Toward a Definitive Compressibility Measure for Repetitive Sequences

Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza

***Abstract*—While the kth order empirical entropy is an accepted measure of the compressibility of individual sequences on classical text collections, it is useful only for small values of k and thus fails to capture the compressibility of repetitive sequences. In the absence of an established way of quantifying the latter, ad-hoc measures like the size z of the Lempel–Ziv parse are frequently used to estimate repetitiveness. The size $b \le z$ of the smallest bidirectional macro scheme captures better what can be achieved via copy-paste processes, though it is NP-complete to compute, and it is not monotone upon appending symbols. Recently, a more principled measure, the size γ of the smallest *string attractor*, was introduced. The measure $\gamma \le b$ lower-bounds all the previous relevant ones, while length-n strings can be represented and efficiently indexed within space $O(\gamma \log \frac{n}{\gamma})$, which also upper-bounds many measures, including z. Although γ is arguably a better measure of repetitiveness than b, it is also NP-complete to compute and not monotone, and it is unknown if one can represent all strings in $o(\gamma \log n)$ space.**

In this paper, we study an even smaller measure, $\delta \le \gamma$, which can be computed in linear time, is monotone, and allows encoding every string in $O(\delta \log \frac{n}{\delta})$ space because $z = O(\delta \log \frac{n}{\delta})$. We argue that δ better captures the compressibility of repetitive strings. Concretely, we show that (1) δ can be strictly smaller than γ, by up to a logarithmic factor; (2) there are string families needing $\Omega(\delta \log \frac{n}{\delta})$ space to be encoded, so this space is optimal for every n and δ; (3) one can build run-length context-free grammars of size $O(\delta \log \frac{n}{\delta})$, whereas the smallest (non-run-length) grammar can be up to $\Theta(\log n / \log\log n)$ times larger; and (4) within $O(\delta \log \frac{n}{\delta})$ space, we can not only represent a string but also offer logarithmic-time access to its symbols, computation of substring fingerprints, and efficient indexed searches for pattern occurrences. We further refine the above results to account for the alphabet size σ of the string, showing that $\Theta(\delta \log \frac{n \log \sigma}{\delta \log n})$ space is necessary and sufficient to represent the string and to efficiently support access, fingerprinting, and pattern matching queries.

***Index Terms*—Data compression; Lempel–Ziv parse; Repetitive sequences; String attractors; Substring complexity**

I. Introduction

The recent rise in the amount of data we aim to handle is driving research into studying compressed data representations that can be used directly in compressed form [2]. Interestingly, much of today's fastest-growing data is highly repetitive: 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, compression of highly repetitive text collections is gaining attention as it enables space reductions of orders of magnitude [3].

On classical text collections, the kth order empirical entropy is an established measure of compressibility, lower-bounding the space that any kth order statistical compressor applied on the successive symbols in the sequence can achieve. Such a measure, however, is useful only for small values of k [4], which makes it essentially blind to large-scale repetitiveness [5]. Other kinds of compressors, such as Lempel–Ziv [6], grammar compression [7], and the run-length-compressed Burrows–Wheeler transform [8], sharply outperform kth order statistical compressors on repetitive text collections. A notion capturing this repetitiveness, that is, measuring *how much compression can be achieved on repetitive collections*, or alternatively, *how to measure data (compressibility by exploiting) repetitiveness*, has been elusive. Beyond Kolmogorov's complexity [9], which is uncomputable, repetitiveness is measured in ad-hoc terms, as the result of what specific compressors achieve. A list of such measures on a string $S[1\,..\,n]$ includes:

- *Lempel–Ziv compression* [6] parses S into a sequence of *phrases*, each phrase being the longest string that occurs starting to the left in S. The associated measure is the number $z(S)$ of phrases produced, which can be computed in $O(n)$ time [10].
- *Bidirectional macro schemes* [11] extend the Lempel–Ziv parsing so that the source of each phrase may precede or follow it as long as no circular dependencies are introduced. The associated measure $b(S)$ is the number of phrases of the smallest parsing. It satisfies $b \le z = O(b \log \frac{n}{b})$ [12], but computing b is NP-complete [13].
- *Grammar-based compression* [7] builds a context-free grammar that generates (only) S. The associated measure is the size $g(S)$ of the smallest grammar (the sum of the lengths of the right-hand sides of the rules). It satisfies $g = O(n/\log_\sigma n)$ (where σ is the alphabet size) and $z \le g = O(z \log \frac{n}{z})$. While computing g is NP-hard, grammars of size $O(z \log \frac{n}{z})$ can be constructed in linear time [14], [15], [16].
- *Run-length grammar compression* [17] allows in addition rules $A \to B^t$ (t repetitions of B), assumed to be of constant size. The measure is the size $g_{rl}(S)$ of the smallest run-length grammar, and it satisfies $z \le g_{rl} \le g$ and $g_{rl} = O(b \log \frac{n}{b})$ [12].
- *Collage systems* [18] 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

A previous partial version of this article appeared in *Proc. LATIN 2020* [1].

Tomasz Kociumaka is with Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany. Gonzalo Navarro is with Center for Biotechnology and Bioengineering (CeBiB) and Millennium Institute for Foundational Research on Data (IMFD), Dept. of Computer Science, University of Chile, Chile. Nicola Prezza is with Ca' Foscari University of Venice, Italy.

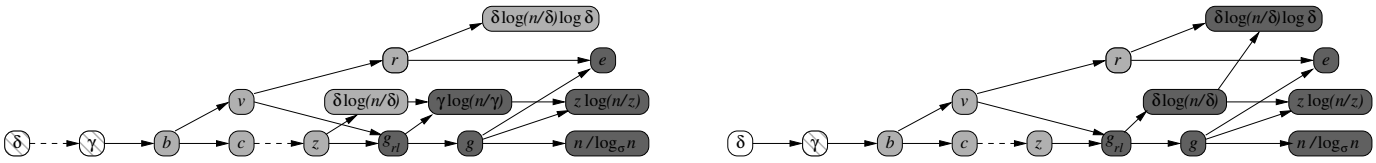

Fig. 1. The relation between compressibility measures before (left) and after (right) our findings in this article. White nodes depict unreachable measures, light gray nodes depict reachable ones (for hatched nodes, the reachability is not resolved), and dark gray nodes depict measures providing logarithmic-time access to the string. Arrows mean that the source measure is asymptotically never larger than the target measure, and solid arrows mean that there are string families where the source is strictly smaller than the target (by a super-constant factor).

$c(S)$ satisfies $c \leq g_{rl}$, $b = O(c)$, and $c = O(z)$ [12].

Burrows–Wheeler transform (BWT) [19] is a permutation of S that has long runs of equal letters if S is repetitive. The number $r(S)$ of maximal equal-letter runs in the BWT can be found in linear time. It holds that $\frac{b}{2} \leq r = O(b \log^2 n)$ [12], [20].

CDAWGs [21] are automata that recognize every substring of S. The associated measure of repetitiveness is $e(S)$, the size of the smallest such automaton (compressed by dissolving states of in-degree and out-degree one), which can be built in linear time [21]. The measure e is always larger than r, g, and z [22], [23].

Lex parsing [12] is analogous to Lempel–Ziv parsing, but each phrase must point to a lexicographically smaller source. The lex parsing is computed in linear time. Its number of phrases, $v(S)$, satisfies $\frac{b}{2} \leq v \leq 2r$ and $v \leq g_{rl}$ [12].

For each measure $x(S)$ above (which we write just as x when S is clear from context), we can represent $S[1..n]$ in space $O(x)$ (meaning $O(x \log n)$ bits in this article). As seen, the measures form a complex hierarchy of dominance relations [3], where b asymptotically dominates all the others (see the left part of Fig. 1). A problem with b (and also z, c, r, and v) is that it is unknown how to *access* S (i.e., extract any character $S[i]$) efficiently (say, in $n^{o(1)}$ time, i.e., without decompressing much of S) within space $O(b)$ (or $O(\max(z, c, r, v))$). This has been achieved in time $O(\log n)$, but only within space $O(z \log \frac{n}{z})$ [15], [14], $O(e)$ [24], $O(r \log \frac{n}{r})$ [25], $O(g)$ [26], [27], and even $O(g_{rl})$ [28]; the latter is $O(b \log \frac{n}{b})$ and subsumes all the other aforementioned results. Providing direct access to the sequences is essential for manipulating them in compressed form, without ever having to decompress them.

Just accessing the string is not sufficient, however, for many applications. One of the most fundamental text processing tasks is *string matching*: find all the occurrences in S of a short string P. This is particularly challenging when the string S is large and scanning it sequentially is not viable. We then resort to *indexes*, which are data structures offering $n^{o(1)}$-time string matching (and possibly further more sophisticated functionality) over a collection of strings. Text indexes based on the kth order empirical entropy are already mature [29] but, as explained earlier, insensitive to repetitiveness. Various more recent compressed indexes build on the repetitiveness measures above; see a thorough review in [30]. The smallest of those find the occ occurrences of $P[1..m]$ in $O(g_{rl})$ space and $O(m \log n + occ \log^{\epsilon} n)$ time, for any constant $\epsilon > 0$ [28], or $O(r)$ space and $O(m \log \log \sigma + occ)$ time [25], [31]. Just the number of occurrences can be computed in space $O(g)$ (but not $O(g_{rl})$) and time $O(m \log^{2+\epsilon} n)$ [28], or space $O(r)$ and time $O(m \log \log \sigma)$ [25], [31].

A relevant recent development in measuring repetitiveness is the concept of a *string attractor* [32]. An attractor Γ is a set of positions in S such that any substring of S has an occurrence covering a position in Γ. Since $\gamma = O(b)$ [32], the size of the smallest attractor asymptotically lower-bounds all the repetitiveness measures listed above; however, it is unknown if one can represent any string in $O(\gamma)$ space. We can do so in space $O(\gamma \log \frac{n}{\gamma})$, which already allows accessing any symbol of S in time $O(\log \frac{n}{\gamma})$ [32], and even indexed text searching [33] within time as low as $O(m + (occ + 1) \log^{\epsilon} n)$ for locating all the occurrences and $O(m + \log^{2+\epsilon} n)$ time for counting them [28]. It is known that $g_{rl} = O(\gamma \log \frac{n}{\gamma})$ [28], though g_{rl} can be smaller than $\gamma \log \frac{n}{\gamma}$ by up to a logarithmic factor, $\log \frac{n}{\gamma}$, so the slower index of size $O(g_{rl})$ offers better space in general.

In terms of measuring repetitiveness, γ and b share some unsatisfactory aspects. Both are NP-hard to compute [13], [32], both are non-monotone when S grows by appending characters at the endpoints of the string [3], [34], and both can grow by a constant factor upon a single edit in S [35].

A. 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 [28], its properties and full potential have not been explored. It is known that $\delta \leq \gamma$ holds for every string, that δ can be computed in $O(n)$ time [28], and that one can encode any string in $O(\delta \log \frac{n}{\delta})$ space because $z = O(\delta \log \frac{n}{\delta})$ [37]. Other bounds we have given hold more tightly in terms of δ, like $r = O(\delta \log \frac{n}{\delta} \log \delta)$ [20]. Further, δ is insensitive to string reversals and alphabet permutations, and monotone upon appending symbols, unlike γ, b, or z [3], [34]. Also, unlike those measures, δ grows by only at most one unit upon single edits of the string [35]. We prove several further properties related to δ:

1) In Section III, we show that δ can be strictly smaller than γ, by up to a logarithmic factor. More precisely, for any n and δ, there are strings with $\gamma = \Omega(\delta \log \frac{n}{\delta})$. We therefore show that the already known upper bounds $\gamma, b, c, z = O(\delta \log \frac{n}{\delta})$ are tight for every n and δ.

TABLE I

THE BEST TIME-TIME TRADEOFFS FOR COUNTING, SEARCHING, ACCESSING, AND FINGERPRINTING ON REPETITIVE TEXTS OF LENGTH n. HERE, m IS THE LENGTH OF THE SEARCH PATTERN, occ IS THE NUMBER OF OCCURRENCES OF THE PATTERN, ℓ IS THE LENGTH OF THE SUBSTRING TO ACCESS, σ IS THE ALPHABET SIZE, w IS THE NUMBER OF BITS IN THE COMPUTER WORD, AND $\epsilon > 0$ IS ANY POSITIVE CONSTANT.

Counting

Reference	Space	Time
Ours (Thm. VI.4)	$O(\delta \log(n/\delta))$	$O(m \log^{2+\epsilon} n)$
[28, Thm. A.5]	$O(g)$	$O(m \log^{2+\epsilon} n)$
[28, Thm. 7.5]	$O(\gamma \log(n/\gamma))$	$O(m + \log^{2+\epsilon} n)$
[28, Thm. 7.6]	$O(\gamma \log(n/\gamma) \log n)$	$O(m)$
[31, Thm. 9]	$O(r)$	$O(m \log \log_w \sigma)$
[25, Thm. 4.10]	$O(r \log \log_w n)$	$O(m)$

Searching

Reference	Space	Time
[28, Thm. A.4]	$O(g_{rl})$	$O(m \log n + occ \log^{\epsilon} n)$
Ours (Cor. VI.1)	$O(\delta \log(n/\delta))$	$O(m \log n + occ \log^{\epsilon} n)$
[28, Thm. 6.8]	$O(\gamma \log(n/\gamma))$	$O(m + (occ+1) \log^{\epsilon} n)$
[28, Thm. 6.12]	$O(\gamma \log(n/\gamma) \log^{\epsilon} n)$	$O(m + occ)$
[36, Thm. 3 & 4]	$O(z \log n)$	$O(m \log m + occ \log \log n)$
[31, Thm. 9]	$O(r)$	$O(m \log \log_w \sigma + occ)$
[25, Thm. 4.10]	$O(r \log \log_w n)$	$O(m + occ)$
[24, Thm. 1]	$O(e)$	$O(m + occ)$

Accessing

Reference	Space	Time
[28, Thm. A.1]	$O(g_{rl})$	$O(\ell + \log n)$
Ours (Thm. VI.7)	$O(\delta \log(n/\delta))$	$O(\lceil \ell / \log_\sigma n \rceil \log(n/\delta))$
[27, Thm. 1]	$O(g)$	$O(\ell / \log_\sigma n + \log n)$
[32, Thm. 5.3]	$O(\gamma \log(n/\gamma))$	$O(\ell / \log_\sigma n + \log(n/\gamma))$
[32, Thm. 5.3]	$O(\gamma \log(n/\gamma) \log^{\epsilon} n)$	$O(\ell / \log_\sigma n + \log(n/\gamma) / \log \log n)$
[25, Thm. 5.1]	$O(r \log(n/r))$	$O(\ell \log(\sigma)/w + \log(n/r))$

Fingerprinting

Reference	Space	Time
[28, Thm. A.3]	$O(g_{rl})$	$O(\log n)$
Ours (Thm. VI.9)	$O(\delta \log(n/\delta))$	$O(\log(n/\delta))$
[33, Lem. 1]	$O(\gamma \log(n/\gamma))$	$O(\log(n/\gamma))$

2) In Section IV, we show that $O(\delta \log \frac{n}{\delta} \log n)$ bits, a space one can reach by Lempel–Ziv compression due to $z = O(\delta \log \frac{n}{\delta})$, is indeed tight: for every n and δ, there are string families where it is impossible to encode every string within $o(\delta \log \frac{n}{\delta} \log n)$ bits. In contrast, the upper bound $O(\gamma \log \frac{n}{\gamma})$ [32] is not known to be tight. We summarize these results in a direct-converse information-theoretic theorem establishing that the measure $\delta \log \frac{n}{\delta} \log n$ bits is asymptotically reachable and necessary for encoding strings of length n with measure δ.
3) In Section V, we show that not only the Lempel–Ziv parsing but also run-length context-free grammars can always represent a string within $O(\delta \log \frac{n}{\delta})$ space; thus also $v, g_{rl} = O(\delta \log \frac{n}{\delta})$. However, this is not true for standard context-free grammars: for every n and δ, there are strings satisfying $g = \Omega(\delta \log^2 \frac{n}{\delta} / \log \log \frac{n}{\delta})$. In particular, if $\delta = n^{1-\Omega(1)}$, this bound simplifies to $g = \Omega(\delta \log^2 n / \log \log n)$, which is almost a logarithmic factor away from $\delta \log \frac{n}{\delta} = \Theta(\delta \log n)$.
4) In Section VI, we combine our preceding result with previous ones on run-length grammars [28] to show that, within space $O(\delta \log \frac{n}{\delta})$, we can not only represent a string but also provide access to any position of it in time $O(\log n)$, compute substring fingerprints in time $O(\log n)$, find the occ occurrences of any pattern string $P[1..m]$ in time $O(m \log n + occ \log^{\epsilon} n)$ for any constant $\epsilon > 0$, and count them in time $O(m \log^{2+\epsilon} n)$. Furthermore, we show that the block tree data structure [38], which provides access to string symbols and substring fingerprints in time $O(\log \frac{n}{z})$, is of size $O(\delta \log \frac{n}{\delta})$, improving upon our first result and on previous analyses and variants of block trees [32], [33], [39].

Fig. 1 shows how the relation between compressibility measures [3] are modified with our findings in (1)–(3). Table I puts in context the contributions in (4): we obtain improved space/time tradeoffs for counting, accessing, and fingerprinting (those are shown in boldface). We have simplified the results in the discussion above for readability; see the lemmas and theorems for the precise statements. In particular, our bounds are also tight with respect to the alphabet size σ: the spaces we have written as $\delta \log \frac{n}{\delta}$ are actually $\delta \log \frac{n \log \sigma}{\delta \log n}$, and similarly the terms $\log \frac{n}{\delta}$ in the time complexities are actually $\log \frac{n \log \sigma}{\delta \log n}$.

II. BASIC CONCEPTS AND THE MEASURE δ

We consider strings $S[1..n]$ as sequences of $|S| = n$ symbols $S[1], \ldots, S[n]$, all drawn from an alphabet $\Sigma = \{0, \ldots, \sigma - 1\}$. We write this as $S \in [0..\sigma)^n$ in short. For simplicity, we assume that every symbol of Σ appears in S for the lower bounds, though our upper bounds hold as long as $\sigma = n^{O(1)}$. The concatenation of strings S and S' is denoted $S \cdot S'$; we can also identify individual symbols of Σ with the corresponding string of length 1. A substring of S is denoted $S[i..j] = S[i] \cdots S[j]$ and the empty string is denoted ε. Prefixes $S[1..i]$ and suffixes $S[i..n]$ of S are also denoted $S[..i]$ and $S[i..]$, respectively.

We assume the transdichotomous RAM model, which is a word RAM model on a machine word of $\Theta(\log n)$ bits. Consequently, when we measure the space in words (the default), we consider that each word holds $\Theta(\log n)$ bits. Therefore, $O(x)$ space is equivalent to $O(x \log n)$ bits of space.

In the rest of this section we describe in more detail the repetitiveness measures that are most relevant for this article, with particular emphasis on δ.

A. Lempel–Ziv compression and bidirectional macro schemes

There are several flavors of Lempel–Ziv compression [6]. The one we use yields a clean repetitiveness measure.

We traverse the string S left to right, parsing it into a sequence of phrases. If we have already parsed $S[..i]$, then we look for the longest prefix of $S[i+1..]$ that occurs in S starting at a position in $[1..i]$. Let $S[i+1..j]$ be that longest prefix. Then the next phrase is $S[i+1..j]$ and the parsing continues from $S[j+1..]$. In case the length of the phrase is zero (i.e., $S[i+1]$ does not occur in $S[1..i]$), we create instead a single phrase $S[i+1]$ and continue from $j = i+2$.

We call $z(S)$ the number of phrases produced by the process above. The Lempel–Ziv compression essentially encodes S in $O(z(S))$ space by writing a sequence of tuples, one per phrase, indicating where, in the text already seen, can the next phrase be copied from: it suffices to encode the starting position and the length of the substring to copy, or the explicit symbol when it appears for the first time.

For example, the string $S = \mathsf{alabaralalabarda\$}$ is parsed into $z(S) = 11$ phrases as follows (we use vertical bars to delimit the phrases): $\mathsf{a|l|a|b|a|r|ala|labar|d|a|\$}$.

A bidirectional macro scheme generalizes Lempel–Ziv parsing by allowing the phrases to be obtained from occurrences that are ahead in the text, too. The only restriction is that, in the process of decoding a symbol at $S[i]$, we do not return to the position i again. We call $b(S)$ the minimum number of phrases in a valid bidirectional macro scheme for string S.

For example, a valid bidirectional macro scheme for $S = \mathsf{alabaralalabarda\$}$ parses it as $\mathsf{ala|b|a|r|a|l|alabar|d|a|\$}$, where the first phrase is obtained from $S[7..9]$ and the seventh from $S[1..6]$. To obtain $S[9]$, for example, we are redirected to $S[1]$ and then to $S[7]$, then obtaining the a from the explicit phrase. Since this macro scheme has 10 phrases, we know that $b(S) \le 10$.

B. Grammar compression and run-length grammars

Grammar compression [7] consists in building a context-free grammar that generates (only) the string S. If S is repetitive, it will be possible to build a small grammar generating it. The *size* of the grammar is defined as the sum of the lengths of all the right-hand sides of the rules. We call $g(S)$ the size of the smallest grammar generating S.

For example, a grammar generating $S = \mathsf{alabaralalabarda\$}$ is $A \to \mathsf{al}$, $B \to A\mathsf{abar}$, and the initial symbol $C \to BAB\mathsf{da\$}$. The size of this grammar is 13, so we know that $g(S) \le 13$.

A less classical concept is that of a run-length (context-free) grammar, which in addition allows rules of the form $A \to B^t$. This does not add to the expressive power of context-free grammars, but it enables smaller grammars to generate a string S if we assume that the size of the above rule is constant instead of t.

For example, the smallest grammar generating string $S = \mathsf{a}^n$ is of size $\Theta(\log n)$, $A \to \mathsf{aa}$, $B \to AA$, $C \to BB$, ... Instead, a run-length grammar of constant size, $A \to a^n$, generates it.

C. String attractors

An important compressibility measure is given via the concept of a *string attractor*, introduced by Kempa and Prezza [32] as a tool for studying dictionary compressors (including Lempel–Ziv, the run-length Burrows-Wheeler transform, and grammar compressors) under a common framework.

Definition II.1 ([32])**.** *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 an occurrence $S[i'..j'] = S[i..j]$ that covers an attractor position $p \in \Gamma \cap [i'..j']$. We will denote $\gamma(S)$ the cardinality of a smallest string attractor for the string S, and will use just γ when S is clear from the context.*

Indeed, it can be shown that, letting α be the output size of any of those dictionary compressors, one can build a corresponding string attractor of cardinality $O(\alpha)$. Conversely, given a string attractor of cardinality γ for a string of length n, one can build a Lempel–Ziv factorization or a context-free grammar of size $O(\gamma \ \mathrm{polylog}(n))$. The cardinality γ of a smallest string attractor is then an important measure of string repetitiveness.

For example, an attractor for $S = \mathsf{alabaralalabarda\$}$ is $\Gamma = \{4, 5, 6, 8, 15, 17\}$, because every substring of S has a copy (possibly, the substring itself) including some position in Γ. We know it is of minimal size $|\Gamma| = \gamma = 6$ because the alphabet size of S is also $\sigma = 6$, and it must obviously hold that $\gamma \ge \sigma$.

D. The measure δ

The measure δ was recently defined by Christiansen et al. [28, Sec. 5.1], though it is based on the expression $d_k(S)/k$, introduced by Raskhodnikova et al. [37] to approximate z. The set of values $d_k(S)$ are known as the substring complexity of S, so δ is a function of it. In this section, we summarize what is known about δ.

Definition II.2. *Let $d_k(S)$ be the number of distinct length-k substrings in S. Then*

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

We will also use δ when S is clear from the context.

In our example $S = \mathsf{alabaralalabarda\$}$ it holds that $\delta(S) = 6$, because $d_1(S)/1 = 6/1 = 6$, $d_2(S)/2 = 9/2 = 4.5$, $d_3(S)/3 = 10/3 \approx 3.3$, etc.

First, we note that measure δ is upper-bounded by γ, which implies in particular that $\delta = O(n/\log_\sigma n)$.

Lemma II.3 ([28, Lemma 5.6])**.** *Every string S satisfies $\delta(S) \le \gamma(S)$.*

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

On the other hand, $\delta \log \frac{n}{\delta}$ is an asymptotically reachable compressibility measure, that is, one can encode any string within $O(\delta \log \frac{n}{\delta})$ space. This is directly obtained from

Raskhodnikova et al. [37, Lemma 3], but we now establish a more refined result that takes into account the alphabet size.

Lemma II.4. *Every string $S \in [0\,..\,\sigma)^n$ with measure $\delta = \delta(S)$ satisfies $z(S) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$.*

Proof. For each $k \in \mathbb{Z}_+$, let n_k denote the number of length-k phrases in the Lempel–Ziv factorization of S, excluding the last phrase. The inequality $\sum_{k=1}^{\ell} k n_k \le 2\ell(\delta+1)$ holds for every $\ell \in \mathbb{Z}_+$, as shown in [37, Claim 4]. Thus, due to $\sum_{k=\ell}^{2\ell-1} \ell n_k \le \sum_{k=\ell}^{2\ell-1} k n_k \le 4\ell(\delta+1)$, it follows that $\sum_{k=\ell}^{2\ell-1} n_k \le \frac{1}{\ell} \cdot 4\ell(\delta+1) = O(\delta)$. At the same time, $\sum_{k=\ell}^{\infty} n_k \le \frac{n}{\ell}$ (because $\sum_{k=1}^{\infty} k n_k \le n$) and $\sum_{k=1}^{\ell} n_k \le \sum_{k=1}^{\ell} \sigma^k = O(\sigma^\ell)$ (because phrases are distinct). Combining these three bounds, in particular applying the first one a logarithmic number of times to handle the second summation, we obtain

$$\begin{aligned}\sum_{k=1}^{\infty} n_k &\le \sum_{k=1}^{\lfloor \log_\sigma \delta \rfloor} n_k + \sum_{k=1+\lfloor \log_\sigma \delta \rfloor}^{\lceil n/\delta \rceil - 1} n_k + \sum_{k=\lceil n/\delta \rceil}^{\infty} n_k \\ &= O(\delta) + O(\delta \cdot (\log \tfrac{n}{\delta} - \log\log_\sigma \delta)) + O(\delta) \\ &= O(\delta \log \tfrac{n \log \sigma}{\delta \log \delta}).\end{aligned}$$

Consequently, $z(S) \le 1 + \sum_{k=1}^{n} n_k = O(\delta \log \frac{n \log \sigma}{\delta \log \delta})$. The term $\log \frac{n \log \sigma}{\delta \log \delta}$ is $O(\log \frac{n \log \sigma}{\delta \log n})$: if $\delta \ge \sqrt{n}$, then $\log \delta = \Theta(\log n)$; otherwise both $\log \frac{n \log \sigma}{\delta \log \delta}$ and $\log \frac{n \log \sigma}{\delta \log n}$ are $\Theta(\log n)$. □

Since γ, b, and c are $O(z)$, these three measures are all upper bounded by $O(\delta \log \frac{n \log \sigma}{\delta \log n}) \subseteq O(\delta \log \frac{n}{\delta})$. Additionally, we conclude that $g_{rl} \le g = O(z \log \frac{n}{z}) = O(\delta \log^2 \frac{n}{\delta})$, though we obtain a tight result $g_{rl} = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ later in this article.

It is also known that δ can be computed in $O(n)$ time [28, Lemma 5.7], but we present a simpler and more practical construction here.

Lemma II.5. *The measure $\delta(S)$ can be computed in $O(n)$ time and space from $S[1\,..\,n]$.*

Proof. Let us assume for technical convenience that S ends with a unique endmarker \$ that is smaller than all the other symbols (otherwise, we add it and correct the result at the end). Let $SA[1\,..\,n]$ be the suffix array of S, that is, it holds that $S[SA[i]\,..] < S[SA[i+1]\,..]$ in lexicographic order, for all $1 \le i < n$; this is well defined because of the endmarker. Let $lcp(X, Y)$ be the length of the longest common prefix between strings X and Y. Let us define the array $LCP[1\,..\,n]$ so that $LCP[1] = 0$ and, for every $i > 1$, $LCP[i] = lcp(S[SA[i]\,..], S[SA[i-1]\,..])$. The arrays SA and LCP can be computed in $O(n)$ time [40], [41].

We now note that

$$d_k(S) = |\{i \in [1\,..\,n] : LCP[i] < k\}| - k + 1.$$

This is because all the occurrences of each distinct substring C of length k prefix consecutive suffixes in SA, appearing in a range $SA[i\,..\,j]$, so $LCP[i] < k$ and $LCP[t] \ge k$ for all $t \in [i+1\,..\,j]$. The main part of the above formula counts, precisely, all those positions i, one per distinct substring C of length k. It also counts, however, all the suffixes $S[n-p\,..]$ for $0 \le p \le k-2$, which are shorter than k and hence should not be counted in $d_k(S)$. Note that, thanks to the endmarker, we have counted each such string exactly once, so we remove them by subtracting $k-1$.

The computation then proceeds as follows. We initialize an array $D[k] \leftarrow 0$ for all $k \in [0\,..\,n-1]$. Then, we increment $D[LCP[i]]$ for all $i \in [1\,..\,n]$, so at the end $D[k] = |\{i \in [1\,..\,n] : LCP[i] = k\}|$. We next accumulate $D[k] \leftarrow D[k] + D[k-1] - 1$ for k from 1 to $n-1$, which can be easily seen by induction to produce $D[k] = d_{k+1}(S)$ according to our formula above. Finally, we compute $\delta \leftarrow \max\{D[k]/(k+1) : k \in [0\,..\,n-1]\}$. If we had to insert the endmarker \$, then every $D[k]$ is counting the spurious suffix $S[n-k+1\,..]$, so we compute instead $\delta \leftarrow \max\{(D[k]-1)/(k+1) : k \in [0\,..\,n-1]\}$. □

Finally, we note some obvious positive properties of δ as a compressibility measure: it is insensitive to reversing the string and to alphabet permutations, and it is monotone when we add/remove prefixes/suffixes to/from S. String reversals and alphabet permutations clearly preserve the repetitiveness of the string, thus a good compressibility measure should be invariant under these operations. Furthermore, monotonicity is also a desirable property for such a measure: the complexity of a substring of S should always be smaller than the complexity of S itself (indeed, it is not hard to see that Kolmogorov complexity has all these desirable properties). Other measures, like z, b, or γ, are not monotone, z is sensitive to reversals, and v and r are also sensitive to alphabet permutations [3], [34]. Another desirable property of a repetitiveness measure is that it changes slowly as we edit the text. A recent study [35] shows that, upon a single symbol edit in S, measure δ grows at most by 1, whereas γ, b, z, and others grow, in the worst case, by a constant factor; r may grow by $\Theta(\log n)$ at least.

III. Lower bounds on attractors

In this section, we show that there exist string families where $\delta = o(\gamma)$; in fact, δ can be smaller by up to a logarithmic factor. More precisely, for any string length n and value $\delta = \delta(S) \in [2\,..\,n]$, we build a string satisfying $\gamma(S) = \Omega(\delta \log \frac{n}{\delta})$. This shows that the bound $\gamma \le b \le z = O(\delta \log \frac{n}{\delta})$ is asymptotically tight if we do not consider the alphabet size; we then refine our result to consider σ.

We build our results on (variants of) the following string family.

Definition III.1. *Consider an infinite string $S_\infty[1\,..]$, where $S_\infty[i] = \mathtt{b}$ if $i = 2^j$ for some integer $j \ge 0$, and $S_\infty[i] = \mathtt{a}$ otherwise. For $n \ge 1$, let $S_n = S_\infty[1\,..\,n]$. We then define the string family $\mathcal{S} = \{S_n : n \ge 1\}$ as the set of the prefixes of S_∞.*

We first prove that the strings in $\mathcal{S}$ satisfy $\delta = O(1)$ and $\gamma = \Omega(\log n)$.

Lemma III.2. *For every $n \ge 1$, the string S_n satisfies $\delta(S_n) \le 2$ and $\gamma(S_n) \ge \frac{1}{2}\lfloor \log n \rfloor$.*

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 $i \in [0\,..\,k)$. Hence, the distinct length-k substrings of S_∞ are those starting up to position 2^{j-1}, $S_\infty[i\,..\,i+k)$ for $i \in [1\,..\,2^{j-1}]$, and the $k+1$ already mentioned strings, for a total of $d_k(S_\infty) \leq 2^{j-1}+k+1$. Choosing $j = \lceil \log k \rceil$, we get $d_k(S_\infty) \leq 2^{\lceil \log k \rceil - 1} + k + 1 \leq 2^{\log k} + k = 2k$, yielding 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 position. Consequently, $\gamma(S_n) \geq \frac{k}{2}$ holds for all integers $n \geq 2^k$ and $k \geq 0$. Choosing $k = \lfloor \log n \rfloor$, we get $\gamma(S_n) \geq \frac{1}{2}\lfloor \log n \rfloor$. □

We now generalize the basic result to every integer value $2 \leq \delta = o(n)$, showing for each such δ there are strings S satisfying $\delta(S) = \delta$ and $\gamma(S) = \Omega(\delta \log \frac{n}{\delta})$, that is, γ can be asymptotically larger than δ.

Theorem III.3. *For every length n and integer 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 $n-m+1 \geq 3m$ into $\sum_{i=1}^{m} n_i = n-m+1$ 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 delimiters $\$_i$ for $i \in [1\,..\,m)$. For this, we take $S^{(i)}$ to be the string S_{n_i} of Definition III.1, with the alphabet $\{\mathtt{a},\mathtt{b}\}$ replaced by $\{\mathtt{a}_i,\mathtt{b}_i\}$, and we define $S = S^{(1)}\,\$_1\,S^{(2)}\,\$_2 \cdots \$_{m-1}\,S^{(m)}$, which is of length n.

Notice that, for each $k \in [1\,..\,n]$, 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. Since $d_k(S^{(i)}) \leq 2k$ by Lemma III.2, we have that $d_k(S) \leq (3m-1)k$, and thus $\delta(S) \leq 3m-1$. In fact, $\delta(S) = 3m-1$ because $d_1(S) = 3m-1$. Furthermore, $\gamma(S) \geq \sum_{i=1}^{m} \gamma(S^{(i)}) \geq \sum_{i=1}^{m} \frac{1}{2}\lfloor \log n_i \rfloor = \Omega(m \log \frac{n}{m}) = \Omega(\delta \log \frac{n}{\delta})$, where the first inequality holds because the alphabets of $S^{(i)}$ are disjoint and the second is due to Lemma III.2.

This construction proves the theorem for $\delta = 3m-1$ and $n \geq 4m-1$. If $\delta < \frac{3}{4}n$ and $\delta \bmod 3 \neq 2$, we use $m = \lfloor \frac{\delta+1}{3} \rfloor$ and initially construct a string S of length $n-(\delta+1) \bmod 3 \geq 4m-1$. Next, we append $(\delta+1) \bmod 3$ additional delimiters, which results in each of the measures $\delta(S)$, $\gamma(S)$, and n increased by $(\delta+1) \bmod 3$. Finally, we note that if $\delta \geq \frac{3}{4}n = \Omega(n)$, then the claim reduces to $\gamma = \Omega(\delta)$ and therefore follows directly from Lemma II.3. □

Note that $\delta(S)$ needs not be an integer. Still, by the theorem, given an arbitrary value of δ, there is a string S with $\delta(S) = \lfloor \delta \rfloor \leq \delta$ and $\gamma(S) = \Omega(\delta \log \frac{n}{\delta})$, which still gives the desired separation. A more real limitation of the theorem is that the strings it builds have alphabet size $\sigma = \delta$. We now obtain a more general result that holds for smaller alphabet sizes.

Theorem III.4. *For every alphabet size $\sigma \geq 2$, length $n \geq 1$, and integer parameter $9\sigma \leq \Delta \leq \frac{n \log \sigma}{2 \log n}$, there exists a string $S \in [0\,..\,\sigma)^n$ such that $\delta(S) \leq \Delta$ and $\gamma(S) = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n})$.*

Proof. Let $d = \lfloor \frac{\Delta}{9} \rfloor$, $\ell = \lceil \log_\sigma d \rceil$, and $\mathsf{r} : [0\,..\,d) \to [0\,..\,\sigma)^\ell$ be an arbitrary injective function. Consider an infinite string

$$S = \bigodot_{i=0}^{\infty} S_i = \bigodot_{i=0}^{\infty}\bigodot_{j=0}^{d-1} S_{i,j} = \bigodot_{i=0}^{\infty}\bigodot_{j=0}^{d-1} \mathsf{r}(j) \cdot \mathtt{1} \cdot \mathtt{0}^{2^{i+1}\ell} \cdot \mathtt{1},$$

where $\cdot$ and $\bigodot$ stand for concatenation.

Let us first argue that $\delta(S) \leq \Delta$. For this, fix a substring length $k \in \mathbb{Z}_+$. If $k < \ell$, then $d_k(S) \leq \sigma^k \leq \sigma^{\ell-1} \leq \sigma^{\log_\sigma d} = d < \Delta k$ holds as claimed. Thus, we henceforth assume $k \geq \ell$ and consider the smallest integer $\tau \in \mathbb{Z}_{\geq 0}$ such that $2^{\tau+1}\ell \geq k$; observe that $2^\tau \ell < k$. The number of length-k substrings starting within $S_0 \cdots S_{\tau-1}$ does not exceed $\sum_{i=0}^{\tau-1} |S_i| = \sum_{i=0}^{\tau-1} d(1+\ell+2^{i+1}\ell+1) \leq \sum_{i=0}^{\tau-1} d\ell(3+2^{i+1}) \leq 5d\ell 2^\tau \leq 5dk$. The remaining length-$k$ substrings occur within $\mathtt{0}^k \cdot \mathtt{1} \cdot \mathsf{r}(j) \cdot \mathtt{1} \cdot \mathtt{0}^{k-1}$ for some $j \in [0\,..\,d)$. The number of such substrings does not exceed $d(k+\ell+2) \leq 4dk$. Overall, $d_k(S) \leq 9dk \leq \Delta k$ holds as claimed.

Next, we shall prove that $\gamma(S[1\,..\,n]) = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n})$ holds when $\Delta \leq \frac{n \log \sigma}{2 \log n}$. For this, observe that each substring $S_{i,j}$ has a unique occurrence within S, and all these occurrences are disjoint. Consequently, if $n \geq 5d\ell 2^\tau \geq \sum_{i=0}^{\tau-1} |S_i|$, then $\gamma(S[1\,..\,n]) \geq \tau d$. Define $\tau = \lfloor \log \frac{n}{5d\ell} \rfloor$ and observe that $\frac{n}{5d\ell} \geq \frac{n \log \sigma}{10d \log d} \geq \frac{9n \log \sigma}{10\Delta \log n} \geq \frac{9}{5}$ implies $\tau = \Omega(\log \frac{n \log \sigma}{\Delta \log n})$. Thus, $\gamma(S[1\,..\,n]) \geq \tau d = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n})$ holds as claimed. □

This implies that we can have a separation between δ and γ as long as $\delta = o(n/\log_\sigma n)$. In Theorem III.3 we used $\sigma = \delta$, thus the condition boiled down to $\delta = o(n)$, but more generally we can separate δ and γ unless $\delta = \Theta(n/\log_\sigma n)$. This is unavoidable because in that case it also holds that $\gamma = \Theta(n/\log_\sigma n)$, because this is the upper bound of both measures, holding for incompressible texts. For example, a de Bruijn sequence of order k has $d_k(S) = \sigma^k$ and $n = \sigma^k + k - 1$, so $\delta \geq \sigma^k/k = \Theta(n/\log_\sigma n)$.

IV. Lower bounds on text entropy

In this section, we prove that there are string families where not all members can be encoded in $o(\delta \log n)$ space: for every length n and every integer value $\delta \in [2\,..\,n]$, there is a string family where some elements require $\Omega(\delta \log \frac{n}{\delta})$ space, or $\Omega(\delta \log \frac{n}{\delta} \log n)$ bits, to be represented. Therefore, using $O(\delta \log \frac{n}{\delta})$ space is worst-case optimal for every δ. Further, when considering the alphabet size, the lower bound refines to $\Omega(\delta \log \frac{n \log \sigma}{\delta \log n})$. This is a reachable bound, because every string can be represented within $O(z) \subseteq O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space. In comparison, it is not known if the upper bound $O(\gamma \log \frac{n}{\gamma})$ to encode every string family [32] is also tight. We build on the convenient concept of worst-case entropy.

Definition IV.1. *The* worst-case entropy *of a set $\mathcal{S}$, $\log_2 |\mathcal{S}|$, is the minimum number of bits that any coder needs to represent any arbitrary element of $\mathcal{S}$. That is, for every coder $C : \mathcal{S} \to \{0,1\}^*$, there exists at least one element $S \in \mathcal{S}$ such that*

$|C(S)| \geq \log_2 |\mathcal{S}|$. *Indeed, if all codes were of length at most* $\ell < \log_2 |\mathcal{S}|$ *bits, then there would only be at most* $2^\ell < |\mathcal{S}|$ *codes.*

We will use this simple concept to prove lower bounds on the number of bits needed to represent any member of some string families. Concretely, we will devise families where all the elements have low δ measure, though the families are large enough to enforce a sufficiently high worst-case entropy, that is, to forbid encoders that assign too short codes to all the strings. To start with the simple version that disregards σ, we consider a family of variants of the infinite string S_∞ of Definition III.1, where the positions of $\mathtt{b}$s are further apart and slightly perturbed.

Definition IV.2. *The family* $\mathcal{S}^{\mathtt{p}}$ *is formed by all the infinite strings* S *over* $\{\mathtt{a}, \mathtt{b}\}$ *where the first* $\mathtt{b}$ *is placed at* $S[1]$ *and, for* $j \geq 2$*, the* j*th* $\mathtt{b}$ *is placed anywhere in* $S[2 \cdot 4^{j-2} + 1 \,..\, 4^{j-1}]$*. The family* $\mathcal{S}^{\mathtt{p}}_n$ *consists of the length-*n *prefixes of the infinite strings of the family* $\mathcal{S}^{\mathtt{p}}$*, that is,* $\mathcal{S}^{\mathtt{p}}_n = \{S[1 \,..\, n] : S \in \mathcal{S}^{\mathtt{p}}\}$.

Lemma IV.3. *For every integer* $n \geq 1$*, the family* $\mathcal{S}^{\mathtt{p}}_n$ *has worst-case entropy* $\Omega(\log^2 n)$.

Proof. In our definition of $\mathcal{S}^{\mathtt{p}}$, the location of the jth $\mathtt{b}$ can be chosen among $2 \cdot 4^{j-2}$ positions, and each combination of these choices generates a different string in $\mathcal{S}^{\mathtt{p}}_n$ as long as $n \geq 4^{j-1}$. Hence, $|\mathcal{S}^{\mathtt{p}}_n| = \prod_{j=2}^{i+1}(2 \cdot 4^{j-2}) = 2^{\Omega(i^2)} = 2^{\Omega(\log^2 n)}$ for $i = \lfloor \log_4 n \rfloor$. To distinguish strings in $\mathcal{S}^{\mathtt{p}}_n$, any encoding needs $\log |\mathcal{S}^{\mathtt{p}}_n| = \Omega(\log^2 n)$ bits, the worst-case entropy of $\mathcal{S}^{\mathtt{p}}_n$. □

We now build on the family $\mathcal{S}^{\mathtt{p}}_n$, and of S_n of Definition III.1, to prove two lemmas which, combined, establish the lower bound of $\Omega(\delta \log \frac{n}{\delta} \log n)$ bits to encode the elements of certain string families.

Lemma IV.4. *For every length* n *and integer* $\delta \in [2 \,..\, \lceil \frac{3n}{4} \rceil)$*, there exists a family of length-*n *strings of common measure* δ *with worst-case entropy* $\Omega(\delta \log^2 \frac{n}{\delta})$.

Proof. As in the proof of Lemma III.2, we prove that $\delta(S)$ for any string $S \in \mathcal{S}^{\mathtt{p}}_n$ is at most 2. Starting from position $4^{j-1} + 1$, the distance between any two consecutive $\mathtt{b}$s is at least 4^j. Therefore, the distinct substrings of length $k \leq 4^j$ are those that start at position $i \in [1 \,..\, 4^{j-1}]$ and those of the form $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}\mathtt{a}^{k-i-1}$ for $i \in [0 \,..\, k)$, which yields a total of $d_k(S) \leq 4^{j-1} + k + 1$. Choosing $j = \lceil \log_4 k \rceil$, we get $d_k(S) \leq 4^{\lceil \log_4 k \rceil - 1} + k + 1 \leq 4^{\log_4 k} + k = 2k$. By definition of δ, we conclude that $\delta(S) \leq 2$ for every $S \in \mathcal{S}^{\mathtt{p}}_n$. Thus, by Lemma IV.3, encoding $\mathcal{S}^{\mathtt{p}}_n$ requires $\Omega(\delta \log^2 \frac{n}{\delta})$ bits.

We now generalize the result to larger δ. As in the proof of Theorem III.3, let $m \geq 1$, $n \geq 4m - 1$, and $n - m + 1 = \sum_{i=1}^m n_i$, where $n_i = \Omega(\frac{n}{m})$ and $n_i \geq 3$. Let $S^{(i)}$, of length n_i, be built from some $S \in \mathcal{S}^{\mathtt{p}}_{n_i}$, with $\mathtt{a}$ replaced by $\mathtt{a}_i$ and $\mathtt{b}$ replaced by $\mathtt{b}_i$. Finally, let $S^* = S^{(1)}\,\$_1\,S^{(2)}\,\$_2 \cdots \$_{m-1}\,S^{(m)}$. Since $d_k(S^{(i)}) \leq 2k$ as per the previous paragraph, it holds, just as in the proof of Theorem III.3, that $\delta(S^*) = 3m - 1$. Let $\mathcal{S}^*_n$ be the set of possible strings S^* of length n we obtain by choosing the strings $S^{(i)}$. Even fixing the lengths n_i, we have $|\mathcal{S}^*_n| = \prod_{i=1}^m 2^{\Omega(\log^2 n_i)}$, and thus the worst-case entropy of $\mathcal{S}^*_n$ is $\log |\mathcal{S}^*_n| = \sum_{i=1}^m \Omega(\log^2 n_i) = \Omega(m \log^2 \frac{n}{m}) = \Omega(\delta \log^2 \frac{n}{\delta})$. The case where $\delta < \frac{3}{4}n$ is not of the form $3m-1$ is handled as in the proof of Theorem III.3. □

Lemma IV.5. *For every length* n *and integer* $\delta \in [2 \,..\, n]$*, there exists a family of length-*n *strings of common measure* δ *with worst-case entropy* $\Omega(\delta \log \frac{n}{\delta} \log \delta)$.

Proof. Recall the string S_n of Definition III.1 and fix an integer $m \geq 1$. Let $\mathcal{S}^{\mathtt{r}}_n \subseteq \{\mathtt{a}, \mathtt{b}_1, \ldots, \mathtt{b}_m\}^n$ be the family of strings obtained from S_n by replacing every $\mathtt{b}$ with $\mathtt{b}_r$ for some $r \in [1 \,..\, m]$. Since $|\mathcal{S}^{\mathtt{r}}_n| = m^{1+\lfloor \log n \rfloor}$, we need at least $\log |\mathcal{S}^{\mathtt{r}}_n| = \Omega(\log n \log m)$ bits to represent a string in $\mathcal{S}^{\mathtt{r}}_n$.

On the other hand, as in the proof of Lemma III.2, the distinct substrings of length $k \leq 2^j$ in $S^{\mathtt{r}}_n \in \mathcal{S}^{\mathtt{r}}_n$ are those starting in $S^{\mathtt{r}}_n[1 \,..\, 2^{j-1}]$ and those of the form $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}_r\mathtt{a}^{k-i-1}$ for $i \in [0 \,..\, k)$ and $r \in [1 \,..\, m]$. Thus, $d_k(S^{\mathtt{r}}_n) \leq 2^{j-1} + 1 + km$. Choosing $j = \lceil \log k \rceil$, we get $d_k(S^{\mathtt{r}}_n) \leq k(m+1)$, and therefore $\delta(S^{\mathtt{r}}_n) \leq m + 1$.

Similarly to the proof of Theorem III.3, let $n \geq 4m - 1$ and $n - 3m + 1 = \sum_{i=1}^m n_i$, where $n_i = \Omega(\frac{n}{m})$ are positive integers. Let us choose any $S^{(i)}_{n_i} \in \mathcal{S}^{\mathtt{r}}_{n_i}$ and define $S^* = S^{(1)}_{n_1}\,\$_1\,S^{(2)}_{n_2}\,\$_2 \cdots \$_{m-1}\,S^{(m)}_{n_m}\,S'$, where, in principle, $S' = \mathtt{a}^{2m}$. Consider the distinct length-k substrings of S^* for $k \leq 2^j$. These include the (at most) $k(m-1)$ substrings containing a delimiter $\$_i$ (with $i \in [1 \,..\, k)$) and the (at most) $2^{j-1} \cdot m$ substrings starting within the first 2^{j-1} position of some $S^{(i)}_{n_i}$ (with $i \in [1 \,..\, k]$). As argued in the previous paragraph, the remaining substrings are of the form $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}_r\mathtt{a}^{k-i-1}$ for $r \in [1 \,..\, m]$ and $i \in [1 \,..\, k)$; note that this analysis includes substrings overlapping $S^{(m)}_{n_m}$ and $S' = \mathtt{a}^{2m}$ despite the lack of delimiter between them. Consequently, we get $d_k(S^*) \leq k(m-1) + 2^{j-1}m + km + 1 \leq (3m-1)k$, setting $j = \lceil \log k \rceil$. At the same time, $m+1 \leq d_1(S^*) \leq 2m$, because S^* contains $m-1$ delimiters, $\mathtt{a}$, and between 1 and m distinct symbols $\mathtt{b}_r$ with $r \in [1 \,..\, m]$.

We conclude that $m + 1 \leq \delta(S^*) \leq 3m - 1$. We now modify S' so that $\delta(S^*) = 3m - 1$. For this, we replace the subsequent symbols $S'[2m], S'[2m-1], \ldots$ by fresh delimiters $\$_m, \$_{m+1}, \ldots$, stopping as soon as $d_k(S^*) \geq (3m-1)k$ holds for some $k \in [1 \,..\, n]$. Since each value $d_k(S^*)$ grows by at most 1 per delimiter added, we have $\delta(S^*) = 3m - 1$ upon termination of the process. Moreover, $d_1(S^*)$ grows by exactly 1 per delimiter added, so the process terminates in at most $3m - 1 - (m+1) = 2m - 2$ steps, which means that S' is long enough to fit the delimiters.

We then obtain a family of possible strings $S^*[1 \,..\, n]$ of common measure $\delta = 3m - 1$. Note that the suffix S' is uniquely determined by the prefix containing the strings $S^{(i)}_{n_i}$. Even fixing the lengths n_i, the number of possible strings S^* we obtain by choosing the strings $S^{(i)}_{n_i}$ is $\prod_{i=1}^m |\mathcal{S}^{\mathtt{r}}_{n_i}| = \prod_{i=1}^m m^{1+\lfloor \log n_i \rfloor} > \prod_{i=1}^m m^{\log n_i}$. The worst-case entropy of the family is thus $\sum_{i=1}^m \log n_i \log m = \Omega(m \log \frac{n}{m} \log m) = \Omega(\delta \log \frac{n}{\delta} \log \delta)$.

The case where $\delta < \frac{3}{4}n$ is not of the form $3m - 1$ is handled as in the proof of Theorem III.3. For $\delta \geq \frac{3}{4}n$, on the other hand, we construct a different family of length-n strings with common measure δ, with strings of the form $S = \$_{\pi(1)} \cdots \$_{\pi(\delta)} \cdot \$^{n-\delta}_{\pi(\delta)}$ for a permutation π of $[1 \,..\, \delta]$.

Each length $k \in [1\,..\,n]$ satisfies $d_k(S) = \min(\delta, n-k+1)$, so $\delta(S) = \delta$. Since there are $\delta!$ possible strings S, we need $\log_2 \delta! = \Omega(\delta \log \delta) = \Omega(\delta \log \frac{n}{\delta} \log \delta)$ bits to represent family members. □

Lemma IV.6. *For every length n and integer $\delta \in [2\,..\,n]$, there exists a family of length-n strings of common measure δ with worst-case entropy $\Omega(\delta \log \frac{n}{\delta} \log n)$.*

Proof. Follows from taking the union of the families built in Lemmas IV.4 and IV.5, since $\max(\log \delta, \log \frac{n}{\delta}) = \Omega(\log n)$ and $\log \delta = \Omega(\log \frac{n}{\delta})$ for $\delta \geq \lceil \frac{3n}{4} \rceil$. □

Note that, since δ can be rational in general, we are not guaranteeing that for every possible value of δ there are large enough string families; only for integer values. Still, for any value of δ we can build a family with common measure $\lfloor \delta \rfloor$ and worst-case entropy $\Omega(\delta \log \frac{n}{\delta} \log n)$, which still serves the purpose of showing that $\Omega(\delta \log \frac{n}{\delta})$ space is necessary in general. Once again, a limitation of our result is that the family of strings it builds has a large alphabet size σ. We now establish a more refined result that holds for many values of σ. We now prove a lemma that generalizes Lemmas IV.4 and IV.5, and then obtain the final direct-converse theorem.

Lemma IV.7. *For every alphabet size $\sigma \geq 2$, length $n \geq 1$, and integer parameter $10\sigma \leq \Delta \leq \frac{n \log \sigma}{2 \log n}$, there is a family of strings $S[1\,..\,n]$ over an alphabet of size σ with $\delta(S) \leq \Delta$ and worst-case entropy $\Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n} \log n)$.*

Proof. Let $d = \lfloor \frac{\Delta}{10} \rfloor$, $\ell = \lceil \log_\sigma d \rceil$, and $\mathsf{r} : [0\,..\,d) \to [0\,..\,\sigma)^\ell$ be an arbitrary injective function. For each integer $i \in \mathbb{Z}_{\geq 0}$, consider a family of strings $\mathsf{F}_i := \{\mathsf{r}(j) \cdot \mathtt{1} \cdot \mathtt{0}^e \cdot \mathtt{1} : j \in [0\,..\,d) \text{ and } e \in [2^{i+1}\ell\,..\,2^{i+2}\ell)\}$. Further, let $\mathsf{F} = \bigodot_{i=0}^{\infty} \mathsf{F}_i^d$ be a family of infinite strings obtained by concatenating any d strings from F_0, followed by any d strings from F_1, and so on.

Let us first argue that $\delta(S) \leq \Delta$ holds for every $S \in \mathsf{F}$. For this, fix a substring length $k \in \mathbb{Z}_+$. If $k < \ell$, then $d_k(S) \leq \sigma^k \leq \sigma^{\ell-1} \leq \sigma^{\log_\sigma d} = d < \Delta k$ holds as claimed. Thus, we henceforth assume $k \geq \ell$ and consider the smallest integer $\tau \in \mathbb{Z}_{\geq 0}$ such that $2^{\tau+1}\ell \geq k$; observe that $2^\tau \ell < k$. The number of length-k substrings starting within a prefix $S_1 \cdots S_{\tau-1}$ of S, with $S_i \in \mathsf{F}_i^d$, does not exceed $\sum_{i=0}^{\tau-1} |S_i| \leq \sum_{i=0}^{\tau-1} d(1+\ell+2^{i+2}\ell) \leq \sum_{i=0}^{\tau-1} d\ell(2+2^{i+2}) \leq 6d\ell 2^\tau \leq 6dk$. The remaining length-$k$ substrings occur within $\mathtt{0}^k \mathsf{r}(j) \mathtt{1} 0^{k-1}$ for some $j \in [0\,..\,d)$. The number of such substrings does not exceed $d(k+\ell+2) \leq 4dk$. Overall, $d_k(S) \leq 10dk \leq \Delta k$ holds as claimed.

Next, observe that the family of length-n prefixes of strings in F is of size at least $\prod_{i=0}^{\tau-1} |\mathsf{F}_i|^d$ as long as $n \geq 6d\ell 2^\tau$. Consequently, the considered worst-case entropy is at least

$$\sum_{i=0}^{\tau-1} d \log |\mathsf{F}_i| \geq d \sum_{i=0}^{\tau-1} \log(d\ell \cdot 2^{i+1}) = \Omega(d\tau(\log(d\ell) + \tau)).$$

Define $\tau = \lfloor \log \frac{n}{6d\ell} \rfloor$ and observe that $\frac{n}{6d\ell} \geq \frac{n \log \sigma}{12d \log d} \geq \frac{10n \log \sigma}{12\Delta \log n} \geq \frac{5}{3}$ implies $\tau = \Omega(\log \frac{n \log \sigma}{\Delta \log n})$. Thus, the entropy is at least $\Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n} \cdot (\log \frac{\Delta \log \Delta}{\log \sigma} + \log \frac{n \log \sigma}{\Delta \log n})) = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n} \log \frac{n \log \Delta}{\log n}) = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n} \log n)$. □

Considering that we can encode any string in $O(z) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and that we have now proved that this space is necessary for some string families, we have that $\Theta(\delta \log \frac{n \log \sigma}{\delta \log n})$ is a tight bound on the space needed to represent strings. This can be expressed with the following direct-converse information-theoretic theorem:

Theorem IV.8. *The following statements hold:*

1) *Let $S \in \Sigma^n$ be any string of length n over alphabet Σ of size $\sigma \leq n^{O(1)}$ and having measure $\delta = \delta(S)$. Then, there exists a coder that compresses S using a number of bits bounded by $\alpha \cdot \delta \log \frac{n \log \sigma}{\delta \log n} \log n$, where α is a universal constant.*
2) *There exists a constant β such that, for every length n, alphabet Σ of size σ, and measure $\Delta \in [14\sigma\,..\,\frac{n \log \sigma}{2 \log n}]$, there exists a family of strings $S \in \Sigma^n$, of measure $\delta \leq \Delta$ where some members need $\beta \cdot \Delta \log \frac{n \log \sigma}{\Delta \log n} \log n$ bits to be encoded.*

Proof. (1) From Lemma II.4, we have that $z(S) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$. Assuming a polynomial alphabet size $\sigma \leq n^{O(1)}$, a naive Lempel–Ziv encoding uses $O(\log n)$ bits per each of the $z(S)$ phrases: $O(\log n)$ bits to encode either an explicit alphabet symbol or the phrase source, and $\log n$ bits to encode the phrase length in the second case. We conclude that there exists a universal constant α such that this encoding always uses at most $\alpha \cdot \delta \log \frac{n \log \sigma}{\delta \log n} \log n$ bits to compress S.
(2) Follows from Lemma IV.7. □

V. Bounds on grammar sizes

In this section, we study the relation between δ and the sizes of the smallest context-free grammar and run-length context-free grammar generating S, denoted by g and g_{rl}, respectively. As observed in Section II, Lemma II.4 implies $g_{rl} \leq g = O(\delta \log^2 \frac{n}{\delta})$. Our first contribution is a lower bound construction for g: for every length n and value $\delta \in [2\,..\,n]$, we construct a string S satisfying $g = \Omega(\delta \log^2 \frac{n}{\delta} / \log \log \frac{n}{\delta})$.

Rather surprisingly, the situation for run-length context-free grammars is very different: we prove that $g_{rl} = O(\delta \log \frac{n}{\delta})$, which is tight due to Theorem III.3 and $g_{rl} = \Omega(\gamma)$. Our argument is constructive and we derive a randomized algorithm that, in $O(n)$ expected time, constructs a run-length context-free grammar of size $O(\delta \log \frac{n}{\delta})$ generating a given string S.

A. A lower bound on grammar size

For convenience, among all context-free grammars generating a single string S, we only consider *straight-line programs* (SLPs), where the right-hand side of each production is of size exactly 2. We denote by $\mathsf{slp}(S)$ the minimum number of symbols (terminals and non-terminals) in any SLP generating S. Each context-free grammar generating S can be transformed to an SLP, and thus $\mathsf{slp}(S) = \Theta(g(S))$ holds for every string S.

Our construction relies on the family $\mathcal{S}_n^{\mathsf{p}}$ of Definition IV.2 and the following consequence of Lemma IV.3.

Corollary V.1. *For every integer $n \geq 1$, there exists a string $S_n^{\mathsf{p}} \in \mathcal{S}_n^{\mathsf{p}}$ satisfying $\mathsf{slp}(S_n^{\mathsf{p}}) = \Omega(\log^2 n / \log \log n)$.*

Proof. Recall that each string $S \in \mathcal{S}^{\mathsf{p}}_n$ is binary and therefore can be encoded in $O(\mathsf{slp}(S)\log \mathsf{slp}(S))$ bits: every symbol is assigned a $\lceil \log \mathsf{slp}(S)\rceil$-bit identifier (with 0 and 1 reserved for $\mathtt{a}$ and $\mathtt{b}$, respectively) so that each production is encoded using $O(\log \mathsf{slp}(S))$ bits. At the same time, Lemma IV.3 proves that every encoding distinguishing members of $\mathcal{S}^{\mathsf{p}}_n$ requires $\Omega(\log^2 n)$ bits. In particular, our SLP-based encoding uses $\Omega(\log^2 n)$ bits for some string $S^{\mathsf{p}}_n \in \mathcal{S}^{\mathsf{p}}_n$. This string satisfies $\mathsf{slp}(S^{\mathsf{p}}_n)\log \mathsf{slp}(S^{\mathsf{p}}_n) = \Omega(\log^2 n)$, and therefore $\mathsf{slp}(S^{\mathsf{p}}_n) = \Omega(\log^2 n/\log\log n)$. □

Note that the proof of Corollary V.1 does not apply to run-length context-free grammars because a production of the form $A \to B^t$ may need $\Theta(\log t)$ bits to represent the exponent t. In fact, since $\delta(S^{\mathsf{p}}_n) = 2$ holds for $n \geq 3$, the upper bound $g_{rl} = O(\delta \log \frac{n}{\delta})$ proved in the next section shows that $g_{rl}(S^{\mathsf{p}}_n) = O(\log n)$; generalizing Corollary V.1 to run-length context-free grammars is therefore inherently impossible.

Corollary V.1 shows that $g = \Omega(\delta \log^2 \frac{n}{\delta}/\log\log \frac{n}{\delta})$ is possible for $\delta = 2$ and any length $n \geq 3$. We generalize this construction to arbitrary $\delta \in [2\,..\,n]$ as in the proof of Lemma IV.4. Our argument requires the following auxiliary lemma.

Lemma V.2. *Consider a string $S = L \cdot R$. If the alphabets $\Sigma(L)$ of L and $\Sigma(R)$ of R are disjoint, then $\mathsf{slp}(S) \geq \mathsf{slp}(L) + \mathsf{slp}(R)$.*

Proof. Our goal is to construct SLPs generating L and R with $\mathsf{slp}(S)$ symbols in total. Let us fix an SLP generating S with $\mathsf{slp}(S)$ symbols. To generate L (R), we start from the SLP of S and remove every terminal not in $\Sigma(L)$ ($\Sigma(R)$). Then, transitively remove nonterminals with empty right-hand sides and suppress nonterminals with right-hand sides of length 1.

Now consider a rule $A \to BC$ in the SLP of S: either C disappears in the SLP that generates L, or B disappears in the SLP that generates R, or both. In all cases, the rule survives in at most one of the two SLPs; therefore the total number of symbols in the SLPs we build for L and R is at most $\mathsf{slp}(S)$. Consequently, $\mathsf{slp}(L) + \mathsf{slp}(R) \leq \mathsf{slp}(S)$. □

Theorem V.3. *For every length n and integer value $\delta \in [2\,..\,n]$, there is a string $S[1\,..\,n]$ with $g = \Omega(\delta \log^2 \frac{n}{\delta}/\log\log \frac{n}{\delta})$.*

Proof. As in the proof of Lemma IV.4, let $m \geq 1$, $n \geq 4m-1$, and $n - m + 1 = \sum_{i=1}^{m} n_i$, where $n_i = \Omega(\frac{n}{m})$ and $n_i \geq 3$. Moreover, let $S = S^{(1)}\,\$_1\,S^{(2)}\,\$_2 \cdots \$_{m-1}\,S^{(m)}$, where $S^{(i)}$ is obtained from the string $S^{\mathsf{p}}_{n_i} \in \mathcal{S}^{\mathsf{p}}_{n_i}$ of Corollary V.1 by replacing every $\mathtt{a}$ with $\mathtt{a}_i$ and every $\mathtt{b}$ with $\mathtt{b}_i$. Note that S belongs to the family constructed in the proof of Lemma IV.4, and thus $\delta(S) = 3m - 1$. Furthermore, since the alphabets of strings $S^{(i)}$ (for $i \in [1\,..\,m]$) and $\$_i$ (for $i \in [1\,..\,m)$) are pairwise disjoint, Lemma V.2 yields $\mathsf{slp}(S) \geq \sum_{i=1}^{m} \mathsf{slp}(S^{(i)}) = \sum_{i=1}^{m} \mathsf{slp}(S^{\mathsf{p}}_{n_i}) = \Omega(m \log^2 \frac{n}{m}/\log\log \frac{n}{m})$.

This construction proves the theorem for $\delta = 3m - 1$ and $n \geq 4m-1$. The case where $\delta < \frac{3}{4}n$ is not of the form $3m-1$ is handled as in the proof of Theorem III.3. Finally, we note that if $\delta \geq \frac{3}{4}n = \Omega(n)$, then the claim reduces to $g = \Omega(\delta)$ and therefore follows from Lemma II.3 due to $\gamma = O(g)$. □

B. An upper bound on run-length grammar size

In this section, we prove that every string $S \in \Sigma^n$ can be generated using a run-length context-free grammar of size $O(\delta \log \frac{n}{\delta})$. We obtain our grammar in the process of building a locally consistent parsing on top of S. Our parsing is based on the recompression technique by Jeż [16], who used it to design a simple $O(n)$-time algorithm constructing a (standard) context-free grammar of size $O(z \log \frac{n}{z})$. More specifically, we rely on a very recent *restricted recompression* by Kociumaka et al. [42], which delays processing symbols generating long substrings. Birenzwige et al. [43] applied a similar idea to transform the locally consistent parsing of Sahinalp and Vishkin [44].

1) Run-length grammar construction via restricted recompression: Both recompression and restricted recompression, given a string $S \in \Sigma^+$, construct a sequence of strings $(S_k)_{k=0}^{\infty}$ over the alphabet $\mathcal{A}$ of *symbols* defined as the least fixed point of the following equation:

$$\mathcal{A} = \Sigma \cup (\mathcal{A} \times \mathcal{A}) \cup (\mathcal{A} \times \mathbb{Z}_{\geq 2}).$$

Symbols in $\mathcal{A} \setminus \Sigma$ are *non-terminals* with *productions* $(A_1, A_2) \to A_1A_2$ for $(A_1, A_2) \in \mathcal{A} \times \mathcal{A}$ and $(A_1, m) \to A_1^m$ for $(A_1, m) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}$. With any $A \in \mathcal{A}$ designated as the start symbol, this yields a run-length straight-line program (RLSLP). The following *expansion* function $\exp : \mathcal{A} \to \Sigma^+$ retrieves the string generated by this RLSLP:

$$\exp(A) = \begin{cases} A & \text{if } A \in \Sigma, \\ \exp(A_1) \cdot \exp(A_2) & \text{if } A = (A_1, A_2) \in \mathcal{A}^2, \\ \exp(A_1)^m & \text{if } A = (A_1, m) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}. \end{cases}$$

Intuitively, $\mathcal{A}$ forms a *universal* RLSLP: for every RLSLP with symbols $\mathcal{S}$ and terminals $\Sigma \subseteq \mathcal{S}$, there is a unique homomorphism $f : \mathcal{S} \to \mathcal{A}$ such that $f(A) = A$ if $A \in \Sigma$, $f(A) \to f(A_1)f(A_2)$ if $A \to A_1A_2$, and $f(A) \to f(A_1)^m$ if $A \to A_1^m$. As a result, $\mathcal{A}$ provides a convenient formalism to argue about procedures generating RLSLPs.

The main property of strings $(S_k)_{k=0}^{\infty}$ generated using (restricted) recompression is that $\exp(S_k) = S$ holds for all $k \in \mathbb{Z}_{\geq 0}$, where $\exp : \mathcal{A} \to \Sigma^+$ is lifted to $\exp : \mathcal{A}^* \to \Sigma^*$ by setting $\exp(A_1 \cdots A_a) = \exp(A_1) \cdots \exp(A_a)$ for $A_1 \cdots A_a \in \mathcal{A}^*$. The subsequent strings S_k, starting from $S_0 = S$, are obtained by alternate applications of the following two functions which decompose a string $T \in \mathcal{A}^+$ into *blocks* and then *collapse* blocks into appropriate symbols. In Definition V.4, all blocks of length at least 2 are maximal blocks of the same symbol, and they are collapsed to symbols in $\mathcal{A} \times \mathbb{Z}_{\geq 2}$. In Definition V.5, there are no blocks of length more than 2, and all blocks of length 2 are collapsed to symbols in $\mathcal{A} \times \mathcal{A}$.

Definition V.4 (Restricted run-length encoding [42])**.** *Given $T \in \mathcal{A}^+$ and $\mathcal{B} \subseteq \mathcal{A}$, we define $\mathsf{rle}_{\mathcal{B}}(T) \in \mathcal{A}^+$ to be the string obtained as follows by decomposing T into blocks and collapsing these blocks:*

1) *For $i \in [1\,..\,|T|)$, place a* block boundary *between $T[i]$ and $T[i+1]$ unless $T[i] = T[i+1] \in \mathcal{B}$.*
2) *Replace each block $T[i\,..\,i+m) = A^m$ of length $m \geq 2$ with a symbol $(A, m) \in \mathcal{A}$.*

Definition V.5 (Restricted pair compression [42]). *Given $T \in \mathcal{A}^+$ and disjoint sets $\mathcal{L}, \mathcal{R} \subseteq \mathcal{A}$, we define $\mathsf{pc}_{\mathcal{L},\mathcal{R}}(T) \in \mathcal{A}^+$ to be the string obtained as follows by decomposing T into blocks and collapsing these blocks:*

1) *For $i \in [1\,..\,|T|)$, place a* block boundary *between $T[i]$ and $T[i+1]$ unless $T[i] \in \mathcal{L}$ and $T[i+1] \in \mathcal{R}$.*
2) *Replace each block $T[i\,..\,i+1]$ of length 2 with a symbol $(T[i], T[i+1]) \in \mathcal{A}$.*

The original recompression uses $\mathsf{rle}_{\mathcal{B}}$ with $\mathcal{B} = \mathcal{A}$ and $\mathsf{pc}_{\mathcal{L},\mathcal{R}}$ with $\mathcal{A} = \mathcal{L} \cup \mathcal{R}$. In the restricted version, symbols in $\mathcal{A} \setminus \mathcal{B}$ and $\mathcal{A} \setminus (\mathcal{L} \cup \mathcal{R})$, respectively, are forced to form length-1 blocks. In the kth round of restricted recompression, this mechanism is applied to symbols A whose expansion $\exp(A)$ is longer than a certain threshold ℓ_k.

With this intuition, we are now ready to formally define the sequence $(S_k)_{k=0}^{\infty}$ constructed through restricted recompression.

Construction V.6 (Restricted recompression [42]). *Given a string $S \in \Sigma^+$, the strings S_k for $k \in \mathbb{Z}_{\geq 0}$ are constructed as follows, based on $\ell_k := (\frac{8}{7})^{\lceil k/2 \rceil - 1}$ and $\mathcal{A}_k := \{A \in \mathcal{A} : |\exp(A)| \leq \ell_k\}$:*

- *If $k = 0$, then $S_k = S$.*
- *If $k > 0$ is odd, then $S_k = \mathsf{rle}_{\mathcal{A}_k}(S_{k-1})$.*
- *If $k > 0$ is even, then $S_k = \mathsf{pc}_{\mathcal{L}_k,\mathcal{R}_k}(S_{k-1})$, where $\mathcal{A}_k = \mathcal{L}_k \cup \mathcal{R}_k$ is a uniformly random partition into disjoint classes.*

It is easy to see that $\exp(S_k) = S$ indeed holds for all $k \in \mathbb{Z}_{\geq 0}$. As we argue below, almost surely (with probability 1), there exists $h \in \mathbb{Z}_{\geq 0}$ such that $|S_h| = 1$. In particular, an RLSLP generating S can be obtained by setting $S_h[1]$ as the starting symbol of the RLSLP derived from by $\mathcal{A}$. While this RLSLP contains infinitely many symbols, it turns out that we can remove symbols that do not occur in any string S_k. Formally, for each $k \in \mathbb{Z}_{\geq 0}$, let us define the family $\mathcal{S}_k := \{S_k[j] : j \in [1\,..\,|S_k|]\} \subseteq \mathcal{A}$ of symbols occurring in S_k. Observe that each symbol in S_0 belongs to Σ and, for $k \in \mathbb{Z}_+$, each symbol in S_k was either copied from S_{k-1} or obtained by collapsing a block in S_{k-1}. Consequently, the family $\mathcal{S} := \bigcup_{k=0}^{\infty} \mathcal{S}_k$ satisfies $\mathcal{S} \subseteq \Sigma \cup (\mathcal{S} \times \mathcal{S}) \cup (\mathcal{S} \times \mathbb{Z}_{\geq 2})$. In other words, for every non-terminal $A \in \mathcal{S}$, the symbols on the right-hand side of the production of A also belong to $\mathcal{S}$. Hence, the remaining symbols can indeed be removed from the RLSLP generating S. As the size of resulting run-length context-free grammar is proportional to $|\mathcal{S}|$, we are left with the task of bounding $\mathbb{E}[|\mathcal{S}|]$.

2) Analysis of the grammar size: Our argument relies on several properties of restricted recompression proved in [42]. Due to the current status of [42] being an unpublished manuscript, the proofs are provided in Section A for completeness.

Fact V.7 ([42]). *For every $k \in \mathbb{Z}_{\geq 0}$, if $\exp(x) = \exp(x')$ holds for two fragments of S_k, then $x = x'$.*

Corollary V.8 ([42]). *For every odd $k \in \mathbb{Z}_{\geq 0}$, there is no $j \in [1\,..\,|S_k|)$ such that $S_k[j] = S_k[j+1] \in \mathcal{A}_{k+1}$.*

Recall that $\exp(S_k) = S$ for every $k \in \mathbb{Z}_{\geq 0}$. Hence, for every $j \in [1\,..\,|S_k|]$, we can associate $S_k[j]$ with a fragment $S(|\exp(S_k[1\,..\,j))|\,..\,|\exp(S_k[1\,..\,j])|] = \exp(S_k[j])$; these fragments are called *phrases* (of S) induced by S_k. We also define a set B_k of *phrase boundaries* induced by S_k:

$$B_k = \{|\exp(S_k[1\,..\,j])| : j \in [1\,..\,|S_k|)\}.$$

Lemma V.9 ([42]). *Let $\alpha \in \mathbb{Z}_{\geq 1}$ and let $i, i' \in [\alpha\,..\,n-\alpha]$ be such that $S(i-\alpha\,..\,i+\alpha] = S(i'-\alpha\,..\,i'+\alpha]$. For every $k \in \mathbb{Z}_{\geq 0}$, if $\alpha \geq 16\ell_k$, then $i \in B_k \iff i' \in B_k$.*

Lemma V.10 ([42]). *For every $k \in \mathbb{Z}_{\geq 0}$, we have $\mathbb{E}[|S_k|] < 1 + \frac{4n}{\ell_{k+1}}$.*

Lemma V.10 can be used to confirm that almost surely $|S_k| = 1$ holds for some $k \in \mathbb{Z}_{\geq 0}$.

Corollary V.11. *With probability 1, there exists $k \in \mathbb{Z}_{\geq 0}$ such that $|S_k| = 1$.*

Proof. For a proof by contradiction, suppose that $\varepsilon := \Pr[\min_{k=0}^{\infty} |S_k| > 1] > 0$. In particular, this yields $\mathbb{E}[|S_k|] \geq 1 + \varepsilon$ for every $k \in \mathbb{Z}_{\geq 0}$. However, Lemma V.10 implies $\lim_{k\to\infty} |S_k| \leq 1 + \lim_{k\to\infty} \frac{4n}{\ell_{k+1}} = 1$, a contradiction. □

Our main goal is to prove that $\mathbb{E}[|\mathcal{S}|] = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ (Corollary V.17). As a stepping stone, we show that $\mathbb{E}[|\mathcal{A}_{k+1} \cap \mathcal{S}_k|] = O(\delta)$ holds for all $k \in \mathbb{Z}_{\geq 0}$ (Lemma V.14). The restriction to symbols in $\mathcal{A}_{k+1}$ is not harmful because every symbol in S_k that does not belong to $\mathcal{A}_{k+1}$ forms a length-1 block that gets propagated to S_{k+1}. The strategy behind the proof of Lemma V.14 is to consider the phrases induced by the leftmost occurrences of all symbol in $\mathcal{A}_{k+1} \cap \mathcal{S}_k$. Using Lemma V.9, we construct $O(\delta)$ fragments of S of total length $O(\ell_k \delta)$ guaranteed to overlap all these phrases, and we show that these fragments in expectation overlap $O(\delta)$ phrases induced by S_k. The latter claim is a consequence of the following generalization of Lemma V.10.

Lemma V.12. *For every $k \in \mathbb{Z}_{\geq 0}$ and every interval $I \subseteq [1\,..\,n)$, we have*

$$\mathbb{E}[|B_k \cap I|] < 1 + \tfrac{4|I|}{\ell_{k+1}}.$$

Proof. We proceed by induction on k. For $k = 0$, we have $|B_k \cap I| = |I| < 1 + 4|I| = 1 + \frac{4|I|}{\ell_1}$. If k is odd, we note that $B_k \subseteq B_{k-1}$ and therefore $\mathbb{E}[|B_k \cap I|] \leq \mathbb{E}[|B_{k-1} \cap I|] < 1 + \frac{4|I|}{\ell_k} = 1 + \frac{4|I|}{\ell_{k+1}}$. Thus, it remains to consider even values $k > 0$.

Claim V.13. *If $k > 0$ is even, then, conditioned on any fixed S_{k-1}, we have $\mathbb{E}\left[|B_k \cap I| \;\middle|\; S_{k-1}\right] < \frac{1}{4} + \frac{|I|}{2\ell_k} + \frac{3}{4}|B_{k-1} \cap I|$.*

Proof. Let us define

$$J = \{j \in [1\,..\,|S_{k-1}|) : S_{k-1}[j] \notin \mathcal{A}_k \text{ or } S_{k-1}[j+1] \notin \mathcal{A}_k\},$$
$$J_I = \{j \in J : |\exp(S_{k-1}[1\,..\,j])| \in I\} \subseteq B_{k-1} \cap I.$$

Since $A \notin \mathcal{A}_k$ yields $|\exp(A)| > \ell_k$, we have $|J_I| < 1 + \frac{2|I|}{\ell_k}$. Moreover, observe that if $j \in [1\,..\,|S_{k-1}|) \setminus J$, then $S_{k-1}[j]$

and $S_{k-1}[j+1]$ are, by Corollary V.8, distinct symbols in $\mathcal{A}_k$. Consequently,

$$\Pr[S_{k-1}[j] \in \mathcal{L}_k \text{ and } S_{k-1}[j+1] \in \mathcal{R}_k] = \tfrac{1}{4}.$$

Thus, the probability that $\mathsf{pc}_{\mathcal{L}_k,\mathcal{R}_k}(S_{k-1})$ places a block boundary after position $j \in [1\,..\,|S_{k-1}|) \setminus J$ is $\frac{3}{4}$. Therefore,

$$\begin{aligned}\mathbb{E}\left[|B_k \cap I| \;\middle|\; S_{k-1}\right] &= |J_I| + \tfrac{3}{4}(|B_{k-1} \cap I| - |J_I|)\\ &= \tfrac{1}{4}|J_I| + \tfrac{3}{4}|B_{k-1} \cap I| < \tfrac{1}{4} + \tfrac{|I|}{2\ell_k} + \tfrac{3}{4}|B_{k-1} \cap I|. \quad \square\end{aligned}$$

Since the partition $\mathcal{A}_k = \mathcal{L}_k \cup \mathcal{R}_k$ is independent of S_{k-1}, Claim V.13 and the inductive assumption yield

$$\begin{aligned}\mathbb{E}[|B_k \cap I|] &< \tfrac{1}{4} + \tfrac{|I|}{2\ell_k} + \tfrac{3}{4}\mathbb{E}[|B_{k-1} \cap I|] < \tfrac{1}{4} + \tfrac{|I|}{2\ell_k} + \tfrac{3}{4} + \tfrac{3|I|}{\ell_k}\\ &= 1 + \tfrac{7|I|}{2\ell_k} = 1 + \tfrac{4|I|}{\ell_{k+1}}. \quad \square\end{aligned}$$

Next, we apply Lemmas V.9 and V.12 to bound the expected size of $\mathcal{S}_k \cap \mathcal{A}_{k+1}$.

Lemma V.14. *For every $k \in \mathbb{Z}_{\geq 0}$ and every string $S \in \Sigma^+$ with measure δ, we have $\mathbb{E}[|\mathcal{S}_k \cap \mathcal{A}_{k+1}|] = O(\delta)$.*

Proof. Let us fix integers $\alpha \geq 16\ell_k$ and $m = 2\alpha + \lfloor \ell_{k+1} \rfloor$. Moreover, define

$$L = \left\{ i \in [0\,..\,n-m] : \begin{array}{c} S(i\,..\,i+m] = S(i'\,..\,i'+m] \\ \text{for some } i' \in [0\,..\,i) \end{array} \right\}$$

and

$$P = \{i \in [1\,..\,n] : i - \alpha \notin L\}.$$

Claim V.15. *We have $|\mathcal{S}_k \cap \mathcal{A}_{k+1}| \leq 1 + |B_k \cap P|$.*

Proof. Let $S_k[j]$ be the leftmost occurrence in S_k of $A \in \mathcal{A}_{k+1} \cap \mathcal{S}_k$. Moreover, let $p = |\exp(S_k[1\,..\,j))|$ and $q = |\exp(S_k[1\,..\,j])|$ so that $S(p\,..\,q] = \exp(A)$ is the phrase induced by S_k corresponding to $S_k[j]$.

We shall prove that $j = 1$ or $p \in B_k \cap P$. This will complete the proof of the claim because distinct symbols A yield distinct positions j and p.

For a proof by contradiction, suppose that $j \in (1\,..\,|S_k|)$ yet $p \notin B_k \cap P$. Since $p \in B_k$ holds due to $j > 1$, we derive $p \notin P$, which implies $p - \alpha \in L$. Consequently, there is a position $p' \in [\alpha\,..\,p)$ such that $S(p-\alpha\,..\,p-\alpha+m] = S(p'-\alpha\,..\,p'-\alpha+m]$. In particular, $S(p-\alpha\,..\,p+\alpha] = S(p'-\alpha\,..\,p'+\alpha]$, so Lemma V.9 yields $p' \in B_k$. Similarly, due to $q - p = |\exp(A)| \leq \lfloor \ell_{k+1} \rfloor = m - 2\alpha$, we have $S(q-\alpha\,..\,q+\alpha] = S(q'-\alpha\,..\,q'+\alpha]$ for $q' := p' + |\exp(A)|$, and therefore $q' \in B_k$ holds due to $q \in B_k$. Lemma V.9 further implies $B_k \cap (p'\,..\,q') = \emptyset = B_k \cap (p\,..\,q)$. Consequently, $S(p'\,..\,q']$ is a phrase induced by S_k, and, since $p' < p$, it corresponds to $S_k[j']$ for some $j' < j$. By Fact V.7, we have $S_k[j'] = S_k[j] = A$, which contradicts the choice of $S_k[j]$ as the leftmost occurrence of A in S_k. $\square$

Consequently, it remains to prove that $\mathbb{E}[|B_k \cap P|] = O(\delta)$. For this, we characterize P as follows.

Claim V.16. *The set P can be covered by $O(\delta)$ intervals of total length $O(m\delta)$.*

Proof. Note that $P' = \bigcup_{i \in P}(i-\alpha\,..\,i+m-\alpha]$ is a superset of P. Moreover, each position $j \in P' \cap [m\,..\,n-m]$ is contained in the leftmost occurrence of a length-m substring of S and then $S(j-m\,..\,j+m]$ is the leftmost occurrence of a length-$2m$ substring of S. Consequently, $|P' \cap [m\,..\,n-m]| \leq 2m\delta$. Since $P' \setminus [m\,..\,n-m] = (1-\alpha\,..\,m) \cup (n-m\,..\,n+m-\alpha]$ is of size $O(m)$, we conclude that $|P'| = O(m\delta)$. Next, recall that P' is a union of length-m integer intervals. Merging overlapping intervals, we get a decomposition into disjoint intervals of length at least m. The number of intervals does not exceed $\frac{1}{m}|P'| = O(\delta)$. $\square$

Now, let $\mathcal{I}$ be the family of intervals covering P obtained using Claim V.16. For each $I \in \mathcal{I}$, Lemma V.12 implies $\mathbb{E}[|B_k \cap I|] \leq 1 + \frac{4|I|}{\ell_{k+1}}$. By linearity of expectation and the bounds in Claim V.16, this yields the claimed result:

$$\mathbb{E}[|B_k \cap P|] \leq |\mathcal{I}| + \tfrac{4}{\ell_{k+1}} \sum_{I \in \mathcal{I}} |I| = O(\delta + \tfrac{4\delta m}{\ell_{k+1}}) = O(\delta). \quad \square$$

The proof of our main bound $\mathbb{E}[|\mathcal{S}|] = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ combines Lemmas V.10 and V.14.

Corollary V.17. *For every string $S \in [0\,..\,\sigma)^n$ of measure δ, we have $\mathbb{E}[|\mathcal{S}|] = O(\delta \log \frac{n \log \sigma}{\delta \log n})$.*

Proof. Note that $|\mathcal{S}| \leq 1 + \sum_{k=0}^{\infty} |\mathcal{S}_k \setminus \mathcal{S}_{k+1}|$. We combine three upper bounds on $\mathbb{E}[|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|]$: a naive one as well as two bounds following from Lemmas V.10 and V.14, respectively.

First, we note that $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \leq |[0\,..\,\sigma)^{\leq \ell_k}| = O(\sigma^{\ell_k})$. Next, we observe that Construction V.6 guarantees $\mathcal{S}_k \setminus \mathcal{S}_{k+1} \subseteq \mathcal{S}_k \cap \mathcal{A}_{k+1}$ and thus $\mathbb{E}[|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|] \leq \mathbb{E}[|\mathcal{S}_k \cap \mathcal{A}_{k+1}|] = O(\delta)$ holds due to Lemma V.14. Furthermore, we note that $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| = 0$ if $|S_k| = 1$ and $|\mathcal{S}_k \setminus \mathcal{S}_{k+1}| \leq |S_k|$ otherwise. Consequently, Markov inequality and Lemma V.10 yield

$$\begin{aligned}\mathbb{E}[|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|] &\leq \mathbb{E}[|S_k|] - \mathbb{P}[|S_k| = 1]\\ &= \mathbb{E}[|S_k|] - 1 + \mathbb{P}[|S_k| \geq 2]\\ &= \mathbb{E}[|S_k|] - 1 + \mathbb{P}[|S_k| - 1 \geq 1]\\ &\leq \mathbb{E}[|S_k|] - 1 + \mathbb{E}[|S_k| - 1]\\ &= 2\mathbb{E}[|S_k|] - 2\\ &\leq \tfrac{8n}{\ell_{k+1}}.\end{aligned}$$

Thus, $\mathbb{E}[|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|] = O((\frac{7}{8})^{k/2} n)$.

We apply the first bound if $k \leq 2\log_{8/7} \log_\sigma \delta$ and the third bound if $k \geq 2\log_{8/7} \frac{n}{\delta}$. In the intermediate cases, we use the second bound. This yields

$$\begin{aligned}\sum_{k=0}^{\infty} \mathbb{E}[|\mathcal{S}_k \setminus \mathcal{S}_{k+1}|] &= \sum_{k=0}^{\lfloor 2\log_{8/7} \log_\sigma \delta \rfloor} O(\sigma^{\ell_k})\\ &\quad + (2\log_{8/7} \tfrac{n}{\delta} - 2\log_{8/7} \log_\sigma \delta) \cdot O(\delta)\\ &\quad + \sum_{k=\lceil 2\log_{8/7} \frac{n}{\delta} \rceil}^{\infty} O\left((\tfrac{7}{8})^{k/2}\delta\right)\\ &= \sum_{i=0}^{\lfloor \log_\sigma \delta \rfloor} O(\sigma^i) + O(\delta \log \tfrac{n \log \sigma}{\delta \log \delta}) + \sum_{i=0}^{\infty} O\left((\tfrac{7}{8})^{i/2}\delta\right)\\ &= O(\delta) + O(\delta \log \tfrac{n \log \sigma}{\delta \log \delta}) + O(\delta)\\ &= O(\delta \log \tfrac{n \log \sigma}{\delta \log \delta}).\end{aligned}$$

Consequently, $\mathbb{E}[|\mathcal{S}|] = 1 + O(\delta \log \frac{n \log \sigma}{\delta \log \delta}) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ holds as claimed. □

Finally, we note that Corollaries V.11 and V.17 allow bounding the size of the smallest run-length grammar generating S.

Theorem V.18. *Every string $S \in [0\,..\,\sigma)^n$ satisfies $g_{rl}(S) = O(\delta \log \frac{n \log \sigma}{\delta \log n})$.*

Proof. We apply Construction V.6 on top of the given string S. By Corollaries V.11 and V.17, the random choices within Construction V.6 can be fixed so that $|\mathcal{S}| = O(\delta \log \frac{n \log \sigma}{\delta \log n})$ and $|S_k| = 1$ holds for sufficiently large k. We build a run-length grammar with symbols $\mathcal{S}$. Each symbol $A \in \Sigma$ is a terminal symbol, each symbol $A = (A_1, A_2) \in \mathcal{A} \times \mathcal{A}$ is associated with a production $A \to A_1 A_2$, and each symbol $A = (A_1, m) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}$ is associated with a production $A \to A_1^m$. It easy to see that the auxiliary symbols A_1, A_2 belong to $\mathcal{S}$ and that the expansion of each symbol A (within the grammar) is $\exp(A)$. In particular, if we set the only symbol of S_k for sufficiently large k as the starting symbol, then the grammar generates S. □

C. Efficient construction of a small run-length grammar

In this section, we convert the proof of Theorem V.18 to a fast algorithm generating a run-length grammar of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$.

Proposition V.19. *There exists a randomized algorithm that, given a string $S \in [0\,..\,\sigma)^n$ of measure $\delta = \delta(S)$, in $O(n)$ expected time constructs a run-length grammar of expected size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ generating S.*

Proof. The algorithm simulates Construction V.6 constructing $\mathcal{S}$, which can be interpreted as a grammar generating S (see the proof of Theorem V.18). Each symbol $A \in \Sigma$ is stored explicitly, each symbol $A = (A_1, A_2) \in \mathcal{A} \times \mathcal{A}$ keeps pointers to A_1 and A_2, and each symbol $A = (A_1, m) \in \mathcal{A} \times \mathbb{Z}_{\geq 2}$ keeps m and a pointer to A_1. Additionally, each symbol A is augmented with $|\exp(A)|$ and an *identifier* $\mathsf{id}_k(A)$ for every k such that $A \in \mathcal{S}_k$, where $\mathsf{id}_k : \mathcal{S}_k \to [1\,..\,|\mathcal{S}_k|]$ is a bijection. The inverse mappings id_k^{-1} are implemented as arrays.

The string $S_0 = S$ is given as input. In order to construct $\mathcal{S}_0$ and id_0, we sort the characters of S and assign them consecutive positive integer identifiers. This step takes $O(n)$ time due to the assumption $\sigma = n^{O(1)}$.

In order to construct S_k, $\mathcal{S}_k$, and id_k, we process S_{k-1} depending on the parity of k. If k is odd, we scan S_{k-1} from left to right outputting subsequent symbols of S_k. Initially, each symbol $A \in \mathcal{S}_k$ is represented as $(\mathsf{id}_{k-1}(A_1), m)$ (if $A = (A_1, m) \in \mathcal{S}_k \setminus \mathcal{S}_{k-1}$) or as $\mathsf{id}_{k-1}(A)$ (otherwise). Suppose that $S_{k-1}[j\,..]$ is yet to be processed. If $|\exp(S_{k-1}[j])| > \ell_k$ or $S_{k-1}[j] \neq S_{k-1}[j+1]$, we output $\mathsf{id}_{k-1}(S_{k-1}[j])$ as the next symbol of S_k and continue processing $S_{k-1}[j+1\,..]$. Otherwise, we determine the maximum integer $m \geq 2$ such that $S_{k-1}[j'] = S_{k-1}[j]$ for $j' \in [j\,..\,j+m)$, output $(\mathsf{id}_{k-1}(S_{k-1}[j]), m)$ as the next symbol of S_k, and continue processing $S_{k-1}[j+m\,..]$. (By Fact V.7, $(S_{k-1}[j], m) \notin \mathcal{S}_{k-1}$ in this case.) Note that the symbols in $\mathcal{S}_k$ are initially represented as elements of $[1\,..\,|S_{k-1}|] \cup [1\,..\,|S_{k-1}|]^2$. Sorting these values allows constructing symbols in $\mathcal{S}_k \setminus \mathcal{S}_{k-1}$ and the identifier function id_k in $O(|S_{k-1}|)$ time.

If k is even, we first randomly partition $\mathcal{A}_k$ into $\mathcal{L}_k$ and $\mathcal{R}_k$. Technically, this step consists in iterating over symbols $A \in \mathcal{S}_{k-1}$ and appropriately marking A if $|\exp(A)| \leq \ell_k$. Next, we scan S_{k-1} from left to right outputting subsequent symbols of S_k. Initially, each symbol A in S_k is represented as $(\mathsf{id}_{k-1}(A_1), \mathsf{id}_{k-1}(A_2))$ (if $A = (A_1, A_2) \notin \mathcal{S}_{k-1}$) or as $\mathsf{id}_{k-1}(A)$ (otherwise). Suppose that $S_{k-1}[j\,..]$ is yet to be processed. If $S_{k-1}[j] \notin \mathcal{L}_k$ or $S_{k-1}[j+1] \notin \mathcal{R}_k$, we output $\mathsf{id}_{k-1}(S_{k-1}[j])$ as the next symbol of S_k and continue processing $S_{k-1}[j+1\,..]$. Otherwise, we output $(\mathsf{id}_{k-1}(S_{k-1}[j]), \mathsf{id}_{k-1}(S_{k-1}[j+1]))$ as the next symbol of S_k and continue processing $S_{k-1}[j+2\,..]$. (By Fact V.7, $(S_{k-1}[j], S_{k-1}[j+1]) \notin \mathcal{S}_{k-1}$ in this case.) Note that $|S_k| \leq |S_{k-1}|$ and that the characters of S_k are initially represented as elements of $[1\,..\,|S_{k-1}|] \cup [1\,..\,|S_{k-1}|]^2$. Sorting these values allows constructing symbols in $\mathcal{S}_k \setminus \mathcal{S}_{k-1}$ and the identifier function id_k in $O(|S_{k-1}|)$ time.

The algorithm terminates when it encounters a string S_h with $|S_h| = 1$, marking the only symbol of S_h as the starting symbol of the grammar. The overall running time is $O(\sum_{k=0}^{h} |S_k|)$, which is $O(\sum_{k=0}^{h} \frac{n}{\ell_{k+1}}) = O(n)$ in expectation by Lemma V.12. The expected grammar size is $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ by Corollary V.17. □

Finally, we adapt the algorithm to output a small grammar in the worst case.

Theorem V.20. *There exists a randomized algorithm that, given a string $S \in [0\,..\,\sigma)^n$ with $\delta(S) = \delta$, constructs a run-length grammar of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ generating S. The running time is $O(n)$ in expectation and $O(n \log n)$ w.h.p.*

Proof. We compute δ using Lemma II.5 and determine an upper bound on the expected size of the grammar produced using Proposition V.19, as well as an upper bound on the expected running time of the algorithm of Proposition V.19.

Then, we repeatedly call the algorithm of Proposition V.19 with a time limit of 4 times the expected running time. If the call does not finish within the limit, we interrupt the execution and proceed to the next call. Similarly, we proceed to the next call if the size of the produced grammar exceeds 4 times the expected size. Otherwise, the grammar is of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$, so return it to the output.

By Markov's inequality, a call does not terminate within four times the expected running time with probability at most $1/4$. Similarly, each call produces a grammar of size more than four times the expected size with probability at most $1/4$. By the union bound, an attempt fails with probability at most $1/2$. Consequently, the number of calls is constant in expectation. The probability that i independent calls fail (i.e., do not terminate within four times the expected running time *and* produce a grammar of size more than four times the expected size) is at most 2^{-i}. We conclude that $c \log n$ calls are sufficient to guarantee success probability $1 - n^{-c}$ for any constant c. Since each call has a time limit of $O(n)$, the algorithm terminates in $O(n \log n)$ time with high probability. □

VI. Accessing and indexing in δ-bounded space

Christiansen et al. [28, Appendix A] showed how, given a run-length context-free grammar of size g_{rl} generating a string S, we can build a data structure of size $O(g_{rl})$ that supports access and indexed searches on S. Both structures are extensions of their corresponding versions for basic context-free grammars: For accessing, one decomposes the parse tree of S into heavy paths and creates structures that allow one navigating directly to the last heavy path node towards the desired leaf; only $O(\log n)$ heavy path switches are needed to reach the leaf. For indexing, one defines the grammar tree as the pruning of the parse tree that converts every second occurrence of a nonterminal into a leaf. One then cuts S into phrases according to the leaves of the grammar tree. Every occurrence of a pattern is called primary if it spans more than one phrase and secondary if not. The primary occurrences are found with a geometric structure that relates reversed prefixes with the corresponding suffixes meeting at each phrase border, and the secondary occurrences are tracked from the primary ones by following the grammar tree. See Christiansen et al. [28] for more details and to see how the techniques are generalized to permit run-length rules.

A direct corollary of their results and Theorem V.20 is that we can not only represent a string within $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, but also support fast access and indexed searches within that space. We can also support the computation of Karp–Rabin signatures [45] on arbitrary substrings of S.

Corollary VI.1. *Given a string $S \in [0\,..\,\sigma)^n$ with measure δ, there exists a data structure of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ that can be built in $O(n \log n)$ expected time and (1) can retrieve any substring $S[i\,..\,i+\ell]$ in time $O(\ell + \log n)$, (2) can compute the Karp–Rabin fingerprint of any substring of S in time $O(\log n)$, and (3) can report the occ occurrences of any pattern $P[1\,..\,m]$ in S in time $O(m \log n + occ \log^{\epsilon} n)$, for any constant $\epsilon > 0$ fixed at construction time.*

Proof. Points (1), (2), and (3) follow from Theorem V.20 combined with Theorems A.1, A.3, and A.4 of Christiansen et al. [28], respectively. Those structures are built in $O(n \log n)$ expected time, which dominates the $O(n)$ expected time needed to build the run-length grammar. □

On the other hand, no known $O(g_{rl})$-space index can efficiently *count* the number of times $P[1\,..\,m]$ occurs in S. Christiansen et al. [28, Appendix A] instead show an index of size $O(g)$ that can count in time $O(m \log^{2+\epsilon} n)$ for any fixed constant $\epsilon > 0$. We now show that the same can be obtained within space $O(\delta \log \frac{n}{\delta})$. Though this space is always $\Omega(g_{rl})$ (Theorem V.18), it can be $o(g)$ (Theorem V.3).

Later, we show how the results of Corollary VI.1 can be improved in some cases by using block trees [38] instead of grammars. The block tree is a data structure designed to represent repetitive strings $S[1\,..\,n]$ in $O(z \log \frac{n \log \sigma}{z \log n})$ space while accessing individual symbols of S and computing fingerprints in time $O(\log \frac{n \log \sigma}{z \log n})$, that is, faster than in Corollary VI.1 when z is not too small. We show that the block tree is easily tuned to use the worst-case-optimal $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space while retaining its access time.

A. Counting

As explained, it is possible to count how many times $P[1\,..\,m]$ occurs in S in space proportional to any context-free grammar generating S. The idea, however, has not been extended to handle run-length rules of the form $A \to B^t$. Christiansen et al. [28, Section 7] accomplished this only for their particular run-length context-free grammar, of size $O(\gamma \log \frac{n}{\gamma})$. We now show that their result can be carried over to our run-length grammar of Section V.

We start proving a technical point about our grammar, which we will need to establish the result.

Definition VI.2 ([46])**.** *A* period *of a string $S[1\,..\,n]$ is a positive integer p such that $S[p+1\,..\,n] = S[1\,..\,n-p]$. We denote the smallest period of S with $\mathsf{per}(S)$.*

Lemma VI.3 (cf. [20, Lemma 6.17])**.** *For every rule of the form $A \to B^t$ in our run-length grammar, $\mathsf{per}(\exp(A)) = |\exp(B)|$.*

Proof. Observe that $|\exp(B)|$ is a period of $\exp(A)$ because $\exp(A) = \exp(B)^t$. Thus, by the Periodicity Lemma [47], $\mathsf{per}(\exp(A)) = \mathsf{per}(\exp(B)) = \frac{1}{s}|\exp(B)|$ holds for some integer $s \geq 1$. For a proof by contradiction, suppose that $s > 1$.

Let ℓ be the minimum level such that A occurs in S_ℓ, and let us fix an occurrence of A in S_ℓ. In each string S_k with $k \in [0\,..\,\ell]$, this occurrence corresponds to a fragment x_k satisfying $\exp(x_k) = \exp(A)$. Note that $x_\ell = A$ and $x_{\ell-1} = B^t$. Consequently, for each $k \in [0\,..\,\ell)$, we have $x_k = Y_k^t$ for some string $Y_k \in \mathcal{A}^+$; in particular, $Y_0 = \exp(B)$ and $Y_{\ell-1} = B$.

Let us fix the largest $k \in [0\,..\,\ell)$ such that $Y_k = Z_k^s$ for some string $Z_k \in \mathcal{A}^+$; note that k is well-defined due to $\mathsf{per}(Y_0) = \frac{1}{s}|Y_0|$ and $k < \ell - 1$ due to $\mathsf{per}(Y_{\ell-1}) = |Y_{\ell-1}| = 1$.

Let us decompose x_k into st equal-length fragments $x_k = z_{k,1} \cdots z_{k,st}$, and note that $z_{k,i} = Z_k$ for $i \in [1\,..\,st]$. Consider the partition of S_k into blocks that are then collapsed to form S_{k+1}. Since x_k gets collapsed to x_{k+1}, block boundaries are placed before $z_{k,1}$ and after $z_{k,st}$. Since $z_{k,1} \cdots z_{k,s} = Y_k$ gets collapsed to a fragment matching Y_{k+1}, a block boundary is also placed between $z_{k,s}$ and $z_{k,s+1}$. However, according to Definitions V.4 and V.5, block boundaries are placed solely based on the two adjacent symbols, and therefore a block boundary is placed between every $Z_k[|Z_k|]$ and $Z_k[1]$ and, in particular, between $z_{k,i}$ and $z_{k,i+1}$ for all $i \in [1\,..\,st)$. Consequently, each fragment $z_{k,i}$ is collapsed to a fragment of S_{k+1}, which we denote $z_{k+1,i}$. Since $\exp(z_{k+1,i}) = \exp(z_{k,i}) = \exp(Z_k)$ holds for all $i \in [1\,..\,st]$, Fact V.7 yields a string $Z_{k+1} \in \mathcal{A}^+$ satisfying $z_{k+1,i} = Z_{k+1}$ for all $i \in [1\,..\,st]$. This implies $Y_{k+1} = Z_{k+1}^s$, contradicting the choice of k. Hence, $s = 1$. □

We are now ready to establish the main result.

Theorem VI.4. *Given a string $S[1\,..\,n]$ with measure δ, there exists a data structure of size $O(\delta \log \frac{n}{\delta})$ that can be built in $O(n \log n)$ expected time and can count the number of times any pattern $P[1\,..\,m]$ occurs in S in time $O(m \log^{2+\epsilon} n)$, for any constant $\epsilon > 0$ fixed at construction time.*

Proof. We use the technique that Christiansen et al. [28, Section 7] developed for their particular run-length grammar. The only point where their structure requires some specific property of their grammar is their Lemma 7.2, which we have reproved for our grammar as Lemma VI.3.

The total space is proportional to the size of the grammar, which in our case is $O(\delta \log \frac{n}{\delta})$. Their expected construction time is $O(n \log n)$, which dominates the time to build our run-length grammar. From the time analysis in [28], it follows that the query time is $O(m \log^{2+\epsilon} n)$ because we split P in $m-1$ places, not just in $O(\log m)$ places as their special grammar allows. □

B. Block trees

Given integer parameters τ 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 $n = s \cdot \tau^t$ for some integer t).[1] Blocks are then classified into *marked* and *unmarked*. If two adjacent blocks B', B'' form the leftmost occurrence of the underlying substring $B' \cdot B''$, then both B' and B'' are marked. Blocks B that remain unmarked are replaced by a pointer to the pair of adjacent blocks B', B'' that contains the leftmost occurrence of B, and the offset $\epsilon \geq 0$ where B starts inside B'. Marked blocks are divided into τ equal-sized sub-blocks, which form the children of the current block tree's level, and processed recursively in the same way. Let σ be the alphabet size. The level where the blocks' lengths fall below $\log_\sigma n$ contains the leaves of the block tree, whose blocks store their plain string content using $O(\log n)$ bits. The height of the block tree is then $h = O(\log_\tau \frac{n/s}{\log_\sigma n}) = O(\log_\tau \frac{n \log \sigma}{s \log n}) \subseteq O(\log \frac{n}{s})$.

The block tree construction guarantees that the blocks B' and B'' 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', B'' with offset ϵ, we switch either to $B'[\epsilon + i]$ or to $B''[\epsilon + i - |B'|]$, which are marked blocks. When reaching a leaf, we extract the symbol directly from the string stored explicitly in the leaf.

By storing further data associated with marked and unmarked blocks, the block tree offers the following functionality [38]:

- access: any substring $S[i \,..\, i + \ell - 1]$ is extracted in time $O(h\lceil \ell / \log_\sigma n \rceil)$;
- rank: the number of times symbol a occurs in $S[1 \,..\, i]$, denoted $rank_a(S, i)$, is computed in time $O(h)$ by multiplying the space by $O(\sigma)$;
- select: the position of the jth occurrence of symbol a in S, denoted $select_a(S, j)$, is computed in time $O(pred(s, n) + h\, pred(\tau, n))$ by multiplying the space by $O(\sigma)$, where $pred(x, n)$ is the time of a predecessor query (read below for a more formal definition of predecessor queries) on a set of x elements from the universe $[1 \,..\, n]$.

A predecessor data structure pre-processes a set $X \subseteq [1 \,..\, n]$ so that, later, predecessor queries are answered quickly: the predecessor of $y \in [1 \,..\, n]$ in X is $\max\{z \in X : z \leq y\}$. For example, simply sorting X allows finding predecessors with binary search in $pred(|X|, n) = O(\log |X|)$ time. More advanced (linear-space) data structures drastically reduce this running time [48].

It is shown that there are only $O(z\tau)$ blocks (either marked or unmarked) in each level of the block tree (except the first, which has s blocks); therefore, the size of the block tree is $O(s + z\tau \log_\tau \frac{n \log \sigma}{s \log n})$.

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

Let us call level k of the block tree the one where blocks are of length τ^k (recall that we assume $n = s \cdot \tau^t$). In level k, then, S is covered regularly with blocks $B = S[\tau^k(i-1)+1 \,..\, \tau^k i]$ of length τ^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.

Lemma VI.5. *The number of marked blocks of length τ^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 $2 \cdot \tau^k$ characters equal to $\#$ at the beginning of S and $2\tau^k$ characters equal to $\#$ at the end of S. We index the added prefix in negative positions (up to index 0), so that $S[-2 \cdot \tau^k + 1 \,..\, 0] = \#^{2 \cdot \tau^k}$. Now consider all the τ^k text positions p belonging to a marked block B. The *long* substring $E = S[p - 2 \cdot \tau^k \,..\, p + 2 \cdot \tau^k - 1]$ centered at p, of length $4\tau^k$, 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 $2 \cdot \tau^k$ and a suffix of length $2\tau^k$ consisting of character $\#$ to S, the number of distinct substrings of length $4\tau^k$ is at most $d_{4\tau^k}(S) + 4\tau^k$. Therefore, there can be at most $d_{4\tau^k}(S) + 4\tau^k$ 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 τ^k distinct positions p, there are at most $(d_{4\tau^k}(S) + 4\tau^k)/\tau^k$ marked blocks B of length τ^k. The total number of marked blocks of length τ^k is thus at most $(d_{4\tau^k}(S) + 4\tau^k)/\tau^k = 4 \cdot d_{4\tau^k}(S)/(4\tau^k) + 4\tau^k/\tau^k \leq 4\delta + 4$. □

[1] Otherwise, we simply pad S with spurious symbols at the end; whole spurious blocks are not represented. The extra space incurred is only $O(\tau h)$ for a tree of height h.

Since the block tree has at most $O(\delta)$ marked blocks per level, it has $O(\delta\tau)$ blocks across all the levels except the root level, because each marked block has τ children blocks in the next level. This observation yields the following result.

Theorem VI.6. *Let $S[1\,..\,n]$, over alphabet $[1\,..\,\sigma]$, have measure δ. Then the block tree of S, with parameters τ and s, is of size $O(s + \delta\tau \log_\tau \frac{n\log\sigma}{s\log n})$ words and height $h = O(\log_\tau \frac{n\log\sigma}{s\log n})$.*

2) Operations on block trees: By properly parameterizing the block tree, we obtain a structure that uses the same asymptotic space and, in some cases, extracts substrings faster than the result of Corollary VI.1.

Corollary VI.7. *Let $S \in [0\,..\,\sigma)^n$ have measure $\delta = \delta(S)$. Then a block tree of S can use $O(\delta \log \frac{n\log\sigma}{\delta\log n})$ space and extract a substring of length ℓ from S in time $O(\lceil \ell/\log_\sigma n\rceil \log \frac{n\log\sigma}{\delta\log n})$.*

Proof. We obtain the desired space $O(\delta \log \frac{n\log\sigma}{\delta\log n}) \subseteq O(\delta \log \frac{n}{\delta})$ by using Theorem VI.6 with $s = \delta$ and $\tau = O(1)$. The height is $O(\log \frac{n\log\sigma}{\delta\log n})$, and thus the substring extraction costs $O(\lceil \ell/\log_\sigma n\rceil \log \frac{n\log\sigma}{\delta\log n})$. □

Navarro and Prezza [33] show how the Karp–Rabin signature of any $S[i\,..\,j]$ can be computed in time $O(\log \frac{n}{\gamma})$ on their Γ-tree variant of the block tree, which is of size $O(\gamma \log \frac{n}{\gamma})$. We now extend their result so as to compute the *fingerprint* $\phi(S[i\,..\,j]) = \phi(S[i]) \circ \phi(S[i{+}1]) \circ \cdots \circ \phi(S[j])$ for any group operator $\circ$ within $O(\delta \log \frac{n\log\sigma}{\delta\log n})$ space and using $O(\log \frac{n\log\sigma}{\delta\log n})$ group operations, by enhancing the original block trees. This includes computing the Karp–Rabin signature of $S[i\,..\,j]$ in time $O(\log \frac{n\log\sigma}{\delta\log n})$, because all the required group operations can be supported in constant time on those signatures [33].

Definition VI.8. *Let Σ be an alphabet and $(\mathbb{G}, \circ, ^{-1}, 0)$ a group. A function $\phi : \Sigma^* \to \mathbb{G}$ is a* fingerprint *on Σ^* if $\phi(\varepsilon) = 0$ and $\phi(S \cdot a) = \phi(S) \circ \phi(a)$ for every $S \in \Sigma^*$ and $a \in \Sigma$.*

Theorem VI.9. *Let $S \in \Sigma^n$, $(\mathbb{G}, \circ, ^{-1}, 0)$ be a group where $\log|\mathbb{G}| = O(\log n)$, and $\phi : \Sigma^* \to \mathbb{G}$ a fingerprint on Σ^*. Then there is a data structure of size $O(\delta \log \frac{n\log\sigma}{\delta\log n})$ that can compute any $\phi(S[i\,..\,j])$ in time $O(\log \frac{n\log\sigma}{\delta\log n})$.*

Proof. We use a block tree for S with arity $\tau = O(1)$ and where the leaves correspond to strings of length $\ell = \lfloor \frac{1}{2}\log_\sigma \delta \rfloor$, not $\log_\sigma n$ as before. The height of this block tree then grows to $h = O(\log \frac{n/\delta}{\frac{1}{2}\log_\sigma \delta}) = O(\log \frac{n\log\sigma}{\delta\log\delta}) = O(\log \frac{n\log\sigma}{\delta\log n})$.

In addition, we augment the internal blocks as follows. Together with every marked block $B = S[\tau^k(i-1)+1\,..\,\tau^k i]$ at every level k, we store its fingerprint $\phi(B)$ (using constant space). Furthermore, at the top level, say level $k = \kappa$, we store the fingerprint $\phi(S[1\,..\,\tau^\kappa(i-1)])$ at the block-tree node corresponding to the block $B = S[\tau^\kappa(i-1)+1\,..\,\tau^\kappa i]$. In addition, let $B_1, \ldots, B_\tau$ be the children at level $k-1$ of a marked block B of level k. We store, at each such child B_j, the fingerprint $\phi(B_1 \cdots B_{j-1})$. Finally, for unmarked blocks $B[1\,..\,\tau^k] = B'[i\,..\,\tau^k] \cdot B''[1\,..\,i-1]$, we store at B the fingerprint $\phi(B'[i\,..\,\tau^k])$. See Fig. 2 for an example. We also store a table with $\sigma^\ell \le \sqrt{\delta}$ rows, one representing each possible string of length ℓ. The row representing string $L[1\,..\,\ell]$ stores $\phi(L[1\,..\,i])$ for $1 \le i \le \ell$, that is, for all prefixes of L. The space for this table is just $o(\delta)$. Each block tree leaf points to the corresponding row of this table, using $O(\log\delta)$ bits.

We will retrieve any fingerprint $\phi(S[i\,..\,j])$ as $\phi(S[1\,..\,i-1])^{-1} \circ \phi(S[1\,..\,j])$; therefore we focus on computing only fingerprints of the form $\phi(S[1\,..\,i])$ for arbitrary i. At the top level κ, the prefix $S[1\,..\,i]$ spans a sequence $B_1 \cdots B_t$ of blocks followed by a (possibly empty) prefix C of block B_{t+1}. Since $\phi(B_1 \cdots B_t)$ is explicitly stored at block B_{t+1}, the problem reduces to computing $\phi(C)$ and then returning $\phi(B_1 \cdots B_t) \circ \phi(C)$. The following is needed only if $C \neq \varepsilon$.

To compute the fingerprint of a prefix $B[1\,..\,l]$ of a block B at level $k \le \kappa$ (so $1 \le l \le \tau^k$), we distinguish two cases.

1) B is a marked block, with children $B_1, \ldots, B_\tau$ at level $k-1$, so that $B[l]$ belongs to the child B_j (i.e., $j = \lceil l/\tau^{k-1}\rceil$). We then return $\phi(B_1 \cdots B_{j-1}) \circ \phi(B_j[1\,..\,l \bmod \tau^{k-1}])$, where the first term is stored at B_j and the second is computed from the next level (only necessary if $l \bmod \tau^{k-1} \neq 0$).
2) B is an unmarked block, pointing to a previous occurrence inside $B' \cdot B''$ at the same level k, with both B' and B'' marked. If the occurrence of B spans only one marked block, B', then we replace B by B' in our query, and we are back in case (1). Otherwise, let $B[1\,..\,\tau^k] = B'[i\,..\,\tau^k] \cdot B''[1\,..\,i-1]$. We consider two subcases.
   a) If $l \ge \tau^k - i + 1$, then $B[1\,..\,l] = B'[i\,..\,\tau^k] \cdot B''[1\,..\,l - (\tau^k - i + 1)]$. We then return $\phi(B'[i\,..\,\tau^k]) \circ \phi(B''[1\,..\,l - (\tau^k - i + 1)])$, where the first term is stored at B and the second is a new problem on level k, now in case (1).
   b) If $l < \tau^k - i + 1$, then $B[1\,..\,l] = B'[i\,..\,i+l-1]$. Although this is neither a prefix nor a suffix of a block, note that $B[1\,..\,l] \cdot B'[i+l\,..\,\tau^k] = B'[i\,..\,i+l-1] \cdot B'[i+l\,..\,\tau^k] = B'[i\,..\,\tau^k]$. We then return $\phi(B'[i\,..\,\tau^k]) \circ \phi(B'[i{+}l\,..\,\tau^k])^{-1}$. The first fingerprint is stored at B, whereas the second is the fingerprint of the suffix of the marked block B'. We compute it as $\phi(B'[i+l\,..\,\tau^k]) = \phi(B'[1\,..\,i{+}l{-}1])^{-1} \circ \phi(B')$, where $\phi(B')$ is stored at B' and the first term is again the fingerprint of a prefix at the same level, yet now in case (1).

To sum up, computing a prefix of an explicit block at level k reduces to the problem of computing a prefix of an explicit block at level $k-1$ plus a constant amount of group operations to combine values. In the worst case, we navigate down to the leaves, where fingerprints of any leaf prefix are found in constant time by following the stored pointers to the table. Since the height of this block tree is $O(\log \frac{n\log\sigma}{\delta\log n})$, this is also the cost to compute the whole fingerprint, counting group operations and other costs. □

We note that it is unknown if a result like Theorem VI.9 can

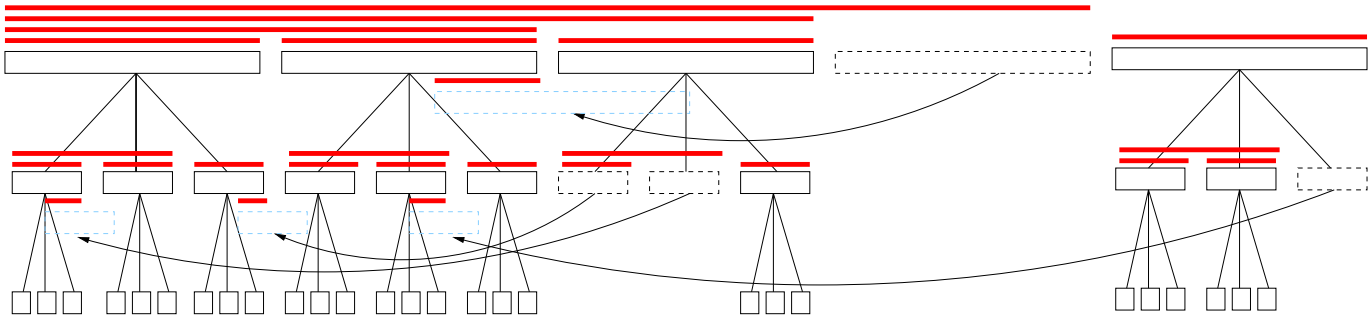

Fig. 2. A block tree with $s = 5$, $\tau = 3$, and height 2. Marked blocks are shown as solid rectangles, with their τ children, whereas unmarked blocks are shown as dashed rectangles, with a curved arrow pointing to their source in the same level, where the same block is shown in shadowed form. The thick lines correspond to all the segments of S whose fingerprints are stored in our data structure: the $s-1$ increasing prefixes on top, the $\tau-1$ increasing prefixes of marked blocks, the marked blocks, and the suffix of the left block pointed by unmarked blocks.

be obtained on a semigroup. For example, it is not known how to compute the smallest symbol contained in any substring in polylogarithmic time on block trees [49], [38].

Let us simplify our time to $O(\log \frac{n}{\delta})$ for the final discussion of this section. Though our time for accessing and fingerprinting seems to be larger than those obtained on other block tree variants, which are of height $O(\log \frac{n}{z})$ [38] or $O(\log \frac{n}{\gamma})$ [33], we next show that $\log \frac{n}{\delta}$ is asymptotically equal to $\log \frac{n}{g}$, which also encompasses all the intermediate measures, $\delta \le \gamma \le b \le c \le z \le g_{rl} \le g$.

Lemma VI.10. *Let $x = O(\delta \log^c \frac{n}{\delta})$ for some constant $c > 0$. Then $\log \frac{n}{\delta} = O(\log \frac{n}{x})$. As a consequence, $\log \frac{n}{\delta} = \Theta(\log \frac{n}{g})$.*

Proof. From the hypohesis it follows that $\frac{n}{\delta} = O(\frac{n}{x} \log^c \frac{n}{\delta})$. Since $\log^c \frac{n}{\delta} = O(\sqrt{\frac{n}{\delta}})$, it holds that $\sqrt{\frac{n}{\delta}} = O(\frac{n}{x})$, and thus $\log \frac{n}{\delta} = O(\log \frac{n}{x})$. The final consequence follows from $\delta \le g = O(z \log \frac{n}{z}) = O(\delta \log \frac{n}{\delta} \log \frac{n}{z}) = O(\delta \log^2 \frac{n}{\delta})$ [14], [15], [37]. □

Hence, the times we obtain using $O(\delta \log \frac{n}{\delta})$ space, not only for access but also for rank and select, and for computing fingerprints, are asymptotically the same as those obtained in $O(\gamma \log \frac{n}{\gamma})$ space [33], [39] or in $O(z \log \frac{n}{z})$ space [38]. Concretely, we can write our access and fingerprinting time as $O(\log \frac{n \log \sigma}{g \log n})$.

Finally, it is also possible to obtain the same result as point (3) of Corollary VI.1 using block trees; see the conference version of this article [1].

VII. 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 can be computed in linear time, while finding γ is NP-hard. It is also insensitive to simple string transformations (reversals, alphabet permutations) and, unlike γ, monotone with respect to appending symbols and changing by only 1 unit upon single-symbol edits.
2) It lower bounds the previous measure, $\delta \le \gamma$, with up to a logarithmic-factor separation: For every string length n, alphabet size σ, and value $10\sigma \le \Delta \le \frac{n \log \sigma}{2 \log n}$, there are string families in $[0..\sigma)^n$ where $\delta \le \Delta$ and $\gamma = \Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n})$.
3) We can always encode S in $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space, and this is worst-case optimal in terms of δ: For any string length n, alphabet size σ, and value $14\sigma \le \Delta \le \frac{n \log \sigma}{2 \log n}$, there are string families in $[0..\sigma)^n$ with $\delta \le \Delta$ and needing $\Omega(\Delta \log \frac{n \log \sigma}{\Delta \log n})$ space to be represented. Thus, $o(\delta \log n)$ space is unreachable in general. Instead, no string family is known to require $\omega(\gamma)$ space, nor it is known if $o(\gamma \log n)$ space can always be reached.
4) We can build a run-length context-free grammar of size $O(\delta \log \frac{n \log \sigma}{\delta \log n})$, which then upper bounds the size g_{rl} of the smallest such grammar, and transitively the measures γ, b, c, v, and z. At the same time, there are string families where the smallest context-free grammar is of size $g = \Omega(\delta \log^2 \frac{n}{\delta} / \log \log \frac{n}{\delta})$. No such separation is known for γ.
5) There are encodings using $O(\delta \log \frac{n \log \sigma}{\delta \log n})$ space and supporting direct access, fingerprinting, and indexed searches, with the same complexities obtained within the larger attractor-bounded space $O(\gamma \log \frac{n}{\gamma})$ [33]. An exception is a very recent faster index [28].

An ideal compressibility measure capturing repetitiveness should be reachable, monotone, resistant to simple string transformations, efficient to compute, and optimal within a hopefully refined partition of the strings. The measure $\delta \log \frac{n \log \sigma}{\delta \log n} \subseteq \delta \log \frac{n}{\delta}$ is reachable, monotone, resistant, fast to compute, and optimal within the class of strings with the same n, σ, and δ values.

In comparison, measure b (the size of the smallest bidirectional macro scheme) is reachable and invariant, but it is non-monotone and NP-hard to compute. On the other hand, it is optimal within a more refined partition, since it is always $O(\delta \log \frac{n \log \sigma}{\delta \log n})$. The size $\gamma \le b$ of the smallest attractor is unknown to be reachable, and it is non-monotone and NP-hard to compute, yet invariant. If it turns out that one can always encode a string within $O(\gamma)$ space, then γ would be a reachable measure even more refined than b.

The measure $\delta \log \frac{n \log \sigma}{\delta \log n}$, or its simpler version $\delta \log \frac{n}{\delta}$, is then a good candidate in the fascinating quest for an ideal measure of repetitiveness. Its main weakness is that it is

optimal within a partition of the strings that, though reasonably refined, is improved by other measures (which have other weaknesses).

On the more algorithmic side, it would be useful to compute δ within little space. The $O(n)$-time computation we provided in Lemma II.5, even if much lighter than the previous one based on suffix trees [28], still requires $O(n)$ space, which can be unaffordable for very large text collections. Bernardini et al. [50], for example, show how to compute it in time $O(n^3/s^2)$ and $O(s)$ space. It would also be worth obtaining faster indexes of size $O(\delta \log \frac{n}{\delta})$. Our index requires $O(m \log n + occ \log^{\epsilon} n)$ search and $O(m \log^{2+\epsilon} n)$ count time, while in $O(\gamma \log \frac{n}{\gamma})$ space it is possible to search in $O(m + (occ+1)\log^{\epsilon} n)$ and count in $O(m + \log^{2+\epsilon} n)$ time [28].

Appendix A
Proof from [42]

In this Appendix, we reproduce proofs from an unpublished manuscript [42].

Fact V.7 ([42]). *For every $k \in \mathbb{Z}_{\geq 0}$, if $\exp(x) = \exp(x')$ holds for two fragments of S_k, then $x = x'$.*

Proof. We proceed by induction on k. Let x, x' be fragments of S_k satisfying $\exp(x) = \exp(x')$. If $k = 0$, then $x = x'$ holds due to $\exp(x) = x$ and $\exp(x') = x'$. Otherwise, let $\bar{x}$ and $\bar{x}'$ be the fragments of S_{k-1} obtained from x and x', respectively, by expanding collapsed blocks. Note that $\exp(\bar{x}) = \exp(x) = \exp(x') = \exp(\bar{x}')$, so the inductive assumption guarantees $\bar{x} = \bar{x}'$. Inspecting Definitions V.4 and V.5, observe that if $S_{k-1}[i] = S_{k-1}[i']$ and $S_{k-1}[i+1] = S_{k-1}[i'+1]$, then block boundaries after positions $i, i' \in [1\,..\,|S_{k-1}|)$ are placed consistently: either after both of them or after neither of them. Consequently, block boundaries within $\bar{x}$ and $\bar{x}'$ are placed consistently. Moreover, both $\bar{x}$ and $\bar{x}'$ consist of full blocks (since they are collapsed to x and x', respectively). Thus, $\bar{x}$ and $\bar{x}'$ are consistently partitioned into full blocks. Matching blocks get collapsed to matching symbols both in Definitions V.4 and V.5, so we derive $x = x'$. □

Corollary V.8 ([42]). *For every odd $k \in \mathbb{Z}_{\geq 0}$, there is no $j \in [1\,..\,|S_k|)$ such that $S_k[j] = S_k[j+1] \in \mathcal{A}_{k+1}$.*

Proof. For a proof by contradiction, suppose that $S_k[j] = S_k[j+1] \in \mathcal{A}_{k+1}$ holds for some $j \in [1\,..\,|S_k|)$. Let $x = S_{k-1}(i-\ell\,..\,i]$ and $x' = S_{k-1}(i\,..\,i+\ell']$ be blocks of S_{k-1} collapsed to $S_k[j]$ and $S_k[j+1]$, respectively. Due to $\exp(x) = \exp(S_k[j]) = \exp(S_k[j+1]) = \exp(x')$, Fact V.7 guarantees $x = x'$ and, in particular, $\ell = \ell'$. If $\ell = 1$, then $S_{k-1}[i] = S_k[j] = S_k[j+1] = S_{k-1}[i+1] \in \mathcal{A}_{k+1}$. Otherwise, $x = x' = A^{\ell}$ for some symbol $A \in \mathcal{A}_k$, which means that $S_{k-1}[i] = S_{k-1}[i+1] = A$. In either case, $S_{k-1}[i] = S_{k-1}[i] \in \mathcal{A}_k = \mathcal{A}_{k+1}$, which means that $\mathsf{rle}_{\mathcal{A}_k}(S_{k-1})$ does not place a block boundary after position i in S_{k-1}. This contradicts the choice of i as the boundary between blocks x and x'. □

Lemma V.9 ([42]). *Let $\alpha \in \mathbb{Z}_{\geq 1}$ and let $i, i' \in [\alpha\,..\,n-\alpha]$ be such that $S(i-\alpha\,..\,i+\alpha] = S(i'-\alpha\,..\,i'+\alpha]$. For every $k \in \mathbb{Z}_{\geq 0}$, if $\alpha \geq 16\ell_k$, then $i \in B_k \iff i' \in B_k$.*

Proof. We proceed by induction on k, with a weaker assumption $\alpha \geq 15\ell_k$ for odd k. In the base case of $k = 0$, the claim is trivial due to $B_k = [1\,..\,n)$. Next, we shall prove that the claim holds for integers $k > 0$ and $\alpha > \ell_k$ assuming that it holds for $k-1$ and $\alpha - \lfloor \ell_k \rfloor$. This is sufficient for the inductive step: If $\alpha \geq 16\ell_k$ for even k, then $\alpha - \lfloor \ell_k \rfloor \geq 15\ell_k = 15\ell_{k-1}$. Similarly, if $\alpha \geq 15\ell_k$ for odd k, then $\alpha - \lfloor \ell_k \rfloor \geq 14\ell_k = 16\ell_{k-1}$.

For a proof by contradiction, suppose that $S(i-\alpha\,..\,i+\alpha] = S(i'-\alpha\,..\,i'+\alpha]$ yet $i \in B_k$ and $i' \notin B_k$ for some $i, i' \in [\alpha\,..\,n-\alpha]$. By the inductive assumption (applied to positions i, i'), $i \in B_k \subseteq B_{k-1}$ implies $i' \in B_{k-1}$. Let us set j, j' so that $i = |\exp(S_{k-1}[1\,..\,j])|$ and $i' = |\exp(S_{k-1}[1\,..\,j'])|$. Since a block boundary was not placed between $S_{k-1}[j']$ and $S_{k-1}[j'+1]$, we have $S_{k-1}[j'], S_{k-1}[j'+1] \in \mathcal{A}_k$ (see Definitions V.4 and V.5). Consequently, the phrases $S(i'-\ell\,..\,i'] = \exp(S_{k-1}[j'])$ and $S(i'\,..\,i'+r] = \exp(S_{k-1}[j'+1])$ around position i' are of length at most $\lfloor \ell_k \rfloor$. By the inductive assumption (applied to positions $i+\delta, i'+\delta$ for $\delta \in [-\ell\,..\,r] \subseteq [-\lfloor \ell_k \rfloor\,..\,\lfloor \ell_k \rfloor]$), there are matching phrases $S(i-\ell\,..\,i]$ and $S(i\,..\,i+r]$ around position i. Due to Fact V.7, this yields $S_{k-1}[j] = S_{k-1}[j']$ and $S_{k-1}[j+1] = S_{k-1}[j'+1]$. Consequently, a block boundary was not placed between $S_{k-1}[j]$ and $S_{k-1}[j+1]$, which contradicts $i \in B_k$. □

Lemma V.10 ([42]). *For every $k \in \mathbb{Z}_{\geq 0}$, we have $\mathbb{E}[|S_k|] < 1 + \frac{4n}{\ell_{k+1}}$.*

Proof. We proceed by induction on k. For $k = 0$, we have $|S_0| = n < 1 + 4n = 1 + \frac{4n}{\ell_1}$. If k is odd, we note that $|S_k| \leq |S_{k-1}|$ and therefore $\mathbb{E}[|S_k|] \leq \mathbb{E}[|S_{k-1}|] < 1 + \frac{4n}{\ell_k} = 1 + \frac{4n}{\ell_{k+1}}$. Thus, it remains to consider even integers $k > 0$.

Claim A.1. *For every even $k > 0$, conditioned on any fixed S_{k-1}, we have $\mathbb{E}\left[|S_k| \;\middle|\; S_{k-1}\right] < \frac{1}{4} + \frac{n}{2\ell_k} + \frac{3}{4}|S_{k-1}|$.*

Proof. Let us define $J = \{j \in [1\,..\,|S_{k-1}|] : j = |S_{k-1}| \text{ or } S_{k-1}[j] \notin \mathcal{A}_k \text{ or } S_{k-1}[j+1] \notin \mathcal{A}_k\}$. Since $A \notin \mathcal{A}_k$ yields $|\exp(A)| > \ell_k$, we have $|J| < 1 + \frac{2n}{\ell_k}$. Moreover, observe that if $j \in [1\,..\,|S_{k-1}|] \setminus J$, then $S_{k-1}[j]$ and $S_{k-1}[j+1]$ are, by Corollary V.8, distinct symbols in $\mathcal{A}_k$. Consequently, $\Pr[S_{k-1}[j] \in \mathcal{L}_k \text{ and } S_{k-1}[j+1] \in \mathcal{R}_k] = \frac{1}{4}$. Thus, the probability that $\mathsf{pc}_{\mathcal{L}_k,\mathcal{R}_k}(S_{k-1})$ places a block boundary after position $j \in [1\,..\,|S_{k-1}|] \setminus J$ is $\frac{3}{4}$. Therefore,

$$\begin{aligned}\mathbb{E}\left[|S_k| \;\middle|\; S_{k-1}\right] &= |J| + \tfrac{3}{4}(|S_{k-1}| - |J|) \\ &= \tfrac{1}{4}|J| + \tfrac{3}{4}|S_{k-1}| < \tfrac{1}{4} + \tfrac{n}{2\ell_k} + \tfrac{3}{4}|S_{k-1}|.\end{aligned}$$ □

Since the partition $\mathcal{A}_k = \mathcal{L}_k \cup \mathcal{R}_k$ is independent of S_{k-1}, Claim V.13 and the inductive assumption yield

$$\begin{aligned}\mathbb{E}[|S_k|] &< \tfrac{1}{4} + \tfrac{n}{2\ell_k} + \tfrac{3}{4}\mathbb{E}[|S_{k-1}|] < \tfrac{1}{4} + \tfrac{n}{2\ell_k} + \tfrac{3}{4} + \tfrac{3n}{\ell_k} \\ &= 1 + \tfrac{7n}{2\ell_k} = 1 + \tfrac{4n}{\ell_{k+1}}.\end{aligned}$$ □

Acknowledgements

Part of this work was carried out during the Dagstuhl Seminar 19241, "25 Years of the Burrows–Wheeler Transform". We also thank Travis Gagie for pointing us the early reference related to δ [37].

TK carried out this work while at the Bar-Ilan University, Israel, 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, and while at University of California, Berkeley, supported in part by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship. GN was supported by Basal Funds FB0001, ANID – Millennium Science Initiative Program – Code ICN17_002, and Fondecyt Grant 1-200038, Chile. NP was supported by ERC grant no. 101039208 (REGINDEX) under the European Union's Horizon Europe Research and Innovation Programme.

References

[1] T. Kociumaka, G. Navarro, and N. Prezza, "Towards a definitive measure of repetitiveness," in *Proc. 14th Latin American Symposium on Theoretical Informatics (LATIN)*, ser. LNCS, vol. 12118. Springer, 2020. doi: 10.1007/978-3-030-61792-9_17 pp. 207–219.

[2] G. Navarro, *Compact Data Structures – A practical approach*. Cambridge University Press, 2016. ISBN 978-1-10-715238-0

[3] ——, "Indexing highly repetitive string collections, part I: Repetitiveness measures," *ACM Computing Surveys*, vol. 54, no. 2, p. article 29, 2022. doi: 10.1145/3434399 Most recent version at https://arxiv.org/abs/2004.02781.

[4] T. Gagie, "Large alphabets and incompressibility," *Information Processing Letters*, vol. 99, no. 6, pp. 246–251, 2006. doi: 10.1016/j.ipl.2006.04.008

[5] S. Kreft and G. Navarro, "On compressing and indexing repetitive sequences," *Theoretical Computer Science*, vol. 483, pp. 115–133, 2013. doi: 10.1016/ j.tcs.2012.02.006

[6] A. Lempel and J. Ziv, "On the complexity of finite sequences," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 75–81, 1976. doi: 10.1109/ TIT.1976.1055501

[7] J. C. Kieffer and E. Yang, "Grammar-based codes: A new class of universal lossless source codes," *IEEE Transactions on Information Theory*, vol. 46, no. 3, pp. 737–754, 2000. doi: 10.1109/18.841160

[8] V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki, "Storage and retrieval of highly repetitive sequence collections," *Journal of Computational Biology*, vol. 17, no. 3, pp. 281–308, 2010. doi: 10.1089/cmb.2009.0169

[9] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *International Journal of Computer Mathematics*, vol. 2, no. 1-4, pp. 157–168, 1968. doi: 10.1080/00207166808803030

[10] M. Rodeh, V. R. Pratt, and S. Even, "Linear algorithm for data compression via string matching," *Journal of the ACM*, vol. 28, no. 1, pp. 16–24, 1981. doi: 10.1145/322234.322237

[11] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *Journal of the ACM*, vol. 29, no. 4, pp. 928–951, 1982. doi: 10.1145/ 322344.322346

[12] G. Navarro, C. Ochoa, and N. Prezza, "On the approximation ratio of ordered parsings," *IEEE Transactions on Information Theory*, vol. 67, no. 2, pp. 1008–1026, 2021. doi: 10.1109/TIT.2020.3042746

[13] J. K. Gallant, "String compression algorithms," Ph.D. dissertation, Princeton University. ISBN 979-8-204-67683-1 1982. [Online]. Available: https://www.proquest.com/dissertations-theses/string-compression-algorithms/docview/303254400/se-2

[14] W. Rytter, "Application of Lempel-Ziv factorization to the approximation of grammar-based compression," *Theoretical Computer Science*, vol. 302, no. 1-3, pp. 211–222, 2003. doi: 10.1016/S0304-3975(02)00777-6

[15] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat, "The smallest grammar problem," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2554–2576, 2005. doi: 10.1109/TIT.2005.850116

[16] A. Jeż, "A really simple approximation of smallest grammar," *Theoretical Computer Science*, vol. 616, pp. 141–150, 2016. doi: 10.1016/j.tcs.2015.12.032

[17] 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)*, 2016. doi: 10.4230/LIPIcs.MFCS.2016.72 pp. 72:1–72:15.

[18] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa, "Collage system: A unifying framework for compressed pattern matching," *Theoretical Computer Science*, vol. 298, no. 1, pp. 253–272, 2003. doi: 10.1016/S0304-3975(02)00426-7

[19] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digital Equipment Corporation, Tech. Rep. 124, 1994. [Online]. Available: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf

[20] D. Kempa and T. Kociumaka, "Resolution of the Burrows-Wheeler transform conjecture," in *Proc. 61st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020. doi: 10.1109/focs46700.2020.00097 pp. 1002–1013.

[21] 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*, vol. 34, no. 3, pp. 578–595, 1987. doi: 10.1145/28869.28873

[22] 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)*. Springer, 2015. doi: 10.1007/978-3-319-19929-0_3 pp. 26–39.

[23] D. Belazzougui and F. Cunial, "Representing the suffix tree with the CDAWG," in *Proc. 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2017. doi: 10.4230/LIPIcs.CPM.2017.7 pp. 7:1–7:13.

[24] ——, "Fast label extraction in the CDAWG," in *Proc. 24th International Symposium on String Processing and Information Retrieval (SPIRE)*, 2017. doi: 10.1007/978-3-319-67428-5_14 pp. 161–175.

[25] T. Gagie, G. Navarro, and N. Prezza, "Fully-functional suffix trees and optimal text searching in BWT-runs bounded space," *Journal of the ACM*, vol. 67, no. 1, pp. 1–54, 2020. doi: 10.1145/3375890

[26] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann, "Random access to grammar-compressed strings and trees," *SIAM Journal on Computing*, vol. 44, no. 3, pp. 513–539, 2015. doi: 10.1137/130936889

[27] D. Belazzougui, P. H. Cording, S. J. Puglisi, and Y. Tabei, "Access, rank, and select in grammar-compressed strings," in *Proc. 23rd Annual European Symposium on Algorithms (ESA)*, 2015. doi: 10.1007/978-3-662-48350-3_13 pp. 142–154.

[28] A. R. Christiansen, M. B. Ettienne, T. Kociumaka, G. Navarro, and N. Prezza, "Optimal-time dictionary-compressed indexes," *ACM Transactions on Algorithms*, vol. 17, no. 1, pp. 8:1–8:39, 2021. doi: 10.1145/3426473

[29] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *ACM Computing Surveys*, vol. 39, no. 1, 2007. doi: 10.1145/1216370.1216372

[30] G. Navarro, "Indexing highly repetitive string collections, part II: Compressed indexes," *ACM Computing Surveys*, vol. 54, no. 2, p. article 26, 2022. doi: 10.1145/3432999 Most recent version at https://arxiv.org/abs/2004.02781.

[31] T. Nishimoto and Y. Tabei, "Optimal-time queries on BWT-runs compressed indexes," in *Proc. 48th International Colloquium on Automata, Languages, and Programming (ICALP)*, vol. 198. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ICALP.2021.101 pp. 101:1–101:15.

[32] D. Kempa and N. Prezza, "At the roots of dictionary compression: String attractors," in *Proc. 50th Annual ACM Symposium on the Theory of Computing (STOC)*, 2018. doi: 10.1145/3188745.3188814 pp. 827–840.

[33] G. Navarro and N. Prezza, "Universal compressed text indexing," *Theoretical Computer Science*, vol. 762, pp. 41–50, 2019. doi: 10.1016/j.tcs.2018.09.007

[34] S. Mantaci, A. Restivo, G. Romana, G. Rosone, and M. Sciortino, "A combinatorial view on string attractors," *Theoretical Computer Science*, vol. 850, pp. 236–248, 2021. doi: 10.1016/j.tcs.2020.11.006

[35] T. Akagi, M. Funakoshi, and S. Inenaga, "Sensitivity of string compressors and repetitiveness measures," 2021. [Online]. Available: https://arxiv.org/abs/2107.08615

[36] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S. J. Puglisi, "LZ77-based self-indexing with faster pattern matching," in *Proc. 11th Latin American Symposium on Theoretical Informatics (LATIN)*. Springer, 2014. doi: 10.1007/978-3-642-54423-1_63 pp. 731–742.

[37] S. Raskhodnikova, D. Ron, R. Rubinfeld, and A. D. Smith, "Sublinear algorithms for approximating string compressibility," *Algorithmica*, vol. 65, no. 3, pp. 685–709, 2013. doi: 10.1007/s00453-012-9618-6
[38] D. Belazzougui, M. Cáceres, T. Gagie, P. Gawrychowski, J. Kärkkäinen, G. Navarro, A. O. Pereira, S. J. Puglisi, and Y. Tabei, "Block trees," *Journal of Computer and System Sciences*, vol. 117, pp. 1–22, 2021. doi: 10.1016/j.jcss.2020.11.002
[39] N. Prezza, "Optimal rank and select queries on dictionary-compressed text," in *Proc. 30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, 2019. doi: 10.4230/LIPIcs.CPM.2019.4 pp. 4:1–4:12.
[40] J. Kärkkäinen, P. Sanders, and S. Burkhardt, "Linear work suffix array construction," *Journal of the ACM*, vol. 53, no. 6, pp. 918–936, 2006. doi: 10.1145/1217856.1217858
[41] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-time longest-common-prefix computation in suffix arrays and its applications," in *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM)*, ser. LNCS, A. Amir and G. M. Landau, Eds., vol. 2089. Springer, 2001. doi: 10.1007/3-540-48194-x_17 pp. 181–192.
[42] T. Kociumaka, J. Radoszewski, W. Rytter, and T. Waleń, "Internal pattern matching queries in a text and applications," 2021, unpublished manuscript.
[43] O. Birenzwige, S. Golan, and E. Porat, "Locally consistent parsing for text indexing in small space," in *Proc. 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020. doi: 10.1137/1.9781611975994.37 pp. 607–626.
[44] S. C. Sahinalp and U. Vishkin, "On a parallel-algorithms method for string matching problems," in *Proc. 2nd Italian Conference on Algorithms and Complexity (CIAC)*, ser. LNCS, vol. 778, 1994. doi: 10.1007/3-540-57811-0_3 pp. 22–32.
[45] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987. doi: 10.1147/rd.312.0249
[46] M. Crochemore and W. Rytter, *Jewels of Stringology*. World Scientific, 2002.
[47] N. J. Fine and H. S. Wilf, "Uniqueness theorems for periodic functions," *Proceedings of the American Mathematical Society*, vol. 16, no. 1, pp. 109–114, 1965. doi: 10.1090/S0002-9939-1965-0174934-9
[48] G. Navarro and J. Rojas-Ledesma, "Predecessor search," *ACM Computing Surveys*, vol. 53, no. 5, p. article 105, 2020. doi: 10.1145/3409371
[49] M. Cáceres and G. Navarro, "Faster repetition-aware compressed suffix trees based on block trees," *Information and Computation*, vol. 285, Part B, p. 104749, 2022. doi: 10.1016/j.ic.2021.104749
[50] G. Bernardini, G. Fici, P. Gawrychowski, and S. P. Pissis, "Substring complexity in sublinear space," 2020. [Online]. Available: https://arxiv.org/abs/2007.08357

Tomasz Kociumaka received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Warsaw, Poland. He has been a post-doctoral researcher at the Bar-Ilan University (Israel), the University of California, Berkeley, and the Max Planck Institute for Informatics (Saarbrücken, Germany). His research focuses on designing efficient algorithms for processing strings with a particular focus on sequence similarity measures, approximate pattern matching, lossless data compression, and data structures. He studies string problems from multiple perspectives, including combinatorics on words, dynamic algorithms, fine-grained complexity, streaming and sketching, and sublinear algorithms. He is a co-author of over 100 conference and journal papers and has served as a program committee member for 10 international conferences. Dr. Kociumaka was the recipient of the Cor Baayen Young Researcher Award in 2021 and the Witold Lipski Prize in 2018.

Gonzalo Navarro completed his PhD in Computer Science in 1998 at the University of Chile, where he is currently full professor. His areas of interest include algorithms and data structures, compression, graph databases, and text searching. He has directed the Millennium Nucleus Center for Web Research and projects funded by Yahoo! Research and Google. He currently participates in the Center for Biotecnology and Bioengineering (CeBiB) and the Millennium Institute for Foundational Research on Data (IMFD). He has been PC (co-)chair of over ten international conferences and guest editor of several special issues in relevant journals. He is the Editor in Chief of the ACM Journal of Experimental Algorithmics and a member of the Editorial Board of the ACM Transactions on Algorithms and Information Systems. He has coauthored two books published by Cambridge University Press, about 25 book chapters, 200 papers in international journals, and around 275 in international conferences. He is one of the most prolific and highly cited authors in Latin America. He has received 7 Best Paper Awards in conferences, 4 Google Research awards, a Highest Cited Paper Award from Elsevier, and a Scopus Chile Award. He is an ACM Distinguished Member.

Nicola Prezza received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Udine, Italy. He spent research periods (as a postdoc) at the universities of Pisa (Italy) and DTU (Copenhagen, Denmark), and later has been Assistant professor at LUISS (Rome, Italy). He is now associate professor at Ca' Foscari University, Venice, Italy. His research focuses on the interplay between data structures, data compression, and regular language theory by studying compressed data structures for supporting fast pattern matching queries on regular languages. He is a co-author of 50 conference and journal papers, has served as a program committee member for 11 international conferences, and has been invited speaker in 4 international conferences. In 2018 he received the "best Italian young researcher in Theoretical Computer Science" award from the Italian chapter of the European Association for Theoretical Computer Science (IC-EATCS). In 2021 he was awarded an ERC starting grant on the topic of compressed indexing for labeled graphs and regular languages.