On the Approximation Ratio of Ordered Parsings

Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza

Abstract

Shannon's entropy is a clear lower bound for statistical compression. The situation is not so well understood for dictionary-based compression. A plausible lower bound is b, the least number of phrases of a general bidirectional parse of a text, where phrases can be copied from anywhere else in the text. Since computing b is NP-complete, a popular gold standard is z, the number of phrases in the Lempel-Ziv parse of the text, which is computed in linear time and yields the least number of phrases when those can be copied only from the left. Almost nothing has been known for decades about the approximation ratio of z with respect to b. In this paper we prove that $z = O(b \log(n/b))$, where n is the text length. We also show that the bound is tight as a function of n, by exhibiting a text family where $z = \Omega(b \log n)$. Our upper bound is obtained by building a run-length context-free grammar based on a locally consistent parsing of the text. Our lower bound is obtained by relating b with r, the number of equal-letter runs in the Burrows-Wheeler transform of the text. We continue by observing that Lempel-Ziv is just one particular case of *greedy* parses–meaning that it obtains the smallest parse by scanning the text and maximizing the phrase length at each step–, and of *ordered* parses–meaning that phrases are larger than their sources under some order. As a new example of ordered greedy parses, we introduce *lexicographical* parses, where phrases can only be copied from lexicographically smaller text locations. We prove that the size v of the optimal lexicographical parse is also obtained greedily in $O(n)$ time, that $v = O(b \log(n/b))$, and that there exists a text family where $v = \Omega(b \log n)$. Interestingly, we also show that $v = O(r)$ because r also induces a lexicographical parse, whereas $z = \Omega(r \log n)$ holds on some text families. We obtain some results on parsing complexity and size that hold on some general classes of greedy ordered parses. In our way, we also prove other relevant bounds between compressibility measures, especially with those related to smallest grammars of various types generating (only) the text.

Index Terms

Lempel-Ziv complexity; Repetitive sequences; Optimal bidirectional parsing; Greedy parsing; Ordered parsing; Lexicographic parsing; Run-length compressed Burrows-Wheeler Transform; Context-free grammars; Collage systems

I. Introduction

Shannon [55] defined a measure of entropy that serves as a lower bound to the attainable compression ratio on any source that emits symbols according to a certain probabilistic model. An attempt to measure the compressibility of finite texts $T[1..n]$, other than the non-computable Kolmogorov complexity [38], is the notion of empirical entropy [10], where some probabilistic model is assumed and its parameters are estimated from the frequencies observed in the text. Other measures that, if the text is generated from a probabilistic source, converge to its Shannon entropy, are derived from the Lempel-Ziv parsing [41] or the grammar-compression [35] of the text.

Some text families, however, are not well modeled as coming from a probabilistic source. A case that has been gaining attention is that of *highly repetitive texts*, where most of the text can be obtained by copying long blocks from elsewhere in the same text. Huge highly repetitive text collections are arising from the sequencing of myriads of genomes of the same species (e.g., the 100K Genome Project[1]), from versioned document repositories like Wikipedia, from source code repositories like GitHub, etc. Their growth is outpacing Moore's Law by a wide margin [56]. Understanding the compressibility of highly repetitive texts is important to properly compress those collections.

Lempel-Ziv and grammar compression are particular cases of so-called *dictionary techniques*, where a set of strings is defined and the text is parsed as a concatenation of those strings. On repetitive collections, the empirical entropy ceases to be a relevant compressibility measure; for example the kth order per-symbol entropy of TT is the same as that of T, if $k \ll n$ [40, Lem. 2.6], yet this entropy measure is generally meaningless for $k > \log n$ [17]. Some dictionary measures, instead, capture much better the compressibility of repetitive texts. For example, while an individual genome can rarely be compressed to much less than 2 bits per symbol, Lempel-Ziv has been reported to compress collections of human genomes to less than 1% [16]. Similar compression ratios are reported in Wikipedia.[2]

Despite the obvious practical relevance of these compression methods, there is not a clear entropy measure useful for highly repetitive texts. The number z of phrases generated by the Lempel-Ziv parse [41] is often used as a gold standard, possibly because it can be implemented in linear time [51] and is never larger than g, the size of the smallest context-free grammar that generates the text [52], [8]. However, z is not so satisfactory as an entropy measure: the value may change if we reverse the text, for example. A much more robust lower bound on compressibility is b, the size of the smallest *bidirectional (macro)*

An early version of this paper appeared in *Proc. LATIN'18* [20].

Gonzalo Navarro and Carlos Ochoa are with Center for Biotechnology and Bioengineering (CeBiB), Department of Computer Science, University of Chile, Chile. Gonzalo Navarro is also with Millennium Institute for Foundational Research on Data (IMFD), Chile. Nicola Prezza is with Ca' Foscari University of Venice, Italy.

[1] `https://www.genomicsengland.co.uk/about-genomics-england/the-100000-genomes-project`

[2] In `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`, accessed on Nov 2020, it reads "As of June 2015, the dump of all pages with complete edit history in XML format at enwiki dump progress on 20150602 is about 100 GB compressed using 7-Zip, and 10 TB uncompressed".

TABLE I
NOTATION ASSUMED ALL ALONG THE PAPER, INCLUDING THEOREMS.

Symbol	Meaning
T	Text to be parsed or compressed
n	Text length
σ	Text alphabet size
f	Target-to-source mapping in a parsing of T
H_k	Per-symbol kth order empirical entropy of T
b	Size of smallest bidirectional scheme for T
z	Size of Lempel-Ziv parse for T
z_{no}	Size of Lempel-Ziv parse for T not allowing overlaps
g	Smallest size (number of rules) of an SLP generating T
g_{rl}	Smallest size (number of rules) of an RLSLP generating T
c	Smallest size (number of rules) of an internal collage system generating T
r	Number of runs in the BWT of T
u	Smallest size of an ordered parse for T
v	Size of the lex-parse for T
f_k	kth Fibonacci number
F_k	kth Fibonacci word

scheme [57]. Such a scheme parses the text into phrases such that each phrase appears somewhere else in the text (or it is a single explicit symbol), in a way that makes it possible to recover the text by copying source to target positions in an appropriate order. This is arguably the strongest possible dictionary method, but finding the smallest bidirectional scheme is NP-complete [21]. A relevant question is then how good is the Lempel-Ziv parse as an efficiently implementable approximation to the smallest bidirectional scheme. Almost nothing is known in this respect, except that there are string families where z is as large as nearly $2b$ [57].[3]

In this paper we finally give a tight approximation ratio for z, showing that the gap is larger than what was previously known. We prove that $z = O(b \log(n/b))$, and that this bound is tight as a function of n, by exhibiting a string family where $z = \Omega(b \log n)$. To prove the upper bound, we show how to build a run-length context-free grammar [47] (i.e., allowing constant-size rules of the form $X \to Y^t$) of size $g_{rl} = O(b \log(n/b))$. This is done by carrying out several rounds of locally consistent parsing [27] on top of T, reducing the resulting blocks to nonterminals in each round, and showing that new nonterminals appear only in the boundaries of the phrases of the bidirectional scheme. We then further prove that $z \leq 2g_{rl}$ for any run-length grammar of size g_{rl}, by extending a classical proof [8] that relates grammars with Lempel-Ziv compression. To prove the lower bound, we consider another plausible compressibility measure: the number r of equal-symbol runs in the Burrows-Wheeler transform (BWT) of the text [7]. We prove that the BWT induces a bidirectional scheme, and thus $r = \Omega(b)$. The bound follows from the family of Fibonacci words, where $z = \Theta(\log n)$ [14] and $r = O(1)$ [44]. The latter result, however, assumes that lexicographical comparisons regard the strings as cyclic, instead of the more natural notion we use here. We then study the Fibonacci words under our model, to show that $r = O(1)$ still holds for the even Fibonacci words.

We then show that Lempel-Ziv is just one valid example of interesting parses fulfilling that (i) they can be efficiently computed with a greedy algorithm, and (ii) they impose an increasing order between sources and targets. We define a weak and a strong notion of order, which coincide in the case of the text-precedence order used by Lempel-Ziv. We design a greedy polynomial-time algorithm that always finds the optimum parse that strongly satisfies a given order. We also prove that the optimum parse weakly satisfying a given order is of size $O(g)$, and even $O(g_{rl}) \subseteq O(b \log(n/b))$ if sources can overlap targets.

We then give a concrete parsing arising from our generalization. We define v, the size of the optimal *lexicographic parse* of the text, where each phrase must point to a lexicographically smaller one (both seen as text suffixes). In such an order, the weak and strong versions are also equivalent. Thus, it holds that $v = O(g_{rl}) \subseteq O(b \log(n/b))$. Further, we show that v can be computed in linear time, with a very practical algorithm. We also show that r induces a lexicographical parse, thus $v = O(r)$. Since, instead, z can be $\Omega(r \log n)$, our new greedy parse asymptotically improves the Lempel-Ziv parse on some string families. We also show that $b = O(1)$ and $v = \Theta(\log n)$ (and thus $v = \Omega(b \log n)$) on the odd Fibonacci words, but we have not found a family where $z = o(v)$. We show that v and z perform comparably on a benchmark of repetitive texts.

Finally, we consider the size c of the smallest *collage system* [34], which adds to run-length context-free grammars the power to truncate a prefix or a suffix of a nonterminal. Little was known about this measure, except that $c = O(\min\{g_{rl}, z \log z\})$. By extending the ideas of the article, we prove that $c = O(z)$ and that there exists string families where $c = \Omega(r \log n)$. For a subclass we call *internal collage systems*, where all the productions appear in T, we also prove that $b = O(c)$.

II. BASIC CONCEPTS

We review basic concepts about strings, compression measures, and others. Table I summarizes our notation along the article.

[3] An article implying $z = \Omega(b \log n)$ [24, corollary in 3rd page] has a mistake: their string D is also parsed in $\Theta(N)$ phrases by LZ76, not $\Theta(N \log N)$.

A. Strings and String Families

A *string* (or *word*) is a sequence $S[1\,..\,\ell] = S[1]S[2]\cdots S[\ell]$ of symbols, of length $|S| = \ell$. A *substring* (or *factor*) $S[i]\cdots S[j]$ of S is denoted $S[i\,..\,j]$. A *suffix* of S is a substring of the form $S[i\,..\,\ell] = S[i\,..]$, and a *prefix* is a substring of the form $S[1\,..\,i] = S[..\,i]$. The juxtaposition of strings and/or symbols represents their concatenation; the explicit infix operator "$\cdot$" can also be used.

We will consider parsing or compressing a string $T[1\,..\,n]$, called the *text*, over the alphabet $[1\,..\,\sigma]$. We assume that T is terminated by the special symbol $T[n] = \$$, which is lexicographically smaller than all the others. This makes any lexicographic comparison between suffixes well-defined: when a suffix is a prefix of another, the prefix is lexicographically smaller. We use $S < S'$ to indicate that S is smaller than S' in lexicographic order.

We use various infinite families of strings along the article, to prove lower and upper bounds. An important family we use are the *Fibonacci words*, defined as follows.

Definition 1. *The Fibonacci word family is defined as $F_1 = b$, $F_2 = a$, and for all $k > 2$, $F_k = F_{k-1} \cdot F_{k-2}$. The length of F_k is f_k, the kth Fibonacci number, defined as $f_1 = f_2 = 1$ and, for $k > 2$, $f_k = f_{k-1} + f_{k-2}$.*

To obtain results compatible with the usual convention that a prefix of a suffix is lexicographically smaller than it, we will use a variant of the family that has the terminator \$ (virtually) appended.

Another family we will use is the *de Bruijn sequence* of order k on an alphabet of size σ. It contains all the distinct substrings of length k over $[1\,..\,\sigma]$, and it is of the minimum possible length, $\sigma^k + \sigma - 1$.

B. Bidirectional Schemes (b)

A *bidirectional scheme* [57] partitions $T[1\,..\,n]$ into b *phrases* $B_1, \ldots, B_b$, such that each phrase $B_i = T[t_i\,..\,t_i + \ell_i - 1]$ is either (1) copied from another substring $T[s_i\,..\,s_i + \ell_i - 1]$ (called the phrase *source*) with $s_i \neq t_i$ and $\ell_i \geq 1$, which may overlap $T[t_i\,..\,t_i + \ell_i - 1]$, or (2) formed by $\ell_i = 1$ explicit symbol. The phrases of type (1) are also called *targets* of the copies. The bidirectional scheme is *valid* if there is an order in which the sources $s_i + j$ can be copied onto the targets $t_i + j$ so that all the positions of T can be inferred.

A bidirectional scheme implicitly defines a function $f : [1\,..\,n] \to [1\,..\,n] \cup \{0\}$ so that,

$$\begin{cases} f(t_i + j) = s_i + j, & \text{if } T[t_i\,..\,t_i + \ell_i - 1] \text{ is copied from } T[s_i\,..\,s_i + \ell_i - 1] \text{ and } 0 \leq j < \ell_i \text{ (case 1)}, \\ f(t_i) = 0, & \text{if } T[t_i] \text{ is an explicit symbol (case 2)}. \end{cases}$$

Being a valid scheme is equivalent to saying that f has no cycles, that is, there is no $k > 0$ and p such that $f^k(p) = p$. Equivalently, for each p there exists $k \geq 0$ such that $f^k(p) = 0$. We can then recover each non-explicit text position p from the explicit symbol $T[f^{k-1}(p)]$.

We use b to denote the smallest bidirectional scheme, which is NP-complete to compute [21].

Example: Consider the text $T = alabaralalabarda\$$. A bidirectional scheme of 10 phrases is $ala|\underline{b}|a|\underline{r}|\underline{a}|\underline{l}|alabar|\underline{d}|a|\underline{\$}$, where we have underlined the explicit symbols (so $b \leq 10$ for T). A possible function corresponding to this parse is

$$f[1\,..\,17] = \langle 7, 8, 9, 0, 3, 0, 0, 0, 1, 2, 3, 4, 5, 6, 0, 11, 0\rangle.$$

For example, the source of phrase $B_1 = T[1\,..\,3] = ala$ is $T[7\,..\,9]$, and the source of phrase $B_7 = T[9\,..\,14] = alabar$ is $T[1\,..\,6]$. To extract $T[11]$, we follow the chain $f(11) = 3$, $f(3) = 9$, $f(9) = 1$, $f(1) = 7$, and $f(7) = 0$ because it is an explicit symbol. We then learn that $T[11] = T[3] = T[9] = T[1] = T[7] = a$.

C. Lempel-Ziv Parsing (z, z_{no})

Lempel and Ziv [41] define a parsing of T into the fewest possible phrases, $T = Z_1 \cdots Z_z$, so that each phrase Z_i occurs as a substring (but not a suffix) of $Z_1 \cdots Z_i$, or is an explicit symbol. This means that the source $T[s_i\,..\,s_i + \ell_i - 1]$ of the target $Z_i = T[t_i\,..\,t_i + \ell_i - 1]$ must satisfy $s_i < t_i$, but sources and targets may overlap. A parsing where sources precede targets in T is called *left-to-right*. It turns out that the *greedy* left-to-right parsing, which creates the phrases from Z_1 to Z_z and at each step i maximizes ℓ_i (and inserts an explicit symbol if $\ell_i = 0$), indeed produces the optimal number z of phrases [41, Thm. 1]. Further, the parsing can be obtained in $O(n)$ time [51], [57]. We call this the *Lempel-Ziv parse* of T.

If we disallow that a phrase overlaps its source, that is, Z_i must be a substring of $Z_1 \cdots Z_{i-1}$ or a single symbol, then we call z_{no} the number of phrases obtained. In this case it is also true that the greedy left-to-right parsing produces the optimal number z_{no} of phrases [57, Thm. 10 with $p = 1$]. Since the Lempel-Ziv parsing allowing overlaps is optimal among all left-to-right parses, we also have that $z_{no} \geq z$. This parsing can also be computed in $O(n)$ time [11]. Note that, on a text family like $T = a^n$, it holds that $z = 2$ and $z_{no} = \Theta(\log n)$, and thus it holds that $z_{no} = \Omega(z \log n)$.

Little is known about the relation between b and z except that $z \geq b$ by definition (z is the smallest *left-to-right* parsing) and that, for any constant $\epsilon > 0$, there is an infinite family of strings for which $b < (\frac{1}{2} + \epsilon) \cdot \min(z, z^R)$ [57, Cor. 7.1], where z^R is the z value of the reversed string.

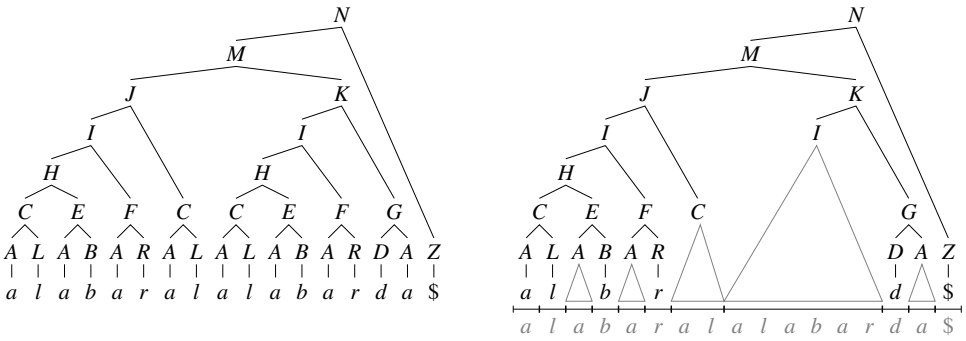

Fig. 1. The parse tree (left) and the grammar tree (right) of an example text. Only the black elements on the right form the grammar tree; the text coverage is conceptual.

Apart from being used as a gold standard to measure repetitiveness, the size of the Lempel-Ziv parse is bounded by the statistical entropy [41]. In particular, if H_k denotes the per-symbol k-th order empirical entropy of the text [45], then it holds that $z_{no} \log_2 n \leq nH_k + o(n \log_\sigma n)$ whenever $k = o(\log_\sigma n)$ [39] (thus, in particular, $z_{no} = O(n/\log_\sigma n)$).

Example: Consider again the text $T = alabaralalabarda\$$. The Lempel-Ziv parse (with or withour overlaps) has $z = z_{no} = 11$ phrases, $\underline{a}|\underline{l}|a|\underline{b}|a|\underline{r}|ala|labar|\underline{d}|a|\underline{\$}$, where we have underlined the explicit symbols. A corresponding function is

$$f[1\,..\,17] = \langle 0, 0, 1, 0, 3, 0, 1, 2, 3, 2, 3, 4, 5, 6, 0, 13, 0\rangle.$$

For example, the source of phrase $Z_7 = T[7\,..\,9] = ala$ is $T[1\,..\,3]$, and the source of phrase $B_8 = T[10\,..\,14]$ is $T[2\,..\,6]$.

D. Grammar Compression (g, g_{rl})

Consider a context-free grammar that generates T and only T [35]. For simplicity we stick to the particular case of *straight-line programs (SLPs)*, which are sequences of rules of the form $A \to a$ or $A \to BC$, where a is a terminal and A, B, C are nonterminals. Each nonterminal is defined as the left-hand side of exactly one rule, and the right-hand nonterminals must have been defined before in the sequence. The *expansion* of each nonterminal is the string it generates, that is, $exp(A) = a$ if $A \to a$ and $exp(A) = exp(B) \cdot exp(C)$ if $A \to BC$. The *initial symbol* of the SLP is the last nonterminal S in the sequence, so that the SLP generates the text $T = exp(S)$. The *size* of the SLP is its number of rules; it is assumed that every nonterminal is reachable from the initial symbol. We can stick to SLPs to obtain asymptotic results because any context-free grammar with size g (sum of lengths of right-hands of rules) can be easily converted into an SLP of at most g rules. In general, we will use g to denote the minimum possible size of an SLP that generates T, which is NP-complete to compute [52], [8].

If we allow, in addition, rules of the form $X \to Y^t$ for an integer $t > 0$, the result is a *run-length SLP (RLSLP)* [47]. The rule, assumed to be of constant size, means that X expands to t copies of Y, $exp(X) = exp(Y)^t$. We will use g_{rl} to denote the number of rules of the smallest RLSLP that generates T. Thus, it is clear that $g_{rl} \leq g$. Further, on the string family $T = a^n$ it holds that $g_{rl} = 2$ $g = \Theta(\log n)$, and thus it holds that $g = \Omega(g_{rl} \log n)$ (as well as $z_{no} = \Omega(g_{rl} \log n)$).

A well-known relation between z_{no} and g is $z_{no} \leq g = O(z_{no} \log(n/z_{no}))$ [52], [8]. Further, it is known that $g = O(z \log(n/z))$ [22, Lem. 8]. Those papers, as well as several others [54], [27], [28], exhibit $O(\log n)$-approximations to the smallest grammar. A negative result about the approximation are string families where $g = \Omega(z_{no} \log n/\log\log n)$ [8], [25] and even $g_{rl} = \Omega(z_{no} \log n/\log\log n)$ [5]. The size g is also bounded in terms of the statistical entropy [35] and of the empirical entropy [48], thus it always holds that $g = O(n/\log_\sigma n)$.

The *parse tree* of an SLP has a root labeled with the initial symbol and leaves labeled with terminals, which spell out T when read left to right. Each internal node labeled with A has a single leaf child labeled with a if $A \to a$, or two internal children labeled with B and C if $A \to BC$. The *grammar tree* (cf. partial parse tree [52]) prunes the parse tree by leaving only one internal node labeled with X for each nonterminal X; all the others are pruned and converted to leaves. Then, for an SLP of size g, the grammar tree has exactly g internal nodes. Since the right-hand sides of the rules are of size 1 or 2, each internal node has 1 or 2 children, but there is at least one with 1 child (the grammar must mention some terminal symbol). Thus, the total number of nodes is at most $2g$, and then the grammar tree has at most g leaves.

Example: We can generate the text $T = alabaralalabarda\$$ with an SLP of 16 rules (so $g \leq 16$): $A \to a$, $B \to b$, $D \to d$, $L \to l$, $R \to r$, $Z \to \$$, $C \to AL$, $E \to AB$, $F \to AR$, $G \to DA$, $H \to CE$, $I \to HF$, $J \to IC$, $K \to IG$, $M \to JK$, $N \to MZ$. The nonterminal N is the initial symbol. Figure 1 illustrates the parse and the grammar trees.

E. Collage Systems (c)

Collage systems [34] are a generalization of RLSLPs that also support *truncation*. Specifically, nonterminals can be of the form $A \to a$ for a terminal a, $A \to BC$ for previous nonterminals B and C, or $A \to B^k$, $A \to B^{[t]}$ or $A \to^{[t]} B$ for a previous nonterminal B and positive integers k and t. The last two rules mean that $exp(A) = exp(B)[1\,..\,t]$ and $exp(A) = exp(B)[|exp(B)| - t + 1\,..\,|exp(B)|]$, respectively (it must hold that $t \leq |exp(B)|$). We denote by c the number of rules of the smallest collage system generating (only) a text T.

Few relations are known between c and other repetitiveness measures, other than $c \leq g_{rl}$ and $c = O(z \log z)$ [34].

Example: The following collage system to generate the text $T = alabaralalabarda\$$ is actually less efficient than the SLP of the previous example (it uses 17 rules), but it illustrates all the operations: $A \to a$, $B \to b$, $D \to d$, $L \to l$, $R \to r$, $Z \to \$$, $C \to AL$, $E \to C^3$, $F \to BA$, $G \to FR$, $H \to DA$, $I \to HZ$, $J \to E^{[5]}$, $K \to JG$, $M \to {}^{[6]}K$, $N \to MK$, $O \to NI$. The nonterminal O is the initial symbol. This example also illustrates that the concepts of parse and grammar tree do not apply to collage systems; for example the nonterminal E expands to $alalal$, which does not appear in the text.

In this article we will be interested in a subclass we call *internal collage systems*, where there is a path of non-truncation rules from the initial symbol to every nonterminal. This implies that, every time we use a truncation rule on a nonterminal A, the whole $exp(A)$ appears somewhere else in T. Since, in internal collage systems, we might not always be allowed to use a prefix plus a suffix truncation to extract a substring of another rule, we explicitly allow in internal collage systems for *substring truncation* rules $A \to B^{[t,t']}$, with $1 \leq t \leq t' \leq |exp(B)|$, meaning that $exp(A) = exp(B)[t\,..\,t']$.

Note that any upper bound we prove for the size c of the smallest internal collage system also holds for the smallest general collage system. In particular, we prove $c = O(z)$, which is an improvement upon the previous result $c = O(z \log z)$ that holds for the smallest general collage system [34]. Instead, an existing lower bound on c of the form $\gamma = O(c)$, where γ is the size of the smallest "attractor" for T [33, Thm. 3.5], holds in fact only for internal collage systems, because it assumes, precisely, that the expansion of every nonterminal appears in T.[4] We also prove that $b = O(c)$ for internal collage systems, which improves upon their result because $\gamma = O(b)$ [33].

F. Suffix Arrays and Runs in the Burrows-Wheeler Transform (r)

The *suffix array* [43] of $T[1\,..\,n]$ is an array $SA[1\,..\,n]$ storing a permutation of $[1\,..\,n]$ so that, for all $1 \leq i < n$, the suffix $T[SA[i]\,..]$ is lexicographically smaller than the suffix $T[SA[i+1]\,..]$. Thus $SA[i]$ is the starting position in T of the ith smallest suffix of T in lexicographic order. The suffix array can be built in $O(n)$ time [36], [37], [30].

The inverse permutation of SA, $ISA[1\,..\,n]$, is called the *inverse suffix array*, so that $ISA[j]$ is the lexicographical position of the suffix $T[j\,..\,n]$ among the suffixes of T. It can be built in linear time by inverting the permutation SA.

The *longest common prefix array*, $LCP[1\,..\,n]$, stores at $LCP[i]$ the length of the longest common prefix between $T[SA[i]\,..]$ and $T[SA[i-1]\,..]$, with $LCP[1] = 0$. It can be built in linear time from T and ISA [31].

The *Burrows-Wheeler Transform* of T, $BWT[1\,..\,n]$ [7], is a string defined as $BWT[i] = T[SA[i]-1]$ if $SA[i] > 1$, and $BWT[i] = T[n] = \$$ if $SA[i] = 1$. That is, BWT has the same symbols of T in a different order, and is a reversible transform.

The array BWT can be easily obtained from T and SA, and thus it can also be built in linear time. To obtain T from BWT in linear time [7], one considers two arrays, $L[1\,..\,n] = BWT$ and $F[1\,..\,n]$, which contains all the symbols of L (or T) in ascending order. Alternatively, $F[i] = T[SA[i]]$, so $F[i]$ follows $L[i]$ in T. We need a function that maps any $L[i]$ to the position j of that same symbol in F. The function is

$$LF(i) \;=\; C[c] + \mathrm{rank}[i],$$

where $c = L[i]$, $C[c]$ is the number of occurrences of symbols less than c in L, and $\mathrm{rank}[i]$ is the number of occurrences of symbol $L[i]$ in $L[1\,..\,i]$. Once C and rank are computed, we reconstruct $T[n] = \$$ and $T[n-k] \leftarrow L[LF^{k-1}(1)]$ for $k = 1, \ldots, n-1$. Note that, if $L[i-1] = L[i]$, then $LF(i-1) = LF(i) - 1$; this result will be relevant later.

The number r of equal-symbol runs in the BWT of T can be bounded in terms of the empirical entropy, $r \leq nH_k + \sigma^k$ [42]. However, the measure is also interesting on highly repetitive collections (where, in particular, z and z_{no} are small). For example, it holds that $z = \Omega(r \log n)$ on the Fibonacci words [50]. However, this result assumes that T is not \$-terminated, but that lexicographical comparisons take T as a circular string. We will obtain similar results on our \$-terminated model, which is compatible with the use of r in compressed text indexes. On the de Bruijn sequences on binary alphabets, instead, it holds that $r = \Omega(z_{no} \log n)$ [1], [50]: we have $r = \Theta(n)$, whereas z_{no} is always $O(n/\log n)$ on binary strings. A recent result [32] is that $r = O(z \log(n/z) \log z)$.

Example: The BWT of $T = alabaralalabarda\$$ is $L = adll\$lrbbaaraaaaa$, which has $r = 10$ runs.

[4]For example, with the collage system $A \to a$, $B \to b$, $A' \to A^5$, $B' \to B^5$, $C \to AB$, $D \to C^{[9]}$, and the initial symbol $E \to^{[8]} D$, we generate the text $T = a^4 b^4$. However, because C does not appear in T, they fail to place an attractor element inside the substring ab.

G. Locally consistent parsing

A string can be parsed in a *locally consistent* way, which means that equal substrings are largely parsed in the same form. We use a variant of locally consistent parsing due to Jeż [27], [26].

Definition 2. *A* repetitive area *in a string is a maximal run of the same symbol, of length 2 or more.*

Definition 3. *Two intervals contained in* $[1\,..\,n]$ overlap *if they are not disjoint nor one contained in the other.*

Definition 4. *A parsing of a string into* blocks *is defined by, first, creating new symbols that represent the repetitive areas. On the resulting sequence, the alphabet (which contains original symbols and created ones) is partitioned into two subsets, left- and right-symbols. Then, every left-symbol followed by a right-symbol are paired in a block. The other isolated symbols form a block on their own.*

Jeż [27] shows that those blocks define a locally consistent parsing and that they shorten the string by a constant factor.

Lemma 1 ([27])**.** *We can choose left- and right-symbols so that Def. 4 partition a string* $S[1\,..\,\ell]$ *into at most* $(3/4)\ell$ *blocks.*

Example: A locally-consistent parsing of $T = alabaralalabarda\$$ can be obtained by considering a to be a left-symbol and all the others to be right-symbols. The resulting parsing into blocks is then $T = al|ab|ar|al|al|ab|ar|d|a\$$, where for example in the two occurrences of $alabar$, the sequence of blocks covering $laba$ are identical, $al|ab|ar$.

III. Upper Bounds

In this section we obtain our main upper bound, $z = O(b\log(n/b))$, along with other byproducts. To this end, we first prove that $g_{rl} = O(b\log(n/b))$, and then that $z \le g_{rl}$. To prove the first bound, we build an RLSLP on top of a bidirectional scheme. The grammar is built in several rounds of locally consistent parsing on the text. In each round, the blocks of the locally consistent parsing are converted into nonterminals and fed to the next round. The key is to prove that distinct nonterminals are produced only near the boundaries of the phrases of the bidirectional scheme. The second bound is an easy extension of the known result $z_{no} \le g$.

Example: Let us show how this works on the bidirectional scheme example of Section II-B, $al\mathbf{a}|\mathbf{b}|\mathbf{a}|\mathbf{r}|\mathbf{a}|\mathbf{l}|alabar|\mathbf{d}|\mathbf{a}|\$$. We have selected (in bold) one of each of the different blocks created in the example of Section II-G. Note that we do not need to select any block that is completely inside a phrase. The next lemma proves that the general case is only slightly worse.

Lemma 2. *Let a string* W *have a bidirectional macro scheme of size* b*. Then, if we cut it into blocks as per Def. 4, there will be at most* $4b$ *different blocks.*

Proof. Recall from Section II-B that our bidirectional scheme represents W as a sequence of *phrases*, by means of a function f. To count the number of different blocks produced, we will pessimistically assume that the first two and the last two blocks intersecting each phrase are all different. The number of such *bordering* blocks is then at most $4b$. On the other hand, we will show that non-bordering blocks do not need to be considered, because they will be counted somewhere else, when they appear near the border of a phrase.

We consider both types of non-bordering blocks resulting from Def. 4. Figure 2 illustrates both cases.

1) The block is a pair of left- and right-alphabet symbols.[5] As these symbols can be an original symbol or a repetitive area, let us write the pair generically as $X = a^{\ell_a}b^{\ell_b}$, for some $\ell_a, \ell_b \ge 1$, and let $\ell = \ell_a + \ell_b$ be the length of the block X. If $W[p\,..\,p+\ell-1] = X$ is not bordering, then it is strictly contained in a phrase. Thus, by the definition of a phrase, it holds that $[f(p-1)\,..\,f(p+\ell)] = [f(p)-1\,..\,f(p)+\ell]$, and that $W[f(p)-1\,..\,f(p)+\ell] = W[p-1\,..\,p+\ell]$. That is, the block appears again at $[f(p)\,..\,f(p)+\ell-1]$, surrounded by the same symbols. Since Def. 4 first compacts repetitive areas, it must be $W[f(p)-1] = W[p-1] \ne a$ and $W[f(p)+\ell] = W[p+\ell] \ne b$. Further, since Def. 4 pairs left- with right-symbols, a^{ℓ_a} must be a left-symbol and b^{ℓ_b} must be a right-symbol. The locally consistent parsing must then also form a block $W[f(p)\,..\,f(p)+\ell-1] = X$. If this block is bordering, then it will be counted. Otherwise, by the same argument, $W[f(p)-1\,..\,f(p)+\ell]$ will be equal to $W[f^2(p)-1\,..\,f^2(p)+\ell]$ and a block will be formed with $W[f^2(p)\,..\,f^2(p)+\ell-1]$. Since f has no cycles, there is a $k>0$ for which $f^k(p)=0$. Thus for some $l \le k$ it must be that $X = W[f^l(p)\,..\,f^l(p)+\ell-1]$ is bordering. At the smallest such l, the block $W[f^l(p)\,..\,f^l(p)+\ell-1]$ will be counted. Therefore, $X = W[p\,..\,p+\ell-1]$ is already counted somewhere else.
2) The block is a single (original or maximal-run) symbol $W[p\,..\,p+\ell-1] = a^\ell = X$, for some $\ell \ge 1$. It also holds that $[f(p-1)\,..\,f(p+\ell)] = [f(p)-1\,..\,f(p)+\ell]$ and $W[f(p)-1\,..\,f(p)+\ell] = W[p-1\,..\,p+\ell]$, because a^ℓ is strictly inside a phrase. Since $W[f(p)-1] = W[p-1] \ne a$ and $W[f(p)+\ell] = W[p+\ell] \ne a$, the parsing forms the same maximal run $X = a^\ell = W[f(p)\,..\,f(p)+\ell-1]$. Moreover, since $W[p\,..\,p+\ell-1]$ is not bordering, the previous and next blocks produced by the parsing, $Y = W[p'\,..\,p-1]$ and $Z = [p+\ell\,..\,p'']$, are also strictly inside the same phrase, and therefore

[5] For this case, we could have defined *bordering* in a stricter way, as the first or last block of a phrase.

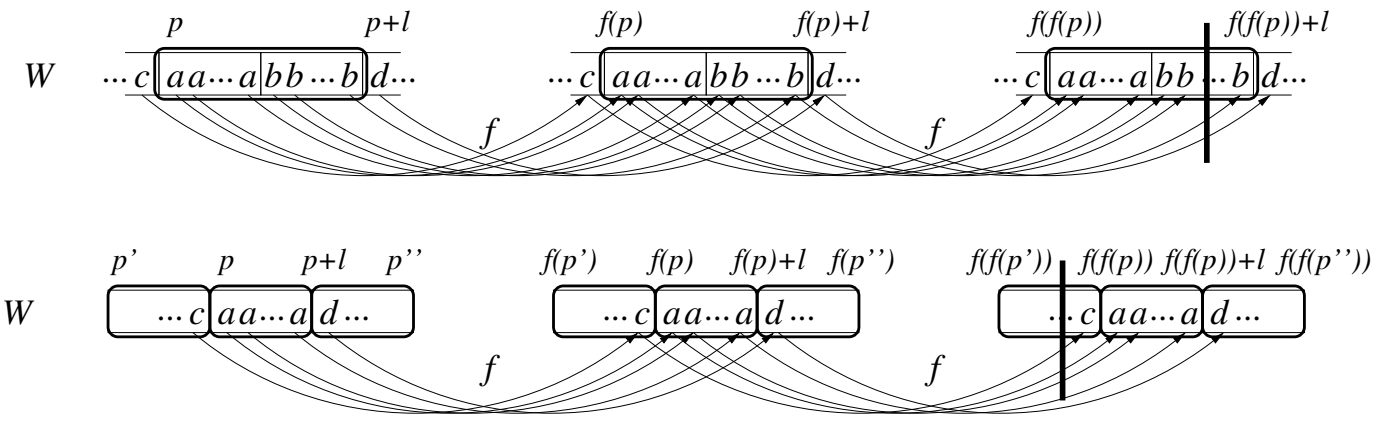

Fig. 2. The two cases of Lemma 2. Case 1 is shown on top, where a block $X = W[p\,..\,p+\ell-1] = a^{\ell_a}b^{\ell_b}$ is formed because they are left- and right-symbols surrounded by $c \neq a$ and $d \neq b$. Since all the symbols are strictly inside a phrase because X is non-bordering, function f maps them together elsewhere in W preserving their contents, so the same block is formed at $W[f(p)\,..\,f(p)+\ell-1] = X$. This is repeated until a phrase boundary (thick vertical line) appears near X (so that the occurrence of X is bordering). Case 2 is shown on the bottom, where $X = W[p\,..\,p+\ell-1] = a^\ell$ is not paired and thus forms a single block surrounded by $c, d \neq a$. Again, the same contents are found, and the same blocks are formed, at $W[f(p)\,..\,f(p)+\ell-1] = X$ because the blocks $Y = W[p'\,..\,p-1]$ and $Z = W[p+\ell\,..\,p'']$ are strictly inside a phrase. Again, this is repeated until hitting a phrase boundary nearby.

they also appear preceding and following $W[f(p)\,..\,f(p)+\ell-1]$, at $Y = W[f(p')\,..\,f(p)-1]$ and $Z = [f(p)+\ell\,..\,f(p'')]$. Since a^ℓ was paired neither Y nor Z at $W[p\,..\,p+\ell-1]$, the parsing will also not pair them at $W[f(p)\,..\,f(p)+\ell-1]$. Therefore, the parsing will leave a^ℓ as a block also in $[f(p)\,..\,f(p)+\ell-1]$. If $W[f(p)\,..\,f(p)+\ell-1]$ is bordering, then it will be counted, otherwise we can repeat the argument with $W[f^2(p)-1\,..\,f^2(p)+\ell]$ and so on, as before.

□

Lemma 2 shows that the number of *different* blocks we form with the locally consistent parsing of Def. 4 is $O(b)$. We now show that the sequence of blocks obtained has a bidirectional macro scheme of size $O(b)$.

Lemma 3. *If W has a bidirectional macro scheme, we can define one on the sequence W' of blocks obtained in Lemma 2, by adding at most two new explicit symbols at the beginning and two at the end of each non-explicit phrase.*

Proof. We define a bidirectional scheme on W' as follows:

1) For each bordering block in W, its nonterminal symbol position in W' is made explicit in the bidirectional scheme of W'. Note that this includes the blocks covering the explicit symbols in the bidirectional scheme of W. This creates at two new explicit symbols at the beginning and two at the end of each non-explicit phrase in W.
2) For the phrases $B_i = W[t_i\,..\,t_i+\ell_i-1]$ of W containing non-bordering blocks (note B_i cannot be an explicit symbol), let B'_i be obtained by trimming from B_i the bordering blocks near the boundaries of B_i. Then B'_i appears inside $W[s_i\,..\,s_i+\ell_i-1]$ (with $s_i = f(t_i)$), where the same sequence of blocks is formed, as shown for each such block in Lemma 2. We then form a phrase in W' with the sequence of blocks in B'_i (all of which are non-bordering), with the function f' of W' pointing to the identical sequence of blocks that appear inside $W[s_i\,..\,s_i+\ell_i-1]$.

It is easy to see that f' is acyclic. Let $m(i)$ be the position of W' where the block containing $W[i]$ is mapped. Then, if $W[p\,..\,p+\ell-1]$ is a block inside some B'_i, and it points to $W[s\,..\,s+\ell-1]$ with $s = f(p)$, we have that $f'(m(p+t)) = m(s+t')$ for all $0 \leq t, t' < \ell$. Thus, if there is a cycle in f', there must be a cycle in f. □

Example: On our preceding example, $al\mathbf{a}|\mathbf{b}|\mathbf{a}|\mathbf{r}|\mathbf{a}|\mathbf{l}|alabar|\mathbf{d}|\mathbf{a}|\$$, we define the blocks $A = ab$, $B = ar$, $C = al$, and $D = a\$$. We then create $W' = C|\underline{A}|\underline{B}|\underline{C}|CAB|\underline{d}|\underline{D}$, where we show the derived bidirectional scheme and underline the explicit symbols. The corresponding function is $f'[1\,..\,9] = \langle 4,0,0,0,1,2,3,0,0\rangle$, obtained by projecting the function f of the example in Section II-B. Recall that, to make this small example interesting, we have been stricter about which blocks are bordering.

We now have all the elements to define our RLSLP grammar of size $O(b\log(n/b))$.

Theorem 4. *There always exists an RLSLP of size $g_{rl} = O(b\log(n/b))$ that generates T.*

Proof. We create the grammar by various rounds of parsing, starting from the string $W_0 = T$ and obtaining string W_k from W_{k-1} in the kth round. Let $b_0 = b$ the size of the smallest bidirectional macro scheme of T, and b_k be the size of the one we build on W_k for $k > 0$. In the kth round, we first apply Lemma 2 on W_{k-1}, producing at most $4b_{k-1}$ distinct blocks. We then associate a nonterminal with each distinct block, and create a reduced sequence W_k from W_{k-1} by replacing all the blocks of length 2 or more by their corresponding nonterminals. The new sequence has length $|W_k| \leq (3/4)|W_{k-1}|$ by Lemma 1.

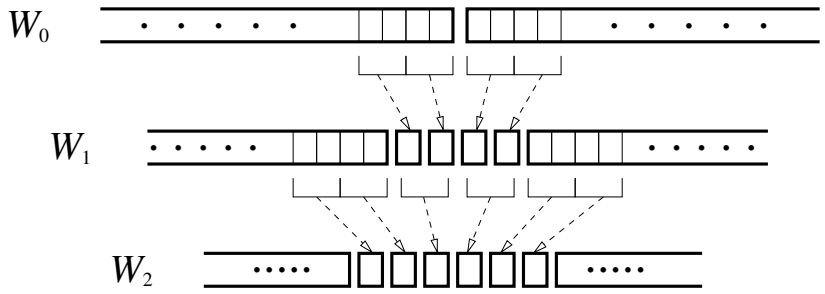

Fig. 3. Illustration of Theorem 4. On top we see the border between two long phrases of W_0. In this example, the blocks always pair two symbols. We show below W_0 the 4 bordering blocks formed with the symbols nearby the boundary. Below, in W_1, those blocks are converted into 4 explicit symmbols. This region of 4 symbols is then parsed into 2 blocks. The parsing also creates 4 new bordering blocks from the boundaries of the long phrases. In W_2, below, we have now a region of 6 explicit symbols. They would have been 8, but we created 2 distinct blocks that reduced their number to 6.

In turn, Lemma 3 shows that we can create a bidirectional macro scheme for W_k from that of W_{k-1}, adding at most 2 new explicit symbols at the beginning and 2 at the end of each non-explicit phrase. The structure of these macro schemes then consists of growing *regions* of explicit symbols at the boundaries of the same b phrases. Consider a phrase X_0 in W_0. In W_1, there may appear at most two explicit symbols at the beginning and two at the end of X_0 (coming from the bordering blocks). Let X_1 be the remainder of X_0, projected to W_1. Then, when forming W_2, two new explicit symbols may appear at the beginning and two at the end end of X_1, and so on. Therefore, if we call e_k the number of explicit symbols in W_k, we have that $e_0 = 0$ and $e_k \leq 4bk$ for every round $k > 0$.

To count how many distinct blocks (and hence nonterminals) are produced, we only have to consider the bordering blocks, as shown in Lemma 2. As discussed above, those bordering blocks are at most $4bk$ in W_k (they correspond to the explicit symbols). However, the parsing of the regions themselves may also produce new distinct blocks. Our aim is to show that the number of those blocks is also bounded because they reduce the length of the regions, which only grow by $4b$ (explicit symbols) per iteration. Intuitively, each block created inside a region decreases its length, and thus both numbers cancel out.

To make the argument more precise, let n_k be the number of new distinct blocks produced when parsing the regions of W_{k-1}. Therefore it holds that the number d_k of distinct blocks produced in the kth iteration is at most $4b + n_k$, and the total number of distinct blocks created up to building W_k is

$$\sum_{i=1}^{k} d_i \;\leq\; 4bk + \sum_{i=1}^{k} n_i.$$

On the other hand, for each of the n_k blocks created when parsing a region of W_{k-1}, the length of the region decreases at least by 1 in W_k, that is, there is one less explicit symbol in W_k. Then it holds that $e_k \leq e_{k-1} + 4b - n_k$, and thus

$$0 \;\leq\; e_k \;\leq\; 4bk - \sum_{i=1}^{k} n_i.$$

It follows that $\sum_{i=1}^{k} n_i \leq 4bk$, and thus

$$\sum_{i=1}^{k} d_i \;\leq\; 8bk.$$

The idea is illustrated in Figure 3.

We may need up to 3 rules to represent each distinct block: for $X = A^{\ell_a}B^{\ell_b}$, if $\ell_a, \ell_b > 1$, we need rules $A' \to A^{\ell_a}$, $B' \to B^{\ell_b}$, and $C \to A'B'$, assuming a and b are already nonterminals. In addition, we may need σ rules of the form $A \to a$ for terminals a. In total, since there are at most $8bk$ distinct blocks, we need at most $3 \cdot 8bk + \sigma \leq 25bk$ rules.

After k rounds, the sequence is of length at most $(3/4)^k n$ and we have generated $O(bk)$ nonterminals. Therefore, if we choose to perform $k = \log_{4/3}(n/b)$ rounds, the sequence will be of length at most b and the RLSLP size will be $O(b \log(n/b))$. To complete the process, we add $O(b)$ nonterminals to reduce the sequence to a single initial symbol. □

With Theorem 4, we can also bound the size z of the Lempel-Ziv parse [41] that allows overlaps. The size without allowing overlaps is known to be bounded by the size of the smallest SLP, $z_{no} \leq g$ [52], [8]. We can easily see that $z \leq g_{rl}$ also holds by extending an existing proof [8, Lem. 9] to handle the run-length rules. We call any parsing of T where every new phrase is a symbol or it occurs previously in T a *left-to-right parse*.

Theorem 5. *The Lempel-Ziv parse (allowing overlaps) of T always produces $z \leq g_{rl}$ phrases.*

Proof. Consider the grammar tree of T (Section II-D), where only the leftmost occurrence of each nonterminal X is an internal node. Our left-to-right parse of T is a sequence Z obtained by traversing the leaves of the grammar tree left to right. For a

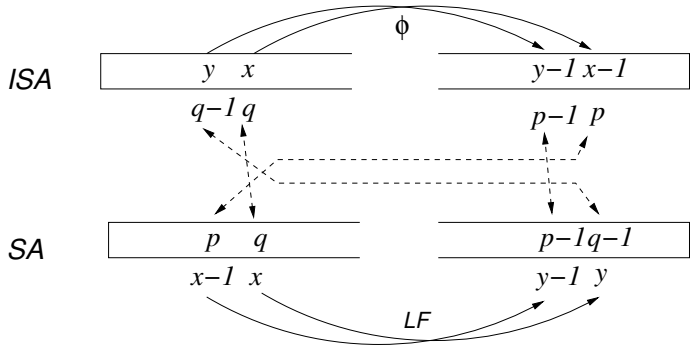

Fig. 4. Illustration of Lemma 8.

terminal leaf, we append the explicit symbol to Z. For a leaf representing nonterminal X, we append to Z a reference to the area $T[x\,..\,y]$ expanded by the leftmost occurrence of X.

To extend grammar trees to RLSLPs, we handle rules $X \to Y^t$ as follows. First, we expand them to $X \to Y \cdot Y^{t-1}$, that is, the node for X has two children for Y, the second annotated with $t-1$. Since the right child of X is not the first occurrence of Y, it must be a leaf. The left child of X may or may not be a leaf, depending on whether Y occurred before or not. Since run-length rules become internal nodes with two children, it still holds that the grammar tree has at most g_{rl} leaves.

Now, when our leaf traversal reaches the right child Y of a node X indicating $t-1$ repetitions, we append to Z a reference to $T[x\,..\,y+(t-2)(y-x+1)]$, where $T[x\,..\,y]$ is the area expanded by the first child of X. Note that source and target overlap if $t > 2$. Thus a left-to-right parse of size g_{rl} exists, and the result follows because Lempel-Ziv is the optimal left-to-right parse [41, Thm. 1]. □

By combining Theorems 4 and 5, we obtain a result on the long-standing open problem of finding the approximation ratio of Lempel-Ziv compared to the smallest bidirectional scheme.

Theorem 6. *The Lempel-Ziv parsing of T allowing overlaps always has $z = O(b \log(n/b))$ phrases.*

We can also derive upper bounds for g and z_{no}. It is sufficient to combine Theorem 6 with the facts that $g = O(z \log(n/z))$ [22, Lem. 8] and $z_{no} \leq g$ [52], [8].

Lemma 7. *It always holds that $g, z_{no} = O(b \log^2(n/b))$.*

IV. Lower Bounds

In this section we prove that the upper bound of Theorem 6 is tight as a function of n, by exhibiting a family of strings for which $z = \Omega(b \log n)$. This confirms that the gap between bidirectionality and unidirectionality is significantly larger than what was previously known. The idea is to define phrases in T according to the r runs in the BWT, and to show that these phrases induce a bidirectional scheme of size $2r$. This proves that $r = \Omega(b)$. Then we resort to a well-known family of strings where $z = \Omega(r \log n)$.

Definition 5. *Let $p_1, p_2, \ldots, p_r$ be the positions that start runs in the BWT, and let $t_1 < t_2 < \ldots < t_r$ be the corresponding positions in T, $\{SA[p_i] \mid 1 \leq i \leq r\}$, in increasing order. Note that $t_1 = 1$ because $BWT[ISA[1]] = \$$ is a size-1 run, and let $t_{r+1} = n+1$, so that T is partitioned into* phrases *$T[t_i\,..\,t_{i+1}-1]$. Let also $\phi(p) = SA[ISA[p]-1]$ if $ISA[p] > 1$ and $\phi(p) = SA[n]$ otherwise. Then we define the* bidirectional scheme of the BWT:

1) *For each $1 \leq i \leq r$, $T[t_i\,..\,t_{i+1}-2]$ is copied from $T[\phi(t_i)\,..\,\phi(t_{i+1}-2)]$.*
2) *For each $1 \leq i \leq r$, $T[t_{i+1}-1]$ is an explicit symbol.*

Example: The BWT runs of the example of Section II-F induces the bidirectional scheme $\underline{a}|\underline{l}|\underline{a}|\underline{b}|a|\underline{r}|a|\underline{l}|alaba|\underline{r}|\underline{d}|\underline{a}|\underline{\$}$, with function $f[1\,..\,17] = \langle 0,0,0,0,7,0,9,0,1,2,3,4,5,0,0,0,0\rangle$.

We build on the following lemma, illustrated in Figure 4. We make use of the function LF defined in Section II-F. Note that $LF(x) = ISA[SA[x]-1]$ if $SA[x] > 1$ and $LF(x) = ISA[n] = 1$ if $SA[x] = 1$. That is, LF moves in SA to the suffix preceding the current one in T. The analogous function moving in T to the suffix preceding the current one in SA is ϕ.

Lemma 8. *Let $[q-1\,..\,q]$ be within a phrase of Def. 5. Then it holds that $\phi(q-1) = \phi(q)-1$ and $T[q-1] = T[\phi(q)-1]$.*

Proof. Consider the pair of positions $T[q-1\,..\,q]$ within a phrase. Let them be pointed from $SA[x] = q$ and $SA[y] = q-1$, therefore $ISA[q] = x$, $ISA[q-1] = y$, and $LF(x) = y$. Now, since q is not a position at the beginning of a phrase, x is not the first position in a BWT run. Therefore, $BWT[x-1] = BWT[x]$. Recalling the formula of Section II-F to compute

$LF(x) = C[c] + \text{rank}[x]$, where $c = BWT[x]$, it follows that $LF(x-1) = LF(x) - 1 = y - 1$. Now let $SA[x-1] = p$, that is, $p = \phi(q)$. Then,

$$\phi(q-1) \;=\; SA[ISA[q-1]-1] \;=\; SA[y-1] \;=\; SA[LF(x-1)] \;=\; SA[x-1]-1 \;=\; p-1 \;=\; \phi(q)-1.$$

It also follows that

$$T[q-1] \;=\; BWT[x] \;=\; BWT[x-1] \;=\; T[p-1] \;=\; T[\phi(q)-1].$$

□

Example: The suffix array of $T = alabaralalabarda\$$ is $SA = \langle 17, 16, 3, 11, 1, 9, 7, 5, 13, 4, 12, 15, 2, 10, 8, 6, 14\rangle$ and the ϕ function is $\phi = \langle 11, 15, 16, 13, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 12, 17, 14\rangle$. For example, $\phi(1) = 11$ because the suffix lexicographically preceding $T[1\,..]$ is $T[11\,..]$. Now, let $q = 10$, which is inside the same phrase of $q-1 = 9$ in the parse $\underline{a}|\underline{l}|\underline{a}|\underline{b}|a|\underline{r}|a|\underline{l}|alaba|\underline{r}|\underline{d}|\underline{a}|\underline{\$}$ induced by the run heads of the BWT of T, $BWT = adll\$lrbbaaraaaaa$. Position $T[q=10]$ is pointed from $SA[x=14]$, whereas $T[q-1=9]$ is pointed from $SA[y=6]$. Thus $LF(x=14) = C[BWT[14] = a] + \text{rank}[14] = 1 + 5 = 6 = y$. Since $q = 10$ does not start a phrase in T, $BWT[x=14]$ does not start a run, thus $BWT[x-1=13] = a$. It then holds that $LF(x-1=13) = C[BWT[13] = a] + \text{rank}[13] = 1 + 4 = 5 = y - 1 = LF(x=14) - 1$. Further, if we call $p = SA[x-1=13] = 2$, it holds that $p = 2 = \phi(q=10)$. One can then verify that $\phi(q-1=9) = SA[y-1=5] = 1 = SA[x-1=13] - 1 = \phi(q=10) - 1$, and that $T[q-1=9] = a = BWT[x=14] = BWT[x-1=13] = T[p-1=1] = T[\phi(q=10)-1]$.

Theorem 9. *The bidirectional scheme of the BWT is a valid bidirectional scheme, thus it always holds $b \leq 2r$.*

Proof. By Lemma 8, it holds that $\phi(q-1) = \phi(q) - 1$ if $[q-1\,..\,q]$ is within a phrase, and that $T[q-1] = T[\phi(q)-1]$. Therefore, we have that $\phi(t_i + k) = \phi(t_i) + k$ for $0 \leq k < \ell_i = t_{i+1} - t_i - 1$, and then $T[\phi(t_i)\,..\,\phi(t_{i+1}-2)]$ is indeed a contiguous range of length ℓ_i. We also have that $T[\phi(t_i)\,..\,\phi(t_{i+1}-2)] = T[t_i\,..\,t_{i+1}-2]$, and therefore the copy is correct.

It is also easy to see that we can recover the whole T from those $2r$ phrases. We can, for example, follow the cycle $\phi^k(n)$, $k = n-1, \ldots, 1$, and copy $T[\phi^k(n)]$ from $T[\phi^{k+1}(n)]$ unless the former is explicitly stored (note that $T[\phi^n(n)] = T[\phi^0(n)] = T[n]$ is stored explicitly). By Lemma 8, it is correct to copy from $T[\phi(p)]$ to $T[p]$ whenever p (which is $q-1$ in Lemma 8) is not at the end of a phrase; this is why we store the explicit symbols at the end of the phrases.

Since the bidirectional scheme of the BWT is of size $2r$, it follows by definition that $2r \geq b$. □

Example: We can recover T from our bidirectional scheme $\underline{a}|\underline{l}|\underline{a}|\underline{b}|a|\underline{r}|a|\underline{l}|alaba|\underline{r}|\underline{d}|\underline{a}|\underline{\$}$ by following positions $\phi^{n-1}(n), \ldots, \phi(n)$ and copying the last explicit symbol seen onto each new position. The sequence, where we indicate in parentheses the explicit symbols visited, is $16(a), 3(a), 11, 1(a), 9, 7, 5, 13, 4(b), 12, 15(d), 2(l), 10, 8(l), 6(r), 14(r)$. For example, the explicit a collected at $T[1]$ is copied onto $T[9]$, $T[7]$, $T[5]$, and $T[13]$.

We can now prove the promised separation betweeen z and b. Before, we prove a further property of the cyclic rotations of the Fibonacci words we make use of.

Lemma 10. *In every* even *Fibonacci word F_k, the lexicographically smallest cyclic rotation is the one that starts at the last character.*

Proof. Mantaci et al. [44] give a characterization of the cyclic rotations of the kth Fibonacci word F_k by defining two functions: $\varrho : [0\,..\,f_k - 1] \to [0\,..\,f_k - 1]$, defined as

$$\varrho(x) = x + f_{k-2} \pmod{f_k},$$

and $\varphi : [0\,..\,f_k - 1] \to \{a, b\}$, defined as

$$\varphi(x) = \begin{cases} a, & \text{if } x < f_{k-1} \\ b, & \text{if } x \geq f_{k-1}, \end{cases}$$

where they index the strings from position 0. They proved that the cyclic rotations of F_k are the words $R_x = r_0 r_1 \cdots r_{f_k - 1}$, where $r_i = \varphi(\varrho^i(x))$, for $0 \leq x \leq f_k - 1$. If k is even, then $F_k = R_{f_{k-2}}$ [44, Thm. 6]. Since $F_k[i] = R_{f_{k-2}}[i] = \varphi(\varrho^i(f_{k-2})) = \varphi(\varrho^{i+1}(0)) = R_0[i+1]$, for $1 \leq i \leq f_k - 1$, it holds that $R_0[2\,..\,f_k] = F_k[1\,..\,f_k - 1]$. Combining that R_0 is a cyclic rotation of F_k and $R_0[2\,..\,f_k] = F_k[1\,..\,f_k - 1]$, we have that $R_0 = F_k[f_k]F_k[1\,..\,f_k - 1]$. The ordering of the cyclic rotations of F_k is $R_0 < R_1 < \cdots < R_{f_k - 1}$ [44, proof of Thm. 9]. R_0 is the lexicographically smallest cyclic rotation. That proved that the lexicographically smallest cyclic rotation of F_k starts at its last position, f_k. Formally, $F_k[f_k]F_k[1\,..\,f_k - 1]$ is the lexicographically smallest cyclic rotation of F_k, for all even k. □

Theorem 11. *There is an infinite family of strings over an alphabet of size 2 for which $r = O(1)$ and $z = \Theta(\log n)$, and thus $z = \Omega(r \log n)$ and $z = \Omega(b \log n)$.*

Proof. As observed by Prezza [50, Thm. 25], for all Fibonacci words we have $r = O(1)$ [44, Thm. 9]. Combining it with the fact that, in all Fibonacci words, it holds $z = \Theta(\log n)$ [14], yields $z = \Omega(r \log n)$.

Note, however, that the result $r = O(1)$ is proved under a BWT definition that is different from ours [44]. Namely, the Fibonacci words are not terminated with \$, but instead the suffixes are compared cyclically, as if F_k were a circular word.

By Lemma 10, however, in each *even* Fibonacci word F_k, the lexicographically smallest cyclic suffix is the one that starts at the last character. From this observation we have that, in every even Fibonacci word F_k, the relative order of the cyclic suffixes is the same as the relative order of the suffixes terminated in \$. Formally, $F_k[i\,..\,f_k]F_k[1\,..\,i-1] < F_k[j\,..\,f_k]F_k[1\,..\,j-1]$ if and only if $F_k[i\,..\,f_k]\$ < F_k[j\,..\,f_k]\$$, for all $i \neq j$, and k even. Thus, in the even Fibonacci words, we have $r = O(1)$, and thus $z = \Omega(r \log n)$. The result $z = \Omega(b \log n)$ then follows from the fact that $b = O(r)$, by Theorem 9. □

Finally, by relating g with the empirical entropy of T, we can also prove a separation between r and g.

Lemma 12. *It always holds that $g \log_2 n \leq nH_k + o(n \log \sigma)$ for any $k = o(\log_\sigma n)$, thus $g = O(n/\log_\sigma n)$.*

Proof. Let z_{78} be the size of the Lempel-Ziv 1978 (LZ78) parsing [59] of T. Then, it holds that $z_{78} \log_2 n \leq nH_k + o(n \log \sigma)$ for $k = o(\log_\sigma n)$ [39, Thm. A.4] (noting that their c is $O(n/\log_\sigma n)$ and assuming $k = o(\log_\sigma n)$). Since this parsing can be converted into an SLP of size z_{78}, it holds that $g \leq z_{78}$ and the result follows. The final claim is a consequence of the fact that $H_k \leq \log \sigma$ for all k. □

Theorem 13. *There is an infinite family of strings over an alphabet of size 2 for which $r = \Omega(g \log n)$.*

Proof. By Lemma 12, the smallest SLP on a binary alphabet is always of size $g = O(n/\log n)$. On a de Bruijn sequence of order k on a binary alphabet we have $r = \Theta(n)$ [1]. The result follows. □

V. Greedy and Ordered Parses

In this section we extend the Lempel-Ziv parse, where sources must start before targets in the text, to the more general concepts of ordered parsings, and prove some general results on them.

Definition 6. *An* ordered parse *of $T[1\,..\,n]$ is a partition of T into u phrases $B_1, \ldots, B_u$, such that each phrase $B_i = T[t_i\,..\,t_i + \ell_i - 1]$ either is an explicit symbol or it is copied from a source $T[s_i\,..\,s_i + \ell_i - 1]$, such that $s_i \neq t_i$ and $T[s_i + j\,..] \prec T[t_i + j\,..]$ for all $0 \leq j < \ell_i$, for some suitable total order $\preceq$ on the suffixes of T.*[6]

By the way we define them, ordered parses are bound to be bidirectional schemes, and bidirectional schemes are ordered parses under some suitable order.

Lemma 14. *Every ordered parse is a bidirectional scheme.*

Proof. Let f be the function associated with the ordered parse, that is, $f(t_i + j) = s_i + j$ for all $0 \leq j < \ell_i$ if phrase $B_i = T[t_i\,..\,t_i + \ell_i - 1]$ is copied from $T[s_i\,..\,s_i + \ell_i - 1]$. There cannot be a cycle in f because, by definition, $T[f(p)\,..] \prec T[p\,..]$ for every position p inside every such phrase B_i. □

Lemma 15. *Every bidirectional scheme is an ordered parse under some suitable order $\preceq$.*

Proof. Let f be the function associated with the bidirectional scheme. Let us assign to every suffix $T[p\,..]$ the value $h(p) = \min\{k \geq 0 \mid f^k(p) = 0\}$. Now $\preceq$ can be any total order on $[1\,..\,n]$ compatible with $h(p)$, that is, such that if $h(p') < h(p)$ then $T[p'\,..] \prec T[p\,..]$ (e.g., topological sorting produces a valid order $\preceq$). Since the bidirectional scheme copies $T[p]$ from $T[f(p)]$ and $h(p) = 1 + h(f(p)) > h(f(p))$, it holds that $T[f(p)\,..] \prec T[p\,..]$. The parsing is then ordered under order $\preceq$. □

We are interested in parses that, while respecting a given order $\preceq$, produce the smallest number of phrases.

Definition 7. *A parse is* ordered-optimal *with respect to a total order $\preceq$ if no other ordered parse respecting the order $\preceq$ has fewer phrases on any text $T[1\,..\,n]$. We may or may not allow that sources and targets overlap to define optimality.*

Lempel-Ziv is an ordered parse with respect to the order $T[s_i\,..] \prec T[t_i\,..]$ iff $s_i < t_i$. The parses that respect this order are called left-to-right parses. As we have seen, then, Lempel-Ziv is ordered-optimal, either with or without overlaps [41], [57]. Further, the methods that obtain those optimal parses [51], [11] are *greedy*, under the following definition.

Definition 8. *A method to obtain an ordered parse of $T[1\,..\,n]$ is* greedy *if it proceeds left to right on T producing one phrase at each step, and such phrase is the longest possible one that starts at that position and has a smaller source in T under the order $\preceq$. If the longest possible phrase is of length 0 or 1, the parse may use an explicit symbol.*

Greedy methods are attractive on ordered parses because they produce the ordered-optimal parse and can be computed in polynomial time.

Theorem 16. *Every greedy parse is ordered-optimal.*

[6]The order is called $\preceq$ because it must be reflexive, yet we use $x \prec y$ to indicate $x \preceq y$ and $x \neq y$, that is, x is strictly smaller than y under order $\preceq$.

Proof. Let $B_1, \ldots, B_u$ be the result of the greedy parsing of T under order $\preceq$. Since the first phrase always starts at position 1, if there is another ordered parse $B'_1, \ldots, B'_{u'}$, where $u' < u$ and $B'_i = T[t'_i \,..\, t'_i + \ell'_i - 1]$ for $1 \leq i \leq u'$, then there must be a first phrase where $t'_{i+1} > t_{i+1}$. Since it is the first, it must hold that $t'_i \leq t_i < t_{i+1} < t'_{i+1}$. Let us call $\delta = t_i - t'_i < \ell'_i = t'_{i+1} - t'_i$. Therefore, there is a source $T[s'_i \,..\, s'_i + \ell'_i - 1] = T[t'_i \,..\, t'_i + \ell'_i - 1]$ such that $T[s'_i + j \,..] \prec T[t'_i + j \,..]$ for all $0 \leq j < \ell'_i$. Then it also holds that $T[s'_i + \delta \,..\, s'_i + \ell'_i - 1] = T[t_i \,..\, t'_{i+1} - 1]$ and that $T[s'_i + \delta + j \,..] \prec T[t_i + j \,..]$ for all $0 \leq j < t'_{i+1} - t_i$. Therefore, there exists a suffix $T[s'_i + \delta \,..]$ that shares with $T[t_i \,..]$ a prefix of length $t'_{i+1} - t_i > t_{i+1} - t_i = \ell_i$ and it can be used under order $\preceq$. This is impossible because our parsing is greedy and it did not choose that suffix when producing the phrase $T[t_i \,..]$. □

Theorem 17. *The greedy parse of any ordered parse can be obtained in $O(n^3)$ evaluations of $\prec$.*

Proof. We obtain the phrase lengths ℓ_i left to right, so that their starting points are $s_1 = 1$ and $s_{i+1} = s_i + \ell_i$. To find the length ℓ_i of each new phrase $T[s_i \,..\, s_i + \ell_i - 1]$, we compare the suffix $T[s_i \,..]$ with every other suffix $T[p \,..]$ symbol by symbol, until we find the smallest $j_p \geq 0$ such that $T[p + j_p] \succ T[s_i + j_p]$ or $p + j_p > n$ or $s_i + j_p > n$. We then have $\ell_i = \max_{p \neq s_i} j_p$. If $j_i = 0$ we create an explicit symbol in the parse. □

Of course, particular greedy parses, like Lempel-Ziv, can be obtained faster, in this case in time $O(n)$ [51], [11]. Interestingly, the fact that ordered-optimal parses are computed easily implies that we cannot efficiently find the order $\preceq$ that produces the smallest ordered parse.

Lemma 18. *Finding the order $\preceq$ that produces the smallest ordered parse on T is NP-hard.*

Proof. One of those orders $\preceq$ yields the smallest bidirectional scheme, by Lemma 15. Once we have the best order $\preceq$, we find the parse itself greedily in time $O(n^3)$, by Theorems 16 and 17. Thus we obtain the smallest bidirectional scheme, which is NP-hard to find [21]. □

On the other hand, we now show that, under certain favorable kinds of orders $\preceq$, the size of the ordered-optimal parses is upper bounded by the size of the smallest grammar. In particular, ordered-optimal parses that let sources and targets overlap are of size $O(b \log(n/b))$.

Definition 9. *A total order $\preceq$ on text suffixes is* extensible *if $T[s \,..] \prec T[t \,..]$ and $T[s] = T[t]$ implies that $T[s{+}1 \,..] \prec T[t{+}1 \,..]$.*

For example, the order of left-to-right parses, $T[s \,..] \prec T[t \,..]$ iff $s < t$, is extensible.

Theorem 19. *Any ordered-optimal parse of T, for any extensible order $\preceq$, produces $u \leq g$ phrases. Thus, $u \log_2 n \leq nH_k + o(n \log \sigma)$ for any $k = o(\log_\sigma n)$, $u = O(n/\log_\sigma n)$, and there are string families where $r = \Omega(u \log n)$.*

Proof. It suffices to show how to build an ordered parse of size at most g. Analogously to the proof of Theorem 5, consider a variant of the grammar tree of T where the only internal node labeled X and expanding to $T[x_i \,..\, z_i]$ is the one with the smallest suffix $T[x_i \,..]$ under order $\preceq$. This tree has up to g leaves, just like the original grammar tree. We then define an ordered parse of T by converting every terminal leaf to an explicit symbol, and every leaf covering $T[x'_i \,..\, z'_i]$, labeled by nonterminal X, to a phrase that points to the area $T[x_i \,..\, z_i]$ corresponding to the only internal node labeled X. Such a parse is of size $u \leq g$ and is ordered because the order is extensible: since $T[x'_i \,..\, z'_i] = T[x_i \,..\, z_i]$ and $T[x_i \,..] \prec T[x'_i \,..]$, it follows that $T[x_i + j \,..] \prec T[x'_i + j \,..]$ for all $0 \leq j \leq z_i - x_i$.

Since this is an ordered parse, the ordered-optimal parse is also of size $u \leq g$. The other results are immediate consequences of Lemma 12 and Theorem 13. □

Theorem 20. *Any ordered-optimal parse of T that allows sources and targets overlap, under any extensible order $\preceq$, produces $u \leq g_{rl}$ phrases. Thus it holds that $u = O(b \log(n/b))$.*

Proof. We extend the proof of Theorem 19 so as to consider the rules $X \to Y^t$. These can be expanded either to $X \to Y \cdot Y^{t-1}$ or to $X \to Y^{t-1} \cdot Y$. In both cases, the child Y is handled as usual (i.e., pruned if its suffix is not the smallest one labeled Y, or expanded otherwise). If we choose $X \to Y \cdot Y^{t-1}$, let Y expand to $T[x \,..\, y-1]$ and Y^{t-1} expand to $T[y \,..\, z]$. We then define $T[y \,..\, z]$ as the target of the source $T[x \,..\, x + z - y]$. If we instead choose $X \to Y^{t-1} \cdot Y$, then we define $T[x \,..\, x + z - y]$ as the target of the source $T[y \,..\, z]$. In both cases, the target overlaps the source if $t > 2$.

Note that one of those two cases must copy a source to a larger target, depending on whether $T[x \,..] \prec T[y \,..]$ or $T[y \,..] \prec T[x \,..]$. Further, the argument used in the proof of Theorem 19 to show that the copy is valid when the order is extensible, is also valid when source and target overlap. Thus, we obtain an ordered parse. Since we have at most g_{rl} leaves in the grammar tree, the ordered parse is of size at most g_{rl}, and therefore the optimal one is also of size $u \leq g_{rl}$. By Theorem 4, we also have $u = O(b \log(n/b))$. □

Finally, we show that greedy parsings can be computed much faster on extensible orders.

Theorem 21. *Any ordered parse, under any extensible order $\preceq$, can be computed greedily in $O(n)$ expected time or $O(n \log\log \sigma)$ worst-case time, and $O(n)$ space, given an array $O[1 \,..\, n]$ with the suffixes of T sorted by $\preceq$.*

Proof. We first compute the suffix array SA of T in $O(n)$ time (recall Section II-F), and from it, the *suffix tree* of T [58] can also be built in $O(n)$ time [31]. The suffix tree is a compact trie storing all the suffixes of T, so that we can descend from the root and, in time $O(m)$, find the interval $SA[sp\,..\,ep]$ of all the suffixes starting with a given string of length m.

We also create in $O(n)$ time the inverse permutation $IO[1\,..\,n]$ of $O[1\,..\,n]$, that is, $IO[p]$ is the rank of $T[p\,..]$ among all the other suffixes, in the order $\preceq$. With it, we build in $O(n)$ time a *range minimum query* data structure on the array $K[k] = IO[SA[k]]$, so that $RMQ(i,j) = \arg\min_{i\le k\le j} K[k]$ is computed in constant time [15]. Therefore, if $SA[sp\,..\,ep]$ is the suffix array interval of all the suffixes $T[p\,..]$ starting with a string S, then $RMQ(sp, ep)$ gives the suffix starting with S with the minimum rank in the order $\preceq$.

We now create the parse phrase by phrase. To produce the next phrase $T[p\,..]$, we enter the suffix tree from the root with the successive symbols $T[p+j]$, for $j \ge 0$. At each step, the suffix tree gives us the range $SA[sp_j, ep_j]$ of the suffixes of T starting with $T[p\,..\,p+j]$. We then find $K[RMQ(sp_j, ep_j)]$, which is the smallest rank of any occurrence of $T[p\,..\,p+j]$ in T. If this is less than $IO[p]$, then there is a smaller occurrence of $T[p\,..\,p+j]$ and we continue with the next value of j. The process stops when $K[RMQ(sp_j, ep_j)] = IO[p]$, that is, $T[p\,..\,p+j]$ is its smallest occurrence, so we cannot copy it from a smaller one. At this point, the new phrase is $T[p\,..\,p+j-1]$ if $j > 0$, or the explicit symbol $T[p]$ if $j = 0$.

Since we descend to a suffix tree child for every symbol of T, the total traversal time is $O(n)$ as well. There is a caveat, however. To provide constant-time traversal to children, the suffix tree must implement perfect hashing on the children of each node, which can be built in constant expected time per element. In this case, the whole parsing takes $O(n)$ expected time. Alternatively, each node can store its children with a predecessor data structure, so that each traversal to a child costs $O(\log\log\sigma)$ time, and the structure can be built in worst-case time $O(n\log\log\sigma)$ [3, Sec. A.1 & A.2], which dominates the total worst-case time of the parsing. The total space used is $O(n)$ in both variants. If array $O[1\,..\,n]$ is not given, we can compute it with a classical sorting algorithm in $O(n\log n)$ evaluations of $\prec$. □

VI. Lexicographic Parses

In this section we study a particularly interesting ordered parse we call lexicographic parse.

Definition 10. *A* lexicographic parse *of $T[1\,..\,n]$ is an ordered parse where $T[s_i\,..] \prec T[t_i\,..]$ iff the former suffix is smaller than the latter in lexicographic order, or which is the same, if $ISA[s_i] < ISA[t_i]$.*

We first note that the order we use is extensible.

Lemma 22. *The order $T[s\,..] \prec T[t\,..]$ iff the suffix $T[s\,..]$ lexicographically precedes $T[t\,..]$, is extensible.*

Proof. If $T[s\,..]$ lexicographically precedes $T[t\,..]$ and $T[s] = T[t]$, then by definition of lexicographic order, $T[s+1\,..]$ lexicographically precedes $T[t+1\,..]$. □

By Lemma 14, then, any lexicographic parse is a bidirectional scheme. One example of a lexicographic parse is the bidirectional scheme based on the BWT we introduced in Section IV.

Lemma 23. *The bidirectional scheme induced by the BWT in Def. 5 is a lexicographic parse of size $2r$.*

Proof. The definition uses the function $f(p) = \phi(p) = SA[ISA[p]-1]$ to copy from $T[\phi(t_i)\,..\,\phi(t_i)+\ell_i-1]$ to $T[t_i\,..\,t_i+\ell_i-1]$, where $\ell_i = t_{i+1} - t_i - 1$ (recall Theorem 9). Therefore it holds that $ISA[s_i] = ISA[\phi(t_i)] = ISA[t_i] - 1 < ISA[t_i]$. □

Another lexicographic parse is *lcpcomp* [12]. This algorithm uses a queue to find the largest entry in the LCP array (recall Section II-F). This information is then used to define a new phrase of the factorization. LCP entries covered by the phrase are then removed from the queue, LCP values affected by the creation of the new phrase are decremented, and the process is repeated until there are no text substrings that can be replaced with a pointer to lexicographically smaller positions. The output of lcpcomp is a series of source-length pairs interleaved with plain substrings (that cannot be replaced by pointers).

Lemma 24. *The* lcpcomp *factorization [12] is a lexicographic parse.*

Proof. The property can be easily seen from step 2 of the algorithm [12, Sec. 3.2]: the authors create a phrase (i.e., source-length pair) $(SA[i-1], LCP'[i])$ expanding to text substring $T[SA[i]\,..\,SA[i] + LCP'[i] - 1]$. We write $LCP'[i]$ because entries of the LCP array may be decremented in step 4, therefore $LCP'[i] \le LCP[i]$ at any step of the algorithm for any $1 \le i \le n$. This however preserves the two properties of lexicographic parsings: $T[SA[i]\,..\,SA[i] + LCP'[i] - 1] = T[SA[i-1]\,..\,SA[i-1] + LCP'[i] - 1]$ (phrases are equal to their sources) and, clearly, $i - 1 < i$ (sources are lexicographically smaller than phrases). □

Since the lexicographic order is extensible, we can find the optimal lexicographic parse greedily, in $O(n\log\log\sigma)$ time, by Theorem 21. We now show that, just as Lempel-Ziv, it can be found in $O(n)$ time, in a surprisingly simple way.

Definition 11. *The* lex-parse *of $T[1\,..\,n]$, with arrays SA, ISA, and LCP, is defined as a partition $T = L_1, \ldots, L_v$ such that $L_i = T[t_i\,..\,t_i + \ell_i - 1]$, satisfying (1) $t_1 = 1$ and $t_{i+1} = t_i + \ell_i$, and (2) $\ell_i = LCP[ISA[t_i]]$, with the exception that if $\ell_i = 0$*

we set $\ell_i = 1$ and make L_i an explicit symbol. The non-explicit phrases $T[t_i \,..\, t_i + \ell_i - 1]$ are copied from $T[s_i \,..\, s_i + \ell_i - 1]$, where $s_i = SA[ISA[t_i] - 1]$.

Example: The lex-parse of our example string is $a|\underline{l}|a|\underline{b}|a|\underline{r}|ala|labar|\underline{d}|\underline{a}|\underline{\$}$, where we underlined the explicit symbols. The corresponding function is $f[1\,..\,17] = \langle 11, 0, 16, 0, 7, 0, 9, 10, 11, 2, 3, 4, 5, 6, 0, 0, 0\rangle$.

Since ISA and LCP can be built in linear time, it is clear that the lex-parse of T can be built in $O(n)$ time. Let us show that it is indeed a valid lexicographic parse.

Lemma 25. *The lex-parse is a lexicographic parse, thus $b \leq v$.*

Proof. First, the parse covers T and it copies sources to targets with the same content: Let $x = ISA[t_i]$ and $y = ISA[t_i] - 1$. Then $\ell_i = LCP[x]$ is the length of the shared prefix between the suffixes starting at $t_i = SA[x]$ and $s_i = SA[y]$. Therefore we can copy $T[s_i \,..\, s_i + \ell_i - 1]$ to $T[t_i \,..\, t_i + \ell_i - 1]$. Second, the parse is lexicographic: $ISA[s_i] = ISA[t_i] - 1 < ISA[t_i]$. Since lexicographic parses are ordered parses, we have $b \leq v$ by Lemma 14. □

From now on we will use v as the size of the lex-parse. Let us show that the lex-parse is indeed ordered-optimal.

Theorem 26. *The lex-parse is the smallest lexicographic parse. Thus, $v \leq 2r$, $v \leq |lcpcomp|$, $v = O(b \log(n/b))$, $v \log_2 n \leq nH_k + o(n \log \sigma)$ for any $k = o(\log_\sigma n)$, $v = O(n/\log_\sigma n)$, and there are text families where $r = \Omega(v \log n)$.*

Proof. By Theorem 16, it suffices to show that Def. 11 defines a greedy parse under lexicographic ordering. Indeed, $\ell_i = LCP[ISA[t_i]]$ is the longest prefix shared between $T[t_i\,..]$ and any other suffix that is lexicographically smaller than it.

The other results are immediate consequences of Lemmas 23 and 24, Theorem 20, Lemma 12, and Theorem 19. □

Note that, unlike v, z can be $\Omega(r \log n)$, as shown in Theorem 11. Thus, v offers a better asymptotic bound with respect to the number of runs in the BWT. The following corollary is immediate.

Theorem 27. *There is an infinite family of strings over an alphabet of size 2 for which $z = \Omega(v \log n)$.*

We now show that the bound $v = O(b \log(n/b))$ is tight as a function of n.

Theorem 28. *There is an infinite family of strings over an alphabet of size 2 for which $v = \Omega(b \log n)$.*

Proof. We first prove that $b \leq 4$ for all Fibonacci words, and then that $v = \Omega(\log n)$ on the *odd* Fibonacci words (on the even ones it holds $v = O(1)$, by Theorem 11). The proof is rather technical, so we defer it to Appendix A. □

An interesting remaining question is whether v is always $O(z)$ or there is a string family where $z = o(v)$. While we have not been able to settle this question, we can exhibit a string family for which $z < \frac{3}{5}v$.

Lemma 29. *On the alphabet $\{1, \ldots, \sigma + 1\}$, where σ is not a multiple of 3, consider the string $S_1 = (2\,3 \ldots \sigma\,1)^3$. Then, for $i = 1, \ldots, \sigma - 1$, string S_{i+1} is formed by changing $S_i[3\sigma - 3i]$ to $\sigma + 1$. Our final text is then $T = S_1 \cdot S_2 \cdots S_\sigma$, of length $n = 3\sigma^2$. In this family, $z = 3\sigma - 2$ and $v = 5\sigma - 2$.*

Proof. In the Lempel-Ziv parse of T, we first have $\sigma + 1$ phrases of length 1 to cover the first third of S_1, and then a phrase that extends in T until the first edit of S_2. Since then, each edit forms two phrases: one covers the edit itself (since σ is not a multiple of 3, each edit is followed by a distinct symbol), and the other covers the range until the next edit. This adds up to $z = 3\sigma - 2$.

A lex-parse starts similarly, since the Lempel-Ziv phrases indeed point to lexicographically smaller ones. However, it needs 2σ further phrases to cover $S_\sigma = 2\,3\,(\sigma+1)\,5\,6\,(\sigma+1)\ldots$ with phrases of alternating length 2 and 1: each such pair of suffixes $S_\sigma[3i + 1\,..]$ and $S_\sigma[3i + 3\,..]$, for $i = 0, \ldots, \sigma - 1$, do appear in previous substrings S_j, but all these are lexicographically larger (because σ is not a multiple of 3, and thus symbols 1 are never replaced by $\sigma + 1$). Therefore, only length-2 strings of symbols not including $\sigma + 1$ can point to, say, S_1 (this reasoning has been verified computationally as well). This makes a total of $v = 5\sigma - 2$ phrases. □

A. Experimental Comparison with Lempel-Ziv

As a test on the practical relevance of the lex-parse, we measured v, z, and r on various synthetic, pseudo-real, and real repetitive collections obtained from PizzaChili (`http://pizzachili.dcc.uchile.cl`) and on four repetitive collections (`boost`, `bwa`, `samtools`, `sdsl`) obtained by concatenating the first versions of github repositories (`https://github.com`) until obtaining a length of $5 \cdot 10^8$ characters for each collection.

Table II shows the results. Our new lex-parse performs better than Lempel-Ziv on the synthetic texts, especially on the Fibonacci words (`fib41`), the family for which we know that $v = o(z)$ (recall Theorems 11 and 27).[7] On the others (Run-Rich String and Thue-Morse sequences), z is about 30% larger than v.

[7]The file `fib41` uses a variant where $F_1 = a$, $F_2 = ba$, and $F_k = F_{k-2}F_{k-1}$.

file	n	r	z	v	z/v	r/v
fib41	267,914,296	4	41	4	> 10	1.000
rs.13	216,747,218	77	52	40	1.300	1.925
tm29	268,435,456	82	56	43	1.302	1.907
dblp.xml.00001.base	104,857,600	172,489	59,573	59,821	0.996	2.883
dblp.xml.00001.prev	104,857,600	175,617	59,556	61,580	0.967	2.852
dblp.xml.0001.base	104,857,600	240,535	78,167	83,963	0.931	2.865
dblp.xml.0001.prev	104,857,600	270,205	78,158	100,605	0.777	2.686
sources.001.prev	104,857,600	1,213,428	294,994	466,643	0.632	2.600
dna.001.base	104,857,600	1,716,808	308,355	307,329	1.003	5.586
proteins.001.base	104,857,600	1,278,201	355,268	364,093	0.976	3.511
english.001.prev	104,857,600	1,449,519	335,815	489,034	0.687	2.964
boost	500,000,000	61,814	22,680	22,418	1.012	2.757
einstein.de	92,758,441	101,370	34,572	37,721	0.917	2.687
einstein.en	467,626,544	290,239	89,467	97,442	0.918	2.979
bwa	438,698,066	311,427	106,655	107,117	0.996	2.907
sdsl	500,000,000	345,325	113,591	112,832	1.007	3.061
samtools	500,000,000	458,965	150,988	150,322	1.004	3.053
world_leaders	46,968,181	573,487	175,740	179,696	0.978	3.191
influenza	154,808,555	3,022,822	769,286	768,623	1.001	3.933
kernel	257,961,616	2,791,368	793,915	794,058	1.000	3.515
cere	461,286,644	11,574,641	1,700,630	1,649,448	1.031	7.017
coreutils	205,281,778	4,684,460	1,446,468	1,439,918	1.005	3.253
escherichia_coli	112,689,515	15,044,487	2,078,512	2,014,012	1.032	7.470
para	429,265,758	15,636,740	2,332,657	2,238,362	1.042	6.986

TABLE II
VARIOUS REPETITIVENESS MEASURES OBTAINED FROM SYNTHETIC, PSEUDO-REAL, AND REAL TEXTS (EACH CATEGORY FORMS A BLOCK IN THE TABLE).

Pseudo-real texts are formed by taking a real text and replicating it many times; a few random edits are then applied to the copies. The fraction of edits is indicated after the file name, for example, `sources.001` indicates a probability of 0.001 of applying an edit at each position. In the names with suffix `.base`, the edits are applied to the base version to form the copy, whereas in those with suffix `.prev`, the edits are cumulatively applied to the previous copy. It is interesting to note that, in this family, v and z are very close under the model of edits applied to the base copy, but z is generally significantly smaller when the edits are cumulative. The ratios actually approach the fraction $\frac{3}{5} = 0.6$ we obtained in Lemma 29 using a particular text that, incidentally, follows the model of cumulative edits.

On real texts, both measures are very close. Still, it can be seen that in collections like `einstein.de` and `einstein.en`, which feature cumulative edits (those collections are formed by versions of the Wikipedia page on Einstein in German and English, respectively), z is about 8% smaller than v. On the other hand, v is about 3%–4% smaller than z on biological datasets such as `cere`, `escherichia_coli` and `para`, where the model is closer to random edits applied to a base text. The lex-parse is also about 1% smaller than the Lempel-Ziv parse on github versioned collections, except `bwa`.

To conclude, the comparison between r and v shows that the sub-optimal lexicographic parse induced by the Burrows-Wheeler transform is often much larger (typically 2.5–4.0 times, but more than 7 times on the biological datasets) than the optimal lex-parse. Interestingly, on Fibonacci words the optimal parse is already found by the Burrows-Wheeler transform.

VII. BOUNDS ON COLLAGE SYSTEMS

In this section we use our previous findings to prove that $c = O(z)$, $b = O(c)$, and that there exist string families where $c = \Omega(b \log n)$, where c is the size of the smallest (internal) collage system.

Theorem 30. *There is always an internal collage system of $c \leq 4z$ rules generating T.*

Proof. We proceed by induction on the Lempel-Ziv parse. At step i, we obtain a collage system with initial symbol S_i that generates the prefix $T[1\,..\,p_i]$ of T covered by the first i phrases. The initial symbol for the whole T is then S_z.

For the first phrase, which must be an explicit symbol a, we insert the rule $S_1 \to a$. Let us now consider the phrases $i > 1$. If the ith phrase is an explicit symbol a, then we add rules $A_i \to a$ and $S_i \to S_{i-1}A_i$.

Otherwise, let the ith phrase point to a source that is completely inside $T[1\,..\,p_{i-1}]$, precisely $T[x\,..\,y]$ with $y \leq p_{i-1}$. Then we add rule $N_i \to S_{i-1}^{[x,y]}$, and then $S_i \to S_{i-1}N_i$.

If, instead, the ith phrase points to a source that overlaps it, $T[x\,..\,y]$ with $p_{i-1} < y < p_i$, then $T[x\,..\,y]$ is periodic with period $p = p_{i-1} - x + 1$, that is, $T[x\,..\,y-p] = T[x+p\,..\,y]$. Therefore, the new phrase is formed by $q = \lfloor \frac{y-x+1}{p} \rfloor$ copies of $T[x\,..\,x+p-1] = T[x\,..\,p_{i-1}]$ plus $T[x\,..\,x+((y-x+1) \bmod p)-1]$ if p does not divide $y-x+1$. This can be obtained with $O_i \to^{[p]} S_{i-1}$, $O'_i \to O_i^{[(y-x+1) \bmod p]}$, $R_i \to O_i^q$, $N_i \to R_iO'_i$, and $S_i \to S_{i-1}N_i$.

Figure 5 illustrates both cases schematically. It is easy to see that the collage system is internal: we apply truncation only on the symbols S_i and O_i, all of which are reachable from the initial symbol. □

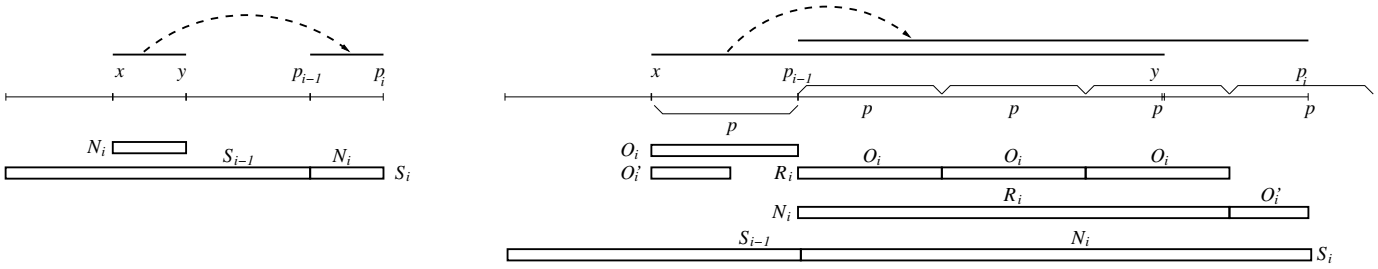

Fig. 5. Conversion of a Lempel-Ziv parse into a collage system using Theorem 30. On the left, the nonoverlapping case. On the right, the overlapping case.

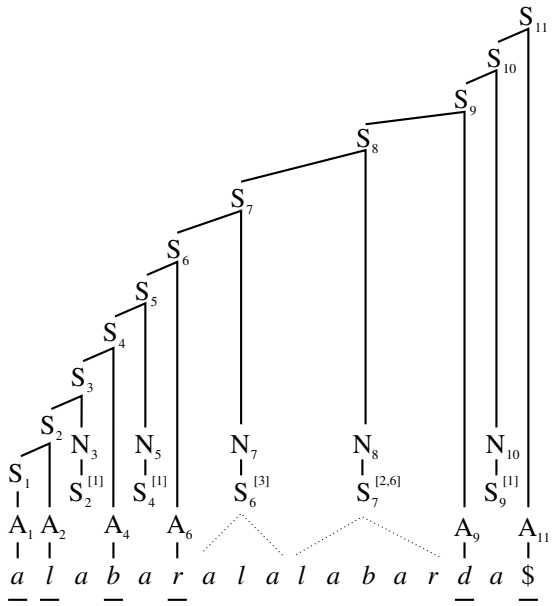

Fig. 6. Creation of an internal collage system from the Lempel-Ziv parse of $T = \underline{a}|\underline{l}|a|\underline{b}|a|\underline{r}|ala|labar|\underline{d}|a|\underline{\$}$, using Theorem 30.

Example: Consider the Lempel-Ziv parse $T = \underline{a}|\underline{l}|a|\underline{b}|a|\underline{r}|ala|labar|\underline{d}|a|\underline{\$}$ of Section II-C, where we have underlined the explicit symbols. Figure 6 illustrates the application of Theorem 30 to this parse.

Theorem 31. *For any T with an internal collage system of size c there is a bidirectional scheme of size $b \leq c$.*

Proof. We extend the idea of Theorem 5 to handle substring rules. We draw the parse tree of T, starting from the initial symbol. When we reach a nonterminal defined by a substring rule, we convert it into a leaf. Just as for grammar trees, we also convert into leaves all but the leftmost occurrence of each other nonterminal in the parse tree. Analogously to grammar trees, the resulting tree has at most c leaves, because we are just adding substring rules, each of which adds a new leaf.

We now generate a bidirectional macro scheme exactly as we defined the left-to-right parse in Theorem 5. Further, each leaf representing a substring rule $A \to B^{[t,t']}$ is converted into a single phrase pointing to $T[x+t-1\,..\,x+t'-1]$, where the leftmost occurrence of B in the parse tree covers the text $T[x\,..\,y]$. Such occurrence always exists in internal collage systems.

The resulting parse may not be left-to-right anymore. However, it is a valid bidirectional scheme. To see this, let us label each position p in T with the index in the sequence of rules of the leaf of the grammar tree covering $T[p]$. This means that the labels of text positions descending from an internal node A are smaller than the index of A, because nonterminals are defined in terms of earlier nonterminals. Any position p of T below a grammar leaf A or $A^{[t,t']}$ is labeled with the index of A, and $f(p)$ has a smaller label: that of a grammar leaf descending from the internal node A. □

Example: The following collage system to generate the text $T = alabaralalabarda\$$ is an internal variant of the one given in Section II-E: $A \to a$, $B \to b$, $D \to d$, $L \to l$, $R \to r$, $Z \to \$$, $C \to AL$, $E \to CC$, $F \to BA$, $G \to FR$, $H \to DA$, $I \to HZ$, $J \to EA$, $K \to JG$, $M \to {}^{[6]}K$, $N \to MK$, $O \to NI$. The corresponding bidirectional scheme induces the parse $T = alabar|\underline{a}|\underline{l}|al|a|\underline{b}|a|\underline{r}|\underline{d}|a|\underline{\$}$, with function $f[1\,..\,17] = \langle 9, 10, 11, 12, 13, 14, 0, 0, 7, 8, 7, 0, 7, 0, 0, 7, 0\rangle$.

Theorem 32. *There exists an infinite family of strings over an alphabet of size 2 for which $c = \Omega(r \log n)$, and thus also $c = \Omega(b \log n)$, for any general collage system of size c.*

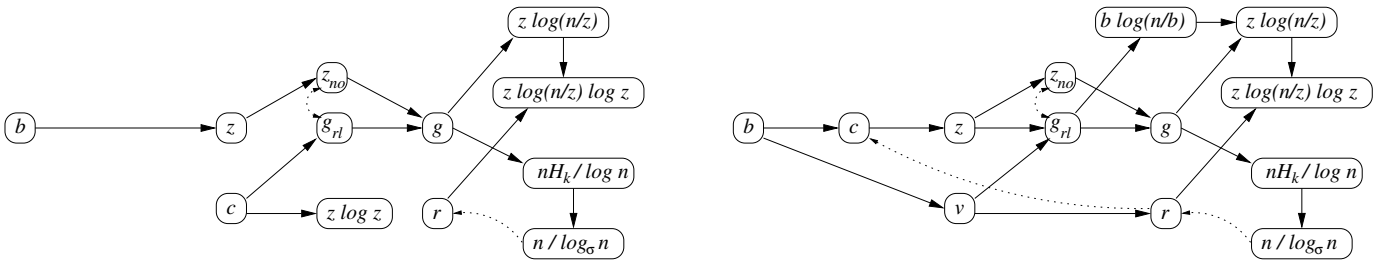

Fig. 7. Previously known (left) and new (right) asymptotic bounds between repetitiveness measures. A solid arrow from x to y means that $x = O(y)$ for every string family. The arrow $b \to c$ holds for internal collage systems only. For most arrows, a logarithmic gap for some string family is known, except $c \to z$. There are also logarithmic gaps for some incomparable measures, shown in dotted lines (one is less than logarithmic, $g_{rl} = \Omega(z_{no} \log n / \log\log n)$).

Proof. Fibonacci words do not contain 4 consecutive repetitions of the same substring [29]. Therefore, no internal collage system generating a Fibonacci word contains run-length rules $A \to B^k$ with $k > 3$, because $exp(A)$ does appear in T. Run-length rules with $k \leq 3$ can be replaced by one or two non-run-length rules. Therefore, if a Fibonacci word of length n is generated by an internal collage system of size c, then it is also generated by an internal collage system of size at most $2c$ with no run-length rules.

Just as with SLPs, no such collage system can generate a string of length more than 2^{2c}; the substring rules do not help in obtaining long strings with fewer rules. As a consequence, it holds that $c = \Omega(\log n)$. On the other hand, by Theorem 11, it holds that even Fibonacci words have $r = O(1)$ (and also $v, b = O(1)$).

We can extend the result to general collage systems by noting that every nonterminal $A \to B^k$ with $k > 4$ must be shortened via truncation by more than $|exp(B)|$ symbols, so as to use it to form T. Thus, it can be replaced by $A \to B^4$, and then be further replaced by two non-run-length rules. □

VIII. Conclusions

We have essentially closed the question of what the approximation ratio of the (unidirectional, left-to-right) Lempel-Ziv parse is with respect to the optimal bidirectional parse, therefore contributing to the understanding of the quality of this popular heuristic that can be computed in linear time, while computing the optimal bidirectional parse is NP-complete. Our bounds, which are shown to be tight, imply that the gap is in fact logarithmic, wider than what was previously known.

We have then generalized Lempel-Ziv to the class of optimal ordered parsings, where there must be an increasing relation between source and target positions in a copy. We proved that some features of Lempel-Ziv, such as converging to the empirical entropy, being limited by the smallest RLSLP, and being larger than the optimal bidirectional scheme by at most a logarithmic factor, hold in fact for all optimal ordered parsings.

As an example of such a parse, we introduced the lex-parse, which is the optimal left-to-right parse in the lexicographical order of the involved suffixes. This new parse is shown to be computable greedily in linear time and to have many of the good bounds of the Lempel-Ziv parse with respect to other measures, even improving on some. For example, being an optimal ordered parse, the lex-parse is upper-bounded by the smallest RLCFG and it is an approximation to the smallest bidirectional parse with a logarithmic gap. In addition, the lex-parse is bounded by the number of runs in the BWT of the text, which is not the case of the Lempel-Ziv parse. We exhibit a family of strings where the lex-parse is asymptotically smaller than the Lempel-Ziv parse, and another where the latter is smaller than the lex-parse, though only by a constant factor. Experimentally, the lex-parse is shown to behave similarly to the Lempel-Ziv parse, although it is somewhat larger on versioned collections with cumulative edits.

Finally, we showed that the smallest collage systems are of the order of the Lempel-Ziv parse, and have a logarithmic gap with the number of BWT runs on some string families. A restricted variant we call internal collage systems are shown to asymptotically bound the smallest bidirectional scheme. Many other results are proved along the way.

Figure 7 illustrates the contributions of this article to the knowledge of the asymptotic bounds between repetitiveness measures. Note that the solid arrow relations are transitive, because they hold for every string family. Dotted arrows, instead, are not transitive because they hold for specific string families.

There are various interesting avenues of future work. For example, it is unknown if there are string families where $z = o(v)$ or $c = o(z)$, nor if $b = O(c)$ holds for general collage systems. We can prove the latter if it holds that b grows only by a constant factor when we remove a prefix of T, but this is an open question. We can even prove $z = O(c)$ for general collage systems if it holds that there is only a constant gap between z for T and for its reverse, which is another open question. It might also be that our Theorem 4 can be proved without using run-length rules, then yielding $g = O(b \log(n/b))$.

Another interesting line of work is that of optimal ordered parses, which can be built efficiently and compete with z, which has been the gold-standard approximation for decades. Are there other convenient parses apart from our lex-parse? In particular, are there parses that can compete with z while offering efficient random access time to T? Right now, only parses of size

$O(g)$ (and $O(g_{rl})$ [9]) allow for efficient ($O(\log n)$ time) access to T; all the other measures need a logarithmic blowup in space to support efficient access [2], [6], [4], [53], [18], [19], [20]. This is also crucial to build small and efficient compressed indexes on T [46, Sec. 13.2].

ACKNOWLEDGEMENTS

This work was partially funded by Basal Funds FB0001, Conicyt, by Fondecyt Grant 1-200038, Chile, by the Millennium Science Initiative Program - Code ICN17_002, by the Danish Research Council Fund DFF-4005-00267, and by the project MIUR-SIR CMACBioSeq, grant n. RBSI146R5L.

REFERENCES

[1] D. Belazzougui, F. Cunial, T. Gagie, N. Prezza, and M. Raffinot. Composite repetition-aware data structures. In *Proc. 26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 26–39, 2015.
[2] D. Belazzougui, T. Gagie, P. Gawrychowski, J. Kärkkäinen, A. Ordóñez, S. J. Puglisi, and Y. Tabei. Queries on LZ-bounded encodings. In *Proc. 25th Data Compression Conference (DCC)*, pages 83–92, 2015.
[3] D. Belazzougui and G. Navarro. Optimal lower and upper bounds for representing sequences. *ACM Transactions on Algorithms*, 11(4):article 31, 2015.
[4] D. Belazzougui, S. J. Puglisi, and Y. Tabei. Access, rank, select in grammar-compressed strings. In *Proc. 23rd Annual European Symposium on Algorithms (ESA)*, LNCS 9294, pages 142–154, 2015.
[5] P. Bille, T. Gagie, I. Li Gørtz, and N. Prezza. A separation between RLSLPs and LZ77. *Journal of Discrete Algorithms*, 50:36–39, 2018.
[6] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. S. Rao, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015.
[7] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
[8] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
[9] A. R. Christiansen, M. B. Ettienne, T. Kociumaka, G. Navarro, and N. Prezza. Optimal-time dictionary-compressed indexes. *CoRR*, 1811.12779v3, 2019. To appear in *ACM Transactions on Algorithms*.
[10] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.
[11] M. Crochemore, C. S. Iliopoulos, M. Kubica, W. Rytter, and T. Waleń. Efficient algorithms for three variants of the LPF table. *Journal of Discrete Algorithms*, 11:51–61, 2012.
[12] P. Dinklage, J. Fischer, D. Köppl, M. Löbel, and K. Sadakane. Compression with the tudocomp framework. In *Proc. 16th International Symposium on Experimental Algorithms, (SEA)*, 2017.
[13] X. Droubay. Palindromes in the Fibonacci word. *Information Processing Letters*, 55(4):217 – 221, 1995.
[14] G. Fici. Factorizations of the Fibonacci infinite word. *Journal of Integer Sequences*, 18(9):article 3, 2015.
[15] J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.
[16] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, pages 734–740, 2011.
[17] T. Gagie. Large alphabets and incompressibility. *Information Processing Letters*, 99(6):246–251, 2006.
[18] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. A faster grammar-based self-index. In *Proc. 6th International Conference on Language and Automata Theory and Applications (LATA)*, pages 240–251, 2012.
[19] T. Gagie, P Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 731–742, 2014.
[20] T. Gagie, G. Navarro, and N. Prezza. Fully-functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1):article 2, 2020.
[21] J. K. Gallant. *String Compression Algorithms*. PhD thesis, Princeton University, 1982.
[22] P. Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic. In *Proc. 19th Annual European Symposium on Algorithms (ESA)*, pages 421–432, 2011.
[23] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman, 2nd edition, 1994.
[24] H. Helfgott and M. Cohn. On maximal parsings. In *Proc. 7th Data Compression Conference (DCC)*, pages 291–299, 1997.
[25] D. Hucke, M. Lohrey, and C. P. Reh. The smallest grammar problem revisited. In *Proc. 23rd International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS 9954, pages 35–49, 2016.
[26] T. I. Longest common extensions with recompression. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LIPIcs 78, pages 18:1–18:15, 2017.
[27] A. Jeż. Approximation of grammar-based compression via recompression. *Theoretical Computer Science*, 592:115–134, 2015.
[28] A. Jeż. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016.
[29] J. Kahrumäki. On cube-free ω-words generated by binary morphisms. *Discrete Applied Mathematics*, 5:279–297, 1983.
[30] J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.
[31] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 181–192, 2001.
[32] D. Kempa and T. Kociumaka. Resolution of the Burrows-Wheeler Transform conjecture. *CoRR*, 1910.10631, 2019. To appear in *FOCS 2020*.
[33] D. Kempa and N. Prezza. At the roots of dictionary compression: String attractors. In *Proc. 50th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 827–840, 2018.
[34] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003.
[35] J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.
[36] D. K. Kim, J. S. Sim, H. Park, and K. Park. Constructing suffix arrays in linear time. *Journal of Discrete Algorithms*, 3(2-4):126–142, 2005.
[37] P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, 3(2-4):143–156, 2005.
[38] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems on Information Transmission*, 1(1):1–7, 1965.
[39] R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 2000.
[40] S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013.
[41] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.
[42] V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005.

[43] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
[44] S. Mantaci, A. Restivo, and M. Sciortino. Burrows-Wheeler transform and Sturmian words. *Information Processing Letters*, 86(5):241–246, 2003.
[45] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.
[46] G. Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.
[47] T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda. Fully dynamic data structure for LCE queries in compressed space. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 72:1–72:15, 2016.
[48] C. Ochoa and G. Navarro. Repair and all irreducible grammars are upper bounded by high-order empirical entropy. *IEEE Transactions on Information Theory*, 65(5):3160–3164, 2019.
[49] G. Pirillo. Fibonacci numbers and words. *Discrete Mathematics*, 173(1):197 – 207, 1997.
[50] N. Prezza. *Compressed Computation for Text Indexing*. PhD thesis, University of Udine, 2016.
[51] M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *Journal of the ACM*, 28(1):16–24, 1981.
[52] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1-3):211–222, 2003.
[53] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1230–1239, 2006.
[54] H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms*, 3(2–4):416–430, 2005.
[55] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:398–403, 1948.
[56] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, Z. Chenxiang, M. J. Efron, R. Iyer, M. C. Shatz, S. Sinha, and G. E. Robinson. Big data: Astronomical or genomical? *PLoS Biology*, 17(7):e1002195, 2015.
[57] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.
[58] P. Weiner. Linear Pattern Matching Algorithms. In *Proc. 14th IEEE Symp. on Switching and Automata Theory (FOCS)*, pages 1–11, 1973.
[59] J. Ziv and A. Lempel. Compression of individual sequences via variable length coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

APPENDIX A
A SEPARATION BETWEEN b AND v

In this section we prove that $b \leq 4$ for all Fibonacci words, and then that $v = \Omega(\log n)$ on the odd Fibonacci words. We first state a couple of results on Fibonacci words F_k.

Lemma 33. *For each $k \geq 5$, it holds that $F_{k-1}F_{k-2} = H_k ba$ and $F_{k-2}F_{k-1} = H_k ab$ if k is even, and $F_{k-1}F_{k-2} = H_k ab$ and $F_{k-2}F_{k-1} = H_k ba$ if k is odd. Note that $|H_k| = f_k - 2$.*

Proof. It is easy to see by induction that $F_k = F_{k-1}F_{k-2}$ finishes with ab if k is odd and with ba if k is even. The fact that $F_{k-1}F_{k-2} = H_k xy$ and $F_{k-2}F_{k-1} = H_k yx$ was proved by Pirillo [49, Lem. 1]. □

Lemma 34. *F_{k-1} only appears at position 1 in F_k.*

Proof. Consider the following derivation (which is also used later), obtained by applying Def. 1 several times:

$$\begin{aligned} F_k &= F_{k-1}F_{k-2} \\ &= F_{k-2}F_{k-3}F_{k-2} && (1) \\ &= F_{k-2}F_{k-3}F_{k-3}F_{k-4} && (2) \\ &= F_{k-2}F_{k-3}F_{k-4}F_{k-5}F_{k-4} \\ &= F_{k-2}F_{k-2}F_{k-5}F_{k-4}. && (3) \end{aligned}$$

Assume, by contradiction, that F_{k-1} appears in two different positions inside F_k. From Eq. (3), we have that $F_n = F_{k-2}F_{k-2}F_{k-5}F_{k-4}$. Also, no occurrence of F_{k-1} can start after position f_{k-2} in F_k (because it would exceed F_k unless it starts at $p = f_{k-2} + 1$, but this is also outruled because $F_k = F_{k-1}F_{k-2} \neq F_{k-2}F_{k-1}$ by Lemma 33). Thus, the second occurrence of F_{k-1} must start at a position $p \leq f_{k-2}$. Then, by Eq. (3) again, there is a third occurrence of F_{k-2} within $F_{k-2}F_{k-2}$, which means that F_{k-2} appears twice in the circular rotations of F_{k-2}. Yet, this is a contradiction because all the circular rotations on the Fibonacci words are different [13, Cor. 3.2]. □

Lemma 35. *Every word F_k has a bidirectional scheme of size $b \leq 4$.*

Proof. Up to $k = 4$ we have $|F_k| \leq 3$, so the claim is trivial. For $F_5 = abaab$ we can copy the last ab from the first to have $b = 4$. For $k \geq 6$, consider the following partition of $F_k = F_{k-1}F_{k-2}$ into 4 chunks:

1) The first chunk is $B_1 = F_k[1\,..\,f_{k-1} - 2]$ (i.e., all the symbols of F_{k-1} except the last two).
2) The second and third chunks are explicit symbols ($B_2 = F_k[f_{k-1} - 1] = b$ and $B_3 = F_k[f_{k-1}] = a$, if k is even, and $B_2 = F_k[f_{k-1} - 1] = a$ and $B_3 = F_k[f_{k-1}] = b$, if k is odd).
3) The fourth chunk is $B_4 = F_k[f_{k-1} + 1\,..\,f_k]$ (i.e., all the symbols of F_{k-2}).

The source of the first chunk, B_1, is $F_k[f_{k-2}+1\,..\,f_k-2]$, and the source of the fourth chunk, B_4, is $F_k[f_{k-2}+1\,..\,2f_{k-2}]$. Note that the sources of B_1 and B_4 start at the same position. We now prove that this is a valid bidirectional scheme.

First, we prove that B_1 and B_4 are equal to their sources. By Eq. (3), $F_k = F_{k-2}F_{k-2}F_{k-5}F_{k-4}$, so there is an occurrence of F_{k-2} starting at position $f_{k-2} + 1$ of F_k. Hence, $B_4 = F_k[f_{k-2} + 1\,..\,2f_{k-2}]$. Further, by Eq. (1), we have that $F_k = F_{k-2}F_{k-3}F_{k-2}$, and from Lemma 33 we have that $B_1 = H_{k-1} = F_k[f_{k-2} + 1\,..\,f_k - 2]$.

Thus, the sources of B_1 and B_4 are correctly defined. We now prove there are no cycles. Our bidirectional scheme defines the function $f : [1\,..\,f_k] \to [1\,..\,f_k] \cup \{0\}$ as follows:

$$f(p) = \begin{cases} 0, & \text{if } p = f_{k-1} - 1 \text{ or } p = f_{k-1} \\ p + f_{k-2}, & \text{if } p < f_{k-1} - 1 \\ p - f_{k-3}, & \text{if } p > f_{k-1} \end{cases}$$

Assume that f has cycles and that a shortest one starts at position p. Successive applications of f either increase the current position by f_{k-2} or decrease the current position by f_{k-3}. So, a cycle starting at position p means that $p+xf_{k-2}-yf_{k-3} = p$, where $x + y$ is the number of times f was applied; note $x, y > 0$ This is equivalent to $xf_{k-2} = yf_{k-3}$. Since f_{k-2} and f_{k-3} are coprime[8], f_{k-3} divides x and f_{k-2} divides y. Thus, $x \geq f_{k-3}$, $y \geq f_{k-2}$, and $x + y \geq f_{k-1}$. The number of positions involved in a cycle is then at least f_{k-1}, and they must all be different because the cycle is minimal. Yet, the first f_{k-2} positions of F_k cannot be involved in any cycle: once f is applied in one of the first f_{k-2} positions there is not way to get back there. So, we are left with $f_{k-1} - 2$ positions to be involved in a cycle, because $f(f_{k-1} - 1) = f(f_{k-1}) = 0$. That is a contradiction. □

Before delving into the proof of the lower bound that relates v and b, we prove two further properties of the Fibonacci words we make use of.

Lemma 36. *The strings bb, aaa, and $ababab$ never occur within a Fibonacci word.*

Proof. It is easy to see that all F_k, for $k \geq 3$, start with ab. Further, by Lemma 33, they end with ab or ba. Then the lemma for bb and aaa easily follows by induction because, when concatenating $F_k = F_{k-1}F_{k-2}$, the new substrings of length 3 we create are substrings of $abab$ or $baab$. For the third string we easily see that, for $k \geq 5$, every F_k starts with $abaa$ and ends with $baab$ (odd k) or $baba$ (even k). Thus, as before, it is impossible to form $ababab$ when concatenating any F_{k-1} with F_{k-2}. □

Lemma 37. *Given a Fibonacci word F_k, for all $4 \leq i \leq k$, every factor W_i of F_k of length f_i that begins with F_{i-1} has only two possible forms, $W_i = F_{i-1}F_{i-2}$ or $W_i = F_{i-2}F_{i-1}$.*

Proof. We use strong induction on i. For the base cases $i = 4$ and $i = 5$, we use the substrings bb and aaa excluded by Lemma 36: If $i = 4$, then $f_4 = 3$, and $F_3 = ab$. Then, any factor W_4 of F_k of length 3 that begins with ab can only be $W_4 = aba = F_3F_2$. If $i = 5$, then $f_5 = 5$, and $F_4 = aba$. Then, any factor W_5 of F_k of length 5 that begins with aba can only be equal to $W_5 = abaab = F_4F_3$ or $W_5 = ababa = F_3F_4$.

Assume now by induction that, for all $i \geq 4$, every factor W_i of F_k of length f_i that begins with F_{i-1} has only two possible forms, $W_i = F_{i-1}F_{i-2}$ or $W_i = F_{i-2}F_{i-1}$. We now prove that every factor W_{i+1} of F_k, of length f_{i+1} and beginning with F_i, has only two possible forms, $W_{i+1} = F_iF_{i-1}$ or $W_{i+1} = F_{i-1}F_i$.

The factor W_{i+1} is equal to F_iG_{i-1}, where G_x will stand for any string of length f_x. Thus, $W_{i+1} = F_{i-1}F_{i-2}G_{i-1}$. Since $|F_{i-2}G_{i-1}| = f_i > f_{i-1}$, we can apply the induction hypothesis to the first f_{i-1} symbols of this substring. Two outcomes are then possible: (i) $W_{i+1} = F_{i-1}F_{i-2}F_{i-3}G_{i-2}$ or (ii) $W_{i+1} = F_{i-1}F_{i-3}F_{i-2}G_{i-2}$.

Case (i) implies $W_{i+1} = F_{i-1}F_{i-1}G_{i-2}$. By the induction hypothesis, $F_{i-1}G_{i-2} = F_{i-1}F_{i-2}$ or $F_{i-1}G_{i-2} = F_{i-2}F_{i-1}$. This implies $W_{i+1} = F_{i-1}F_i$ or $W_{i+1} = F_iF_{i-1}$. Thus, W_{i+1} has the desired form.

In case (ii), the suffix $F_{i-2}G_{i-2}$ of W_{i+1} has length over f_{i-1} and starts with F_{i-2}, so we can apply the induction hypothesis to obtain subcases (a) $W_{i+1} = F_{i-1}F_{i-3}F_{i-2}F_{i-3}G_{i-4}$ or (b) $W_{i+1} = F_{i-1}F_{i-3}F_{i-3}F_{i-2}G_{i-4}$. We now show that neither subcase is possible. In case (a), by Def. 1, it holds that

$$\begin{aligned} W_{i+1} &= F_{i-1}F_{i-3}F_{i-2}F_{i-3}G_{i-4} \\ &= F_{i-2}F_{i-3}F_{i-3}F_{i-2}F_{i-3}G_{i-4} \\ &= F_{i-2}F_{i-3}F_{i-3}F_{i-3}F_{i-4}F_{i-3}G_{i-4}. \end{aligned}$$

If $i+1 = 6$ or 7, then $F_{i-3} = a$ or ab, and there would be 3 consecutive occurrences of a or ab in F_k, contradicting Lemma 36. If $i + 1 \geq 8$, then by Lemma 33, $F_{i-4}F_{i-3}$ begins with F_{i-3}, and then there would be 4 consecutive occurrences of F_{i-3} within F_k, contradicting the fact that Fibonacci words do not contain 4 consecutive repetitions of the same substring [29]. In case (b), by Def. 1, it holds that

$$\begin{aligned} W_{i+1} &= F_{i-1}F_{i-3}F_{i-3}F_{i-2}G_{i-4} \\ &= F_{i-2}F_{i-3}F_{i-3}F_{i-3}F_{i-2}G_{i-4} \\ &= F_{i-2}F_{i-3}F_{i-3}F_{i-3}F_{i-3}F_{i-4}G_{i-4}, \end{aligned}$$

which also contains 4 occurrences of F_{i-3} within F_k, a contradiction again [29]. □

[8]Applying Euclid's algorithm, we have $\gcd(f_{k-2}, f_{k-3}) = \gcd(f_{k-3}, f_{k-2} - f_{k-3}) = \gcd(f_{k-3}, f_{k-4})$, which is traced down to $\gcd(f_2, f_1) = 1$.

Theorem 28. *There is an infinity family of strings over an alphabet of size 2 for which $v = \Omega(b \log n)$.*

Proof. Such a family is formed by the *odd* Fibonacci words, where $b = O(1)$ by Lemma 35. Specifically, we prove that the number of phrases in the lex-parse of the odd Fibonacci words forms an arithmetic progression with step 1.

Let F_k be an odd Fibonacci word with $k \geq 9$. We first prove that the length $\ell_1 = LCP[ISA[1]]$ (see Def. 11) of the first phrase of the lex-parse of F_k is $f_{k-1} - 2$. From Eq. (1), we have that $F_k = F_{k-2}F_{k-3}F_{k-2}$, and from Lemma 33, we have that $F_k = H_{k-1}baF_{k-2} = F_{k-2}H_{k-1}ab$. Additionally, $H_{k-1}ab$ is lexicographically smaller than $H_{k-1}ba$ and they have a common prefix of length $f_{k-1} - 2$. Thus, $\ell_1 \geq f_{k-1} - 2$. We prove that there are no common prefixes of length greater than $f_{k-1} - 2$ between F_k and any of its suffixes. Assume the prefix P_{k-1} of length $f_{k-1} - 1$ of F_{k-1} appears in F_k. By the proof of Lemma 36, F_k finishes with $baab$ and F_{k-1} finishes with $baba$. Then P_{k-1} finishes with bab and F_k finishes with aab, so P_{k-1} is not a suffix of F_k. Also, b can only be followed by a within F_k, by Lemma 36. Hence, if there is an occurrence of P_{k-1} within F_k, then there is also an occurrence of F_{k-1}. Yet, the only occurrence of F_{k-1} in F_k is at the beginning, by Lemma 34. Therefore, it is also impossible to find an occurrence of length f_{k-1} or more.

Next, we prove that the length $\ell_2 = LCP[ISA[f_{k-1} - 1]]$ of the second phrase of the lex-parse of F_k is $f_{k-4} + 2$. By Eq. (1), we have that $F_{k-2} = F_{k-4}F_{k-5}F_{k-4}$. Since F_{k-5} finishes with ba, baF_{k-4} is a prefix and a suffix of baF_{k-2}. Since the suffix is followed by \$, it is lexicographically smaller than the prefix. Further, since the second phrase starts with the prefix baF_{k-4}, we have $\ell_2 \geq f_{k-4} + 2$. We now show that the second phrase is not longer.

By the characterization of the Fibonacci words of Mantaci et al. [44, Thm. 6], and the ordering of the cyclic rotations of the Fibonacci words stated in there [44, proof of Thm. 9], the lexicographically smallest cyclic rotation of F_k is the one that starts at position $x + 1$, where $x < f_k$ is the unique solution to the congruence equation $f_{k-2} - 1 + xf_{k-2} \equiv 0 \pmod{f_k}$[9]. Using Cassini's identity, $f_k f_{k-2} - f_{k-1}^2 = 1$ [23], we replace $f_k = f_{k-1} + f_{k-2}$ to get $f_{k-1}f_{k-2} + f_{k-2}^2 - f_{k-1}^2 = f_{k-1}f_{k-2} + (f_{k-2} + f_{k-1})(f_{k-2} - f_{k-1}) = f_{k-1}f_{k-2} + f_k(f_{k-2} - f_{k-1}) = 1$. This implies $f_{k-1}f_{k-2} \equiv 1 \pmod{f_k}$. Thus, x is equal to $f_{k-1} - 1$, and the the lexicographically smallest cyclic rotation of F_k starts at position f_{k-1}.

This means that the second phrase of the lex-parse of F_k starts one position before the lexicographically smallest cyclic rotation of F_k. So, now considering the terminator \$, if a suffix S of F_k is lexicographically smaller than $F_k[f_{k-1} - 1\,..] = baF_{k-2}$ (i.e., the suffix that starts at the beginning of the second phrase of the lex-parse of F_k) and both share a common prefix P, then $S = P$ and $|S| < f_{k-2} + 2$. Let us prove that baF_{k-4} is the largest string that is a prefix and a suffix of baF_{k-2}.

The string F_{k-4} only occurs at positions $1, f_{k-4} + 1$, and $f_{k-3} + 1$ within F_{k-2}: By Eq. (3), we have that $F_{k-2} = F_{k-4}F_{k-4}F_{k-7}F_{k-6}$. There are no occurrences of F_{k-4} at positions $1 < p \leq f_{k-4}$, by the same argument of Lemma 34. By Eq. (2), we also have that $F_{k-2} = F_{k-4}F_{k-5}F_{k-5}F_{k-6}$. There are no occurrences of F_{k-4} at positions $f_{k-4} + 1 < p \leq f_{k-3}$, because $F_{k-4} = F_{k-5}F_{k-6}$ and then F_5 would occur more than twice within F_5F_5, which is not possible again by the argument of Lemma 34. The last occurrence of F_{k-4} within $F_{k-2} = F_{k-3}F_{k-4}$ must then be at position $f_{k-3} + 1$. By Lemma 33, the only one of those three occurrences that is preceded by ba is the last one.

So the first two phrases of the lex-parse of F_k are of lengths $\ell_1 = f_{k-1} - 2$, and $\ell_2 = f_{k-4} + 2$, respectively. The rest R_k of F_k is then of length f_{k-3}. From Eq. (3), we have that $F_k = F_{k-2}F_{k-2}F_{k-5}F_{k-4}$, so $R_k = F_{k-5}F_{k-4} = H_{k-3}ab$, by Lemma 33. Since R_k starts with H_{k-3}, which starts with F_{k-4} by Lemma 33, and it finishes with F_{k-4}, which is the lexicographically smallest occurrence of F_{k-4}, we have $\ell_3 \geq f_{k-4}$.

By Lemma 37, we have that all the suffixes of F_k that start at position $1 \leq p \leq 2f_{n-2}$, and begin with F_{k-4}, also begin with $F_{k-4}F_{k-5} = H_{k-3}ba > R_k$, by Lemma 33, or with $F_{k-5}F_{k-4} = R_k$. Since the suffix R_k is followed by \$, those suffixes are lexicographically larger than R_k. Also, F_{k-4} occurs only at the beginning and at the end of $R_k = H_{k-3}ab$: F_{k-4} only occurs at the beginning of H_{k-3}, by Lemmas 33 and 34, and because R_k and F_{k-4} both finish with ab, F_{k-4} does not occur as a suffix of $H_{k-3}a$. So, the third phrase of the lex-parse of F_k is of length f_{k-4}.

The new rest R'_k is of length f_{k-5}. Also, by Eq. (3),

$$\begin{aligned} F_k &= F_{k-2}F_{k-2}F_{k-5}F_{k-4} \\ &= F_{k-2}F_{k-2}F_{k-5}F_{k-5}F_{k-6} \\ &= F_{k-2}F_{k-2}F_{k-5}F_{k-6}F_{k-7}F_{k-6} \\ &= F_{k-2}F_{k-2}F_{k-4}F_{k-7}F_{k-6}. \end{aligned}$$

Then $R_{k-1} = F_{k-7}F_{k-6}$. Similarly as for R_k, by Lemma 37, all the occurrences of F_{k-6} starting at positions $1 \leq p \leq 2f_{k-2} + f_{k-4}$ are lexicographically larger than R_{k-1}. Also, F_{k-6} occurs only at the beginning and at the end of $F_{k-7}F_{k-6}$. We then have that the fourth phrase is of length f_{k-6}.

The process continues in the same way up to f_5. At this point, the rest of F_k is aab. We prove that the last three phrases of the lex-parse of F_k are of length 1. First, the suffix aab is the lexicographically smallest suffix of F_k that begins with a, by Lemma 36 and because F_k is terminated in \$. Thus, the first a of aab is an explicit phrase of length 1. Then, the suffixes

[9]Using the notation of Lemma 10, $R_{f_{k-2}-1}$ is the odd Fibonacci word F_k of length $f_{k-1} + f_{k-2}$, and R_0 is the smallest cyclic rotation of F_k. Thus, after x applications of ϱ starting at $f_{k-2} - 1$, we get the first symbol of R_0 from the first symbol of F_k (i.e., $\varrho^x(f_{k-2} - 1) = 0$).

that are lexicographically smaller than ab begin with aa. Thus, the length of the next phrase is also 1. Finally, the suffix b is the lexicographically smallest suffix of F_k that begins with b. Thus, b is an explicit phrase of length 1.

Therefore, the lengths of the phrases of the lex-parse of F_k are

$$f_{k-1} - 1, f_{k-4} + 2, f_{k-4}, f_{k-6}, \ldots, f_5, 1, 1, 1$$

and the number of phrases is $5 + \frac{k-7}{2}$. □