Iterated Straight-Line Programs*

Gonzalo Navarro and Cristian Urbina

CeBiB — Centre for Biotechnology and Bioengineering
Departament of Computer Science, University of Chile

Abstract. We explore an extension to straight-line programs (SLPs) that outperforms, for some text families, the measure δ based on substring complexity, a lower bound for most measures and compressors exploiting repetitiveness (which are crucial in areas like Bioinformatics). The extension, called iterated SLPs (ISLPs), allows rules of the form $A \rightarrow \Pi_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$, for which we show how to extract any substring of length λ, from the represented text $T[1 \mathinner{.\,.} n]$, in time $O(\lambda + \log^2 n \log \log n)$. This is the first compressed representation for repetitive texts breaking δ while, at the same time, supporting direct access to arbitrary text symbols in polylogarithmic time. As a byproduct, we extend Ganardi et al.'s technique to balance any SLP (so it has a derivation tree of logarithmic height) to a wide generalization of SLPs, including ISLPs.

Keywords: Grammar compression · Substring complexity · Repetitiveness measures

1 Introduction

Motivated by the data deluge, and by the observed phenomenon that many of the fastest-growing text collections are highly repetitive, recent years have witnessed an increasing interest in (1) defining measures of compressibility that are useful for highly repetitive texts, (2) develop compressed text representations whose size can be bounded in terms of those measures, and (3) provide efficient (i.e., polylogarithmic time) access methods to those compressed texts, so that algorithms can be run on them without ever decompressing the texts [23,24]. We call *lower-bounding measures* those satisfying (1), *reachable measures* those (asymptotically) reached by the size of a compressed representation (2), and *accessible measures* those reached by the size of representations satisfying (3).

For example, the size γ of the smallest "string attractor" of a text T is a lower-bounding measure, unknown to be reachable [14], and smaller than the size reached by known compressors. The size b of the smallest "bidirectional macro scheme" of T [31], and the size z of the Lempel-Ziv parse of T [20], are reachable measures. The size g of the smallest context-free grammar generating (only) T [5] is an accessible measure [3]. It holds $\gamma \leq b \leq z \leq g$ for every text.

One of the most attractive lower-bounding measures devised so far is δ [29,6]. Let $T[1 \mathinner{.\,.} n]$ be a text over alphabet $[1 \mathinner{.\,.} \sigma]$, and T_k be the number of distinct substrings of length k in T, which define its so-called substring complexity. Then the measure is $\delta(T) = \max_k T_k/k$. This measure has several attractive properties: it can be computed in linear time and lower-bounds all previous measures of compressibility, including γ, for every text. While δ is known to be unreachable, the measure $\delta' = \delta \log \frac{n \log \sigma}{\delta \log n}$ has all the desired properties: $\Omega(\delta')$ is the space needed to represent some text family for each n, σ, and δ; within $O(\delta')$ space it is possible to represent every text T and access any length-λ substring of T in time $O(\lambda + \log n)$ [18], together with more powerful operations [18,17,13].

As for g, a *straight-line program (SLP)* is a context-free grammar that generates (only) T, and has size-2 rules of the form $A \to BC$, where B and C are nonterminals, and size-1 rules $A \to \mathtt{a}$, where $\mathtt{a}$ is a terminal symbol. The SLP size is the sum of all its rule sizes. A *run-length SLP (RLSLP)* may contain, in addition, size-2 rules of the form $A \to B^k$, representing k repetitions of nonterminal B. A RLSLP of size g_{rl} can be represented in $O(g_{rl})$ space, and within that space we can offer fast string access and other operations [6, App. A]. It holds $\delta \leq g_{rl} = O(\delta')$, where g_{rl} is the smallest RLSLP that generates T [23,18] (the size g of the smallest grammar or SLP, instead, is not always $O(\delta')$).

While δ lower-bounds all previous measures on every text, δ' is not the smallest accessible measure. In particular, g_{rl} is always $O(\delta')$, and it can be smaller by up to a logarithmic factor. Indeed, g_{rl} is a minimal accessible measure as far as we know. It is asymptotically between z and g [23]. An incomparable accessible measure is $z_{end} \geq z$, the size of the LZ-End parse of the text [19,15].

The belief that δ is a lower bound to every reachable measure was disproved by the recently proposed L-systems [26,27]. L-systems are like SLPs where all the symbols are nonterminals and the derivation ends at a specified depth in the derivation tree. The size ℓ of the smallest L-system generating $T[1 \mathinner{.\,.} n]$ is a reachable measure of repetitiveness and was shown to be as small as $O(\delta/\sqrt{n})$ on some text families, thereby sharply breaking δ as a lower bound. Measure ℓ, however, is unknown to be accessible, and thus one may wonder whether there exist accessible text representations that are smaller than δ.

In this paper we devise such a representation, which we call *iterated SLP (ISLP)*. ISLPs extend SLPs (and RLSLPs) by allowing a more complex version of the rule $A \to B^k$, namely $A \to \Pi_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$ of size $2 + 2t$. We show how to extract a substring of length λ from the ISLP of a text T in time $O(\lambda + \log^2 n \log \log n)$ provided the ISLP is balanced, that is, its derivation tree is of height $O(\log n)$.

Just like SLPs and RLSLPs can be balanced [10,28] while retaining their asymptotic size, we show how to balance a more general class of SLP extensions we call generalized SLPs (GSLPs). GSLPs, which include ISLPs, allow rules of the form $A \to x$, where x is a *program* that outputs the right-hand side of the rule. We show that, if every nonterminal appearing in x's output does it at least twice, then the GSLP can be balanced in the same way as SLPs. This byproduct

of our results can be of independent interest to provide polylogarithmic-time access to other extensions of context-free grammars.

2   Preliminaries

We explain some concepts and notation used in the rest of the paper.

Strings Let $\Sigma = [1 \mathinner{\ldotp\ldotp} \sigma]$ be an *alphabet*. A *string* $T[1 \mathinner{\ldotp\ldotp} n]$ of length n is a finite sequence $T[1] \, T[2] \ldots T[n]$ of n symbols in Σ. We denote by ε the unique string of length 0. We denote by Σ^* the set of all finite strings with symbols in Σ. The i-th symbol of T is denoted by $T[i]$, and the sequence $T[i] \ldots T[j]$ is denoted by $T[i \mathinner{\ldotp\ldotp} j]$. The *concatenation* of $X[1 \mathinner{\ldotp\ldotp} n]$ and $Y[1 \mathinner{\ldotp\ldotp} m]$ is defined as $X \cdot Y = X[1] \ldots X[n] \, Y[1] \ldots Y[m]$ (we omit the dot when there is no ambiguity). If $T = XYZ$, then X (resp. Y, resp. Z) is a *prefix* (resp. *substring*, resp. *suffix*) of T. A *power* T^k stands for k consecutive concatenations of the string T. We denote by $|T|_a$ the number of occurrences of the symbol a in T. A *string morphism* is a function $\varphi : \Sigma^* \to \Sigma^*$ such that $\varphi(xy) = \varphi(x) \cdot \varphi(y)$ for any strings x and y.

Straight-Line Programs A *straight-line program* (SLP) is a context-free grammar [30] that contains only terminal rules of the form $A \to \mathtt{a}$ with $\mathtt{a} \in \Sigma$, and binary rules of the form $A \to BC$ for variables B and C whose derivations cannot reach again A. These restrictions ensure that each variable of the SLP generates a unique string, defined as $\mathtt{exp}(A) = \mathtt{a}$ for a rule $A \to \mathtt{a}$, and as $\mathtt{exp}(A) = \mathtt{exp}(B) \cdot \mathtt{exp}(C)$ for a rule $A \to BC$. A *run-length straight-line program* (RLSLP) is an SLP that also admits run-length rules of the form $A \to B^k$ for some $k \geq 3$, with their expansion defined as $\mathtt{exp}(A) = \mathtt{exp}(B)^k$. The *size* of an SLP is the sum of the lengths of the right-hand sides of its rules; the size of an RLSLP is defined similary, assuming that rules $A \to B^k$ are of size 2 (i.e., two integers to represent B and k).

The *derivation tree* of an SLP is an ordinal tree where the nodes are the variables, the root is the initial variable, and the leaves are the terminal variables. The children of a node are the variables appearing in the right-hand side of its rule (in left-to-right order). The *height* of an SLP is the length of the longest path from the root to a leaf node in the derivation tree. The height of an RLSLP is obtained by *unfolding* its run-length rules, that is, writing a rule B^k as $BB \ldots B$ where B appears k times, to obtain an equivalent SLP (actually, a slight extension where the right-hand sides can feature more than two variables).

SLPs and RLSLPs yield measures of repetitiveness g and g_{rl}, defined as the size of the smallest SLP and RLSLP generating the text, respectively. Clearly, it holds that $g_{rl} \leq g$. It also has been proven that g is NP-hard to compute [5].

Other Repetitiveness Measures For self-containedness, we describe the most important repetitiveness measures and relate them with the accessible measures g and g_{rl}; for more details see a survey [23].

Burrows-Wheeler Transform. The *Burrows-Wheeler Transform* (BWT) [4] is a reversible permutation of T, which we denote by $\texttt{bwt}(T)$. It is obtained by sorting lexicographically all the rotations of the string T and concatenating their last symbols, which can be done in $O(n)$ time. The measure r is defined as the size of the *run-length encoding* of $\texttt{bwt}(T)$. Usually, T is assumed to be appended with a sentinel symbol $\$$ strictly smaller than any other symbol in T, and then we call $r_\$$ the size of the run-length encoding of $\texttt{bwt}(T\$)$. This measure is then reachable, and fully-functional indexes of size $O(r_\$)$ exist [8], but interestingly, it is unknown to be accessible. While this measure is generally larger than others, it can be upper-bounded by $r_\$ = O(\delta \log \delta \log \frac{n}{\delta})$ [16].

Lempel-Ziv Parsing. The *Lempel-Ziv parsing* (LZ) [20] of a text $T[1 \mathinner{.\,.} n]$ is a *factorization* into non-empty *phrases* $T = X_1 X_2 \ldots X_z$ where each X_i is either the first occurrence of a symbol or the longest prefix of $X_i \ldots X_z$ with a copy in T starting at a position in $[1 \mathinner{.\,.} |X_1 \ldots X_{i-1}|]$. LZ is called a *left-to-right* parsing because each phrase has its *source* starting to the left, and it is optimal among all parsings satisfying this condition. It can be constructed greedily from left to right in $O(n)$ time. The measure z is defined as the number of phrases in the LZ parsing of the text, and it has been proved that $z \leq g_{rl}$ [25]. While z is obviously reachable, it is unknown to be accessible. A close variant $z_{end} \geq z$ [19] that forces phrase sources to be end-aligned with a preceding phrase, has been shown to be accessible [15].

Bidirectional macro schemes. A *bidirectional macro scheme* (BMS) [31] is a factorization of a text $T[1 \mathinner{.\,.} n]$ where each phrase can have its source starting either to the left or to the right. The only requeriment is that by following the pointers from phrases to sources, we should eventually be able to fully decode the text. The measure b is defined as the size of the smallest BMS representing the text. Clearly, b is reachable, but it is unknown to be accessible. It holds that $b \leq z$, and it was proved that $b \leq r_\$$ [25]. Computing b is NP-hard [9].

String Attractors. A *string attractor* for a text $T[1 \mathinner{.\,.} n]$ is a set of positions $\Gamma \subseteq [1 \mathinner{.\,.} n]$ such that any substring of $T[i \mathinner{.\,.} j]$ has an occurrence $T[i' \mathinner{.\,.} j']$ crossing at least one of the positions in Γ (i.e., there exist $k \in \Gamma$ such that $i' \leq k \leq j'$). The measure γ is defined as the size of the smallest string attractor for the string T, and it is NP-hard to compute [14]. It holds that γ lower bounds the size b of the smallest bidirectional macro scheme and can sometimes be asymptotically smaller [2]. On the other hand, it is unknown if γ is reachable.

Substring Complexity. Let $T[1 \mathinner{.\,.} n]$ be a text and T_k be the number of distinct substrings of length k in T, which define its so-called substring complexity. Then the measure is $\delta = \max_k T_k/k$ [29,6]. This measure can be computed in $O(n)$ time and lower-bounds γ, and thus all previous measures of compressibility, for every text. On the other hand, it is known to be unreachable [18]. The related measure $\delta' = \delta \log \frac{n \log \sigma}{\delta \log n}$ is reachable and accessible, and still lower-bounds b and all other reachable measures on some text family for every n, σ, and δ [18]. Besides, g_{rl} (and thus z, b, and γ, but not g) are upper-bounded by $O(\delta \log \frac{n \log \sigma}{\delta \log n})$.

L-systems. An *L-system* (for compression) is a tuple $L = (V, \varphi, \tau, S, d, n)$ extending a Lindenmayer system [21,22], where V is the set of variables (which are also considered as terminal symbols), $\varphi : V \to V^+$ is the set of rules (and also a morphism of strings), $\tau : V \to V$ is a coding, $S \in V$ the initial variable, and d and n are integers. The string generated by the system is $\tau(\varphi^d(S))[1 \mathbin{.\,.} n]$. The measure ℓ is defined as the size of the smallest L-system generating the string. It has been proven that ℓ is incomparable to δ (ℓ can be smaller by a $\sqrt{n}$ factor) and almost any other repetitiveness measure considered in the literature [26,27].

3   Iterated Straight-Line Programs

We now define iterated SLPs and show that they can be much smaller than δ. Some proofs in this section are omitted due to space constraints.

Definition 1. *An* iterated straight-line program *of* degree d *(d-ISLP) is an SLP that allows in addition* iteration rules *of the form* $A \to \prod_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$, *where* $1 \leq k_1, k_2$, $0 \leq c_1, \ldots, c_t \leq d$ *are integers and* $B_1 \ldots B_t$ *are variables that cannot reach A (so the ISLP generates a unique string). Iteration rules have size $2 + 2t = O(t)$ and expand to* $\mathrm{exp}(A) = \prod_{i=k_1}^{k_2} \mathrm{exp}(B_1)^{i^{c_1}} \cdots \mathrm{exp}(B_t)^{i^{c_t}}$, *where if $k_1 > k_2$ the iteration goes from $i = k_1$ downwards to $i = k_2$. The size $\mathrm{size}(G)$ of a d-ISLP G is the sum of the sizes of all of its rules.*

Definition 2. *The measure $g_{it(d)}(T)$ is defined as the size of the smallest d-ISLP that generates T, whereas $g_{it}(T) = \min_{d \geq 0} g_{it(d)}(T)$.*

The following observations show that ISLPs subsume RLSLPs, and thus, can be smaller than the smallest L-system.

Proposition 1. *For any $d \geq 0$, it always holds that $g_{it(d)} \leq g_{rl}$.*

Proof. Just note that a rule $A \to \prod_{i=1}^{k} B^{i^0}$ from an ISLP simulates a rule $A \to B^k$ from a RLSLP. In particular, 0-ISLPs are equivalent to RLSLPs. □

Proposition 2. *For any $d \geq 0$, there exists a string family where $g_{it(d)} = o(\ell)$.*

Proof. Navarro and Urbina show a string family where $g_{rl} = o(\ell)$ [27]. Hence, $g_{it(d)}$ is also $o(\ell)$ in this family. □

We now show that $d = 1$ suffices to obtain ISLPs that are significantly smaller than δ for some string families.

Lemma 1. *Let $d \geq 1$. There exists a string family with $g_{it(d)} = O(1)$ and $\delta = \Omega(\sqrt{n})$.*

Proof. Such a family is formed by the strings $s_k = \prod_{i=1}^{k} \mathtt{a}^i \mathtt{b}$. The 1-ISLPs with initial rule $S_k \to \prod_{i=1}^{k} A^i B$, and rules $A \to \mathtt{a}$, $B \to \mathtt{b}$, generate each string s_k in the family using $O(1)$ space. On the other hand, it has been proven that $\delta = \Omega(\sqrt{n})$ in the family $\mathtt{c} s_k$ [27]. As δ can only decrease by 1 after the deletion of a character [1], $\delta = \Omega(\sqrt{n})$ in the family s_k too. □

On the other hand, ISLPs can perform worse than other compressed representations; recall that $\delta \leq \gamma \leq b \leq r_\$$.

Lemma 2. *Let $\mu \in \{r, r_\$, \ell\}$. There exists a string family with $g_{it(d)} = \Omega(\log n)$ and $\mu = O(1)$.*

Lemma 3. *There exists a string family satisfying that $z = O(\log n)$ and $g_{it(d)} = \Omega(\log^2 n / \log \log n)$.*

One thing that makes ISLPs robust is that they are not very sensitive to reversals, morphism application, or edit operations (insertions, deletions, and substitutions of a single character). This makes $g_{it(d)}$ more robust than measures like r and $r_\$$, which are sensitive to all these transformations [11,12,7,1].

Lemma 4. *Let G be a d-ISLP generating T. Then there exists a d-ISLP of size $|G|$ generating the reversed text T^R. Let φ be a morphism. Then there exists a d-ISLP of size $|G| + c_\varphi$ generating the text $\varphi(T)$, where c_φ is a constant depending only on φ. Moreover, there exists a d-ISLP of size at most $O(|G|)$ generating T' where T and T' differ by one edit operation.*

4   Accessing ISLPs

We have shown that $g_{it(d)}$ breaks the lower bound δ already for $d \geq 1$. We now show that the measure is accessible. Concretely, we will show that any substring of length λ can be extracted in time $O(\lambda + (h + \log n) \log n \log \log n)$, where h is the height of the grammar tree, and in Section 5 we show that ISLPs can be balanced so they have $h = O(\log n)$. In total, we obtain the following result.

Theorem 1. *Let $T[1\mathinner{\ldotp\ldotp}n]$ be represented by a d-ISLP of size g_{it}. Then, there exists a data structure of size $O(g_{it})$ that extracts any substring of T of length λ in time $O(\lambda + \log^2 n \log \log n)$ on a RAM machine of $\Theta(\log n)$ bits, using $O(\log^2 n \log \log n)$ additional words of working space.*

In fact, our extraction time is $O(\lambda + d \log d \log n + d^2 \log d)$ using $O(d^2 \log d)$ working space, which reduces to $O(\lambda + \log n)$ time and $O(1)$ working space for $d = O(1)$ (recall that 1-ISLPs already break the δ lower-bound), and yields the result in the theorem if $d = O(\log n)$. For larger d, we start with a technical result that shows that we can always force d to be $O(\log n)$ without asymptotically increasing the size. From now on in the paper, we will disregard for simplicity the case $k_1 > k_2$ in the rules $A \rightarrow \Pi_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$, as their treatment is analogous to that of the case $k_1 \leq k_2$.

Lemma 5. *If a d-ISLP G generates $T[1\mathinner{\ldotp\ldotp}n]$, then there is also a d'-ISLP G' of the same size that generates T, for some $d' \leq \log_2 n$.*

Proof. For any rule $A = \prod_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$, any $i \in [k_1 \mathinner{\ldotp\ldotp} k_2]$, and any c_j, it holds that $n \geq |\mathtt{exp}(A)| \geq i^{c_j}$, and therefore $c_j \leq \log_i n$, which is bounded by $\log_2 n$ for $i \geq 2$. Therefore, if $k_2 \geq 2$, all the values c_j can be bounded by some $d' \leq \log_2 n$. A rule with $k_1 = k_2 = 1$ is the same as $A \rightarrow B_1 \cdots B_t$, so all values c_j can be set to 0 without changing the size of the rule at all.   □

4.1   Data Structures

We define some data structures that extend ISLPs allowing us to efficiently navigate it within $O(g_{it})$ space. Per Lemma 5, we assume $d = O(\log n)$.

Consider a rule $A \rightarrow \prod_{i=k_1}^{k_2} B_1^{i^{c_1}} \ldots B_t^{i^{c_t}}$. Though t can be large, there are only $d+1$ distinct values c_j. We will make use of auxiliary polynomials

$$f_r(i) = \sum_{j=1}^{r} |\mathsf{exp}(B_j)| \cdot i^{c_j},$$

for $r \in [1, t]$, to navigate within the "blocks" i: $f_r(i)$ computes cumulative lengths inside the product expression $B_1^{i^{c_1}} \ldots B_t^{i^{c_t}}$, up to the variable B_r, for a given i.

We now show how to compute any $f_r(i)$ in time $O(d)$ using $O(t)$ space for each A. An array $S_A[1 .. t]$ stores cumulative length information, as follows $S_A[r] = \sum_{1 \le j \le r, c_j = c_r} |\mathsf{exp}(B_j)|$. That is, $S_A[r]$ adds up the lengths of the symbol expansions up to B_r that must be multiplied by i^{c_r}. A second array, $C_A[1 .. t]$, stores the values $c_1, \ldots, c_t$. We preprocess C_A to solve predecessor queries of the form $pred(A, r, c) = \max\{j \le r, C_A[j] = c\}$, that is, the latest occurrence of c in C_A to the left of position r, for every $c = 0, \ldots, d$. This query can be answered in $O(d)$ time because the elements in C_A are also in $\{0, \ldots, d\}$: cut C_A into chunks of length $d+1$, and for each chunk $C_A[(d+1) \cdot j + 1 .. (d+1) \cdot (j+1)]$ store precomputed values $pred(A, (d+1) \cdot j, c)$ for all $c \in \{0, \ldots, d\}$. This requires $O(t)$ space. To compute the values $r_c = pred(A, r, c)$ for all c, find the chunk $j = \lceil r/(d+1) \rceil - 1$ where r belongs, initialize every $r_c = pred(A, (d+1) \cdot j, c)$ for every c (which is stored with the chunk j), and then scan the chunk prefix $C_A[(d+1) \cdot j + 1 .. r]$ left to right, correcting every $r_c \leftarrow k$ if $c = C_A[k]$, for $k = (d+1) \cdot j + 1 .. r$. We can then evaluate $f_r(i)$ in $O(d)$ time by computing all values r_c as explained (i.e., the last position to the left of r where the exponent is c), and adding up $S_A[r_c] \cdot i^c$ (because $S_A[r_c]$ adds up all $|\mathsf{exp}(B_j)|$ that must be multiplied by i^c in $f_r(i)$). We also define the polynomial

$$f^+(k) = \sum_{i=k_1}^{k} f_t(i)$$

to select a "block": $f^+(k)$ computes the cumulative sum of the length of the whole expressions $B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$ until $i = k$. Note we cannot afford to store all the $k_2 - k_1 + 1$ values $f^+(k)$, but we can exploit the fact that the polynomials $f_t(i)$ have degree at most d, and thus $f^+(k)$ is a polynomial on k of degree at most $d+1$. Storing f^+ as a polynomial, then, requires only $O(d)$ space, instead of the $O(k)$ space needed to store all of its values. This can still be excessive, however, as it blows the space by an $O(\log n)$ factor in a rule like $A \rightarrow \Pi_{i=k_1}^{k_2} B^{i^d}$, which is of size 4 but f^+ is of degree $d+1$. We will instead compute $f^+(k)$ in $O(d)$ arithmetic operations by reusing the same data structures we store for $f_r(i)$: for each $c = 0, \ldots, d$, we compute $t_c = pred(A, t, c)$ and $s_c = S_A[t_c]$. Instead of accumulating $s_c \cdot i^c$, however, we accumulate $s_c \cdot \sum_{i=k_1}^{k} i^c = s_c \cdot (p_c(k) - p_c(k_1 - 1))$, where $p_c(k) = \sum_{i=1}^{k} i^c$.

	1	2	3	4	5	6	7	8
S_A	2	3	6	7	14	13	5	3
C_A	1	2	1	0	0	1	2	3

$f_8(i) = 3i^3 + 5i^2 + 13i + 14$

$f^+(k) = \frac{9}{12}k^4 + \frac{38}{12}k^3 + \frac{117}{12}k^2 + \frac{256}{12}k$

Fig. 1. Data structures built for the ISLP rule $A \to \prod_{i=1}^{5} B^i C^{i^2} D^i EEE^i B^{i^2} C^{i^3}$, with $|\exp(B)| = 2$, $|\exp(C)| = 3$, $|\exp(D)| = 4$, and $|\exp(E)| = 7$. We show some of the polynomials to be simulated with these data structures.

We cannot afford storing all the $O(kd)$ values $p_c(k)$, but since there are only $d + 1 = O(\log n)$ functions p_c and each one is a polynomial of degree $c + 1 = O(\log n)$, they can be represented as polynomials using $O(\log^2 n)$ integers. Further, they can be computed at query time[1], before anything else, in $O(d^2)$ arithmetic operations using, for each c, the formula[2]

$$p_c(k) \quad = \quad k^c + \frac{1}{c+1} \cdot \sum_{j=0}^{c} \binom{c+1}{j} B_j \cdot k^{c+1-j},$$

which is a polynomial on k of degree at most $d + 1$. The formula requires $O(c)$ arithmetic operations once the numbers B_j are computed. Those B_j are the Bernoulli (rational) numbers. All the Bernoulli numbers from B_0 to B_d can be computed in $O(d^2)$ arithmetic operations using the recurrence $\sum_{j=0}^{d} \binom{d+1}{j} B_j = 0$, from $B_0 = 1$. The numerators and denominators of the rationals B_j fit in $O(j \log j) = O(d \log d) = O(\log n \log d)$ bits,[3] so they can be operated in $O(\log d)$ time in a RAM machine with word size $\Theta(\log n)$. Therefore, the total preprocessing time to later compute any $f^+(k)$ is $O(d^2 \log d)$. We note, however, that due to the length of the numerators and denominators of the fractional Bernoulli numbers, the time to compute any $f^+(k)$ is $O(d \log d)$.

Example 1. Consider the ISLP of Proposition 2, defined by the rules $S \to \prod_{i=1}^{k_2} A^i B$, $A \to \mathtt{a}$, and $B \to \mathtt{b}$. The polynomials associated with the representation of the rule S are $i^{c_1} = i$ and $i^{c_2} = 1$. Then, we construct the auxiliary polynomials $f_1(i) = |\exp(A)| i^{c_1} = i$ and $f_2(i) = |\exp(A)| i^{c_1} + |\exp(B)| i^{c_2} = i + 1$. Finally, we construct the auxiliary polynomial $f^+(k) = \sum_{i=1}^{k} f_2(i) = \sum_{i=1}^{k} (i+1) = \frac{1}{2}k^2 + \frac{3}{2}k$. Figure 1 shows a more complex example to illustrate C_A and S_A.

---

[1] Indeed, the polynomials $p_c(k)$ are independent of the grammar, so they can be computed once for all queries and for all grammars.

[2] See Wolfram Mathworld's `https://mathworld.wolfram.com/BernoulliNumber.html`, Eqs. (34) and (47).

[3] See `https://www.bernoulli.org`, sections "Structure of the denominator", "Structure of the nominator", and "Asymptotic formulas".

4.2   Direct Access in Time $O((h + \log n)\, d \log d)$

We start with the simplest query: given the data structures of size $O(g_{it})$ defined in the previous sections, return the symbol $T[l]$ given an index l.

For SLPs with derivation tree of height h, the problem is easily solved in $O(h)$ time by storing the expansion size of every nonterminal, and descending from the root to the corresponding leaf using $|\exp(B)|$ to determine whether to descend to the left or to the right of every rule $A \to BC$. This is easy to generalize in RLSLP rules $A \to B^k$, because every repetition corresponds to the same string, of length $|\exp(B)|$. The general idea for d-ISLPs is similar, but now determining which child to follow in repetition rules is more complex.

To access the l-th character of the expansion of $A \to \prod_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$ we first find the value i such that $f^+(i-1) < l \leq f^+(i)$ by using binary search. Then, we find the value r such that $f_{r-1}(i) < l - f^+(i-1) \leq f_r(i)$ by using binary search in the subindex of the polynomials. We then know that the search follows by B_r, with offset $l - f^+(i-1) - f_{r-1}(i)$ inside $|\exp(B_r)|^{i^{c_r}}$. The offset within B_r is then easily computed with a modulus, as in RLSLPs. Algorithm 1 gives the details.

We carry out the first binary search so that, for every i we try, if $f^+(i) < l$ we immediately answer $i+1$ if $l \leq f^+(i+1)$; instead, if $l \leq f^+(i)$, we immediately answer i if $f^+(i-1) < l$. As a result, the search area is initially of length $|\exp(A)|$ and, if the answer is i, the search has finished by the time the search area is of length $\leq f^+(i) - f^+(i-1) = f_t(i)$. Thus, there are $O(1 + \log(|\exp(A)|/f_t(i)))$ binary search steps. The second binary search is modified analogously so that it carries out $O(1 + \log(f_t(i)/(i^{c_r}|\exp(B_r)|)))$ steps, for a total of at most $O(1 + \log(|\exp(A)|/|\exp(B_r)|))$ steps. As the search continues by B_r, the sum of binary search steps telescopes to $O(h + \log n)$ on an ISLP of height h, and the total time is $O((h + \log n)\, d \log d) = O((h + \log n) \log n \log \log n)$.

Example 2. We show how to access the b at position 14 of the string $T = \prod_{i=1}^{5} \mathtt{a}^i \mathtt{b}$. Consider the ISLP G and its auxiliary polynomials computed in Example 1. We start by computing $f^+(2) = 5$. As $l > 5$, we go right in the binary search and compute $f^+(4) = 14$. As $l \leq 14$ we go left, compute $f^+(3) = 9$ and find that $i = 4$. Hence, $T[l]$ lies in the expansion of $A^i B = A^4 B$ at position $l_1 = l - f^+(i-1) = 5$. Then, we compute $f_1(4) = 4$. As $l_1 > 4$, we turn right and compute $f_2(4) = 5$, finding that $r = 2$. Hence, $T[l]$ lies in the expansion of $B^{i^0} = B^1$ at position $l_2 = l_1 - f_{r-1}(i) = 1$.

4.3   Extracting substrings

Once we have accessed $T[l]$, it is possible to output the substring $T[l \,..\, l + \lambda - 1]$ in $O(\lambda + h)$ additional time, as we return from the recursion in Algorithm 1. We carry the parameter λ of the number of symbols (yet) to output, which is first decremented when we finally arrive at line 3 and find the first symbol, $T[l]$, which we now output immediately. From that point, as we return from the recursion, instead of returning the symbol $T[l]$, we return the number λ of symbols yet to output, doing some extra work until $\lambda = 0$.

Algorithm 1 Direct access for ISLPs in $O((h + \log n)\, d \log d)$ time

---

Input: An ISLP G of height h, a variable A of G, and a position $l \in [1, |\mathtt{exp}(A)|]$.
Output: The character $\mathtt{exp}(A)[l]$ at position l in $\mathtt{exp}(A)$.
 1: **function** ACCESS(G, A, l)
 2:      **if** $A \to a$ **then**
 3:          **return** a
 4:      **if** $A \to BC$ **then**
 5:          **if** $l \leq |\mathtt{exp}(B)|$ **then**
 6:              **return** ACCESS(G, B, l)
 7:          **else**
 8:              **return** ACCESS($G, A, l - |\mathtt{exp}(B)|$)
 9:      **if** $A \to \prod_{i=k_1}^{k_2} B_1^{i^{c_1}} \ldots B_t^{i^{c_t}}$ **then**
10:          $i \leftarrow \mathtt{arg\_successor}([f^+(k_1) \,..\, f^+(k_2)], l)$
11:          $l \leftarrow l - f^+(i-1)$
12:          $r \leftarrow \mathtt{arg\_successor}([f_1(i) \,..\, f_t(i)], l)$
13:          $l \leftarrow l - f_{r-1}(i)$
14:          **return** ACCESS($G, B_r, l \bmod |\mathtt{exp}(B_r)|$)

---

1. If we return from line 5, we output $\min(\lambda, |\mathtt{exp}(C)|)$ symbols from nontermi-nal C, by invoking a new procedure REPORT(G, C, λ), which returns the new number λ of symbols yet to report; this number is then returned by ACCESS.
2. If we return from line 7, we just return the current value of λ to the caller.
3. If we return from line 13, we must report:
   (a) $i^{c_r} - \lceil l/|\mathtt{exp}(B_r)| \rceil$ further copies of $\mathtt{exp}(B_r)$.
   (b) i^{c_s} copies of $\mathtt{exp}(B_s)$, for $s = r+1, \ldots, t$.
   (c) the expansions $\mathtt{exp}(B_1)^{j^{c_1}} \cdots \mathtt{exp}(B_t)^{j^{c_t}}$, for $j = i+1, \ldots, k_2$.
   For each expansion $\mathtt{exp}(C)$ to report, we invoke REPORT(G, C, λ) and update λ to the new number of symbols yet to report. We stop if $\lambda = 0$.

Procedure REPORT(G, C, λ) outputs $\mathtt{exp}(C)$ in $O(|\mathtt{exp}(C)|)$ time if $\lambda \geq |\mathtt{exp}(C)|$, as it simply traverses the leaves of a tree without unary paths. In this case it returns $\lambda - |\mathtt{exp}(C)|$. Otherwise, it traverses only the first λ leaves of the derivation tree of C, in time $O(\lambda + h)$, and returns zero. Once a call to REPORT returns zero, it is never called again; therefore the total time we spend is $O(\lambda + h)$.

5   Balancing ISLPs

We show that any d-ISLP can be balanced so that its derivation tree is of height $O(\log n)$. Actually, we introduce a new type of SLP, which allows us to prove a more general balancing result that subsumes ISLPs.

Definition 3. *A* generalized straight-line program *(GSLP) is an SLP that allows special rules of the form $A \to x$, where x is a* program *(in any Turing-complete language) of length $|x|$ whose output $\mathtt{OUT}(x)$ is a nonempty sequence of variables, none of which can reach A. The rule $A \to x$ contributes $|x|$ to the*

size of the GSLP; the standard SLP rules contribute as usual. If it holds for all special rules that no variable appears exactly once inside $\mathtt{OUT}(x)$*, then the GSLP is said to be* balanceable.

We can choose any desired language to describe the programs x. Though in principle $|x|$ can be taken as the Kolmogorov complexity of $\mathtt{OUT}(x)$, we will focus on very simple programs and on the asymptotic value of x. In particular, RLSLPs allow rules of the form $A \to B^k$ of size 2, and we can have a program of size $O(1)$ that outputs k copies of B; ISLPs allow rules of the form $\prod_{k_1}^{k_2} B_1^{i_1^c} \cdots B_t^{i^{c_t}}$ of size $2 + 2t$, and we can have a program of size $O(t)$ that writes the corresponding $f^+(k_2)$ symbols. Note that in both cases the GSLP is balanceable as long as special rules satisfy $k > 1$ (for RLSLPs), or if $k_1 \neq k_2$ (for ISLPs); otherwise they can be replaced by alternative rules of the same asymptotic size.

We will prove that any balanceable GSLP can be balanced without increasing its asymptotic size. Our proof generalizes that of Ganardi et al. [10, Thm. 1.2] for SLPs in a similar way to how it was extended to balance RLSLPs [28]. Just as Ganardi et al., in this section we will allow SLPs to have rules of the form $A \to B_1 \cdots B_t$, of size t, where each B_j is a terminal or a nonterminal; this can be converted into a strict SLP of the same asymptotic size.

A *directed acyclic graph* (DAG) is a directed multigraph $D = (V, E)$ without cycles (nor loops). We denote by $|D|$ the number of edges in this DAG. For our purposes, we assume that any DAG has a distinguished node r called the *root*, satisfying that any other node can be reached from r and r has no incoming edges. We also assume that if a node has k outgoing edges, they are numbered from 1 to k, so edges are of the form (u, i, v). The *sink nodes* of a DAG are the nodes without outgoing edges. The set of sink nodes of D is denoted by W. We denote the number of paths from u to v as $\pi(u, v)$, and $\pi(u, V) = \sum_{v \in V} \pi(u, v)$ for a set V of nodes. The number of paths from the root to the sink nodes is $n(D) = \pi(r, W)$.

One can interpret an SLP G generating a string T as a DAG D: There is a node for each variable in the SLP, the root node is the initial variable, variables of the form $A \to a$ are the sink nodes, and a variable with rule $A \to B_1 B_2 \ldots B_t$ has outgoing edges (A, i, B_i) for $i \in [1, t]$. Note that if D is a DAG representing G, then $n(D) = |\mathtt{exp}(G)| = |T|$.

Definition 4. *(Ganardi et al. [10, page 5]) Let D be a DAG, and define the pairs $\lambda(v) = (\lfloor \log_2 \pi(r, v) \rfloor, \lfloor \log_2 \pi(v, W) \rfloor)$. The* symmetric centroid decomposition *(SC-decomposition) of a DAG D produces a set of edges between nodes with the same λ pairs defined as $E_{scd}(D) = \{(u, i, v) \,|\, \lambda(u) = \lambda(v)\}$, partitioning D into disjoint paths called* SC-paths *(some of them possibly of length 0).*

The set E_{scd} can be computed in $O(|D|)$ time. If D is the DAG of an SLP G, then $|D|$ is $O(|G|)$. The following lemma justifies the name "SC-paths".

Lemma 6. *(Ganardi et al. [10, Lemma 2.1]) Let $D = (V, E)$ be a DAG. Then every node has at most one outgoing and at most one incoming edge from $E_{scd}(D)$. Furthermore, every path from the root r to a sink node contains at most $2 \log_2 n(D)$ edges that do not belong to $E_{scd}(D)$.*

Note that the sum of the lengths of all SC-paths is at most the number of nodes of the DAG, or equivalently, the number of variables of the SLP.

The following definition and technical lemma are needed to construct the building blocks of our balanced GSLPs.

Definition 5. *(Ganardi et al. [10, page 7]) A* weighted string *is a string $T \in \Sigma^*$ equipped with a* weight function $|| \cdot || : \Sigma \to \mathbb{N} \backslash \{0\}$*, which is extended homomorphically. If A is a variable in an SLP G, then we write $||A||$ for the weight of the string $\exp(A)$ derived from A.*

Lemma 7. *(Ganardi et al. [10, Proposition 2.2]) For every non-empty weighted string T of length n one can construct in linear time an SLP G generating T with the following properties:*

- *G contains at most $3n$ variables*
- *All right-hand sides of G have length at most 4*
- *G contains suffix variables $S_1, ..., S_n$ producing all non-trivial suffixes of T*
- *every path from S_i to some terminal symbol a in the derivation tree of G has length at most $3 + 2(\log_2 ||S_i|| - \log_2 ||a||)$*

Theorem 2. *Given a balanceable GSLP G generating a string T, it is possible to construct an equivalent GSLP G' of size $O(|G|)$ and height $O(\log n)$.*

Proof. Transform the GSLP G into an SLP H by replacing their special rules $A \to x$ by $A \to \texttt{OUT}(x)$, and then obtain the SC-decomposition $E_{scd}(D)$ of the DAG D of H. Observe that the SC-paths of H use the same variables of G, so it holds that the sum of the lengths of all the SC-paths of H is less than the number of variables of G. Also, note that any special variable $A \to x$ of G is necessarily the endpoint (i.e., the last node of a directed path) of an SC-path in D. To see this note that $\lambda(A) \neq \lambda(B)$ for any B that appears in $\texttt{OUT}(x)$, because $\log_2 \pi(A, W) \geq \log_2(|\texttt{OUT}(x)|_B \cdot \pi(B, W)) \geq 1 + \log_2 \pi(B, W)$ where $|\texttt{OUT}(x)|_B \geq 2$ because G is balanceable. This implies that the balancing procedure of Ganardi et al. on H, which transforms the rules of variables that are not the endpoint of an SC-path in the DAG D, will not touch variables that were originally special variables in G.

Let $\rho = (A_0, d_0, A_1), (A_1, d_1, A_2), \ldots, (A_{p-1}, d_{p-1}, A_p)$ be an SC-path of D. It holds that for each A_i with $i \in [0 \mathinner{.\,.} p - 1]$, in the SLP H its rule goes to two distinct variables, one to the left and one to the right. Thus, for each variable A_i, with $i \in [0 \mathinner{.\,.} p - 1]$, there is a variable A'_{i+1} that is not part of the path. Let $A'_1 A'_2 \ldots A'_p$ be the sequence of these variables. Let $L = L_1 L_2 \ldots L_s$ be the subsequence of left variables of the previous sequence. Then construct an SLP of size $O(s) \subseteq O(p)$ for the sequence L (seen as a string) as in Lemma 7, using $|\exp(L_i)|$ in H as the weight function. In this SLP, any path from the suffix nonterminal S_i to a variable L_j has length at most $3 + 2(\log_2 ||S_i|| - \log_2 ||L_j||)$. Similarly, construct an SLP of size $O(t) \subseteq O(p)$ for the sequence $R = R_1 R_2 \ldots R_t$ of right symbols in reverse order, as in Lemma 7, but with prefix variables P_i instead of suffix variables. Each variable A_i, with $i \in [0 \mathinner{.\,.} p-1]$, derives the same

string as $w_l A_p w_r$, for some suffix w_l of L and some prefix w_r of R. We can find rules deriving these prefixes and suffixes in the SLPs produced in the previous step, so for any variable A_i, we construct an equivalent rule of length at most 3. Add these equivalent rules, and the left and right SLP rules to a new GSLP G'. Do this for all SC-paths. Finally, add the original terminal variables and special variables (which are left unmodified) of the GSLP G, so G' is equivalent to G.

The SLP constructed for L has all its rules of length at most 4, and $3s \leq 3p$ variables. The same happens with R. The other constructed rules also have a length of at most 3, and there are p of them. Summing over all SC-paths, we have $O(|G|)$ size. The special variables cannot sum up to more than $O(|G|)$ size. Thus, the GSLP G' has size $O(|G|)$.

Any path in the derivation tree of G' is of length $O(\log n)$. To see why, let $A_0, \ldots, A_p$ be an SC-path. Consider a path from a variable A_i to an occurrence of a variable that is in the right-hand side of A_p in G'. Clearly, this path has length at most 2. Now consider a path from A_i to a variable A'_j in L with $i < j \leq p$. By construction this path is of the form $A_i \rightarrow S_k \rightarrow^* A'_j$ for some suffix variable S_k (if the occurrence of A'_j is a left symbol), and its length is at most $1 + 3 + 2(\log_2 \|S_k\| - \log_2 \|A'_j\|) \leq 4 + 2\log_2 \|A_i\| - 2\log_2 \|A'_j\|$. Analogously, if A'_j is a right variable, the length of the path is bounded by $1+3+2(\log_2 \|P_k\| - \log_2 \|A'_j\|) \leq 4+2\log_2 \|A_i\| - 2\log_2 \|A'_j\|$. Finally, consider a maximal path to a leaf in the derivation tree of G'. Factorize it as $A_0 \rightarrow^* A_1 \rightarrow^* \cdots \rightarrow^* A_k$ where each A_i is a variable of H (and also of G). Paths $A_i \rightarrow^* A_{i+1}$ are like those defined in the paragraph above, satisfying that their length is bounded by $4 + 2\log_2 \|A_i\| - 2\log_2 \|A_{i+1}\|$. Observe that between each A_i and A_{i+1}, in the DAG D there is almost an SC-path, except that the last edge is not in E_{scd}. The length of this path is at most

$$\sum_{i=0}^{k-1}(4 + 2\log_2 \|A_i\| - 2\log_2 \|A_{i+1}\|) \leq 4k + 2\log_2 \|A_0\| - 2\log_2 \|A_k\|$$

By Lemma 6, $k \leq 2\log_2 n$, which yields the upper bound $O(\log n)$.

The resulting GSLP can be modified to contain standard SLP rules of size at most two, with only a constant increase in size and depth.   □

By the above theorem, Lemma 5, and because ISLPs can be made balanceable, we obtain the following.

Corollary 1. *Given a d-ISLP G generating a string T, there is an equivalent d'-ISLP G' of size $O(|G|)$, with $d' \leq d$, $d' = O(\log n)$, and height $h' = O(\log n)$.*

6   Conclusions

We have introduced a new extension to straight-line programs (SLPs) and run-length SLPs (RLSLPs) called *iterated SLPs (ISLPs)*. ISLPs permit so-called *iteration rules* of the form $A \rightarrow \Pi_{i=k_1}^{k_2} B_1^{i^{c_1}} \cdots B_t^{i^{c_t}}$, of size $O(t)$. While it had already been shown that the lower-bound (and unreachable) measure δ, which

was text-wise smaller than every preceding measure of repetitiveness, could be outperformed by a *reachable* measure (L-systems) on some text families [26,27], the size g_{it} of the smallest ISLP generating a text is the first *accessible* measure that also outperforms δ (by the same margin, $O(\delta/\sqrt{n})$ on a text of length n).

With SLPs or RLSLPs representing a text $T[1 \mathbin{.\,.} n]$, an arbitrary symbol of T can be accessed in $O(\log n)$ time. We have shown that, just as SLPs and RLSLPs [10,28], ISLPs can be balanced without asymptotically increasing their space, and used it to devise an algorithm to access any arbitrary text position in time $O(\log^2 n \log \log n)$ within $O(g_{it})$ space. They are also similarly resistant to edits and other text manipulations.

References

1. Akagi, T., Funakoshi, M., Inenaga, S.: Sensitivity of string compressors and repetitiveness measures. Information and Computation **291**, 104999 (2023)
2. Bannai, H., Funakoshi, M., I, T., Köppl, D., Mieno, T., Nishimoto, T.: A separation of γ and b via Thue-Morse words. In: Proc. 28th International Symposium on String Processing and Information Retrieval (SPIRE). pp. 167–178 (2021)
3. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Rao, S.S., Weimann, O.: Random access to grammar-compressed strings and trees. SIAM Journal on Computing **44**(3), 513–539 (2015)
4. Burrows, M., Wheeler, D.: A block sorting lossless data compression algorithm. Tech. Rep. 124, Digital Equipment Corporation (1994)
5. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. IEEE Transactions on Information Theory **51**(7), 2554–2576 (2005)
6. Christiansen, A.R., Ettienne, M.B., Kociumaka, T., Navarro, G., Prezza, N.: Optimal-time dictionary-compressed indexes. ACM Transactions on Algorithms **17**(1), article 8 (2020)
7. Fici, G., Romana, G., Sciortino, M., Urbina, C.: On the Impact of Morphisms on BWT-Runs. In: Bulteau, L., Lipták, Z. (eds.) 34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 259, pp. 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023)
8. Gagie, T., Navarro, G., Prezza, N.: Fully-functional suffix trees and optimal text searching in BWT-runs bounded space. Journal of the ACM **67**(1), article 2 (2020)
9. Gallant, J.K.: String Compression Algorithms. Ph.D. thesis, Princeton University (1982)
10. Ganardi, M., Jeż, A., Lohrey, M.: Balancing straight-line programs. J. ACM **68**(4) (jun 2021)
11. Giuliani, S., Inenaga, S., Lipták, Z., Prezza, N., Sciortino, M., Toffanello, A.: Novel results on the number of runs of the burrows-wheeler-transform. In: Bureš, T., Dondi, R., Gamper, J., Guerrini, G., Jurdziński, T., Pahl, C., Sikora, F., Wong, P.W. (eds.) SOFSEM 2021: Theory and Practice of Computer Science. pp. 249–262. Springer International Publishing, Cham (2021)
12. Giuliani, S., Inenaga, S., Lipták, Z., Romana, G., Sciortino, M., Urbina, C.: Bit catastrophes for the burrows-wheeler transform. In: Drewes, F., Volkov, M. (eds.) Developments in Language Theory. pp. 86–99. Springer Nature Switzerland, Cham (2023)

13. Kempa, D., Kociumaka, T.: Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. CoRR **2308.03635** (2023)
14. Kempa, D., Prezza, N.: At the roots of dictionary compression: String attractors. In: Proc. 50th Annual ACM Symposium on the Theory of Computing (STOC). pp. 827–840 (2018)
15. Kempa, D., Saha, B.: An upper bound and linear-space queries on the LZ-End parsing. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2847–2866 (2022)
16. Kempa, D., Kociumaka, T.: Resolution of the burrows-wheeler transform conjecture. In: Proc. 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS). pp. 1002–1013 (2020)
17. Kociumaka, T., Navarro, G., Olivares, F.: Near-optimal search time in δ-optimal space, and vice versa. CoRR **2206.00781** (2023)
18. Kociumaka, T., Navarro, G., Prezza, N.: Towards a definitive compressibility measure for repetitive sequences. IEEE Transactions on Information Theory **69**(4), 2074–2092 (2023)
19. Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theoretical Computer Science **483**, 115–133 (2013)
20. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Transactions on Information Theory **22**(1), 75–81 (1976)
21. Lindenmayer, A.: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. Journal of Theoretical Biology **18**(3), 280–299 (1968)
22. Lindenmayer, A.: Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. Journal of Theoretical Biology **18**(3), 300–315 (1968)
23. Navarro, G.: Indexing highly repetitive string collections, part I: Repetitiveness measures. ACM Computing Surveys **54**(2), article 29 (2021)
24. Navarro, G.: Indexing highly repetitive string collections, part II: Compressed indexes. ACM Computing Surveys **54**(2), article 26 (2021)
25. Navarro, G., Ochoa, C., Prezza, N.: On the approximation ratio of ordered parsings. IEEE Transactions on Information Theory **67**(2), 1008–1026 (2021)
26. Navarro, G., Urbina, C.: On stricter reachable repetitiveness measures. In: Proc. 28th International Symposium on String Processing and Information Retrieval (SPIRE). pp. 193–206 (2021)
27. Navarro, G., Urbina, C.: L-systems for measuring repetitiveness. In: Proc. 34th Annual Symposium on Combinatorial Pattern Matching (CPM). p. article 14 (2023)
28. Navarro, G., Olivares, F., Urbina, C.: Balancing run-length straight-line programs. In: Proc. 29th International Symposium on String Processing and Information Retrieval (SPIRE). pp. 117–131 (2022)
29. Raskhodnikova, S., Ron, D., Rubinfeld, R., Smith, A.D.: Sublinear algorithms for approximating string compressibility. Algorithmica **65**(3), 685–709 (2013)
30. Sipser, M.: Introduction to the Theory of Computation. Introduction to the Theory of Computation, Cengage Learning (2012)
31. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. Journal of the ACM **29**(4), 928–951 (1982)