Towards a Definitive Measure of Repetitiveness[⋆]

Tomasz Kociumaka[1,⋆⋆][0000−0002−2477−1702], Gonzalo Navarro[2,3,⋆⋆⋆][0000−0002−2286−741X], and Nicola Prezza[4][0000−0003−3553−4953]

[1] Dept. of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl
[2] Millennium Institute for Foundational Research on Data (IMFD), Chile
[3] Dept. of Computer Science, University of Chile, Chile gnavarro@dcc.uchile.cl
[4] Dept. of Business and Management, Luiss Guido Carli, Rome, Italy
nprezza@luiss.it

Abstract. Unlike in statistical compression, where Shannon's entropy is a definitive lower bound, no such clear measure exists for the compressibility of repetitive sequences. Since statistical entropy does not capture repetitiveness, ad-hoc measures like the size z of the Lempel–Ziv parse are frequently used to estimate repetitiveness. Recently, a more principled measure, the size γ of the smallest string *attractor*, was introduced. The measure γ lower bounds all the previous relevant ones (including z), yet length-n strings can be represented and efficiently indexed within space $O(\gamma \log \frac{n}{\gamma})$, which also upper bounds most measures (including z). While γ is certainly a better measure of repetitiveness than z, it is NP-complete to compute, and no $o(\gamma \log n)$-space representation of strings is known.

In this paper, we study a smaller measure, $\delta \leq \gamma$, which can be computed in linear time. We show that δ better captures the compressibility of repetitive strings. For every length n and every value $\delta \geq 2$, we construct a string such that $\gamma = \Omega(\delta \log \frac{n}{\delta})$. Still, we show a representation of any string S in $O(\delta \log \frac{n}{\delta})$ space that supports direct access to any character $S[i]$ in time $O(\log \frac{n}{\delta})$ and finds the *occ* occurrences of any pattern $P[1..m]$ in time $O(m \log n + occ \log^{\varepsilon} n)$ for any constant $\varepsilon > 0$. Further, we prove that no $o(\delta \log n)$-space representation exists: for every length n and every value $2 \leq \delta \leq n^{1-\varepsilon}$, we exhibit a string family whose elements can only be encoded in $\Omega(\delta \log \frac{n}{\delta})$ space. We complete our characterization of δ by showing that, although γ, z, and other repetitiveness measures are always $O(\delta \log \frac{n}{\delta})$, for strings of any length n, the smallest context-free grammar can be of size $\Omega(\delta \log^2 n / \log \log n)$. No such separation is known for γ.

Keywords: Data compression · Lempel–Ziv parse · Repetitive sequences

[⋆] Part of this work was carried out during the Dagstuhl Seminar 19241, "25 Years of the Burrows–Wheeler Transform".

[⋆⋆] Supported by ISF grants no. 1278/16, 824/17, and 1926/19, a BSF grant no. 2018364, and an ERC grant MPM (no. 683064) under the EU's Horizon 2020 Research and Innovation Programme.

[⋆⋆⋆] Supported in part by Fondecyt grant 1-170048, Chile; Millennium Institute for Foundational Research on Data (IMFD), Chile.

1 Introduction

The recent rise in the amount of data we aim to handle [41] is driving research into compressed data representations that can be used directly in compressed form [32]. Interestingly, much of today's fastest-growing data is highly repetitive, which enables space reductions of orders of magnitude [19]: genome collections, versioned text and software repositories, periodic sky surveys, and other sources produce data where each element in the collection is very similar to others.

Since a significant fraction of the data of interest consists of sequences, text indexes are important actors in this research. These are data structures that offer fast pattern matching (and possibly other more sophisticated capabilities) over a collection of strings. Though compressed text indexes are already mature [33] and offer fast pattern searching within space close to the statistical entropy of the string collection, such kind of entropy is unable to capture repetitiveness [28,32]. Achieving orders-of-magnitude space reductions requires instead to resort to other kinds of compressors, such as Lempel–Ziv [29], grammar compression [26], run-length compressed Burrows–Wheeler transform [19], and others. Various compressed indexes build on those methods; see a thorough review [19].

Unlike statistical compression, where Shannon's notion of entropy [40] is a clear lower bound to what compressors can achieve, a similar notion capturing repetitiveness has been elusive. Beyond Kolmogorov's complexity [27], which is uncomputable, repetitiveness is measured in ad-hoc terms, as the results of what specific compressors achieve. A list of such measures on a string $S[1\,..\,n]$ follows:

Lempel–Ziv compression [29] parses S into a sequence of *phrases*, with each phrase defined as the longest string that has appeared previously in S. The associated measure is the number z of phrases produced. The measure can be computed in $O(n)$ time [38].

Bidirectional macro schemes [42] extend Lempel–Ziv so that the source of each phrase may precede or follow it, as long as no circular dependencies are introduced. The associated measure b is the number of phrases of the smallest parsing. It holds $b \le z = O(b \log \frac{n}{b})$ [18], but computing b is NP-complete [20].

Grammar-based compression [26] builds a context-free grammar that generates S and only S. The associated measure is the size g of the smallest grammar (i.e., the total length of the right-hand sides of the rules). It holds $z \le g = O(z \log \frac{n}{z})$ and, while it is NP-complete to compute g, grammars of size $O(z \log \frac{n}{z})$ can be constructed in linear time [39,11,21].

Run-length grammar compression [35] allows in addition rules $A \rightarrow B^t$ (t repetitions of B) of constant size. The measure is the size g_{rl} of the smallest run-length grammar, and it holds $\frac{z}{2} \le g_{rl} \le g$ and $g_{rl} = O(b \log \frac{n}{b})$ [18].

Collage systems [25] extend run-length grammars by allowing truncation: in constant space we can refer to a prefix or a suffix of another nonterminal. The associated measure c satisfies $c \le g_{rl}$ and $c = O(z)$ [31].

Burrows–Wheeler transform (BWT) [10] is a permutation of S that tends to have long runs of equal letters if S is repetitive. The number r of maximal equal-letter runs in the BWT can be found in linear time. It is known that $g_{rl} = O(r \log \frac{n}{r})$ [19] and $\frac{b}{2} \leq r = O(b \log^2 n) = O(z \log^2 n)$ [18,23].

CDAWGs [9] are automata that recognize every substring of S. The associated measure of repetitiveness is e, the size of the smallest such automaton (compressed by dissolving states of in-degree and out-degree one), which is built in linear time [9]. The measure e is always larger than r, g, and z [4,3].

An improvement to this situation is the recent introduction of the concept of string *attractor* [24]. An attractor Γ is a set of positions in S such that any substring of S has an occurrence covering a position in Γ. The size γ of the smallest attractor asymptotically lower bounds all the repetitiveness measures listed above. Recent results [24,34,36,13] show that efficient queries can be supported within $O(\gamma \log \frac{n}{\gamma})$ space[5] and that $g_{rl} = O(\gamma \log \frac{n}{\gamma})$. Previous solutions support random access to S, or indexed searches on S, within space $O(z \log \frac{n}{z})$ [5,6,12,17], $O(g)$ [14,15,16,8,1], $O(g_{rl})$ [19], $O(r)$ or $O(r \log \frac{n}{r})$ [30,4,19], and $O(e)$ [2,3], none improving in general upon the space $O(\gamma \log \frac{n}{\gamma})$ within which one can offer efficient access [24] and indexing [34,13]. Using indexes based on γ is not exempt of problems, however. Computing γ is NP-hard [24], and therefore one has to resort to approximations like z, in which case the representation is only guaranteed to be of size $O(z \log \frac{n}{z})$. While this problem has been recently sidestepped [13], it is still unclear whether γ is the definitive measure of repetitiveness. In particular, it is unknown whether one can always represent S within $O(\gamma)$ space (while this is possible in $O(b)$ space) or even within $o(\gamma \log n)$ space.

Our contributions. In this paper, we study a new measure of repetitiveness, δ, which arguably captures better the concept of compressibility in repetitive strings and is more convenient to deal with. Although this measure was already introduced in a stringology context [37] and used to build indexes of size $O(\gamma \log \frac{n}{\gamma})$ without knowing γ [13], its properties and full potential have not been explored. It always holds that $\delta \leq \gamma$, and δ can be computed in $O(n)$ time [13]. First, we show that δ can be asymptotically strictly smaller than γ: for every length n and every value $\delta \geq 2$, there exist a string such that $\gamma = \Omega(\delta \log \frac{n}{\delta})$. Still, we develop a representation of S of size $O(\delta \log \frac{n}{\delta})$ that allows accessing any character $S[i]$ in time $O(\log \frac{n}{\delta})$ and finds the *occ* occurrences of any pattern $P[1\,..\,m]$ in time $O(m \log n + occ \log^{\varepsilon} n)$ for any constant $\varepsilon > 0$. For this, we reduce the size of block trees [5] to $O(\delta \log \frac{n}{\delta})$. Therefore, we obtain improved space and the same time performance compared to previous results based on γ [24,34,36].[6] Further, we show that, for every length n and every value $2 \leq \delta \leq n^{1-\varepsilon}$ (where $\varepsilon > 0$ is an arbitrary constant), there exists a string family whose elements can only be represented in $\Omega(\delta \log \frac{n}{\delta})$ space. Thus, $o(\delta \log n)$ space is unreachable in general; no such limit is known for γ. We complete our characterization of δ by proving that, although γ, b, z, and c are always $O(\delta \log \frac{n}{\delta})$, the smallest

[5] Throughout the paper, the size of data structures is measured in machine words.

[6] The most recent index [13] locates patterns in $O(m + (occ + 1) \log^{\epsilon} n)$ time and $O(\gamma \log \frac{n}{\gamma})$ space (being thus faster but still using more space).

context-free grammar can be of size $g = \Omega(\delta \log^2 n / \log\log n)$ for strings of any length n. Again, no such lower bound is known to hold on γ.

2 Measure δ

The measure δ has recently been defined by Christiansen et al. [13, Section 5.1], though it is based on the expression $d_k(S)/k$, introduced by Raskhodnikova et al. [37] to approximate z. Below we summarize what is known about it.

Definition 1. *Let $d_k(S)$ be the number of distinct length-k substrings in S. Then*

$$\delta = \max\{d_k(S)/k : k \in [1\,..\,n]\}.$$

Lemma 1 (Based on [37, Lemma 3]). *It always holds that $z = O(\delta \log \frac{n}{\delta})$.*

Proof. Raskhodnikova et al. [37] prove that if $d_\ell(S) \le m \cdot \ell$ for every $\ell \le \ell_0$, then $z \le 4(m \log \ell_0 + \frac{n}{\ell_0})$. Plugging $\ell_0 = \frac{n}{\delta}$ and $m = \delta$, we conclude that $z \le 4(\delta \log \frac{n}{\delta} + \delta) = O(\delta \log \frac{n}{\delta})$. □

Since b, c, and γ are $O(z)$, these three measures are all upper bounded by $O(\delta \log \frac{n}{\delta})$. Additionally, we conclude that $g_{rl} \le g = O(z \log \frac{n}{z}) = O(\delta \log^2 \frac{n}{\delta})$, and note that $r = O(\delta \log^2 n)$ has been proved recently [23].

Before we proceed, let us recall the concept of an attractor.

Definition 2 (Kempa and Prezza [24]). *An* attractor *of a string $S[1\,..\,n]$ is a set of positions $\Gamma \subseteq [1\,..\,n]$ such that every substring $S[i\,..\,j]$ has at least one occurrence $S[i'\,..\,j'] = S[i\,..\,j]$ that covers an attractor position $p \in \Gamma \cap [i'\,..\,j']$.*

Lemma 2 ([13, Lemma 5.6]). *Every string S satisfies $\delta \le \gamma$.*

Proof. Every length-k substring has an occurrence covering an attractor position, so there can be at most $k\gamma$ distinct substrings, i.e., $d_k(S)/k \le \gamma$ for all $k \le n$. □

Lemma 3 ([13, Lemma 5.7]). *The measure δ can be computed in $O(n)$ time and space given $S[1\,..\,n]$.*

Proof. One can use the suffix tree or the LCP table of S to retrieve $d_k(S)$ for all $k \in [1\,..\,n]$ in $O(n)$ time, and then compute δ from this information. □

3 Lower Bounds in Terms of δ

In this section, we prove lower bounds in terms of the measure δ. First, we show that there exist string families where $\delta = o(\gamma)$; in fact, δ can be smaller by up to a logarithmic factor. Second, we prove that there are string families that cannot be encoded in $o(\delta \log n)$ space: for every length n and every value $2 \le \delta \le n^{1-\varepsilon}$ (where $\varepsilon > 0$ is an arbitrary constant), there is a string family whose elements require $\Omega(\delta \log \frac{n}{\delta})$ space to represent. Third, although in the next section we give an $O(\delta \log \frac{n}{\delta})$-space representation, below we construct a family of strings which cannot be represented using context-free grammars of size $O(\delta \log \frac{n}{\delta})$; a nearly logarithmic-factor separation exists.

3.1 Lower bounds on attractors

Consider an infinite string $S_\infty[1\,..]$, where $S_\infty[i] = \mathtt{b}$ if $i = 2^j$ for some integer $j \geq 0$, and $S_\infty[i] = \mathtt{a}$ otherwise. For $n \geq 1$, let S_n be the length-n prefix of S. We shall prove that the strings in this family satisfy $\delta = O(1)$ and $\gamma = \Omega(\log n)$.

Lemma 4. *For every $n \geq 1$, the string S_n satisfies $\delta \leq 2$ and $\gamma \geq \frac{1}{2}\lfloor \log n \rfloor$.*

Proof. For each $j \geq 1$, every pair of consecutive $\mathtt{b}$s in $S_\infty[2^{j-1}+1\,..]$ is at distance at least 2^j. Therefore, the only distinct substrings of length $k \leq 2^j$ in $S_\infty[2^{j-1}+1\,..]$ are of the form $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}\mathtt{a}^{k-i-1}$ for $i \in [0\,..\,k-1]$. Hence, the distinct length-k substrings of S_∞ are those starting up to position 2^{j-1}, $S_\infty[i\,..\,i+k-1]$ for $i \in [1\,..\,2^{j-1}]$, and the $k+1$ already mentioned strings, for a total of $d_k(S_\infty) \leq 2^{j-1}+k+1$. Plugging $j = \lceil \log k \rceil$, we get $d_k(S_\infty) \leq 2^{\lceil \log k \rceil - 1} + k + 1 \leq 2^{\log k} + k \leq 2k$, concluding that $\delta(S_n) \leq 2$ holds for every n.

Next, observe that for each $j \geq 0$, the substring $\mathtt{b}\mathtt{a}^{2^j-1}\mathtt{b}$ has its unique occurrence in S_∞ at $S_\infty[2^j\,..\,2^{j+1}]$. The covered regions are disjoint across *even* integers j, so each one requires a distinct attractor element. Consequently, $\gamma(S_n) \geq \frac{j}{2}$ for $n \geq 2^j$. Plugging $j = \lfloor \log n \rfloor$, we get $\gamma(S_n) \geq \frac{1}{2}\lfloor \log n \rfloor$. □

We can also show that there are strings with $\delta = o(\gamma)$ as long as $2 \leq \delta \leq o(n)$.

Theorem 1. *For every length n and value $\delta \in [2\,..\,n]$, there is a string $S[1\,..\,n]$ with $\gamma = \Omega(\delta \log \frac{n}{\delta})$.*

Proof. Let us first fix an integer $m \geq 1$ such that $n \geq 4m-1$ and decompose $n - m + 1 = \sum_{i=1}^m n_i$ roughly equally (so that $n_i \geq 3$ and $n_i = \Omega(\frac{n}{m})$). We shall build a string S over an alphabet consisting of $3m-1$ characters: $\mathtt{a}_i$ and $\mathtt{b}_i$ for $i \in [1\,..\,m]$ and $\$_i$ for $i \in [1\,..\,m-1]$. For this, we take $S^{(i)}$ to be the string S_{n_i} built for Lemma 4, with alphabet $\{\mathtt{a}, \mathtt{b}\}$ replaced by $\{\mathtt{a}_i, \mathtt{b}_i\}$, and we define S to be the concatenation of the strings $S^{(i)}$ interleaved with sentinels $\$_i$.

Notice that, for each k, we have $d_k(S) \leq (m-1)k + \sum_{i=1}^m d_k(S^{(i)})$ because every substring contains $\$_i$ or is contained in $S^{(i)}$ for some i. Hence, $\delta(S) \leq 3m-1$. (In fact, $\delta(S) = 3m-1$ because $d_1(S) = 3m-1$.) Furthermore, $\gamma(S) \geq \sum_{i=1}^m \gamma(S^{(i)}) = \Omega(m \log \frac{n}{m}) = \Omega(\delta \log \frac{n}{\delta})$ since the alphabets of $S^{(i)}$ are disjoint.

This construction proves the theorem for $\delta = 3m-1$ and $n \geq 4m-1$. If $\delta \bmod 3 \neq 2$, we pad the string with $O(1)$ additional sentinels. Each one increases $\delta(S)$, $\gamma(S)$, and n by 1. Finally, we note that the claim for $\delta = \Omega(n)$ reduces to $\gamma = \Omega(\delta)$, and the latter relation follows directly from Lemma 2. □

3.2 Lower bounds on text entropy and grammar size

We now show that there are string families that cannot be encoded in $o(\delta \log n)$ space, that is, $o(\delta \log^2 n)$ bits. It is not known if the same occurs with γ.

Consider a family $\mathcal{S}^*$ consisting of variants of the infinite string S_∞ constructed in the previous section, where the positions of $\mathtt{b}$s are further apart and slightly perturbed. More specifically, for each $S \in \mathcal{S}^*$, the first $\mathtt{b}$ is placed at position

$S[1]$ and then, for $j \geq 2$, the jth `b` is placed anywhere in $S[2 \cdot 4^{j-2}+1\,..\,4^{j-1}]$. The family $\mathcal{S}^*_n$ consists of length-n prefixes of the infinite strings of the family $\mathcal{S}^*$.

Lemma 5. *For every $n \geq 1$, the family $\mathcal{S}^*_n$ needs $\Omega(\log^2 n)$ bits to be encoded.*

Proof. In our definition of $\mathcal{S}^*$, the location of the jth `b` can be chosen among $2 \cdot 4^{j-2}$ positions, and each combination of these choices generates a different string in $\mathcal{S}^*_n$ as long as $n \geq 4^{j-1}$. Hence, $|\mathcal{S}^*_n| = \prod_{j=2}^{i+1} 2 \cdot 4^{j-2} = 2^{\Omega(i^2)}$ for $n \geq 4^i$. To distinguish strings in $\mathcal{S}^*_n$, any encoding needs $\log|\mathcal{S}^*_n| = \Omega(\log^2 n)$ bits. □

Theorem 2. *For every length n and value $\delta \in [2\,..\,n]$, there exists a family of length-n strings of common measure δ that needs $\Omega(\delta \log^2 \frac{n}{\delta})$ bits to be encoded.*

Proof. By Lemma 5, encoding $\mathcal{S}^*_n$ requires $\Omega(\log^2 n)$ bits. Below, we prove that the measure δ for any string in $\mathcal{S}^*_n$ is at most 2. Starting from position $4^{j-1}+1$, the distances between two consecutive `b`s are at least 4^j. Therefore, the distinct substrings of length $k \leq 4^j$ are either those that start at position $i \in [1\,..\,4^{j-1}]$ or those of the form $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}\mathtt{a}^{k-i-1}$ for $i \in [0\,..\,k-1]$, which yields a total of $d_k(S) \leq 4^{j-1}+k+1$. Plugging $j = \lceil \frac{1}{2}\log k \rceil$, we get $d_k(S) \leq 4^{\lceil \frac{1}{2}\log k\rceil -1}+k+1 \leq 4^{\frac{1}{2}\log k}+k \leq 2k$. By definition of δ, we conclude that $\delta(S) \leq 2$ for every $S \in \mathcal{S}^*_n$.

As in the proof of Theorem 1, one can generalize this result to larger δ. □

The family $\mathcal{S}^*_n$ also gives strings that do not satisfy $g = O(\delta \log n)$.

Theorem 3. *For every length n, there is a string with $g = \Omega(\delta \log^2 n / \log\log n)$.*

Proof. Consider the same family $\mathcal{S}^*_n$, which needs $\Omega(\log^2 n)$ bits to be represented. If we could encode it with a grammar of size g, each grammar element would be a nonterminal that could be encoded with $O(\log g)$ bits. Therefore, our grammar representation would require $O(g \log g)$ bits. Since this must be $\Omega(\log^2 n)$, it follows that $g = \Omega(\log^2 n/\log\log n)$ for any grammar of size g encoding $\mathcal{S}^*_n$. Since $\delta = O(1)$ for every string $S \in \mathcal{S}^*_n$, it follows that $g = \Omega(\delta \log^2 n/\log\log n)$. □

4 Block Trees in δ-Bounded Space

The block tree [5] is a data structure designed to represent repetitive strings $S[1\,..\,n]$ in $O(z \log \frac{n}{z})$ space while offering efficient access. In this section, we show that the block tree is easily tuned to use $O(\delta \log \frac{n}{\delta})$ space while retaining its functionality. Note that, given the lower bounds of Section 3, we cannot hope for a representation of size $o(\delta \log \frac{n}{\delta})$.

4.1 Block trees

Given integer parameters r and s, the root of the block tree divides S into s equal-sized (that is, with the same number of characters) blocks (assume for simplicity that $n = s \cdot r^t$ for some integer t).[7] Blocks are then classified into *marked* and *unmarked*. If two adjacent blocks B_1, B_2 form the leftmost occurrence of the underlying substring B_1B_2, then both B_1 and B_2 are marked. Blocks B that remain unmarked are replaced by a pointer to the pair of adjacent blocks B_1, B_2 that contains the leftmost occurrence of B, and the offset $\epsilon \geq 0$ where B starts inside B_1. Marked blocks are divided into r equal-sized sub-blocks, which form the children of the current block tree's level, and processed similarly in a recursive fashion. Let σ be the alphabet size. The level where the blocks become of length below $\log_\sigma n$ corresponds to the leaves of the block tree, and its blocks store their plain string content using $O(\log n)$ bits. The height of the block tree is then $h = O(\log_r \frac{n/s}{\log_\sigma n}) = O(\log_r \frac{n \log \sigma}{s \log n}) \subseteq O(\log \frac{n}{s})$.

The block tree construction guarantees that the blocks B_1 and B_2 to which any unmarked block points exist and are marked. Therefore, any access to a position $S[i]$ can be carried out in $O(h)$ time, by descending from the root to a leaf and spending $O(1)$ time in each level: To obtain $B[i]$ from a marked block B, we simply compute to which sub-block $B[i]$ belongs among the children of B. To obtain $B[i]$ from an unmarked block B pointing to B_1, B_2 with offset ϵ, we switch either to $B_1[\epsilon + i]$ or to $B_2[\epsilon + i - |B_1|]$, which are marked blocks.

By storing further data associated with marked and unmarked blocks, the block tree offers the following functionality [5]:

Access: any substring $S[i\,..\,i+\ell-1]$ is extracted in time $O(h\lceil \ell/\log_\sigma n\rceil)$.

Rank: $rank_a(S,i)$ is the number of times symbol a occurs in $S[1\,..\,i]$. It is computed in time $O(h)$ by multiplying the space by $O(\sigma)$.

Select: $select_a(S,j)$ is the position of the jth occurrence of symbol a in S. It is computed in time $O(\log\log \frac{n}{s} + h \log\log r)$ by multiplying the space by $O(\sigma)$.

It is shown that there are only $O(zr)$ blocks in each level of the block tree (except the first, which has s); therefore its size is $O(s + zr \log_r \frac{n \log \sigma}{s \log n})$.

4.2 Bounding the space in terms of δ

We now prove that there are only $O(\delta r)$ blocks in each level of the block tree, and therefore, choosing $s = \delta$ yields a structure of size $O(\delta r \log_r \frac{n \log \sigma}{\delta \log n})$ with height $O(\log_r \frac{n \log \sigma}{\delta \log n})$. For $r = O(1)$, the space is $O(\delta \log \frac{n}{\delta})$ and the height is $O(\log \frac{n}{\delta})$.

Let us call level k of the block tree the one where blocks are of length r^k. In level k, then, S is covered regularly with blocks $B = S[r^k(i-1)+1\,..\,r^k i]$ of length r^k (though not all of them are present in the block tree). Note that k reaches its maximum in the root (where we have the largest blocks) and the minimum in the leaves of the block tree.

[7] If not, we simply pad S with spurious symbols at the end; whole spurious blocks are not represented. The extra space incurred is only $O(rh)$ for a block tree of height h. The actual construction [5] uses instead blocks of sizes $\lfloor n/s \rfloor$ and $\lceil n/s \rceil$.

Lemma 6. *The number of marked blocks of length r^k in the block tree is $O(\delta)$.*

Proof. Any marked block B must belong to a sequence of three blocks, $B^- \cdot B \cdot B^+$, such that B is inside the leftmost occurrence of $B^- \cdot B$ or $B \cdot B^+$, or both (B^- and B^+ do not exist for the first and last block, respectively).

For the sake of computing our bound, let # be a symbol not appearing in S and let us add $2 \cdot r^k$ characters equal to # at the beginning of S and r^k characters equal to # at the end of S. We index the added prefix in negative positions (up to index 0), so that $S[-2 \cdot r^k + 1\,..\,0] = \#^{2 \cdot r^k}$. Now consider all the r^k text positions p belonging to a marked block B. The long substring $E = S[p - 2 \cdot r^k\,..\,p + 2 \cdot r^k - 1]$ centered at p, of length $4r^k$, contains $B^- \cdot B \cdot B^+$, and thus E contains the leftmost occurrence L of $B^- \cdot B$ or $B \cdot B^+$. All those long substrings E must then be distinct: if two long substrings E and E' are equal, and E' appears after E in S, then E' does not contain the leftmost occurrence of any substring L.

Since we added a prefix of length $2 \cdot r^k$ and a suffix of length r^k consisting of character # to S, the number of distinct substrings of length $4r^k$ is at most $d_{4r^k}(S) + 3r^k$. Therefore, there can be at most $d_{4r^k}(S) + 3r^k$ long substrings E as well, because they must all be distinct. Since each position p inside a block B induces a distinct long substring E, and each marked block B contributes r^k distinct positions p, there are at most $(d_{4r^k}(S) + 3r^k)/r^k$ marked blocks B of length r^k. The total number of marked blocks of length r^k is thus at most $(d_{4r^k}(S) + 3r^k)/r^k = 4 \cdot d_{4r^k}(S)/(4r^k) + 3r^k/r^k \le 4\delta + 3$. □

Since the block tree has at most $4\delta + 3$ marked blocks per level, it has $O(\delta r)$ blocks across all the levels except the first. This yields the following result.

Theorem 4. *Let $S[1\,..\,n]$, over alphabet $[1\,..\,\sigma]$, have compressibility measure δ. Then the block tree of S, with parameters r and s, is of size $O(s + \delta r \log_r \frac{n \log \sigma}{s \log n})$ words and height $h = O(\log_r \frac{n \log \sigma}{s \log n})$.*

Note that $\frac{n}{\delta} = O(\frac{n}{z} \log \frac{n}{\delta}) = O(\frac{n}{z}\sqrt{\frac{n}{\delta}})$ due to Lemma 1, so $\log \frac{n}{\delta} = O(\log \frac{n^2}{z^2})$ $= O(\log \frac{n}{z}) = O(\log \frac{n}{\gamma})$. Hence, the query time we obtain using $O(\delta \log \frac{n}{\delta})$ space is asymptotically the same as the $O(\log \frac{n}{\gamma})$ time obtained in $O(\gamma \log \frac{n}{\gamma})$ space [34,36] or the $O(\log \frac{n}{z})$ time obtained in $O(z \log \frac{n}{z})$ space [5].

5 Text Indexing in δ-Bounded Space

We now show that not only efficient access of S can be supported within $O(\delta \log \frac{n}{\delta})$ space, but also text indexing, that is, efficiently listing all the positions in S where a pattern $P[1\,..\,m]$ appears. For consistency with previous works, in this section we speak of a text $T[1\,..\,n]$ instead of a string $S[1\,..\,n]$.

Our index builds on top of a slight variant of the block tree of the previous sections, with $r = 2$, $s = \delta$, and stopping only when the leaves are of length 1. This block tree is of size $O(\delta \log \frac{n}{\delta})$ and of height $O(\log \frac{n}{\delta})$.

To build the index, we follow the same ideas of the "universal index" [34], whose space will be improved without affecting its search time complexities. That index builds on a variant of block trees designed for attractors: the Γ-tree has a first level with γ equal-sized blocks, and at any other level k, it marks the blocks that are at distance $< 2^k$ from an attractor position. Unmarked blocks B then point to some copy of B that crosses an attractor position (the blocks overlapping that copy are marked by definition). In the Γ-tree pointers can go leftward or rightward, not necessarily to a leftmost occurrence. The space of the Γ-tree is $\Theta(\gamma \log \frac{n}{\gamma})$, which we now know, by Theorem 4, that is never asymptotically smaller than that of block trees with parameters $r = 2$ and $s = \delta$.

Karp–Rabin fingerprinting [22] assigns a string $S[1\,..\,\ell]$ the signature $\kappa(S) = (\sum_{i=1}^{\ell} S[i] \cdot c^{i-1}) \bmod \mu$ for suitable integers $c > 1$ and prime μ. It is possible to build a signature formed by a pair of functions $\langle \kappa_1, \kappa_2 \rangle$ guaranteeing no collisions between substrings of $S[1\,..\,n]$, in $O(n \log n)$ expected time [7]. Our index will need to compute Karp–Rabin fingerprints $\kappa(T[i\,..\,j])$ in time $O(\log \frac{n}{\delta})$. This is done on block trees by using the same algorithm described for the Γ-tree.

Lemma 7. *Let $T[1\,..\,n]$ have compressibility measure δ, and let κ be a Karp–Rabin function. Then we can store a data structure of size $O(\delta \log \frac{n}{\delta})$ supporting the computation of κ on any substring of T in $O(\log \frac{n}{\delta})$ time.*

Proof. The structure is the described block tree variant, with some further fields. We store $\kappa(T[1\,..\,2^k i])$ at the ith top-level block, for all i and $k = \lceil \log \frac{n}{\delta} \rceil$. We also store $\kappa(B)$ for each block B stored in the tree and, for the unmarked blocks B pointing to B_1, B_2 with offset ϵ, we also store $\kappa(B_1[1 + \epsilon\,..])$. Navarro and Prezza [34, Lem. 1] show that this suffices to compute $\kappa(T[i\,..\,j])$ within $O(1)$ time per level of the Γ-tree; their proof holds verbatim for the block tree. $\square$

Let us say that a block is *explicit* if it is stored in the block tree. Thus, a block is explicit if and only if it is marked or it is the child of a marked block.

Lemma 8 (See [34, Lem. 2]). *Any substring $T[i\,..\,j]$ of length at least 2 either overlaps two consecutive explicit blocks or is completely inside an unmarked block.*

Proof. The leaves of the block tree, read left to right, partition T into a sequence of explicit blocks. The leaves are either unmarked blocks or blocks of length 1. Since $|T[i\,..\,j]| \geq 2$, if it is not completely inside an unmarked block, it cannot be contained in a leaf, so it must cross a boundary between two explicit blocks. $\square$

We now divide the possible occurrences of $P[1\,..\,m]$ in T into *primary* (those overlapping two consecutive explicit blocks) and *secondary* (those inside an unmarked block). The technique used on Γ-trees [34, Sec. 3] applies verbatim here: Primary occurrences are found using a grid of $(s-1) \times (s-1)$, where $s = O(\delta \log \frac{n}{\delta})$ is the number of leaves in the block tree, which finds the occ_p primary occurrences in time $O((m + occ_p) \log^{\varepsilon} s)$, for any constant $\varepsilon > 0$. The ranges to search in the grid are obtained using their following result [34, Lem. 3].

Lemma 9. *Let $\mathcal{X}$ be a sorted set of suffixes of T, and κ a Karp–Rabin function. If one can extract a substring of length ℓ from T in time $f_e(\ell)$ and compute κ on* *it in time $f_h(\ell)$, then one can build a data structure of size $O(|\mathcal{X}|)$ that obtains the lexicographic ranges in $\mathcal{X}$ of the $m-1$ suffixes of a given pattern P in worst-case time $O(m(f_h(m)+\log m)+f_e(m))$, provided that κ is collision-free among substrings of T whose lengths are powers of two.*

Since in our case $f_e(m) = O(m \log \frac{n}{\delta})$ and $f_h(m) = O(\log \frac{n}{\delta})$, we can find all the ranges to search for in time $O(m \log \frac{nm}{\delta})$. The occ_s secondary occurrences are obtained as on Γ-trees [34, Sec. 3.2], within $O((occ_p + occ_s) \log\log \frac{n}{\delta})$ time.

Theorem 5. *Let $T[1\,..\,n]$ have measure δ. Then there exists a data structure of size $O(\delta \log \frac{n}{\delta})$ such that the occurrences of any pattern $P[1\,..\,m]$ in T can be located in time $O(m \log n + occ \log^{\varepsilon} n)$, for any constant $\varepsilon > 0$.*

6 Conclusions

We have made a step towards establishing the right measure of repetitiveness for a string $S[1\,..\,n]$. Compared with the most principled prior measure, the size γ of the smallest attractor, the proposed measure δ has several important advantages:

1. It lower bounds the previous measure, $\delta \le \gamma$, and can be computed in linear time, while finding γ is NP-hard.
2. We can always encode S in $O(\delta \log \frac{n}{\delta})$ space, and this is worst-case optimal in terms of δ: for any length n and any value $2 \le \delta \le n^{1-\varepsilon}$ (where $\varepsilon > 0$ is an arbitrary constant), there are text families needing $\Omega(\delta \log \frac{n}{\delta})$ space. Thus, $o(\delta \log n)$ space is unreachable. Instead, no text family is known to require $\omega(\gamma)$ space, nor it is known if $o(\gamma \log n)$ space can be reached.
3. Measures γ, b, c, and z are upper bounded by $O(\delta \log \frac{n}{\delta})$, and $g = O(\delta \log^2 \frac{n}{\delta})$, but there are text families where the smallest context-free grammar is of size $g = \Omega(\delta \log^2 n / \log\log n)$. This lower bound is not known to hold on γ.
4. The encodings using $O(\delta \log \frac{n}{\delta})$ space support direct access and indexed searches, with the same complexities obtained within attractor-bounded space, $O(\gamma \log \frac{n}{\gamma})$. An exception is a very recent faster index [13].

An ideal compressibility measure for repetitive sequences should be always reachable and string-wise optimal, apart from being practical to compute. Measure $\delta \log \frac{n}{\delta}$ is reachable and fast to compute, though optimal only in a coarse sense (i.e., not string-wise but within the class of all the strings with the same δ value).

Note that we do not know if one can always encode a string within $O(\gamma)$ space. If this was the case, then γ would be a better measure than $\delta \log \frac{n}{\delta}$, except for being hard to compute. Otherwise, a good alternative could be b, which is always reachable and might be string-wise optimal within some broad class of representations that exploit repetitiveness, yet NP-hard to compute. It is not known, however, if b or γ are monotone, that is, smaller on T than on TT', whereas δ clearly is. This fascinating quest is then still open.

On the more practical side, it would be interesting to obtain faster indexes of size $O(\delta \log \frac{n}{\delta})$. Our index requires $O(m \log n + occ \log^{\epsilon} n)$ search time, while in $O(\gamma \log \frac{n}{\gamma})$ space, it is possible to search in $O(m + (occ + 1) \log^{\epsilon} n)$ time [13].

References

1. Belazzougui, D., Cording, P.H., Puglisi, S.J., Tabei, Y.: Access, rank, and select in grammar-compressed strings. In: Proc. 23rd ESA. pp. 142–154 (2015). https://doi.org/10.1007/978-3-662-48350-3_13
2. Belazzougui, D., Cunial, F.: Fast label extraction in the CDAWG. In: Proc. 24th SPIRE. pp. 161–175 (2017). https://doi.org/10.1007/978-3-319-67428-5_14
3. Belazzougui, D., Cunial, F.: Representing the suffix tree with the CDAWG. In: Proc. 28th CPM. pp. 7:1–7:13 (2017). https://doi.org/10.4230/LIPIcs.CPM.2017.7
4. Belazzougui, D., Cunial, F., Gagie, T., Prezza, N., Raffinot, M.: Composite repetition-aware data structures. In: Proc. 26th CPM. pp. 26–39. Springer (2015). https://doi.org/10.1007/978-3-319-19929-0_3
5. Belazzougui, D., Gagie, T., Gawrychowski, P., Kärkkäinen, J., Pereira, A.O., Puglisi, S.J., Tabei, Y.: Queries on LZ-bounded encodings. In: Proc. 25th DCC. pp. 83–92 (2015). https://doi.org/10.1109/DCC.2015.69
6. Bille, P., Ettienne, M.B., Gørtz, I.L., Vildhøj, H.W.: Time-space trade-offs for Lempel-Ziv compressed indexing. Theor. Comput. Sci. **713**, 66–77 (2018). https://doi.org/10.1016/j.tcs.2017.12.021
7. Bille, P., Gørtz, I.L., Sach, B., Vildhøj, H.W.: Time-space trade-offs for longest common extensions. J. Discrete Algorithms **25**, 42–50 (2014). https://doi.org/10.1016/j.jda.2013.06.003
8. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings and trees. SIAM J. Comput. **44**(3), 513–539 (2015). https://doi.org/10.1137/130936889
9. Blumer, A., Blumer, J., Haussler, D., McConnell, R.M., Ehrenfeucht, A.: Complete inverted files for efficient text retrieval and analysis. J. ACM **34**(3), 578–595 (1987). https://doi.org/10.1145/28869.28873
10. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. Tech. Rep. 124, Digital Equipment Corporation (1994), `https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf`
11. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Trans. Inf. Theory **51**(7), 2554–2576 (2005). https://doi.org/10.1109/TIT.2005.850116
12. Christiansen, A.R., Ettienne, M.B.: Compressed indexing with signature grammars. In: Proc. 13th LATIN. pp. 331–345 (2018). https://doi.org/10.1007/978-3-319-77404-6_25
13. Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes (2019), `https://arxiv.org/abs/1811.12779`
14. Claude, F., Navarro, G.: Self-indexed grammar-based compression. Fundam. Inform. **111**(3), 313–337 (2011). https://doi.org/10.3233/FI-2011-565
15. Claude, F., Navarro, G.: Improved grammar-based compressed indexes. In: Proc. 19th SPIRE. pp. 180–192 (2012). https://doi.org/10.1007/978-3-642-34109-0_19
16. Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: A faster grammar-based self-index. In: Proc. 6th LATA. pp. 240–251 (2012). https://doi.org/10.1007/978-3-642-28332-1_21

17. Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: LZ77-based self-indexing with faster pattern matching. In: Proc. 11th LATIN. pp. 731–742. Springer (2014). https://doi.org/10.1007/978-3-642-54423-1_63
18. Gagie, T., Navarro, G., Prezza, N.: On the approximation ratio of Lempel-Ziv parsing. In: Proc. 13th LATIN. pp. 490–503 (2018). https://doi.org/10.1007/978-3-319-77404-6_36
19. Gagie, T., Navarro, G., Prezza, N.: Fully-functional suffix trees and optimal text searching in BWT-runs bounded space. J. ACM **67**(1), 1–54 (2020). https://doi.org/10.1145/3375890
20. Gallant, J.K.: String Compression Algorithms. Ph.D. thesis, Princeton Univ. (1982)
21. Je, A.: A really simple approximation of smallest grammar. Theor. Comput. Sci. **616**, 141–150 (2016). https://doi.org/10.1016/j.tcs.2015.12.032
22. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. **31**(2), 249–260 (1987). https://doi.org/10.1147/rd.312.0249
23. Kempa, D., Kociumaka, T.: Resolution of the Burrows-Wheeler transform conjecture (2019), `https://arxiv.org/abs/1910.10631`
24. Kempa, D., Prezza, N.: At the roots of dictionary compression: String attractors. In: Proc. 50th STOC. pp. 827–840 (2018). https://doi.org/10.1145/3188745.3188814
25. Kida, T., Matsumoto, T., Shibata, Y., Takeda, M., Shinohara, A., Arikawa, S.: Collage system: A unifying framework for compressed pattern matching. Theor. Comput. Sci. **298**(1), 253–272 (2003). https://doi.org/10.1016/S0304-3975(02)00426-7
26. Kieffer, J.C., Yang, E.: Grammar-based codes: A new class of universal lossless source codes. IEEE Trans. Inf. Theory **46**(3), 737–754 (2000). https://doi.org/10.1109/18.841160
27. Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Int. J. Comput. Math. **2**(1-4), 157–168 (1968). https://doi.org/10.1080/00207166808803030
28. Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theor. Comput. Sci. **483**, 115–133 (2013). https://doi.org/10.1016/j.tcs.2012.02.006
29. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Trans. Inf. Theory **22**(1), 75–81 (1976). https://doi.org/10.1109/TIT.1976.1055501
30. Mäkinen, V., Navarro, G., Sirén, J., Välimäki, N.: Storage and retrieval of highly repetitive sequence collections. J. Comput. Biol. **17**(3), 281–308 (2010). https://doi.org/10.1089/cmb.2009.0169
31. Navarro, G., Ochoa, C., Prezza, N.: On the approximation ratio of ordered parsings (2019), `https://arxiv.org/abs/1803.09517`
32. Navarro, G.: Compact Data Structures – A practical approach. Cambridge University Press (2016). https://doi.org/10.1017/cbo9781316588284
33. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys **39**(1) (2007). https://doi.org/10.1145/1216370.1216372
34. Navarro, G., Prezza, N.: Universal compressed text indexing. Theor. Comput. Sci. **762**, 41–50 (2019). https://doi.org/10.1016/j.tcs.2018.09.007
35. Nishimoto, T., I, T., Inenaga, S., Bannai, H., Takeda, M.: Fully dynamic data structure for LCE queries in compressed space. In: Proc. 41st MFCS. pp. 72:1–72:15 (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.72
36. Prezza, N.: Optimal rank and select queries on dictionary-compressed text. In: Proc. 30th CPM. pp. 4:1–4:12 (2019). https://doi.org/10.4230/LIPIcs.CPM.2019.4
37. Raskhodnikova, S., Ron, D., Rubinfeld, R., Smith, A.D.: Sublinear algorithms for approximating string compressibility. Algorithmica **65**(3), 685–709 (2013). https://doi.org/10.1007/s00453-012-9618-6

38. Rodeh, M., Pratt, V.R., Even, S.: Linear algorithm for data compression via string matching. J. ACM **28**(1), 16–24 (1981). https://doi.org/10.1145/322234.322237
39. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theor. Comput. Sci. **302**(1-3), 211–222 (2003). https://doi.org/10.1016/S0304-3975(02)00777-6
40. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 398–403 (1948). https://doi.org/10.1002/j.1538-7305.1948.tb01338.x
41. Stephens, Z.D., Lee, S.Y., Faghri, F., Campbell, R.H., Zhai, C., Efron, M.J., Iyer, R., Schatz, M.C., Sinha, S., Robinson, G.E.: Big data: Astronomical or genomical? PLOS Biology **13**(7), e1002195 (2015). https://doi.org/10.1371/journal.pbio.1002195
42. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. J. ACM **29**(4), 928–951 (1982). https://doi.org/10.1145/322344.322346