The smallest grammar problem revisited

Hideo Bannai, Momoko Hirayama, Danny Hucke, Shunsuke Inenaga, Artur Jeż, Markus Lohrey and Carl Philipp Reh

***Abstract*—In a seminal paper, Charikar et al. derive upper and lower bounds on the approximation ratios for several grammar-based compressors, but in all cases there is a gap between the lower and upper bound. Here the gaps for LZ78 and BISECTION are closed by showing that the approximation ratio of LZ78 is $\Theta((n/\log n)^{2/3})$, whereas the approximation ratio of BISECTION is $\Theta(\sqrt{n/\log n})$. In addition, the lower bound for RePair is improved from $\Omega(\sqrt{\log n})$ to $\Omega(\log n/\log\log n)$. Finally, results of Arpe and Reischuk relating grammar-based compression for arbitrary alphabets and binary alphabets are improved.**

***Index Terms*—string compression, smallest grammar problem, approximation algorithm, LZ78, RePair**

I. Introduction

A. Grammar-based compression

The idea of grammar-based compression is based on the fact that in many cases a word w can be succinctly represented by a context-free grammar that produces exactly w. Such a grammar is called a *straight-line program* (SLP for short) for w. For instance, $S \to cAABB$, $A \to aab$, $B \to CC$, $C \to cb$ is an SLP for the word $caabaabcbcbcbcb$. SLPs were introduced independently by various authors in different contexts [2], [3], [4], [5] and under different names. For instance, in [4], [5] the term *word chains* was used since SLPs generalize addition chains from numbers to words. Probably the best known example of a grammar-based compressor is the already classical LZ78-compressor of Lempel and Ziv [3]. Indeed, it is straightforward to transform the LZ78-representation of a word w into an SLP for w. Other well-known grammar-based compressors are BISECTION [6], SEQUITUR [7], and RePair [8], just to mention a few.

A central question asked from the very beginning in the area of grammar-based compression is how to measure the quality of an SLP, or, more broadly, the quality of the grammar-based compressor that computes an SLP for a given input word. One can distinguish two main approaches for such quality measures: (i) bit-based approaches, where one analyzes the bit length of a suitable binary encoding of an SLP and (ii) size-based approaches which measure the quality of an SLP by its size. The size of an SLP is defined as the sum of the lengths of all right-hand sides of the SLP (the SLP from the previous paragraph has size 12). Let us briefly survey the literature on these two approaches before we explain our main results in Section I-D.

H. Bannai is with Tokyo Medical and Dental University.
D. Hucke, M.Lohrey and C. P. Reh are with the University of Siegen.
S. Inenaga is with Kyushu University.
M. Hirayama was with Kyushu University.
A. Jeż is with the University of Wroclaw.

A short version of this paper appeared in the Proceedings of SPIRE 2016 [1].

Markus Lohrey was supported by the DFG research project LO 748/10-1 (QUANT-KOMP). A. Jeż was supported under National Science Centre, Poland project number 2017/26/E/ST6/00191. S. Inenaga was supported by JSPS KAKENHI Grant Number JP17H01697 and by JST PRESTO Grant Number JPMJPR1922.

B. Bit-based approaches

It seems that the first attempt at evaluating a grammar-based compressor was done for LZ78 by Ziv and Lempel [3], who developed their own methodology of comparing (finite state) compressors: In essence, define $L_s(w)$ as the length of an appropriate bit encoding of the output produced by LZ78 with window-size s on input w and by $L_s^*(w)$ the smallest bit-size achievable by a finite-state compressor with s states on input w. Fix an infinite word $a_1a_2\cdots$, where each a_i is a letter. It‘was shown that $\lim_{s\to\infty}\limsup_{n\to\infty}\frac{L_s(a_1\cdots a_n)}{L_s^*(a_1\cdots a_n)} = 1$. In other words, LZ78 is optimal (up to lower order terms) among finite-state compressors, assuming that it runs long enough.

Later, a systematic evaluation of grammar-based compressors was done using the information theoretic paradigm. In [9], [6], [10], [11], grammar-based compressors have been used in order to construct universal codings in the following sense: for every finite state source and every input string w of length n (that is emitted with non-zero probability by the source), the coding length of w is bounded by $-\log_2 P(w) + R(n)$, where $P(w)$ is the probability that the source emits w (thus $-\log_2 P(w)$ is the self-information of w) and $R(n)$ is a function in $o(n)$. The function $R(n)/n$ is called the redundancy; it converges to zero. In [9], [6], [10], [11] the code for w is constructed in two steps: First, an SLP is computed for w using a grammar-based compressor. In a second step this SLP is encoded by a bit string using a suitable binary encoding (see also [12] for the problem of encoding SLPs within the information-theoretic limit). In [9] it was shown that the redundancy can be bounded by $\mathcal{O}(\log\log n/\log n)$ provided the grammar-based compressor produces an SLP of size $\mathcal{O}(n/\log n)$ for every input string of length n (this assumes an alphabet of constant size σ; otherwise an additional factor $\log\sigma$ enters the bounds). The size bound $\mathcal{O}(n/\log n)$ holds for all grammar-based compressors that produce so-called irreducible SLPs [9], which roughly speaking means that certain redundancies in the SLP are eliminated. Moreover, every SLP can be easily made irreducible by a simple post-processing [9]. In [10], the redundancy bound from [9] was improved to $\mathcal{O}(1/\log n)$ for so-called structured grammar-based codes.

Recently, bounds in terms of the k-th order empirical entropy $H_k(w)$ of the input string w have been shown for grammar-based compressors [13], [14]. Again, these results assume a suitable binary encoding of SLPs. In [14] it was shown that the length of the binary encoding (using the encoding from [9]) of an irreducible SLP for a string w can be bounded by $H_k(w) \cdot |w| + \mathcal{O}(nk \log \sigma / \log_\sigma n)$, where n is the length of the input string and σ is the size of the alphabet. Note that the additional additive term $\mathcal{O}(nk \log \sigma / \log_\sigma n)$ is in $o(n \log \sigma)$ under the standard assumption that $k = o(\log_\sigma n)$. In [13] similar bounds are derived for more natural binary encodings of SLPs. On the other hand, a lower bound of $H_k(w) \cdot |w| + \Omega(nk \log \sigma / \log_\sigma n)$ was recently shown for a wide class of "natural" grammar-based compressors [15]. Hence, the mentioned upper bounds from [13], [14] are tight.

C. Size-based approaches

Bit-based approaches analyze the length of the binary encoding of the SLP. For this, one has to fix a concrete binary encoding. In contrast, the size of the SLP (the sum of the lengths of all right-hand sides) abstracts away from the concrete binary encoding of the SLP. Analyzing this more abstract quality measure has also some advantages: SLPs turned out to be particularly useful for the algorithmic processing of compressed data. For many algorithmic problems on strings, efficient algorithms are known in the setting where the input strings are represented by SLPs, see [16], [17] for some examples. For the running time of these algorithms, the size of the input SLPs is the main parameter whereas the concrete binary encoding of the SLPs is not relevant. Another research direction where only the SLP size is relevant arises from the recent work on string attractors, where the size of a smallest SLP for a string is compared with other string parameters that arise from dictionary compression (number of phrases in the LZ77 parse, minimal number of phrases in a bidirectional parse, number of runs in the Burrows-Wheeler transform) [18].

Another important aspect when comparing the bit-based approach (in particular, entropy bounds for binary encoded SLPs) and the size-based approach was also emphasized in [19, Section VI]: entropy bounds are often no longer useful when low-entropy strings are considered; see also [20] for an investigation in the context of Lempel-Ziv compression. Consider for instance the entropy bounds in [13], [14]. Besides the k-th order empirical entropy of the input string, these bounds also contain an additive term of order $\mathcal{O}(nk \log \sigma / \log_\sigma n)$ (and by the result from [15] this is unavoidable). Similar remarks apply to the redundancy bound in [9], where the output bit length of the grammar-based compressor is bounded by the self-information of the input string (with respect to a k-th order finite state source) plus a term of order $\mathcal{O}(n(k+\log \log_\sigma n)/\log_\sigma n)$. For input strings with low entropy/self-information these additive terms can be much larger than the entropy/self-information. For such input strings the existing entropy/redundancy bounds do not make useful statements about the performance of a grammar-based compressor. Low entropy strings are common in practice, for example they appear frequently in natural languages.

A first investigation of the SLP size was done by Berstel and Brlek [4], who proved that the function $g(\sigma, n) = \max\{g(w) \mid w \in \{1, \ldots, \sigma\}^n\}$, where $g(w)$ is the size of a smallest SLP for the word w, is in $\Theta(n/\log_\sigma n)$. Note that $g(\sigma, n)$ measures the worst case SLP-compression over all words of length n over an alphabet of size σ. It is worth noting that addition chains [21] are basically SLPs over a singleton alphabet and that $g(1, n)$ is the size of a smallest addition chain for n (up to a constant factor).

Constructing a smallest SLP for a given input word is known as the *smallest grammar problem*. Storer and Szymanski [22] and Charikar et al. [19] proved that it cannot be solved in polynomial time unless $\mathsf{P} = \mathsf{NP}$. Moreover, Charikar et al. [19] showed that, unless $\mathsf{P} = \mathsf{NP}$, one cannot compute in polynomial time for a given word w an SLP of size $< (8569/8568) \cdot g(w)$. The construction in [19] uses an alphabet of unbounded size, and it was unknown whether this lower bound holds also for words over a fixed alphabet. In [19] it is stated that the construction in [22] shows that the smallest grammar problem for words over a ternary alphabet cannot be solved in polynomial time unless $\mathsf{P} = \mathsf{NP}$. But this is not clear at all, see the recent paper [23] for a detailed explanation. In the same paper [23] it was shown that the smallest grammar problem for an alphabet of size 24 cannot be solved in polynomial time unless $\mathsf{P} = \mathsf{NP}$ using a rather complex construction. It is far from clear whether this construction can be adapted so that it works also for a binary alphabet. Another idea for showing NP-hardness of the smallest grammar problem for binary words is to reduce the smallest grammar problem for unbounded alphabets to the smallest grammar problem for a binary alphabet. This route was investigated in [24], where the following result was shown for every constant c: If there is a polynomial time grammar-based compressor that computes an SLP of size $c \cdot g(w)$ for a given binary input word w, then for every $\varepsilon > 0$ there is a polynomial time grammar-based compressor that computes an SLP of size $(24c + \varepsilon) \cdot g(w)$ for a given input word w over an arbitrary alphabet. The construction in [24] uses a quite technical block encoding of arbitrary alphabets into a binary alphabet.

A size-based quality measure for grammar-based compressors is the approximation ratio [19]: For a given grammar-based compressor $\mathcal{C}$ that computes from a given word w an SLP $\mathcal{C}(w)$ for w one defines the *approximation ratio* of $\mathcal{C}$ on w as the quotient of the size of $\mathcal{C}(w)$ and the size $g(w)$ of a smallest SLP for w. The *approximation ratio* $\alpha_{\mathcal{C}}(n)$ is the maximal approximation ratio of $\mathcal{C}$ among all words of length n over any alphabet. The approximation ratio is a useful measure for the worst-case performance of a grammar-based compressor, where the worst-case over all strings of a certain length is considered. This includes also low-entropy strings, for which the existing entropy/redundancy bounds are no longer useful as argued above. In this context one should also emphasize the fact that the entropy/redundancy bounds from [9], [14] apply to all grammar-based compressors that produce irreducible SLPs. As mentioned above, this property can be easily enforced by a simple post-processing of the SLP. This shows that the entropy/redundancy bounds from [9], [14] are not useful for a fine-grained comparison of grammar-based compressors. In contrast, the approximation ratios of grammar-based compressors can differ significantly (even if they produce irreducible SLPs). In other words, the concept of approximation ratio can detect fine differences between grammar-based compressors that behave exactly the same in terms of entropy/redundancy bounds.

Charikar et al. [19] initiated a systematic investigation of the approximation ratio of various grammar-based compressors (LZ78, BISECTION, Sequential, RePair, LongestMatch, Greedy). They proved lower and upper bounds for the approximation ratios of theses compressor, but for none of them the lower and upper bounds match. Moreover, Charikar et al. present a linear time grammar-based compressor with an approximation ratio of $\mathcal{O}(\log n)$. Other linear time grammar-based compressors which achieve the same approximation ratio can be found in [25], [26], [27], [28]. It is unknown whether there exist grammar-based compressors that work in polynomial time and have an approximation ratio of $o(\log n)$. Getting a polynomial time grammar-based compressor with an approximation ratio of $o(\log n/\log\log n)$ would solve a long-standing open problem on addition chains [19], [21].

D. Results of the paper

Our first main contribution (Section III) is an improved analysis of the approximation ratios of LZ78, BISECTION, and RePair. These compression algorithms are among the most popular grammar-based compressors. LZ78 is a classical algorithm and the basis of several widely used text compressors such as LZW (Lempel-Ziv-Welch). RePair shows in many applications the best compression results among the tested grammar-based compressors [29], [30] and found applications, among others, in web graph compression [31], different scenarios related to word-based text compression [32], searching compressed text [33], suffix array compression [34] and (in a slightly modified form in) XML compression [35]. Some variants and improvements of RePair can be found in [29], [36], [30], [37], [38]. BISECTION was first studied in the context of universal lossless compression [6] (called MPM there). On bit strings of length 2^n, BISECTION produces in fact the ordered binary decision diagram (OBDD) of the Boolean function represented by the bit string; see also [39]. OBDDs are a widely used data structure in the area of hardware verification.

For LZ78 and BISECTION we close the gaps for the approximation ratio from [19]. For this we improve the corresponding lower bounds from [19] and obtain the approximation ratios $\Theta((n/\log n)^{1/2})$ for BISECTION and $\Theta((n/\log n)^{2/3})$ for LZ78. We prove both lower bounds using a binary alphabet. These are the first exact (up to constant factors) approximation ratios for practical grammar-based compressors. We also improve the lower bound for RePair from $\Omega\left(\sqrt{\log n}\right)$ to $\Omega\left(\log n/\log\log n\right)$ using a binary alphabet (Theorem III.7). The previous lower bound from [19] used a family of words over an alphabet of unbounded size. Our new lower bound for RePair is still quite far away from the best known upper bound of $\mathcal{O}((n/\log n)^{2/3})$ [19]. On the other hand, the lower bound $\Omega\left(\log n/\log\log n\right)$ is of particular interest, since it was shown in [19] that a grammar-based compressor with an approximation ratio of $o(\log n/\log\log n)$ would improve Yao's method for computing a smallest addition chain for a set of numbers [21], which is a long standing open problem. Our new lower bound excludes RePair as a candidate for improving Yao's method. Let us also remark that RePair belongs to the class of so-called *global grammar-based compressors* (other examples are LongestMatch [9] and Greedy [40]). Analyzing the approximation ratio of global algorithms seems to be very difficult. We can quote here Charikar et al. [19]: "Because they [global algorithms] are so natural and our understanding is so incomplete, global algorithms are one of the most interesting topics related to the smallest grammar problem that deserve further investigation." In the specific context of singleton alphabets, a detailed investigation of the approximation ratios of global grammar-based compressors was recently examined in [41].

Our second main contribution deals with the hardness of the smallest grammar problem for words over a binary alphabet. As mentioned above, it is open whether this problem is NP-hard. This is one of the most intriguing unsolved problems in the area of grammar-based compression. Recall that Arpe and Reischuk [24] used a quite technical block encoding to show that if there is a polynomial time grammar-based compressor with approximation ratio c (a constant) on binary words, then there is a polynomial time grammar-based compressor with approximation ratio $24c+\varepsilon$ for every $\varepsilon > 0$ on arbitrary words. Here, we present a very simple construction, which encodes the i-th alphabet symbol by a^ib, and yields the same result as in [24] but with $24c+\varepsilon$ replaced by $6c$ (Theorem IV.1). In order to show NP-hardness of the smallest grammar problem for binary strings, one would have to reduce the factor 6 to at most $8569/8568$. This follows from the inapproximability result for the smallest grammar problem from [19].

E. Limitations of our techniques

We think that our new lower bound $\Omega\left(\log n/\log\log n\right)$ for RePair can be further improved. Unfortunately, the approach from the proof of Theorem III.7 is unlikely to yield an improvement beyond an approximation ratio of $\Omega(\log n)$. The words $s_k \in \{a,b\}^*$ used to achieve our lower bound are of the form $s_k = a^{n_1}ba^{n_2}\cdots ba^{n_k}$, where $k \in \Theta(\log|s_k|)$. The choice of those words is inspired by the strong connection between an addition chain for a set of numbers $n_1,\ldots,n_k$ and an SLP for words of the form $a^{n_1}b_1a^{n_2}\cdots b_{k-1}a^{n_k}$ over the alphabet $\{a,b_1,\ldots,b_{k-1}\}$. The reader can find a detailed explanation of this connection in [19, Thm. 2]. However, for words of this form it is not hard to see that RePair produces an SLP of size $\mathcal{O}(k\cdot\log(\max\{n_1,\ldots,n_k\}))$ while a smallest SLP has size at least $\Omega(k)$. This limits the achievable approximation ratio to $\Omega(\log n)$ in our setting.

In general, it is worth mentioning that the words used to prove our lower bounds for LZ78, BISECTION and RePair are specifically designed to exploit the weaknesses of the considered grammar-based compressors. It is therefore unlikely that our constructions lead to general results regarding the approximation ratio for a large class of grammar-based compressors.

Concerning Theorem IV.1, we do not see how to further reduce the constant 6 with our techniques (recall that in order to show NP-hardness of the smallest grammar problem for binary strings, one would have to push to constant down to 8569/8568). Our proof of Theorem IV.1 involves two steps:

- First we define an encoding function $\varphi\colon \Sigma^* \to \{0,1\}^*$ such that from an SLP for a string $w \in \Sigma^*$ one can construct in polynomial time an SLP for $\varphi(w)$, which is at most three times larger than the initial SLP for w (Lemma IV.2).
- Second, we show that from an SLP for $\varphi(w)$ one can construct in polynomial time an SLP $\mathbb{A}$ for w, which is at most two times larger than the initial SLP for $\varphi(w)$. (Lemma IV.3).

The constant 6 in Theorem IV.1 is then the product of two factors (2 and 3) in Lemmas IV.2 and IV.3. One might try to come up with a better encoding φ for which the product of two blow-up factors in Lemmas IV.2 and IV.3 is strictly below 6. So far, we neither see how to reduce the factor 3 in Lemma IV.2 nor the factor 2 in Lemma IV.3.

II. Straight-line Programs

Let $w = a_1 \cdots a_n$ $(a_1, \dots, a_n \in \Sigma)$ be a *word* over an *alphabet* Σ. The length $|w|$ of w is n and we denote by ε the word of length 0. Let $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ be the set of nonempty words. For $w \in \Sigma^+$, we call $v \in \Sigma^+$ a *factor* of w if there exist $x, y \in \Sigma^*$ such that $w = xvy$. If $x = \varepsilon$ (respectively $y = \varepsilon$) then we call v a *prefix* (respectively *suffix*) of w. A factorization of w is a decomposition $w = f_1 \cdots f_\ell$ into factors $f_1, \dots, f_\ell$. For words $w_1, \dots, w_n \in \Sigma^*$, we further denote by $\prod_{i=j}^{n} w_i$ the word $w_j w_{j+1} \cdots w_n$ if $j \le n$ and ε otherwise.

A *straight-line program*, briefly SLP, is a context-free grammar that produces a single word $w \in \Sigma^+$. Formally, it is a tuple $\mathbb{A} = (N, \Sigma, P, S)$, where N is a finite set of nonterminals with $N \cap \Sigma = \emptyset$, $S \in N$ is the start nonterminal, and P is a finite set of productions (or rules) of the form $A \to w$ for $A \in N$, $w \in (N \cup \Sigma)^+$ such that: (i) For every $A \in N$, there exists exactly one production of the form $A \to w$, and (ii) the binary relation $\{(A,B) \in N \times N \mid (A \to w) \in P,\ B \text{ occurs in } w\}$ is acyclic. Every nonterminal $A \in N$ produces a unique string $\mathrm{val}_{\mathbb{A}}(A) \in \Sigma^+$. The string defined by $\mathbb{A}$ is $\mathrm{val}(\mathbb{A}) = \mathrm{val}_{\mathbb{A}}(S)$. We omit the subscript $\mathbb{A}$ when it is clear from the context. The *size* of the SLP $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A \to w) \in P} |w|$. We denote by $g(w)$ the size of a smallest SLP producing the word $w \in \Sigma^+$. It is easy to see that $g(w) \le |w|$ since for each word w there is a trivial SLP with the only rule $S \to w$. We will use the following lemma which summarizes known results about SLPs.

Lemma II.1. *Let Σ be a finite alphabet of size σ.*

1) *For every word $w \in \Sigma^+$ of length n, there exists an SLP $\mathbb{A}$ of size $\mathcal{O}\big(\frac{n}{\log_\sigma n}\big)$ such that $\mathrm{val}(\mathbb{A}) = w$.*
2) *For an SLP $\mathbb{A}$ and a number $n > 0$, there exists an SLP $\mathbb{B}$ of size $|\mathbb{A}| + \mathcal{O}(\log n)$ such that $\mathrm{val}(\mathbb{B}) = \mathrm{val}(\mathbb{A})^n$.*
3) *For SLPs $\mathbb{A}_1$ and $\mathbb{A}_2$ there exists an SLP $\mathbb{B}$ of size $|\mathbb{A}_1| + |\mathbb{A}_2|$ such that $\mathrm{val}(\mathbb{B}) = \mathrm{val}(\mathbb{A}_1)\mathrm{val}(\mathbb{A}_2)$.*
4) *For words $w_1, \dots, w_n \in \Sigma^*$, $u \in \Sigma^+$ and SLPs $\mathbb{A}_1, \mathbb{A}_2$ with $\mathrm{val}(\mathbb{A}_1) = u$ and $\mathrm{val}(\mathbb{A}_2) = w_1 x w_2 x \cdots w_{n-1} x w_n$ for a symbol $x \notin \Sigma$, there exists an SLP $\mathbb{B}$ of size $|\mathbb{A}_1| + |\mathbb{A}_2|$ such that $\mathrm{val}(\mathbb{B}) = w_1 u w_2 u \cdots w_{n-1} u w_n$.*
5) *A string $w \in \Sigma^*$ contains at most $g(w) \cdot k$ distinct factors of length k.*

Statement 1 can be found for instance in [4]. Statements 2, 3 and 5 are shown in [19]. The proof of 4 is straightforward: Simply replace in the SLP $\mathbb{A}_2$ every occurrence of the terminal x by the start nonterminal of $\mathbb{A}_1$ and add all rules of $\mathbb{A}_1$ to $\mathbb{A}_2$.

The maximal size of a smallest SLP for all words of length n over an alphabet of size k is

$$g(k,n) = \max\{g(w) \mid w \in [1,k]^n\},$$

where $[1,k] = \{1, \dots, k\}$. By point 1 of Lemma II.1 we have $g(k,n) \in \mathcal{O}(n/\log_k n)$. In fact, Berstel and Brlek proved in [4] that $g(k,n) \in \Theta(n/\log_k n)$. As a first minor result, we show that there are words of length $2k^2+2k+1$ over an alphabet of size k for which the size of a smallest SLP equals the word length. Additionally, we show that all longer words have strictly smaller SLPs. Together this yield the following proposition:

Proposition II.2. *Let $n_k = 2k^2+2k+1$ for $k > 0$. Then (i) $g(k,n) < n$ for $n > n_k$ and (ii) $g(k,n) = n$ for $n \le n_k$.*

Proof: Let $\Sigma_k = \{a_1, \dots, a_k\}$ and let $M_{n,\ell} \subseteq \Sigma_k^*$ be the set of all words w where a factor v of length ℓ occurs at least n times without overlap. It is easy to see that $g(w) < |w|$ if and only if $w \in M_{3,2} \cup M_{2,3}$. Hence, we have to show that every word $w \notin M_{3,2} \cup M_{2,3}$ has length at most $2k^2+2k+1$. Moreover, we present words $w_k \in {\Sigma_k}^*$ of length $2k^2+2k+1$ such that $w_k \notin M_{3,2} \cup M_{2,3}$.

Let $w \notin M_{3,2} \cup M_{2,3}$. Consider a factor $a_i a_j$ of length two. If $i \ne j$ then this factor does not overlap itself, and thus $a_i a_j$ occurs at most twice in w. Now consider $a_i a_i$. Then w contains at most four (possibly overlapping) occurrence of $a_i a_i$, because five occurrences of $a_i a_i$ would yield at least three non-overlapping occurrences of $a_i a_i$. It follows that w has at most $2(k^2-k)+4k$ positions where a factor of length 2 starts, which implies $|w| \le 2k^2+2k+1$.

Now we create a word $w_k \notin M_{3,2} \cup M_{2,3}$ which realizes the above maximal occurrences of factors of length 2:

$$w_k = \left(\prod_{i=1}^{k} a_{k-i+1}^5\right) \prod_{i=1}^{k-1} \left(\prod_{j=i+2}^{k} (a_j a_i)^2\right) a_{i+1} a_i a_{i+1}$$

For example we have $w_3 = a_3^5 a_2^5 a_1^5 (a_3 a_1)^2 a_2 a_1 a_2 a_3 a_2 a_3$. One can check that $|w_k| = 2k^2+2k+1$ and $w_k \notin M_{3,2} \cup M_{2,3}$. ∎

III. Approximation ratio

As mentioned in the introduction, there is no polynomial time algorithm that computes a smallest SLP for a given word, unless P = NP [19], [22]. This result motivates approximation algorithms which are called *grammar-based compressors*.

A grammar-based compressor $\mathcal{C}$ computes for a word w an SLP $\mathcal{C}(w)$ such that $\mathrm{val}(\mathcal{C}(w)) = w$. The *approximation ratio* $\alpha_{\mathcal{C}}(w)$ of $\mathcal{C}$ for an input w is defined as $|\mathcal{C}(w)|/g(w)$. The worst-case approximation ratio $\alpha_{\mathcal{C}}(k,n)$ of $\mathcal{C}$ is the maximal approximation ratio over all words of length n over an alphabet of size k:

$$\begin{aligned}\alpha_{\mathcal{C}}(k,n) &= \max\{\alpha_{\mathcal{C}}(w) \mid w \in [1,k]^n\}\\ &= \max\{|\mathcal{C}(w)|/g(w) \mid w \in [1,k]^n\}\end{aligned}$$

In this definition, k might depend on n. Of course we must have $k \leq n$ and we write $\alpha_{\mathcal{C}}(n)$ instead of $\alpha_{\mathcal{C}}(n,n)$. This corresponds to the case where there is no restriction on the alphabet at all and it is the definition of the worst-case approximation ratio in [19]. The grammar-based compressors studied in our work are BISECTION [6], LZ78 [3] and RePair [8]. We will abbreviate the approximation ratio of BISECTION by α_{BI}. The families of words which we will use to improve the lower bounds of $\alpha_{\mathsf{BI}}(n)$ and $\alpha_{\mathsf{LZ78}}(n)$ are inspired by the constructions in [19].

A. BISECTION

The BISECTION algorithm [6] first splits an input word w with $|w| \geq 2$ as $w = w_1w_2$ such that $|w_1| = 2^j$ for the unique number $j \geq 0$ with $2^j < |w| \leq 2^{j+1}$. This process is recursively repeated with w_1 and w_2 until we obtain words of length 1. During the process, we introduce a nonterminal for each distinct factor of length at least two and create a rule with two symbols on the right-hand side corresponding to the split. Note that if $w = u_1u_2\cdots u_k$ with $|u_i| = 2^n$ for all $i, 1 \leq i \leq k$, then the SLP produced by BISECTION contains a nonterminal for each distinct word u_i ($1 \leq i \leq k$).

Example III.1. BISECTION *constructs an SLP for* $w = ababbbaabbaaab$ *as follows:*

- $w = w_1w_2$ *with* $w_1 = ababbbaa$, $w_2 = bbaaab$
  Introduced rule: $S \to W_1W_2$
- $w_1 = x_1x_2$ *with* $x_1 = abab$, $x_2 = bbaa$, *and* $w_2 = x_2x_3$ *with* $x_3 = ab$
  Introduced rules: $W_1 \to X_1X_2$, $W_2 \to X_2X_3$, $X_3 \to ab$
- $x_1 = x_3x_3$, $x_2 = y_1y_2$ *with* $y_1 = bb$ *and* $y_2 = aa$
  Introduced rules: $X_1 \to X_3X_3$, $X_2 \to Y_1Y_2$, $Y_1 \to bb$, $Y_2 \to aa$

BISECTION performs asymptotically optimal on unary words a^n since it produces an SLP of size $\mathcal{O}(\log n)$. Therefore $\alpha_{\mathsf{BI}}(1,n) \in \Theta(1)$. The following bounds on the approximation ratio for alphabets of size at least two are proven in [19, Thm. 5 and 6]:

$$\alpha_{\mathsf{BI}}(2,n) \in \Omega(\sqrt{n}/\log n) \tag{1}$$

$$\alpha_{\mathsf{BI}}(n) \in \mathcal{O}(\sqrt{n/\log n}) \tag{2}$$

We improve the lower bound (1) so that it matches the upper bound (2):

Theorem III.2. *For every* $k, 2 \leq k \leq n$ *we have* $\alpha_{\mathsf{BI}}(k,n) \in \Theta(\sqrt{n/\log n})$.

Proof: The upper bound (2) implies that $\alpha_{\mathsf{BI}}(k,n) \in \mathcal{O}(\sqrt{n/\log n})$ for all $k, 2 \leq k \leq n$. So it suffices to show $\alpha_{\mathsf{BI}}(2,n) \in \Omega(\sqrt{n/\log n})$. We first show that $\alpha_{\mathsf{BI}}(3,n) \in \Omega(\sqrt{n/\log n})$. In a second step, we encode a ternary alphabet into a binary alphabet while preserving the approximation ratio.

For every $k \geq 2$ let $\mathrm{bin}_k\colon \{0,1,\dots,k-1\} \to \{0,1\}^{\lceil \log_2 k \rceil}$ be the function where $\mathrm{bin}_k(j)$ $(0 \leq j \leq k-1)$ is the binary representation of j padded with leading zeros (e.g. $\mathrm{bin}_9(3) = 0011$). We further define for every $k \geq 2$ the word

$$u_k = \left(\prod_{j=0}^{k-2} \mathrm{bin}_k(j) a^{m_k}\right) \mathrm{bin}_k(k-1),$$

where $m_k = 2^{k-\lceil \log_2 k \rceil} - \lceil \log_2 k \rceil$. For instance $k = 4$ leads to $m_k = 2$ and $u_4 = 00aa01aa10aa11$. We analyze the approximation ratio $\alpha_{\mathsf{BI}}(s_k)$ for the word

$$s_k = \left(u_k a^{m_k+1}\right)^{m_k} u_k.$$

Claim 1. *The SLP produced by* BISECTION *on input* s_k *has size* $\Omega(2^k)$.

Proof. If s_k is split into non-overlapping factors of length $m_k + \lceil \log_2 k \rceil = 2^{k-\lceil \log_2 k \rceil}$, then the resulting set F_k of factors is

$$F_k = \{a^i \mathrm{bin}_k(j) a^{m_k-i} \mid 0 \leq j \leq k-1,\ 0 \leq i \leq m_k\}.$$

For example s_4 consecutively consists of the factors $00aa$, $01aa$, $10aa$, $11aa$, $a00a$, $a01a$, $a10a$, $a11a$, $aa00$, $aa01$, $aa10$ and $aa11$. The size of F_k is $(m_k+1) \cdot k \in \Theta(2^k)$, because all factors are pairwise different and $m_k \in \Theta(2^k/k)$. It follows that the SLP produced by BISECTION on input s_k has size $\Omega(2^k)$, because the length of each factor in F_k is a power of two and thus BISECTION creates a nonterminal for each distinct factor in F_k. (*end proof of Claim 1*)

Claim 2. *A smallest SLP producing* s_k *has size* $\mathcal{O}(k)$.

Proof. There is an SLP of size $\mathcal{O}(\log m_k) = \mathcal{O}(k)$ for the word a^{m_k} by Lemma II.1 (point 2). This yields an SLP for u_k of size $\mathcal{O}(k) + g(u_k')$ by Lemma II.1 (point 4), where $u_k' = (\prod_{i=0}^{k-2} \mathrm{bin}_k(i)x)\mathrm{bin}_k(k-1)$ is obtained from u_k by replacing all occurrences of a^{m_k} by a fresh symbol x. The word u_k' has length $\Theta(k \log k)$. Applying point 1 of Lemma II.1 (note that u_k' is a word over a ternary alphabet) it follows that

$$\begin{aligned} g(u_k') &\in \mathcal{O}\left(\frac{k \log k}{\log(k \log k)}\right)\\ &= \mathcal{O}\left(\frac{k \log k}{\log k + \log\log k}\right)\\ &= \mathcal{O}(k). \end{aligned}$$

Hence $g(u_k) \in \mathcal{O}(k)$. Finally, the SLP of size $\mathcal{O}(k)$ for u_k yields an SLP of size $\mathcal{O}(k)$ for s_k again using Lemma II.1 (points 2 and 3). (*end proof of Claim 2*)

In conclusion: We showed that a smallest SLP for s_k has size $\mathcal{O}(k)$, while BISECTION produces an SLP of size $\Omega(2^k)$. This implies $\alpha_{\mathsf{BI}}(s_k) \in \Omega(2^k/k)$. Let $n = |s_k|$. Since s_k is the concatenation of $\Theta(2^k)$ factors of length $\Theta(2^k/k)$, we have $n \in \Theta(2^{2k}/k)$ and thus $\sqrt{n} \in \Theta(2^k/\sqrt{k})$. This yields

$\alpha_{\mathsf{BI}}(s_k) \in \Omega(\sqrt{n/k})$. Together with $k \in \Theta(\log n)$ we obtain $\alpha_{\mathsf{BI}}(3, n) \in \Omega(\sqrt{n/\log n})$.

Let us now encode words over $\{0, 1, a\}$ into words over $\{0, 1\}$. Consider the homomorphism $f\colon \{0, 1, a\}^* \to \{0, 1\}^*$ with $f(0) = 00$, $f(1) = 01$ and $f(a) = 10$. Then we can prove the same approximation ratio of BISECTION for the input $f(s_k) \in \{0, 1\}^*$ that we proved for s_k above: The size of a smallest SLP for $f(s_k)$ is at most twice as large as the size of a smallest SLP for s_k, because an SLP for s_k can be transformed into an SLP for $f(s_k)$ by replacing every occurrence of a symbol $x \in \{0, 1, a\}$ by $f(x)$. Moreover, if we split $f(s_k)$ into non-overlapping factors of twice the length as we considered for s_k, then we obtain the factors from $f(F_k)$, whose length is again a power of two. Since f is injective, we have $|f(F_k)| = |F_k| \in \Theta(2^k)$. ∎

B. LZ78

The LZ78 algorithm on input $w \in \Sigma^+$ implicitly creates a list of words $f_1, \ldots, f_\ell$ (which we call the LZ78-*factorization*) with $w = f_1 \cdots f_\ell$ such that the following properties hold, where we set $f_0 = \varepsilon$:

- $f_i \neq f_j$ for all $i, j, 0 \leq i, j \leq \ell - 1$, with $i \neq j$.
- For all $i, 1 \leq i \leq \ell - 1$, there exist $j, 0 \leq j < i$ and $a \in \Sigma$ such that $f_i = f_j a$.
- $f_\ell = f_i$ for some $0 \leq i \leq \ell - 1$.

Note that the LZ78-factorization is unique for each word w. To compute it, the LZ78 algorithm needs ℓ steps performed by a single left-to-right pass. In the k^{th} step ($1 \leq k \leq \ell - 1$) it chooses the factor f_k as the shortest prefix of the unprocessed suffix $f_k \cdots f_\ell$ such that $f_k \neq f_i$ for all $i < k$. If there is no such prefix, then the end of w is reached and the algorithm sets f_ℓ to the (possibly empty) unprocessed suffix of w.

The factorization $f_1, \ldots, f_\ell$ yields an SLP for w of size at most 3ℓ as described in the following example:

Example III.3. *The* LZ78*-factorization of*

$$w = aabaaababababaa$$

is a, ab, aa, aba, b, abab, aa and leads to an SLP with the following rules:

- $S \to F_1 F_2 F_3 F_4 F_5 F_6 F_3$
- $F_1 \to a,\ F_2 \to F_1 b,\ F_3 \to F_1 a,\ F_4 \to F_2 a,\ F_5 \to b,\ F_6 \to F_4 b$

We have a nonterminal F_i for each factor f_i ($1 \leq i \leq 6$) such that $\mathrm{val}_{\mathbb{A}}(F_i) = f_i$. The last factor aa is represented in the start rule by the nonterminal F_3.

The LZ78-factorization of a^n ($n > 0$) is $a^1, a^2, \ldots, a^m, a^k$, where $k \in \{0, \ldots, m\}$ such that $n = k + \sum_{i=1}^m i$. Note that $m \in \Theta(\sqrt{n})$ and thus $\alpha_{\mathsf{LZ78}}(1, n) \in \Theta(\sqrt{n}/\log n)$. The following bounds for the worst-case approximation ratio of LZ78 were shown in [19, Thm. 3 and 4]:

$$\alpha_{\mathsf{LZ78}}(2, n) \in \Omega(n^{2/3}/\log n) \tag{3}$$

$$\alpha_{\mathsf{LZ78}}(n) \in \mathcal{O}((n/\log n)^{2/3}) \tag{4}$$

We will improve the lower bound so that it matches the upper bound in (4).

Theorem III.4. *For every k with $2 \leq k \leq n$ we have $\alpha_{\mathsf{LZ78}}(k, n) \in \Theta((n/\log n)^{2/3})$.*

Proof: Due to (4) it suffices to show $\alpha_{\mathsf{LZ78}}(2, n) \in \Omega((n/\log n)^{2/3})$. For $k \geq 2$ and $m \geq 1$, let $u_{m,k} = ((a^k b^{(2m+1)} a)^m (a^k b^{(m+1)})^2)^k a^k$ and $v_{m,k} = \left(\prod_{i=1}^m b^i a^k\right)^{k^2}$. We now analyze the approximation ratio of LZ78 on the words

$$s_{m,k} = a^{\frac{k(k+1)}{2}} b^{m(2m+1)}\ u_{m,k}\ v_{m,k}.$$

For example we have $u_{2,4} = ((a^4 b^5 a)^2 (a^4 b^3)^2)^4 a^4$, $v_{2,4} = (ba^4 b^2 a^4)^{16}$ and $s_{2,4} = a^{10}\, b^{10}\, u_{2,4}\, v_{2,4}$.

Claim 3. *The SLP produced by* LZ78 *on input $s_{m,k}$ has size $\Theta(k^2 m)$.*

Proof. We consider the LZ78-factorization $f_1, \ldots, f_\ell$ of $s_{m,k}$. Example III.5 gives a complete example. The prefix $a^{k(k+1)/2}$ produces the factors $f_i = a^i$ for every $i, 1 \leq i \leq k$ and the substring $b^{m(2m+1)}$ produces the factors $f_{k+i} = b^i$ for every $i, 1 \leq i \leq 2m$.

We next show that the substring $u_{m,k}$ then produces all factors from

$$\begin{aligned}\bigcup_{j=1}^{k} (\{a^{k-j+1} b^{i+1}, b^{2m-i} a^j \mid 0 \leq i \leq m-1\} \\ \cup \{a^{k-j+1} b^{m+1}, a^k b^{m+1} a^j\}).\end{aligned} \tag{5}$$

Let

$$\begin{aligned} u_{m,k,j} &= a^{k-j+1} b^{2m+1} a (a^k b^{2m+1} a)^{m-1} (a^k b^{m+1})^2 a^j \\ &= (a^{k-j+1} b^{2m+1} a^j)^m a^{k-j+1} b^{m+1} a^k b^{m+1} a^j. \end{aligned}$$

Then, $u_{m,k} = u_{m,k,1} \cdots u_{m,k,k}$. We show that each $u_{m,k,j}$ produces the factors from

$$\begin{aligned}\{a^{k-j+1} b^{i+1}, b^{2m-i} a^j \mid 0 \leq i \leq m-1\} \\ \cup \{a^{k-j+1} b^{m+1}, a^k b^{m+1} a^j\},\end{aligned} \tag{6}$$

for each $j, 1 \leq j \leq k$, thus obtaining (5).

Consider

$$u_{m,k,1} = (a^k b^{2m+1} a)^m a^k b^{m+1} a^k b^{m+1} a. \tag{7}$$

From the factorization of the prefix $a^{\frac{k(k+1)}{2}} b^{m(2m+1)}$ of $s_{m,k}$, the first $a^k b^{2m+1} a$ is factorized into $a^k b$ and $b^{2m} a$. Next, for each of the following $a^k b^{2m+1} a$, we can see that the new factors are $a^k b^2$ and $b^{2m-1} a$, $a^k b^3$ and $b^{2m-2} a, \ldots, a^k b^m$ and $b^{m+1} a$. Finally, the remaining $a^k b^{m+1} a^k b^{m+1} a$ is factorized to $a^k b^{m+1}$ and $a^k b^{m+1} a$. Therefore, (6) gives the factors of $u_{m,k,j}$ for $j = 1$.

Next, suppose that $u_{m,k,j'}$ produces the factors shown in (6) for all $1 < j' < j$, and consider $u_{m,k,j}$. By the induction hypothesis, we see that $a^{k-j+1} b$ and $b^{2m} a^j$ are the first two factors. Similarly, we see that each of the following $a^{k-j+1} b^{2m+1} a^j$ is factorized to $a^{k-j+1} b^2$ and $b^{2m-1} a^j$, $a^{k-j+1} b^3$ and $b^{2m-2} a^j, \ldots, a^{k-j+1} b^m$ and $b^{m+1} a^j$. Finally, the remaining suffix $a^{k-j+1} b^{m+1} a^k b^{m+1} a^j$ is factorized to $a^{k-j+1} b^{m+1}$ and $a^k b^{m+1} a^j$. It follows that the factorization of $u_{m,k}$ yields the factors shown in (5).

Next, we will show that the remaining suffix $v_{m,k}$ of $s_{m,k}$ produces the set of factors

$$\left\{a^i b^p a^j \mid 0 \leq i \leq k-1,\ 1 \leq j \leq k,\ 1 \leq p \leq m\right\}.$$

Observe that from the factors produced so far, only the factors $a^j b^i$ for $0 \le j \le k, 0 \le i \le m$ can be used for the factorization of $v_{m,k}$. The reason for this is that all other factors contain an occurrence of b^{m+1}, which does not occur in $v_{m,k}$.

Let $x = k+2m+k(2m+2)$ and note that this is the number of factors that we have produced so far. The factorization of $v_{m,k}$ in $s_{m,k}$ slightly differs when m is even, resp., odd. We now assume that m is even and explain the difference to the other case afterwards. The first factor of $v_{m,k}$ in $s_{m,k}$ is $f_{x+1} = ba$. We already have produced the factors $a^{k-1}b^i$ for every $i, 1 \le i \le m$, and hence $f_{x+i} = a^{k-1}b^i a$ for every $i, 2 \le i \le m$ and $f_{x+m+1} = a^{k-1}ba$. The next m factors are $f_{x+m+i} = a^{k-1}b^i a^2$ if i is even, $f_{x+m+i} = a^{k-2}b^i a$ if i is odd $(2 \le i \le m)$ and $f_{x+2m+1} = a^{k-2}ba$. This pattern continues: The next m factors are $f_{x+2m+i} = a^{k-1}b^i a^3$ if i is even, $f_{x+2m+i} = a^{k-3}b^i a$ if i is odd $(2 \le i \le m)$ and $f_{x+3m+1} = a^{k-3}ba$ and so on. Hence, we get the following sets of factors for $(\prod_{i=1}^m b^i a^k)^k$:

(i) $\{a^{k-i}b^p a \mid 1 \le i \le k,\ 1 \le p \le m,\ p \text{ is odd}\}$ for $f_{x+1}, f_{x+3} \dots, f_{x+km-1}$
(ii) $\{a^{k-1}b^p a^j \mid 1 \le j \le k,\ 1 \le p \le m,\ p \text{ is even}\}$ for $f_{x+2}, f_{x+4}, \dots, f_{x+km}$

The remaining word then starts with the factor $f_{y+1} = ba^2$, where $y = x + km$. Now the former pattern can be adapted to the next k repetitions of $\prod_{i=1}^m b^i a^k$ which gives us

(i) $\{a^{k-i}b^p a^2 \mid 1 \le i \le k,\ 1 \le p \le m,\ p \text{ is odd}\}$ for $f_{y+1}, f_{y+3} \dots, f_{y+km-1}$
(ii) $\{a^{k-2}b^p a^j \mid 1 \le j \le k,\ 1 \le p \le m,\ p \text{ is even}\}$ for $f_{y+2}, f_{y+4}, \dots, f_{y+km}$

The iteration of this process then reveals the whole pattern and thus yields the claimed factorization of $v_{m,k}$ in $s_{m,k}$ into factors $a^i b^p a^j$ for every $i, 0 \le i \le k-1$, $j, 1 \le j \le k$ and $p, 1 \le p \le m$. If m is odd then the patterns in (i) and (ii) switch after each occurrence of $\prod_{i=1}^m b^i a^k$, which does not affect the result but makes the pattern slightly more complicated. But the case that m is even suffices in order to derive the lower bound from the theorem.

We conclude that there are exactly $k+2m+k(2m+2)+k^2m$ factors (ignoring $f_\ell = \varepsilon$) and hence the SLP produced by LZ78 on input $s_{m,k}$ has size $\Theta(k^2m)$. *(end proof of Claim 3)*

Claim 4. *A smallest SLP producing $s_{m,k}$ has size $\mathcal{O}(\log k + m)$.*

Proof. We will combine the points stated in Lemma II.1 to prove this claim. Points 2 and 3 yield an SLP of size $\mathcal{O}(\log k + \log m)$ for the prefix $a^{k(k+1)/2}\ b^{m(2m+1)}\ u_{m,k}$ of $s_{m,k}$. To bound the size of an SLP for $v_{m,k}$ note at first that there is an SLP of size $\mathcal{O}(\log k)$ producing a^k by point 2 of Lemma II.1. Applying point 4 and again point 2, it follows that there is an SLP of size $\mathcal{O}(\log k) + g(v'_{m,k})$ producing $v_{m,k}$, where $v'_{m,k} = \prod_{i=1}^m b^i x$ for some fresh letter x. To get a small SLP for $v'_{m,k}$, we can introduce m nonterminals $B_1, \dots, B_m$ producing $b^1, \dots, b^m$ by adding rules $B_1 \to b$ and $B_{i+1} \to B_i b$ $(1 \le i \le m-1)$. This is enough to get an SLP of size $\mathcal{O}(m)$ for $v'_{m,k}$ and therefore an SLP of size $\mathcal{O}(\log k + m)$ for $v_{m,k}$. Together with our first observation and point 3 of Lemma II.1 this yields an SLP of size $\mathcal{O}(\log k + m)$ for $s_{m,k}$. *(end proof of Claim 4)*

Claims 3 and 4 imply $\alpha_{\mathsf{LZ78}}(s_{m,k}) \in \Omega(k^2m/(\log k + m))$. Let us now fix $m = \lceil \log k \rceil$. We get $\alpha_{\mathsf{LZ78}}(s_{m,k}) \in \Omega(k^2)$. Moreover, for the length $n = |s_{m,k}|$ of $s_{m,k}$ we have $n \in \Theta(k^3m + k^2m^2) = \Theta(k^3 \log k)$. We get $\alpha_{\mathsf{LZ78}}(s_{m,k}) \in \Omega((n/\log k)^{2/3})$ which together with $\log n \in \Theta(\log k)$ finishes the proof. ∎

Example III.5. *Here is the complete* LZ78 *factorization of*

$$s_{2,4} = a^{10}b^{10}\underbrace{((a^4b^5a)^2(a^4b^3)^2)^4a^4}_{u_{2,4}}\underbrace{(ba^4b^2a^4)^{16}}_{v_{2,4}}.$$

Factors of a^{10}: $a,\ a^2,\ a^3,\ a^4$

Factors of b^{10}: $b,\ b^2,\ b^3,\ b^4$

Factors of $u_{2,4}$:

a^4b	b^4a	a^4b^2	b^3a	a^4b^3	a^4b^3a
a^3b	b^4a^2	a^3b^2	b^3a^2	a^3b^3	$a^4b^3a^2$
a^2b	b^4a^3	a^2b^2	b^3a^3	a^2b^3	$a^4b^3a^3$
ab	b^4a^4	ab^2	b^3a^4	ab^3	$a^4b^3a^4$

Factors of $v_{2,4}$:

ba	a^3b^2a
a^3ba	$a^3b^2a^2$
a^2ba	$a^3b^2a^3$
aba	$a^3b^2a^4$
ba^2	a^2b^2a
a^3ba^2	$a^2b^2a^2$
a^2ba^2	$a^2b^2a^3$
aba^2	$a^2b^2a^4$
ba^3	ab^2a
a^3ba^3	ab^2a^2
a^2ba^3	ab^2a^3
aba^3	ab^2a^4
ba^4	b^2a
a^3ba^4	b^2a^2
a^2ba^4	b^2a^3
aba^4	b^2a^4

C. RePair

For a given SLP $\mathbb{A} = (N, \Sigma, P, S)$, a word $\gamma \in (N \cup \Sigma)^+$ is called a *maximal string* of $\mathbb{A}$ if

- $|\gamma| \ge 2$,
- γ appears at least twice without overlap in the right-hand sides of $\mathbb{A}$,
- and no strictly longer word appears at least as many times on the right-hand sides of $\mathbb{A}$ without overlap.

A *global grammar-based compressor* starts on input w with the trivial SLP $\mathbb{A} = (\{S\}, \Sigma, \{S \to w\}, S)$. In each round, the algorithm selects a maximal string γ of $\mathbb{A}$ and updates $\mathbb{A}$ by replacing a largest set of pairwise non-overlapping occurrences of γ in $\mathbb{A}$ by a fresh nonterminal X. Additionally, the algorithm introduces the rule $X \to \gamma$. The algorithm stops when no maximal string occurs. The global grammar-based compressor RePair [8] selects in each round a most frequent maximal string. Note that the replacement is not unique, e.g. the word a^5 with the maximal string $\gamma = aa$ yields SLPs with rules $S \to XXa, X \to aa$ or $S \to XaX, X \to aa$

or $S \to aXX, X \to aa$. We assume the first variant in this paper, i.e. maximal strings are replaced from left to right.

The above description of RePair is taken from [19]. In most papers on RePair the algorithm works slightly different: It replaces in each step a digram (a string of length two) with the maximal number of pairwise non-overlapping occurrences in the right-hand sides. For example, for the string $w = abcabc$ this produces the SLP $S \to BB$, $B \to Ac$, $A \to ab$, whereas the RePair-variant from [19] produces the smaller SLP $S \to AA$, $A \to abc$.

The following lower and upper bounds on the approximation ratio of RePair were shown in [19]:

$$\alpha_{\mathsf{RePair}}(n) \in \Omega\left(\sqrt{\log n}\right) \tag{8}$$
$$\alpha_{\mathsf{RePair}}(2,n) \in \mathcal{O}\left((n/\log n)^{2/3}\right)$$

The proof of the lower bound (8) assumes an alphabet of unbounded size. To be more accurate, the authors construct for every k a word w_k of length $\Theta(\sqrt{k}2^k)$ over an alphabet of size $\Theta(k)$ such that $g(w) \in \mathcal{O}(k)$ and RePair produces a grammar of size $\Omega(k^{3/2})$ for w_k. We will improve this lower bound using only a binary alphabet. To do so, we first need to know how RePair compresses unary words.

Example III.6 (unary inputs)**.** RePair *produces on input* a^{27} *the SLP with rules* $X_1 \to aa$*,* $X_2 \to X_1X_1$*,* $X_3 \to X_2X_2$ *and* $S \to X_3X_3X_3X_1a$*, where* S *is the start nonterminal. For the input* a^{22} *only the start rule* $S \to X_3X_3X_2X_1$ *is different.*

In general, RePair creates on unary input a^m ($m \geq 4$) the rules $X_1 \to aa$, $X_i \to X_{i-1}X_{i-1}$ for $2 \leq i \leq \lfloor \log m \rfloor - 1$ and a start rule, which is strongly related to the binary representation of m since each nonterminal X_i produces the word a^{2^i}. To be more accurate, let $b_{\lfloor \log m \rfloor} b_{\lfloor \log m \rfloor -1} \cdots b_1 b_0$ be the binary representation of m and define the mappings f_i ($i \geq 0$) by:

- $f_0 \colon \{0,1\} \to \{a, \varepsilon\}$ with $f_0(1) = a$ and $f_0(0) = \varepsilon$,
- $f_i \colon \{0,1\} \to \{X_i, \varepsilon\}$ with $f_i(1) = X_i$ and $f_i(0) = \varepsilon$ for $i \geq 1$.

Then the start rule produced by RePair on input a^m is

$$\begin{aligned} S \to\ & X_{\lfloor \log m \rfloor -1} X_{\lfloor \log m \rfloor -1} f_{\lfloor \log m \rfloor -1}(b_{\lfloor \log m \rfloor -1}) \\ & \cdots f_1(b_1) f_0(b_0). \end{aligned}$$

This means that the symbol a only occurs in the start rule if $b_0 = 1$, and the nonterminal X_i ($1 \leq i \leq \lfloor \log m \rfloor - 2$) occurs in the start rule if and only if $b_i = 1$. Since RePair only replaces words with at least two occurrences, the most significant bit $b_{\lfloor \log m \rfloor} = 1$ is represented by $X_{\lfloor \log m \rfloor -1} X_{\lfloor \log m \rfloor -1}$. Note that for $1 \leq m \leq 3$, RePair produces the trivial SLP $S \to a^m$.

For the proof of the new lower bound, we use *de Bruijn sequences* [42]. A binary de Bruijn sequence of order n is a string $B_n \in \{0,1\}^*$ of length 2^n such that every string from $\{0,1\}^n$ is either a factor of B_n or a suffix of B_n concatenated with a prefix of B_n. Moreover, every word of length at least n occurs at most once as factor in B_n. As an example, the string 1100 is a de Bruijn sequence of order 2, since 11, 10 and 00 occur as factors and 01 occurs as a suffix concatenated with a prefix. Lemma II.1 (point 5) implies that every SLP for B_n has size $\Omega(2^n/n)$.

Theorem III.7. $\alpha_{\mathsf{RePair}}(2,n) \in \Omega\left(\log n / \log\log n\right)$

Proof: We start with a binary de Bruijn sequence $B_{\lceil \log k \rceil} \in \{0,1\}^*$ of length $2^{\lceil \log k \rceil}$. We have $k \leq |B_{\lceil \log k \rceil}| < 2k$. Since de Bruijn sequences are not unique, we fix a de Bruijn sequence which starts with 1 for the remaining proof. We define a homomorphism $h \colon \{0,1\}^* \to \{0,1\}^*$ by $h(0) = 01$ and $h(1) = 10$. The words w_k of length $2k$ are defined as

$$w_k = h(B_{\lceil \log k \rceil}[1:k]).$$

For example $k = 4$ and $B_2 = 1100$ yield $w_4 = 10100101$. We will analyze the approximation ratio of RePair for the binary words

$$\begin{aligned} s_k &= \prod_{i=1}^{k-1} \left(a^{w_k[1:k+i]} b\right) a^{w_k} \\ &= a^{w_k[1:k+1]} b a^{w_k[1:k+2]} b \cdots a^{w_k[1:2k-1]} b a^{w_k}, \end{aligned}$$

where the prefixes $w_k[1:k+i]$ for $1 \leq i \leq k$ are interpreted as integers given by their binary representations. For example we have $s_4 = a^{20}ba^{41}ba^{82}ba^{165}$.

Since $B_{\lceil \log k \rceil}[1] = w_k[1] = 1$, we have $2^{k+i-1} \leq \left|a^{w_k[1:k+i]}\right| \leq 2^{k+i} - 1$ for $1 \leq i \leq k$ and thus $|s_k| \in \Theta\left(4^k\right)$.

Claim 5. *A smallest SLP producing* s_k *has size* $\mathcal{O}(k)$*.*

Proof. There is an SLP $\mathbb{A}$ of size $\mathcal{O}(k)$ for the first a-block $a^{w_k[1:k+1]}$ of length $\Theta(2^k)$. Let A be the start nonterminal of $\mathbb{A}$. For the second a-block $a^{w_k[1:k+2]}$ we only need one additional rule: If $w_k[k+2] = 0$, then we can produce $a^{w_k[1:k+2]}$ by the fresh nonterminal B using the rule $B \to AA$. Otherwise, if $w_k[k+2] = 1$, then we use $B \to AAa$. The iteration of that process yields for each a-block only one additional rule of size at most 3. If we replace the a-blocks in s_k by nonterminals as described, then the resulting word has size $2k+1$ and hence $g(s_k) \in \mathcal{O}(k)$. (*end proof of Claim 5*)

Claim 6. *The SLP produced by* RePair *on input* s_k *has size* $\Omega(k^2/\log k)$*.*

Proof. On unary inputs of length m, the start rule produced by RePair is strongly related to the binary encoding of m as described above. On input s_k, the algorithm begins to produce a start rule which is similarly related to the binary words $w_k[1:k+i]$ for $1 \leq i \leq k$. Consider the SLP $\mathbb{G}$ which is produced by RePair after $(k-1)$ rounds on input s_k. We claim that up to this point RePair is not affected by the b's in s_k and therefore has introduced the rules $X_1 \to aa$ and $X_i \to X_{i-1}X_{i-1}$ for $2 \leq i \leq k-1$. If this is true, then the first a-block is modified in the start rule S after $k-1$ rounds as follows

$$X_{k-1}X_{k-1}f_{k-1}(w_k[2])f_{k-2}(w_k[3]) \cdots f_0(w_k[k+1])b$$

where $f_0(1) = a$, $f_0(0) = \varepsilon$ and $f_i(1) = X_i$, $f_i(0) = \varepsilon$ for $i \geq 1$. All other a-blocks are longer than the first one, hence each factor of the start rule which corresponds to an a-block begins with $X_{k-1}X_{k-1}$. Therefore, the number of occurrences of $X_{k-1}X_{k-1}$ in the SLP is at least k. Since the symbol b occurs only $k-1$ times in s_k, it follows that our assumption is correct and RePair is not affected by the b's in the first $(k-1)$ rounds on input s_k. Also, for each block $a^{w_k[1:k+i]}$, the $k-1$ least significant bits of $w_k[1:k+i]$ $(1 \le i \le k)$ are represented in the corresponding factor of the start rule of $\mathbb{G}$, i.e., the start rule contains non-overlapping factors v_i with

$$\begin{aligned} v_i = \; & f_{k-2}(w_k[i+2])f_{k-3}(w_k[i+3]) \\ & \cdots f_1(w_k[k+i-1])f_0(w_k[k+i]) \end{aligned} \tag{9}$$

for $1 \le i \le k$. For example after 3 rounds on input $s_4 = a^{20}ba^{41}ba^{82}ba^{165}$, we have the start rule

$$S \to \underbrace{X_3X_3X_2}_{a^{20}}\, b\, \underbrace{{X_3}^5a}_{a^{41}}\, b\, \underbrace{{X_3}^{10}X_1}_{a^{82}}\, b\, \underbrace{{X_3}^{20}X_2a}_{a^{165}},$$

where $v_1 = X_2$, $v_2 = a$, $v_3 = X_1$ and $v_4 = X_2a$. The length of the factor $v_i \in \{a, X_1, \dots, X_{k-2}\}^*$ from equation (9) is exactly the number of 1's in the word $w_k[i+2:k+i]$. Since w_k is constructed by the homomorphism h, it is easy to see that $|v_i| \ge (k-3)/2$. Note that no letter occurs more than once in v_i, hence $g(v_i) = |v_i|$. Further, each substring of length $2\lceil \log k \rceil + 2$ occurs at most once in $v_1, \dots, v_k$, because otherwise there would be a factor of length $\lceil \log k \rceil$ occurring more than once in $B_{\lceil \log k \rceil}$. It follows that there are at least

$$k \cdot (\lceil (k-3)/2 \rceil - 2\lceil \log k \rceil - 1) \in \Theta(k^2)$$

different factors of length $2\lceil \log k \rceil + 2 \in \Theta(\log k)$ in the right-hand side of the start rule of $\mathbb{G}$. By Lemma II.1 (point 5) it follows that a smallest SLP for the right-hand side of the start rule has size $\Omega(k^2/\log k)$ and therefore $|\mathsf{RePair}(s_k)| \in \Omega(k^2/\log k)$. (*end proof of Claim 6*)

In conclusion: We showed that a smallest SLP for s_k has size $\mathcal{O}(k)$, while RePair produces an SLP of size $\Omega(k^2/\log k)$. This implies $\alpha_{\mathsf{RePair}}(s_k) \in \Omega(k/\log k)$, which together with $n = |s_k|$ and $k \in \Theta(\log n)$ finishes the proof. ∎

Note that in the above proof, RePair chooses in the first $k-1$ rounds a digram for the replaced maximal string. Therefore, Theorem III.7 also holds for the RePair-variant, where in every round a digram (which is not necessarily a maximal string) is replaced.

IV. Hardness of grammar-based compression for binary alphabets

The goal of this section is to prove the following result:

Theorem IV.1. *Let $c \ge 1$ be a constant. If there exists a polynomial time grammar-based compressor $\mathcal{C}$ with $\alpha_{\mathcal{C}}(2,n) \le c$ then there exists a polynomial time grammar-based compressor $\mathcal{D}$ with $\alpha_{\mathcal{D}}(n) \le 6c$.*

For a factor $24 + \varepsilon$ (with $\varepsilon > 0$) instead of 6 this result was shown in [24] using a more complicated block encoding. Remember that even a factor of 6 does not imply that the smallest grammar problem for binary strings is NP-hard. One would have to further reduce this factor to at most $8569/8568$.

We split the proof of Theorem IV.1 into two lemmas that state translations between SLPs over arbitrary alphabets and SLPs over a binary alphabet. For the rest of this section fix the alphabets $\Sigma = \{c_0, \dots, c_{k-1}\}$ and $\Sigma_2 = \{a, b\}$. To translate between these two alphabets, we define an injective homomorphism $\varphi\colon \Sigma^* \to \Sigma_2^*$ by

$$\varphi(c_i) = a^ib \quad (0 \le i \le k-1). \tag{10}$$

Lemma IV.2. *Let $w \in \Sigma^*$ be such that every symbol from Σ occurs in w. From an SLP $\mathbb{A}$ for w one can construct in polynomial time an SLP $\mathbb{B}$ for $\varphi(w)$ of size at most $3 \cdot |\mathbb{A}|$.*

Proof: To translate $\mathbb{A}$ into an SLP $\mathbb{B}$ for $\varphi(w)$, we first add the productions $A_0 \to b$ and $A_i \to aA_{i-1}$ for every $i, 1 \le i \le k-1$. Finally, we replace in $\mathbb{A}$ every occurrence of $c_i \in \Sigma$ by A_i. This yields an SLP $\mathbb{B}$ for $\varphi(w)$ of size $|\mathbb{A}| + 2k - 1$. Because $k \le |\mathbb{A}|$ (since every symbol from Σ occurs in w), we obtain $|\mathbb{B}| \le 3 \cdot |\mathbb{A}|$. ∎

Lemma IV.3. *Let $w \in \Sigma^*$ such that every symbol from Σ occurs in w. From an SLP $\mathbb{B}$ for $\varphi(w)$ one can construct in polynomial time an SLP $\mathbb{A}$ for w of size at most $2 \cdot |\mathbb{B}|$.*

Proof: Let $\mathbb{B} = (N, \Sigma_2, P, S)$ be an SLP for $\varphi(w)$, where $w \in \Sigma^*$. We can assume that every right-hand side of $\mathbb{B}$ is a non-empty string. Consider a nonterminal $A \in N$ of $\mathbb{B}$. Since $\mathbb{B}$ produces $\varphi(w)$, A produces a factor of $\varphi(w)$, which is a word from $\{a,b\}^*$. We cannot directly translate $\mathrm{val}(A)$ back to a word over Σ^* because $\mathrm{val}(A)$ does not have to belong to the image of φ. But $\mathrm{val}(A)$ is a factor of a string from $\varphi(\Sigma^*)$. Note that a string over $\{a,b\}$ is a factor of a string from $\varphi(\Sigma^*)$ if and only if it does not contain a factor a^i with $i \ge k$. Let $\mathrm{val}(A) = a^{i_1}b \cdots a^{i_n}ba^{i_{n+1}}$ be such a string, where $n \ge 0$, and $0 \le i_1, \dots, i_{n+1} < k$. We factorize $\mathrm{val}(A)$ into three parts in the following way. If $n = 0$ (i.e., $\mathrm{val}(A) = a^{i_1}$) then we split $\mathrm{val}(A)$ into ε, ε, and a^{i_1}. If $n > 0$ then we split $\mathrm{val}(A)$ into $a^{i_1}b$, $a^{i_2}b \cdots a^{i_n}b$, and $a^{i_{n+1}}$. Let us explain the intuition behind this factorization. We concentrate on the case $n > 0$; the case $n = 0$ is simpler. Note that irrespective of the context in which an occurrence of $\mathrm{val}(A)$ appears in $\mathrm{val}(\mathbb{B})$, we can translate the middle part $a^{i_2}b \cdots a^{i_n}b$ into $c_{i_2} \cdots c_{i_n}$. We will therefore introduce in the SLP $\mathbb{A}$ for w a variable A' that produces $c_{i_2} \cdots c_{i_n}$. For the left part $a^{i_1}b$ we can not directly produce c_{i_1} because an occurrence of $\mathrm{val}(A)$ could be preceded by an a-block a^{i_0}, yielding the symbol $c_{i_0+i_1}$. Therefore, the algorithm that produces $\mathbb{A}$ will only memorize the symbol c_{i_1} without writing it directly on a right-hand side of an $\mathbb{A}$-production. Similarly, the algorithm will memorize the length i_{n+1} of the final a-block of $\mathrm{val}(A)$.

Let us now come to the formal details of the proof. As usual, we write $\mathbb{Z}_k$ for $\{0, 1, \dots, k-1\}$ and w.l.o.g. we assume that $\Sigma \cap \mathbb{Z}_k = \emptyset$. Consider a word $s = a^{i_1}b \cdots a^{i_n}ba^{i_{n+1}}$, where $n \ge 0$, and $0 \le i_1, \dots, i_{n+1} < k$. Motivated by the above discussion, we define $\ell(s) \in \Sigma \cup \{\varepsilon\}$, $m(s) \in \Sigma^*$ and $r(s) \in \mathbb{Z}_k$ as follows:

$$\ell(s) = \begin{cases} c_{i_1} & \text{if } n \geq 1, \\ \varepsilon & \text{if } n = 0, \end{cases}$$
$$m(s) = c_{i_2} \cdots c_{i_n},$$
$$r(s) = i_{n+1}.$$

Note that $\ell(s) = \varepsilon$ implies $m(s) = \varepsilon$. Finally, we define the word $\psi(s) \in \Sigma^* \mathbb{Z}_k$ as

$$\psi(s) = \ell(s) m(s) r(s).$$

For a nonterminal $A \in N$ we define $\ell(A) = \ell(\mathrm{val}(A))$, $m(A) = m(\mathrm{val}(A))$ and $r(A) = r(\mathrm{val}(A))$. We now define an SLP $\mathbb{A}'$ that contains for every nonterminal $A \in N$ a nonterminal A' such that $\mathrm{val}(A') = m(A)$. Moreover, the algorithm also computes $\ell(A) \in \Sigma \cup \{\varepsilon\}$ and $r(A) \in \mathbb{Z}_k$.

We define the productions of $\mathbb{A}'$ inductively over the structure of $\mathbb{B}$. Consider a production $(A \to \alpha) \in P$, where $\alpha = v_0 A_1 v_1 A_2 \cdots v_{n-1} A_n v_n \neq \varepsilon$ with $n \geq 0$, $A_1, \ldots, A_n \in N$, and $v_0, v_1, \ldots, v_n \in \Sigma_2^*$. Let $\ell_i = \ell(A_i) \in \Sigma \cup \{\varepsilon\}$ and $r_i = r(A_i) \in \mathbb{Z}_k$, which have already been computed. The right-hand side for A' is obtained as follows. We start with the word

$$\psi(v_0)\, \ell_1\, A_1'\, r_1\, \psi(v_1)\, \ell_2\, A_2'\, r_2 \cdots \psi(v_{n-1})\, \ell_n\, A_n'\, r_n\, \psi(v_n). \tag{11}$$

Note that each of the factors $\ell_i A_i' r_i$ produces (by induction) $\psi(\mathrm{val}(A_i))$. Next we remove every A_i' that derives the empty word (which is equivalent to $m(A_i) = \varepsilon$). After this step, every occurrence of a symbol $i \in \mathbb{Z}_k$ in (11) is either the last symbol of the above word or it is followed by a symbol from $\mathbb{Z}_k \cup \Sigma$ (but not followed by a nonterminal A_j'). To see this, recall that $\ell_j = \varepsilon$ implies $m(A_j) = \varepsilon$, in which case A_j' is removed in (11).

The above fact allows us to eliminate all occurrences of symbols $i \in \mathbb{Z}_k$ in (11) except for the last one using the two reduction rules $i\,j \to i + j$ for $i, j \in \mathbb{Z}_k$ (which corresponds to $a^i a^j = a^{i+j}$) and $i\, c_j \to c_{i+j}$ (which corresponds to $a^i a^j b = a^{i+j} b$). If we perform these rules as long as possible (the order of applications is not relevant since these rules form a confluent and terminating system), only a single occurrence of a symbol $i \in \mathbb{Z}_k$ at the end of the string will remain. The resulting string α' produces $\psi(A)$. Hence, we obtain the right-hand side for the nonterminal A' by removing the first symbol of α' if it is from Σ (this symbol is then $\ell(A)$) and the last symbol of α', which must be from $\mathbb{Z}_k$ (this symbol is $r(A)$). Note that if α' does not start with a symbol from Σ, then α' belongs to $\mathbb{Z}_k$, in which case we have $\ell(A) = \varepsilon$.

Note that $\psi(\varphi(w)) = w0$ for every $w \in \Sigma^*$, so for the start variable S of $\mathbb{B}$ we must have $r(S) = 0$, since $\mathrm{val}_{\mathbb{B}}(S) \in \varphi(\Sigma^*)$. Let $S' \to \sigma$ be the production for S' in $\mathbb{A}'$. We obtain the SLP $\mathbb{A}$ by replacing this production by $S' \to \ell(S)\sigma$. Since $\mathrm{val}_{\mathbb{A}'}(S') = m(S)$ and $\mathrm{val}_{\mathbb{B}}(S) = \varphi(w)$ we have $\mathrm{val}_{\mathbb{A}}(S') = \ell(S) m(S) = w$.

To bound the size of $\mathbb{A}$ consider the word in (11) from which the right-hand side of the nonterminal A' is computed. All occurrences of symbols from $\mathbb{Z}_k$ are eliminated when forming this right-hand side. This leaves a word of length at most $|\alpha| + n$ (where α is the original right-hand side of the nonterminal A). The additive term n comes from the symbols $\ell_1, \ldots, \ell_n$. Hence, $|\mathbb{A}'|$ is bounded by the size of $\mathbb{B}$ plus the total number of occurrences of nonterminals in right-hand sides of $\mathbb{B}$, which is at most $2|\mathbb{B}| - 1$ (there is at least one terminal occurrence in a right-hand side). Since $|\mathbb{A}| = |\mathbb{A}'| + 1$ we get $|\mathbb{A}| \leq 2|\mathbb{B}|$.

The algorithm's runtime for a production $A \to \alpha$ is linear in $|\alpha|$. This is because we start with the string (11) which can be computed in time $\mathcal{O}(|\alpha|)$. From this string, we remove all the A_i' that produce ε and we also apply the two rewriting rules. Both of these can be done in a single left-to-right sweep over the string. The number of operations needed is linear in $|\alpha|$, where each operation needs constant time, i.e. removing an A_i' takes constant time, and using one of the rewriting rules also takes constant time. Since the algorithm uses the structure of $\mathbb{B}$ to visit each of its productions once, we overall obtain a linear running time in the size of $\mathbb{B}$. ■

Example IV.4. *Consider the production*

$$A \to a^3 b a^5 A_1 a^3 A_2 a^2 b^2 A_3 a^2$$

and assume that $\mathrm{val}(A_1) = a^2$, $\mathrm{val}(A_2) = aba^3ba$ *and* $\mathrm{val}(A_3) = ba^2ba^3$. *Hence, when we produce the right-hand side for* A' *we have:* $\mathrm{val}(A_1') = \varepsilon$, $\mathrm{val}(A_2') = c_3$, $\mathrm{val}(A_3') = c_2$, $\ell_1 = \varepsilon$, $r_1 = 2$, $\ell_2 = c_1$, $r_2 = 1$, $\ell_3 = c_0$, $r_3 = 3$. *We start with the word (every digit is a single symbol)*

$$c_3\, 5\, A_1'\, 2\, 3\, c_1 A_2'\, 1\, c_2 c_0\, 0\, c_0 A_3'\, 3\, 2.$$

Then we replace A_1' *by* ε *and obtain*

$$c_3\, 5\, 2\, 3\, c_1 A_2'\, 1\, c_2 c_0\, 0\, c_0 A_3'\, 3\, 2.$$

Applying the reduction rules finally yields

$$c_3 c_{11} A_2' c_3 c_0 c_0 A_3' 5.$$

Hence, we have $\ell(A) = c_3$, $r(A) = 5$ *and the production for* A' *is* $A' \to c_{11} A_2' c_3 c_0 c_0 A_3'$.

Proof of Theorem IV.1: Let $\mathcal{C}$ be an arbitrary grammar-based compressor working in polynomial time such that $\alpha_{\mathcal{C}}(2, n) \leq c$. The grammar-based compressor $\mathcal{D}$ works for an input word w over an arbitrary alphabet as follows: Let $\Sigma = \{c_0, \ldots, c_{k-1}\}$ be the set of symbols that occur in w and let φ be defined as in (10). Using $\mathcal{C}$, one first computes an SLP $\mathbb{B}$ for $\varphi(w)$ such that $|\mathbb{B}| \leq c \cdot g(\varphi(w))$. Then, using Lemma IV.3, one computes from $\mathbb{B}$ an SLP $\mathbb{A}$ for w such that $|\mathbb{A}| \leq 2c \cdot g(\varphi(w))$. Lemma IV.2 implies $g(\varphi(w)) \leq 3 \cdot g(w)$ and hence $|\mathbb{A}| \leq 6c \cdot g(w)$, which proves the theorem. ■

V. Open problems

One should try to narrow the gaps between the lower and upper bounds for the other grammar-based compressors analyzed in [19]. In particular, the gap between the known lower and upper bounds for the so-called global algorithms from [19] (like RePair) is still quite big. Charikar et al. [19] prove an upper bound $\Theta((n/\log n)^{2/3})$ for every global algorithm and nothing better is known for the three global algorithms

RePair, LongestMatch, Greedy studied in [19]. Comparing to this upper bound, the known lower bounds are quite small: $\Omega(\log n/\log\log n)$ for RePair (by our Theorem III.7), $\Omega(\log\log n)$ for longest match [19], and $1.348\ldots$. The latter is a very recent result from [41].[1]

Another open research problem is improving the constant 6 in Theorem IV.1. Recall that lowering this constant to at most $8569/8568$ would imply that the smallest grammar problem for binary strings cannot be solved in polynomial time unless $\mathsf{P}=\mathsf{NP}$.

References

[1] D. Hucke, M. Lohrey, and C. P. Reh, "The smallest grammar problem revisited," in *String Processing and Information Retrieval - 23rd International Symposium, SPIRE 2016, Beppu, Japan, October 18-20, 2016, Proceedings*, ser. Lecture Notes in Computer Science, S. Inenaga, K. Sadakane, and T. Sakai, Eds., vol. 9954, 2016, pp. 35–49. [Online]. Available: https://doi.org/10.1007/978-3-319-46049-9_4

[2] F. Rubin, "Experiments in text file compression," *Commun. ACM*, vol. 19, no. 11, pp. 617–623, 1976. [Online]. Available: https://doi.org/10.1145/360363.360368

[3] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Information Theory*, vol. 24, no. 5, pp. 530–536, 1978. [Online]. Available: https://doi.org/10.1109/TIT.1978.1055934

[4] J. Berstel and S. Brlek, "On the length of word chains," *Inf. Process. Lett.*, vol. 26, no. 1, pp. 23–28, 1987. [Online]. Available: https://doi.org/10.1016/0020-0190(87)90031-7

[5] A. A. Diwan, "A new combinatorial complexity measure for languages," *Tata Institute, Bombay, India*, 1986.

[6] J. C. Kieffer, E. Yang, G. J. Nelson, and P. C. Cosman, "Universal lossless compression via multilevel pattern matching," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1227–1245, 2000. [Online]. Available: https://doi.org/10.1109/18.850665

[7] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *J. Artif. Intell. Res.*, vol. 7, pp. 67–82, 1997. [Online]. Available: https://doi.org/10.1613/jair.374

[8] N. J. Larsson and A. Moffat, "Offline dictionary-based compression," in *Data Compression Conference, DCC 1999, Snowbird, Utah, USA, March 29-31, 1999*. IEEE Computer Society, 1999, pp. 296–305. [Online]. Available: https://doi.org/10.1109/DCC.1999.755679

[9] J. C. Kieffer and E. Yang, "Grammar-based codes: A new class of universal lossless source codes," *IEEE Trans. Information Theory*, vol. 46, no. 3, pp. 737–754, 2000. [Online]. Available: https://doi.org/10.1109/18.841160

[10] ——, "Structured grammar-based codes for universal lossless data compression," *Communications in Information and Systems*, vol. 2, pp. 29–52, 07 2002.

[11] E. Yang and J. C. Kieffer, "Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform - part one: Without context models," *IEEE Trans. Information Theory*, vol. 46, no. 3, pp. 755–777, 2000. [Online]. Available: https://doi.org/10.1109/18.841161

[12] Y. Tabei, Y. Takabatake, and H. Sakamoto, "A succinct grammar compression," in *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, ser. Lecture Notes in Computer Science, J. Fischer and P. Sanders, Eds., vol. 7922. Springer, 2013, pp. 235–246. [Online]. Available: https://doi.org/10.1007/978-3-642-38905-4_23

[13] M. Gańczorz, "Entropy bounds for grammar compression," *CoRR*, vol. abs/1804.08547, 2018. [Online]. Available: http://arxiv.org/abs/1804.08547

[14] C. Ochoa and G. Navarro, "Repair and all irreducible grammars are upper bounded by high-order empirical entropy," *IEEE Trans. Information Theory*, vol. 65, no. 5, pp. 3160–3164, 2019. [Online]. Available: https://doi.org/10.1109/TIT.2018.2871452

[1]The table on page 2556 in [19] states the better lower bound of $1.37\ldots$, but the authors only show the lower bound $1.137\ldots$, see [19, Theorem 11].

[15] M. Gańczorz, "Entropy lower bounds for dictionary compression," in *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, ser. LIPIcs, N. Pisanti and S. P. Pissis, Eds., vol. 128. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 11:1–11:18. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2019.11

[16] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann, "Random access to grammar-compressed strings and trees," *SIAM J. Comput.*, vol. 44, no. 3, pp. 513–539, 2015. [Online]. Available: https://doi.org/10.1137/130936889

[17] M. Lohrey, "Algorithmics on slp-compressed strings: A survey," *Groups Complexity Cryptology*, vol. 4, no. 2, pp. 241–299, 2012. [Online]. Available: https://doi.org/10.1515/gcc-2012-0016

[18] D. Kempa and N. Prezza, "At the roots of dictionary compression: string attractors," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, I. Diakonikolas, D. Kempe, and M. Henzinger, Eds. ACM, 2018, pp. 827–840. [Online]. Available: https://doi.org/10.1145/3188745.3188814

[19] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The smallest grammar problem," *IEEE Trans. Information Theory*, vol. 51, no. 7, pp. 2554–2576, 2005. [Online]. Available: https://doi.org/10.1109/TIT.2005.850116

[20] S. R. Kosaraju and G. Manzini, "Compression of low entropy strings with Lempel-Ziv algorithms," *SIAM J. Comput.*, vol. 29, no. 3, pp. 893–911, 1999. [Online]. Available: https://doi.org/10.1137/S0097539797331105

[21] A. C. Yao, "On the evaluation of powers," *SIAM J. Comput.*, vol. 5, no. 1, pp. 100–103, 1976. [Online]. Available: https://doi.org/10.1137/0205008

[22] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *J. ACM*, vol. 29, no. 4, pp. 928–951, 1982. [Online]. Available: https://doi.org/10.1145/322344.322346

[23] K. Casel, H. Fernau, S. Gaspers, B. Gras, and M. L. Schmid, "On the complexity of grammar-based compression over fixed alphabets," in *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, ser. LIPIcs, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, Eds., vol. 55. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 122:1–122:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2016.122

[24] J. Arpe and R. Reischuk, "On the complexity of optimal grammar-based compression," in *2006 Data Compression Conference (DCC 2006), 28-30 March 2006, Snowbird, UT, USA*. IEEE Computer Society, 2006, pp. 173–182. [Online]. Available: https://doi.org/10.1109/DCC.2006.59

[25] A. Jeż, "Approximation of grammar-based compression via recompression," *Theor. Comput. Sci.*, vol. 592, pp. 115–134, 2015. [Online]. Available: https://doi.org/10.1016/j.tcs.2015.05.027

[26] ——, "A really simple approximation of smallest grammar," *Theor. Comput. Sci.*, vol. 616, pp. 141–150, 2016. [Online]. Available: https://doi.org/10.1016/j.tcs.2015.12.032

[27] W. Rytter, "Application of Lempel-Ziv factorization to the approximation of grammar-based compression," *Theor. Comput. Sci.*, vol. 302, no. 1-3, pp. 211–222, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00777-6

[28] H. Sakamoto, "A fully linear-time approximation algorithm for grammar-based compression," *J. Discrete Algorithms*, vol. 3, no. 2-4, pp. 416–430, 2005. [Online]. Available: https://doi.org/10.1016/j.jda.2004.08.016

[29] P. Bille, I. L. Gørtz, and N. Prezza, "Space-efficient Re-Pair compression," in *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2017, pp. 171–180. [Online]. Available: https://doi.org/10.1109/DCC.2017.24

[30] T. Gagie, T. I, G. Manzini, G. Navarro, H. Sakamoto, and Y. Takabatake, "Rpair: Rescaling RePair with Rsync," in *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, N. R. Brisaboa and S. J. Puglisi, Eds., vol. 11811. Springer, 2019, pp. 35–44. [Online]. Available: https://doi.org/10.1007/978-3-030-32686-9_3

[31] F. Claude and G. Navarro, "Fast and compact web graph representations," *TWEB*, vol. 4, no. 4, pp. 16:1–16:31, 2010. [Online]. Available: https://doi.org/10.1145/1841909.1841913

[32] R. Wan, "Browsing and searching compressed documents," Ph.D. dissertation, Dept. of Computer Science and Software Engineering, University of Melbourne, 2003.

[33] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa, "Collage system: a unifying framework for compressed pattern matching," *Theor. Comput. Sci.*, vol. 298, no. 1, pp. 253–272, 2003. [Online]. Available: https://doi.org/10.1016/S0304-3975(02)00426-7

[34] R. González and G. Navarro, "Compressed text indexes with fast locate," in *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, ser. Lecture Notes in Computer Science, B. Ma and K. Zhang, Eds., vol. 4580. Springer, 2007, pp. 216–227. [Online]. Available: https://doi.org/10.1007/978-3-540-73437-6_23

[35] M. Lohrey, S. Maneth, and R. Mennicke, "XML tree structure compression using RePair," *Inf. Syst.*, vol. 38, no. 8, pp. 1150–1167, 2013. [Online]. Available: https://doi.org/10.1016/j.is.2013.06.006

[36] I. Furuya, T. Takagi, Y. Nakashima, S. Inenaga, H. Bannai, and T. Kida, "MR-RePair: Grammar compression based on maximal repeats," in *Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26-29, 2019*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2019, pp. 508–517. [Online]. Available: https://doi.org/10.1109/DCC.2019.00059

[37] M. Gańczorz and A. Jeż, "Improvements on Re-Pair grammar compressor," in *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2017, pp. 181–190. [Online]. Available: https://doi.org/10.1109/DCC.2017.52

[38] T. Masaki and T. Kida, "Online grammar transformation based on Re-Pair algorithm," in *2016 Data Compression Conference, DCC 2016, Snowbird, UT, USA, March 30 - April 1, 2016*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2016, pp. 349–358. [Online]. Available: https://doi.org/10.1109/DCC.2016.69

[39] J. C. Kieffer, P. Flajolet, and E. Yang, "Universal lossless data compression via binary decision diagrams," *CoRR*, vol. abs/1111.1432, 2011. [Online]. Available: http://arxiv.org/abs/1111.1432

[40] A. Apostolico and S. Lonardi, "Some theory and practice of greedy off-line textual substitution," in *Data Compression Conference, DCC 1998, Snowbird, Utah, USA, March 30 - April 1, 1998*. IEEE Computer Society, 1998, pp. 119–128. [Online]. Available: https://doi.org/10.1109/DCC.1998.672138

[41] D. Hucke, "Approximation ratios of RePair, LongestMatch and Greedy on unary strings," in *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, N. R. Brisaboa and S. J. Puglisi, Eds., vol. 11811. Springer, 2019, pp. 3–15. [Online]. Available: https://doi.org/10.1007/978-3-030-32686-9_1

[42] N. G. de Bruijn, "A combinatorial problem," *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, vol. 49, no. 7, pp. 758–764, 1946.

[43] A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds., *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*. IEEE, 2017. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/7921793/proceeding

[44] N. R. Brisaboa and S. J. Puglisi, Eds., *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11811. Springer, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-32686-9

Hideo Bannai

Momoko Hirayama

Danny Hucke

Shunsuke Inenaga

Artur Jeż received his PhD from the University of Wrocław, Poland in 2010. From 2010 till 2018 he was an assistant professor at University of Wrocław, at temporal leave from 2012 till 2014, when he was Humboldt Postdoc Fellow at Max Planck Institute for Computer Science in Saarbrücken. From 2018 he is an associate professor at University of Wroclaw, Poland. His research interest are formal languages and automata theory, data compression and string algorithms.

Markus Lohrey received his PhD from the University of Stuttgart, Germany, in 1999. From 2007 till 2013 he was professor for algebraic and logical foundations at University of Leipzig, Germany. Since 2013 he is professor for theoretical computer science at the University of Siegen, Germany. His working areas are algorithmic algebra, automata theory, and data compression. He published more than 170 papers in international journals and conferences.

Carl Philipp Reh is a research associate in the Theoretical Computer Science group at the University of Siegen, Germany. He received his diploma in Informatics in 2013 and his PhD in 2019, both at the University of Siegen. His research focuses on algorithms on compressed data.