Approximation ratios of **RePair**, **LongestMatch** and **Greedy** on unary strings

Danny Hucke

University of Siegen, Germany, hucke@eti.uni-siegen.de

Abstract. A grammar-based compressor computes for a given input w a context-free grammar that produces only w. So-called global grammar-based compressors (RePair, LongestMatch and Greedy) achieve impressive practical compression results, but the recursive character of those algorithms makes it hard to achieve strong theoretical results. To this end, this paper studies the approximation ratio of those algorithms for unary input strings, which is strongly related to the field of addition chains. We show that in this setting, RePair and LongestMatch produce equal size grammars that are by a factor of at most $\log_2(3)$ larger than a smallest grammar. We also provide a matching lower bound. The main result of this paper is a new lower bound for Greedy of 1.348..., which improves the best known lower bound for arbitrary (not necessarily unary) input strings.

Keywords: Data compression · Grammar-based compression · Approximation algorithm · Addition chain

1 Introduction

The goal of grammar-based compression is to represent a word w by a small context-free grammar that produces exactly $\{w\}$. Such a grammar is called a straight-line program (SLP) for w. In the best case, one gets an SLP of size $\Theta(\log n)$ for a word of length n, where the size of an SLP is the total length of all right-hand sides of the rules of the grammar. A grammar-based compressor is an algorithm that produces an SLP for a given word w. There are various grammar-based compressors that can be found at many places in the literature. Well-known examples are the classical LZ78-compressor[1] of Lempel and Ziv [21], BISECTION [12] and SEQUITUR [17], just to mention a few. In this paper, we study the class of global grammar-based compressors which are also called global algorithms. A key concept of those algorithms are maximal strings. A maximal string of an SLP $\mathbb{A}$ is a word that has length at least two and occurs at least twice without overlap as a factor of the right-hand sides of the rules of $\mathbb{A}$. Further, no strictly longer word appears at least as many times without overlap as a factor of the right-hand sides of $\mathbb{A}$. For an input word w, a global grammar-based

[1] While LZ78 was not introduced as a grammar-based compressor, it is straightforward to compute from the LZ78-factorization of w an SLP for w of roughly the same size.

compressor starts with the SLP that has a single rule $S \to w$, where S is the start nonterminal of the grammar. The SLP is then recursively updated by choosing a maximal string γ of the current SLP and replacing a maximal set of pairwise non-overlapping occurrences of γ by a fresh nonterminal X. Additionally, a new rule $X \to \gamma$ is introduced. The algorithm stops when the obtained SLP has no maximal string. The probably best known example for a global algorithm is RePair [13], which selects in each round a most frequent maximal string. Note that the RePair algorithm as it is proposed in [13] always selects a word of length 2, but in this paper we follow the definition of [8], where the algorithm possibly selects longer words. However, both definitions coincide for unary input strings as considered in this work. Other global algorithms are LongestMatch [11], which chooses a longest maximal string in each round, and Greedy [2–4], which selects a maximal string that minimizes the size of the SLP obtained in the current round. It is again worth mentioning that the Greedy algorithm as originally presented in [2–4] is different from the version studied in this work as well as in [8]: The original Greedy algorithm only considers the right-hand side of the start rule for the choice and the replacement of the maximal string. In particular, all other rules do not change after they are introduced.

In the seminal work of Charikar et al. [8], the worst case approximation ratio of grammar-based compressors is studied. For a grammar-based compressor $\mathcal{C}$ that computes an SLP $\mathcal{C}(w)$ for a given word 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. In [8] the authors provide upper and lower bounds for the approximation ratios of several grammar-based compressors (among them are all compressors mentioned so far), but for none of the compressors the lower and upper bounds match. For LZ78 and BISECTION those gaps were closed in [15]. For all global algorithms, the best upper bound on the approximation ratio is $\mathcal{O}((n/\log n)^{2/3})$ [8], while the best known lower bounds are $\Omega(\log n/\log\log n)$ for RePair [14], $\Omega(\log\log n)$ for LongestMatch and $5/(3\log_3(5)) = 1.137...$ for Greedy [8]. In the context of our work, it is worth mentioning that the lower bound for Greedy uses words over a unary alphabet. In general, the achieved bounds "leave a large gap of understanding surrounding the global algorithms" as the authors in [8] conclude.

We aim to strengthen the understanding of global grammar-based compressors in this paper by studying the behavior of these algorithms on unary inputs, i.e., words of the form a^n for some symbol a. Grammar-based compression on unary words is strongly related to the field of addition chains, which has been studied for decades (see [16, Chapter 4.6.3] for a survey) and still is an active topic due to the strong connection to public key cryptosystems (see [18] for a review from that point of view). An addition chain for an integer n of size m is a sequence of integers $1 = k_1, k_2, \ldots, k_m = n$ such that for each d $(2 \leq d \leq m)$, there exists i, j $(1 \leq i, j < d)$ such that $k_i + k_j = k_d$. It is straightforward to compute from an addition chain for an integer n of size m an SLP for a^n of size $2m - 2$. Vice versa, an SLP for a^n of size m yields an addition chain for n of size m. So this paper can also be interpreted as a study of global algorithms as addition chain solvers. For RePair and LongestMatch, the restriction to unary inputs allows a full understanding of the produced SLPs and it turns out that for all unary inputs the SLP produced by RePair has the same size as the SLP produced by LongestMatch. In fact, both algorithms are basically identical to the binary method that produces an addition chain for n by creating powers of two using repeated squaring, and then the integer n is represented as the sum of those powers of two that correspond to a one in the binary representation of n. Based on that information, we show that for any unary input w the produced SLPs of RePair and LongestMatch have size at most $\log_2(3) \cdot g(w)$, and we provide a matching lower bound.

Unfortunately, even for unary inputs it is hard to analyze the general behavior of Greedy due to the discrete optimization problem in each round of the algorithm. The (probably weak) upper bound that we achieve for the approximation ratio of Greedy on unary inputs is $\mathcal{O}(n^{1/4}/\log n)$. We derive this bound by estimating the size of the SLP obtained by Greedy after three rounds, which already indicates space for improvement. For the original Greedy algorithm where only the start rule is compressed, it is a direct consequence of our analysis that the approximation ratio on unary input strings is $\Theta(\sqrt{n}/\log n)$. On the positive side, we provide a new lower bound of 1.348... for the approximation ratio of Greedy (in the variant where all right-hand sides are considered) that improves the best known lower bound for inputs over arbitrary alphabets. The key to achieve the new bound is the sequence $y_k = y_{k-1}^2 + 1$ with $y_0 = 2$, which has been studied in [1] (among other sequences), where it is shown that $y_k = \lfloor \gamma^{2^k} \rfloor$ for $\gamma = 2.258...$. In order to prove the lower bound, we show that the SLP produced by Greedy on input a^{y_k} has size $3 \cdot 2^k - 1$, while a smallest SLP for a^{y_k} has size $3 \cdot \log_3(\gamma) \cdot 2^k + o(2^k)$ (this follows from a construction used to prove the lower bound for Greedy in [8]).

Related work. One of the first appearances of straight-line programs in the literature are [6, 9], where they are called word chains (since they generalize addition chains from numbers to words). In [6], Berstel and Brlek prove that the function $g(k,n) = \max\{g(w) \mid w \in \{1,\ldots,k\}^n\}$ is in $\Theta(n/\log_k n)$. Recall that $g(w)$ is the size of a smallest SLP for the word w and thus $g(k,n)$ measures the worst case SLP-compression over all words of length n over a k-letter alphabet.

The smallest grammar problem is the problem of computing a smallest SLP for a given input word. It is known from [8, 20] that in general no grammar-based compressor can solve the smallest grammar problem in polynomial time unless $\mathsf{P} = \mathsf{NP}$. Even worse, unless $\mathsf{P} = \mathsf{NP}$ one cannot compute in polynomial time for a given word w an SLP of size at most $\frac{8569}{8568} \cdot g(w)$ [8]. One should mention that the constructions to prove those hardness results use alphabets of unbounded size. While in [8] it is remarked that the construction in [20] works for words over a ternary alphabet, Casel et al. [7] argue that this is not clear at all and provide a construction for fixed alphabets of size at least 24. However, for grammar-based compression on unary strings as it is studied in this work (as well as for the problem of computing a smallest addition chain), there is no NP-hardness result, so there might be an optimal polynomial-time algorithm even though it is widely believed that there is none.

Other notable systematic investigations of grammar-based compression are provided in [11, 19]. Whereas in [11], grammar-based compressors are used for universal lossless compression (in the information-theoretical sense), it is shown in [19] that the size of so-called irreducible SLPs (that include SLPs produced by global algorithms) can be upper bounded by the (unnormalized) k-th order empirical entropy of the produced string plus some lower order terms.

2 Preliminaries

For $i, j \in \mathbb{N}$, let $[i, j] = \{i, i+1, \dots, j\}$ for $i \leq j$ and $[i, j] = \emptyset$ otherwise. For integers m, n, we denote by m div n the integer division of m and n. We denote by m mod n the modulo of m and n, i.e., $m \bmod n \in [0, n-1]$ and

$$m = (m \text{ div } n) \cdot n + (m \bmod n).$$

If m/n or $\frac{m}{n}$ is used, then this refers to the standard division over $\mathbb{R}$. Note that $m \text{ div } n = \lfloor m/n \rfloor$ and $(m \text{ div } n) + (m \bmod n) \geq m/n$.

For an *alphabet* Σ, let $w = a_1 \cdots a_n$ $(a_1, \dots, a_n \in \Sigma)$ be a *word* or *string* over Σ. The length $|w|$ of w is n and we denote by ε the word of length 0. A *unary word* is a word of the form a^n for $a \in \Sigma$. Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ be the set of all nonempty words. For $w \in \Sigma^+$, we call $v \in \Sigma^+$ a *factor* of w if there exist $x, y \in \Sigma^*$ such that $w = xvy$.

2.1 Straight-line programs.

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:

- For every $A \in N$, there exists exactly one production of the form $A \to w$, and
- 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, nonempty word. The word defined by the SLP $\mathbb{A}$ is the word produced by the start nonterminal S. 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 $w \in \Sigma^+$. We will use the following inequalities that can be found in [8]:

Lemma 1 ([8]). *For all unary words w of length n, we have*

$$3 \log_3(n) - 3 \leq g(w) \leq 3 \log_3(n) + o(\log n).$$

Note that the first inequality also holds when w is a word over an arbitrary alphabet. The proof of the first inequality can be found in Lemma 1 of [8] and the second inequality is shown in the proof of Theorem 11 of [8].

Approximation ratio. A *grammar-based compressor* $\mathcal{C}$ is an algorithm that computes for a nonempty word w an SLP $\mathcal{C}(w)$ that produces the word 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:

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

In this paper we are mainly interested in the case $k = 1$, i.e., we are interested in grammar-based compression on unary words.

3 Global algorithms

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

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

A *global grammar-based compressor* starts on input w with the SLP $\mathbb{A}_0 = (\{S\}, \Sigma, \{S \to w\}, S)$. In each round $i \geq 1$, the algorithm selects a maximal string γ of $\mathbb{A}_{i-1}$ and updates $\mathbb{A}_{i-1}$ to $\mathbb{A}_i$ by replacing a largest set of pairwise non-overlapping occurrences of γ in $\mathbb{A}_{i-1}$ by a fresh nonterminal X. Additionally, the algorithm introduces the rule $X \to \gamma$ in $\mathbb{A}_i$. The algorithm stops when no maximal string occurs. Note that the replacement is not unique, e.g. the word a^5 has a unique maximal string $\gamma = aa$, which 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.

3.1 Greedy

The global grammar-based compressor Greedy selects in each round $i \geq 1$ a maximal string of $\mathbb{A}_{i-1}$ such that $\mathbb{A}_i$ has minimal size among all possible choices of maximal strings of $\mathbb{A}_{i-1}$.

We start with the main result of this paper, which is a new lower bound for the approximation ratio of Greedy. The best known lower bound [8] so far is

$$\alpha_{\mathsf{Greedy}}(k,n) \geq \frac{5}{3\log_3(5)} = 1.13767699...$$

for all $k \geq 1$ and infinitely many n. This bound is achieved using unary input strings. A key concept to prove a better lower bound is the sequence x_n described in the following lemma by [1]:

Lemma 2 ([1, Example 2.2]). *Let $x_{n+1} = x_n^2 + 1$ with $x_0 = 1$ and*

$$\beta = \exp\left(\sum_{i=1}^{\infty} \frac{1}{2^i} \log\left(1 + \frac{1}{x_i^2}\right)\right).$$

We have $x_n = \left\lfloor \beta^{2^n} \right\rfloor$.

In this work, we use the shifted sequence $y_n = x_{n+1}$, i.e., we start with $y_0 = 2$. It follows that $y_n = \left\lfloor \gamma^{2^n} \right\rfloor$, where $\gamma = \beta^2 = 2.25851845...$. Additionally, we need the following lemma:

Lemma 3. *Let $m \geq 1$ be an integer. Let $f_m : \mathbb{R}_{>0} \to \mathbb{R}$ with*

$$f_m(x) = x + \frac{m^2+1}{x}.$$

We have $f_m(x) > 2m$ for all $x > 0$.

Proof. The unique minimum of $f_m(x)$ is $2\sqrt{m^2+1}$ for $x = \sqrt{m^2+1}$. It follows that $f_m(x) \geq 2\sqrt{m^2+1} > 2\sqrt{m^2} = 2m$.

Now we are able to prove the new lower bound for Greedy:

Theorem 1. *For all $k \geq 1$ and infinitely many n, we have*

$$\alpha_{\mathsf{Greedy}}(k,n) \geq \frac{1}{\log_3(\gamma)} = 1.34847194... \ .$$

Proof. Let $\Sigma = \{a\}$ be a unary alphabet. We define $w_k = a^{y_k}$. By Lemma 2, we have $|w_k| \leq \gamma^{2^k}$. Applying Lemma 1 yields

$$g(w_k) \leq 3 \cdot \log_3(\gamma) \cdot 2^k + o(2^k).$$

In the remaining proof we show that on input w_k, Greedy produces an SLP of size $3 \cdot 2^k - 1$, which directly implies $\alpha_{\mathsf{Greedy}}(1,n) \geq 3/(3\log_3(\gamma))$. We start with the SLP $\mathbb{A}_0$ which has the single rule $S \to a^{y_k}$. Consider now the first round of the algorithm, i.e., we need to find a maximal string a^x of $\mathbb{A}_0$ such that the grammar $\mathbb{A}_1$ with rules

$$X_1 \to a^x, \ \ S \to X_1^{y_k \text{ div } x} a^{y_k \bmod x}$$

has minimal size. We have $|\mathbb{A}_1| = x + (y_k \text{ div } x) + (y_k \bmod x) \geq x + y_k/x$. By the definition of y_k we have $|\mathbb{A}_1| \geq x + (y_{k-1}^2 + 1)/x$. Applying Lemma 3 yields $|\mathbb{A}_1| \geq 2y_{k-1} + 1$. Note that for $x = y_{k-1}$ this minimum is achieved, i.e., we can assume that Greedy selects the maximal string $a^{y_{k-1}}$ and $\mathbb{A}_1$ is

$$X_1 \to a^{y_{k-1}}, \ S \to X_1^{y_{k-1}} a.$$

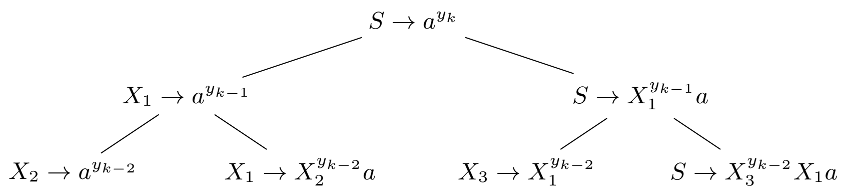

Fig. 1. Three rounds of Greedy on input a^{y_k}.

Each maximal string of $\mathbb{A}_1$ is either a unary word over X or a unary word over a, i.e., we can analyze the behavior of Greedy on both rules independently. The rule $X_1 \to a^{y_{k-1}}$ is obviously treated similarly as the initial SLP $\mathbb{A}_0$, so we continue with analyzing $S \to X_1^{y_{k-1}} a$. But again, the same arguments as above show that Greedy introduces a rule $X_3 \to X_1^{y_{k-2}}$ which yields $S \to X_3^{y_{k-2}} X_1 a$ as the new start rule. This process can be iterated using the same arguments for the leading unary strings of length y_i for some $i \in [1, k]$.

The reader might think of this process as a binary tree, where each node is labelled with a rule (the root is labelled with $S \to a^{y_k}$) and the children of a node are the two rules obtained by Greedy when the rule has been processed. We assume that the left child represents the rule for the chosen maximal string and the right child represents the parent rule where all occurrences of the maximal string are replaced by the fresh nonterminal. In Figure 1 this binary tree is depicted for the steps we discussed above. Note that when a rule is processed, the longest common factor of the two new rules has length 1 (the remainder). More generally, after each round there is no word of length at least two that occurs as a factor in two different rules, since a possibly shared remainder has length 1 and otherwise only fresh nonterminals are introduced. It follows that we can iterate this process independently for each rule until no maximal string occurs. This is the case when each rule starts with a unary string of length $y_0 = 2$ or, in terms of the interpretation as a binary tree, when a full binary tree of height k is produced. Each right branch occurring in this tree adds a new remainder to those remainders that already occur in the parent rule and a left branch introduces a new (smaller) instance of the start problem. We show by induction that on level $i \in [0, k]$ of this full binary tree of height k, there is one rule of size $y_{k-i} + i$ and 2^{i-j-1} many rules of size $y_{k-i} + j$ for $j \in [0, i-1]$. On level 0, this is true since there is only a single rule of size $y_k + 0$. Assuming that our claim is true on level $i < k$, we derive from each rule on level i two new rules on level $i+1$: A right branch yields a rule that starts with a leading unary string of size y_{k-i-1} and adds a new remainder to the parent rule. A left branch yields a rule that contains only a unary string of size y_{k-i-1}. If we first consider the left branches, we derive that each of the 2^i many rules on level i adds a rule of size y_{k-i-1} on level $i+1$. For the right branches, the single rule of size $y_{k-i} + i$ on level i yields a rule of size $y_{k-i-1} + i + 1$ on level $i+1$. Further, each of the 2^{i-j-1} many rules of size $y_{k-i} + j$ ($j \in [0, i-1]$) yields a rule of size $y_{k-i-1} + j + 1$. When we put everything together, we get that on level $i+1$

there is a single rule of size $y_{k-i-1}+i+1$ and 2^{i-j} many rules of size $y_{k-i-1}+j$ for $j \in [0,i]$. That finishes the induction. It follows that the final SLP (which consists of the rules on level k) has a single rule of size $y_0+k = 2+k$ and 2^{k-j-1} many rules of size $2+j$ for $j = 0,\dots,k-1$. This gives a total size of

$$\begin{aligned}
2+k+\sum_{j=0}^{k-1} 2^{k-j-1}(2+j) &= 2+k+2^k\sum_{j=0}^{k-1}2^{-j}+2^k\sum_{j=0}^{k-1}2^{-j-1}j \\
&= 2+k+2^k(2-2^{-k+1})+2^k(-2^{-k}k-2^{-k}+1) \\
&= 2+k+2^{k+1}-2-k-1+2^k \\
&= 2^{k+1}+2^k-1 \\
&= 3\cdot 2^k-1.
\end{aligned}$$

In the remaining part of this section, we prove an upper bound on the size of the SLP produced by Greedy on input a^n:

Theorem 2. *The SLP produced by* Greedy *(after three rounds) on input a^n has size $\mathcal{O}(n^{1/4})$.*

Proof. Consider an input a^n with $n \geq 4$ (otherwise $S \to a^n$ is the final SLP since there is no maximal string). The SLP $\mathbb{A}_1$ obtained by Greedy after the first round has the form

$$X \to a^x, \;\; S \to X^{n \text{ div } x} a^{n \text{ mod } x}, \tag{1}$$

where a^x is the selected maximal string. We first show

$$\frac{1}{3}\sqrt{n} \leq x \leq 3\sqrt{n}, \;\; \frac{1}{3}\sqrt{n} \leq n \text{ div } x \leq 3\sqrt{n}, \;\; n \text{ mod } x < 3\sqrt{n}.$$

Assume $x = \lceil\sqrt{n}\rceil$ in equation (1). In this case, the size of the SLP is

$$\lceil\sqrt{n}\rceil + \left\lfloor \frac{n}{\lceil\sqrt{n}\rceil} \right\rfloor + n \text{ mod } \lceil\sqrt{n}\rceil \leq 3\sqrt{n}+1.$$

Since the maximal string a^x is selected greedily such that $\mathbb{A}_1$ has minimal size, we have $|\mathbb{A}_1| \leq 3\sqrt{n}+1$. It follows that $x \leq 3\sqrt{n}$, because otherwise the size of $\mathbb{A}_1$ would be at least $3\sqrt{n}+2$ due to the fact that a maximal string (represented by the nonterminal X) occurs at least twice. It follows that $n \text{ mod } x < 3\sqrt{n}$ and $n \text{ div } x \geq 1/3\sqrt{n}$. Further, we have $n \text{ div } x \leq 3\sqrt{n}$ because otherwise the size of $\mathbb{A}_1$ would be at least $3\sqrt{n}+2$ due to the fact that $x \geq 2$ (a maximal string has length at least two). It follows that $x \geq 1/3\sqrt{n}$. Actually, a slightly more careful analysis allows sharper bounds for x, n div x and n mod x, but for the matter of this proof it is easier to work with the constants 3 and 1/3.

Now the only maximal strings occurring in $\mathbb{A}_1$ are of the form X^y or a^z (for integers $y, z \geq 2$) since no other factor of the right-hand sides of $\mathbb{A}_1$ occurs at least twice. Note that both optimization problems (for X^y and a^z) are independent, so we assume the chosen maximal string in the second round has the form X^z, afterwards we proceed with a^z. Let $d = n \text{ div } x$, where a^x is again the maximal string that has been selected in the first round. Then the SLP $\mathbb{A}_2$ obtained after the second round of Greedy has the form

$$X \to a^x, \;\; Y \to X^y, \;\; S \to Y^{d \text{ div } y} X^{d \bmod y} a^{n \bmod x}. \tag{2}$$

Let $g(x) = x + (n \bmod x)$, which is the size of those parts of $\mathbb{A}_2$ that are independent of the choice of the maximal string X^y. Assume $y = \lceil n^{1/4} \rceil$ in (2) and let n be large enough such that Y occurs at least twice in the start rule. This yields an SLP of size

$$\lceil n^{\frac{1}{4}} \rceil + \left\lfloor \frac{d}{\lceil n^{\frac{1}{4}} \rceil} \right\rfloor + d \bmod \lceil n^{\frac{1}{4}} \rceil + g(x) \leq 5n^{\frac{1}{4}} + 1 + g(x).$$

The inequality is achieved by using $d = n \text{ div } x \leq 3\sqrt{n}$ as argued above. It follows again from the greedy nature of the algorithm that $|\mathbb{A}_2| \leq 5n^{1/4} + 1 + g(x)$ and similar arguments as above show that the exponents y, $d \text{ div } y$ and $d \bmod y$ can be upper bounded by $5n^{1/4}$.

All maximal strings occurring in $\mathbb{A}_2$ are again unary words, but since a^x has length at least $1/3\sqrt{n}$ and the lengths of all other unary factors over X or Y are bounded by $5n^{1/4}$, we can assume that (for n large enough) Greedy selects a^z for some integer $z \geq 2$ as the maximal string in order to achieve a minimal size SLP $\mathbb{A}_3$. Note that if we would have assumed that the chosen maximal string in round two is a^z instead of X^y, then similar arguments would show that X^y is selected in round three if n is large enough. Now, let $e = n \bmod x$, then the SLP $\mathbb{A}_3$ obtained after the third round has the form

$$\begin{gathered} Z \to a^z, \;\; X \to Z^{x \text{ div } z} a^{x \bmod z}, \;\; Y \to X^y, \\ S \to Y^{d \text{ div } y} X^{d \bmod y} Z^{e \text{ div } z} a^{e \bmod z}. \end{gathered} \tag{3}$$

Let $h(y) = y + (d \text{ div } y) + (d \bmod y)$, which is the size of those parts of $\mathbb{A}_3$ that are independent of the choice of the maximal string a^z. Assume now $z = \lceil n^{1/4} \rceil$ and let n be large enough such that Z occurs at least twice in the right-hand sides of (3). The obtained SLP has size

$$\lceil n^{1/4} \rceil + \left\lfloor \frac{x}{\lceil n^{1/4} \rceil} \right\rfloor + (x \bmod \lceil n^{1/4} \rceil) + \left\lfloor \frac{e}{\lceil n^{1/4} \rceil} \right\rfloor + (e \bmod \lceil n^{1/4} \rceil) + h(y).$$

Using $x \leq 3\sqrt{n}$ and $e = n \bmod x < 3\sqrt{n}$ we can upper bound this size by $9n^{1/4} + 1 + h(y)$. Note that we have already bounded the size of $h(y)$ by $5n^{1/4} + 1$ in the previous step, so the SLP $\mathbb{A}_3$ obtained by Greedy after three rounds has size at most $14n^{1/4} + 2$.

The bound on the approximation ratio is now achieved using Lemma 1, which shows that a smallest grammar for a^n has size $\Omega(\log n)$.

Corollary 1. *We have $\alpha_{\mathsf{Greedy}}(1,n) \in \mathcal{O}(n^{1/4}/\log n)$.*

The reader might wonder why our estimation stops after three rounds of Greedy. The most important reason is that a precise invariant is missing in order to iterate our arguments for a non constant number of rounds. On the other hand, it seems likely that similar arguments as provided in the proof of Theorem 2 can be used to show that the SLP produced by Greedy after some more rounds has size $\mathcal{O}(n^{1/8})$ and maybe again after some rounds has size $\mathcal{O}(n^{1/16})$. However, further analysis would require more and more case distinctions since it is not clear anymore that the selected maximal string is always unary as the reader can see in (3), where factors of the form Z^*a^* can occur more than once on the right-hand sides. It seems therefore necessary to apply some new information in order to improve the upper bound beyond $\mathcal{O}(n^{1/2^k})$ for some fixed k.

An interesting consequence of the proof of Proposition 2 applies to the originally proposed Greedy variant [2–4]. Recall from the introduction that in this setting, the algorithm recursively chooses the maximal string only in dependence on the right-hand side of the start rule and replaces the occurrences of the chosen string only there. In other words, the right-hand side of the start rule is compressed in a greedy way and all other rules do not change after they are introduced. Note that in the first round both variants of Greedy (the one studied here and the original one) are identical, because in this case the only rule of the SLP is the start rule $S \to a^n$. Hence, our analysis of the first step in the proof of Proposition 2 applies to the original variant as well. We have shown that the selected maximal string in the first round (and thus the right-hand side of the introduced rule) has length $\Theta(\sqrt{n})$ and since the original variant does not modify the corresponding rule any further, it follows directly that the SLP produced by the original algorithm has size $\Omega(\sqrt{n})$. But since the modified start rule has also size $\Theta(\sqrt{n})$ after the first step, it follows that the SLP produced by the original algorithm has size $\mathcal{O}(\sqrt{n})$ as well (the size of the SLP does not increase later). Together with Lemma 1, it follows that this variant of the Greedy algorithm has approximation ratio $\Theta(\sqrt{n}/\log n)$ on unary inputs of length n.

3.2 RePair and LongestMatch

In this section we analyze the global grammar-based compressors RePair and LongestMatch. In each round i, RePair selects a most frequent maximal string of $\mathbb{A}_{i-1}$ and LongestMatch selects a longest maximal string of $\mathbb{A}_{i-1}$.

We will abbreviate the approximation ratio $\alpha_{\mathsf{LongestMatch}}$ by α_{LM} for better readability. We will first show that RePair and LongestMatch produce SLPs of equal size for unary inputs a^n and we prove the exact size of those SLPs in dependency on n. In a second step, we use this information to obtain our result for $\alpha_{\mathsf{RePair}}(1,n)$, respectively $\alpha_{\mathsf{LM}}(1,n)$. Fix an integer $n \geq 2$ and consider the binary representation

$$n = \sum_{i=0}^{\lfloor \log_2 n \rfloor} b_i \cdot 2^i \qquad (4)$$

of n, where $b_i \in \{0,1\}$ for $i \in [0, \lfloor \log_2 n \rfloor]$. We denote by $\nu(n)$ the number of 1's in the binary representation of n, i.e.,

$$\nu(n) = \sum_{i=0}^{\lfloor \log_2 n \rfloor} b_i.$$

For example, we have $11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ and thus $b_0 = b_1 = b_3 = 1$, $b_2 = 0$ and $\nu(11) = 3$.

Proposition 1. *For $n \geq 2$, let $\mathbb{A}$ be the SLP produced by* RePair *on input a^n and $\mathbb{B}$ be the SLP produced by* LongestMatch *on input a^n. We have*

$$|\mathbb{A}| = |\mathbb{B}| = 2\lfloor \log_2 n \rfloor + \nu(n) - 1.$$

Proof. If $n = 2$ or $n = 3$ (we only consider $n \geq 2$), then a^n has no maximal string and thus the final SLP of any global algorithm has a single rule $S \to a^n$. The reader can easily verify the claimed result for those cases.

We assume $n \geq 4$ in the following. Let $m = \lfloor \log_2 n \rfloor - 1$. We prove the claim for RePair first, afterwards we proceed with LongestMatch. On input a^n, RePair runs for exactly m rounds and creates rules $X_1 \to aa$ and $X_i \to X_{i-1}X_{i-1}$ for $i \in [2, m]$, i.e., the nonterminal X_i produces the string a^{2^i}. This rules have total size $2m$. After this steps, the start rule is

$$S \to X_m X_m X_m^{b_m} X_{m-1}^{b_{m-1}} \cdots X_1^{b_1} a^{b_0},$$

where the b_i's are the coefficients occurring in the binary representation of n, see equation (4). In other words, the symbol a only occurs in the start rule if the least significant bit $b_0 = 1$, and the nonterminal X_i ($i \in [1, m-1]$) 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 $b_{m+1} = 1$ is represented by $X_m X_m$. A third X_m occurs in the start rule if and only if $b_m = 1$. The size of the start rule is $2 + \sum_{i=0}^{m} b_i$. It follows that the total size of the SLP produced by RePair on input a^n is $2m + 2 + \sum_{i=0}^{m} b_i$, which together with $m = \lfloor \log n \rfloor - 1$ and $b_{\lfloor \log n \rfloor} = 1$ (the most significant bit is always 1) yields the claimed size.

Now we prove the same result for LongestMatch. In the first round, the chosen maximal string is $a^{\lfloor n/2 \rfloor}$, which yields rules $X_1 \to a^{\lfloor n/2 \rfloor}$ and $S \to X_1 X_1 a^{b_0}$, i.e., the symbol a occurs in the start rule if and only if n is odd and thus the least significant bit $b_0 = 1$. Assuming $n \geq 8$, this procedure is now repeated for the rule $X_1 \to a^{\lfloor n/2 \rfloor}$ (for $n < 8$ there is no maximal string and the algorithm stops after the first round). This yields $X_2 \to a^{\lfloor n/4 \rfloor}$, $X_1 \to X_2 X_2 a^{b_1}$ and $S \to X_1 X_1 a^{b_0}$ (note that $\lfloor (\lfloor n/2 \rfloor)/2 \rfloor = \lfloor n/4 \rfloor$). After $m = \lfloor \log n \rfloor - 1$ steps, the iteration of that process results in the final SLP with rules $S \to X_1 X_1 a^{b_0}$, $X_i \to X_{i+1} X_{i+1} a^{b_i}$ for $i \in [1, m-1]$ and $X_m \to aaa^{b_m}$. The size of this SLP is $2 \cdot (m+1) + \sum_{i=0}^{m} b_i$, which directly implies the claimed result for LongestMatch.

Using Proposition 1, we prove the matching bounds for $\alpha_{\mathsf{RePair}}(1, n)$, $\alpha_{\mathsf{LM}}(1, n)$:

Theorem 3. *For all n, we have $\alpha_{\mathsf{RePair}}(1, n) = \alpha_{\mathsf{LM}}(1, n) \leq \log_2(3)$.*

Proof. As a consequence of Proposition 1, RePair and LongestMatch produce on input a^n SLPs of size at most $3\log_2 n$, since $\nu(n) - 1 \leq \log_2 n$. By Lemma 1, we have $g(a^n) \geq 3\log_3 n - 3$. The equality $\log_2 n / \log_3 n = \log_2(3)$ finishes the proof.

Theorem 4. *For infinitely many n, we have $\alpha_{\mathsf{RePair}}(1, n) = \alpha_{\mathsf{LM}}(1, n) \geq \log_2(3)$.*

Proof. Let $w_k = a^{2^k-1}$. We have $2^k - 1 = \sum_{i=0}^{k-1} 2^i$ and thus $\nu(2^k - 1) = k$. By Proposition 1, the size of the SLPs produced by RePair and LongestMatch is $3k - 3$. By Lemma 1, we have

$$g(w_k) \leq 3\log_3(2^k - 1) + o(\log(2^k - 1) \leq 3\log_3(2) \cdot k + o(k).$$

The equality $1/\log_3(2) = \log_2(3)$ finishes the proof.

4 Future work

The obvious question concerns the gap between the lower and upper bound for Greedy. First of all, it might be possible to improve our lower bound by finding a similar sequence such that Greedy produces larger remainders in each round, but care has to be taken since for larger remainders it is not true anymore that the rules can be analyzed independently because the rules could share factors of length greater 1. Concerning the upper bound, we conjecture that Greedy achieves logarithmic compression for all unary inputs and thus the approximation ratio is constant, but the direct analysis of the algorithm as we tried in Theorem 2 misses a clear invariant for a non constant number of rounds. For arbitrary alphabets, a non-constant lower bound for Greedy as well as an improvement of the upper bound of $\mathcal{O}((n/\log n)^{2/3})$ for any global algorithm seems to be natural starting points for future work.

References

1. A. V. Aho and N. J. A. Sloane. Some doubly exponential sequences. *Fib. Quart.*, 11, 429-437, 1973.
2. A. Apostolico and S. Lonardi. Some Theory and Practice of Greedy Off-Line Textual Substitution. *Proc. DCC*, 1998, pages 119–128, IEEE Computer Society, 1998.
3. A. Apostolico and S. Lonardi. Compression of Biological Sequences by Greedy Off-Line Textual Substitution. *Proc. DCC*, 2000, pages 143–152, IEEE Computer Society, 2000.
4. A. Apostolico and S. Lonardi. Off-Line Compression by Greedy Textual Substitution. *Proc. IEEE 88(11)*, 2000.
5. J. Arpe and R. Reischuk. On the complexity of optimal grammar-based compression. *Proc. DCC 2006*, pages 173–182. IEEE Computer Society, 2006.
6. J. Berstel and S. Brlek. On the length of word chains. *Inf. Process. Lett.*, 26(1):23–28, 1987.
7. K. Casel, H. Fernau, S. Gaspers, B. Gras, and M. L. Schmid. On the complexity of grammar-based compression over fixed alphabets. *Proc. ICALP 2016*, Lecture Notes in Computer Science. Springer, 1996.

8. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
9. A. A. Diwan. A new combinatorial complexity measure for languages. Tata Institute, Bombay, India, 1986.
10. A. Jeż. Approximation of grammar-based compression via recompression. *Theor. Comput. Sci.*, 592:115–134, 2015.
11. J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. Inf. Theory*, 46(3):737–754, 2000.
12. J. C. Kieffer, E.-H. Yang, G. J. Nelson, and P. C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inf. Theory*, 46(4):1227–1245, 2000.
13. N. J. Larsson and A. Moffat. Offline dictionary-based compression. *Proc. DCC 1999*, pages 296–305. IEEE Computer Society, 1999.
14. D. Hucke, A. Jeż, and M. Lohrey. Approximation ratio of RePair. Technical report, arxiv.org, 2017, https://arxiv.org/abs/1703.06061
15. D. Hucke, M. Lohrey, and P. Reh. The smallest grammar problem revisited. *Proceedings of SPIRE 2016*, LNCS 9954, pages 35–49. Springer 2017.
16. D. E. Knuth. The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition, Addison-Wesley, 1998.
17. C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997.
18. A. M. Noma, A. Muhammed, M. A. Mohamed, and Z. A. Zulkarnain A Review on Heuristics for Addition Chain Problem: Towards Efficient Public Key Cryptosystems. Journal of Computer Science 13(8):275-289, 2017
19. C. Ochoa and G. Navarro. RePair and All Irreducible Grammars are Upper Bounded by High-Order Empirical Entropy. IEEE Trans. Information Theory 65(5): 3160–3164, 2019
20. J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982.
21. J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1977.