


Edit and Alphabet-Ordering Sensitivity of Lex-Parse

Yuto Nakashima ✉ 

Department of Informatics, Kyushu University, Fukuoka, Japan

Dominik Köppl ✉ 

Department of Computer Science and Engineering, University of Yamanashi, Kofu, Japan
M&D Data Science Center, Tokyo Medical and Dental University, Japan

Mitsuru Funakoshi ✉ 

NTT Communication Science Laboratories, Kyoto, Japan

Shunsuke Inenaga ✉ 

Department of Informatics, Kyushu University, Fukuoka, Japan

Hideo Bannai ✉ 

M&D Data Science Center, Tokyo Medical and Dental University, Japan

Abstract

We investigate the compression sensitivity [Akagi et al., 2023] of lex-parse [Navarro et al., 2021] for two operations: (1) single character edit and (2) modification of the alphabet ordering, and give tight upper and lower bounds for both operations (i.e., we show $\Theta(\log n)$ bounds for strings of length n). For both lower bounds, we use the family of Fibonacci words. For the bounds on edit operations, our analysis makes heavy use of properties of the Lyndon factorization of Fibonacci words to characterize the structure of lex-parse.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Compression sensitivity, Lex-parse, Fibonacci words

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.75

Related Version *Full Version*: <https://arxiv.org/abs/2402.19223>

Funding *Yuto Nakashima*: JSPS KAKENHI Grant Numbers JP21K17705, JP23H04386, and JST ACT-X Grant Number JPMJAX200K

Dominik Köppl: JSPS KAKENHI Grant Numbers JP23H04378

Shunsuke Inenaga: JSPS KAKENHI Grant Numbers JP20H05964, JP22H03551, JP23K24808

Hideo Bannai: JSPS KAKENHI Grant Numbers JP20H04141, JP24K02899

1 Introduction

Dictionary compression is a scheme of lossless data compression that is very effective, especially for highly repetitive text collections. Recently, various studies on dictionary compressors and repetitiveness measures have attracted much attention in the field of stringology (see [25, 26] for a detailed survey).

The *sensitivity* [1] of a string compressor/repetitiveness measure c is defined as the maximum difference in the sizes of c for a text T and for a single-character edited string T' , which can represent the robustness of the compressor/measure w.r.t. small changes/errors of the input string. Akagi et al. [1] gave upper and lower bounds on the worst-case multiplicative sensitivity of various compressors and measures including the Lempel–Ziv parse family [29, 30], the run-length encoded Burrows–Wheeler transform (RLBWT) [4], and the smallest string attractors [17].



© Yuto Nakashima, Dominik Köppl, Mitsuru Funakoshi, Shunsuke Inenaga, and Hideo Bannai; licensed under Creative Commons License CC-BY 4.0

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).

Editors: Rastislav Kráľovič and Antonín Kučera; Article No. 75; pp. 75:1–75:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the other hand, some structures built on strings, including the output of text compressors, can also depend on the order of the alphabet, meaning that a different alphabet ordering can result in a different structure for the same string. Optimization problems of these kinds of structures have recently been studied, e.g., for the RLBWT [2], the RLBWT based on general orderings [12], or the Lyndon factorization [13]. Due to their hardness, efficient exact algorithms/heuristics for minimization have been considered [5, 7, 8, 22, 23].

This paper is devoted to the analysis of the sensitivity of lex-parse. Lex-parse [27] is a greedy left-to-right partitioning of an input text T into phrases, where each phrase starting at position i is $T[i..i + \max\{1, \ell\})$ and ℓ is the longest common prefix between $T[i..n]$ and its lexicographic predecessor $T[i'..n]$ in the set of suffixes of T . Each phrase can be encoded by a pair $(0, T[i])$ if $\ell = 0$, or (ℓ, i') otherwise. By using the lex-parse of size v of a string, we can represent the string with v derivation rules.¹ Lex-parse was proposed as a new variant in a family of ordered parsings that is considered as a generalization of the Lempel–Ziv parsing and a subset of bidirectional macro schemes [28]. We stress that lex-parse can have much fewer factors than the Lempel–Ziv parsing; for instance the number of Lempel–Ziv factors of the k -th Fibonacci word is k while we have only four factors for lex-parse regardless of k (assuming that k is large enough) [27]. Besides having potential for lossless data compression, it helped to gain new insights into string repetitiveness: For instance, a direct relation $v \in O(r)$ between v and the size r of the RLBWT, one of the most important dictionary compressors, holds [27]. Hence, combinatorial studies on lex-parse can lead us to further understanding in string repetitiveness measures and compressors.

The contribution of this paper is twofold. We first consider the sensitivity of lex-parse w.r.t. edit operations, and give tight upper and lower bounds which are logarithmic in the length of the input text. Interestingly, lex-parse is the third measure with super-constant bounds out of (about) 20 measures [1]. Second, we consider a new variant of sensitivity, the alphabet ordering sensitivity (AO-sensitivity) of lex-parse, defined as the maximum difference in the number of phrases of lex-parse between any two alphabet orderings, and give tight upper and lower bounds. For both lower bounds, we use the Fibonacci word. Moreover, we also use properties of the Lyndon factorization [6] for the edit sensitivity to characterize the structure of lex-parse. These insights may be of independent interest. Properties of the Fibonacci word can contribute to the analysis of algorithm complexity. In fact, there are several results regarding lower bounds based on the Fibonacci word (i.e., [9, 14, 16, 27]).

Related work. Inspired by the results of Lagarde and Perifel [19], Akagi et al. [1] pioneered the systematic study of compression sensitivity of various measures w.r.t. edit operations. Giuliani et al. [14] showed an improved lower bound for the additive sensitivity of the run-length BWT. They also use the family of Fibonacci words to obtain their lower bound. Fujimaru et al. [11] presented tight upper and lower bounds for the additive and multiplicative sensitivity of the size of the compact directed acyclic word graph (CDAWG) [3, 10], when edit operations are restricted to the beginning of the text.

2 Preliminaries

Strings

Let Σ be an *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is the string of length 0. Let Σ^+ be the set of non-empty strings, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For any strings x and y , let $x \cdot y$ (or sometimes xy) denote

¹ We stick to the convention to denote the size of lex-parse by v as done in literature such as [27, 26].

the concatenation of the two strings. For a string $w = xyz$, x , y and z are called a *prefix*, *substring*, and *suffix* of w , respectively. They are called a *proper prefix*, a *proper substring*, and a *proper suffix* of w if $x \neq w$, $y \neq w$, and $z \neq w$, respectively. A proper substring that is both a prefix and a suffix of w is also called a *border* of w . The i -th symbol of a string w is denoted by $w[i]$, where $1 \leq i \leq |w|$. For a string w and two integers $1 \leq i \leq j \leq |w|$, let $w[i..j]$ denote the substring of w that begins at position i and ends at position j . For convenience, let $w[i..j] = \varepsilon$ when $i > j$. Also, let $w[..i] = w[1..i]$ and $w[i..] = w[i..|w|]$. For a string w , let $w^1 = w$ and let $w^k = ww^{k-1}$ for any integer $k \geq 2$. A string w is said to be *primitive* if w cannot be written as x^k for any $x \in \Sigma^+$ and integer $k \geq 2$. The following property is well known.

► **Lemma 1** ([20]). *w is primitive if and only if w occurs exactly twice in w^2 .*

For a string w , let $w' = w[1..|w|-1]$, $w'' = w[1..|w|-2]$. A sequence of k strings w_1, \dots, w_k is called a *parsing* or a *factorization* of a string w if $w = w_1 \cdots w_k$. For a binary string w , \bar{w} denotes the bitwise reversed string of w (e.g., $\overline{aab} = baa$ over $\{a, b\}$). Let \prec denote a (strict) total order on an alphabet Σ . A total order \prec on the alphabet induces a total order on the set of strings called the *lexicographic order* w.r.t. \prec , also denoted as \prec , i.e., for any two strings $x, y \in \Sigma^*$, $x \prec y$ if and only if x is a proper prefix of y , or, there exists $1 \leq i \leq \min\{|x|, |y|\}$ s.t. $x[1..i-1] = y[1..i-1]$ and $x[i] \prec y[i]$. For a string w , let SA_w denote the *suffix array* of w , where the i -th entry $SA_w[i]$ stores the index j of the lexicographically i -th suffix $w[j..]$ of w .

Lex-parse

The lex-parse of a string w is a parsing $w = w_1, \dots, w_v$, such that each phrase w_j starting at position $i = 1 + \sum_{k < j} |w_k|$ is $w[i..i + \max\{1, \ell\}]$, where ℓ is the length of the longest common prefix between $w[i..]$ and its lexicographic predecessor $w[i'..]$ in the set of suffixes of w . Each phrase can be encoded by a pair $(0, w[i])$ if $\ell = 0$ (called an *explicit phrase*), or (ℓ, i') otherwise. We will call $w[i'..]$ the *previous suffix* of $w[i..]$. The size (the number of phrases) of the lex-parse of a string w will be denoted by $v(w)$. Note that a phrase starting at position i is explicit if and only if $w[i..]$ is the lexicographically smallest suffix starting with $w[i]$ and thus there are $|\Sigma|$ of them. Let $w = ababbaaba$. The lex-parse of w is aba, b, ba, a, b, a . Since the previous suffix of $w[1..]$ is $w[7..]$ and the longest common prefix between them is aba , the first phrase is aba . In this example, the last two phrases are explicit phrases.

Lyndon factorizations

A string w is a *Lyndon word* [21] w.r.t. a lexicographic order \prec , if and only if $w \prec w[i..]$ for all $1 < i \leq |w|$, i.e., w is lexicographically smaller than all its proper suffixes with respect to \prec . The *Lyndon factorization* [6] of a string w , denoted $LF(w)$, is a unique factorization $\lambda_1^{p_1}, \dots, \lambda_m^{p_m}$ of w , such that each $\lambda_i \in \Sigma^+$ is a Lyndon word, $p_i \geq 1$, and $\lambda_i \succ \lambda_{i+1}$ for all $1 \leq i < m$. A suffix x of w is said to be *significant* if there exists an integer i such that $x = \lambda_i^{p_i} \cdots \lambda_m^{p_m}$ and, for every j satisfying $i \leq j < m$, $\lambda_{j+1}^{p_{j+1}} \cdots \lambda_m^{p_m}$ is a prefix of $\lambda_j^{p_j}$ (cf. [15]). Let $w = bbabbaabaabbaabaabaaba$. The Lyndon factorization of w is $b^2, abb, (aabaabb)^2, aab, a^2$. The suffix a^2 that is the last Lyndon factor is always significant. Since a^2 is a prefix of $(aab)a^2$, $(aab)a^2$ is also significant. Since a^2 and $(aab)a^2$ is a prefix of $(aabaabb)^2(aab)a^2$, $(aabaabb)^2(aab)a^2$ is also significant.

■ **Table 1** Fibonacci words F_k up to $k = 8$.

k	F_k	k	F_k
1	b	2	a
3	ab	4	aba
5	$abaab$	6	$abaababa$
7	$abaababaabaab$	8	$abaababaabaababaababa$

Fibonacci words

The k -th (finite) Fibonacci word F_k over a binary alphabet $\{a, b\}$ is defined as follows: $F_1 = b$, $F_2 = a$, $F_k = F_{k-1} \cdot F_{k-2}$ for any $k \geq 3$ (see also an example in Table 1). Let f_k be the length of the k -th Fibonacci word (i.e., $f_k = |F_k|$).

We also use the infinite Fibonacci word $\mathcal{F} = \lim_{k \rightarrow \infty} F_k$ over an alphabet $\{a, b\}$. We also use $G_k = F_{k-2}F_{k-1}$ which will be useful for representing relations between even and odd Fibonacci words.

► **Lemma 2** (Useful properties of Fibonacci words (cf. [27])). *The following properties hold for every Fibonacci word F_k .*

1. *The length of the longest border of F_k is f_{k-2} .*
2. *F_k has exactly three occurrences of F_{k-2} at position 1, $f_{k-2} + 1$, and $f_{k-1} + 1$ (suffix) for every $k \geq 6$.*
3. *$F_k = G_k[1..f_k - 2] \cdot \overline{G_k[f_k - 1..f_k]}$.*
4. *$G_k = F_k[1..f_k - 2] \cdot \overline{F_k[f_k - 1..f_k]}$.*
5. *For every k , aaa and bb do not occur as substrings in F_k [27, Lemma 36].*
6. *F_k is primitive for every k (we can easily obtain the fact from Property 1).*

The next lemma is also useful for our proof which can be obtained from the above properties.

► **Lemma 3.** *For any $k \geq 8$, F_{k-4} occurs exactly eight times in F_k .*

Proof. From property 2 of Lemma 2, F_k has at least eight occurrences of F_{k-4} (since the suffix occurrence of F_{k-4} in the second F_{k-2} and the prefix occurrence of F_{k-4} in the third F_{k-2} are the same position). Suppose to the contrary that there exists an occurrence of F_{k-4} in F_k that is different from the eight occurrences. From property 2 of Lemma 2, the occurrence crosses the boundary of the first and the second F_{k-2} . Since F_{k-4} is both a prefix and a suffix of F_{k-2} , the occurrence implies that F_{k-4}^2 has F_{k-4} as a substring that is neither a prefix nor a suffix. This contradicts Lemma 1. ◀

Sensitivity of lex-parse

In this paper, we consider two compression sensitivity variants of lex-parse. The first variant is the sensitivity by (single character) edit operations (cf. [1]). For any two strings w_1 and w_2 , let $\text{ed}(w_1, w_2)$ denote the edit distance between w_1 and w_2 . The worst-case multiplicative sensitivity of lex-parse w.r.t. a substitution is defined as follows:

$$\text{MS}_{\text{sub}}(v, n) = \max_{w_1 \in \Sigma^n} \text{MS}_{\text{sub}}(v, w_1),$$

where $MS_{\text{sub}}(v, w_1) = \max\{v(w_2)/v(w_1) \mid w_2 \in \Sigma^n, \text{ed}(w_1, w_2) = 1\}$. $MS_{\text{ins}}(v, n)$ (resp. $MS_{\text{del}}(v, n)$) is defined by replacing the condition $w_2 \in \Sigma^n$ with $w_2 \in \Sigma^{n+1}$ (resp. $w_2 \in \Sigma^{n-1}$). The second variant is the sensitivity by alphabet orderings. For any string w and a lexicographic order \prec , let $v(w, \prec)$ be the size of the lex-parse of w under \prec . We define the alphabet-ordering sensitivity of lex-parse as follows:

$$AOS(v, n) = \max_{w \in \Sigma^n} AOS(v, w),$$

where $AOS(v, w) = \max_{\prec_1, \prec_2 \in A} \{v(w, \prec_2)/v(w, \prec_1)\}$ and A is the set of orderings over Σ .

3 Upper bounds

We first give upper bounds for both operations. We can obtain the following result by using known connections regarding the bidirectional macro scheme and lex-parse.

► **Theorem 4.** $MS_{\text{sub}}(v, n), MS_{\text{ins}}(v, n), MS_{\text{del}}(v, n), AOS(v, n) \in O(\log n)$.

Proof. For any string w , let $b(w)$ be the size of the smallest bidirectional macro scheme [28]. Then, $v(w) \geq b(w)$ holds [27, Lemma 25]. For any two strings w_1 and w_2 with $\text{ed}(w_1, w_2) = 1$, $v(w_2) \in O(b(w_2) \log(n/b(w_2)))$ [27, Theorem 26] and $b(w_2) \leq 2b(w_1)$ [1, Theorem 11] hold. Hence, for $|w_2| \in \Theta(n)$,

$$\frac{v(w_2)}{v(w_1)} \leq \frac{v(w_2)}{b(w_1)} \in O\left(\frac{b(w_2) \log(n/b(w_2))}{b(w_1)}\right) \subseteq O\left(\frac{b(w_1) \log(n/b(w_1))}{b(w_1)}\right) = O(\log(n/b(w_1))).$$

For any alphabet order \prec on Σ , $v(w, \prec) \in O(b(w) \log(n/b(w)))$ and $v(w, \prec) \geq b(w)$ holds. Hence, for any two alphabet orders \prec_1 and \prec_2 ,

$$\frac{v(w, \prec_2)}{v(w, \prec_1)} \leq \frac{v(w, \prec_2)}{b(w)} \in O\left(\frac{b(w) \log(n/b(w))}{b(w)}\right) = O(\log(n/b(w))).$$

These facts imply this theorem. ◀

4 Lower bounds for edit operations

In this section, we give tight lower bounds for edit operations with the family of Fibonacci words.

► **Theorem 5.** $MS_{\text{sub}}(v, n), MS_{\text{ins}}(v, n), MS_{\text{del}}(v, n) \in \Omega(\log n)$.

We devote this section to show $MS_{\text{sub}}(v, n) \in \Omega(\log n)$ since a similar argument can obtain the others. We obtain the claimed lower bound by combining the result of [27] (also in Lemma 20 in Section 5) stating that $v(F_{2k}) = 4$, and the following Theorem 6.

► **Theorem 6.** *For every integer $k \geq 6$, there exists a string w of length f_{2k} such that $\text{ed}(F_{2k}, w) = 1$ and $v(w) = 2k - 2$.*

Let T_{2k} denote the string obtained from F_{2k} by substituting the rightmost b of F_{2k} with a , i.e., $T_{2k} = F_{2k}'' \cdot aa$. We show that the lex-parse of T_{2k} has $2k - 2$ phrases. More specifically, we show that the lengths of the phrases are

$$f_{2k-1} - 1, f_{2k-4} - 1, f_{2k-5} + 1, \dots, f_4 - 1, f_3 + 1, 1, 2, 1.$$

There are three types of phrases as follows: (1) first phrase, (2) inductive phrases, (3) last three short phrases. Phrases of Type (1) or Type (3) can be obtained by simple properties of Fibonacci words. However, phrases of Type (2) need a more complicated discussion. We use the Lyndon factorizations of the strings to characterize the inductive phrases. Intuitively, we show that every suffix of T_{2k} that has an odd inductive phrase as a prefix can be written as the concatenation of the odd inductive phrase and a significant suffix.

(1) First phrase and (3) Short phrases

We start from Type (1). From the third property of Lemma 2 and the edit operation, $T_{2k}[1..]$ and $T_{2k}[f_{2k-2} + 1..]$ have a (longest) common prefix $x = F'_{2k-1}$ of length $f_{2k-1} - 1$ and $T_{2k}[1..] \succ T_{2k}[f_{2k-2} + 1..]$ holds. $T_{2k}[f_{2k-2} + 1..]$ can be written as $T_{2k}[f_{2k-2} + 1..] = x \cdot a$. We show that the previous suffix of $T_{2k}[1..]$ is $T_{2k}[f_{2k-2} + 1..]$. Suppose to the contrary that there exists a suffix y of F_{2k} that satisfies $T_{2k}[1..] \succ y \succ T_{2k}[f_{2k-2} + 1..]$. Since both $T_{2k}[f_{2k-2} + 1..]$ and $T_{2k}[1..]$ have x as a prefix, y can be written as $y = x \cdot a \cdot z_1$ or $y = x \cdot b \cdot z_2$ for some strings z_1, z_2 . Since F_{2k-1} ends with aab and thus aa is a suffix of x , the existence of $y = x \cdot a \cdot z_1$ contradicts the fact that a^3 only occurs at the edit position. On the other hand, the existence of $y = x \cdot b \cdot z_2$ contradicts the fact that $x \cdot b = F_{2k-1}$ only occurs as a prefix of T_{2k} , since otherwise it would violate the second property of Lemma 2. Thus $T_{2k}[f_{2k-2} + 1..]$ is the previous suffix of $T_{2k}[1..]$, and the length of the first phrase is $|x| = f_{2k-1} - 1$.

Next, we consider Type (3) phrases. Since T_{2k} ends with $baaa$ and no Fibonacci word has aaa as a substring, we conclude that $SA_{T_{2k}}[1] = f_{2k}$, $SA_{T_{2k}}[2] = f_{2k} - 1$, and $SA_{T_{2k}}[3] = f_{2k} - 2$. In particular, ba^3 is the smallest suffix of T_{2k} that begins with b . These facts imply that $T_{2k}[f_{2k}] = a$ and $T_{2k}[f_{2k} - 3] = b$ are the explicit phrases, and $T_{2k}[f_{2k} - 2..f_{2k} - 1] = a^2$ between the explicit phrases is a phrase. Thus the last three phrases are b, a^2, a .

(2) Inductive phrases

In the rest of this section, we explain Type (2) phrases. Firstly, we study the Lyndon factorization $LF(\mathcal{F}) = \ell_1, \ell_2, \dots$ of the (infinite) Fibonacci word. To characterize these Lyndon factors ℓ_i , we use the string morphism ϕ with $\phi(a) = aab, \phi(b) = ab$ as defined in [24, Proposition 3.2].

► **Lemma 7** ([24]). $\ell_1 = ab, \ell_{k+1} = \phi(\ell_k)$, and $|\ell_k| = f_{2k+1}$ holds.

In order to show our result, we consider the Lyndon factorization of $F''_{2k} (= T''_{2k})$ (Lemma 10). Lemmas 8 and 9 explain the Lyndon factorization of a finite prefix of the (infinite) Fibonacci word by using properties of the morphism ϕ .

► **Lemma 8.** Given a string $w \in \{a, b\}^+$, let $LF(w) = \lambda_1^{p_1}, \dots, \lambda_m^{p_m}$. Then $LF(\phi(w)) = \phi(\lambda_1^{p_1}), \dots, \phi(\lambda_m^{p_m})$.

Proof. For any two binary strings x and y , it is clear that $\phi(x) \succ \phi(y)$ if $x \succ y$. In the rest of this proof, we show that $\phi(x)$ is a Lyndon word for a Lyndon word x over $\{a, b\}$. If $|x| = 1$, then the statement clearly holds. Suppose that $|x| \geq 2$. Let \tilde{x} be a non-empty proper suffix of $\phi(x)$. Then \tilde{x} can be represented as $\tilde{x} = \phi(y)$ for some suffix y of x , or $\tilde{x} = \alpha \cdot \phi(y)$ for some suffix y of x and $\alpha \in \{ab, b\}$. Since $x \prec y$, $\phi(x) \prec \tilde{x}$ if $\tilde{x} = \phi(y)$ for some suffix y of x . Also in the case of $\tilde{x} = \alpha \cdot \phi(y)$, $\phi(x) \prec \tilde{x}$ holds since $x[1] = a$ (from x is a Lyndon word) and $\phi(x)[1..3] = aab$. Thus, $\phi(x) \prec \tilde{x}$ holds for all non-empty proper suffixes \tilde{x} of x . This implies that $\phi(x)$ is a Lyndon word. Therefore, the statement holds. ◀

► **Lemma 9.** For every integer $i \geq 1$, $LF(\ell'_i) = \phi^{i-1}(a), \dots, \phi^0(a)$, where $\ell'_i = \ell_i[..\ell_i| - 1]$.

Proof. We prove the statement by induction on i . For the base case $i = 1$, $LF(\ell'_1) = a = \phi^0(a)$. Assume that the lemma holds for all $i \leq j$ for some $j \geq 1$. By the definition of the morphism ϕ , $\ell'_{j+1} = \phi(\ell_j)' = \phi(\ell'_j) \cdot a$ holds because ℓ_j ends with b . Also, by using Lemma 8 and the induction hypothesis,

$$LF(\ell'_{j+1}) = LF(\phi(\ell'_j) \cdot a) = LF(\phi(\ell'_j)), a = \phi(LF(\ell'_j)), a = \phi^j(a), \dots, \phi^1(a), \phi^0(a).$$

Therefore, the lemma holds. ◀

► **Lemma 10.** Let L_i be the i -th Lyndon factor of F''_{2k} . For every $k \geq 2$,

$$L_i = \begin{cases} \phi^{i-1}(ab) & \text{if } 1 \leq i < k-1, \\ \phi^{2k-i-3}(a) & \text{if } k-1 \leq i \leq 2k-3. \end{cases}$$

Proof. From Lemma 7,

$$\sum_{i=1}^{k-1} |\ell_i| = \sum_{i=1}^{k-1} f_{2i+1} = f_3 + f_5 + \dots + f_{2k-1} = f_2 + (f_3 + f_5 + \dots + f_{2k-1}) - 1 = f_{2k} - 1.$$

Thus $LF(F_{2k}) = \ell_1, \dots, \ell_{k-1}, a$ holds. Also, $L_i = \ell_i$ holds for all $i < k-1$ since ℓ_i is not a prefix of ℓ_{i-1} . In other words, $LF(F''_{2k}) = \ell_1, \dots, \ell_{k-2}, LF(\ell'_{k-1})$. From Lemma 9,

$$LF(F''_{2k}) = \ell_1, \dots, \ell_{k-2}, \phi^{k-2}(a), \dots, \phi^0(a).$$

Then the statement also holds. ◀

Moreover, we can find the specific form of suffixes characterized by the Lyndon factorization of F''_{2k} as described in the next lemma.

► **Lemma 11.** For every integer $i \geq 1$, $\phi^{i-1}(a) \dots \phi^0(a)$ is a prefix of $\phi^i(a)$.

Proof. We prove the lemma by induction on i . For the base case $i = 1$, $\phi^0(a) = a$ is a prefix of $\phi^1(a) = aab$. Assume that the statement holds for all $i \leq j$ for some $j \geq 1$. For $i = j+1$,

$$\phi^{j+1}(a) = \phi^j(\phi(a)) = \phi^j(aab) = \phi^j(a)\phi^j(a)\phi^j(b).$$

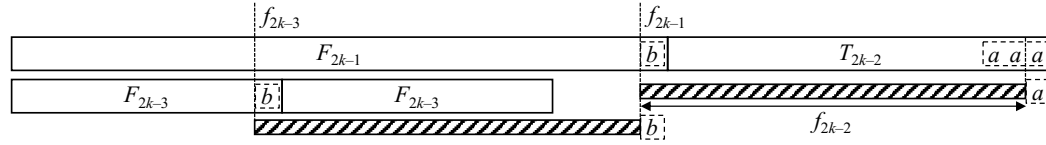
By induction hypothesis, $\phi^j(a) = \phi^{j-1}(a) \dots \phi^0(a) \cdot x$ for some string x . Then $\phi^{j+1}(a) = \phi^j(a) \cdot \phi^{j-1}(a) \dots \phi^0(a) \cdot x \cdot \phi^j(b)$. This implies that the statement also holds for $i = j+1$. Therefore, the lemma holds. ◀

With the Lemmas 10 and 11, we obtain the following lemma, which characterizes the first $k+1$ entries of the suffix array $SA_{T_{2k}}$ of T_{2k} .

Firstly, we consider the order of the significant suffixes of F''_{2k} .

► **Lemma 12.** $SA_{F''_{2k}}[i] = f_{2k} - 1 - \sum_{j=0}^{i-1} |\phi^j(a)|$ for every $i \in [1..k-1]$.

Proof. We can see that $\phi^{k-2}(a) \dots \phi^0(a)$ is a suffix of F''_{2k} by Lemma 10. Our claim is that $\phi^{i-1}(a) \dots \phi^0(a)$ is the i -th lexicographically smallest suffix of F''_{2k} for every $0 \leq i \leq k-1$. We prove the statement by induction on i . For the base case $i = 1$, $\phi^0(a) = a$ is the lexicographically smallest suffix of F''_{2k} . Assume that the statement holds for all $i \leq i'$ for some $i' \geq 1$. Let α be a suffix of $\phi^{i'+1}(a) \dots \phi^0(a)$ such that there is no d with $\alpha = \phi^d(a) \dots \phi^0(a)$. (Otherwise we already know that α is a prefix of $\phi^{i'+1}(a)$, which we already processed for a



■ **Figure 2** Illustration of T_{2k} for proof of Lemma 16.

► **Lemma 16.** *The lexicographically largest suffix maxsuf of T_{2k} is $T_{2k}[f_{2k-3}..]$.*

Proof. It is known that the lexicographically largest suffix of F_{2k} is $F_{2k}[f_{2k-1}..]$ and the lexicographically second largest suffix of F_{2k} is $F_{2k}[f_{2k-3}..]$ (shown in [18]). Namely, $SA_{F_{2k}}[f_{2k}] = f_{2k-1}$, $SA_{F_{2k}}[f_{2k} - 1] = f_{2k-3}$. Due to the edit operation, $T_{2k}[f_{2k-3}..] \succ T_{2k}[f_{2k-1}..]$ and the length of the longest prefix of these suffixes is f_{2k-2} (see Fig. 2). Assume on the contrary that there is a suffix $T_{2k}[i..]$ that is lexicographically larger than $T_{2k}[f_{2k-3}..]$. With Property 5 of Lemma 2, the suffix aaa of T_{2k} acts as a unique delimiter such that the suffix $T_{2k}[f_{2k-3}..]$ cannot be a prefix of any other suffixes of T_{2k} . Let j be the smallest positive integer such that $T_{2k}[f_{2k-3}..f_{2k-3} + j] = a$ and $T_{2k}[i..i + j] = b$. If $\max(i + j, f_{2k-3} + j) < f_{2k} - 1$, then $F_{2k}[f_{2k-3}..] \prec F_{2k}[i..]$ holds. This contradicts the fact that the lexicographically second largest suffix of F_{2k} is $F_{2k}[f_{2k-3}..]$. We can observe that there is no j with $\max(i + j, f_{2k-3} + j) \geq f_{2k} - 1$ and $i > f_{2k-3}$ since $T_{2k}[f_{2k} - 1..] = aa$ does not contain b . If $\max(i + j, f_{2k-3} + j) \geq f_{2k} - 1$ and $i < f_{2k-3}$, F_{2k-3} has a beginning position d of an occurrence satisfying $2 \leq d \leq f_{2k-3}$. This contradicts Lemma 2. Therefore, the lemma holds. ◀

To prove Lemma 18, we also introduce the following corollary.

► **Corollary 17** (of Lemma 2). *For every i satisfying $i \geq 6$, T_i has exactly two occurrences of F_{i-2} . Moreover, T_i can be written as $T_i = F_{i-2} \cdot F_{i-2} \cdot w$ for some string w .*

► **Lemma 18.** *For every $k \geq 2$ and i satisfying $1 \leq i \leq k - 3$, the previous suffix of X_i w.r.t. T_{2k} is $Y_i \cdot a \cdot \text{maxsuf}$.*

Proof. Firstly, we show that $Y_i \cdot a \cdot \text{maxsuf}$ is a suffix of T_{2k} . It is clear from definitions and properties of Lemma 2 that T_{2k} can be written as $T_{2k} = F'_{2k-3} \cdot \text{maxsuf} = F''_{2k-3} \cdot a \cdot \text{maxsuf} = F_{2k-4} \cdot F''_{2k-5} \cdot a \cdot \text{maxsuf} = F_{2k-5} \cdot F''_{2k-4} \cdot a \cdot \text{maxsuf}$. Since $Y_{k-3} = b \cdot T''_{2k-4} = b \cdot F''_{2k-4}$ and the last character of F_{2k-5} is b (from $2k - 5$ is odd), $Y_{k-3} \cdot a \cdot \text{maxsuf}$ is a suffix of T_{2k} . Moreover, F_{2i+2} is a suffix of F_{2k-4} for every i that satisfying $1 \leq i \leq k - 3$. This implies that $Y_i \cdot a \cdot \text{maxsuf}$ is a suffix of T_{2k} for every i that satisfying $1 \leq i \leq k - 3$.

Now we go back to our main proof of the lemma. Since $X_i = Y_i \cdot b \cdot \phi^i(a) \cdots \phi^0(a) \cdot aa$, $Y_i \cdot a \cdot \text{maxsuf} \prec X_i$ holds. Moreover, $Y_i \cdot a \cdot \text{maxsuf}$ is the lexicographically largest suffix of T_{2k} that has $Y_i \cdot a$ as a prefix. Hence, it is sufficient to prove that X_i is the lexicographically smallest suffix of T_{2k} that has $Y_i \cdot b$ as a prefix. From Property 5 of Lemma 2, no bb occurs in T_{2k} , so every occurrence of $Y_i \cdot b$ is also an occurrence of $Y_i \cdot ba$. We consider occurrences of $Y_i \cdot ba$ in a suffix X_i . From Observation 14 and Corollary 17, $(Y_i \cdot ba)[2..](= F_{2i+2})$ has exactly two occurrences in $X_i(= b \cdot T_{2i+4})$. At the first occurrence, $(Y_i \cdot ba)[2..]$ is preceded by b . At the second occurrence, $(Y_i \cdot ba)[2..]$ is preceded by a . Hence, the rightmost occurrence of $Y_i \cdot ba$ in T_{2k} is at the prefix of X_i . By combining with Lemma 13, we can see that there is no suffix w such that $Y_i \cdot ba \cdot w \prec X_i$ holds. Thus, X_i is the lexicographically smallest suffix of T_{2k} that has $Y_i \cdot b$ as a prefix. Therefore, the lemma holds. ◀

Lex-parse of T_{2k}

Now we can explain the lex-parse of T_{2k} . Recall that the length of the first phrase is $f_{2k-1} - 1$ and the last three phrases are b, a^2, a . Hence, from Lemma 18, the second phrase is a prefix Y_{k-3} of $X_{k-3} (= b \cdot T_{2k-2})$. Since the remaining suffix is Z_{k-3} , the next phrase is a prefix $Z_{k-4}[\cdot | Z_{k-4}| - 1]$ of Z_{k-3} from Corollary 15. By applying this argument repeatedly, we can finally obtain the lex-parse of size $2k - 2$ of T_{2k} as follows:

$$F_{2k}[\cdot f_{2k-1} - 1], (Y_{k-3}, Z_{k-4}[\cdot | Z_{k-4}| - 1]), \dots, (Y_1, Z_0[\cdot | Z_0| - 1]), b, a^2, a.$$

Furthermore, we can easily obtain $v(F_{2k}'' \cdot a) = 2k - 2$ for a delete operation. Consider the case for the insertion operation such that $\$$ (which is the smallest character) is inserted to the preceding position of the last b . We can then show that the lex-parse is of size $2k + 1$ by a similar argument as follows:

$$F_{2k}[1 \cdot f_{2k-1} - 2], (a \cdot Y_{k-3}, Z_{k-4}[\cdot | Z_{k-4}| - 2]), \dots, (a \cdot Y_1, Z_0[\cdot | Z_0| - 2]), a, ba, \$, b, a.$$

5 Lower bounds for Alphabet-Ordering

In this section, we give tight lower bound $\text{AOS}(v, n) \in \Omega(\log n)$ with the family of Fibonacci words. Since $b(F_k) \leq 4$ for $k \geq 5$ [27, Lemma 35] also holds, our lower bound is tight for any $n \in \{f_i\}_i$. More precisely, we prove the following theorem that determines the number of lex-parse phrases of the Fibonacci words on any alphabet ordering.

► **Theorem 19.** *For any $k \geq 6$,*

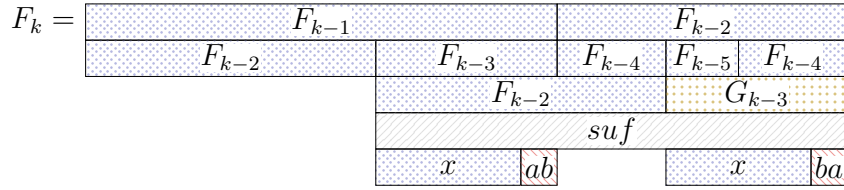
$$v(F_k, \prec) = \begin{cases} \lceil \frac{k}{2} \rceil + 1 & (a) \text{ if } k \text{ is odd and } a \prec b, \quad (\text{Lemma 23}) \\ 4 & (b) \text{ if } k \text{ is odd and } b \prec a, \quad (\text{Lemma 24}) \\ 4 & (c) \text{ if } k \text{ is even and } a \prec b, \quad (\text{Lemma 20}) \\ \lceil \frac{k}{2} \rceil + 1 & (d) \text{ if } k \text{ is even and } b \prec a. \quad (\text{Lemma 25}) \end{cases}$$

Although the results for a smaller than b have been proven by Navarro et al. [27], we here give alternative proofs for this case that leads us to the proof for the case when b is smaller than a .

5.1 Lex-parse with constant number of phrases

We start with Cases (b) and (c). Since $F_k[f_k - 1..f_k] = ba$ for even k and $F_k[f_k - 1..f_k] = ab$ for odd k , we already know in the cases (b) and (c) that each of the last two characters forms an explicit phrase. It is left to analyze the non-explicit phrases, where we start with the first phrase. From Property 1 of Lemma 2, F_k has the border F_{k-2} and thus $F_{k-2} = F_k[f_{k-1} \cdot] \prec F_k$ could be used as the reference for the first phrase, given its length is f_{k-2} . To be an eligible reference for lex-parse, we need to check that $F_k[f_{k-1} \cdot]$ is the previous suffix of $F_k[1 \cdot]$. However, Property 2 of Lemma 2 states that there is exactly one other occurrence of F_{k-2} in F_k , starting at $f_{k-2} + 1$. The proof of the following lemma shows that the suffix starting with that occurrence is lexicographically larger than F_k , and thus indeed the first phrase has length f_{k-2} , and the second phrase starting with that occurrence can make use of F_k as a reference for a phrase that just ends before the two explicit phrases at the end.

► **Lemma 20.** *Assume that $k \geq 6$ is even and $a \prec b$ (Case (c) of Thm. 19). Then the lex-parse of F_k is $F_{k-2}, F_k[f_{k-2} + 1..f_k - 2], b, a$.*



■ **Figure 3** Illustration of the proof of Lemma 20 for k even. If k is odd, the blocks ab and ba are swapped (this gives the setting Lemma 24).

Proof. F_k can be represented as

$$F_k = F_{k-1} \cdot F_{k-2} = F_{k-2} \cdot F_{k-3} \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}.$$

Let $suf = F_{k-2} \cdot G_{k-3}$ (a suffix of F_k) and x be the longest common prefix of F_{k-3} and G_{k-3} . Then $F_k = F_{k-2} \cdot x \cdot ab \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}$ and $suf = F_{k-2} \cdot x \cdot ba$ holds since k is even and $k-3$ is odd. See Fig. 3 for a sketch. This implies that the three suffixes that have F_{k-2} as a prefix satisfy $F_{k-2} \prec F_k \prec suf$. By combining with Property 2 of Lemma 2, the first phrase is F_{k-2} and the second phrase is $F_{k-2} \cdot x$ (which is the longest common prefix of F_k and suf). The third phrase is b which is an explicit phrase of character b since the suffix ba is the lexicographically smallest suffix that has b as a prefix. Finally, the last phrase is an explicit phrase a . ◀

5.2 Lex-parse with logarithmic number of phrases

Next, we discuss the cases that have a logarithmic number of phrases. For any odd $k \geq 7$ and even i satisfying $4 \leq i \leq k-3$, let

$$\begin{aligned} suf_i &= (F_i \cdot F_{i-2} \cdot F_{i-3} \cdots F_4) \cdot ab = F_{i+1}, \\ suf_i^+ &= F_i \cdot suf_i = G_{i+2}. \end{aligned}$$

From the definitions, the following properties hold.

► **Lemma 21.** For odd $k \geq 7$, suf_i and suf_i^+ are suffixes of F_k , for every even i with $4 \leq i \leq k-3$. In particular, we have

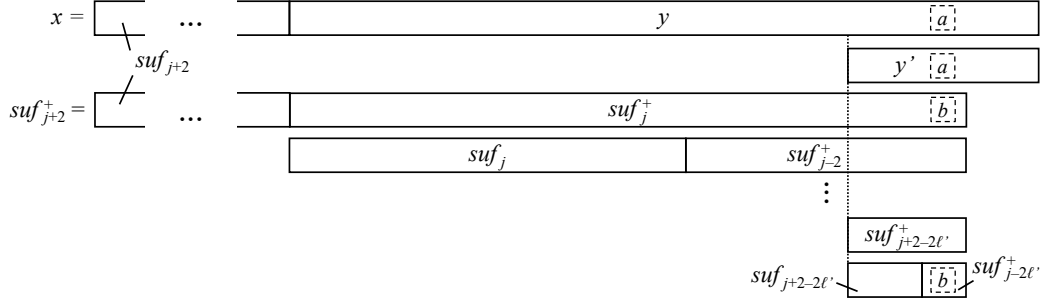
- (a) $suf_i = F_{i+1}$ is a prefix of $suf_i^+ = G_{i+2} = F_i F_{i+1}$, and
- (b) $suf_i^+ = F_i F_{i+1} = F_i F_{i-1} F_{i-2} F_{i-1} = F_{i+1} G_i = suf_i \cdot suf_{i-2}^+$.

Proof. By definitions, $F_k = F_{k-2} F_{k-3} F_{k-2} = F_{k-2} F_{k-3} suf_{k-3} = F_{k-2} suf_{k-2}^+$. Since suf_{i-2} is a suffix of suf_i , the claim holds for suf_i for every i . Writing $suf_i = F_{i-1} \cdot F_{i-2} \cdot (F_{i-2} \cdot F_{i-3} \cdots F_4) \cdot ab$, we can see that suf_{i-2}^+ is a suffix of suf_i . ◀

We use these suffixes to characterize the lex-parse. The following lemma shows that suf_i is the previous suffix of suf_i^+ w.r.t. F_k for every i , where $f_k - |suf_i^+| + 1$ and $f_k - |suf_i| + 1 - 1$ are the starting positions of suf_i^+ and suf_i in F_k , respectively.

► **Lemma 22.** Assume that $k \geq 7$ is odd and $a \prec b$. The previous suffix of suf_i^+ w.r.t. F_k is suf_i for every even i satisfying $4 \leq i \leq k-3$.

Proof. Since $suf_i^+ = F_i \cdot F_{i-2} \cdots F_4 \cdot ab \cdot \alpha$ for some string α , suf_i is a prefix of suf_i^+ . Thus, $suf_i \prec suf_i^+$. We prove that there is no suffix x of F_k with $suf_i \prec x \prec suf_i^+$ by induction on i .



■ **Figure 4** Illustration of the proof of Lemma 22.

Let $i = 4$ for the base case. Assume on the contrary that there exists a suffix x of F_k such that $suf_4 \prec x \prec suf_4^+$. Since $suf_4 = F_5 = F_4 \cdot ab = aba \cdot ab$ and $suf_4^+ = F_4 \cdot F_5 = F_4 \cdot abaab$, x can be written as $x = F_4 \cdot abaaa \cdot y$ for some string y . However, aaa is not a substring of F_k according to Property 5 of Lemma 2, so x cannot exist. Thus, the statement holds for the base case.

Assume that the statement holds for all even $i \leq j$ for some even $j \geq 4$. Suppose on the contrary that there exists a suffix x of F_k such that $suf_{j+2} \prec x \prec suf_{j+2}^+$. Since suf_{j+2} is a prefix of suf_{j+2}^+ , we can represent x as $x = suf_{j+2} \cdot y$ for some string y . There are two cases w.r.t. the length of x . (1) Assume that $|suf_{j+2}| < |x| < |suf_{j+2}^+|$. Because of the assumption and the fact that F_{j+2}^2 is a prefix of suf_{j+2}^+ from Lemma 21(a), F_{j+2} occurs as a substring that is neither prefix nor a suffix of F_{j+2}^2 . However, F_{j+2}^2 cannot have such occurrence of F_{j+2} since F_{j+2} is primitive (from Property 6 of Lemma 2), a contradiction (from Lemma 1). (2) Assume that $|suf_{j+2}^+| < |x|$. We now use that $suf_i^+ = suf_i \cdot suf_{i-2}^+$ holds from Lemma 21(b). By the assumption, y and suf_j^+ mismatch with a and b , respectively (since x and suf_{j+2}^+ have suf_{j+2} as a prefix). From Lemma 21, suf_{j+2}^+ can be represented as

$$suf_{j+2}^+ = suf_{j+2} \cdot suf_j \cdots suf_{j+2-2\ell} \cdot suf_{j-2\ell}^+ \quad (1)$$

for some integer $\ell \geq 0$. Let ℓ' be the largest integer ℓ such that the mismatch position is in the factor $suf_{j-2\ell}^+$ of the suf_{j+2}^+ -factorization in Eq. 1, and y' be the suffix of y that has the factor $suf_{j+2-2\ell'}$ of the factorization in Eq. 1 as a prefix (i.e., $y' = y[|suf_j \cdots suf_{j+2-2(\ell'-1)}| + 1..]$). Since $suf_{j+2-2\ell'}$ is a prefix of y' , and y' and $suf_{j-2\ell'}^+$ mismatch at the same position, then $suf_{j+2-2\ell'} \prec y' \prec suf_{j+2-2\ell'}^+$ holds. This fact contradicts the induction hypothesis (see also Fig. 4). ◀

Now we can show the following main lemma from the above lemmas.

► **Lemma 23.** *Assume that $k \geq 7$ is odd and $a \prec b$. Then the lex-parse of F_k is*

$$F_k[1..f_{k-1} - 2], baF_{k-4}, F_{k-4}, F_{k-6}, \dots, F_5, a, a, b.$$

Proof. Let x be the longest common prefix of F_{k-3} and G_{k-3} . Due to Property 2 of Lemma 2, there are three suffixes $F_k = F_{k-2} \cdot x \cdot ba \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}$, $suf = F_{k-2} \cdot x \cdot ab$, and F_{k-2} that have F_{k-2} as a prefix. This implies that $F_{k-2} \prec suf \prec F_k$. Thus, the first phrase is $F_{k-2} \cdot x$. Then the remaining suffix is $ba \cdot suf_{k-3} = ba \cdot F_{k-2}$. We show the second phrase is $ba \cdot F_{k-4}$ by proving that the previous suffix of $ba \cdot F_{k-2}$ w.r.t. F_k is $ba \cdot F_{k-4}$. It is clear that $ba \cdot F_{k-4} \prec ba \cdot F_{k-2}$ since $ba \cdot F_{k-4}$ is a prefix of $ba \cdot F_{k-2}$. Suppose on the contrary that there exists a suffix y of F_k that satisfies $ba \cdot F_{k-4} \prec y \prec ba \cdot F_{k-2}$. From the assumption, $ba \cdot F_{k-4}$ is a prefix of y . We can observe that there are three occurrences of $ba \cdot F_{k-4}$ in

F_k from Lemma 3 (i.e., the third, the fourth, and the sixth occurrence of F_{k-4} in F_k). This implies that suffix $ba \cdot F_{k-4} \cdot G_{k-1}$ (regarding the third occurrence of F_{k-4}) of F_k is the only candidate of y . However,

$$y = ba \cdot F_{k-4} \cdot G_{k-1} = ba \cdot F_{k-4} \cdot F_{k-3} \cdot F_{k-2} \succ ba \cdot F_{k-4} \cdot G_{k-3} = ba \cdot F_{k-2}$$

holds. Thus, the previous suffix of $ba \cdot F_{k-2}$ w.r.t. F_k is $ba \cdot F_{k-4}$, and the second phrase is $ba \cdot F_{k-4}$. Then, the remaining suffix is suf_{k-5}^+ . From Lemma 22, the next phrase is $\text{suf}_{k-5} = F_{k-4}$ and the remaining suffix is suf_{k-7}^+ . This continues until the remaining suffix is aab . It is easy to see that the last three phrases are a, a, b . ◀

We can also prove the following lemmas similarly.

► **Lemma 24.** *Assume that $k \geq 7$ is odd and $b \prec a$. Then the lex-parse of F_k is*

$$F_{k-2}, F_k[f_{k-2} + 1..f_k - 2], a, b.$$

► **Lemma 25.** *Assume that $k \geq 6$ is even and $b \prec a$. Then the lex-parse of F_k is*

$$F_k[1..f_{k-1} - 2], abF_{k-4}, F_{k-4}, F_{k-6}, \dots, F_6, b, a.$$

Overall, Theorem 19 holds.

6 Conclusion

In this paper, we considered the compression sensitivity of lex-parse for two operations: single character edit and modification of the alphabet ordering, and gave $\Theta(\log n)$ bounds for both operations. A further work in this line of research on the alphabet orderings is the problem of computing optimal alphabet orderings for the lex-parse. The problems for the RLBWT and the Lyndon factorization are known to be NP-hard [2, 13].

References

- 1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *Information and Computation*, 291:104999, 2023. doi:10.1016/j.ic.2022.104999.
- 2 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of BWT-runs minimization via alphabet reordering. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.15.
- 3 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.
- 4 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL SRC RESEARCH REPORT, 1994.
- 5 Davide Cenzato, Veronica Guerrini, Zsuzsanna Lipták, and Giovanna Rosone. Computing the optimal BWT of very large string collections. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2023, Snowbird, UT, USA, March 21-24, 2023*, pages 71–80. IEEE, 2023. doi:10.1109/DCC55655.2023.00015.
- 6 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. IV. The quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.

- 7 Amanda Clare and Jacqueline W. Daykin. Enhanced string factoring from alphabet orderings. *Inf. Process. Lett.*, 143:4–7, 2019. doi:10.1016/J.IPL.2018.10.011.
- 8 Amanda Clare, Jacqueline W. Daykin, Thomas Mills, and Christine Zarges. Evolutionary search techniques for the Lyndon factorization of biosequences. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 1543–1550. ACM, 2019. doi:10.1145/3319619.3326872.
- 9 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.
- 10 Maxime Crochemore and Renaud VÉrin. Direct construction of compact directed acyclic word graphs. In *Proc. CPM*, volume 1264 of *LNCS*, pages 116–129, 1997. doi:10.1007/3-540-63220-4_55.
- 11 Hiroto Fujimaru, Yuto Nakashima, and Shunsuke Inenaga. On sensitivity of compact directed acyclic word graphs. In Anna E. Frid and Robert Mercas, editors, *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13899 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2023. doi:10.1007/978-3-031-33180-0_13.
- 12 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. A new class of string transformations for compressed text indexing. *Inf. Comput.*, 294:105068, 2023. doi:10.1016/J.IC.2023.105068.
- 13 Daniel Gibney and Sharma V. Thankachan. Finding an optimal alphabet ordering for Lyndon factorization is hard. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPICs*, pages 35:1–35:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.STACS.2021.35.
- 14 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Bit catastrophes for the Burrows–Wheeler transform. In Frank Drewes and Mikhail Volkov, editors, *Developments in Language Theory - 27th International Conference, DLT 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13911 of *Lecture Notes in Computer Science*, pages 86–99. Springer, 2023. doi:10.1007/978-3-031-33264-7_8.
- 15 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.
- 16 Hiroe Inoue, Yoshiaki Matsuoka, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Factorizing strings into repetitions. *Theory Comput. Syst.*, 66(2):484–501, 2022. doi:10.1007/S00224-022-10070-3.
- 17 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC 2018*, pages 827–840, 2018.
- 18 Dominik Köppl and Tomohiro I. Arithmetics on suffix arrays of Fibonacci words. In Florin Manea and Dirk Nowotka, editors, *Combinatorics on Words - 10th International Conference, WORDS 2015, Kiel, Germany, September 14-17, 2015, Proceedings*, volume 9304 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2015. doi:10.1007/978-3-319-23660-5_12.
- 19 Guillaume Lagarde and Sylvain Perifel. Lempel-Ziv: A “one-bit catastrophe” but not a tragedy. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1478–1495. SIAM, 2018. doi:10.1137/1.9781611975031.97.
- 20 M Lothaire. *Applied combinatorics on words*, volume 105. Cambridge University Press, 2005.
- 21 R. C. Lyndon. On Burnside’s problem. *Transactions of the American Mathematical Society*, 77:202–215, 1954.

- 22 Lily Major, Amanda Clare, Jacqueline W. Daykin, Benjamin Mora, Leonel Jose Peña Gamboa, and Christine Zarges. Evaluation of a permutation-based evolutionary framework for Lyndon factorizations. In Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature – PPSN XVI – 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5–9, 2020, Proceedings, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 390–403. Springer, 2020. doi:10.1007/978-3-030-58112-1_27.
- 23 Lily Major, Amanda Clare, Jacqueline W. Daykin, Benjamin Mora, and Christine Zarges. Heuristics for the run-length encoded Burrows–Wheeler transform alphabet ordering problem. *CoRR*, abs/2401.16435, 2024. doi:10.48550/arXiv.2401.16435.
- 24 Guy Melançon. Lyndon factorization of sturmian words. *Discrete Mathematics*, 210(1):137–149, 2000. doi:10.1016/S0012-365X(99)00123-5.
- 25 Gonzalo Navarro. Indexing highly repetitive string collections. *CoRR*, abs/2004.02781, 2020. arXiv:2004.02781.
- 26 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.
- 27 Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inf. Theory*, 67(2):1008–1026, 2021. doi:10.1109/TIT.2020.3042746.
- 28 James A. Storer and Thomas G. Szymanski. The macro model for data compression (extended abstract). In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1–3, 1978, San Diego, California, USA*, pages 30–39. ACM, 1978. doi:10.1145/800133.804329.
- 29 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.
- 30 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.