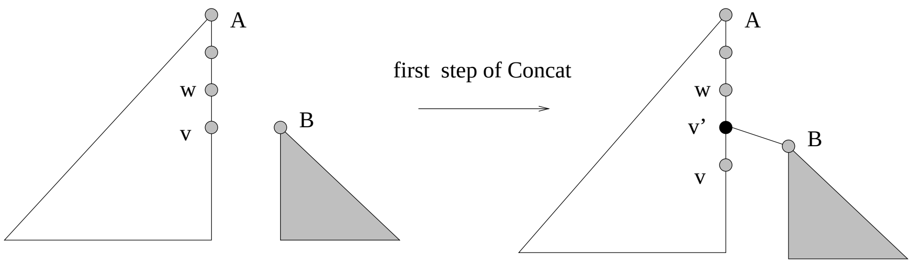

Figure 3: The first step of $Concat(A, B)$. The edge (w, v) is split into (w, v'), (v', v), where $height(v) = height(B)$ or $height(v) = height(B) + 1$. The node v' is a newly created node. The corresponding grammar productions are added.

The resulting tree can be unbalanced (by at most 2) on the rightmost branch. Suitable rotations are to be done, see Figure 4. The concatenating algorithm for AVL-trees can be applied to the parse-trees and automatically extended to the case of AVL-grammars. The real parse-tree could be even of an exponential size, however what we need is only its rightmost or leftmost branch, which can be recovered from the grammar going top-down.

There is one more technical detail distinguishing it from the concatenation of trees. It can happen that only a constant number of rotations have been done, which is reflected by an introduction of several new productions of the grammar, see Figure 4. However this would imply creating copies of old terminals on the path from v' to the root, due to the change of subtrees rooted at v. However the number of affected nonterminals is only $O(|height(A) - height(B)|)$. If we change production rule for a nonterminal in $Tree(G)$ we should do it on its newly created copy, since this nonterminal can occur in other places, and we cannot affect other parts of the tree. Possibly the structure of the tree is changed in one place at the bottom of the rightmost path. However for all nodes on this path the corresponding nonterminals have to change names to new ones, since sequences of leaves in their subtrees have changed (by a single symbol). The rebalancing has to be done only on the rightmost branch bottom-up starting at v. The part of this branch is of length $O(|height(A) - height(B)|)$. $\square$

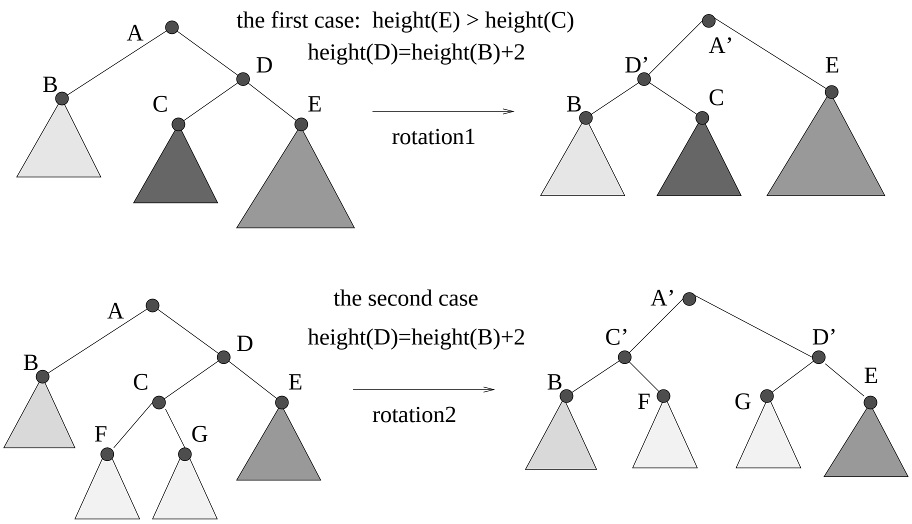

Figure 4: All nodes are well balanced except the root A, which is overbalanced to the right. There are two cases. A single rotation in $Tree(G)$ corresponds to a local change of constant number of productions and creation of some new nonterminals. The root becomes balanced, but its father or some node upwards can be still unbalanced and the processing goes up.

4 Construction of small grammar-based compression

Assume we have an LZ-factorization $f_1 f_2 \ldots f_k$ of w. We convert it into a grammar whose size increases by a logarithmic factor. Assume we have LZ-factorization $w = f_1 f_2 \ldots f_k$ and we have already constructed *good* (*AVL-balanced* and of size $O(i \cdot \log n)$) grammar G for the prefix $f_1 f_2 \ldots f_{i-1}$. If f_i is a terminal symbol generated by a nonterminal A then we set $G := Concat(G, A)$. Otherwise we locate the segment corresponding to f_i in the prefix $f_1 f_2 \ldots f_{i-1}$.

Due to the fact that G is balanced we can find a logarithmic number of nonterminals $S_1, S_2, \ldots S_{t(i)}$ of G such that $f_i = val(S_1) \cdot val(S_2) \cdot \ldots val(S_{t(i)})$, see Figure 5. The sequence $S_1, S_2, \ldots S_{t(i)}$ is called the *grammar decomposition* of the factor f_i.

We concatenate the parts of the grammar corresponding to this nonterminals with G, using the operation *Concat* mentioned in Lemma 2. Assume the first $|\Sigma|$ nonterminals corresponds to letters of the alphabet, so they exist at the beginning. We initialize G to the grammar generating the first symbol of w and containing all nonterminals for terminal symbols, they don't need to be initially *connected* to the string symbol. The algorithm starts with the computation of LZ-factorization, this can be done using suffix trees in $O(n \log |\Sigma|)$ time, see [4].

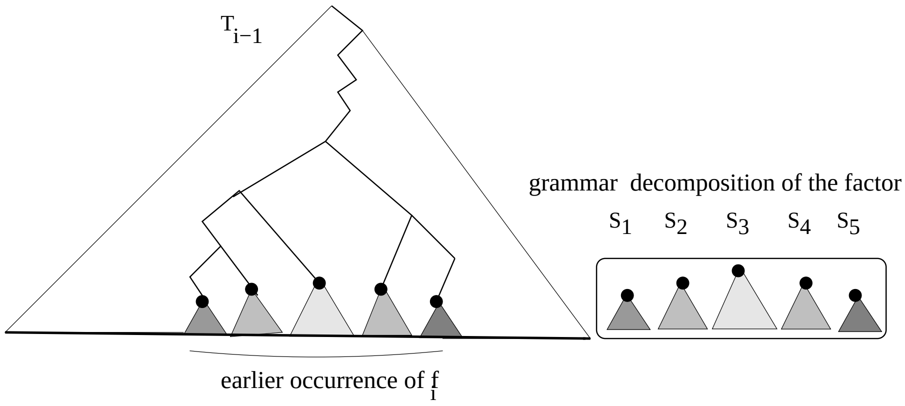

Figure 5: The next factor f_i is split into segments corresponding to nonterminals occurring to the left. There are $O(\log n)$ segments since the height of the parse-tree is $O(\log n)$. Observe that $height(S1) \leq height(S-2) \leq height(S3) \geq height(S4) \geq height(S5)$. The sequence of heights of subtree $S_1, S_2, \ldots S_{t(i)}$ is *bitonic.*

If LZ-factorization is too large (exceeds $n/\log n$) then we neglect it and write a trivial grammar of size n generating a given string. Otherwise we have only $k \leq n \cdot \log n$ factors, they are processed from left to right. We perform :

> **ALGORITHM** *Construct-Grammar*(w); $\{|w| = n\}$
> compute LZ factorization $f_1 f_2 f_3 \ldots f_k$
> { in $O(n \log |\Sigma|)$ time , using suffix trees}
> **if** $k > n/\log(n)$ **then return** trivial $O(n)$ size grammar
> **else**
> **for** $i = 1$ **to** k **do**
> **(1)** Let $S_1, S_2, \ldots, S_{t(i)}$ be grammar decomposition of f_i;
> **(2)** $H := Concat(S_1, S_2, \ldots, S_{t(i)})$;
> **(3)** $G := Concat(G, H)$;
> **return** G;

Due to Lemma 2 we have $t(i) = O(\log n)$, so the number of two-arguments concatenations needed to implement single step (2) is $O(\log n)$, each of them adding $O(\log n)$ nonterminals. Steps (1) and (3) can be done in $O(\log n)$ time, since the height of the grammar is logarithmic. Hence the algorithm gives $O(\log^2(n)$-ratio approximation.

At the cost of slightly more complicated implementation of step (2) $\log^2 n$-ratio can be improved to a $\log n$-ratio approximation. The key observation is that the sequence of heights of subtrees corresponding to segments S_i of next LZ-factor is *bitonic*, see Figure 5. We can split this sequence into two subsequences: height-nondecreasing sequence $R_1, R_2, \ldots R_k$, called *right-sided*, and height-nonincreasing sequence $L_1, L_2, \ldots L_r$, called *left-sided.*

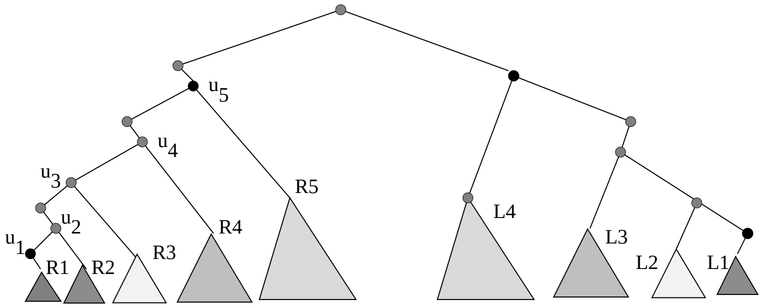

Figure 6: An example of the grammar decomposition of the next factor f_i into a sequence of right-sided and a sequence of left-sided subtrees: $R1 \cdot R2 \cdot R3 \cdot R4 \cdot R5 \cdot L4 \cdot L3 \cdot L2 \cdot L1$ The *right-sided* sequence of subtrees is $R1, R2, \ldots R5$.

Lemma 3 *Assume $R_1, R_2, \ldots R_k$ is a right-sided sequence, and G_i is the AVL-grammar which results by concatenating $R_1, R_2, \ldots R_i$ from left-to-right. Then*

$$|height(R_i) - height(G_{i-1})| \;\leq\; \max\,\{(height(R_i) - height(R_{i-1}),\; 1\}$$

Proof: We use the following obvious fact holding for any two nonterminals A, B. Denote $h = \max\{height(A),\; height(B)\}$, then we have:

$$h \leq height(Concat(A, B)) \leq h + 1 \tag{1}$$

Let u_i be the father of the node corresponding to R_i, see Figure 6. We show:

Claim. $height(G_i) \leq height(u_i)$.

The proof of the claim is by induction. For $i = 1$ we have $G_i = R_i$. In this case $height(u_1) = height(R_1){+}1 \geq height(G_1)$. Assume the claim holds for $i{-}1$: $height(G_{i-1}) \leq height(u_{i-1})$. There are two possibilities.

Case 1. $height(G_{i-1}) \leq height(R_i)$.
Then, according to Equation 4: $height(G_i) \leq height(G_{i-1}{+}1$, and due to the inductive assumption $height(G_i) \leq height(u_{i-1}) + 1 \leq height(u_i)$.

Case 2. $height(G_{i-1}) \leq height(R_i)$.
Then, again using Equation 4, $height(G_i) \leq height(R_i) + 1 \leq height(u_i)$.

This completes the proof of the claim. We go now to the main part of the proof of the lemma.
If $height(G_{i-1}) \geq height(R_i)$ then

$$\begin{aligned}|height(R_i) - height(G_{i-1}))| &= height(G_{i-1}) - height(R_i)\\ &\leq height(u_{i-1} - height(R_i) \leq 1.\end{aligned}$$

The last inequality follows from the AVL-property.

If $height(G_{i-1}) \leq height(R_i)$ then

$$\begin{aligned}|height(R_i) - height(G_{i-1})| &= height(R_i) - height(G_{i-1})\\ &\leq height(R_i) - height(R_{i-1}),\end{aligned}$$

since $height(G_{i-1}) \geq height(R_{i-1})$. This completes the proof. □

Theorem 2 *We can construct in a $O(n \log |\Sigma|)$ time a $O(\log n)$-ratio approximation of a minimal grammar-based compression.*
Given LZ-factorization of length k we can construct a corresponding grammar of size $O(k \log n)$ in time $O(k \log n)$.

Proof: The next factor f_i is decomposed into segments $S_1, S_2, \ldots S_{t(i)}$. It is enough to show that we can create in $O(\log n)$ time an AVL-grammar for the concatenation of $S_1, S_2, \ldots, S_{t(i)}$ by adding only $O(\log n)$ nonterminals and productions to G, assuming that the grammars for $S_1, S_2, \ldots, S_{t(i)}$ are available.

The sequence $(S_1, S_2, \ldots, S_{t(i)}$ consists of a right-sided sequence and left-sided sequence. The grammars H', H'' corresponding to these sequences are computed (by adding logarithmically many nonterminals to G), due to Lemma 3. Then H', H'' are concatenated. Assume $R_1, R_2, \ldots R_k$ are right-sided subtrees. Then the total work and number of extra nonterminals needed to concatenate $R_1, R_2, \ldots R_k$ can be estimated as follows:

$$\sum_{i=2}^{k} |height(R_i) - height(G_{i-1})| \leq \sum_{i=2}^{k} \max \{height(R_i) - height(R_{i-1}),\ 1\}$$

$$\leq \sum_{i=2}^{k} (height(R_i) - height(R_{i-1})) + \sum_{i=2}^{k} 1 \leq height(R_k) + k = O(\log n).$$

The same applies to the left-sided sequence in a symmetric way. Altogether processing each factor f_i enlarges the grammar by an $O(\log n)$ additive factor and needs $O(\log n)$ time. To get $\log n$-ratio we consider only the case when the number k of factors is $O(n/\log n)$. LZ-factorization is computed in $O(n \log |\Sigma|)$ time using suffix trees, ($O(n)$ time for integer alphabets, see [7])). □

5 From $\log n$-ratio to $\log(n/g)$-ratio approximation

There is possible a rather cosmetic improvement of the approximation ratio. Let g be the size of the minimal grammar-based compression and assume we have a greedy LZ-factorization of a string w of size n into s factors, the number s is also a lower bound on g. The improvement is a direct application of a method from the paper on compressed matching of Farach and Thorup [5], (In their notation $n = U,\ g = n$). In [5] they improved a starting factor $\log n$ to $\log(n/g)$ by introducing new cut-points and refining factorization. Exactly in the same way $\log n$ can be improved to get $\log(n/g)$.

We insert virtually in the uncompressed string s cuts at positions which are multiples of n/s. In this way we get a new factorization $w = f_1 f_2 \dots f_r$, since possibly some factors in LZ-factorization were split, now each factor is of size at most n/s.

The input string w is split by s new cut-points into subwords $w_1, w_2, \dots w_s$ each of size n/s. Previously we processed factors f_i for $i = 1, 2, \dots r$ and produced incrementally the grammar for $f_1 f_2 .. f_i$. Now we process the factors in *packages*. Each of $w_1, w_2, \dots w_s$ is processed separately, in the order $1, 2, \dots, s$, producing separate grammars for each of $w_1, w_2, \dots w_s$. The sets of non-terminals of the grammars are not necessarily disjoint. When processing w_i we have already the grammars for $w_1,\ w_2,\ ..w_{i-1}$.

Assume w_i consists of factors $f_t, f_{t+1}, \dots f_{t'}$. The processing of w_i consists in considering consecutive factors $f_t, f_{t+1}, \dots f_{t'}$. For each factor f_p, $p\ =\ t, t+1, \dots, t'$, the grammar for $f_t, f_{t+1}, \dots, f_p$ is created by concatenating the grammar for $f_t, f_{t+1}, \dots, f_{p-1}$ with the non-terminal for f_p. The concatenation is done as concatenation of AVL-grammars described previously.

In this way the height of all non-terminals is at most $\log(n/s)$. Afterward we add $s-1$ non-terminals to create from non-terminals for $w_1, w_2, \dots w_s$ the grammar for the whole string. Each time we process a factor we add $O(\log(n/s))$ non-terminals (bounded by a maximal height), there are $r = O(s)$ factors. At the end we add $s-1$ non-terminals. Altogether the resulting binary grammar has $O(n \log(n/s))$ non-terminals.

Let g be the size of the minimal grammar based compression of a given string of length n. We have $g \log(n/g) \geq s \log(n/s)$ since $g \geq s$. In this way we have proved the following fact.

Theorem 3 *We can construct in polynomial time $O(\log(n/g))$-ratio approximation of a minimal grammar compression, where g is the size of the minimal grammar based compression of a given string of length n.*

6 Final remarks

The main result is $\log n$-ratio approximation of a minimal grammar-based compression. However the transformation of LZ-encodings into grammars is of the same importance (or maybe even more important). The grammars are easier to deal than LZ-encodings, particularly in compressed pattern-matching, see [5]. Our method leads to an simpler alternative algorithm for LZ77-compressed pattern-matching. Another useful feature of our grammars is their logarithmic height. We can take any grammar G (straight-line program) generating a single text and produce the G-factorization. Then we can transform it into a balanced grammar in the same way as it is done for LZ-factorization. This gives an alternative algorithm for balancing grammars and straight line programs, it has been originally done using the methods from parallel tree contraction.

Theorem 4 *Assume G is a grammar (straight-line program) of size k generating a single string of size n. Then we can construct in $O(k \log n)$ time an equivalent grammar of height $O(\log n)$.*

Assume we have a grammar-compressed pattern and a text, where $m_1,\ m_2$ are the sizes of their compressed versions. In [16] an improved algorithm for fully compressed pattern-matching algorithm has been given, which works in time $O(m_1 \cdot m_2 \cdot h_1 \cdot h_2)$, where $h_1,\ h2$ are the heights of corresponding grammars. We can use AVL-grammar together with the algorithm from [16] to texts which are polynomially related to their compressed versions. This gives an improvement upon the result of [6] for LZ fully compressed matching in case when encodings are polynomially related to explicit texts (which is a typical case). Let notation $\widetilde{O}(g(k))$ stand for $O(g(k) \log^c(k))$, where c is a constant.

Theorem 5 *Given LZ-encodings of sizes m_1 and m_2 of the pattern P and a text T respectively. Assume that the original texts are polynomially related to their compressed versions. Then we can do fully compressed pattern-matching in time $\widetilde{O}(m_1 \cdot m_2)$.*

Grammar compression can be also considered for two-dimensional texts, but this case is much more complicated, see [2].

References

[1] A. Apostolico, S. Leonardi, Some theory and practice of greedy off-line textual substitution, DCC 1998, pp. 119-128

[2] P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. W. Rytter, On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts, *Proceedings of the* 8^{th}

Annual Symposium on Combinatorial Pattern Matching LNCS 1264, Edited by A. Apostolico and J. Hein, (1997), pp. 40–51. Full version to appear in JCSS 2002.

[3] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai and Abhi Shelat, Approximating The Smallest Grammar: Kolmogorov Complexity in Natural Models, STOC 2002

[4] M. Crochemore and W. Rytter, *Text Algorithms,* Oxford University Press, New York (1994).

[5] M. Farach and M. Thorup, String matching in Lempel-Ziv compressed strings, *Proceedings of the* 27^{th} *Annual Symposium on the Theory of Computing* (1995), pp. 703–712.

[6] L. Gąsieniec, M. Karpinski, W. Plandowski and W. Rytter, Efficient Algorithms for Lempel-Ziv Encoding, *Proceedings of the* 5^{th} *Scandinavian Workshop on Algorithm Theory*. Springer-Verlag (1996).

[7] Martin Farach, Optimal suffix tree construction with large alphabets, FOCS 1997

[8] D. Gussfield, Algorithms on strings and arrays, Cambridge University Press, 1997

[9] M. Hirao, A. Shinohara, M. Takeda, S. Arikawa, Faster fully compressed pattern matching algorithm for balanced straight-line programs", Proc. of 7th International Symposium on String Processing and Information Retrieval (SPIRE2000), pp. 132-138. IEEE Computer Society, September 2000

[10] M. Karpinski, W. Rytter and A. Shinohara, Pattern-matching for strings with short description, *Nordic Journal of Computing*, 4(2):172-186, 1997

[11] J. Kieffer, E. Yang, Grammar-based codes: a new class of universal lossless source codes, IEEE Trans. on Inf. Theory 46 (2000) pp. 737-754

[12] D. Knuth, *The Art of Computing, Vol. III Second edition.* Addison-Wesley (1998), page.474

[13] J. K. Lanctot, Ming Li, En-hui Yang, Estimating DNA Sequence Entropy, SODA 2000

[14] E. Lehman, A. Shelat, Approximation algorithms for grammar-based compression, SODA 2002

[15] J. Ziv and A. Lempel, A Universal algorithm for sequential data compression, *IEEE Transactions on Information Theory* IT–23 (1977), pp. 337–343.

[16] M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern-matching algorithm for strings in terms of straight-line programs, Journal of Discrete Algorithms, Vol.1, pp. 187-204, 2000.

[17] C. Nevill-Manning, Inferring sequential structure, PhD thesis, University of Waikato, 1996

[18] W. Rytter, Application of Lempel-Ziv Factorization to the Approximation of Grammar-Based Compression. Combinatorial Pattern Matching, June 2002, Lecture Notes in Computer Science 2373, 20-31, 2002

[19] W. Rytter, Compressed and fully compressed pattern-matching in one and two-dimensions, Proceedings of IEEE, November 2000, Volume 88, Number 11, pp. 1769-1778