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 the optimal one when phrases can be copied only from the left. While $z$ can be computed in linear time with a greedy algorithm, almost nothing has been known for decades about its approximation ratio 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 proceed by observing that Lempel-Ziv is just one particular case of greedy parses, meaning that the optimal value of $z$ is obtained by scanning the text and maximizing the phrase length at each step, and of ordered parses, meaning that there is an increasing order between phrases and their sources. 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 [5] 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 very recent case 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, from versioned document repositories like Wikipedia, from source code repositories like GitHub, etc. Their growth is outpacing Moore’s Law by a wide margin [55]. Understanding the compressibility of highly repetitive texts is important to properly compress those huge 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 $k$th order per-symbol entropy of $TT$ is the same as that of $T$, if $k \ll n$ [40] Lem. 2.6], whereas 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.

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) scheme [57]. Such a scheme parses the text into phrases such that each phrase appears somewhere else in the text (or it is

---

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 Department of Computer Science, University of Pisa, Italy.

https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia
We review basic concepts about strings, compression measures, and others. Table I summarizes our notation along the article.

\^An article implying \(z = \Omega(b \log n)\) [25] 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 $S[1 \ldots \ell] = S[1]S[2] \cdots S[\ell]$ of symbols, of length $|S| = \ell$. A substring (or factor) $S[i] \cdots S[j]$ of $S$ is denoted $S[i \ldots j]$. A suffix of $S$ is a substring of the form $S[i \ldots \ell] = S[i]$, and a prefix is a substring of the form $S[1 \ldots i] = S[1 \ldots i]$. The juxtaposition of strings and/or symbols represents their concatenation; the explicit infix operator “.” can also be used.

We will consider parsing or compressing a string $T[1 \ldots n]$, called the text, over alphabet $[1 \ldots \sigma]$. 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 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 $k$th 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 $\sigma$. It contains all the distinct substrings of length $k$ over $[1 \ldots \sigma]$, and it is of the minimum possible length, $\sigma^k + \sigma - 1$.

B. Bidirectional Schemes (b)

A bidirectional scheme \[57\] partitions $T[1 \ldots n]$ into $b$ phrases $B_1, \ldots, B_b$, such that each phrase $B_i = T[t_i \ldots t_i + \ell_i - 1]$ is either (1) copied from another substring $T[s_i \ldots s_i + \ell_i - 1]$ (called the phrase source) with $s_i \neq t_i$ and $\ell_i \geq 1$, which may overlap $T[t_i \ldots t_i + \ell_i - 1]$, 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 $f : [1 \ldots n] \rightarrow [1 \ldots n] \cup \{-1\}$ so that,

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

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

We use $b$ to denote the smallest bidirectional scheme, which is NP-complete to compute \[22\].

Example: Consider the text $T = \text{alabarala}\text{barad}a\$. A bidirectional scheme of $b = 10$ phrases is $\text{ala}|b|a|\text{x}|a|\text{x}|\text{alab}|a||\text{albar}|d|a|\$, where we have underlined the explicit symbols. For example, the source of phrase $B_3 = T[1 \ldots 3] = \text{ala}$ is $T[7 \ldots 9]$, and the source of phrase $B_7 = T[9 \ldots 14] = \text{alabar}$ is $T[1 \ldots 6]$. To extract $T[11]$, we follow the chain $f(11) = 3, f(3) = 9, f(9) = 1, f(1) = 7,$ and $f(7) = -1$ 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 an explicit symbol. This means that the source $T[s_i \ldots s_i + \ell_i - 1]$ of the target $Z_i = T[t_i \ldots t_i + \ell_i - 1]$ 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 $\ell_i$ (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 \[51, 57\]. This is what we call 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.

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 $k = o(\log_\sigma n)$ \[59\] (thus, in particular, $z_{no} = O(n/\log_\sigma n)$).
denote the minimum possible size of an SLP that generates a rule, assumed to be of size 2, means that for a previous nonterminal $g$, $g$ expands to $\text{exp}(r) = 2$, total number of nodes is at most $\text{exp}(g) = 2$. Further, on the string family $T = ahbca$, it holds that $g_{rt} \leq 2$ and $g = \Theta(\log n)$, and thus it holds that $g = \Omega(g_{rt} \log n)$ (as well as $z_{no} = \Omega(g_{rt} \log n)$).

A well-known relation between $z_{no}$ and $g$ is $z_{no} \leq g = O(z_{no} \log n/z_{no}))$ [23, Lem. 8]. Those papers exhibit $O(\log n)$-approximations to the smallest grammar, as well as several others [34, 28, 29]. A negative result about the approximation are string families where $g = \Omega(z_{no} \log n/\log \log n)$ [8, 26] and even $g_{rt} = \Omega(z_{no} \log n/\log \log n)$ [8]. The size $g$ is also bounded in terms of the statistical entropy [35] and of the empirical entropy [48], thus it always holds $g = O(n/\log \log n)$.

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 $A$ has a single leaf child labeled $a$ if $A \rightarrow a$, or two internal children labeled $B$ and $C$ if $A \rightarrow BC$. The grammar tree prunes the parse tree by leaving only one internal node labeled $X$ 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, and thus the total number of nodes is at most $2g + 1$. Therefore, the grammar tree has at most $g + 1$ leaves.

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

E. Collage Systems [34] are a generalization of RLSPs that also support truncation. Specifically, nonterminals can be of the form $A \rightarrow a$ for a terminal $a$, $A \rightarrow BC$ for previous nonterminals $B$ and $C$, and $A \rightarrow B^k$, $A \rightarrow B^{|t|}$ and $A \rightarrow |t| B$ for a previous nonterminal $B$ and positive integers $k$ and $t$. The last two rules mean that $\text{exp}(A) = \text{exp}(B)[1 \ldots t]$ and $\text{exp}(A) = \text{exp}(B)[\text{exp}(B)] - t + 1 \ldots \text{exp}(B)]$, respectively (it must hold that $t \leq |\text{exp}(B)|$). 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 $c \leq g_{rt}$ and $c = O(z \log z)$ [34].
Example: The following collage system to generate the text \( T = alabaralalabardas \) is actually less efficient than the SLP (it uses \( c = 17 \) rules), but it illustrates all the operations: \( A \to a, B \to b, D \to d, L \to l, R \to r, Z \to S, C \to AL, E \to C^3, F \to BA, G \to FR, H \to DA, I \to HZ, J \to E[5], K \to JG, M \to [6]K, N \to MK, O \to NI \). The nonterminal \( O \) is the initial symbol. This example also illustrates that the concept of parse and grammar tree do not apply to collage systems; for example the nonterminal \( E \) expands to \( alalal \), which does not exist 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 \( \exp_i(A) \) appears somewhere else in \( T \). Since it is not obvious that, in internal collage systems, we can 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 \( 1 \leq t \leq t' \leq |\exp_i(B)| \), meaning that \( \exp_i(A) = \exp_i(B)[t..t'] \).

Note that any upper bound we provide 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 \( [5] \). Instead, an existing lower bound on \( c \) of the form \( \gamma = O(c) \), where \( \gamma \) 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 \( T \) \( [33] \). We also prove that \( b = O(c) \) for internal collage systems, which improves upon that result because they also prove that \( \gamma = O(b) \) \( [33] \).

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 \( i \)th smallest suffix of \( T \) in lexicographic order. The suffix array can be built in \( O(n) \) time \( [36, 37, 31] \).

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 \( ISA \) \( [32] \).

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 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

\[
LF(i) = C[c] + rank[i],
\]

where \( r = \log_2 n \) is the number of occurrences of symbols less than \( c \) in \( L \), and \( rank[i] \) is the number of occurrences of symbol \( L[i] \) in \( L[1..i] \). Once \( C \) and \( rank \) are computed, we reconstruct \( T[n] = \$ \) and \( T[n-k] = L[LF[k-1](1)] \) 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 \( z = \Omega(r \log n) \) 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 \( r = O(z_{no} \log n) \) \( [11, 50] \): we have \( r = \Theta(n) \), whereas \( z_{no} \) is always \( O(n/ \log n) \) on binary strings.

Example: The BWT of \( T = alabaralalabardas \) is \( L = adl$lrbbarabaaaaa \), which has \( r = 10 \) runs.

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 Jež \( [28, 27] \).

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 obtained by, first, creating new symbols that represent the repetitive areas. On the resulted 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.

Jež [28] shows that there is a way to choose left- and right-symbols so that the partition into blocks enjoys useful properties, including a form of local consistency.

Lemma 1 ([28]). We can partition a string \( S[1 \ldots \ell] \) into at most \((3/4)\ell\) blocks so that, for every pair of identical substrings \( S[i \ldots j] = S[i' \ldots j']\), if neither \( S[i+1 \ldots j-1] \) nor \( S[i' + 1 \ldots j'-1] \) overlap a repetitive area, then the sequence of blocks covering \( S[i+1 \ldots j-1] \) and \( S[i' + 1 \ldots j'-1] \) are identical.

Proof. It is clear that, if \( S[i+1 \ldots j-1] \) and \( S[i' + 1 \ldots j'-1] \) do not overlap repetitive areas, then the parsing of \( S[i \ldots j] \) and \( S[i' \ldots j'] \) may differ only in their first position (if it is part of a repetitive area ending there, or if it is a right-symbol that becomes paired with the preceding one) and in their last position (if it is part of a repetitive area starting there, or if it is a left-symbol that becomes paired with the following one). Jež [28] shows how to choose the pairs so that \( S \) contains at most \((3/4)\ell\) blocks.

Example: A locally-consistent parsing of \( T = \text{alabara|labardo}s \) 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 \( T = \text{al|ab|ar|al|al|ab|ar|d|a}s \), where for example in the two occurrences of \( \text{alabar} \), the sequence of blocks covering \( \text{lab}a \) are identical, \( \text{al|ab|ar} \).

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 \( g_r = O(b \log(n/b)) \), and then that \( z \leq g_r + 1 \). 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 \( z_{no} \leq g + 1 \).

Theorem 2. There always exists an RLSLP of size \( g_r = O(b \log(n/b)) \) that generates \( T \).

Proof. Consider the locally consistent parsing of Def. 4 cutting \( W = T \) into blocks. We will count the number of different blocks we form, as this corresponds to the number of nonterminals produced in the first round.

Recall from Section II-B that our bidirectional scheme represents \( T \) 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 4\( b \). 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 extreme of a phrase.

Example: Let us show how this works on the bidirectional scheme example of Section II-B: \( \text{a|a|b|a|a|a|l|alabar|d|a}s \). We have selected (in bold) one different block from those created in the example of Section II-G. Note that we do not need to select any block that is completely inside a phrase. We now prove that the general case is only slightly worse.

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\(^4\). 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 \( \ell_a, \ell_b \geq 1 \), and let \( \ell = \ell_a + \ell_b \) be the length of the block \( X \). If \( W[p \ldots p + \ell - 1] = X \) is not bordering, then it is strictly contained in a phrase. Thus, by the definition of a phrase, it holds that \( [f(p-1) \ldots f(p + \ell)] = [f(p-1) \ldots f(p + \ell)] \), and that \( W[f(p-1) \ldots f(p) + \ell] = W[p-1 \ldots p + \ell] \). That is, the block appears again at \( [f(p) \ldots f(p + \ell)] \), surrounded by the same symbols. Since Def. 4 first compacts repetitive areas, it must be \( W[f(p) - 1] = W[p - 1] \neq a \) and \( W[f(p) + \ell] = W[p + \ell] \neq b \). Further, since Def. 4 pairs left-with-right symbols, \( a^{\ell_a} \) must be a left-symbol and \( b^{\ell_b} \) must be a right-symbol. The locally consistent parsing must then also form a block \( W[f(p) \ldots f(p + \ell)] \). If this block is bordering, then it will be counted. Otherwise, by the same argument, \( W[f(p - 1) \ldots f(p) + \ell] \) will be equal to \( W[f^2(p - 1) \ldots f^2(p) + \ell] \) and a block will be formed with \( W[f^2(p) \ldots f^2(p + \ell) + 1] \). Since \( f \) has no cycles, there is a \( k > 0 \) for which \( f^k(p) = -1 \). Thus for some \( l \leq k \) it must be that \( X = W[f^l(p) \ldots f^l(p + \ell) + 1] \) is bordering. At the smallest such \( l \), the block \( W[f^l(p) \ldots f^l(p + \ell) + 1] \) will be counted. Therefore, \( X = W[p \ldots p + \ell - 1] \) is already counted somewhere else.

2) The block is a single (original or maximal-run) symbol \( W[p \ldots p + \ell - 1] = a^\ell = X \), for some \( \ell \geq 1 \). It also holds that \( [f(p-1) \ldots f(p + \ell)] = [f(p-1) \ldots f(p) + \ell] \) and \( W[f(p) - 1] = W[p - 1] \neq a \) and \( W[f(p) + \ell] = W[p + \ell] \neq a \), the parsing forms the same maximal run \( X = a^\ell = W[f(p) \ldots f(p + \ell) + 1] \). Moreover, since \( W[p \ldots p + \ell - 1] \) is not bordering, the previous and next blocks

\(^4\)For this case, we could have defined bordering in a stricter way, as the first or last block of a phrase.
on our preceding example, and need 3 nonterminals, $Y = \text{block surrounded by } W$.

Fig. 2. The two cases of Theorem 2. On top, case 1, where a block $X = W[p \ldots p + \ell - 1] = a^l b^k$ 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 $W(f(p) \ldots f(p) + \ell - 1) = X$. This is repeated until a phrase boundary (thick vertical line) appears near $X$ (so that occurrence of $X$ is bordering). On the bottom, case 2, where $X = W[p \ldots p + \ell - 1] = a^l$ 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 $W[f(p) \ldots f(p) + \ell - 1] = X$ because the blocks $Y = W[p' \ldots p - 1]$ and $Z = W[p + \ell \ldots p'']$ are strictly inside a phrase. Again, this is repeated until hitting a phrase boundary nearby.

produced by the parsing, $Y = W[p' \ldots p - 1]$ and $Z = [p + \ell \ldots p'']$, are also strictly inside the same phrase, and therefore they also appear preceding and following $W[f(p) \ldots f(p) + \ell - 1]$, at $Y = W[f(p') \ldots f(p) - 1]$ and $Z = [f(p) + \ell \ldots f(p'')]$.

Since $a^l$ was not paired with $Y$ nor $Z$ at $W[p \ldots p + \ell - 1]$, the parsing will also not pair them at $W[f(p) \ldots f(p) + \ell - 1]$. Therefore, the parsing will leave $a^l$ as a block also in $[f(p) \ldots f(p) + \ell - 1]$. If $W[f(p) \ldots f(p + \ell - 1)]$ is bordering, then it will be counted, otherwise we can repeat the argument with $W[f^2(p - 1) \ldots f^2(p) + \ell]$ and so on, as before.

Therefore, we produce at most 4b distinct blocks, and the RLSP has at most 12b nonterminals (for $X = a^l b^k$ we may need 3 nonterminals, $A \rightarrow a^k$, $B \rightarrow b^k$, and $C \rightarrow AB$).

For the second round, we create a reduced sequence $W'$ from $W$ by replacing all the blocks of length 2 or more by their corresponding nonterminals. The new sequence is guaranteed to have length at most $(3/4)n$ by Lemma 1.

We then define a new bidirectional scheme (recall Section II-B) 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 $W'$.
2. For the phrases $B_i = W[t_i \ldots t_i + \ell_i - 1]$ 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 $W[s_i \ldots s_i + \ell_i - 1]$ (with $s_i = f(t_i)$), where the same sequence of blocks is formed by our arguments above. We then form a phrase in $W'$ preceding the sequence of nonterminals associated with the blocks of $B'_i$ (all of which are non-bordering), pointing to the identical sequence of nonterminals that appear as blocks inside $W[s_i \ldots s_i + \ell_i - 1]$.

Example: On our preceding example, $alabar|a|a|l|alabar|d|a|l$, we define the nonterminals $A \rightarrow ab$, $B \rightarrow ar$, $C \rightarrow al$, and $D \rightarrow a|l$. We then create $W' = C[A][B][C][C][A][B][D][D]$, where we show the derived bidirectional scheme and underline the explicit symbols. Recall that, to make this small example interesting, we have been stricter about which blocks are bordering.

To bound the total number of nonterminals generated, let us call $W_k$ the sequence $W$ after $k$ iterations (so $T = W_0$) and $N_k$ the number of distinct blocks created when converting $W_k$ into $W_{k+1}$.

In the first iteration, since there may be up to 4 bordering blocks around each phrase limit, we may create $N_1 \leq 4b$ distinct blocks. Those blocks become new explicit symbols in the bidirectional scheme of $W' = W_1$. Note that those explicit symbols are grouped into $b$ regions of up to 4 consecutive symbols. In each new iteration, $W_k$ is parsed into blocks again. We have shown that the non-bordering blocks formed are not distinct, so we can focus on the number of new blocks produced to parse each of the $b$ regions and near their surrounding phrase boundaries. The parsing produces at most 4 new distinct blocks extending each region (i.e., at the phrase boundaries surrounding the region). 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 decrease the length of the regions, which only grow by $4b$ (explicit symbols) per iteration. Intuitively, each new nonterminal created inside a region decreases its length, and thus both numbers cancel out. We now make the argument more precise.

Let $n_k$ be the number of new distinct blocks produced when parsing the regions themselves. Therefore it holds that the number $N_k$ of distinct blocks produced in the $k$th iteration is at most $4b + n_k$, and the total number of distinct blocks created up to building $W_k$ is

$$\sum_{i=0}^{k-1} N_i \leq 4bk + \sum_{i=0}^{k-1} n_i.$$
On the other hand, for each of the $n_k$ blocks created when parsing a region, the length of the region decreases at least by 1 in $W_{k+1}$, that is, there is one explicit symbol less in $W_{k+1}$. Let us call $C_k$ the number of explicit symbols in $W_k$. Since only the 4 new bordering blocks surrounding each region are converted into explicit symbols when creating $W_k$, it holds that $C_k \leq 4bk$ for all $k > 0$. Moreover, it holds that $C_{k+1} \leq C_k + 4b - n_k$, and thus

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

It follows that $\sum_{i=0}^{k-1} n_i \leq 4bk$, and thus

$$\sum_{i=0}^{k-1} N_i \leq 8bk.$$ 

Since each nonterminal may need 3 rules to represent a block, a bound on the number of nonterminals created is $24bk$. The idea is illustrated in Figure 3.

After $k$ rounds, the sequence is of length at most $(3/4)^k n$ and we have generated at most $24bk$ 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 3, we can also bound the size $z$ of the Lempel-Ziv parse that allows overlaps. The size without allowing overlaps is known to be bounded by the size of the smallest SLP, $z_{no} \leq g + 1$ [52], [8]. We can easily see that $z \leq g_{rl} + 1$ 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 3. The Lempel-Ziv parse (allowing overlaps) of $T$ always produces $z \leq g_{rl} + 1$ 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 $Z[1..z]$ obtained by traversing the leaves of the grammar tree left to right. For a 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 $X \rightarrow Y^t$ as follows. First, we expand them to $X \rightarrow Y \cdot Y^{t-1}$, 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 $g_{rl} + 1$ 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 $g_{rl} + 1$ exists, and the result follows because Lempel-Ziv is the optimal left-to-right parse [41 Thm. 1].

By combining Theorems 2 and 3, 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 4. 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 3 with the facts that $g = O(z \log(n/z))$ [23] Lem. 8] and $z_{no} \leq g + 1$ [52], [8].

**Lemma 5.** It always holds that $g, z_{no} = O(b \log^2(n/b))$. 

Fig. 3. Illustration of Theorem 2. On top we see the limit between two long phrases of $W_0$. In this example, the blocking always pairs 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 symbols. 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.
IV. LOWER BOUNDS

In this section we prove that the upper bound of Theorem \[\text{II-F}\] 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 valid bidirectional scheme of size \(2r\). This proves that \(r = \Omega(b)\). Then we use 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 \(\text{BWT}[\text{ISA}[1]] = \$\) is a size-1 run, and let \(t_{r+1} = n + 1\), so that \(T\) is partitioned into phrases \(T[t_1 \ldots t_{r+1} - 1]\). Let also \(\phi(p) = \text{ISA}[\text{ISA}[p] - 1]\) if \(\text{ISA}[p] > 1\) and \(\phi(p) = \text{ISA}[n]\) otherwise. Then we define the bidirectional scheme of the BWT:

1. For each \(1 \leq i \leq r\), \(T[t_i \ldots t_{i+1} - 2]\) is copied from \(T[\phi(t_i) \ldots \phi(t_{i+1} - 2)]\).
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 \(\text{albaralabarada}\$\).

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

**Lemma 6.** Consider the pair of positions \(T[q - 1 \ldots q]\) within a phrase. Let them be pointed from \(\text{ISA}[x] = q\) and \(\text{ISA}[y] = q - 1\), therefore \(\text{ISA}[q] = x\), \(\text{ISA}[q - 1] = y\), and \(\phi(p) = \phi(q)\). Now, since \(q\) is not a position at the beginning of a phrase, \(x\) is not the first position in a BWT run. Therefore, \(\text{BWT}[x - 1] = \text{BWT}[x]\). Recalling the formula of Section II-F to compute \(\phi(p) = \text{ISA}[\text{ISA}[x] - 1]\), this follows that \(\phi(p) = \phi(q)\).

**Proof.** Then,

\[
\phi(q - 1) = \text{ISA}[\text{ISA}[q - 1] - 1] = \text{ISA}[y - 1] = \text{ISA}[\phi(p) - 1] = \text{ISA}[\phi(q) - 1] = \phi(q) - 1.
\]

It also follows that

\[
T[q - 1] = \text{BWT}[x] = \text{BWT}[x - 1] = T[p - 1] = T[\phi(q) - 1].
\]

**Example:** The suffix array of \(T = \text{albaralabarada}\$\) is \(\text{ISA} = (17, 16, 3, 11, 1, 9, 7, 5, 13, 4, 12, 15, 2, 10, 8, 6, 14)\) and the \(\phi\) function is \(\phi = (11, 15, 16, 13, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 12, 17, 14)\). For example, \(\phi(1) = 11\) because the suffix lexicographically preceding \(T[11\ldots] = T[11\ldots]\). Now, let \(q = 10\), which is inside the same phrase of \(q - 1 = 9\) in the parse \(\text{albaralabarada}\$\). Then \(\text{ISA}[x = 14]\), whereas \(T[q - 1] = 9\) is pointed from \(\text{ISA}[y = 6]\). Thus \(\text{BWT}[x - 1] = \text{BWT}[x]\).}

**Theorem 7.** The bidirectional scheme of the BWT is a valid bidirectional scheme, thus it always holds \(b \leq 2r\).
Proof. By Lemma $[6]$ it holds that $\phi(g - 1) = \phi(g) - 1$ if $[g - 1..g]$ is within a phrase, and that $T[q - 1] = T[\phi(g) - 1]$. Therefore, we have that $\phi(t_i + k) = \phi(t_i) + k$ for $0 \le k < \ell_i = t_{i+1} - t_i$, and then $T[\phi(t_i)...\phi(t_{i+1} - 2)]$ is indeed a contiguous range of length $\ell_i$. 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 $[5]$ it is correct to copy from $T[\phi(p)]$ to $T[p]$ whenever $p$ (which is $q - 1$ in Lemma $[6]$) 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 $2r \ge b$.

\[ \Box \]

Example: We can recover $T$ from our bidirectional scheme $\phi[|a|a|b|a|c|a|a|a|b|a|c|a|a|b|a|c|a|b|a|c|a|a|b|a|c|a|a|b|a|c|a|a|b|a|c] \$ 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 between $z$ and $b$. Before, we prove a further property of the cyclic rotations of the Fibonacci words we make use of.

Lemma 8. In every even Fibonacci separation $F_k$, 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 $k$th Fibonacci word $F_k$ by defining two functions: $\varrho: [0..f_{k-1}] \rightarrow [0..f_k - 1]$, defined as

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

and $\varphi: [0..f_{k-1}] \rightarrow \{a, b\}$, defined as

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

where they index the strings from position 0. They prove that the cyclic rotations of $F_k$ are the words $R_x = r_0r_1 \cdots r_{f_k-1}$, where $r_k = \varphi(\varrho(x))$, for $0 \le x \le f_k - 1$. The lexicographic ordering of the cyclic rotations of $F_k$ is $R_0 < R_1 < \cdots < R_{f_k - 1}$ [44] proof of Thm. 9]. If $k$ is even, then $F_{k} = R_{f_k - 2} [44]$ Thm. 6]. Then, since $F_k[i] = R_{f_k - 2}[i] = \phi(\varrho(f_{k-2})) = \phi(\varrho(i + 1)) = R_0[i + 1]$, and $R_0$ is the lexicographically smallest cyclic rotation, 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 $k$.

\[ \Box \]

Theorem 9. 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 $z = O(\log n)$ $[14]$, 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 $[8]$ 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 $F_k[i..f_k]\prec F_k[j..f_k]$, 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 $z = \Omega(\log n)$ then follows from the fact that $b = O(r)$, by Theorem $[7]$.

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

Lemma 10. It always holds that $g \log_2 n \le n H_k + o(n \log \sigma)$ for any $k = o(\log \sigma n)$, thus $g = O(n/\log \sigma n)$.

Proof. Let $z_{758}$ be the size of the Lempel-Ziv 1978 (LZ78) parsing $[59]$ of $T$. Then, it holds that $z_{758} \log_2 n \le n H_k + o(n \log \sigma)$ for $k = o(\log \sigma n)$ $[49]$ 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 $z_{758}$, it holds that $g \le z_{758}$ and the result follows. The final claim is a consequence of the fact that $H_k \le \log \sigma$ for all $k$.

\[ \Box \]

Theorem 11. There is an infinite family of strings over an alphabet of size 2 for which $r = \Omega(g \log n)$.

Proof. By Lemma $[10]$ 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 $r = \Theta(n)$ $[11]$. The result follows.

\[ \Box \]
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 parses, and prove some general results on them.

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

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

Lemma 12. 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 \(B_i = T[t_i \ldots t_i + \ell_i - 1]\) is copied from \(T[s_i \ldots s_i + \ell_i - 1]\). There cannot be a cycle in \(f\) because, by definition, \(T[f(p) \ldots] < T[p \ldots]\) for every position \(p\) inside every such phrase \(B_i\).

Lemma 13. 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 \(T[p \ldots]\) the value \(h(p) = \min\{k \geq 0, f^k(p) = -1\}\). Now \(\preceq\) can be any total order on \([1 \ldots n]\) compatible with \(h(p)\), that is, such that if \(h(p') < h(p)\) then \(p' \prec p\) (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 \(T[f(p) \ldots] < T[p \ldots]\). 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 \(T[1 \ldots n]\). 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 \(T[s_i \ldots] < T[t_i \ldots]\) defined as \(s_i < t_i\). 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 [51], [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 \(T[1 \ldots n]\) 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 14. Every greedy parse is ordered-optimal.

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 \(B'_1 = T[t'_1 \ldots t'_i + \ell'_i - 1]\) for \(1 \leq i \leq u'\) and \(u' < u\), then there must be a first phrase where \(t'_{i+1} > t_{i+1}\). Since it is the first, it must hold that \(t'_i \leq t_i < t_{i+1} \prec t'_{i+1}\). Let us call \(\delta = t_i - t'_i < \ell'_i = t'_{i+1} - t'_i\). Therefore, there is a source \(T[s'_i + \delta \ldots s'_i + \ell'_i - 1] = T[t'_i \ldots t'_i + \ell'_i - 1]\) such that \(T[s'_i + j \ldots] < T[t'_i + j \ldots]\) for all \(0 \leq j < \ell'_i\). Then it also holds that \(T[s'_i + \delta \ldots s'_i + \ell'_i - 1] = T[t_i \ldots t_{i+1} - t_i = \ell_i\) and that \(T[s'_i + \delta + j \ldots] < T[t_i + j \ldots] = T[t_i + j \ldots] \) for all \(0 \leq j < \ell'_{i+1} - t_i = \ell_i\). Therefore, there exists a suffix \(T[s'_i + \delta \ldots]\) that shares with \(T[t_i \ldots]\) 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 \(T[t_i \ldots]\).

Theorem 15. The greedy parse of any ordered parse can be obtained in \(O(n^3)\) evaluations of \(\prec\).

Proof. We obtain the phrase lengths \(\ell_i\) left to right, so that their starting points are \(s_1 = 1\) and \(s_{i+1} = s_i + \ell_i\). To find the length \(\ell_i\) of each new phrase \(T[s_i \ldots s_i + \ell_i - 1]\), we compare the suffix \(T[s_i \ldots]\) with every other suffix \(T[p \ldots]\) symbol by symbol, until we find the smallest \(j_p\) such that \(T[p + j_p] > T[s_i + j_p]\) or \(p + j_p > n\) or \(s_i + j_p > n\). Then we have \(s_i = \max_{p \neq s_i} j_p\). 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 16. Finding the order \(\preceq\) that produces the smallest ordered parse on \(T\) is NP-hard.

5The order is called \(\preceq\) because it must be reflexive, yet we use \(x < y\) to indicate \(x \prec y\) and \(x \neq y\), that is, \(x\) is strictly smaller than \(y\) under order \(\preceq\).
Proof. One of those orders $\leq$ yields the smallest bidirectional scheme, by Lemma 13. Once we have the best order $\leq$, we find the parse itself greedily in time $O(n^3)$, by Theorems 14 and 15. Thus we obtain the smallest bidirectional scheme, which is NP-hard to find.

On the other hand, we now show that, under certain favorable kinds of orders $\leq$, 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 $\leq$ on text suffixes is extensible if $T[s.] \prec T[t.]$ and $T[s] = T[t]$ implies $T[s+1.] \prec T[t+1.]$.

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

Theorem 17. Any ordered-optimal parse of $T$, for any extensible order $\leq$, produces $u \leq g + 1$ 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 $g + 1$. Analogously to the proof of Theorem 3, 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 $\leq$. This tree has up to $g + 1$ 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 $u \leq g + 1$ 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 $u \leq g + 1$. The other results are immediate consequences of Lemma 10 and Theorem 11.

Theorem 18. Any ordered-optimal parse of $T$ that allows sources and targets overlap, under any extensible order $\leq$, produces $u \leq g_{rt} + 1$ phrases. Thus it holds that $u = O(b \log(n/b))$.

Proof. We prove the theorem of Theorem 17 so as to consider the rules $X \rightarrow Y^r$. These can be expanded either to $X \rightarrow Y \cdot Y^{r-1}$ or to $X \rightarrow Y^{t-1} \cdot Y$. 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 $X \rightarrow Y \cdot Y^{t-1}$, let $Y$ expand to $T[x \cdot y \cdot ... z \cdot y]$ and $Y^{t-1}$ expand to $T[y \cdot z \cdot y]$. We then define $T[y \cdot z]$ as the target of the source $T[x \cdot x + z - y]$. If we instead choose $X \rightarrow Y^{t-1} \cdot Y$, we then define $T[x \cdot x + z - y]$ as the target of the source $T[y \cdot z \cdot y]$. 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 $T[x \cdot y \cdot ... z \cdot y]$ or $T[y \cdot z]$ is the target of the source $T[x \cdot x + z - y]$. Further, the argument used in the proof of Theorem 17 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 $g_{rt} + 1$ leaves in the grammar tree, the ordered parse is of size $u \leq g_{rt} + 1$, and therefore the optimal one is also of size $u \leq g_{rt} + 1$. By Theorem 2 we also have $u = O(b \log(n/b))$.

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

Theorem 19. Any ordered parse, under any extensible order $\leq$, 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 $\leq$.

Proof. We first compute the suffix array $SA$ of $T$ in $O(n)$ time (recall Section 11.3), and from it, the suffix tree of $T$ [58] can also be built in $O(n)$ time [32]. The suffix tree is a compact trie storing all the suffixes of $T$, so that we can descend from the root and, in time $O(n)$, find the interval $SA[sp \ldots ep]$ of all the suffixes starting with a given string of length $m$.

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

We now create the parse phrase by phrase. To produce the next phrase $T[p \ldots]$, we enter the suffix tree from the root with the successive symbols $T[p + j]$, for $j \geq 0$. At each step, the suffix tree gives us the range $SA[sp_j \ldots ep_j]$ of the suffixes of $T$ starting with $T[p \ldots p + j]$. We then find $K[RMQ(sp_j, ep_j)]$, which is the smallest rank of any occurrence of $T[p \ldots p + j]$ in $T$. If this is less than $IO[p]$, then there is a smaller occurrence of $T[p \ldots p + j]$ and we continue with the next value of $j$. The process stops when $K[RMQ(sp_j, ep_j)] = IO[p]$, that is, $T[p \ldots p + j]$ is its smallest occurrence, so we cannot copy it from a smaller one. At this point, the new phrase is $T[p \ldots p + j - 1]$ if $j > 0$, or the explicit symbol $T[y]$ 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
Theorem 24. The lex-parse is the smallest lexicographic parse. Thus, $v \leq |lcpcomp|$, $v = O(b \log(n/b))$, $v \log_2 n \leq nH_b + o(n \log \sigma)$ for any $k = o(\log \sigma)$, $v = O(n/\log \sigma)$, and there are text families where $r = \Omega(v \log n)$.

Proof. By Theorem 14 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 $T[t_i \ldots]$ and any other suffix that is lexicographically smaller than it.

The other results are immediate consequences of Lemmas 21 and 22, Theorem 18, Lemma 10, and Theorem 17.

VI. LEXICOGRAPHIC Parses

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

**Definition 10.** A lexicographic parse of $T[1 \ldots n]$ is an ordered parse where $T[s_1 \ldots] < T[t_1 \ldots]$ 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 20.** The order $T[s_1 \ldots] < T[t_1 \ldots]$ iff the suffix $T[s_1 \ldots]$ lexicographically precedes $T[t_1 \ldots]$, is extensible.

Proof. If $T[s_1 \ldots]$ lexicographically precedes $T[t_1 \ldots]$ and $T[s] = T[t]$, then by definition of lexicographic order, $T[s + 1 \ldots]$ lexicographically precedes $T[t + 1 \ldots]$.

By Lemma 12, 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 21.** The bidirectional scheme induced by the BWT in Def. 5 is a lexicographic parse of size 2$r$.

Proof. The definition uses function $f(p) = \phi(p) = SA[ISA[p] - 1]$ to copy from $T[\phi(t_1) \ldots \phi(t_i) + \ell_i - 1]$ to $T[t_i \ldots t_i + \ell_i - 1]$, where $\ell_i = t_{i+1} - t_i - 1$ (recall Theorem 7). 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 IV). 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 22.** The lexicographic factorization $lcpcomp$ is a lexicographic parse.

Proof. The property can be easily seen from step 2 of the algorithm [12, Sec. 3.2]: the authors report a phrase (i.e., source-length pair) $(SA[i - 1], LCP[i])$ expanding to text substring $T[SA[i] \ldots SA[i] + LCP[i] - 1]$. We write $LCP[i]$ because entries of the LCP array may be decremented in step 4, therefore $LCP[i] \leq LCP[i]$ at any step of the algorithm for any $1 \leq i \leq n$. This however preserves the two properties of lexicographic parsings: $T[SA[i] \ldots SA[i] + LCP[i] - 1] = T[SA[i - 1] \ldots SA[i - 1] + LCP[i] - 1]$ (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 19. 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 $T[1 \ldots n]$, with arrays $SA$, $ISA$, and $LCP$, is defined as a partition $T = L_1, \ldots, L_v$ such that $L_i = T[t_i \ldots t_i + t_i - 1]$, 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 $T[t_i \ldots t_i + \ell_i - 1]$ are copied from $T[s_i \ldots s_i + \ell_i - 1]$, where $s_i = SA[ISA[t_i] - 1]$.

Example: The lex-parse of our example string is $a[a][a][a][a][a][alabar][alabar][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a][a]
Note that, unlike $v$, $z$ can be $\Omega(r \log n)$, as shown in Theorem 25. Thus, $v$ offers a better asymptotic bound with respect to the number of runs in the BWT. The following corollary is immediate.

**Theorem 25.** 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 26.** 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 6). 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 $z < \frac{5}{3}v$.

**Lemma 27.** On the alphabet $\{1, \ldots, \sigma + 1\}$, where $\sigma$ is not a multiple of 3, consider the string $S_1 = (23 \ldots \sigma 1)^3$. 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 $\sigma$ 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 $2\sigma$ further phrases to cover $S_\sigma = 23(\sigma+1)56(\sigma+1)\ldots$ with phrases of alternating length 2 and 1: each such pair of suffixes $S_\sigma[3i+1\ldots]$ and $S_\sigma[3i+3\ldots]$, for $i = 0, \ldots, \sigma - 1$, do appear in previous substrings $S_j$, but all these are lexicographically larger (because $\sigma$ 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 III 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 25 and 26). On the others (Run-Rich String and Thue-Morse sequences), $z$ is about 30% larger than $v$.

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 .1, the edits are applied to the base version to form the copy, whereas in those with suffix .2, 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 $\frac{z}{v} = 0.6$ we obtained in Lemma 27 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 28.** There is always an internal collage system of $c \leq 4z$ rules generating $T$.

6The file fib41 uses a variant where $F_1 = a$, $F_2 = ba$, and $F_k = F_{k-2}F_{k-1}$.
of Section II-C, where we have underlined the explicit symbols. Figure 6 illustrates the application of Theorem 28 to this parse.

**Proof.** We proceed by induction on the Lempel-Ziv parse. At step \( i \), we obtain a collage system with initial rule \( S_1 \) 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 \( S_2 \).

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 \( i \)th 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 \( i \)th phrase point to a source that is completely inside \( T[1..p_{i-1}] \), precisely \( T[x..y] \) with \( x < p_{i-1} \). Then we add rule \( N_1 \to S_{i-1}^{[x,y]} \), and then \( S_i \to S_{i-1} N_i \).

If, instead, the \( i \)th 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 \( q = \left\lfloor \frac{y-x+1}{p} \right\rfloor \) copies of \( T[x..p+p-1] = T[x..p_{i-1}] \) plus \( T[x..x+(y-x+1) \mod p - 1] \) if \( p \) does not divide \( y-x+1 \) (note that \( q \) may be zero). This can be obtained with \( O_i \to \overline{[p]} S_{i-1} N_i \), \( O_i' \to O_i' \), \( R_i \to R_i' \), \( N_i \to R_i N_i' \), and \( S_i \to S_{i-1} N_i \).

Figure 5 illustrates both cases schematically. □

**Example:** Consider the Lempel-Ziv parse \( T = a | l | a | b | a | r | a | a | [a b a r] | d | a | S \) of Section II-C where we have underlined the explicit symbols. Figure 5 illustrates the application of Theorem 28 to this parse.

**Theorem 29.** There is always a bidirectional scheme of size \( b \leq c + 1 \) for \( T \), for an internal collage system of size \( c \).

**Proof.** We extend the idea of Theorem 3 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 \( c + 1 \) 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 3. Further, each leaf representing a substring rule \( A \rightarrow B[l,t] \) is converted into a single phrase pointing to \( T[x + t - 1 \ldots x + t' - 1] \), where the leftmost occurrence of \( B \) in the parse tree covers the text \( T[x \ldots y] \).

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 \( A \). Since nonterminals are defined in terms of earlier nonterminals, it holds that every position \( p \) of \( T \) is defined in terms of a position \( f(p) \) with a smaller label. \( \square \)

**Example:** The following collage system to generate the text \( T = \text{alabaralabarado} \) is an internal variant of the one given in Section [11][11] \( A \rightarrow a, B \rightarrow b, D \rightarrow d, L \rightarrow l, R \rightarrow r, Z \rightarrow \$, C \rightarrow AL, E \rightarrow CC, F \rightarrow BA, G \rightarrow FR, H \rightarrow DA, I \rightarrow HZ, J \rightarrow EA, K \rightarrow JG, M \rightarrow [6]^K, N \rightarrow MK, O \rightarrow NI \). The corresponding bidirectional scheme induces the parse \( T = \text{alabar}\{a\mid a\mid a\}d[a]\$, where the first phrase is defined by a forward pointer to \( T[9 \ldots 14] \).

**Theorem 30.** There exists an infinite family of strings over an alphabet of size 2 for which \( c = \Omega(v \log n) \), and thus also \( c = \Omega(b \log n) \), for any general collage system of size \( c \).

**Proof.** Fibonacci words do not contain 4 consecutive repetitions of the same substring [30]. Therefore, no internal collage system generating a Fibonacci word contains run-length rules \( A \rightarrow B^k \) with \( k > 3 \), because \( \exp(A) \) does appear in \( T \). Run-length rules with \( k \leq 3 \) can be replaced by one or two rules that are not 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 strings of some length with fewer rules. As a consequence, it holds that \( c = \Omega((\log n) \) On the other hand, by Theorem [33] it holds that Fibonacci words have bidirectional schemes of \( O(1) \) blocks. Further, by Theorems [9] and [24] it holds that \( v = O(1) \) on the even Fibonacci words.

We can extend the result to general collage systems by noting that every nonterminal \( A \rightarrow B^k \) with \( k > 4 \) must be shortened via truncation by more than \( |\exp(B)| \) symbols, before appearing in \( T \). Thus, it can be replaced by \( A \rightarrow B^{k-1} \) and, iteratively, by \( A \rightarrow B^3 \), and thus be replaced by two rules that are not run-length rules. \( \square \)

**VIII. Conclusions**

We have essentially closed the question of which 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, whereas 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 worse 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. A restricted variant we call internal collage systems are shown not to be asymptotically smaller than the smallest bidirectional scheme, and have a logarithmic gap with the lex-parse on some string families. 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. We have also no upper bounds on \( r \) in terms of other measures, for example, can \( r \) be more than \( O(\log n) \) times larger than \( z \) or \( g \)? It might also be that our Theorem 2 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 \( O(g_t) \) \([9]\)) allow for efficient \( O(\log n) \) time access to \( T \); all the other measures need a logarithmic blowup in space to support efficient access \([2, 6, 4, 53, 18, 19, 21]\). 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 1-170048, Chile, by the Millennium Institute for Foundational Research on Data (IMFD), 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 Gortz, and N. Prezza. A separation between run-length SLPs and LZ77. CoRR, abs/1711.07270, 2017.
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 31.** For each $k \geq 5$, it holds that $F_{k-1}F_{k-2} = H_k ab$ and $F_{k-2}F_{k-1} = H_k ba$ 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 $F_{k-1}F_{k-2} = H_k ab$ and $F_{k-2}F_{k-1} = H_k ba$ was proved by Pirillo [49] Lem. 1. \(\square\)

**Lemma 32.** $F_{k-1}$ 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:

\[
F_k = F_{k-1}F_{k-2} = F_{k-2}F_{k-3}F_{k-2} = F_{k-2}F_{k-3}F_{k-3}F_{k-4} = F_{k-2}F_{k-3}F_{k-4}F_{k-5}F_{k-4} = F_{k-2}F_{k-3}F_{k-4}F_{k-5}F_{k-4}.
\]

Assume, by contradiction, that $F_{k-1}$ appears in different positions inside $F_k$. From Eq. 3, we have that $F_a = F_{k-2}F_{k-3}F_{k-4}$. 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 31). Thus, the second occurrence of $F_{k-1}$ must start at a position $p < f_{k-2}$. 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. \(\square\)

**Lemma 33.** 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 $F_5 = ababab$ 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 $B_1 = F_k[1 \ldots f_{k-1} - 2]$ (i.e., all the symbols of $F_{k-1}$ except the last two).

2) The second and third chunks are explicit symbols ($B_2 = F_k[f_{k-1} - 1] = a$ and $B_3 = F_k[f_{k-1}] = b$, if $k$ is odd).

3) The fourth chunk is $B_4 = F_k[f_{k-1} + 1 \ldots f_k]$ (i.e., all the symbols of $F_{k-2}$).

The source of the first chunk, $B_1$, is $F_k[f_{k-2} + 1 \ldots f_k - 2]$, and the source of the fourth chunk, $B_4$, is $F_k[f_{k-2} + 1 \ldots 2f_{k-2}]$. 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, $F_k = F_{k-2}F_{k-3}F_{k-4}F_{k-5}F_{k-4}$, so there is an occurrence of $F_{k-2}$ starting at position $f_{k-2} + 1$ of $F_k$. Hence, $B_4 = F_k[f_{k-2} + 1 \ldots 2f_{k-2}]$. Further, by Eq. 1, we have that $F_k = F_{k-2}F_{k-3}F_{k-2}$, and from Lemma 31 we have that $B_1 = H_{k-1} = F_k[f_{k-2} + 1 \ldots f_k - 2]$.

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 $f : [1 \ldots f_k] \rightarrow [1 \ldots f_k] \cup \{-1\}$ as follows:

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

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 $p + x f_{k-2} - y f_{k-3} = p$, 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, $f_{k-3}$ 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 no way to get back there. So, we are left with $f_{k-1} - 2$ positions to be involved in a cycle, because $f(f_{k-1} - 1) = f(f_{k-1}) = -1$. That is a contradiction. \(\square\)

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.

7Applying 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 $\gcd(f_2, f_1) = 1$. 

Lemma 34. 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 21 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 $abab$ 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 35. 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 34. If $i = 4$, then $f_4 = 3$, and $F_5 = ab$. Then, any factor $W_4$ of $F_i$ of length 3 that begins with $ab$ can only be $W_4 = abF_3$. If $i = 5$, then $f_5 = 5$, and $F_5 = aba$. Then, any factor $W_5$ of $F_k$ of length 5 that begins with $aba$ can only be equal to $W_5 = ababF_3$ or $W_5 = ababa = F_3F_3$.

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

The factor $W_{i+1}$ is equal to $F_{i}G_{i-1}$, where $G_{i}$ will stand for any string of length $f_i$. Thus, $W_{i+1} = F_{i-1}F_{i-2}G_{i-1}$. Since $|F_{i-2}G_{i-1}| = f_{i-1}$, 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) $W_{i+1} = F_{i-1}F_{i-2}F_{i-3}G_{i-1}$.

Case (i) implies $W_{i+1} = F_{i-1}F_{i-2}G_{i-2}$. 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 $W_{i+1} = F_{i-1}F_{i-2}$ or $W_{i+1} = F_{i-1}F_{i-1}$. 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

$W_{i+1} = F_{i-1}F_{i-3}F_{i-2}F_{i-3}G_{i-4} = F_{i-2}F_{i-3}F_{i-3}F_{i-2}F_{i-3}G_{i-4} = F_{i-2}F_{i-3}F_{i-3}F_{i-4}G_{i-4}$.

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 34. If $i+1 \geq 8$, then by Lemma 31 $F_{i-1}F_{i-3}$ 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 [30]. In case (b), by Def. 1 it holds that

$W_{i+1} = F_{i-1}F_{i-3}F_{i-3}F_{i-2}G_{i-4} = F_{i-2}F_{i-3}F_{i-3}F_{i-2}F_{i-3}G_{i-4} = F_{i-2}F_{i-3}F_{i-3}F_{i-4}F_{i-4}G_{i-4}$,

which also contains 4 occurrences of $F_{i-3}$ within $F_k$, a contradiction again [30].

Theorem 26. 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 33. 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 $\ell_1 \leq LCP[ISA[1]]$ (see Def. 11) of the first phrase of the lex-parse of $F_k$ is $f_{k-1} - 2$. From Eq. 11, we have that $F_k = F_{k-2}F_{k-3}F_{k-2}$, and from Lemma 31 we have that $F_k = H_kbaF_{k-2} = F_{k-2}H_{k-1}ab$. 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 $F_{k-1}$ of length $f_{k-1} - 1$ of $F_{k-1}$ appears in $F_k$. By the proof of Lemma 24, $F_k$ finishes with $baab$ and $F_{k-1}$ finishes with $baba$. Then $P_{k-1}$ finishes with $bab$ and $F_{k-1}$ 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 24. 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 32. 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. 11, 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 $S$, 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 \(f_{k-2} - 1 + x f_k = 0 \pmod{f_k}\).

Using Cassini’s identity, \(f_k f_{k-2} - f_k^2 = 1\) \(^{24}\), we replace \(f_k = f_{k-1} + f_{k-2}\) to get \(f_{k-1} f_{k-2} + f_{k-2}^2 - f_{k-1}^2 = f_k f_{k-2} + (f_{k-2} + f_{k-1})(f_{k-2} - f_{k-1}) = f_k f_{k-2} + f_k (f_{k-2} - f_{k-1}) = 1\). This implies \(f_{k-1} f_{k-2} + 1 = 0 \pmod{f_k}\). 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 \(a\), if a suffix \(S\) of \(F_k\) is lexicographically smaller than \(F_k[f_{k-1} - 1 \ldots] = b a F_k - 2\) 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 \(b a F_k - 4\) is the largest string that is a prefix and a suffix of \(b a F_k - 2\).

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. \(6\), we have that \(F_k - 2 = F_k - 4 F_k - 5 F_k - 6\). There are no occurrences of \(F_k - 4\) at positions \(1 < p < f_k - 4\), by the same argument of Lemma \(32\) By Eq. \(6\), we also have that \(F_k - 2 = F_k - 4 F_k - 5 F_k - 6\). There are no occurrences of \(F_k - 4\) at positions \(f_k - 4 < p < f_k - 3\), because \(F_k - 4 = F_k - 5 F_k - 6\) and then \(F_k - 2\) would occur more than twice within \(F_k - 3 F_k - 5\), which is not possible again by the argument of Lemma \(32\). 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 \(31\) the only one of those three occurrences that is preceded by \(b a\) 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. \(5\), we have that \(F_k = F_k - 2 F_k - 2 F_k - 5 F_k - 4\), so \(R_k = F_k - 5 F_k - 4 = H_k - 3 a b\), by Lemma \(31\). Since \(R_k\) starts with \(H_k - 3\), which starts with \(F_k - 4\) by Lemma \(31\) 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 \(35\) we have that all the suffixes of \(F_k\) that start at position \(1 < p < 2 f_{k-2}\), and begin with \(F_k - 4\), also begin with \(F_k - 4 F_k - 5 = H_k - 3 a b > R_k\), by Lemma \(31\) or with \(F_k - 5 F_k - 4 = R_k\). Since the suffix \(R_k\) is followed by \(a\), those suffixes are lexicographically larger than \(R_k\). Also, \(F_k - 4\) occurs only at the beginning and at the end of \(R_k = H_k - 3 a b\). \(F_k - 4\) only occurs at the beginning of \(H_k - 3\), by Lemmas \(31\) and \(32\) and because \(R_k\) and \(F_k - 4\) both finish with \(a b\), \(F_k - 4\) 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. \(3\),

\[
F_k = F_k - 2 F_k - 2 F_k - 5 F_k - 4
= F_k - 2 F_k - 2 F_k - 5 F_k - 5 F_k - 6
= F_k - 2 F_k - 2 F_k - 5 F_k - 6 F_k - 7 F_k - 6
= F_k - 2 F_k - 2 F_k - 4 F_k - 7 F_k - 6.
\]

Then \(R_k - 1 = F_k - 7 F_k - 6\). Similarly as for \(R_k\), by Lemma \(33\) all the occurrences of \(F_k - 6\) starting at positions \(1 < p < 2 f_{k-2} + f_{k-4}\) 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 \(a a b\). We prove that the last three phrases of the lex-parse of \(F_k\) are of length \(1\). First, the suffix \(a a b\) is the lexicographically smallest suffix of \(F_k\) that begins with \(a\), by Lemma \(34\) and because \(F_k\) is terminated in \(\$\). Thus, the first \(a\) of \(a a b\) is an explicit phrase of length \(1\). Then, the suffixes that are lexicographically smaller than \(a b\) begin with \(a a\). 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 \(5 + \frac{k-7}{2}\).

\[^{21}\]Using the notation of Lemma \(3\) \(R_{f_{k-2}-1}\) 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 \(\varphi\) starting at \(f_{k-2} - 1\), we get the first symbol of \(R_0\) from the first symbol of \(F_k\) (i.e., \(\varphi^x(f_{k-2} - 1) = 0\)).