On the Hardness of Smallest RLSLPs and Collage Systems

Akiyoshi Kawamoto[†], Tomohiro I[†], Dominik Köppl[*,+], and Hideo Bannai[*]

[†]Kyushu Institute of Technology, Fukuoka, Japan

kawamoto.akiyoshi256@mail.kyutech.jp, tomohiro@ai.kyutech.ac.jp

[*]M&D Data Center, TMDU, Tokyo, Japan, {koeppl,hdbn}.dsc@tmd.ac.jp

[+]University of Yamanashi, Kōfu, Japan, dkppl@yamanashi.ac.jp

Abstract

We show that computing the smallest run-length straight line program (RLSLP) and the smallest collage system is NP-hard. For computing the smallest RLSLP, we give an encoding in MAX-SAT.

1 Introduction

Repetitiveness measures put the compressibility of input texts into numbers, which represent how good the text will compress for a given compression method. While popular repetitiveness measures such as the number of factors z of the Lempel–Ziv 77 factorization or the character runs r in the Burrows–Wheeler transform are known to be computable in linear time, some measures are found to be NP-hard; among those hard problems are the smallest size of a string attractor [5], the fewest number of LZ-End factors [1], the fewest number of factors b of a bidirectional macro scheme [11], and the smallest size of a grammar [4], or in particular: of a straight-line program (SLP) [9]. In this landscape of compression measures, we shed light on two other compression measures, for which NP-hardness is suspected, but yet formally unproven, namely: the smallest size of a run-length SLP (RLSLP) g_{rl} and of a collage system c [6], whose complexity is left as unknown in Table 1 of [7].[1] We show that computing these two compression measures are NP-hard. Also we give MAX-SAT formulations for computing the smallest RLSLP.

Regarding the smallest size c of a collage system, Kida et al. [6] proved that $c = O(\min\{g_{rl}, z \log z\})$. The second part got improved by Navarro et al. [8] to $c = O(z)$. They also presented the lower bound $c = \Omega(r \log n)$, and showed that $b = O(c)$ for so-called internal collage systems.

[1]Literature has already taken the NP-hardness of smallest RLSLPs for granted (e.g. [5, Sect. 1]); we stress that run-length compressed grammars are not necessarily RLSLPs.

2 Preliminaries

An integer interval $\{i, i+1, \ldots, j\}$ is denoted by $[i, j]$. Let Σ be a finite *alphabet*, a set of characters. An element of Σ^* is called a *string* or *text* over Σ. The length of a string w is denoted by $|w|$. The empty string ε is the string of length 0, that is, $|\varepsilon| = 0$. The concatenation of two strings x and y is denoted by $x \cdot y$ or simply xy. When a string w is represented by the concatenation of strings x, y and z (i.e., $w = xyz$), then x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively.

The i-th symbol of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i..j]$ for $1 \leq i \leq j \leq |w|$, i.e., $w[i..j] = w[i]w[i+1] \cdots w[j]$. Also, let $w[i..j)$ denote $w[i..j-1]$. For a string w and an integer $t \geq 1$, w^t represents the string obtained by concatenating t copies of w.

A *Straight-Line Program (SLP)* is a context-free grammar in Chomsky normal form whose language consists of a single string. Formally, an SLP S for a string $T \in \Sigma^*$ is a context-free grammar $(\{X_1, X_2, \ldots, X_n\}, \Sigma, R, X_n)$, where $\{X_1, X_2, \ldots, X_n\}$ is the set of non-terminals, Σ is the set of terminals (characters), R is the set of production rules that contains, for every variable X_i, exactly one production rule of the form $X_i \to c$ with $c \in \Sigma$, or, $X_i \to X_j X_k$ with $j, k < i$, and the starting non-terminal X_n deterministically derives T. We call $X_i \to c$ a *unary rule* and $X_i \to X_j X_k$ a *binary rule*. For a non-terminal X_i, let $exp(X_i)$ denote the string derived from X_i, i.e., $exp(X_i)$ is obtained by repeatedly applying production rules to X_i until we get the unique sequence of terminals.

We here study two extensions of SLPs: *Run-length SLPs* (RLSLPs) and collage systems. In addition to the aforementioned production rules, RLSLPs can contain *run-length rules* of the form $X_i \to X_j^t$ for $j < i$ and an integer $t \geq 3$, which derives the string $exp(X_j)^t$. Collage systems can have, additionally to RLSLPs, *truncation rules* of the form $X_i \to X_j^{[t]}$ and $X_i \to {}^{[t]}X_j$ for $j < i$ and an integer $t \geq 2$, which derives string $exp(X_j)[1..t]$ and $exp(X_j)[|exp(X_j)| - t + 1..|exp(X_j)|]$, respectively. Although in the original definition [6], the initial rule of a collage system has an arbitrarily long right-hand side, say of length s, we collapse it by introducing $s - 1$ binary rules for simplicity. By doing so, we can measure the *size* of SLPs, RLSLPs and collage systems by the number n of non-terminals (or equivalently production rules). In this paper we assume that SLPs, RLSLPs and collage systems do not contain useless non-terminals/terminals that are not used to obtain the uncompressed string.

For SLPs and RLSLPs, the expansion process of a non-terminal X_i forms an expansion tree, which consists of internal nodes labeled with non-terminals and leaves labeled with terminals. *The expansion tree* of an SLP or RLSLP is the expansion tree of the initial non-terminal. *The grammar tree* of an SLP is the tree obtained by traversing the expansion tree in a depth-first manner and pruning all the descendants if a visited node is labeled with a previously seen non-terminal. Since each non-terminal appears exactly once as the label of an internal node of the grammar tree, the number of internal nodes of the grammar tree is equal to the SLP size. Also, since each binary rule corresponds to exactly one internal

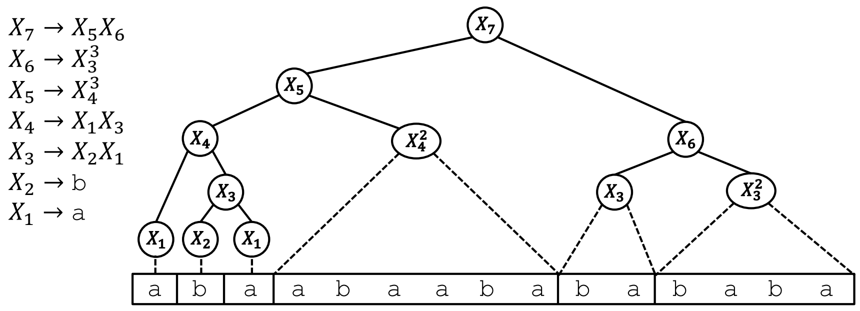

Figure 1: The grammar tree for an RLSLP of the text $T = (\texttt{aba})^3(\texttt{ba})^3$. The RLSLP parsing has 6 factors. The number of factors is equal to the number of non-terminals ($= 7$) minus the alphabet size ($= 2$) plus one. Standard SLPs cannot have rules like for X_5 or X_6.

node having two children in the grammar tree, the number of leaves of the grammar tree is equal to $n + |\Sigma| - 1$.

For RLSLPs we employ the extended grammar tree (also used in Theorem 5 of [8]) such that the node for a run-length rule $X_i \to X_j^t$ has the leftmost X_j for the left child and the rest X_j^{t-1} for the right child. While the right child representing X_j^{t-1} is always a leaf of the grammar tree, the left child may or may not be an internal node depending on whether it is the first node labeled with X_j or not in the depth-first traversal of the expansion tree. For an RLSLP S for a string T, *the RLSLP parsing* of S is the factorization $T = F_1F_2 \cdots F_m$ such that the i-th factor F_i is derived from the i-th leaf of the grammar tree of S, where m is the number of leaves of the grammar tree. Similar to grammar trees of SLPs, it holds that $n + |\Sigma| - 1$ is the number of leaves of the grammar tree, which is also the number of factors in the RLSLP parsing. Figure 1 shows an example of an RLSLP grammar tree and its RLSLP parsing.

The satisfiability (SAT) problem asks for an assignment of variables that satisfies a given Boolean formula, which is usually given in conjunctive normal form (CNF). An extension to SAT is MAX-SAT, where we are allowed to additionally pose an objective function under which we want to optimize the satisfying assignment.

3 Hardness of Smallest Collage System Problem

In this section, we prove that the smallest RLSLP problem (Smallest RLSLP) and the smallest collage system problem (Smallest Collage System) are NP-hard. To this end, we first show that the problem of computing the smallest SLP (Smallest SLP) is NP-hard. Despite that the hardness of Smallest SLP has already been proven by Sakamoto et al. [9], it is straight-forward to deduce hardness for Smallest RLSLP and Smallest Collage System from our hardness proof.

Our proof for the hardness of Smallest SLP is a slight modification of the reduction technique presented by Charikar et al. [4] for *the smallest grammar problem* (Smallest

Grammar) in which a grammar can have an arbitrary long right-hand side and the size of a grammar is measured by the sum of the lengths of the right-hand sides. For a graph (V, E), Charikar et al. [4] considered the string $X \cdot Y \cdot Z$ with

$$X = \prod_{v \in V} (\#v \diamond v\#\diamond)^2, Y = \prod_{v \in V} (\#v\#\diamond), \text{ and } Z = \prod_{(u,v) \in E} (\#u\#v\#\diamond), \tag{1}$$

where each occurrence of $\diamond$ represents a (new) distinct character, additionally to the characters in $\{\#\} \cup V$. Since the number of $\diamond$'s is $4|V|$ in X, $|V|$ in Y, and $|E|$ in Z, the number of characters used in $X \cdot Y \cdot Z$ is $1 + |V| + 4|V| + |V| + |E| = 6|V| + |E| + 1$. Charikar et al. [4] showed that $v \in V$ is in a vertex cover with minimum size if and only if the smallest grammar for $X \cdot Y \cdot Z$ has a rule deriving $\#v\#$.

In contrast to the smallest grammar for $X \cdot Y \cdot Z$, a smallest SLP of $X \cdot Y \cdot Z$ can have $\#v\#$ rules for all vertices to represent Y. This ruins the reduction such that we cannot use same string for proving the hardness of Smallest SLP. Instead, for tackling Smallest SLP, we focus on the concatenation $w := XZ$ of the strings in Eq. (1) without Y. Let Σ be the set of distinct characters in w.

Theorem 3.1. Smallest SLP is NP-hard.

Proof. Given a graph (V, E), we consider a smallest SLP S with the minimum size for $w = XZ$. If a substring s of w of length longer than 1 is not in $\{\#v, v\#, \#v\#\}$, s occurs exactly once in w, and hence, a non-terminal deriving s does not contribute to reducing the SLP size although some of such non-terminals are needed to transform a grammar into Chomsky normal form. Remark that S must contain non-terminals deriving $\#v$ and $v\#$ for every vertex v, since otherwise, $X = \prod_{v \in V}(\#v \diamond v\#\diamond)^2$ cannot be compressed optimally.

Let V_S be the set of vertices v such that S has a non-terminal deriving $\#v\#$, and w_S be the partially expanded string of S considering all non-terminals deriving $\#v$, $v\#$ or $\#v\#$ as terminal characters in addition to the original terminal characters. For each $(u, v) \in E$, a substring $\#u\#v\#\diamond$ in w is represented by three terminal characters in w_S if S has a non-terminal for $\#u\#$ or $\#v\#$ (meaning that $u \in V_S$ or $v \in V_S$, i.e., the edge (u, v) is covered by a node in V_S), and otherwise by four terminal characters because S should make the most of its non-terminals to compress w as small as possible. Hence $Z = \prod_{(u,v) \in E}(\#u\#v\#\diamond)$ is represented by $3x_S + 4(|E| - x_S)$ terminal characters in w_S, where x_S is the number of edges covered by V_S. Also $X = \prod_{v \in V}(\#v \diamond v\#\diamond)^2$ is represented by $8|V|$ terminal characters in w_S, resulting in $|w_S| = 8|V| + 3x_S + 4(|E| - x_S)$. It is clear that the number of unary rules in S should be the number $|\Sigma|$ of distinct characters in w.

The number of binary rules in S is evaluated by the sum of the cost to introduce non-terminals for $\#v$, $v\#$ and $\#v\#$ and the cost to wrap w_S up into a single starting non-terminal. The former cost is $2|V| + |V_S|$, and the latter is exactly $|w_S| - 1$ because every non-terminal over w_S is used exactly once. Hence the number of binary rules in S is $2|V| + |V_S| + 8|V| + 3x_S + 4(|E| - x_S) - 1 = 10|V| + 4|E| + |V_S| - x_S - 1$. To minimize $|V_S| - x_S$, S should choose as few vertices for V_S as possible while maximizing x_S. The graph G_S obtained by removing all edges covered by V_S consists of connected components of size at most two because if there is a vertex u that is connected with two distinct vertices v_1 and v_2 in G_S, then each substring of $\#u\#v_1\#$ and $\#u\#v_2\#$ in w is represented by three terminals in w_S, but we can make it two (and shorten $|w_S|$ by at least two) by introducing a non-terminal for $\#u\#$ spending an additional cost of one, which contradicts that S has the minimum size. Note that V_S might not cover E completely, i.e., there are $|E| - x_S$ connected components of size two in G_S. We can easily extend V_S by adding an arbitrary vertex for each of the $|E| - x_S$ connected components, and build an SLP S' based on the same compression strategy as S but now using the extended set $V_{S'}$ of vertices. Clearly $V_{S'}$ is a vertex cover and S' has the same size as S because $|V_{S'}| - |V_S| = |E| - x_S$ and hence $|V_{S'}| - x_{S'} = |V_{S'}| - |E| = |V_S| - x_S$.

$V_{S'}$ is a minimum vertex cover since otherwise there is an SLP S'' built on a vertex cover $V_{S''}$ with $|V_{S''}| < |V_{S'}|$, which implies that S'' is smaller than S' and S, a contradiction. Hence, given the smallest SLP of w has size $m = |\Sigma| + 10|V| + 4|E| + |V_{S'}| - |E| - 1$, we obtain the size of the minimum vertex cover for (V, E) by $|V_{S'}| = m - |\Sigma| - 10|V| - 3|E| + 1$. This concludes the reduction from VERTEX COVER to SMALLEST SLP. □

We can show that the smallest collage system problem is NP-hard using the same reduction strategy for the smallest SLP problem presented above.

Theorem 3.2. SMALLEST COLLAGE SYSTEM is NP-hard.

Proof. Given a graph (V, E), we consider a smallest collage system C with the minimum size for $w = XZ$. Since SLPs are contained in the class of collage systems, the size of C is at most $m = |\Sigma| + 10|V| + 4|E| + |V_{S'}| - |E| - 1$, the size of a smallest SLP for w, where $V_{S'}$ is a minimum vertex cover. Intuitively, introducing run-length and/or truncation rules does not help improve the compression performance for this specific string w and thus the size of C is equal to m. Since C must contain non-terminals deriving $\#v$ and $v\#$ for every vertex v by the same reason with the SLP case, the challenge is to choose a set V_C of vertices v for which a non-terminal for $\#v\#$ is introduced while minimizing the total cost. Once we have V_C, the sum of the cost to introduce non-terminals for $\#v$, $v\#$ and $\#v\#$ is $2|V| + |V_C|$, which is same as the SLP case and optimal even for collage systems. The optimal cost for compressing the remaining string (defined similarly as w_S) is also the same as the SLP case. Hence, the minimization of the total cost results in m being the minimum size of C, which means that the same reduction of the proof of Theorem 3.1 works. □

The proof of Theorem 3.2 without truncation rules leads to:

Theorem 3.3. SMALLEST RLSLP is NP-hard.

The proofs of Theorems 3.2 and 3.3 imply that there is a vertex cover for a graph (V, E) of size at most k if and only if there is an RLSLP and a collage system for w of size at most $|\Sigma| + 10|V| + 3|E| + k - 1$. Also, given an RLSLP or a collage system of size at most $O(|w|)$, we can decompress it to verify if it represents w in polynomial time. Hence, the decision versions of SMALLEST RLSLP and SMALLEST COLLAGE SYSTEM are NP-complete.

4 MAX-SAT Formulation of RLSLP

Our MAX-SAT formulation encodes an RLSLP as a special kind of text factorization, which is characterized by the following lemma. The idea is based on [2, Lemma 2], which we augment with so-called type-b factors to represent run-length encoded rules.

Lemma 4.1. A factorization $T = F_1 \cdots F_m$ for a text T is the RLSLP parsing of an RLSLP for T if and only if (i) for each factor F_k longer than 1, either (type-a) $F_k = F_{i_k} \cdots F_{j_k}$ for some $i_k < j_k < k$ or (type-b) $F_k = (F_{i_k} \cdots F_{k-1})^{t-1}$ for some $i_k < k$ and $t \geq 3$, and additionally, (ii) any two intervals in

$$\begin{aligned} I =& \{[i_k, j_k] \mid \ F_k = F_{i_k} \cdots F_{j_k} \text{ is a type-a factor}\} \cup \\ & \{[i_k, k-1] \mid \ F_k = (F_{i_k} \cdots F_{k-1})^{t-1} \text{ is a type-b factor with } i_k < k-1\} \cup \\ & \{[i_k, k] \mid \ F_k = (F_{i_k} \cdots F_{k-1})^{t-1} \text{ is a type-b factor}\} \end{aligned}$$

are either disjoint or one is a sub-interval of the other.

Proof. ($\Rightarrow$) Suppose $F_1 \cdots F_m$ is the RLSLP parsing of some RLSLP for T. If F_k is a factor corresponding to the right child of a run-length rule $X \to Y^t$ with $t \geq 3$, then there is $i_k < k$ such that $F_{i_k} \cdots F_{k-1} = \mathit{exp}(Y)$ represents the left child of the run-length rule and $F_{i_k} \cdots F_{k-1}$ and $F_{i_k} \cdots F_k$ correspond to the intervals $[i_k, k-1]$ and $[i_k, k]$, respectively. Otherwise, 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 the set I of intervals satisfying the conditions of the lemma. Since, any two intervals 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 type-a factor F_k, we assign the same non-terminal symbol that we assigned to the internal node corresponding to $F_{i_k} \cdots F_{j_k}$. To each leaf corresponding to a type-b factor $F_k = (F_{i_k} \cdots F_{k-1})^{t-1}$, we assign Y^{t-1}, where Y is the non-terminal assigned to the node for $F_{i_k} \cdots F_{k-1}$. 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 grammar tree for an RLSLP of size $m + \sigma - 1$ for T with $F_1 \cdots F_m$ as its RLSLP parsing. $\square$

We modify the MAX-SAT formulation in [2] for the smallest SLP problem to incorporate run-length rules. For a given text $T[1..n]$ of length n, we define Boolean variables as follows to encode Lemma 4.1.

- $f_{i,\ell}$ for $i \in [1,n]$ and $\ell \in [1, n+1-i]$: $f_{i,\ell} = 1$ iff $T[i..i+\ell)$ is a factor of the RLSLP parsing.
- p_i for $i \in [1, n+1]$: For $i \neq n+1$, $p_i = 1$ iff i is a starting position of a factor of the RLSLP parsing. p_{n+1} is for technical reasons. We set $p_1 = p_{n+1} = 1$.
- $\mathit{ref}_{i' \leftarrow i,\ell}$ for $i \in [1, n-1]$, $\ell \in [2, n-i+1]$ and $i' \in [1, i-\ell]$ s.t. $T[i'..i'+\ell) = T[i..i+\ell)$: $\mathit{ref}_{i' \leftarrow i,\ell} = 1$ iff $T[i..i+\ell)$ is a type-a factor, and the implied internal node of the RLSLP grammar tree corresponds to $T[i'..i'+\ell)$.
- $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell}$ for $i \in [1, n-1]$, $\ell \in [2, n-i+1]$ and $i' \in [i-\ell+1, i-1]$ s.t. $T[i'..i'+\ell) = T[i..i+\ell)$ and $i-i'$ divides ℓ: $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell} = 1$ iff there exists a $t \geq 3$ so that $T[i..i+\ell) = (T[i'..i))^{t-1}$ is a type-b factor.
- $q_{i',\ell}$ for $i' \in [1, n-1]$ and $\ell \in [2, n+1-i']$ s.t. $T[i'..i'+\ell)$ has an occurrence in $T[i'+\ell..n]$: $q_{i',\ell} = 1$ iff $T[i'..i'+\ell)$ corresponds to an internal node of the RLSLP grammar tree that is referenced by at least one type-a factor.
- $q'_{i',\ell}$ for some $i' \in [1, n-1]$ and $\ell \in [2, n+1-i']$ is true if at least one of the following conditions holds: (a) $q_{i',\ell}$ is true; (b) $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i_1,\ell_1}$ is true for some i_1 and ℓ_1 with $\ell = \ell_1 + i_1 - i'$; (c) $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i_2,\ell_2}$ is true with $\ell = i_2 - i'$ for some i_2 and ℓ_2. The semantics is that $q'_{i',\ell}$ is true iff $T[i'..i'+\ell)$ corresponds to a node of the RLSLP grammar tree that (a) is referenced by at least one type-a factor, (b) labeled by a run-length non-terminal, or (c) the left child of a run-length rule.

For any variable x with some missing subscripts specified by "$\circ$", let $[\![x]\!]$ denote the set of (possibly tuples of) feasible subscripts to fill the missing part(s). For example, $[\![\mathit{ref}_{\circ \leftarrow \circ,\circ}]\!] = \{(i', i, \ell) \mid i \in [1, n-1], \ell \in [2, n-i+1], i' \in [1, i-\ell], T[i'..i'+\ell) = T[i..i+\ell)\}$, and for some fixed i and ℓ, $[\![\mathit{ref}_{\circ \leftarrow i,\ell}]\!] = \{i' \mid (i', i, \ell) \in [\![\mathit{ref}_{\circ \leftarrow \circ,\circ}]\!]\}$. We also use "$\cdot$" to represent anonymous (arbitrary) subscripts, which are filtered out when used in $[\![x]\!]$. For example, $[\![\mathit{ref}_{\cdot \leftarrow \circ,\circ}]\!] = \{(i, \ell) \mid (i', i, \ell) \in [\![\mathit{ref}_{\circ \leftarrow \circ,\circ}]\!]\}$.

We next define constraints that the above variables must satisfy.

Since the factors and starting positions of factors must be consistent, we have:

$$\forall (i, \ell) \in [\![f_{\circ,\circ}]\!] : f_{i,\ell} \iff p_i \wedge (\neg p_{i+1}) \wedge \cdots \wedge (\neg p_{i+\ell-1}) \wedge p_{i+\ell}. \tag{2}$$

If $T[i..i+\ell)$ with $\ell > 1$ cannot be a candidate of type-a nor type-b factor, $f_{i,\ell}$ must be false, i.e.,

$$\forall (i, \ell) \in [\![f_{\circ,\circ}]\!] \setminus ([\![\mathit{ref}_{\cdot \leftarrow \circ,\circ}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\cdot \leftarrow \circ,\circ}]\!]) \text{ with } \ell > 1 : \neg f_{i,\ell}. \tag{3}$$

If $T[i..i+\ell)$ is a type-a factor, there exists $i' \in \mathit{ref}_{\circ \leftarrow i,\ell}$ s.t. $\mathit{ref}_{i' \leftarrow i,\ell}$ is true. Also, if $T[i..i+\ell)$ is a type-b factor, there exists $i' \in \mathit{ref}^{\mathrm{r}}_{\circ \leftarrow i,\ell}$ s.t. $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell}$ is true. Conversely, if one of the variables of the form $\mathit{ref}_{i' \leftarrow i,\ell}$ or $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell}$ is true, $T[i..i+\ell)$ must be a factor. This can be encoded as

$$\forall (i,\ell) \in [\![\mathit{ref}_{\cdot \leftarrow \circ,\circ}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\cdot \leftarrow \circ,\circ}]\!] :$$
$$f_{i,\ell} \iff \left(\bigvee_{i' \in [\![\mathit{ref}_{\circ \leftarrow i,\ell}]\!]} \mathit{ref}_{i' \leftarrow i,\ell} \right) \vee \left(\bigvee_{i' \in [\![\mathit{ref}^{\mathrm{r}}_{\circ \leftarrow i,\ell}]\!]} \mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell} \right). \tag{4}$$

For a fixed substring $T[i..i+\ell)$, at most one variable of the form $\mathit{ref}_{i' \leftarrow i,\ell}$ or $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell}$ is allowed to indicate that $T[i..i+\ell)$ is a factor. Thus, we require

$$\forall (i,\ell) \in [\![\mathit{ref}_{\cdot \leftarrow \circ,\circ}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\cdot \leftarrow \circ,\circ}]\!] : \sum_{i' \in [\![\mathit{ref}_{\circ \leftarrow i,\ell}]\!]} \mathit{ref}_{i' \leftarrow i,\ell} + \sum_{i' \in [\![\mathit{ref}^{\mathrm{r}}_{\circ \leftarrow i,\ell}]\!]} \mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell} \leq 1. \tag{5}$$

Constraint 5 for a fixed (i,ℓ) can be encoded in size of $\left| [\![\mathit{ref}_{\circ \leftarrow i,\ell}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\circ \leftarrow i,\ell}]\!] \right| = O(n)$ using an efficient encoding for this kind of "at-most" constraints [10].

If $q_{i',\ell}$ is true, then there is a factor that references $T[i'..i'+\ell)$ and vice versa.

$$\forall (i',\ell) \in [\![q_{\circ,\circ}]\!] : q_{i',\ell} \iff \bigvee_{i \in [\![\mathit{ref}_{i' \leftarrow \circ,\ell}]\!]} \mathit{ref}_{i' \leftarrow i,\ell}. \tag{6}$$

If $q_{i',\ell}$ is true, $T[i'..i'+\ell)$ corresponds to an internal node of the RLSLP grammar tree, and hence, $T[i'..i'+\ell)$ must consist of at least two factors of the RLSLP parsing:

$$\forall (i',\ell) \in [\![q_{\circ,\circ}]\!] : q_{i',\ell} \implies \neg f_{i',\ell} \wedge p_{i'} \wedge p_{i'+\ell}. \tag{7}$$

If $\mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell}$ is true, $T[i'..i+\ell)$ is derived from a run-length non-terminal, and hence, i' must be a starting position of a factor.

$$\forall (i',i,\ell) \in [\![\mathit{ref}^{\mathrm{r}}_{\circ \leftarrow \circ,\circ}]\!] : \mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell} \implies p_{i'}. \tag{8}$$

We let $q'_{i',\ell}$ summarize the information on whether $T[i'..i'+\ell)$ corresponds to a node implied by $\mathit{ref}_{\cdot \leftarrow \cdot,\cdot}$ and $\mathit{ref}^{\mathrm{r}}_{\cdot \leftarrow \cdot,\cdot}$.

$$\forall (i',\ell) \in [\![q'_{\circ,\circ}]\!] : q'_{i',\ell} \iff$$
$$q_{i',\ell} \vee \left(\bigvee_{\substack{(i,\ell') \in [\![\mathit{ref}^{\mathrm{r}}_{i' \leftarrow \circ,\circ}]\!] \\ \text{with } \ell = \ell' + i - i'}} \mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell'} \right) \vee \left(\bigvee_{\substack{(i,\ell') \in [\![\mathit{ref}^{\mathrm{r}}_{i' \leftarrow \circ,\circ}]\!] \\ \text{with } \ell = i - i'}} \mathit{ref}^{\mathrm{r}}_{i' \leftarrow i,\ell'} \right). \tag{9}$$

Each variable $q'_{i',\ell}$ set to true encodes an interval $[i'..i'+\ell)$ corresponding to the expansion of an implied node of the grammar tree. For them, we want that they obey a nested structure (such that we can form a grammar tree from these nodes). In other words, for any two substrings $T[i_1..i_1+\ell_1)$ and $T[i_2..i_2+\ell_2)$ with $i_1 < i_2 < i_1+\ell_1 < i_2+\ell_2$, at most one of $T[i_1..i_1+\ell_1)$ and $T[i_2..i_2+\ell_2)$ can correspond to a node of the RLSLP grammar tree. Thus, we require that

$$\forall (i_1,\ell_1),(i_2,\ell_2) \in [\![q'_{\circ,\circ}]\!] \text{ s.t. } i_1 < i_2 < i_1+\ell_1 < i_2+\ell_2 : \neg q'_{i_1,\ell_1} \vee \neg q'_{i_2,\ell_2}. \qquad (10)$$

In total, we have $O(n^3)$ Boolean variables dominated by $\mathit{ref}_{\cdot\leftarrow\cdot,\cdot}$ or $\mathit{ref}^{\mathrm{r}}_{\cdot\leftarrow\cdot,\cdot}$. The size of each clause is at most $O(n)$. The total size of the resulting CNF is $O(n^4)$, dominated by Constraint 10 where there are $O(n^4)$ clauses of $O(1)$ size each. Our complexities match those observed in [2] for encoding an SLP; compared to their encoding, we have additionally introduced the variables $\mathit{ref}^{\mathrm{r}}_{i'\leftarrow i,\ell}$ and $q'_{i',\ell}$ for encoding the type-b factors.

Correctness of the Encoding We now prove the correctness of our formulation. On the one hand, given an RLSLP, it is clear that a truth assignment to Boolean variables based on the definition will satisfy all constraints.

On the other hand, suppose we are given T and a truth assignment satisfying the above constraints. Starting from the truth assignments of p_i and Constraint 2, we obtain a factorization of T where we regard $T[i..i+\ell)$ as a factor if and only if $f_{i,\ell}=1$. By Constraint 3, $T[i..i+\ell)$ with $\ell > 1$ cannot be a factor if $[\![\mathit{ref}_{\circ\leftarrow i,\ell}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\circ\leftarrow i,\ell}]\!] = \emptyset$. For any factor $T[i..i+\ell)$ with $\ell > 1$, Constraints 3, 4 and 5 respectively ensure that $[\![\mathit{ref}_{\circ\leftarrow i,\ell}]\!] \cup [\![\mathit{ref}^{\mathrm{r}}_{\circ\leftarrow i,\ell}]\!]$ is not empty, at least one variable of the form $\mathit{ref}_{\cdot\leftarrow i,\ell}$ or $\mathit{ref}^{\mathrm{r}}_{\cdot\leftarrow i,\ell}$ is true, and among them exactly one variable is true. Constraint 4 also ensures that $T[i..i+\ell)$ is a factor if there is a true variable of the form $\mathit{ref}_{\cdot\leftarrow i,\ell}$ or $\mathit{ref}^{\mathrm{r}}_{\cdot\leftarrow i,\ell}$. Thus, for each $f_{i,\ell}=1$ with $\ell > 1$ there exists exactly one true variable of the form $\mathit{ref}_{\cdot\leftarrow i,\ell}$ or $\mathit{ref}^{\mathrm{r}}_{\cdot\leftarrow i,\ell}$ and all other variables $\mathit{ref}_{\cdot\leftarrow\cdot,\cdot}$ or $\mathit{ref}^{\mathrm{r}}_{\cdot\leftarrow\cdot,\cdot}$ are false. From Constraint 6, it holds that $q_{i',\ell}=1$ if and only if there is at least one i,ℓ with $\mathit{ref}_{i'\leftarrow i,\ell}=1$, implying that $T[i..i+\ell)$ is a type-a factor that references $T[i'..i'+\ell)$. 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. If $\mathit{ref}^{\mathrm{r}}_{i'\leftarrow i,\ell}=1$, Constraint 8 ensures that $p_{i'}$ is a starting position of a factor. Since $\mathit{ref}^{\mathrm{r}}_{i'\leftarrow i,\ell}=1$ also implies that $p_i=1$, $T[i'..i)$ is represented by a concatenation of one or more factors, which can be formed as the left child of a run-length rule $X \to Y^t$ with the right child $T[i..i+\ell) = (T[i'..i))^{t-1}$. The information on the interval I of Lemma 4.1 is summarized to $q'_{\cdot,\cdot}$ by Constraint 9. Finally, Constraint 10 requires that all intervals in I are either disjoint or that one is a sub-interval of the other.

To sum up, the given variable assignment associates each factor (defined by p_i's) with a necessary reference position and a subinterval that satisfy the conditions of Lemma 4.1. This association implies that the factorization is an RLSLP parsing.

To transform this SAT formulation for finding a feasible RLSLP into a MAX-SAT formulation for finding a *smallest* RLSLP, we add the objective to minimize the number of set p_i's, i.e., $\min |\{i \in [2,n] \mid p_i = 1\}|$.

Experiments We have implemented our encoding called `RLSLP` in the framework `https://github.com/kg86/satcomp` with commit hash `d86b09f`. This framework uses pySAT as a MAX-SAT solver, and provides us with an implementation for computing the smallest

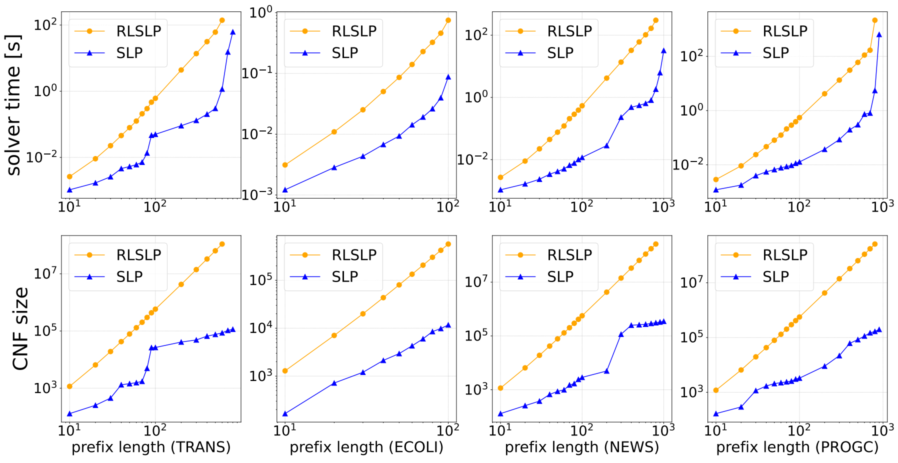

Figure 2: Comparisons on running times and CNF sizes of SLP and RLSLP.

SLP, which we call SLP. Figure 2 shows comparisons between RLSLP and SLP on the running times and the total CNF sizes for prefixes of the datasets TRANS, NEWS, and PROGC from the Calgary corpus, and ECOLI from the large Canterbury corpus. While ECOLI has an alphabet size of 4, the other texts have alphabets sizes in [92, 98]. Compared to SLP, RLSLP took more time and had bigger CNF sizes. That is because we need additional variables and constraints to handle run-length rules expressed by the type-b factors. The experiments have revealed that the maximum difference between RLSLP and SLP sizes is 3 for the tested data.

Open Problems It is left open to find an efficient encoding for collage systems, which are, due to the additional truncation rules, more technical than RLSLPs. The string $w = X \cdot Z$ we used in our hardness proofs in Sect. 3 has non-constant alphabet since each $\diamond$ introduced a new character. We wonder if we can translate recent hardness results for SLPs on constant-sized alphabets [3] to RLSLPs or collage systems.

Acknowledgements This research was supported by JSPS KAKENHI with grant numbers `JP22K11907`, `JP21K17701`, `JP23H04378` and `JP20H04141`.

References

[1] H. Bannai, M. Funakoshi, K. Kurita, Y. Nakashima, K. Seto, and T. Uno. Optimal LZ-End parsing is hard. In *Proc. CPM*, volume 259 of *LIPIcs*, pages 3:1–3:11, 2023.

[2] H. Bannai, K. Goto, M. Ishihata, S. Kanda, D. Köppl, and T. Nishimoto. Computing NP-hard repetitiveness measures via MAX-SAT. In *Proc. ESA*, volume 244 of *LIPIcs*, pages 12:1–12:16, 2022.

[3] K. Casel, H. Fernau, S. Gaspers, B. Gras, and M. L. Schmid. On the complexity of the smallest grammar problem over fixed alphabets. *Theory Comput. Syst.*, 65(2):344–409, 2021.

[4] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Information Theory*, 51(7):2554–2576, 2005.

[5] D. Kempa and N. Prezza. At the roots of dictionary compression: string attractors. In *Proc. STOC*, pages 827–840, 2018.

[6] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theor. Comput. Sci.*, 298(1):253–272, 2003.

[7] G. Navarro. Indexing highly repetitive string collections, part II: compressed indexes. *ACM Comput. Surv.*, 54(2):26:1–26:32, 2021.

[8] G. Navarro, C. Ochoa, and N. Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inf. Theory*, 67(2):1008–1026, 2021.

[9] H. Sakamoto, S. Shimozono, A. Shinohara, and M. Takeda. On the minimization problem of text compression scheme by a reduced grammar transform. Technical Report 195, Department of Informatics, Kyushu University, 2001.

[10] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In P. van Beek, editor, *Proc. CP*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[11] J. A. Storer and T. G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982.