On the Approximation Ratio of Lempel-Ziv Parsing *

Travis Gagie[1,2], Gonzalo Navarro[2,3], and Nicola Prezza[4]

[1] EIT, Diego Portales University, Chile
[2] Center for Biotechnology and Bioengineering (CeBiB), Chile
[3] Department of Computer Science, University of Chile, Chile
[4] DTU Compute, Technical University of Denmark, Denmark

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, where phrases can be copied only from the left. While z can be computed in linear time, 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 string 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. On our way, we prove other relevant bounds between compressibility measures.

1 Introduction

Shannon [33] 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 [21], is the notion of empirical entropy [7], 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 [23] or the grammar-compression [20] of the text.

Some text families, however, are not well modeled as coming from a probabilistic source. A very current case is that of *highly repetitive texts*, where most of the text can be obtained by copying long blocks from elsewhere in the same

* Partially funded by Basal Funds FB0001, Conicyt, by Fondecyt Grants 1-171058 and 1-170048, Chile, and by the Danish Research Council DFF-4005-00267.

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 [34]. 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 kth order persymbol entropy of TT is the same as that of T, if $k \ll n$ [22, Lem. 2.6], whereas this entropy measure is generally meaningless for $k > \log n$ [12]. 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% [11]. Similar compression ratios are reported in Wikipedia.[5]

Despite the obvious practical relevance of these compressibility measures, there is not a clear entropy measure useful for highly repetitive texts. The number z of phrases generated by the Lempel-Ziv parse [23] is often used as a gold standard, possibly because it can be implemented in linear time [30] and is never larger than g, the size of the smallest context-free grammar that generates the text [31, 6]. However, z is not so satisfactory as an entropy measure: the value changes 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* [35]. Such a scheme parses the text into phrases such that each phrase appears somewhere else in the text (or it is a single explicit symbol), so that it is possible to recover the text by copying source to target positions in an appropriate order. This is arguably the strongest possible dictionary method, but finding the smallest bidirectional scheme is NP-complete [13]. A relevant question is then how good is the Lempel-Ziv parse as an efficiently implementable approximation to the smallest bidirectional scheme. Almost nothing is known in this respect, except that there are string families where z is nearly $2b$ [35].

In this paper we finally give a tight approximation ratio for z, showing that the gap is larger than what was previously known. We prove that $z = O(b \log(n/b))$, and that this bound is tight as a function of n, by exhibiting a string family where $z = \Omega(b \log n)$. To prove the upper bound, we show how to build a run-length context-free grammar [28] (i.e., allowing rules of the form $X \to Y^t$ that count as size 1) of size $g_{rl} = O(b \log(n/b))$. This is done by carrying out several rounds of locally consistent parsing [17] on top of T, reducing the resulting blocks to nonterminals in each round, and showing that new nonterminals appear only in the boundaries of the phrases of the bidirectional scheme. We then further prove that $z \leq 2g_{rl}$, by extending a classical proof [6] that relates grammar with Lempel-Ziv compression. To prove the lower bound, we consider

[5] `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`

another plausible compressibility measure: the number r of equal-symbol runs in the Burrows-Wheeler transform (BWT) of the text [5]. We prove that the BWT induces a valid bidirectional scheme, and thus $r = \Omega(b)$. Then the bound follows from known string families where $z = \Omega(r \log n)$ [29].

2 Basic Concepts

A string is a sequence $S[1..\ell] = S[1]S[2]\dots S[\ell]$ of symbols. A substring $S[i]\dots S[j]$ of S is denoted $S[i..j]$. A suffix of S is a substring of the form $S[i..\ell]$. The juxtaposition of strings and/or symbols represents their concatenation. We will consider compressing a string $T[1..n]$, called the *text*.

2.1 Bidirectional Schemes

A bidirectional scheme [35] partitions $T[1..n]$ into b chunks $B_1, \dots, B_b$, such that each $B_i = T[t_i..t_i + \ell_i - 1]$ (called a target) is either (1) copied from another substring $T[s_i..s_i + \ell_i - 1]$ (called a source) with $s_i \neq t_i$, which may overlap $T[t_i..t_i + \ell_i - 1]$, or (2) formed by $\ell_i = 1$ explicit symbol.

We define the function $f : [1..n] \to [1..n]$ so that, in case (1), $f(t_i + j) = s_i + j$ for all $0 \leq j < \ell_i$, and in case (2), $f(t_i) = -1$. Then, 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.

Being a valid scheme is equivalent to saying that f has no cycles, that is, there is no $k > 0$ and p such that $f^k(p) = p$: Initially we can set all the explicit positions (type (2)), and then copy sources with known values to their targets. If f has no cycles, we will eventually complete all the positions in T because, for every $T[p]$, there is a $k > 0$ such that $f^k(p) = -1$, so we can obtain $T[p]$ from the symbol explicitly stored for $T[f^{k-1}(p)]$.

We use b to denote the smallest bidirectional scheme, which is NP-complete to compute [13].

2.2 Lempel-Ziv Parsing

Lempel and Ziv [23] define a parsing of T into the fewest possible phrases $T = Z_1 \dots Z_z$, so that each phrase Z_i is a substring (but not a suffix) of $Z_1 \dots Z_i$, or a single symbol. This means that the source $T[s_i..s_i + \ell_i - 1]$ of the target $Z_i = T[t_i..t_i + \ell_i - 1]$ must satisfy $s_i < t_i$, but sources and targets may overlap. It turns out that the greedy left-to-right parsing indeed produces the optimal number z of phrases [23, Thm. 1]. Further, the parsing can be obtained in $O(n)$ time [30, 35].

If we disallow that a phrase overlaps its source, that is, Z_i must be a substring of $Z_1 \dots Z_{i-1}$ 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 [35, Thm. 10 with $p = 1$]. Since the Lempel-Ziv parsing allowing overlaps is optimal among all left-to-right parsings, we also have

that $z_{no} \geq z$. This parsing can also be computed in $O(n)$ time [8]. Note that, on a text family like $T = a^n$, 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 *unidirectional* parsing) and that, for any constant $\epsilon > 0$, there is an infinite family of strings for which $b < (\frac{1}{2} + \epsilon) \cdot \min(z, z^R)$ [35, Cor. 7.1], where z^R is the z of the reversed string.

2.3 Grammar Compression

Consider a context-free grammar (CFG) that generates T and only T [20]. Each nonterminal must be the left-hand side in exactly one rule, and the size g of the grammar is the sum of the right-hand sides of the rules. In general, we will use g to denote the minimum possible size of a grammar that generates T, which is NP-complete to compute [31, 6].

If we allow, in addition, rules of the form $X \to Y^t$, of size 1, the result is a run-length context-free grammar (RLCFG) [28]. We will use g_{rl} to denote the size of the smallest RLCFG that generates T. Thus, it is clear that $g_{rl} \leq g$. Further, on the string family $T = a^n$ it holds that $g = \Omega(g_{rl} \log n)$.

A well-known relation between z_{no} and g is $z_{no} \leq g = O(z_{no} \log(n/z_{no}))$ [31, 6]. Further, it is known that $g = O(z \log(n/z))$ [14, Lem. 8]. Those papers exhibit $O(\log n)$-approximations to the smallest grammar, as well as several others [32, 17, 18]. A negative result about the approximation are string families where $g = \Omega(z_{no} \log n / \log\log n)$ [6, 15] and, recently, $g_{rl} = \Omega(z_{no} \log n / \log\log n)$ [3].

2.4 Runs in the Burrows-Wheeler Transform

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.

The *suffix array* [25] of $T[1..n]$ is an array $SA[1..n]$ storing a permutation of $[1..n]$ so that, for all $1 \leq i < n$, the suffix $T[SA[i]..]$ is lexicographically smaller than the suffix $T[SA[i+1]..]$. Thus $SA[i]$ is the starting position in T of the ith smallest suffix of T in lexicographic order.

The 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.

The *Burrows-Wheeler Transform* of $T[1..n]$, $BWT[1..n]$ [5], 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, which can be built in $O(n)$ time [19]. To obtain T from BWT [5], 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 formula

is $LF(i) = C[c] + \text{rank}[i]$, where $c = L[i]$, $C[c]$ 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 $T[n-k] \leftarrow L[LF^{k-1}(1)]$ for $k = 1, \ldots, n-1$.

The number of equal-symbol runs r in the BWT of T can be bounded in terms of the empirical entropy [24]. However, the measure is also interesting on highly repetitive collections (where, in particular, z and z_{no} are small). For example, there are string families where $z = \Omega(r \log n)$ [29], and others where $r = \Omega(z_{no} \log n)$ [2, 29].

2.5 Locally consistent parsing

A string can be parsed in a *locally consistent* way, in the sense that equal substrings are largely parsed in the same form. We use a variant of locally consistent parsing called recompression [17, 16].

Definition 1. *A* repetitive area *in a string is a maximal run of the same symbol, of length 2 or more.*

Definition 2. *Two segments contained in* $[1..n]$ overlap *if they are not disjoint nor one contained in the other.*

Lemma 1 ([17]). *We can partition a string $S[1..\ell]$ into at most $(3/4)\ell$ blocks so that, for every pair of identical substrings $S[i..j] = S[i'..j']$, if neither $S[i+1..j-1]$ or $S[i'+1..j'-1]$ overlap a repetitive area, then the sequence of blocks covering $S[i+1..j-1]$ and $S[i'+1..j'-1]$ are identical.*

Proof. The parsing is obtained by, first, creating new symbols that represent the repetitive areas. On the resulting sequence, the alphabet (which contains original symbols and created ones) is partitioned into two subsets, left-symbols and right-symbols. Then, every left-symbol followed by a right-symbol are paired in a block. It is then clear that, if $S[i+1..j-1]$ and $S[i'+1..j'-1]$ do not overlap repetitive areas, then the parsing of $S[i..j]$ and $S[i'..j']$ 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). Jez [17] shows how to choose the pairs so that S contains at most $(3/4)\ell$ blocks. □

The lemma ensures a locally consistent parsing into blocks as long as the substrings do not overlap repetitive areas, though the substrings may fully contain repetitive areas.

3 Upper Bounds

In this section we obtain our main upper bound, $z = O(b \log(n/b))$, along with other byproducts. To this end, we first prove that $g_{rl} = O(b \log(n/b))$, and

then that $z \leq 2g_{rl}$. To prove the first bound, we build a RLCFG 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 at the boundaries of the phrases of the bidirectional scheme. The second bound is an easy extension to the known result $z_{no} \leq g$.

Theorem 1. *Let $T[1..n]$ have a bidirectional scheme of size b. Then there exists a run-length context-free grammar of size $g_{rl} = O(b \log(n/b))$ that generates T.*

Proof. Recalling Lemma 1, consider a locally consistent parsing of $W = T$ 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 2.1 that our bidirectional scheme represents T as a sequence of *chunks*, 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 chunk are all different. The number of such *bordering* blocks is at most $4b$. On the other hand, we will show that non-bordering blocks do not need to be considered, because they will be counted somewhere else, when they appear near the extreme of a chunk.

We show that this is true in both types of non-bordering blocks resulting from Lemma 1:

1. The block is a pair of left- and right-alphabet symbols.[6] As these symbols can be an original symbol or a maximal 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 $W[p..p+\ell-1] = X$ is not bordering, then it is strictly contained in a chunk. Thus, by the definition of a chunk, it holds that $[f(p-1)..f(p+\ell)] = [f(p)-1..f(p)+\ell]$, and that $W[f(p)-1..f(p)+\ell] = W[p-1..p+\ell]$. That is, the block appears again at $[f(p)..f(p)+\ell-1]$, surrounded by the same symbols. Since, by the way Lemma 1 works, it must be $W[f(p)-1] = W[p-1] \neq a$ and $W[f(p)+\ell] = W[p+\ell] \neq b$, and a^{ℓ_a} is a left-symbol and b^{ℓ_b} is a right-symbol, the locally consistent parsing must also form a block $W[f(p)..f(p)+\ell-1] = X$. If this block is bordering, then it will be counted. Otherwise, by the same argument, $W[f(p)-1..f(p)+\ell]$ will be equal to $W[f^2(p)-1..f^2(p)+\ell]$ and a block will be formed with $W[f^2(p)..f^2(p)+\ell-1]$. Since f has no cycles, there is a $k > 0$ for which $f^k(p) = -1$. Thus for some $l < k$ it must be that $X = W[f^l(p)..f^l(p)+\ell-1]$ is not bordering. At the smallest such l, the block $W[f^l(p)..f^l(p)+\ell-1]$ will be counted. Therefore, $X = W[p..p+\ell-1]$ is already counted somewhere else and we do not need to count it at $W[p..p+\ell-1]$.
2. The block is a single (original or maximal-run) symbol $W[p..p+\ell-1] = a^\ell$, for some $\ell \geq 1$. It also holds that $[f(p-1)..f(p+\ell)] = [f(p)-1..f(p)+\ell]$ and

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

$W[f(p)-1..f(p)+\ell] = W[p-1..p+\ell]$, because a^ℓ is strictly inside a chunk. Since $W[f(p)-1] = W[p-1] \neq a$ and $W[f(p)+\ell] = W[p+\ell] \neq a$, the parsing forms the same maximal run $a^\ell = W[f(p)..f(p)+\ell-1]$. Moreover, since $W[p..p+\ell-1]$ is not bordering, the previous and next blocks produced by the parsing, $X = W[p'..p-1]$ and $Y = [p+\ell..p'']$, are also strictly inside the same chunk, and therefore they also appear preceding and following $W[f(p)..f(p)+\ell-1]$, at $X = W[f(p')..f(p)-1]$ and $Y = [f(p)+\ell..f(p'')]$. Since a^ℓ was not paired with X nor Y at $W[p..p+\ell-1]$, the parsing will also not pair them at $W[f(p)..f(p)+\ell-1]$. Therefore, the parsing will leave a^ℓ as a block also in $[f(p)..f(p)+\ell-1]$. If $W[f(p)..f(p+\ell-1)]$ is bordering, then it will be counted, otherwise we can repeat the argument with $W[f^2(p)-1..f^2(p)+\ell]$ and so on, as in the previous item.

Therefore, we produce at most $4b$ distinct blocks, and the RLCFG has at most $12b$ nonterminals (for $X = a^{\ell_a}b^{\ell_b}$ we may need 3 nonterminals, $A \to a^{\ell_a}$, $B \to b^{\ell_b}$, and $C \to AB$).

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 define a new bidirectional scheme (recall Section 2.1) on W', as follows:

1. For each bordering block in W, its nonterminal symbol position in W' is made explicit in the bidirectional scheme of W'. Note that this includes the blocks covering the explicit symbols in the bidirectional scheme of W.
2. For the chunks $B_i = W[t_i..t_i+\ell_i-1]$ of W containing non-bordering blocks (note B_i cannot be an explicit chunk), let B'_i be obtained by trimming from B_i the bordering blocks near the extremes of B_i. Then B'_i appears inside $W[s_i..s_i+\ell_i-1]$ (with $s_i = f(t_i)$), where the same sequence of blocks is formed by our arguments above. We then form a chunk in W' with 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 $W[s_i..s_i+\ell_i-1]$.

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 chunk limit, we may create $N_1 \leq 4b$ distinct blocks. Those blocks become new explicit chunks in the bidirectional scheme of $W' = W_1$. Note that those explicit chunks are grouped into b *regions* of up to 4 consecutive chunks. In each new iteration, W_k is parsed into blocks again. We have shown that the blocks formed outside regions (i.e., non-bordering blocks) are not distinct, so we can focus on the number of new blocks produced to parse each of the b regions. The parsing produces at most 4 new distinct blocks extending each 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 $4b$ per iteration.

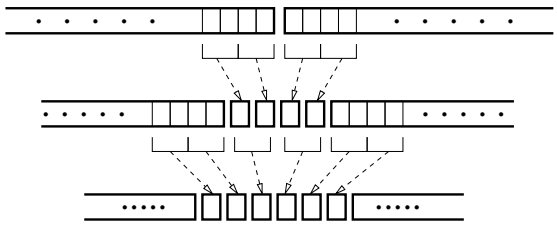

Fig. 1. Illustration of Theorem 1. On top we see the limit between two long chunks of W_0. In this example, the blocking always pairs two symbols. We show below W_0 the 4 bordering blocks formed with the symbols nearby the limit. Below, in W_1, those blocks are converted into 4 explicit chunks (of length 1). This region of 4 symbols is then parsed into 2 blocks. The parsing also creates 4 new bordering blocks from the ends of the long chunks. In W_2, below, we have now a region of 6 explicit chunks. They could have been 8, but we created 2 distinct blocks that reduced their number to 6.

Let n_k be the number of new distinct blocks produced when parsing the regions themselves. Therefore it holds that the number of distinct blocks N_k produced in the kth iteration is at most $4b+n_k$, and the total number of distinct blocks created up to building W_k is $\sum_{i=0}^{k-1} N_i \leq 4bk + \sum_{i=0}^{k-1} n_i$.

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}. Let us call C_k the number of explicit chunks in W_k. Since only the 4 new bordering blocks at each region are converted into explicit chunks, it holds that $C_k \leq 4bk$ for all $k > 0$. Moreover, it holds $C_{k+1} \leq C_k + 4b - n_k$, and thus $0 \leq C_k \leq 4bk - \sum_{i=0}^{k-1} n_i$. Therefore, $\sum_{i=0}^{k-1} n_i \leq 4bk$ 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 $24bk$.

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 grammar 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.

The idea is illustrated in Figure 1. □

With Theorem 1, we can also bound the size z of the Lempel-Ziv parse [23] that allows overlaps. The size without allowing overlaps is known to be bounded by the size of the smallest CFG, $z_{no} \leq g$ [31, 6]. We can easily see that $z \leq 2g_{rl}$ also holds by extending an existing proof [6, Lem. 9] to handle the run-length rules. We call *left-to-right parse* of T any parsing in which each new phrase is a symbol or it occurs previously in T.

Theorem 2. *Let a RLCFG of size g_{rl} expand to a text T. Then the Lempel-Ziv parse (allowing overlaps) of T produces $z \leq 2g_{rl}$ phrases.*

Proof. Consider the parse tree of T, where all internal nodes representing any but the leftmost occurrence of a nonterminal are pruned and left as leaves. The

number of nodes in this tree is precisely g_{rl}. We say that the internal node of nonterminal X is its definition. Our left-to-right parse of T is a sequence $Z[1..z]$ obtained by traversing the leaves of the pruned parse tree left to right. For a terminal leaf, we append the 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.

Rules $X \to Y^t$ are handled as follows. First, we expand them to $X \to Y \cdot Y^{t-1}$, that is, the node for X has two children for Y, and it is 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. 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 $2g_{rl}$ exists, and Lempel-Ziv is the optimal left-to-right parse [23, Thm. 1]. □

By combining Theorems 1 and 2, 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 3. *Let $T[1..n]$ have a bidirectional scheme of size b. Then the Lempel-Ziv parsing of T allowing overlaps has $z = O(b \log(n/b))$ phrases.*

We can also derive upper bounds for g, the size of the smallest CFG, and for z_{no}, the size of the Lempel-Ziv parse that does not allow overlaps. It is sufficient to combine the previous results with the facts that $g = O(z \log(n/z))$ [14, Lem. 8] and $z_{no} \leq g$ [31, 6].

Theorem 4. *Let $T[1..n]$ have a bidirectional scheme of size b. Then there exists a context-free grammar of size $g = O(b \log^2(n/b))$ that generates T.*

Theorem 5. *Let $T[1..n]$ have a bidirectional scheme of size b. Then the Lempel-Ziv parsing of T without allowing overlaps has $z_{no} = O(b \log^2(n/b))$ phrases.*

4 Lower Bounds

In this section we prove that the upper bound of Theorem 3 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 accordingly to the r runs in the BWT, and to show that these phrases induce a valid bidirectional macro 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 3. *Let $p_1, p_2, \ldots, p_r$ be the positions that start runs in the BWT, and let $s_1 < s_2 < \ldots < s_r$ be the corresponding positions in T, $\{SA[p_i], 1 \leq i \leq r\}$, in increasing order. Note that $s_1 = 1$ because $BWT[ISA[1]] = \$$ is a size-1 run,*

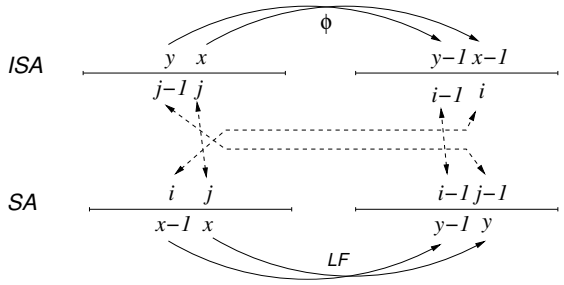

Fig. 2. Illustration of Lemma 2.

and assume $s_{r+1} = n + 1$*, so that* T *is partitioned into* phrases $T[s_i..s_{i+1} - 1]$*. Let also* $\phi(i) = SA[ISA[i] - 1]$ *if* $ISA[i] > 1$ *and* $\phi(i) = SA[n]$ *otherwise. Then we define the* bidirectional scheme of the BWT:

1. *For each* $1 \leq i \leq r$*,* $T[\phi(s_i)..\phi(s_{i+1} - 2)]$ *is copied from* $T[s_i..s_{i+1} - 2]$*.*
2. *For each* $1 \leq i \leq r$*,* $T[\phi(s_{i+1} - 1)]$ *is stored explicitly.*

We build on the following lemma, illustrated in Figure 2.

Lemma 2. *Let* $[j - 1..j]$ *be within a phrase of* T*. Then it holds that* $\phi(j - 1) = \phi(j) - 1$ *and* $T[j - 1] = T[\phi(j) - 1]$*.*

Proof. Consider the pair of positions $T[j - 1..j]$ within a phrase. Let them be pointed from $SA[x] = j$ and $SA[y] = j - 1$, therefore $ISA[j] = x$, $ISA[j - 1] = y$, and $LF(x) = y$. Now, since j is not a position at the beginning of a phrase, x is not the first position in a BWT run. Therefore, $BWT[x - 1] = BWT[x]$, from which it follows that $LF(x-1) = LF(x)-1 = y-1$. Now let $SA[x-1] = i$, that is, $i = \phi(j)$. Then $\phi(j - 1) = SA[ISA[j - 1] - 1] = SA[y - 1] = SA[LF(x - 1)] = SA[x - 1] - 1 = i - 1 = \phi(j) - 1$. It also follows that $T[j - 1] = BWT[x] = BWT[x - 1] = T[i - 1] = T[\phi(j) - 1]$. □

Lemma 3. *The bidirectional scheme of the* BWT *is a valid bidirectional scheme, thus* $2r \geq b$*.*

Proof. By Lemma 2, it holds that $\phi(j-1) = \phi(j)-1$ if $[j-1..j]$ is within a phrase, and that $T[j-1] = T[\phi(j)-1]$. Therefore, we have that $\phi(s_i + k) = \phi(s_i) + k$ for $0 \leq k < s_{i+1} - s_i - 1$, and then $T[\phi(s_i), \ldots, \phi(s_{i+1} - 2)]$ is indeed a contiguous range. We also have that $T[\phi(s_i)..\phi(s_{i+1} - 2)] = T[s_i..s_{i+1} - 2]$, and therefore it is correct to make the copy. Since ϕ is a permutation, every position of T is mentioned exactly once as a target in points 1 and 2.

Finally, it is easy to see that we can recover the whole T from those $2r$ directives. We can, for example, follow the cycle $\phi^k(n)$, $k = 0, \ldots, n - 1$ (note that $T[\phi^0(n)] = T[n]$ is stored explicitly), and copy $T[\phi^k(n)]$ to $T[\phi^{k+1}(n)]$ unless the latter is explicitly stored.

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

We are now ready to obtain the lower bound on bidirectional versus unidirectional parsings.

Theorem 6. *There is an infinite family of strings over an alphabet of size 2 for which* $z = \Omega(b \log n)$.

Proof. Consider the family of the Fibonacci stings, $F_1 = a$, $F_2 = b$, and $F_k = F_{k-1}F_{k-2}$ for all $k > 2$. As observed by Prezza [29, Thm. 25], for F_k we have $r = O(1)$ [26] and $z = \Theta(\log n)$ [10]. By Lemma 3, it also holds that $b = O(1)$, and therefore $z = \Omega(b \log n)$. □

5 Conclusions

We have essentially closed the question of which is the approximation ratio of the (unidirectional) Lempel-Ziv parse 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, show that the gap is in fact wider than what was previously known.

Figure 3 (left) illustrates the known asymptotic bounds that relate the repetitiveness measures we have studied: b, z, z_{no}, g, g_{rl}, and r. We also include e, the size of the CDAWG [4] of T (i.e., the smallest compact automaton that recognizes the substrings of T), which has received some attention recently [2]. It is known that $e \geq \max(z, r)$ [2] and $e = \Omega(g)$ [1].

Figure 3 (right) shows known lower bounds that hold for specific string families. Apart from the lower bounds mentioned in Section 2, there are text families for which $e = \Omega(\max(r, z) \cdot n)$ [2] and thus $e = \Omega(g \cdot n/\log n)$ since $g = O(z \log n)$; and $r = \Omega(g \log n/\log\log n)$ (since on a de Bruijn sequence of order k on a binary alphabet we have $r = \Theta(n)$ [2], $z = O(n/\log n)$, and thus $g = O(z \log(n/z)) = O(n \log\log n/\log n)$). From the upper bounds that hold for every string family, we can also deduce that, for example, there are string families where $r = \Omega(z \log n)$ and thus $r = \Omega(b \log n)$ (since $r = \Omega(z_{no} \log n)$); $\{g, g_{rl}, z_{no}\} = \Omega(r \log n)$ (since $z = \Omega(r \log n)$) and $z = \Omega(b \log n)$ (since $r = \Omega(b)$, Theorem 6). We nevertheless included explicitly the most important of these in the figure.

There are various interesting avenues of future work. For example, it is unknown if r can be more than $O(\log n)$ times larger than z or g. It might also be that our Theorem 1 can be proved without using run-length rules, yielding $g = O(b \log(n/b))$. These are questions of theoretical and also practical relevance, since for example there exist compressed indexes for highly repetitive collections that obtain different search performance depending on which compressibility measure their space is bounded by [27, Sec. 13.2].

Another relevant research avenue is to look for alternatives to Lempel-Ziv compression with a better approximation ratio. For example, a recent bidirectional scheme, *lcpcomp*, seems to always perform better than Lempel-Ziv in practice [9]. It would be interesting to research its approximation ratio with respect to the optimal bidirectional parsing.

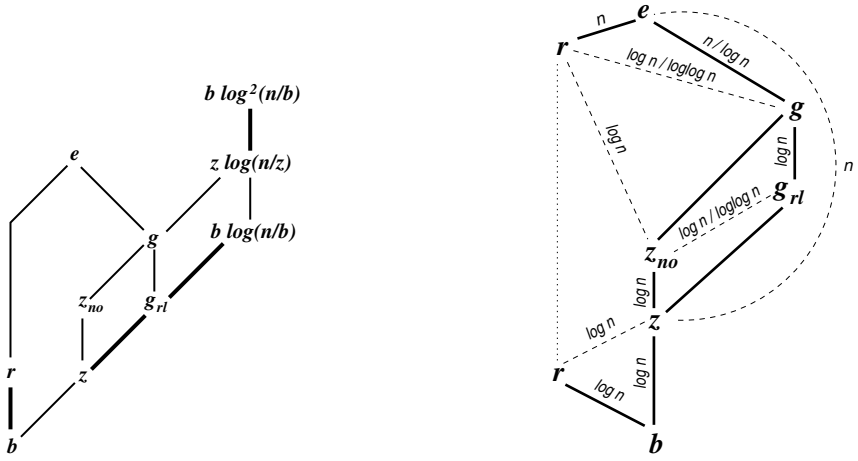

Fig. 3. Known and new asymptotic bounds between repetitiveness measures. The bounds on the left hold for every string family: an edge means that the lower measure is of the order of the upper. The thicker lines were proved in this paper. The dashed lines on the right are lower bounds that hold for some string family. The solid lines are inherited from the left, and since they always hold, they permit propagating the lower bounds. Note that r appears twice.

Acknowledgements

We thank the reviewers for their insightful comments, which helped us improve the presentation significantly.

References

1. D. Belazzougui and F. Cunial. Representing the suffix tree with the CDAWG. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LIPIcs 78, pages 7:1–7:13, 2017.
2. 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.
3. P. Bille, T. Gagie, I. Li Gørtz, and N. Prezza. A separation between run-length SLPs and LZ77. *CoRR*, abs/1711.07270, 2017.
4. A. Blumer, J. Blumer, D. Haussler, R. M. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987.
5. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
6. M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
7. T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.

8. M. Crochemore, C. S. Iliopoulos, M. Kubica, W. Rytter, and T. Waleń. Efficient algorithms for three variants of the LPF table. *Journal of Discrete Algorithms*, 11:51–61, 2012.
9. P. Dinklage, J. Fischer, D. Köppl, M. Löbel, and K. Sadakane. Compression with the tudocomp framework. *CoRR*, abs/1702.07577, 2017.
10. G. Fici. Factorizations of the Fibonacci infinite word. *Journal of Integer Sequences*, 18(9):article 3, 2015.
11. M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, pages 734–740, 2011.
12. T. Gagie. Large alphabets and incompressibility. *Information Processing Letters*, 99(6):246–251, 2006.
13. J. K. Gallant. *String Compression Algorithms*. PhD thesis, Princeton University, 1982.
14. P. Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. *CoRR*, abs/1104.4203, 2011.
15. D. Hucke, M. Lohrey, and C. P. Reh. The smallest grammar problem revisited. In *Proc. 23rd International Symposium on String Processing and Information Retrieval (SPIRE)*, LNCS 9954, pages 35–49, 2016.
16. T. I. Longest common extensions with recompression. In *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, LIPIcs 78, pages 18:1–18:15, 2017.
17. A. Jez. Approximation of grammar-based compression via recompression. *Theoretical Computer Science*, 592:115–134, 2015.
18. A. Jez. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016.
19. J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.
20. J. C. Kieffer and E.-H. Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.
21. A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems on Information Transmission*, 1(1):1–7, 1965.
22. S. Kreft and G. Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013.
23. A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.
24. V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005.
25. U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
26. S. Mantaci, A. Restivo, and M. Sciortino. Burrows-Wheeler transform and Sturmian words. *Information Processing Letters*, 86(5):241–246, 2003.
27. G. Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016.
28. T. Nishimoto, T. I, S. Inenaga, H. Bannai, and M. Takeda. Fully dynamic data structure for LCE queries in compressed space. In *Proc. 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 72:1–72:15, 2016.
29. N. Prezza. *Compressed Computation for Text Indexing*. PhD thesis, University of Udine, 2016.

30. M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *Journal of the ACM*, 28(1):16–24, 1981.
31. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1-3):211–222, 2003.
32. H. Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *Journal of Discrete Algorithms*, 3(24):416–430, 2005.
33. C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:398–403, 1948.
34. Z. D. Sthephens, S. Y. Lee, F. Faghri, R. H. Campbell, Z. Chenxiang, M. J. Efron, R. Iyer, S. Sinha, and G. E. Robinson. Big data: Astronomical or genomical? *PLoS Biology*, 17(7):e1002195, 2015.
35. J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.