RePair Grammars Are the Smallest Grammars for Fibonacci Words

Takuya Mieno[1,2]
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Shunsuke Inenaga
Department of Informatics, Kyushu University, Fukuoka, Japan
PRESTO, Japan Science and Technology Agency, Kawaguchi, Japan

Takashi Horiyama
Faculty of Information Science and Technology, Hokkaido University, Sapporo, Japan

Abstract

Grammar-based compression is a loss-less data compression scheme that represents a given string w by a context-free grammar that generates only w. While computing the smallest grammar which generates a given string w is NP-hard in general, a number of polynomial-time grammar-based compressors which work well in practice have been proposed. *RePair*, proposed by Larsson and Moffat in 1999, is a grammar-based compressor which recursively replaces all possible occurrences of a most frequently occurring bigrams in the string. Since there can be multiple choices of the most frequent bigrams to replace, different implementations of RePair can result in different grammars. In this paper, we show that the smallest grammars generating the Fibonacci words F_k can be completely characterized by RePair, where F_k denotes the k-th Fibonacci word. Namely, all grammars for F_k generated by any implementation of RePair are the smallest grammars for F_k, and no other grammars can be the smallest for F_k. To the best of our knowledge, Fibonacci words are the first non-trivial infinite family of strings for which RePair is optimal.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words

Keywords and phrases grammar based compression, Fibonacci words, RePair, smallest grammar problem

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.26

Related Version *Full Version*: `https://arxiv.org/abs/2202.08447`

Funding *Takuya Mieno*: JSPS KAKENHI Grant Numbers 20H05964 and JP20J11983.
Shunsuke Inenaga: JST PRESTO Grant Number JPMJPR1922.
Takashi Horiyama: JSPS KAKENHI Grant Number 20H05964.

1 Introduction

A context-free grammar in the Chomsky normal form that produces only a single string w is called a *straight-line program* (*SLP*) for w. Highly repetitive strings that contain many long repeats can be compactly represented by SLPs since occurrences of equal substrings can be replaced by a common non-terminal symbol. *Grammar-based compression* is a loss-less data compression scheme that represents a string w by an SLP for w. We are aware of more powerful compression schemes such as run-length SLPs [24, 37, 6], composition systems [19], collage systems [26], NU-systems [36], the Lempel-Ziv 77 family [42, 39, 12, 13], and bidirectional schemes [39]. Nevertheless, since SLPs exhibit simpler structures than those, a number of efficient algorithms that can work directly on SLPs have been proposed,

[1] Corresponding author
[2] Current affiliation: University of Electro-Communications, Japan (`tmieno@uec.ac.jp`)

including pattern matching [25, 24], convolutions [40], random access [8], detection of repeats and palindromes [22], Lyndon factorizations [23], longest common extension queries [21], longest common substrings [34], finger searches [5], and balancing the grammar [17]. More examples of algorithms directly working on SLPs can be found in references therein and the survey [31]. Since these algorithms do not decompress the SLPs, they can be more efficient than solutions on uncompressed strings.

Since the complexities of the algorithms mentioned above depend on the size of the SLP, it is important to compute a small grammar for a given string. *The smallest grammar problem* is to find a grammar that derives a given string w, where the total length of the right-hand sides of the productions is the smallest possible. The smallest grammar problem is known to be NP-hard in general [39, 10]. Namely, there is no polynomial-time algorithm that finds the smallest grammar for *arbitrary* strings, unless P = NP. Notably, the NP-hardness holds even when the alphabet size is bounded by some constant at least 17 [9], on the other hand, it is open whether the NP-hardness holds for strings over a smaller constant alphabet, particularly on binary alphabets.

We consider a slightly restricted version of the smallest grammar problem where the considered grammars are SLPs, i.e., only those in the Chomsky normal form. We follow a widely accepted definition for the *size* of an SLP, which is the number of productions in it. Thus, in the rest of our paper, grammars mean SLPs unless otherwise stated, and our smallest grammar problem seeks the smallest SLP, which generates the input string with the fewest productions[3]. There are some trivial examples of strings whose smallest grammar sizes can be easily determined, e.g., a unary string $(\mathtt{a})^{2^i}$ of length power of two[4], and non-compressible strings in which all the symbols are distinct. It is interesting to identify classes of strings whose smallest grammars can be determined in polynomial-time since it may lead to more and deeper insights to the smallest grammar problem. To the best of our knowledge, however, no previous work shows non-trivial strings whose smallest grammar sizes are computable in polynomial-time.

In this paper, we study the smallest grammars of the *Fibonacci words* $\{F_1, F_2, \ldots, F_n, \ldots\}$ defined recursively as follows: $F_1 = \mathtt{b}$, $F_2 = \mathtt{a}$, and $F_i = F_{i-1}F_{i-2}$ for $i \geq 3$. We show that the smallest grammars of the Fibonacci words can be completely characterized by the famous *RePair* [30] algorithm, which is the best known practical grammar compressor that recursively replaces the most frequently occurring bigram with a new non-terminal symbol in linear total time. We first prove that the size of the smallest grammar of the n-th Fibonacci word F_n is n. We then prove that applying *any implementation* of RePair to F_n always provides a smallest grammar of F_n, and conversely, only such grammars can be the smallest for Fibonacci words. This was partially observed earlier in the experiments by Furuya et al. [15], where five different implementations of RePair produced grammars of the same size for the fib41 string from the Repetitive Corpus of the Pizza&Chili Corpus (`http://pizzachili.dcc.uchile.cl/repcorpus.html`). However, to our knowledge, this paper is the first that gives theoretical evidence.

[3] There is an alternative definition of the size of a grammar, that is, the total sum of the lengths of the right side of its rules. This definition is usually used for non-SLP grammars.

[4] Grammars for unary words are closely related to *addition chains* [28], and the smallest (not necessarily SLP) grammar for $(\mathtt{a})^k$ is non-trivial for general k that is not a power of two. Also, in such a case, RePair does not provide the smallest grammar for $(\mathtt{a})^k$ [20].

Related Work

Although the smallest grammar problem is NP-hard, there exist polynomial-time approximations to the problem: Rytter's AVL-grammar [38] produces an SLP of size $O(s^* \log(N/s^*))$, where s^* denote the size of the smallest SLP for the input string and N is the length of the input string. The α-balanced grammar of Charikar et al. [10] produces a (non-SLP) grammar of size $O(g^* \log(N/g^*))$, where g^* denotes the size of the smallest (non-SLP) grammar. Upper bounds and lower bounds for the approximation ratios of other practical grammar compressors including LZ78 [43], BISECTION [27], RePair [30], SEQUENTIAL [41], LONGEST MATCH [27], and GREEDY [1], are also known [10, 2]. Charikar et al. [10] showed that the approximation ratio of RePair to the smallest (non-SLP) grammar is at most $O((N/\log N)^{2/3})$ and is at least $\Omega(\sqrt{\log N})$. The lower bound was later improved by Bannai et al. [2] to $\Omega(\log N/\log\log N)$. Furthermore, it is known that RePair has a lower bound on the approximation ratio $\log_2(3)$ to the smallest (non-SLP) grammar for *unary* strings [20]. On the other hand, RePair is known to achieve the best compression ratio on many real-world datasets and enjoy applications in web graph compression [11] and XML compression [32]. Some variants of RePair have also been proposed [33, 7, 18, 16, 15, 29].

2 Preliminaries

2.1 Strings

Let Σ be an alphabet. An element in Σ is called a symbol. An element in Σ^* is called a string. The length of string w is denoted by $|w|$. The empty string ε is the string of length 0. For each i with $1 \leq i \leq |w|$, $w[i]$ denotes the i-th symbol of w. For each i and j with $1 \leq i \leq j \leq |w|$, $w[i..j]$ denotes the *substring* of w which begins at position i and ends at position j. For convenience, let $w[i..j] = \varepsilon$ if $i > j$. When $i = 1$ (resp. $j = |w|$), $w[i..j]$ is called a *prefix* (resp. a *suffix*) of w. For non-empty strings w and b with $|b| < |w|$, b is called a *border* of w if b is both a prefix and a suffix of w. If there are no borders of w, then w is said to be *borderless*. For any non-empty string w, we call $w[|w|]w[1..|w|-1]$ the *right-rotation* of w. For a string w, σ_w denotes the number of distinct symbols appearing in w. For a non-empty string w, we denote by w^R the *reversed string* of w, namely $w^R = w[|w|]\cdots w[1]$.

2.2 Fibonacci Words and Related Words

For a binary alphabet $\{a, b\}$, *Fibonacci words* $F_i^{(a,b)}$ (starting with a for $i > 1$) are defined as follows: $F_1^{(a,b)} = b$, $F_2^{(a,b)} = a$, and $F_i^{(a,b)} = F_{i-1}^{(a,b)} F_{i-2}^{(a,b)}$ for $i \geq 3$. We call $F_i^{(a,b)}$ the i-th Fibonacci word (starting with a for $i > 1$). By the above definition of Fibonacci words, $|F_i^{(a,b)}| = f_i$ holds for each i, where f_i denotes the i-th Fibonacci number defined as follows: $f_1 = 1$, $f_2 = 1$, $f_i = f_{i-1} + f_{i-2}$ for $i \geq 3$. There is an alternative definition (e.g. [3]) of Fibonacci words using the *string morphism* $\phi^{(a,b)}$: The i-th Fibonacci word $F_i^{(a,b)}$ (starting with a for $i > 1$) is $(\phi^{(a,b)})^{i-1}(b)$, where $\phi^{(a,b)}$ is a morphism over $\{a, b\}$ such that $\phi^{(a,b)}(a) = ab$ and $\phi^{(a,b)}(b) = a$. We strictly distinguish the morphism $\phi^{(b,a)}$ from $\phi^{(a,b)}$ over the same binary alphabet $\{a, b\}$, namely, $\phi^{(b,a)}$ generates the Fibonacci words $F_i^{(b,a)}$ where a and b are flipped in $F_i^{(a,b)}$. We will omit the superscript (a, b) if it is clear from contexts or it is not essential for the discussion.

Next, we define other words, which will be utilized to analyze the smallest grammar of Fibonacci words. Let $\pi^{(a,b)}$ be the morphism over $\{a, b\}$ such that $\pi^{(a,b)}(a) = ab$ and $\pi^{(a,b)}(b) = abb$. Further, let $\theta^{(a,b)}$ be the morphism over $\{a, b\}$ such that $\theta^{(a,b)}(a) = aab$ and

Table 1 Lists of $F_i^{(\mathtt{a},\mathtt{b})}$ for $i = 1, \ldots, 10$, and $P_i^{(\mathtt{a},\mathtt{b})}$ and $Q_i^{(\mathtt{a},\mathtt{b})}$ for $i = 1, \ldots, 5$.

i	$F_i^{(\mathtt{a},\mathtt{b})}$	length
1	b	1
2	a	1
3	ab	2
4	aba	3
5	abaab	5
6	abaababa	8
7	abaababaabaab	13
8	abaababaabaababaababa	21
9	abaababaabaababaababaabaababaabaab	34
10	abaababaabaababaababaabaababaabaababaababaabaababaabaababa	55
i	$P_i^{(\mathtt{a},\mathtt{b})}$	length
1	a	1
2	ab	2
3	ababb	5
4	ababbababbabb	13
5	ababbababbabbababbababbabbababbabb	34
i	$Q_i^{(\mathtt{a},\mathtt{b})}$	length
1	a	1
2	aab	3
3	aabaabab	8
4	aabaababaabaababaabab	21
5	aabaababaabaababaababaabaababaabaababaababaabaababaabab	55

$\theta^{(a,b)}(b) = ab$. For each positive integer i, we define $P_i^{(a,b)}$ and $Q_i^{(a,b)}$ over $\{a, b\}$ by $P_i^{(a,b)} = (\pi^{(a,b)})^{i-1}(a)$ and $Q_i^{(a,b)} = (\theta^{(a,b)})^{i-1}(a)$, respectively. We treat their superscripts as for that of Fibonacci words. We will later show that $|P_i| = |F_{2i-1}| = f_{2i-1}$ and $|Q_i| = |F_{2i}| = f_{2i}$ for any $i \geq 1$. We show examples for these three words in Table 1. We remark that strings P_i and Q_i can be obtained at some point while RePair is being applied to the Fibonacci words. We will prove this in Section 4.

For a symbol X and a string y, let $\xi_{X \to y}$ be the morphism such that $\xi_{X \to y}(X) = y$ and $\xi_{X \to y}(c) = c$ for any symbol $c \neq X$. Namely, when applied to a string w, $\xi_{X \to y}(w)$ replaces all occurrences of X in w with y but any other symbols than X remain unchanged. For any morphism λ and any sequence $S = (s_1, \ldots, s_m)$ of strings, let $\lambda(S) = (\lambda(s_1), \ldots, \lambda(s_m))$.

2.3 Grammar Compression and RePair

A context-free grammar in the Chomsky normal form that produces a single string w is called a *straight-line program* (*SLP* in short) for w. Namely, any production in a grammar is of form either $X_i \to \alpha$ or $X_i \to X_j X_k$, where α is a terminal symbol and X_i, X_j, and X_k are non-terminal symbols such that $i > j$ and $i > k$, that is, there are no *cycles* in the productions. In what follows, we refer to an SLP that produces w simply as a *grammar of* w. Let $\mathcal{T}(G)$ denote the derivation tree of a grammar G, where each internal node in $\mathcal{T}(G)$ is labeled by the corresponding non-terminal symbol of G. As in [38], we conceptually identify terminal symbols with their parents so that $\mathcal{T}(G)$ is a full binary tree (i.e. every internal node has exactly two children). Let G_1 and G_2 be grammars both deriving the same string w, and

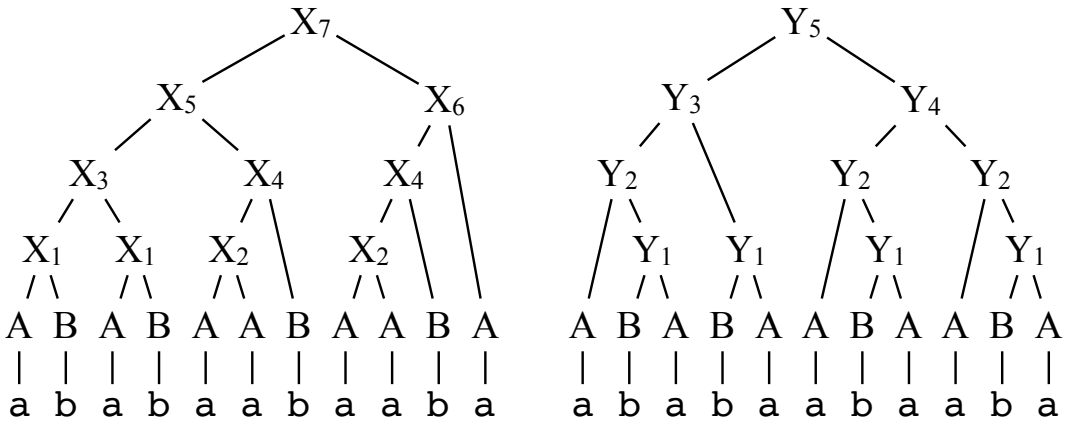

Figure 1 Illustration for the derivation trees of two distinct grammars of string $w = \texttt{ababaabaaba}$. The size of the grammar on the left is 9 since there are nine productions; $\{A \to \texttt{a}, B \to \texttt{b}, X_1 \to AB, X_2 \to AA, X_3 \to X_1X_1, X_4 \to X_2B, X_5 \to X_3X_4, X_6 \to X_4A, X_7 \to X_5X_6, \}$. On the other hand, the size of the grammar on the right is 7. Note that the right one is a RePair grammar of w. In the rest of the paper, we sometimes identify the terminal symbols (leaves) with their parents so the derivation trees are (conceptually) full binary trees.

let Π_1 and Π_2 be the sets of non-terminal symbols of G_1 and G_2, respectively. We say that G_1 and G_2 are *equivalent* if there exists a renaming bijection $f : \Pi_1 \to \Pi_2$ that transforms $\mathcal{T}(G_1)$ to $\mathcal{T}(G_2)$. We say that G_1 and G_2 are *distinct* if they are not equivalent. For example, two grammars $\{A \to a, B \to b, C \to AB, D \to CA\}$ and $\{X \to a, Y \to b, Z \to XY, W \to ZX\}$ are equivalent grammars both deriving string *aba*.

Equivalent grammars form an equivalence class of grammars, and we pick an arbitrary one as the representative of each equivalence class. A *set* S of grammars that derive the same string w is a set which consists of (some) representative grammars, which means that any two grammars in S are mutually distinct. See Figure 1 for examples of distinct grammars for the same string.

The *size* of a grammar G, denoted by $|G|$, is the number of productions in G. We denote by $g^*(w)$ the size of the smallest grammar of string w. Further, we denote by $\mathsf{Opt}(w)$ the set of all the smallest grammars of string w. While computing $g^*(w)$ for a given string w is NP-hard in general [10], a number of practical algorithms which run in polynomial-time and construct small grammars of w have been proposed.

In this paper, we focus on *RePair* [30], which is the best known grammar-based compressor that produce small grammars in practice. We briefly describe the RePair algorithm, which consists of the three stages:

1. Initial stage: All terminal symbols in the input string are replaced with non-terminal symbols. This creates unary productions.
2. Replacement stage: The algorithm picks an *arbitrary* bigram which has the most non-overlapping occurrences in the string, and then replaces all possible occurrences of the bigram with a new non-terminal symbol. The algorithm repeats the same process recursively for the string obtained after the replacement of the bigrams, until no bigrams have two or more non-overlapping occurrences in the string. It is clear that the productions created in the replacement stage are all binary.

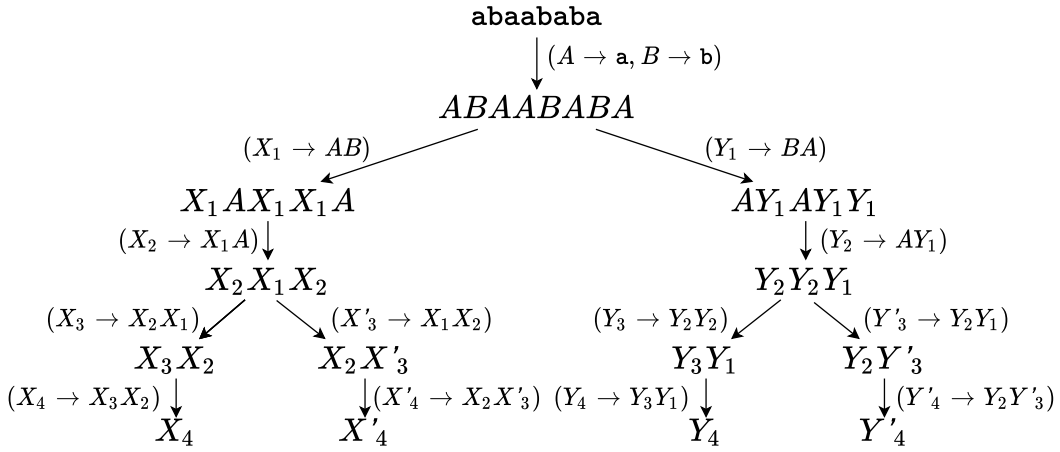

Figure 2 Illustration for the changes of strings and productions to be added when RePair is applied to string $w = \mathtt{abaababa}$. At the second level, the most frequent bigrams in string $ABAABABA$ are AB and BA. If AB is chosen and replaced with non-terminal symbol X_1, the string changes to $X_1AX_1X_1A$ and production $X_1 \to AB$ is added. Otherwise (if BA is chosen and replaced with non-terminal symbol Y_1), the string changes to $AY_1AY_1Y_1$ and production $Y_1 \to BA$ is added. In this example, the size of $\mathsf{RePair}(w)$ is four.

3. Final stage: Trivial binary productions are created from the sequence of non-terminal symbols which are obtained after the last replacement. This ensures that the resulting grammar is in the Chomsky normal form. We remark that when distinct bigrams have the most non-overlapping occurrences in the string, then the choice of the bigrams to replace depends on each implementation of RePair.

A grammar of w obtained by some implementation of RePair is called a *RePair grammar* of w. We denote by $\mathsf{RePair}(w)$ the set of all possible RePair grammars of w. We show an example of RePair grammars in Figure 2.

2.4 LZ-factorization

A sequence $S = (s_1, \ldots, s_m)$ of non-empty strings is called a *factorization* of string w if $w = s_1 \cdots s_m$. Each s_i $(1 \leq i \leq m)$ is called a *phrase* of S. The *size* of the factorization S, denoted $|S|$, is the number m of phrases in S.

For a factorization $S = (s_1, \ldots, s_m)$ of a string w, we say that the i-th phrase s_i is *greedy* if either s_i is a fresh symbol that occurs for the first time in $s_1 \cdots s_i$, or s_i is the longest prefix of $s_i \cdots s_m$ which occurs in $s_1 \cdots s_{i-1}$. A factorization of a string w is called the *LZ-factorization* of w if all the phrases are greedy. Note that this definition of the LZ-factorization is equivalent to the one in [38]. The LZ-factorization of string w is denoted by $LZ(w)$, and the size of $LZ(w)$ is denoted by $z(w)$. We sometimes represent a factorization $(s_1, s_2, \ldots, s_m)$ of w by $s_1|s_2|\ldots|s_m$, where each $|$ denotes the *boundary* of the phrases. For example, The LZ-factorization of $w = \mathtt{ababaabaaba}$ is $\mathtt{a|b|ab|a|aba|aba}$.

3 Basic Properties of Fibonacci and Related Words

In this section, we show some properties of the aforementioned words. We fix the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ in this section. First, for the summation of Fibonacci sequences, the next equations hold:

▶ **Fact 1.** $\sum_{k=1}^{i} f_{2k-1} = f_{2i}$ *and* $\sum_{k=1}^{i} f_{2k} = f_{2i+2} - 1$.

By the definitions of F_n, P_n, and Q_n, we have the following observation:

▶ **Lemma 2.** *For each $k \geq 2$, the most frequent bigrams of F_{2k} are* ab *and* ba*, and the most frequent bigram of F_{2k-1} is* ab*. Also, for each $i \geq 2$ and each $j \geq 3$, the most frequent bigram of P_i and Q_j is* ab*.*

Proof. From the fact that bigram bb and trigram aaa do not occur in any Fibonacci word (e.g., see [3]), we can see that any occurrence of aa is followed by b in Fibonacci words. Thus, aa cannot occur more frequently than ab in any Fibonacci word. Also, since the third and subsequent Fibonacci words start with ab, bigram ab occurs more frequently than aa. Additionally, since all the Fibonacci words F_{2k} of even order end with b and all the Fibonacci words F_{2k-1} of odd order end with a, the statements for the Fibonacci words hold.

Similarly, as for string P_i, it follows from the definition of morphism π that aa does not occur in P_i. Also, bb always succeeds a, and thus, bb cannot occur more frequently than ab. Furthermore, by the definition of morphism π, string P_i starts with aba and ends with b for $i \geq 3$. Thus, the most frequent bigram of P_i is ab for $i \geq 3$ (note that P_2 is trivial).

Finally, as for string Q_j, it follows from the definition of morphism θ that bb does not occur in Q_j. Also, aa always precedes b, and thus, aa cannot occur more frequently than ab. Furthermore, by the definition of morphism θ, string Q_j ends with ab for $j \geq 3$. Thus, the most frequent bigram of Q_j is ab for $j \geq 3$. ◀

A factorization $C = (c_1, \ldots, c_m)$ of a string w is called the *C-factorization* of w if either c_i is a fresh symbol or c_i is the longest prefix of $c_i \cdots c_m$ which occurs twice in $c_1 \cdots c_i$. We can obtain the full characterization of the LZ-factorization of F_n immediately from the C-factorization of F_n, as follows:

▶ **Lemma 3.** *The LZ-factorization of F_n is* $(\mathtt{a}, \mathtt{b}, \mathtt{a}, F_4^R, \ldots, F_{n-2}^R, s)$*, where $s =$* ab *if n is odd, and $s =$* ba *otherwise.*

Proof. It is shown in [4] that the C-factorization of the infinite Fibonacci word $\mathbf{F}$ is $(\mathtt{a}, \mathtt{b}, \mathtt{a}, F_4^R, F_5^R, \ldots)$. Also, for each $i \geq 4$, the (only) reference source of each factor F_i^R is the substring of F_n of length f_i ending at just before the factor, i.e., the source does not overlap with the factor. From these facts, it can be seen that the LZ-factorization of $\mathbf{F}$ is the same as the C-factorization of $\mathbf{F}$. Then, the last phrase of the C-factorization of a finite Fibonacci word is of length two since $f_n = \sum_{i=1}^{n-2} f_i + 1 = (1 + 1 + 2 + \sum_{i=4}^{n-2} f_i) + 1 = 3 + \sum_{i=4}^{n-2} f_i + 2$. Also, since the Fibonacci words of odd order (resp. even order) end with ab (resp. ba), the last phrase is ab (resp. ba). ◀

The next lemma states that P_i and Q_i are the right-rotations of Fibonacci words.

▶ **Lemma 4.** *For each $i \geq 1$, $P_i^{(\mathtt{a},\mathtt{b})}$ is the right-rotation of $F_{2i-1}^{(\mathtt{b},\mathtt{a})}$, and $Q_i^{(\mathtt{a},\mathtt{b})}$ is the right-rotation of $F_{2i}^{(\mathtt{a},\mathtt{b})}$.*

Proof. The next claim can be proven by induction:

▷ Claim 5. For any non-empty string $x \in \{\mathtt{a}, \mathtt{b}\}$, $(\phi^{(\mathtt{b},\mathtt{a})})^2(x)\mathtt{b} = \mathtt{b}\pi^{(\mathtt{a},\mathtt{b})}(x)$ and $(\phi^{(\mathtt{a},\mathtt{b})})^2(x)\mathtt{ab} = \mathtt{ab}\theta^{(\mathtt{a},\mathtt{b})}(x)$ hold.

We prove the lemma by using Claim 5. Assume that the lemma holds for i. Since the last symbol of $F_{2i-1}^{(\mathtt{b},\mathtt{a})}$ is a, we can write $F_{2i-1}^{(\mathtt{b},\mathtt{a})} = x\mathtt{a}$ with some string x. From the induction hypothesis, $P_i^{(\mathtt{a},\mathtt{b})} = \mathtt{a}x$ holds. Then, $P_{i+1}^{(\mathtt{a},\mathtt{b})} = \pi^{(\mathtt{a},\mathtt{b})}(\mathtt{a}x) = \mathtt{ab}\pi^{(\mathtt{a},\mathtt{b})}(x)$. Also, $F_{2i+1}^{(\mathtt{b},\mathtt{a})} =$

$$P_i = \mathtt{a}|\mathtt{b}|\mathtt{a}\ \mathtt{b}|\mathtt{b}\ \mathtt{a}\ \mathtt{b}|\mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}|\mathtt{b}\ \mathtt{a}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}|\cdots\ |(F_k^{(\mathtt{b},\mathtt{a})})^R|\ \cdots\mathtt{b}|\mathtt{b}$$

$$F_{2i-1}^{(\mathtt{b},\mathtt{a})} = \mathtt{b}|\mathtt{a}|\mathtt{b}|\mathtt{b}\ \mathtt{a}\ \mathtt{b}|\mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}|\mathtt{b}\ \mathtt{a}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}\ \mathtt{a}\ \mathtt{b}|\cdots\ |(F_k^{(\mathtt{b},\mathtt{a})})^R|\ \cdots\mathtt{b}|\mathtt{b}\ \mathtt{a}$$

Figure 3 Illustration for the LZ-factorizations of P_i and $F_{2i-1}^{(\mathtt{b},\mathtt{a})}$. $F_{2i-1}^{(\mathtt{b},\mathtt{a})}$ is aligned so that the first position of $F_{2i-1}^{(\mathtt{b},\mathtt{a})}$ is the second of P_i. In both of two factorizations, the k-th phrase is $(F_k^{(\mathtt{b},\mathtt{a})})^R$ for each $k \geq 4$.

$(\phi^{(\mathtt{b},\mathtt{a})})^2(x\mathtt{a}) = (\phi^{(\mathtt{b},\mathtt{a})})^2(x)\mathtt{ba} = \mathtt{b}\pi^{(\mathtt{a},\mathtt{b})}(x)\mathtt{a}$ by Claim 5, and thus, $P_{i+1}^{(\mathtt{a},\mathtt{b})}$ is the right-rotation of $F_{2i+1}^{(\mathtt{b},\mathtt{a})}$. Similarly, since the last symbol of $F_{2i}^{(\mathtt{a},\mathtt{b})}$ is $\mathtt{a}$, we can write $F_{2i}^{(\mathtt{a},\mathtt{b})} = y\mathtt{a}$ with some string y. From the induction hypothesis, $Q_i^{(\mathtt{a},\mathtt{b})} = \mathtt{a}y$ holds. Then, $Q_{i+1}^{(\mathtt{a},\mathtt{b})} = \theta^{(\mathtt{a},\mathtt{b})}(\mathtt{a}y) = \mathtt{aab}\theta^{(\mathtt{a},\mathtt{b})}(y)$. Also, $F_{2i+2}^{(\mathtt{a},\mathtt{b})} = (\phi^{(\mathtt{a},\mathtt{b})})^2(y\mathtt{a}) = (\phi^{(\mathtt{a},\mathtt{b})})^2(y)\mathtt{aba} = \mathtt{ab}\theta^{(\mathtt{a},\mathtt{b})}(y)\mathtt{a}$ by Claim 5, and thus, $Q_{i+1}^{(\mathtt{a},\mathtt{b})}$ is the right-rotation of $F_{2i+2}^{(\mathtt{a},\mathtt{b})}$. ◀

The LZ-factorizations of P_n is as follows:

▶ **Lemma 6.** *Let $i \geq 2$. The j-th phrase of $LZ(P_i^{(\mathtt{a},\mathtt{b})})$ is $(F_j^{(\mathtt{b},\mathtt{a})})^R$ for each j with $1 \leq j \leq 2i-3$. The last phrase is the $(2i-2)$-th phrase, and that is* $\mathtt{b}$.

Proof. The first three phrases of $LZ(P_i)$ are $(\mathtt{a}, \mathtt{a}, \mathtt{ab}) = ((F_1^{(\mathtt{b},\mathtt{a})})^R, (F_2^{(\mathtt{b},\mathtt{a})})^R, (F_3^{(\mathtt{b},\mathtt{a})})^R)$. By Lemma 4, $P_i[2..|P_i|]\mathtt{a} = F_{2i-1}^{(\mathtt{b},\mathtt{a})}$. Namely, $LZ(P_i[2..|P_i|]\mathtt{a}) = LZ(F_{2i-1}^{(\mathtt{b},\mathtt{a})})$. By Lemma 3, for each phrase of length at least four of $LZ(F_{2i-1}^{(\mathtt{b},\mathtt{a})})$, the length-3 prefix of the phrase is either $\mathtt{abb}$ or $\mathtt{bab}$. Since P_i starts with $\mathtt{aba}$, for each $k \geq 4$, the k-th phrase of $LZ(P_i\mathtt{a})$ equals that of $LZ(P_i[2..|P_i|]\mathtt{a}) = LZ(F_{2i-1}^{(\mathtt{b},\mathtt{a})})$, that is, $(F_k^{(\mathtt{b},\mathtt{a})})^R$ (see also Figure 3). Also, $|LZ(P_i)| = |LZ(F_{2i-1}^{(\mathtt{b},\mathtt{a})})| = 2i-2$ holds, and the last phrase of $LZ(P_i)$ is $\mathtt{b}$. ◀

4 RePair Grammars of Fibonacci Words

In this section, we first show a lower bound of the size of the smallest grammar of any string, which is slightly tighter than the well-known result shown by Rytter [38]. Second, we show that the size of RePair grammars of Fibonacci words are always the smallest.

4.1 Tighter Lower Bound of Smallest Grammar Size

The *partial derivation tree* $\mathcal{PT}(G)$ of a grammar G is the maximal subgraph of the derivation tree of G such that for each non-leaf node v in $\mathcal{PT}(G)$, there is no node whose label is the same as v to its left. For a grammar G of a string w, the *g-factorization* of w w.r.t. G, denoted by $gfact(G)$, is the factorization of w where the phrases correspond to the leaves of $\mathcal{PT}(G)$. See also Figure 4 for an example of $\mathcal{PT}(w)$ and $gfact(G)$. It was shown in [38] that $|gfact(G)| \leq |G|$ holds for any grammar G. We show a slightly tighter lower bound of $|G|$ by considering the number of distinct symbols in w.

▶ **Lemma 7.** *For any grammar G of a string w, $|gfact(G)| - 1 + \sigma_w \leq |G|$.*

Proof. A grammar G in the Chomsky normal form consists of two types of productions:
Type 1 $A \to BC$ where A, B, and C are non-terminal symbols.
Type 2 $A \to \alpha$ where A is a non-terminal symbol and α is a terminal symbol.

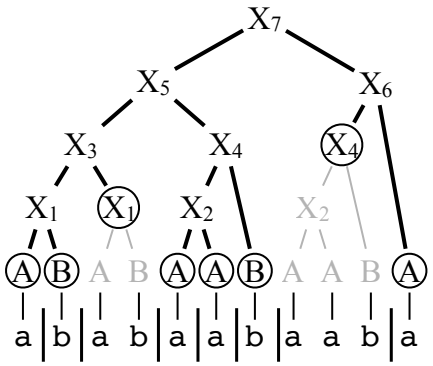

Figure 4 Illustration for $\mathcal{PT}(G)$ of grammar G for string $w = \texttt{ababaabaaba}$. The circled nodes are leaves of $\mathcal{PT}(G)$. For this grammar G of w, $gfact(G) = \texttt{a|b|ab|a|a|b|aab|a}$. Since $|G| = 9$, $|gfact(G)| = 8$, and $\sigma_w = 2$, we can see that Lemma 7 holds for this example.

Let g_1 and g_2 be the numbers of productions of Type 1 and Type 2, respectively. By the definition of $\mathcal{PT}(G)$, the labels of all non-leaf nodes are distinct, and they correspond to the productions of Type 1. Thus, the number m of non-leaf nodes is at most g_1. Also, $\sigma_w \leq g_2$ always holds. Hence, $m + \sigma_w \leq g_1 + g_2 = |G|$. On the other hand, $m = |gfact(G)| - 1$ holds since $\mathcal{PT}(G)$ is a full binary tree and the number of leaves of $\mathcal{PT}(G)$ is $|gfact(G)|$. Therefore, $|gfact(G)| - 1 + \sigma_w \leq |G|$ holds. ◀

We obtain the following tighter lower bound for the size of the smallest grammar(s):

▶ **Theorem 8.** *For any string w, $z(w) - 1 + \sigma_w \leq g^*(w)$ holds.*

Proof. It was shown in [38] that $z(w) \leq |gfact(G)|$ for any grammar G of w. Thus, combining it with Lemma 7, we obtain the theorem. ◀

By regarding the recursive definition of F_n as a grammar, we can construct a size-n grammar of F_n. Also, $g^\star(F_n)$ is at least n by Theorem 8 since $z(F_n) = n - 1$ and $\sigma_{F_n} = 2$. Thus, we obtain the following corollary:

▶ **Corollary 9.** *The smallest grammar size of F_n is n.*

4.2 RePair Grammars are Smallest for Fibonacci Words

By considering the inverse of morphism ϕ, we have the next observation:

▶ **Observation 10.** *By replacing all occurrences of* $\texttt{ab}$ *in* $F_i^{(\texttt{a},\texttt{b})}$ *with* X*, we obtain* $F_{i-1}^{(X,\texttt{a})}$.

The next lemma shows how F, P, and Q can be obtained from one of the others:

▶ **Lemma 11.** $\xi_{\texttt{b}\to\texttt{ba}}(P_i) = F_{2i}$, $\xi_{\texttt{b}\to\texttt{ab}}(P_i) = Q_i$, *and* $\xi_{\texttt{a}\to\texttt{ab}}(Q_i) = P_{i+1}$ *hold.*

Proof. Let $\psi_1 = \xi_{\texttt{b}\to\texttt{ba}}$, $\psi_2 = \xi_{\texttt{b}\to\texttt{ab}}$, and $\psi_3 = \xi_{\texttt{a}\to\texttt{ab}}$. First, we consider compositions of these morphisms. Since $\phi^2(\psi_1(\texttt{a})) = \phi^2(\texttt{a}) = \phi(\texttt{ab}) = \texttt{aba}$, $\phi^2(\psi_1(\texttt{b})) = \phi^2(\texttt{ba}) = \phi(\texttt{aab}) = \texttt{ababa}$, $\psi_1(\pi(\texttt{a})) = \psi_1(\texttt{ab}) = \texttt{aba}$, and $\psi_1(\pi(\texttt{b})) = \psi_1(\texttt{abb}) = \texttt{ababa}$, we have $\phi^2 \circ \psi_1 = \psi_1 \circ \pi$. Also, since $\psi_2(\pi(\texttt{a})) = \psi_2(\texttt{ab}) = \texttt{aab}$, $\psi_2(\pi(\texttt{b})) = \psi_2(\texttt{abb}) = \texttt{aabab}$, $\theta(\psi_2(\texttt{a})) = \theta(\texttt{a}) = \texttt{aab}$, and $\theta(\psi_2(\texttt{b})) = \theta(\texttt{ab}) = \texttt{aabab}$, we have $\psi_2 \circ \pi = \theta \circ \psi_2$. Also, since $\psi_3(\theta(\texttt{a})) = \psi_3(\texttt{aab}) = \texttt{ababb}$, $\psi_3(\theta(\texttt{b})) = \psi_3(\texttt{ab}) = \texttt{aab}$, $\pi(\psi_3(\texttt{a})) = \pi(\texttt{ab}) = \texttt{ababb}$, and $\pi(\psi_3(\texttt{b})) = \pi(\texttt{b}) = \texttt{abb}$, we have $\psi_3 \circ \theta = \pi \circ \psi_3$.

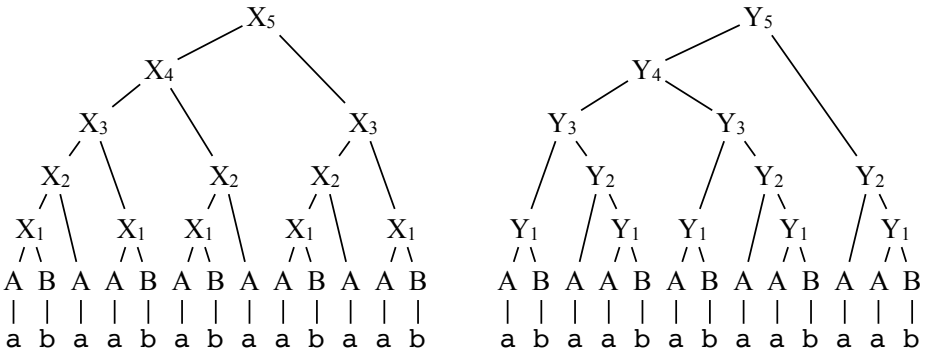

Figure 5 Two RePair grammars of the 7-th Fibonacci word abaababaabaab over {a, b}.

When $i = 1$, the lemma clearly holds. We assume that the lemma holds for $i - 1$ with $i \geq 2$. Then, $\psi_1(P_i) = \psi_1(\pi(P_{i-1})) = \phi^2(\psi_1(P_{i-1})) = \phi^2(F_{2i-2}) = F_{2i}$, $\psi_2(P_i) = \psi_2(\pi(P_{i-1})) = \theta(\psi_2(P_{i-1})) = \theta(Q_{i-1}) = Q_i$, and $\psi_3(Q_i) = \psi_3(\theta(Q_{i-1})) = \pi(\psi_3(Q_{i-1})) = \pi(P_i) = P_{i+1}$. ◀

By considering the inverses of the three morphisms in Lemma 11, we have the next corollary:

▶ **Corollary 12.** *Let X denote a fresh non-terminal symbol. By replacing all occurrences of* ba *in $F_{2i}^{(\mathtt{a},\mathtt{b})}$ with X, we obtain $P_i^{(\mathtt{a},X)}$. By replacing all occurrences of* ab *in $Q_i^{(\mathtt{a},\mathtt{b})}$ with X, we obtain $P_i^{(\mathtt{a},X)}$. By replacing all occurrences of* ab *in $P_{i+1}^{(\mathtt{a},\mathtt{b})}$ with X, we obtain $Q_i^{(X,\mathtt{b})}$.*

We show examples of two RePair grammars of $F_7^{(\mathtt{a},\mathtt{b})}$ in Figure 5.

We are ready to clarify the shape of all the RePair grammars of Fibonacci words.

▶ **Lemma 13.** *The size of every RePair grammar of F_n is n, i.e.,* $\mathsf{RePair}(F_n) \subseteq \mathsf{Opt}(F_n)$. *Also,* $|\mathsf{RePair}(F_n)| = 2\lfloor n/2 \rfloor - 2$.

Proof. By Lemma 2, Observation 10 and Corollary 12, each string, that appears while (an implementation of) the RePair algorithm is running, is one of F, P, and Q over some binary alphabet. The change of the strings can be represented by a directed graph (V, E) such that $V = \{F_i \mid 4 \leq i \leq n\} \cup \{P_i \mid 3 \leq i \leq \lfloor n/2 \rfloor\} \cup \{Q_i \mid 2 \leq i \leq \lfloor n/2 \rfloor - 1\}$ and $E = \{(F_i, F_{i-1}) \mid 5 \leq i \leq n\} \cup \{(F_{2k}, P_k) \mid 3 \leq k \leq \lfloor n/2 \rfloor\} \cup \{(P_i, Q_{i-1}) \mid 3 \leq i \leq \lfloor n/2 \rfloor\}$. See Figure 6 for an illustration of the graph. Each edge represents a replacement of all occurrences of a most frequent bigram, and thus each path from source (F_n) to sinks (F_4 and Q_2) corresponds to a RePair grammar of F_n. The size of a RePair grammar is the number of edges in its corresponding path plus *four*, since the size of a minimal[5] grammar of length-3-binary string, such as F_4 and Q_2, is four. Since the number of edges in any source-to-sinks paths is $n - 4$, the size of each RePair grammar of the n-th Fibonacci word is n. Also, the number of the RePair grammars is *twice* the number of distinct source-to-sinks paths since there are exactly two possible minimal grammars of any length-3 string.

Next, let us count the number of source-to-sinks paths in the graph. There is only one path from F_n to F_4, and there are $\lfloor n/2 \rfloor - 2$ edges from F_{2k} on the upper part to P_k on the lower part for all k with $3 \leq k \leq \lfloor n/2 \rfloor$. Thus, the number of distinct source-to-sinks paths is $\lfloor n/2 \rfloor - 1$. Therefore, the number of distinct RePair grammars is $2\lfloor n/2 \rfloor - 2$. ◀

[5] This means that there are no redundant productions.

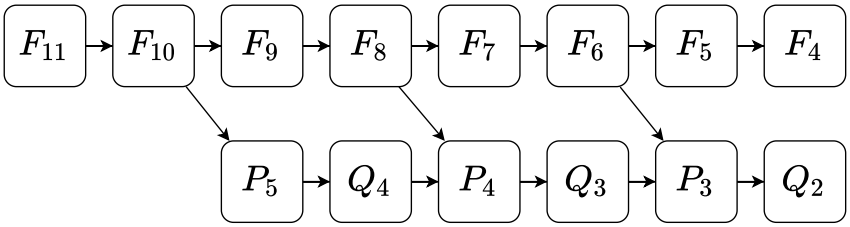

Figure 6 An example of the graph for $n = 11$ described in Lemma 13.

5 Optimality of RePair for Fibonacci Words

In this section, we prove our main theorem:

▶ **Theorem 14.** $\mathsf{Opt}(F_n) = \mathsf{RePair}(F_n)$.

The derivation tree of any grammar (i.e., SLP) G is a full binary tree. Thus, there exists a *bottom-up* algorithm which constructs the grammar G by replacing bigrams with a non-terminal symbol one by one. Thus, it suffices to consider all such algorithms in order to show the optimality of RePair for F_n. We show that any bigram-replacement that does not satisfy the condition of RePair always produces a larger grammar than the RePair grammars. For F_n, P_n, and Q_n, there are 16 strategies that do not satisfy the condition of RePair:

Bigram to replace	aa		ab		ba		bb	
(all/not all of them)	all	not all	all	not all	all	not all	all	not all
F_{2k}	2	3	RePair	1	RePair	4	-	-
F_{2k+1}			RePair		5	6	-	-
P_n	-	-	RePair	7	8	9	10	11
Q_n	15	16	RePair	12	13	14	-	-

The case numbers (1–16) are written inside their corresponding cells in the table. Each hyphen shows the case where the bigram does not occur in the string, which therefore does not need to be considered.

In order to show the non-optimality of each of the above strategies, we utilize the sizes of LZ-factorizations which are lower bounds of the sizes of grammars. Let R be the string obtained by replacing occurrences of a bigram in F_n with a non-terminal symbol X by one of the above 16 strategies. We will show that $z(R) \geq n - 1$ holds for each case. Then, by Theorem 8, the size of the corresponding grammar of F_n becomes at least $(z(R) + |\{X\}| - 1) + \sigma_{F_n} \geq (n-1) + 2 = n + 1$, i.e., that is not the smallest by Corollary 9.

To compare the LZ-factorizations between two strings transformed from the same string F_n, we treat the boundaries as if they are on F_n.

5.1 Non-optimality of Strategies for F_n

We first define a *semi-greedy* factorization $SG(w)$ of string w which will be used in the proof for the first three cases. Let $SG(w)$ be the factorization of w obtained by shifting each boundary of $LZ(w)$ except the ones whose left phase is of length 1 to the left by one. For example, $SG(F_7) = \mathtt{a|b|a|ab|abaab|aab}$ since $LZ(F_7) = \mathtt{a|b|a|aba|baaba|ab}$. Clearly, $|SG(F_n)| = |LZ(F_n)| = n - 1$. By the definition of $SG(F_n)$ and properties of $LZ(F_n)$ (cf. [4, 14]), the following claim holds:

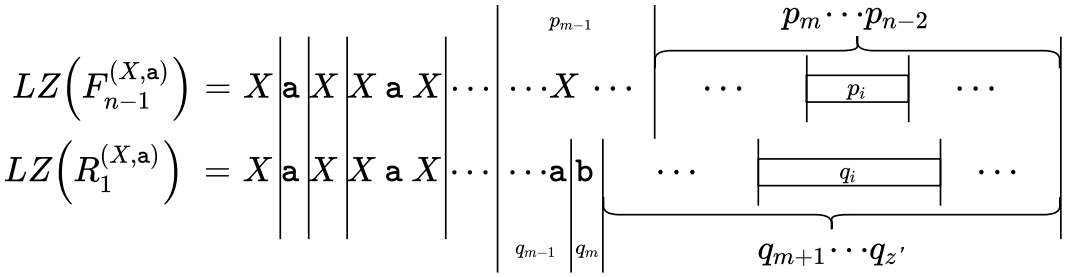

Figure 7 Illustration for contradiction of $LZ(F_{n-1}^{(X,\mathtt{a})}) = (p_1,\ldots,p_{n-2})$ and $LZ(R_1^{(X,\mathtt{a},\mathtt{b})}) = (q_1,\ldots,q_{z'})$ for Case (1). Note that the scale of this figure is based on the length of $F_n \in \{\mathtt{a},\mathtt{b}\}^\star$, not the lengths of phrases.

▷ Claim 15. Let $SG(F_n) = (p_1,\ldots,p_{n-1})$ for $n \geq 5$. The following statements hold:

- The first four phrases are $(p_1,p_2,p_3,p_4) = (\mathtt{a},\mathtt{b},\mathtt{a},\mathtt{ab})$.
- For each i with $5 \leq i \leq n-2$, p_i is the right-rotation of F_i^R and it is a greedy phrase.
- The last phase is $p_{n-1} = \mathtt{aba}$ if n is even, and $p_{n-1} = \mathtt{aab}$ otherwise.
- Each boundary of $SG(F_n)$, except the first and third ones, divides an occurrence of $\mathtt{ba}$.

Case (1): Replacing some but not all the occurrences of ab in F_n

Recall that $F_{n-1}^{(X,\mathtt{a})}$ is obtained by replacing all the occurrences of $\mathtt{ab}$ in $F_n^{(\mathtt{a},\mathtt{b})}$ with X. Let $R_1^{(X,\mathtt{a},\mathtt{b})}$ be any string obtained by replacing some but *not all* the occurrences of $\mathtt{ab}$ in $F_n^{(\mathtt{a},\mathtt{b})}$ with X. Let $LZ(F_{n-1}^{(X,\mathtt{a})}) = (p_1,\ldots,p_{n-2})$ and $LZ(R_1^{(X,\mathtt{a},\mathtt{b})}) = (q_1,\ldots,q_{z'})$ where $z' = |LZ(R_1^{(X,\mathtt{a},\mathtt{b})})|$. See Figure 7 for illustration. The first mismatch of boundaries between two factorizations is the position of the first occurrence of $\mathtt{b}$ in $R_1^{(X,\mathtt{a},\mathtt{b})}$. Since the $\mathtt{b}$ is a fresh symbol, it is a length-1 phrase. Suppose that this length-1 phrase is the m-th phrase ($m \geq 2$) in $LZ(R_1^{(X,\mathtt{a},\mathtt{b})})$. Then, $\xi_{X\to\mathtt{ab}}(p_{m-1}\cdots p_{n-2}) = \xi_{X\to\mathtt{ab}}(q_{m-1}\cdots q_{z'})$ holds. The next corollary follows from Claim 15:

▶ **Corollary 16.** *The factorization $\xi_{X\to\mathtt{ab}}(LZ(F_{n-1}^{(X,\mathtt{a})}))$ of $F_n^{(\mathtt{a},\mathtt{b})}$ is the same as $SG(F_n^{(\mathtt{a},\mathtt{b})})$ except the first phrase. In other words, for each i with $2 \leq i \leq n-2$, $\xi_{X\to\mathtt{ab}}(p_i)$ is the $(i+1)$-th phrase of $SG(F_n^{(\mathtt{a},\mathtt{b})})$.*

From the greediness of $\xi_{X\to\mathtt{ab}}(p_{m-1})$ in $SG(F_n^{(\mathtt{a},\mathtt{b})})$, p_{m-1} is not shorter than q_{m-1}. Thus, $\xi_{X\to\mathtt{ab}}(p_m\cdots p_{n-2})$ is not longer than $\xi_{X\to\mathtt{ab}}(q_{m+1}\cdots q_{z'})$. For the sake of contradiction, we assume that $z' < n-1$. Then, $z'-(m+1)+1 < (n-2)-m+1$ holds, and hence, there must exist a phrase q_i of $LZ(R_1^{(X,\mathtt{a},\mathtt{b})})$ and a phrase p_j of $LZ(F_{n-1}^{(\mathtt{a},\mathtt{b})})$ such that $\xi_{X\to\mathtt{ab}}(q_i)$ contains $\xi_{X\to\mathtt{ab}}(p_j)$ and their ending positions in $F_n^{(\mathtt{a},\mathtt{b})}$ are different. This contradicts the greediness of the phrase $\xi_{X\to\mathtt{ab}}(p_j)$ of $SG(F_n^{(\mathtt{a},\mathtt{b})})$ on $F_n^{(\mathtt{a},\mathtt{b})}$. Therefore, $|LZ(R_1^{(X,\mathtt{a},\mathtt{b})})| = z' \geq n-1$.

Basically, most of the remaining cases can be proven by similar argumentations, however, we will write down the details because there are a few differences.

Case (2): Replacing all the occurrences of aa in F_n

Let $R_2^{(X,\mathtt{a},\mathtt{b})}$ be the string obtained by replacing all the occurrences of $\mathtt{aa}$ in $F_n^{(\mathtt{a},\mathtt{b})}$ with X. The next corollary holds from Claim 15 (see also Figure 8 for a concrete example):

▶ **Corollary 17.** *The factorization $\xi_{X\to\mathtt{aa}}(LZ(R_2^{(X,\mathtt{a},\mathtt{b})}))$ of $F_n^{(\mathtt{a},\mathtt{b})}$ is the same as $SG(F_n^{(\mathtt{a},\mathtt{b})})$ except the first four phrases. In other words, for each i with $5 \leq i \leq n-1$, $\xi_{X\to\mathtt{aa}}(p_i)$ is the i-th phrase of $SG(F_n^{(\mathtt{a},\mathtt{b})})$, where p_i is the i-th phrase of $LZ(R_2^{(X,\mathtt{a},\mathtt{b})})$.*

Thus, $|LZ(R_2^{(X,\mathtt{a},\mathtt{b})})| = |SG(F_n^{(\mathtt{a},\mathtt{b})})| = n-1$.

$SG(F_9) =$ a|b|a|a b|a b a a b|a a b a b a a b|a b a a b a a b a b a a b|a a b

$LZ(R_2) =$ a|b| X |b|a b X b| X b a b X b|a b X b X b a b X b| X b

Figure 8 Two factorizations $SG(F_9)$ and $LZ(R_2)$.

$LZ(F_8) =$ a|b|a|a b a|b a|b a a b a|b a b a a b a|b a|b a

$LZ(P_4) =$ a| X |a| X | X a X |a X X a X | X

Figure 9 Two factorizations $LZ(F_8)$ and $LZ(P_4)$.

Case (3): Replacing some but not all the occurrences of aa in F_n

Let $R_3^{(X,\mathtt{a},\mathtt{b})}$ be any string obtained by replacing some but not all the occurrences of aa in $F_n^{(\mathtt{a},\mathtt{b})}$ with X. Let $LZ(R_2^{(X,\mathtt{a},\mathtt{b})}) = (p_1, \ldots, p_{n-1})$ and $LZ(R_3^{(X,\mathtt{a},\mathtt{b})}) = (q_1, \ldots, q_{z'})$ where $z' = |LZ(R_3^{(X,\mathtt{a},\mathtt{b})})|$. We omit the superscripts in the following. The first mismatch of boundaries between $LZ(R_2)$ and $LZ(R_3)$ is the position of the first occurrence of aa in R_3. Since this is the first occurrence of aa, there has to be a boundary between the two a's. Suppose that the phrase that starts with the second a is the m-th phrase ($m \geq 4$) in $LZ(R_3)$. By Corollary 17, p_{m-1} is not shorter than q_{m-1}. Thus, $\xi_{X\to\mathtt{aa}}(q_m \cdots q_{z'})$ is longer than $\xi_{X\to\mathtt{aa}}(p_m \cdots p_{n-1})$. For the sake of contradiction, we assume that $z' < n-1$. Then, $z' - m + 1 < (n-1) - m + 1$ holds, and hence, there exist phrases q_i of $LZ(R_3)$ and p_j of $LZ(R_2)$ such that $\xi_{X\to\mathtt{aa}}(q_i)$ contains $\xi_{X\to\mathtt{aa}}(p_j)$ and their ending positions in F_n are different. This contradicts the greediness of the phrase $\xi_{X\to\mathtt{aa}}(p_j)$ of $SG(F_n)$. Therefore, $|LZ(R_3)| = z' \geq n-1$.

Case (4): Replacing some but not all the occurrences of ba in F_{2k}

Recall that $P_k^{(\mathtt{a},X)}$ is obtained by replacing all the occurrences of ba in $F_{2k}^{(\mathtt{a},\mathtt{b})}$ with X. Let $R_4^{(X,\mathtt{a},\mathtt{b})}$ be any string obtained by replacing some but not all the occurrences of ba in $F_{2k}^{(\mathtt{a},\mathtt{b})}$ with X. Let $LZ(P_k^{(\mathtt{a},X)}) = (p_1, \ldots, p_{2k-2})$ and $LZ(R_4^{(X,\mathtt{a},\mathtt{b})}) = (q_1, \ldots, q_{z'})$ where $z' = |LZ(R_4^{(X,\mathtt{a},\mathtt{b})})|$. Since the only boundary in $LZ(F_{2k}^{(\mathtt{a},\mathtt{b})})$ that divides an occurrence of ba is the second one, the next holds for the LZ-factorization of $P_k^{(\mathtt{a},X)}$ (see also Figure 9 for a concrete example):

▶ **Corollary 18.** *The factorization $\xi_{X\to\mathtt{ba}}(LZ(P_k^{(\mathtt{a},X)}))$ of $F_n^{(\mathtt{a},\mathtt{b})}$ is the same as $LZ(F_{2k}^{(\mathtt{a},\mathtt{b})})$ except the first three phrases. In other words, for each i with $4 \leq i \leq 2k-2$, $\xi_{X\to\mathtt{ba}}(p_i)$ is the $(i+1)$-th phrase of $LZ(F_{2k}^{(\mathtt{a},\mathtt{b})})$.*

We omit the superscripts in the following. The first mismatch of boundaries between two factorizations is the position of the first occurrence of b in R_4. Since the b is a fresh symbol, it is a length-1 phrase. Let the length-1 phrase be the m-th phrase in $LZ(R_4)$. Then, $\xi_{X\to\mathtt{ba}}(q_m \cdots q_{z'})$ is longer than $\xi_{X\to\mathtt{ba}}(p_m \cdots p_{2k-2})$ by Corollary 18. For the sake of contradiction, we assume that $z' \leq 2k-2$. Then $z' - m + 1 \leq (2k-2) - m + 1$, and hence, there exist phrases q_i of $LZ(R_4)$ and p_j of $LZ(P_k)$ such that $\xi_{X\to\mathtt{ba}}(q_i)$ contains $\xi_{X\to\mathtt{ba}}(p_j)$ and their ending positions in F_{2k} are different. This contradicts that the greediness of phrase $\xi_{X\to\mathtt{ba}}(p_j)$ of $LZ(F_{2k})$, Therefore, $|LZ(R_4)| = z' > 2k-2$.

Case (5): Replacing all the occurrences of ba in F_{2k+1}

Let $R_5^{(X,\mathtt{a},\mathtt{b})}$ be the string obtained by replacing all the occurrences of ba in $F_{2k+1}^{(\mathtt{a},\mathtt{b})}$ with X. Since $F_{2k+1}^{(\mathtt{a},\mathtt{b})}$ ends with b, the last symbol of $R_5^{(X,\mathtt{a},\mathtt{b})}$ is b and it is unique in $R_5^{(X,\mathtt{a},\mathtt{b})}$. We omit the superscripts in the following. Since $F_{2k+1} = F_{2k}F_{2k-2}F_{2k-3}$, P_kP_{k-1} is a prefix of R_5. By Lemma 6, the first $2k-2$ phrases of $LZ(R_5)$ is the same as that of $LZ(P_{k+1})$. Also, the $(2k-1)$-th phrase ends at before b and the $2k$-th phrase is b. Thus, $|LZ(R_5)| = 2k$.

Case (6): Replacing some but not all the occurrences of ba in F_{2k+1}

Let $R_6^{(X,\mathtt{a},\mathtt{b})}$ be any string obtained by replacing some but not all the occurrences of ba in $F_{2k+1}^{(\mathtt{a},\mathtt{b})}$ with X. Let $LZ(R_5^{(X,\mathtt{a},\mathtt{b})}) = (p_1, \ldots, p_{2k})$ and $LZ(R_6^{(X,\mathtt{a},\mathtt{b})}) = (q_1, \ldots, q_{z'})$ where $z' = |LZ(R_6^{(X,\mathtt{a},\mathtt{b})})|$. We omit the superscripts in the following. The first mismatch of boundaries between two factorizations is the position of the first occurrence of b in R_6. Since the b is a fresh symbol, it is a length-1 phrase. Let the length-1 phrase be the m-th phrase in $LZ(R_6)$. Then, $\xi_{X\to\mathtt{ba}}(p_m \cdots p_{2k})$ is not longer than $\xi_{X\to\mathtt{ba}}(q_m \cdots q_{z'})$ by the greediness of $\xi_{X\to\mathtt{ba}}(p_{m-1})$. For the sake of contradiction, we assume that $z' < 2k$. Then $z' - m + 1 < 2k - m + 1$, and hence, there exist phrases q_i of $LZ(R_6)$ and p_j of $LZ(R_5)$ such that $\xi_{X\to\mathtt{ba}}(q_i)$ contains $\xi_{X\to\mathtt{ba}}(p_j)$ and their ending positions in F_{2k+1} are different. This contradicts the greediness of phrase $\xi_{X\to\mathtt{ba}}(p_j)$ of $LZ(F_{2k+1})$. Therefore, $|LZ(R_6)| = z' \geq 2k$.

The proofs for the remaining ten cases can be found in a full version of this paper [35]. We remark that the remaining ten cases can also be proven by similar argumentations.

6 Conclusions

In this paper, we analyzed the smallest grammars of Fibonacci words and completely characterized them by the RePair grammar-compressor. Namely, the set of all smallest grammars that produce only the n-th Fibonacci word F_n equals the set of all grammars obtained by applying (different implementations of) the RePair algorithm to F_n. Further, we showed that the size of the smallest grammars of F_n is n and that the number of such grammars is $2\lfloor n/2 \rfloor - 2$.

To show the smallest grammar size of F_n, we revisited the result on the lower bound of the sizes of grammars shown by Rytter [38]. Here, we gave a slightly tighter lower bound of the grammar size $z(w) - 1 + \sigma_w$ for *any* string w. Independent of the above results on Fibonacci words, this result on a lower bound is interesting since the result will help show the *exact* values of the smallest grammar size of other strings.

It is left as our future work to investigate whether it is possible to characterize the smallest grammars of other binary words, such as Thue-Morse words and Period-doubling words, by similar methods to Fibonacci words.

References

1 Alberto Apostolico and Stefano Lonardi. Compression of biological sequences by greedy off-line textual substitution. In *Data Compression Conference, DCC 2000, Snowbird, Utah, USA, March 28-30, 2000*, pages 143–152. IEEE Computer Society, 2000. doi:10.1109/DCC.2000.838154.

2 Hideo Bannai, Momoko Hirayama, Danny Hucke, Shunsuke Inenaga, Artur Jez, Markus Lohrey, and Carl Philipp Reh. The smallest grammar problem revisited. *IEEE Trans. Inf. Theory*, 67(1):317–328, 2021. doi:10.1109/TIT.2020.3038147.

3 Jean Berstel. *Fibonacci Words – A Survey*, pages 13–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986. doi:10.1007/978-3-642-95486-3_2.

4 Jean Berstel and Alessandra Savelli. Crochemore factorization of sturmian and other infinite words. In Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 157–166. Springer, 2006. doi:10.1007/11821069_14.

5 Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search in grammar-compressed strings. *Theory Comput. Syst.*, 62(8):1715–1735, 2018. doi:10.1007/s00224-017-9839-9.

6 Philip Bille, Travis Gagie, Inge Li Gørtz, and Nicola Prezza. A separation between rlslps and LZ77. *J. Discrete Algorithms*, 50:36–39, 2018. doi:10.1016/j.jda.2018.09.002.

7 Philip Bille, Inge Li Gørtz, and Nicola Prezza. Space-efficient Re-Pair compression. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 171–180. IEEE, 2017. doi:10.1109/DCC.2017.24.

8 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.

9 Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the complexity of the smallest grammar problem over fixed alphabets. *Theory Comput. Syst.*, 65(2):344–409, 2021. doi:10.1007/s00224-020-10013-w.

10 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

11 Francisco Claude and Gonzalo Navarro. Fast and compact web graph representations. *ACM Trans. Web*, 4(4):16:1–16:31, 2010. doi:10.1145/1841909.1841913.

12 Maxime Crochemore. Linear searching for a square in a word. *Bulletin of the European Association of Theoretical Computer Science*, 24:66–72, 1984.

13 Martin Farach and Mikkel Thorup. String matching in Lempel-Ziv compressed strings. *Algorithmica*, 20(4):388–404, 1998. doi:10.1007/PL00009202.

14 Gabriele Fici. Factorizations of the fibonacci infinite word. *J. Integer Seq.*, 18(9):15.9.3, 2015. URL: https://cs.uwaterloo.ca/journals/JIS/VOL18/Fici/fici5.html.

15 Isamu Furuya, Takuya Takagi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Takuya Kida. Practical grammar compression based on maximal repeats. *Algorithms*, 13(4):103, 2020. doi:10.3390/a13040103.

16 Travis Gagie, Tomohiro I, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, and Yoshimasa Takabatake. Rpair: Rescaling RePair with rsync. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval – 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2019. doi:10.1007/978-3-030-32686-9_3.

17 Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. *J. ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.

18 Michal Ganczorz and Artur Jez. Improvements on Re-Pair grammar compressor. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference, DCC 2017, Snowbird, UT, USA, April 4-7, 2017*, pages 181–190. IEEE, 2017. doi:10.1109/DCC.2017.52.

19 Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In Rolf G. Karlsson and Andrzej Lingas, editors, *Algorithm Theory – SWAT '96, 5th Scandinavian Workshop on Algorithm Theory, Reykjavík, Iceland, July 3-5, 1996, Proceedings*, volume 1097 of *Lecture Notes in Computer Science*, pages 392–403. Springer, 1996. doi:10.1007/3-540-61422-2_148.

20 Danny Hucke and Carl Philipp Reh. Approximation ratios of RePair, LongestMatch and Greedy on unary strings. *Algorithms*, 14(2):65, 2021. doi:10.3390/a14020065.

21 Tomohiro I. Longest common extensions with recompression. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

22 Tomohiro I, Wataru Matsubara, Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, Kazuyuki Narisawa, and Ayumi Shinohara. Detecting regularities on grammar-compressed strings. *Inf. Comput.*, 240:74–89, 2015. doi:10.1016/j.ic.2014.09.009.

23 Tomohiro I, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Lyndon factorization algorithms for SLP and LZ78 compressed text. *Theor. Comput. Sci.*, 656:215–224, 2016. doi:10.1016/j.tcs.2016.03.005.

24 Artur Jez. Approximation of grammar-based compression via recompression. *Theor. Comput. Sci.*, 592:115–134, 2015. doi:10.1016/j.tcs.2015.05.027.

25 Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.

26 Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.

27 John C. Kieffer, En-Hui Yang, Gregory J. Nelson, and Pamela C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inf. Theory*, 46(4):1227–1245, 2000. doi:10.1109/18.850665.

28 Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. URL: https://www.worldcat.org/oclc/312898417.

29 Dominik Köppl, Tomohiro I, Isamu Furuya, Yoshimasa Takabatake, Kensuke Sakai, and Keisuke Goto. Re-Pair in small space. *Algorithms*, 14(1):5, 2021. doi:10.3390/a14010005.

30 N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Data Compression Conference, DCC 1999, Snowbird, Utah, USA, March 29-31, 1999*, pages 296–305. IEEE Computer Society, 1999. doi:10.1109/DCC.1999.755679.

31 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

32 Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013. doi:10.1016/j.is.2013.06.006.

33 Takuya Masaki and Takuya Kida. Online grammar transformation based on Re-Pair algorithm. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2016 Data Compression Conference, DCC 2016, Snowbird, UT, USA, March 30 – April 1, 2016*, pages 349–358. IEEE, 2016. doi:10.1109/DCC.2016.69.

34 Wataru Matsubara, Shunsuke Inenaga, Akira Ishino, Ayumi Shinohara, Tomoyuki Nakamura, and Kazuo Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theor. Comput. Sci.*, 410(8-10):900–913, 2009. doi:10.1016/j.tcs.2008.12.016.

35 Takuya Mieno, Shunsuke Inenaga, and Takashi Horiyama. Repair grammars are the smallest grammars for Fibonacci words. *CoRR*, abs/2202.08447, 2022. URL: https://arxiv.org/abs/2202.08447.

36 Gonzalo Navarro and Cristian Urbina. On stricter reachable repetitiveness measures. In Thierry Lecroq and Hélène Touzet, editors, *String Processing and Information Retrieval – 28th International Symposium, SPIRE 2021, Lille, France, October 4-6, 2021, Proceedings*, volume 12944 of *Lecture Notes in Computer Science*, pages 193–206. Springer, 2021. doi:10.1007/978-3-030-86692-1_16.

37 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 – Kraków, Poland*, volume 58 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.72.

38 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

39 James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4):928–951, 1982. doi:10.1145/322344.322346.

40 Toshiya Tanaka, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing convolution on grammar-compressed text. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference, DCC 2013, Snowbird, UT, USA, March 20-22, 2013*, pages 451–460. IEEE, 2013. doi:10.1109/DCC.2013.53.

41 En-Hui Yang and John C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – part one: Without context models. *IEEE Trans. Inf. Theory*, 46(3):755–777, 2000. doi:10.1109/18.841161.

42 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

43 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.