Computing NP-Hard Repetitiveness Measures via MAX-SAT

Hideo Bannai
Tokyo Medical and Dental University, Japan

Keisuke Goto
Independent Researcher, Tokyo, Japan

Masakazu Ishihata
NTT Communication Science Laboratories, Kyoto, Japan

Shunsuke Kanda
Independent Researcher, Tokyo, Japan

Dominik Köppl
Tokyo Medical and Dental University, Japan

Takaaki Nishimoto
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Abstract

Repetitiveness measures reveal profound characteristics of datasets, and give rise to compressed data structures and algorithms working in compressed space. Alas, the computation of some of these measures is NP-hard, and straight-forward computation is infeasible for datasets of even small sizes. Three such measures are the smallest size of a string attractor, the smallest size of a bidirectional macro scheme, and the smallest size of a straight-line program. While a vast variety of implementations for heuristically computing approximations exist, exact computation of these measures has received little to no attention. In this paper, we present MAX-SAT formulations that provide the first non-trivial implementations for exact computation of smallest string attractors, smallest bidirectional macro schemes, and smallest straight-line programs. Computational experiments show that our implementations work for texts of length up to a few hundred for straight-line programs and bidirectional macro schemes, and texts even over a million for string attractors.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases repetitiveness measures, string attractor, bidirectional macro scheme

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.12

Supplementary Material *Software (Source Code)*: `https://github.com/kg86/satcomp`
archived at `swh:1:dir:9f4f16f948f46118492771d403a0ca880a7742ab`

Funding *Hideo Bannai*: Supported by JSPS KAKENHI Grant Number JP20H04141.
Dominik Köppl: Supported by JSPS KAKENHI Grant Numbers JP21H05847 and JP21K17701.

1 Introduction

Text compression is a fundamental topic in computer science with countless practical applications. *Dictionary compression* is a type of text compression where the original input is transformed into a sequence of elements taken from a dictionary, where the dictionary is usually constructed in some way from the input. Due to the advent of *highly repetitive* datasets such as multiple genome sequences from the same species or versioned document collections (e.g., Wikipedia, GitHub), dictionary compression methods have recently (re)gained massive attention since they can better capture more widespread repetitions in such data compared to statistical compression methods [30], and further allow space-efficient full-text indices to be built [31]. Some well known methods that fall in this category are Lempel–Ziv 76/77 factorization based methods [20,23,43], grammar-based compression such as LZ78 [44], Re-Pair [22], SEQUITUR [34], LCA [38], LZD [11], and methods involving bidirectional referencing, such as the run-length encoded Burrows–Wheeler transform (RLBWT) [25], and more recently, lcpcomp [10], plcpcomp [9], lexcomp [32], a method by Russo et al. [36], and LZRR [35].

A vital issue in evaluating and comparing these various methods is to understand how well they can compress a given input compared to the "optimum". While the theoretically smallest representation (aka Kolmogorov complexity) is incomputable [24], Kempa and Prezza [16] regarded the output sizes of these methods as *repetitiveness measures* and characterized them with respect to the new notion of *string attractors*. Namely, they showed that for any input text, the size of the smallest string attractor is a lower bound for the output sizes of all known dictionary compressors. Since then, relations between these various repetitiveness measures have been heavily investigated [2,4,14,17,19,30,32].

In this paper, we consider three such repetitiveness measures: the size γ of the smallest string attractor, the size g of the smallest straight-line program (SLP) [13], and the size b of the smallest bidirectional macro scheme (BMS) [42], all of which are known to be NP-hard to compute [16,39,42]. Thus, any efficient dictionary compression algorithm can (most likely) merely compute approximations of γ, b, or g. Although for any text, the relation $\delta \leq \gamma \leq b \leq z \leq g$ is known, where δ [19] and z [23] are repetitiveness measures known to be computable in linear time (cf. [7, Lemma 5.7] for δ and [8] for z), the gap between the measures can be quite large; string families giving a logarithmic factor gap are known for each pair of measures [2,30]. Since the sizes of some recent data structures such as [7,33], depend on these repetitiveness measures, their exact sizes are crucial knowledge.

While there exist a vast variety of approximation algorithms for computing smallest BMSs and grammars as mentioned above, development of exact algorithms have received very little to almost no attention. For string attractors, the results of Kempa et al. [15] imply a straightforward $O(n2^n)$ time algorithm. For the smallest grammar, Casel et al. [6, Theorem 13] show an $O^*(3^n)$[1] time algorithm. However, we are unaware of any non-trivial implementations or empirical evaluations for computing these measures. In fact, the only publicly available implementation we could find was a straight-forward Python script to compute γ by Michael S. Branicky [12].

The main contribution of this paper is to present MAX-SAT formulations [3] for computing the smallest string attractor, BMS, and SLP, thereby providing the first non-trivial implementation for exact computation of the measures γ, b, and g. The rationale for this approach is that although MAX-SAT is NP-hard, there are highly optimized solvers whose performance has made incredible progress in recent years. These solvers can cope with very large instances and can be leveraged, provided that suitable encodings can be designed [18]. While straight-forward (non-MAX-SAT) implementations become infeasible even for very small text lengths (e.g. 40), computational experiments show that our implementations work for texts of length up to a few hundred for b, g, and even more than 1 million for γ. Since our addressed problems are all NP-hard, there is perhaps little hope for our implementations to obtain exact solutions for larger but practically interesting datasets. Nevertheless, we believe they can make significant impact as a tool for analyzing these repetitiveness measures. We stress that our solutions not only report the sizes γ, b and g, but also give valid instances having exactly these sizes (e.g., an SLP that has size g). It may therefore be possible to

[1] The abstract of [6] mentions $O(3^n)$ while the statement of the theorem is $O^*(3^n)$.

improve compression heuristics by studying some of these optimal instances on smaller input strings. As an example application, we analyzed the recently introduced notion of *sensitivity* [1] of γ by conducting an exhaustive computation of γ for strings up to certain lengths. From these computations, we were able to discover a family of strings that exhibit a multiplicative sensitivity of 2.5, improving the previously known lower bound of 2.0 [1].

Related Work

The exact values for γ, b, and g have been characterized only for a few families of strings. For standard Sturmian words, $\gamma = 2$ [26] and $b = O(1)$ since the RLBWT has constant size [27] and can be regarded as a BMS. For the nth Thue–Morse word, $\gamma = 4$ for $n \geq 4$ [21], and $b = n + 2$ for $n \geq 2$ [2]. For the nth Fibonacci word, $g = n$ [28]. The smallest attractor sizes of automatic sequences have also been studied [40].

2 Preliminaries

Let Σ be a set of σ symbols called the *alphabet*, and let Σ^* denote the set of strings over Σ. Given a string T, if $T = xyz$ for strings x, y, z, then x, y, z are respectively called a *prefix*, *substring*, and *suffix* of T. They are called *proper* if they are not equal to T. The length of T is denoted by $|T|$. For any $i \in [1, |T|]$, let $T[i]$ denote the ith symbol of T, i.e., $T = T[1] \cdots T[|T|]$. For any $1 \leq i \leq j \leq |T|$, let $T[i..j] = T[i] \cdots T[j]$ and $T[i..j) = T[i] \cdots T[j-1]$.

For the rest of this paper, we fix a string T, and let $n := |T|$ denote its length. Further, we assume that each symbol of Σ appears in T. Let $\mathit{occ}(P) = \{i \mid T[i..i+|P|-1] = P, 1 \leq i \leq n - |P| + 1\}$ be the set of starting positions of all occurrences of a substring P in T, and let $\mathit{cover}(P) = \{i + k - 1 \mid i \in \mathit{occ}(P), 1 \leq k \leq |P|\}$ be the set of all text positions covered by all occurrences of P in T.

A set of positions $\Gamma \subseteq [1, n]$ is a *string attractor* [16] of T if every substring P of T has an occurrence in T that contains an element of Γ, that is, $\Gamma \cap \mathit{cover}(P) \neq \emptyset$. We denote the size of the smallest string attractor of T by γ. For example, $[1, n]$ is a trivial string attractor. $\{1, 2, 3\}$ is a (smallest) string attractor of $T = \texttt{banana}$. (See also Figure 2)

A *straight-line program* (SLP) [13] is a grammar in Chomsky normal form whose language consists solely of T. In other words, (1) each production rule is of the form $X \to X_\ell X_r$ or $X \to c$, where X_ℓ, X_r are non-terminals and $c \in \Sigma$, (2) there is exactly one such production rule for any given non-terminal symbol X, and (3) there is a start symbol whose iterative expansion finally leads to T. The *size* of an SLP is the number of its production rules, or equivalently (assuming that each non-terminal is used at least once), the number of distinct non-terminals. We denote the size of the smallest SLP that produces T by g. For example, the set of production rules $\{X_9 \to X_6X_8,\ X_8 \to X_7X_7,\ X_7 \to X_1X_3,\ X_6 \to X_4X_5,\ X_5 \to X_3X_3,\ X_4 \to X_3X_1,\ X_3 \to X_1X_2,\ X_2 \to \texttt{b},\ X_1 \to \texttt{a}\}$ is an SLP of size 9 for $T = \texttt{abaababaabaab}$. See also Figure 3.

A *bidirectional macro scheme* (BMS) [42] of size m representing T, is a factorization $T = F_1, \cdots, F_m$, where each factor (or phrase) is a single symbol (which we call a ground phrase), or, is encoded as a pair of integers (i, j) indicating that it references (i.e., is a copy of) substring $T[i..j]$. A BMS is said to be *valid*, if T can be reconstructed from the representation of such a factorization, i.e., the implied references of each symbol in a non-ground phrase is acyclic, and eventually leads to a ground phrase. We denote the size of the smallest valid BMS that represents T by b. Figure 1 shows a valid BMS $(7, 8), (4, 5), \texttt{a}, \texttt{b}, (5, 7)$ representing the string $\texttt{abaaababa}$. For example, the $\texttt{a}$ at position 9 references position 7, which in turn references position 5, a ground phrase.

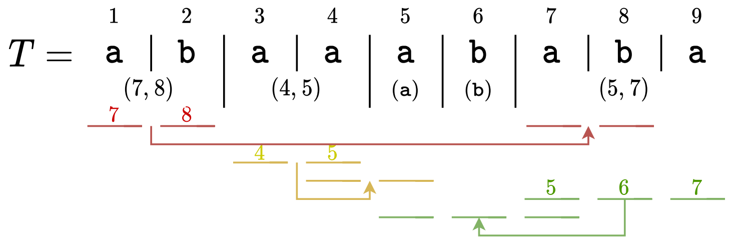

Figure 1 A bidirectional macro scheme (BMS) of $T =$ abaaababa. The figure depicts the BMS $(7,8),(4,5),$ a, b, $(5,7)$. The reference of each non-ground phrase is visualized by an arrow. The phrase references imply a reference for each symbol in non-ground phrases.

The satisfiability (SAT) problem asks for an assignment of variables that satisfies a given Boolean formula [3, 18]. The input formula is usually given in *conjunctive normal form* (CNF), which consists of a conjunction of clauses, and each clause is a disjunction of literals. A *literal* is a Boolean variable or its negation. In this form, the given formula is satisfied if and only if all the clauses (which we will sometimes call constraints) are satisfied. The *size* of a CNF is the sum of the literals in all clauses.

A *maximum satisfiability* (MAX-SAT) problem is an extension of SAT, where two types of clauses, *hard* and *soft*, are considered [3]. A solution to a MAX-SAT instance is a truth assignment of the variables such that the number of satisfied soft clauses is maximized under the restriction that all hard clauses must be satisfied.

We will use 1 to denote true, and 0 to denote false. Furthermore, for a set $\{v_i\}_{i=1}^k$ of Boolean variables, cardinality constraints of the form $\sum_{i=1}^k v_i \leq 1$ are known as *atmost-one constraints*. Although a straightforward encoding has size $\Theta(k^2)$, $O(k)$ size encodings are known [41]. Constraints of the form $\sum_{i=1}^k v_i = 1$ can be encoded using a combination of an atmost-one constraint and a simple disjunction of all the variables (i.e., atleast-one) and thus can also be encoded in $O(k)$ size.

3 Reductions to MAX-SAT

In what follows, we present our encodings for the aforementioned problems. Common to all encodings is the idea that we have a Boolean variable p_i for each text position $i \in [1, n]$, which counts, when set to true, an element of a string attractor, a non-terminal (actually, to be precise, a factor in a grammar parsing) of an SLP, or a phrase of a BMS. Since our goal is to have as few p_i's set to true as possible, our soft clauses have the form $D_i = \neg p_i$ for $i \in [1, n]$. Consequently, all our encodings have the same number of soft clauses, and only differ in how the hard clauses are defined.

3.1 Smallest String Attractor as MAX-SAT

We start with a simple encoding based on the definition of string attractors. Subsequently, we utilize an observation similar to but slightly more generalized than that made in [15], in order to reduce the size of hard clauses.

3.1.1 Simple Encoding

Our idea is to design a CNF so that a MAX-SAT solution will encode a string attractor Γ, where $p_i = 1$ if and only if position i is an element of Γ (i.e., $\Gamma = \{i \mid 1 \leq i \leq n, p_i = 1\}$). Let $\mathcal{S}_T$ denote the set of all non-empty substrings of T, i.e., $\mathcal{S}_T = \{T[i..j] \mid 1 \leq i \leq j \leq n\}$.

For each substring S of $\mathcal{S}_T$, we define a hard clause $C_S = \bigvee_{i \in cover(S)} p_i$. (See Figure 2 for an example.) By the definition of $cover(S)$, the set Γ corresponding to any truth assignment for p_i will be a string attractor if and only if all hard clauses C_S are satisfied. Since our soft clauses have the form $D_i = \neg p_i$ for $i \in [1, n]$, the soft clauses ensure that the MAX-SAT solution minimizes the number of p_i's being true. Thus, we can obtain the smallest string attractor by solving the MAX-SAT on C_S and D_i.

Each hard clause C_S has size $|cover(S)| = O(n)$. Since there are $O(n^2)$ substrings, the number of hard clauses is $O(n^2)$. Hence, the total size of the CNF is $O(n^3)$. In the next subsection, we reduce the size to $O(n^2)$.

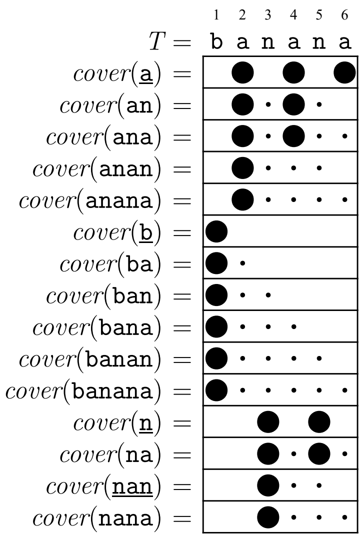

Figure 2 String $T =$ `banana` and the positions that each distinct substring of T covers. We list all distinct substrings of T on the left hand side, and show on the right hand side their covers. A dot at position k in the row for substring S indicates that k is included in $cover(S)$ (i.e. is covered by S), and a large dot indicates $k \in occ(S)$. Underlined substrings are minimal substrings of T. For example, $cover(\texttt{an}) = \{2, 3, 4, 5\}$. The clause defined for `an` in our encoding is $C_{\texttt{an}} = p_2 \vee p_3 \vee p_4 \vee p_5$.

3.1.2 Reducing CNF Clauses via Minimal Substrings

We can reduce the number of hard clauses in our CNF by considering only members of $\mathcal{S}_T$ that are *minimal substrings*[2]. A substring S of string T is called a minimal substring of T if all proper substrings of S occur more often than S in T (i.e., $|occ(S[i..j])| > |occ(S)|$ for every proper substring $S[i..j]$ of S). By the definition of minimal substrings, the following lemma holds.

▶ **Lemma 1.** *For every non-minimal substring S of T, there is a minimal substring $S_{\min}$ of S with $cover(S_{\min}) \subseteq cover(S)$.*

[2] Kempa et al. [15] use a similar idea when reducing the problem to set cover. Their formulation can be regarded as considering only right-minimal substrings (i.e., $|occ(S[1..|S|-1])| > |occ(S)|$), while we consider a potentially smaller subset requiring both right-minimality and left-minimality. For texts in the Calgary corpus, we observed that the difference between minimal and right-minimal substrings can result in a difference as large as 50 times in their total lengths (progp and trans), i.e., the total size of hard clauses.

Proof. Because S is not minimal, it has substrings that have the same number of occurrences as S. Let $S_{\mathsf{min}} = S[e..e+|S_{\mathsf{min}}|-1]$ be one of these substrings that is minimal, for some e. Then by definition, for each occurrence $i_{\mathsf{min}} \in \mathit{occ}(S_{\mathsf{min}})$ of S_{min}, there exists an occurrence $i \in \mathit{occ}(S)$ of S such that $i = i_{\mathsf{min}} - e + 1$. $\{i_{\mathsf{min}}, i_{\mathsf{min}}+1, \ldots, i_{\mathsf{min}} + |S_{\mathsf{min}}| - 1\} \subseteq \{i, i+1, \ldots, i+|S|-1\}$, and hence, $\mathit{cover}(S_{\mathsf{min}}) \subseteq \mathit{cover}(S)$. ◀

In the example $T = \mathtt{banana}$, $|\mathit{occ}(\mathtt{nan})| < |\mathit{occ}(\mathtt{na})|, |\mathit{occ}(\mathtt{an})|, |\mathit{occ}(\mathtt{a})|, |\mathit{occ}(\mathtt{n})|$, and thus, substring `nan` is a minimal substring of T. Furthermore, `nan` is a substring of `nana`, and $|\mathit{occ}(\mathtt{nana})| = |\mathit{occ}(\mathtt{nan})|$. Thus, $\mathit{cover}(\mathtt{nan}) \subseteq \mathit{cover}(\mathtt{nana})$ by Lemma 1. (See also Figure 2)

Lemma 1 ensures that if an assignment of variables satisfies the hard clauses C_S for all minimal substrings S of T, then the assignment satisfies the hard clauses C_S for all substrings S of T. With this observation we can conclude that we can omit the hard clauses for all substrings S of T that are not minimal.

The number m of minimal substrings is $O(n)$ because minimal substrings correspond to minimal strings, defined by Blumer et al. [5] based on an equivalence relation over substrings of T, and their number is known to be $O(n)$ (Lemma 3 in [29]). Hence, the total size of the CNF is reduced to $O(mn) \subseteq O(n^2)$.

In particular, the size of the CNF is $o(n^2)$ if $m = o(n)$. We can show that there exists a family of strings $\{T_d\}_{d\in\mathcal{I}}$ for a non-finite set of natural numbers $\mathcal{I}$ with $|T_d| = d^2$ having $o(d^2)$ minimal substrings (hence, for $n = d^2$, $m = o(n)$). To this end, let T_d be the string $S_1 S_2 \cdots S_d$ of length $n = d^2$ over the alphabet $\Sigma = \{\mathtt{a}, \$_1, \$_2, \ldots, \$_d\}$, where $S_i = \mathtt{a}^{d-1}\$_i$, and $\mathtt{a}^{d-1}$ is the repetition of character `a` with length $d-1$. Then $m = 2\sqrt{n} - 1$ because the minimal substrings of T_d are $\mathtt{a}^1$, $\mathtt{a}^2$, …, $\mathtt{a}^{\sqrt{n}-1}$, $\$_1$, $\$_2$, …, $\$_{\sqrt{n}}$.

3.2 Smallest Straight-Line Program as MAX-SAT

To encode a grammar in SAT, we utilize a notion called *grammar parsing* introduced by Rytter [37]. Given an SLP G that produces T, the *parse tree* of T with respect to G is a derivation tree of T, where internal nodes are non-terminal symbols that derive two non-terminal symbols, and leaves are non-terminal symbols that derive a single terminal symbol. The *partial parse tree* of T with respect to G is the tree obtained by pruning the parse tree of T with respect to G so that any internal node is always a first occurrence in a left to right pre-order traversal of the parse tree, i.e., the non-terminal symbol of an internal node is not used in the partial parse tree for any corresponding substring to its left. In other words, if a non-terminal symbol X that derives two non-terminal symbols is a leaf of the partial parse tree, the existence of a unique internal node having the same non-terminal symbol X corresponding to a substring to its left is implied. We will say that the leaf *references* the internal node. The *grammar parsing* of T with respect to G, is the factorization of T consisting of substrings corresponding to the leaves of the partial parse tree of T with respect to G. See Figure 3 for an example.

The *size* of the grammar parsing is equal to the number of leaves in the partial parse tree. It is easy to see that by definition, the internal nodes in the partial parse tree are distinct, consisting of (all) non-terminal symbols that derive two non-terminal symbols. There are σ more non-terminal symbols that derive a single terminal symbol. Therefore, (# of internal nodes) $+\, \sigma$ is the size of the SLP. Since the partial parse tree is a full binary tree, (# of internal nodes) $=$ (# of leaves) $-\, 1$, and thus the size of the SLP is equal to (size of the grammar parsing) $+\, \sigma - 1$. As σ is independent of the choice of the SLP for T, minimizing the size of the grammar parsing is equivalent to minimizing the SLP.

Our formulation is based on the following lemma.

$$\begin{aligned}
X_9 &\to X_6X_8\\
X_8 &\to X_7X_7\\
X_7 &\to X_1X_3\\
X_6 &\to X_4X_5\\
X_5 &\to X_3X_3\\
X_4 &\to X_3X_1\\
X_3 &\to X_1X_2\\
X_2 &\to \mathtt{b}\\
X_1 &\to \mathtt{a}
\end{aligned}$$

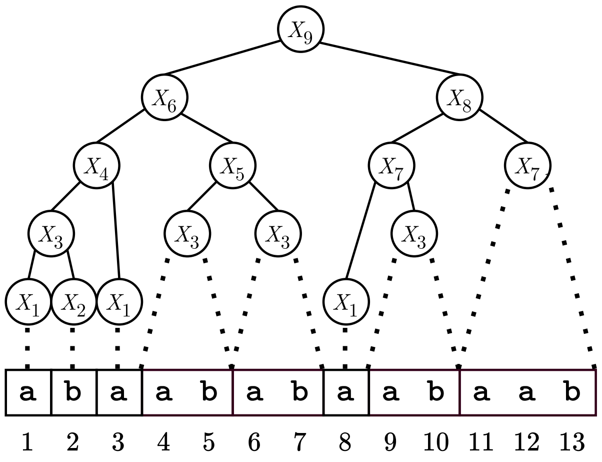

Figure 3 The partial parse tree and the grammar parsing of an SLP for the string $T =$ `abaababaabaab`. Each internal node is a unique non-terminal symbol. The grammar parsing represented by the rectangles partitioning T is `a`, `b`, `a`, `ab`, `ab`, `a`, `ab`, `aab` of size 8. The size of the SLP is $8 + |\{\mathtt{a}, \mathtt{b}\}| - 1 = 9$.

▶ **Lemma 2.** *A factorization $T = F_1 \cdots F_m$ for T is the grammar parsing of an SLP for T if and only if (i) for each factor F_k longer than 1, there exist $i_k < j_k < k$ such that: $F_k = F_{i_k} \cdots F_{j_k}$ and (ii) for any pair of factors $F_x = F_{i_x} \cdots F_{j_x}$ and $F_y = F_{i_y} \cdots F_{j_y}$ longer than 1, (i.e., $(x, y) \in \{(x', y') \mid 1 \le x', y' \le m, |F_{x'}| > 1, |F_{y'}| > 1\}$), the intervals $[i_x..j_x]$ and $[i_y..j_y]$ are either disjoint or one is a sub-interval of the other.*

Proof. ($\Rightarrow$) Suppose $F_1 \cdots F_m$ is the grammar parsing of some SLP for T. Then, any F_k longer than 1 has an implied corresponding internal node to the left in the partial parse tree. Since an internal node derives at least two leaves, it derives $F_{i_k} \cdots F_{j_k}$ corresponding to the interval $[i_k..j_k]$ of the factorization for some $i_k < j_k < k$. Furthermore, since all of these intervals are derived from internal nodes of a tree, they must respect the tree structure, i.e., any two of them must be disjoint or contained in one another.
($\Leftarrow$) Suppose we are given a factorization $T = F_1 \cdots F_m$ of T, as well as for each F_k, a corresponding interval $[i_k..j_k]$ of the factorization satisfying the conditions of the lemma. Since, for any pair of factors $F_x = F_{i_x} \cdots F_{j_x}$ and $F_y = F_{i_y} \cdots F_{j_y}$, the intervals $[i_x..j_x]$ and $[i_y..j_y]$ are disjoint or contained in one another, we can construct a tree with the internal nodes corresponding to the intervals and the leaves corresponding to the factors of the factorization, where a node is a descendant of another if and only if it is a sub-interval. Although such a tree can be multi-ary in general, we can add internal nodes and transform it into a full binary tree while preserving ancestor/descendant relations of nodes/leaves in the original tree (note that the resulting tree may not be determined uniquely, but its size will always be the same). We assign to each internal node a distinct non-terminal symbol. To each leaf corresponding to a factor F_k longer than 1, we assign the same non-terminal symbol that we assigned to the internal node corresponding to $F_{i_k} \cdots F_{j_k}$. Finally, we assign each leaf corresponding to a factor of length 1 a non-terminal symbol that derives the corresponding terminal symbol. The resulting tree is a partial parse tree for an SLP of size $m + \sigma - 1$ for T with $F_1 \cdots F_m$ as its grammar parsing. ◀

We define Boolean variables as follows to encode Lemma 2.

- $f_{i,\ell}$ for $i \in [1,n], \ell \in [1, n+1-i]$: $f_{i,\ell} = 1$ if and only if $T[i..i+\ell)$ is a factor of the grammar parsing.
- p_i for $i \in [1, n+1]$: For $i \neq n+1$, $p_i = 1$ if and only if i is a starting position of a factor of the grammar parsing. p_{n+1} is for technical reasons. We set $p_1 = p_{n+1} = 1$.
- $\mathit{ref}_{i' \leftarrow i,\ell}$ for $i, \ell, i' \in [1,n]$, s.t. $\ell \geq 2$, $i' \leq i - \ell$ and $T[i'..i'+\ell) = T[i..i+\ell)$: $\mathit{ref}_{i' \leftarrow i,\ell} = 1$ if and only if $T[i..i+\ell)$ is a factor of the grammar parsing, and the implied internal node of the partial parse tree corresponds to $T[i'..i'+\ell)$.
- $q_{i',\ell}$ for $i' \in [1, n-1], \ell \in [2, n+1-i']$ s.t. $T[i'..i'+\ell)$ has an occurrence in $T[i'+\ell..n]$: $q_{i',\ell} = 1$ if and only if $T[i'..i'+\ell)$ corresponds to an internal node of the partial parse tree that is referenced by at least one factor of the grammar parsing.

We next define constraints that the above variables must satisfy.

First, since each factor of the grammar parsing is disjoint and the concatenation of all factors must be equal to T, the truth values of $f_{i,\ell}$ must uniquely define the truth values for p_i and vice versa. This can be encoded as

$$\forall i \in [1,n], \ell \in [1, n+1-i] : f_{i,\ell} \iff p_i \wedge (\neg p_{i+1}) \cdots (\neg p_{i+\ell-1}) \wedge p_{i+\ell} \tag{1}$$

For all i and $\ell \geq 2$ such that $T[i..i+\ell)$ is the first occurrence of a substring $S = T[i..i+\ell)$ of T, $T[i..i+\ell)$ cannot be a factor of a grammar parsing. Thus, we require:

$$\forall i \in [1, n-1], \ell \in [2, n-i+1] \text{ s.t. } T[i..i+\ell) \text{ does not occur in } T[1..i) : \neg f_{i,\ell} \tag{2}$$

If $T[i..i+\ell)$ is not the first occurrence of S, $T[i..i+\ell)$ can be a factor. If $T[i..i+\ell)$ is a factor of the grammar parsing of length at least 2, then, there must exist at least one $i' \leq i - \ell$ such that $T[i'..i'+\ell) = S$ and $T[i'..i'+\ell)$ corresponds to an internal node of the partial parse tree. This can be encoded as

$$\forall i \in [1,n], \ell \in [2, n+1-i] \text{ s.t. } T[i..i+\ell) \text{ occurs in } T[1..i) :$$
$$f_{i,\ell} \implies \bigvee_{i' \in \{k \mid T[k..k+\ell) = T[i..i+\ell), k \in [1, i-\ell)\}} \mathit{ref}_{i' \leftarrow i,\ell}. \tag{3}$$

Furthermore, for any i, ℓ, a factor $T[i..i+\ell)$ references at most one position, i.e.,

$$\forall i \in [1,n], \ell \in [2, n+1-i] : \sum_{i' \in \{k \mid T[k..k+\ell) = T[i..i+\ell), k \in [1, i-\ell)\}} \mathit{ref}_{i' \leftarrow i,\ell} \leq 1. \tag{4}$$

On the other hand, $\mathit{ref}_{i' \leftarrow i,\ell} = 1$ implies that $T[i..i+\ell)$ is a factor of the grammar parsing. Therefore,

$$\forall i \in [1,n], \ell \in [2, n+1-i], i' \in \{k \mid T[k..k+\ell) = T[i..i+\ell), k \in [1, i-\ell)\} :$$
$$\mathit{ref}_{i' \leftarrow i,\ell} \implies f_{i,\ell} \tag{5}$$

By definition, it holds that

$$\forall i' \in [1, n-1], \ell \in [2, n+1-i'] \text{ s.t. } T[i'..i'+\ell) \text{ has an occurrence in } T[i'+\ell..n] :$$
$$q_{i',\ell} \iff \bigvee_{1 \leq i'+\ell \leq i \leq n} \mathit{ref}_{i' \leftarrow i,\ell} \tag{6}$$

Next, as shown in Lemma 2, we require that the implied internal node that is referenced by some factor must be an interval of size at least 2 of the factorization. We encode this as:

$$\forall i' \in [1, n-1], \ell \in [2, n+1-i'] \text{ s.t. } T[i'..i'+\ell) \text{ has an occurrence in } T[i'+\ell..n] : \\ q_{i',\ell} \implies \neg f_{i',\ell} \wedge p_{i'} \wedge p_{i'+\ell} \tag{7}$$

Also, for any two such implied internal nodes $T[i_1..i_1+\ell_1)$ and $T[i_2..i_2+\ell_2)$, they must either be disjoint, or one is a sub-interval of the other. In other words, it cannot be that a proper prefix interval of one is a proper suffix interval of the other, i.e.,

$$\forall i_1, i_2, \ell_1, \ell_2 \text{ s.t. } i_1 < i_2 < i_1+\ell_1 < i_2+\ell_2 \text{ s.t.} \\ T[i_k..i_k+\ell) \text{ has an occurrence in } T[i_k+\ell..n] \text{ for } k \in \{1,2\} : \\ \neg q_{i_1,\ell_1} \vee \neg q_{i_2,\ell_2} \tag{8}$$

In total, we have $O(n^3)$ Boolean variables dominated by $\mathit{ref}_{i' \leftarrow i,\ell}$. The size of each clause is at most $O(n)$. The total size of the resulting CNF is $O(n^4)$, dominated by Constraint (8) where there are $O(n^4)$ clauses of $O(1)$ size each.

Correctness of the Encoding

We now prove the correctness of our formulation. From Lemma 2, if we are given some SLP producing T, it is clear that the above Boolean variables corresponding to its partial parse tree, referencing structure, and grammar parsing will satisfy all of the constraints.

Next, suppose we are given T and a truth assignment satisfying the above constraints. Starting from the truth assignments of p_i and Constraint (1), we can obtain a factorization of T where we regard $T[i..i+\ell)$ as a factor if and only if $f_{i,\ell} = 1$. For any $i \in [1, n]$ and $\ell \in [2, n-i+1]$, Constraint (2) ensures that $T[i..i+\ell)$ having an occurrence in $T[1..i)$ is a necessary condition for $f_{i,\ell} = 1$. If $f_{i,\ell} = 1$, Constraint (3) implies that there is some $i' \in [1..i-\ell)$ such that $T[i'..i'+\ell) = T[i..i+\ell)$ and $\mathit{ref}_{i' \leftarrow i,\ell} = 1$. From Constraint (4), we know that there is exactly one such i'. On the other hand, Constraint (5) ensures that $\mathit{ref}_{i' \leftarrow i,\ell} = 0$ for all i' when $f_{i,\ell} = 0$. Thus, for each $f_{i,\ell} = 1$ with $\ell > 1$ there exists exactly one i' such that $\mathit{ref}_{i' \leftarrow i,\ell} = 1$, and all other $\mathit{ref}_{\cdot \leftarrow \cdot,\cdot}$ are 0. From Constraint (6), it holds that $q_{i',\ell} = 1$ if and only if there is at least one i, ℓ with $\mathit{ref}_{i' \leftarrow i,\ell} = 1$ and thus $f_{i,\ell} = 1$. If $q_{i',\ell} = 1$, from Constraint (7), we have $f_{i',\ell} = 0$, $p_{i'} = p_{i'+\ell} = 1$, implying that $T[i'..i'+\ell)$ is not a factor, but is a concatenation of two or more factors. Constraint (8) requires that all such $T[i'..i'+\ell)$ are either disjoint or that one is a sub-interval of the other.

Thus, from the above arguments, we can see that for the factorization defined by the p_i's, we can associate for each factor, a subinterval of the factorization that satisfies the conditions of Lemma 2, thus implying that the factorization is a grammar parsing of some SLP.

3.3 Smallest Bidirectional Macro Scheme to MAX-SAT

For our SLP encoding, we used the fact that only the leftmost occurrences of the non-terminals are internal nodes – we modeled every later occurrence as a leaf referring to this leftmost occurrence. We could therefore evade the problem of constructing reference cycles since all references point in the same direction. However, in a bidirectional scheme, the references can point in either direction, and the difficulty in defining the encoding is how to ensure that no cycles are introduced in the referencing.

Here, we present a solution that again works with a tree structure, but this time we have multiple trees – a forest that represents the references. In detail, we follow Dinklage et al. [9, Definition 6], who represented a bidirectional macro scheme by a reference forest,

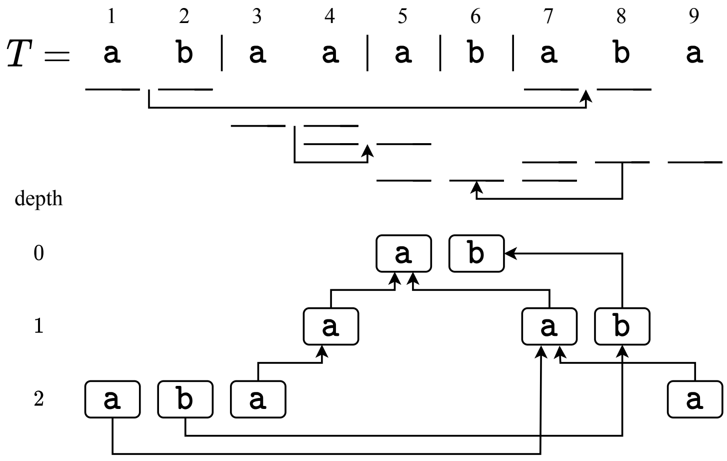

Figure 4 Reference forest of the BMS of Figure 1. The forest consists of two trees. The root of each tree corresponds to one of the two ground phrases of the BMS. For instance, to decode $T[1]$, we first need to decode $T[7]$ (the parent of $T[7]$), which has the tree root $T[5]$ as its parent. Hence, the number of ancestors of a node $T[i]$ is the number of references we need to traverse to obtain a ground phrase storing the character of $T[i]$.

where a text position i has text position j as its parent if the phrase covering $T[i]$ has a reference stating that $T[i]$ is copied from $T[j]$. Figure 4 visualizes such a forest. The roots of this reference forest are the positions of the ground phrases.

In order to find a BMS, we go in the inverse direction, and first encode a reference forest from which we subsequently derive a BMS. Since a forest has no cycles, we can use the edges of the forest to define a valid BMS, where each factor has length one (each factor is represented by a node in the reference forest). The final step is to glue together adjacent positions that have adjacent references into larger factors to obtain BMSs with fewer factors.

We start with the encoding for our reference forest. The nodes of the forest coincide with the text positions, and are therefore enumerated from 1 to n. Since a text position i can reference text position j only when $T[i] = T[j]$, it makes sense to restrict j to belong to the set $M_i := \{j \in [1, n] \mid T[i] = T[j], i \neq j\}$. In that case, we say that j is the parent of i. We make use of the following variables.

- $root_i$ for $i \in [1, n]$: $root_i = 1$ if and only if node i is the root of a tree. All roots are at depth 0.
- $dref_{d,i \to j}$ for $d \in [1, n-1], i \in [1, n], j \in M_i$: $dref_{d,i \to j} = 1$ if and only if node i at depth d has a parent node j at depth $d - 1$.

To obtain a valid reference forest, we define the following constraints. First, each node is a root node or has a parent.

$$\forall i \in [1, n] : root_i + \sum_{d \in [1,n], j \in M_i} dref_{d,i \to j} = 1 \tag{9}$$

According to Constraint (9), a node i at depth $d \geq 2$ must have exactly one parent j, and j must also have a parent node k (since $d \geq 2$). To enforce acyclicity, we additionally want that k is exactly two levels above of i.

$$\forall d \in [2,n], \forall i \in [1,n], \forall j \in M_i : \mathit{dref}_{d,i\to j} \implies \sum_{k\in M_j} \mathit{dref}_{d-1,j\to k} = 1 \tag{10}$$

Next, to translate our reference forest to a BMS, we additionally introduce the following Boolean variables.

- $\mathit{ref}_{i\to j}$ for $i \in [1,n], j \in M_i$: $\mathit{ref}_{i\to j} = 1$ if and only if position i references position j.
- p_i for $i \in [1,n]$: $p_i = 1$ if and only if position i is a beginning of a phrase. Note that $p_1 = 1$.

The connection between the variables of the reference forest and our BMS is as follows. For each position $i \in [1,n]$, i can reference at most one position $j \in M_i$, i.e.,

$$\forall i \in [1,n] : \sum_{j\in M_i} \mathit{ref}_{i\to j} \le 1 \tag{11}$$

A position i references j if, on any depth d of the reference forest, there is an edge from i to its parent j modeled by $\mathit{dref}_{d,i\to j}$.

$$\forall d \in [1,n], \forall i \in [1,n], \forall j \in M_i : \mathit{dref}_{d,i\to j} \implies \mathit{ref}_{i\to j} \tag{12}$$

Finally, the roots in our reference forest model the ground phrases of the BMS. The roots therefore cannot have a reference, but instead introduce a factor (of length one).

$$\forall i \in [1,n] : \mathit{root}_i \implies p_i.\ \text{Additionally, } \forall j \in M_i : \mathit{root}_i \implies \neg \mathit{ref}_{i\to j} \tag{13}$$

Remembering that the phrases are determined by the variables p_i's witnessing their starting positions, it is left to model the constraints for the truth assignment of the p_i's. For that, let us conceptually fix a text position i for which we assume that it references text position j. We consider two cases where $T[i-1]$ and $T[i]$ cannot be in the same phrase. The first case is when i or j are at the start of the text or $j-1 \notin M_{i-1}$:

$$\forall i \in [1,n], j \in M_i \text{ s.t. } i = 1 \text{ or } j = 1 \text{ or } T[i-1] \ne T[j-1] : \mathit{ref}_{i\to j} \implies p_i \tag{14}$$

The second case is when $j-1 \in M_{i-1}$ but the position $i-1$ does not reference position $j-1$ (it may reference a different position, or it could be a ground phrase):

$$\begin{aligned} &\forall i \in [2,n], \forall j \in M_i \text{ s.t. } j > 1 \text{ and } T[i-1] = T[j-1], \\ &\neg \mathit{ref}_{i-1\to j-1} \wedge \mathit{ref}_{i\to j} \implies p_i \end{aligned} \tag{15}$$

In total, we have $O(n^3)$ Boolean variables, dominated by $\mathit{dref}_{d,i\to j}$. The size of the largest clause is $O(n^2)$ due to Constraint (9). The total size of the resulting CNF is $O(n^4)$, dominated by Constraint (10) where there are $O(n^3)$ clauses of $O(n)$ size each.

Correctness of the Encoding

It is easy to see that any valid BMS satisfies the above constraints. We now show that any solution that satisfies the hard clauses yields a valid BMS. The truth assignments for all p_i define a factorization of T. We claim that each position is either a ground phrase, or is assigned exactly one reference consistent with the factorization forming a valid BMS, i.e., the references are acyclic, and, adjacent positions in the same non-ground phrase will refer to adjacent positions thus allowing the phrase to be encoded with the pair of references at both ends of the phrase.

Suppose $p_i = 1$. If $root_i = 1$, then Constraint (9) ensures that all $dref_{\cdot,i\to\cdot} = 0$ and Constraint (13) ensures that all $ref_{i\to\cdot} = 0$, i.e., i does not have a reference. Note that, $p_{i+1} = 0$ implies $ref_{i\to j}$ for some j (shown in the next paragraph), so $p_{i+1} = 1$ must hold. Thus, position i is properly factorized as a ground phrase. If $root_i = 0$, then Constraint (9) ensures that there exist unique d, j such that $dref_{d,i\to j} = 1$. Furthermore, Constraint (12) ensures that $ref_{i\to j} = 1$.

Next, consider the case for $p_i = 0$ (which implies $i > 2$). From Constraint (13) we have $root_i = 0$, and from Constraint (15) we have $\forall j > 1 \in M_i$ s.t. $T[i-1] = T[j-1]$, $ref_{i\to j} \implies ref_{i-1\to j-1}$. Since $root_i = 0$, Constraint (9) ensures that there exists unique d, j such that $dref_{d,i\to j} = 1$. Furthermore, Constraint (12) ensures that $ref_{i\to j} = 1$. Note that due to Constraint (14), neither $j = 1$ nor $T[i-1] \neq T[j-1]$ is possible, since this would imply $p_i = 1$, contradicting the assumption that $p_i = 0$. Thus $j > 1$ and $T[i-1] = T[j-1]$, and thus we have $ref_{i-1\to j-1} = 1$.

The uniqueness of the reference j for each position i of a non-ground phrase is ensured by Constraint (11). Thus, we have that references in adjacent positions in the same non-ground phrase point to adjacent positions. Since the acyclicity of the references are ensured by Constraint (10), we have a valid BMS.

4 Computational Experiments

We have implemented our encodings in PySAT (`https://pysathq.github.io/`) written in the Python language[3]. As datasets we used the files TRANS, NEWS, E.COLI, and PROGC from the Canterbury and Calgary corpus (`https://corpus.canterbury.ac.nz/`).

Here, we evaluated the sum of the literals in all hard clauses, i.e., the size of the encoded CNF, and the execution time of the SAT solver for computing a solution. In Figure 5, we evaluated our approach on different prefix lengths of the chosen datasets, starting from a prefix of 10 characters up to a prefix with 3000 characters. We aborted an execution after reaching one hour of computation or after exceeding 16 GB of RAM, and hence the lines for computing b and g prematurely end due to these limits on all datasets. Our experiments ran on an Ubuntu 20.04 machine with an AMD Ryzen Threadripper 3990X CPU.

As expected, the size of the encoded CNF correlates with the execution time in all instances. We can see that the encoding for γ needs the least number of literals, and is consequently not only the fastest, but also uses the least amount of memory, allowing us to compute γ for moderately large texts. This is followed by g, and lastly by b. Although the size of the CNF for b is smaller than for g in most cases, clauses formed by Constraint (9) for computing b can become quite large, making the computation cumbersome.

5 Application: Sensitivity of γ

Akagi et al. [1] introduced and studied the notion of *sensitivity* of a repetitiveness measure. Given a repetitiveness measure C (such as γ) for a string T, the sensitivity of C measures how much C can increase when a single character edit operation is performed on T. The authors studied an additive and a multiplicative sensitivity measure. The latter, denoted MS_{op}, is defined as:

$$MS_{op}(C, n) := \max_{T\in\Sigma^n, T'\in\Sigma^*} \left\{ \frac{C(T')}{C(T)} \;\middle|\; ed_{op}(T, T') = 1 \right\},$$

[3] As far as we are aware of, this implementation is single threaded.

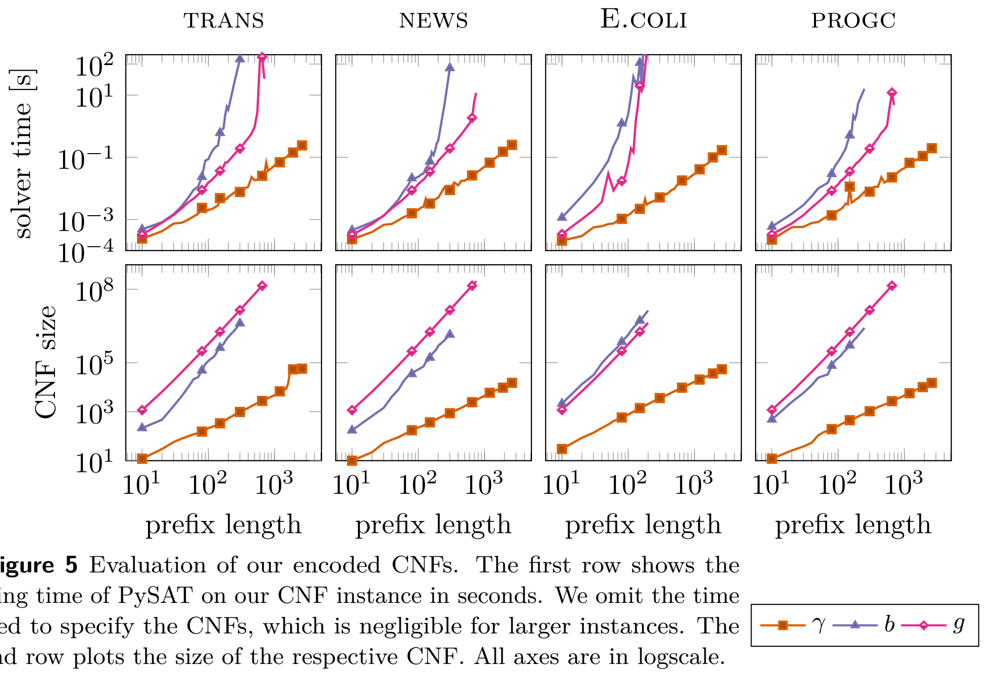

Figure 5 Evaluation of our encoded CNFs. The first row shows the running time of PySAT on our CNF instance in seconds. We omit the time needed to specify the CNFs, which is negligible for larger instances. The second row plots the size of the respective CNF. All axes are in logscale.

i.e., the maximum multiplicative increase over all strings with the same length n, where $ed_{op}(T, T') = 1$ means that T' can be built from T by inserting a character into T, or deleting/replacing a character of T. Parameterizing γ with the input string T, for $C(T) = \gamma(T)$, Akagi et al. showed $2 \leq MS_{op}(\gamma, n) \in O(\log n)$.

To improve the lower bound, we conducted exhaustive search for short binary strings when inserting a unique character. This search led us to the string family $\{T_k\}_{k\geq 2}$ with $T_k := \texttt{abbbaaab}^k$, with which we can improve the lower bound of 2 to 5/2. For that, let us consider $\gamma(T_k)$ and its size after an insertion of a new character $\texttt{c}$. First, we observe that $\gamma(T_k) = \gamma(\texttt{abbb\underline{a}aa\underline{b}}^k) = 2$. This is because a smallest string attractor is given by $\Gamma(T_2) = \{4, 7\}$ and $\Gamma(T_k) = \{5, 8\}$ for $k \geq 3$ (the characters at the positions in $\Gamma(T_k)$ are underlined). Now let T'_k denote T_k after inserting the character $\texttt{c}$ at text position 9. For $k \geq 5$, it holds that T'_k has a string attractor of size 5, i.e., $\gamma(T'_{k'}) = \gamma(\texttt{\underline{a}bb\underline{b}a\underline{a}ab\underline{c}\underline{b}}^{k'}) = 5$ for $k' \geq 4$. A minimal string attractor is given by $\Gamma(T'_{k'}) = \{1, 4, 6, 9, 10\}$. We cannot remove a position from $\Gamma(T'_{k'})$ since $\texttt{abb}, \texttt{ba}, \texttt{aab}, \texttt{c}$, and $\texttt{b}^{k'}$ are five substrings of $T'_{k'}$ having exactly one occurrence in $T'_{k'}$, and all of them are non-overlapping. Since a string attractor has to be in the cover set of all substrings, we need a string attractor with at least five text positions. Consequently, $MS_{op}(\gamma, n) \geq 2.5$ for any $n \geq 13$ with the insertion or replacement operation.

The availability of computer-aided search facilitated the discovery of strings having certain string attractors.

References

1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *CoRR*, abs/2107.08615, 2021. arXiv:2107.08615.

2 Hideo Bannai, Mitsuru Funakoshi, Tomohiro I, Dominik Köppl, Takuya Mieno, and Takaaki Nishimoto. A separation of γ and b via Thue-Morse words. In *Proc. SPIRE*, volume 12944, pages 167–178, 2021. doi:10.1007/978-3-030-86692-1_14.

3 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

4 Philip Bille, Travis Gagie, Inge Li Gørtz, and Nicola Prezza. A separation between RLSLPs and LZ77. *J. Discrete Algorithms*, 50:36–39, 2018. `doi:10.1016/j.jda.2018.09.002`.

5 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. `doi:10.1145/28869.28873`.

6 Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the complexity of the smallest grammar problem over fixed alphabets. *Theory Comput. Syst.*, 65(2):344–409, 2021. `doi:10.1007/s00224-020-10013-w`.

7 Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021. `doi:10.1145/3426473`.

8 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. `doi:10.1016/j.ipl.2007.10.006`.

9 Patrick Dinklage, Jonas Ellert, Johannes Fischer, Dominik Köppl, and Manuel Penschuck. Bidirectional text compression in external memory. In *Proc. ESA*, pages 41:1–41:16, 2019. `doi:10.4230/LIPIcs.ESA.2019.41`.

10 Patrick Dinklage, Johannes Fischer, Dominik Köppl, Marvin Löbel, and Kunihiko Sadakane. Compression with the tudocomp framework. In *Proc. SEA*, volume 75 of *LIPIcs*, pages 13:1–13:22, 2017.

11 Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. LZD factorization: Simple and practical online grammar compression with variable-to-fixed encoding. In *Proc. CPM*, volume 9133, pages 219–230, 2015. `doi:10.1007/978-3-319-19929-0_19`.

12 OEIS Foundation Inc. Maximum, over all binary strings w of length n, of the size of the smallest string attractor for w, entry A339391 in the on-line encyclopedia of integer sequences. Accessed: 2022-04-13. URL: `https://oeis.org/A339391`.

13 Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.

14 Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In Sandy Irani, editor, *Proc. FOCS*, pages 1002–1013. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00097`.

15 Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg. String attractors: Verification and optimization. In *Proc. ESA*, pages 52:1–52:13, 2018. `doi:10.4230/LIPIcs.ESA.2018.52`.

16 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proc. STOC*, pages 827–840. ACM, 2018. `doi:10.1145/3188745.3188814`.

17 Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the lz-end parsing. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2847–2866. SIAM, 2022. `doi:10.1137/1.9781611977073.111`.

18 Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability.* Addison-Wesley Professional, 1st edition, 2015.

19 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *Proc. LATIN*, pages 207–219, 2020.

20 Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theor. Comput. Sci.*, 483:115–133, 2013. `doi:10.1016/j.tcs.2012.02.006`.

21 Kanaru Kutsukake, Takuya Matsumoto, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. On repetitiveness measures of Thue-Morse words. In *Proc. SPIRE*, pages 213–220, 2020. `doi:10.1007/978-3-030-59212-7_15`.

22 N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Proc. DCC*, pages 296–305, 1999. `doi:10.1109/DCC.1999.755679`.

23 Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

24 Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. `doi:10.1007/978-3-030-11298-1`.

25 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nord. J. Comput.*, 12(1):40–66, 2005.

26 Sabrina Mantaci, Antonio Restivo, Giuseppe Romana, Giovanna Rosone, and Marinella Sciortino. A combinatorial view on string attractors. *Theor. Comput. Sci.*, 850:236–248, 2021. `doi:10.1016/j.tcs.2020.11.006`.

27 Sabrina Mantaci, Antonio Restivo, and Marinella Sciortino. Burrows-Wheeler transform and sturmian words. *Inf. Process. Lett.*, 86(5):241–246, 2003. `doi:10.1016/S0020-0190(02)00512-4`.

28 Takuya Mieno, Shunsuke Inenaga, and Takashi Horiyama. Repair grammars are the smallest grammars for fibonacci words. *CoRR*, abs/2202.08447, 2022. `arXiv:2202.08447`.

29 Kazuyuki Narisawa, Hideharu Hiratsuka, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Efficient computation of substring equivalence classes with suffix arrays. *Algorithmica*, 79(2):291–318, 2017. `doi:10.1007/s00453-016-0178-z`.

30 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. `doi:10.1145/3434399`.

31 Gonzalo Navarro. Indexing highly repetitive string collections, part II: compressed indexes. *ACM Comput. Surv.*, 54(2):26:1–26:32, 2021. `doi:10.1145/3432999`.

32 Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Transactions on Information Theory*, 67(2):1008–1026, 2020.

33 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. `doi:10.1016/j.tcs.2018.09.007`.

34 Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.*, 7:67–82, 1997. `doi:10.1613/jair.374`.

35 Takaaki Nishimoto and Yasuo Tabei. LZRR: LZ77 parsing with right reference. *Information and Computation*, page 104859, 2021.

36 Luís M. S. Russo, Ana Sofia D. Correia, Gonzalo Navarro, and Alexandre P. Francisco. Approximating optimal bidirectional macro schemes. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2020, Snowbird, UT, USA, March 24-27, 2020*, pages 153–162. IEEE, 2020. `doi:10.1109/DCC47342.2020.00023`.

37 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. `doi:10.1016/S0304-3975(02)00777-6`.

38 Hiroshi Sakamoto, Takuya Kida, and Shinichi Shimozono. A space-saving linear-time algorithm for grammar-based compression. In *Proc. SPIRE*, volume 3246, pages 218–229, 2004. `doi:10.1007/978-3-540-30213-1_33`.

39 Hiroshi Sakamoto, Shinichi Shimozono, Ayumi Shinohara, and Masayuki Takeda. On the minimization problem of text compression scheme by a reduced grammar transform. Technical Report 195, Department of Informatics, 2001. URL: `https://catalog.lib.kyushu-u.ac.jp/opac_download_md/3045/trcs195.pdf`.

40 Luke Schaeffer and Jeffrey Shallit. String attractors for automatic sequences. *CoRR*, abs/2012.06840, 2020. `arXiv:2012.06840`.

41 Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. `doi:10.1007/11564751_73`.

42 James A Storer and Thomas G Szymanski. Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951, 1982.

43 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.

44 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.