九州大学学術情報リポジトリ
Kyushu University Institutional Repository

On the Minimization Problem of Text Compression Scheme by a Reduced Grammar Transform

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Shimozono, Shinichi
Department of Informatics, Kyushu University

Shinohara, Ayumi
Department of Informatics, Kyushu University

Takeda, Masayuki
Department of Informatics, Kyushu University

https://hdl.handle.net/2324/3045

出版情報：DOI Technical Report. 195, 2001-08. Department of Informatics, Kyushu University
バージョン：
権利関係：

KYUSHU UNIVERSITY

On the Minimization Problem of Text Compression Scheme by a Reduced Grammar Transform

(submitting to 13th annual ACM-SIAM symposium on discrete algorithms, SODA2002)

Hiroshi Sakamoto[†] Shinichi Shimozono[‡] Ayumi Shinohara[†] Masayuki Takeda[†]

Abstract

The complexity of an optimization problem for text compression by grammar transforms is studied. Given a string, the goal of this problem is to find a context-free grammar in Chomsky normal form, called a straight-line program, which strictly derives the string. The measure of this minimization problem is the size of a grammar, i.e., the number of the production rules of the grammar. The first result obtained in this paper is the NP-hardness of this problem. The second result is an approximation algorithm that achieves the worst-case approximation ratio $3n/\log^2 n$ for binary strings. The third result is a 6-approximation algorithm for restricted binary strings such that the strings contains no overlapping factor, called overlap-free strings.

1 Introduction

Text compression is a task of reducing the amount of space needed to store text files on computers or of reducing the amount of time taken to transmit information over a channel of given bandwidth. Users want to reduce the size of their data as small as possible, but want to minimize the time needed for this task. The main criteria for choosing compression methods are therefore the *compression ratio* and the *compression time*. Many studies have been undertaken to develop a new compression method for improving the compression ratio and/or the compression time. The Lempel-Ziv algorithms [16, 17], often referred to as LZ77 and LZ78, and their variants (e.g., [15, 13]) are most widely-used universal lossless compression algorithms, together with the arithmetic coding algorithms (see, e.g., [11]).

Finding the minimum representation of a given text string is regarded as a combinatorial optimization problem. One interesting topic is to analyze the time complexity of this optimization problem under some reasonable encoding scheme. Storer and Szymanski [14] presented several abstract compression schemes, and discussed the intractabilities of the minimization problems under the schemes. Especially, one of the schemes is a generalization of the LZ77 compression, and the corresponding minimization problem was shown to be NP-complete. Also, De Agostino and Storer [2] generalized the LZ78 scheme and then proved that the computation of the optimal parsing is NP-complete.

Recently, there have emerged new compression algorithms, such as Sequitur [9, 8] and Re-Pair [6], which often outperform the Lempel-Ziv family in the compression ratio comparison. Although they require much more compression time than the Lempel-Ziv family, it becomes not so crucial due to the recent progress in computer technology. These compression methods can be generalized into a new compression scheme called *grammar transform* (see [5]). In this scheme, a text string w is transformed into a context-free grammar G that generates the language $\{w\}$, and then G is encoded in some appropriate manner. A context-free grammar that generates a single string is said to be *admissible*. Yang and Kieffer [3] presented a grammar transform algorithm, and showed that the algorithm is asymptotically optimal (i.e. universal) for a broad class of sources, including stationary, ergodic sources. However, the computational complexity of the problem to find the smallest

[†]Dept. of Informatics, Kyushu University, Fukuoka 812-8581, Japan. E-mail: {hiroshi, ayumi, takeda}@i.kyushu-u.ac.jp

[‡]Dept. of Artificial Intelligence, Kyushu Institute of Technology, Iizuka 820-8502, Japan. E-mail: sin@ai.kyutech.ac.jp

not been discussed yet.

In this paper, we prove the NP-hardness of this minimization problem to find a smallest grammar with restriction to the class of admissible grammars in Chomsky normal form, called *straight-line programs*. Despite the restriction, the class of straight-line programs is rich enough in the context of text compression. For example, the Re-Pair compression is a grammar transform in which the grammars are restricted to straight-line programs. Moreover, the LZ78 compression can be regarded as a kind of grammar transform in which all the production rules but the start rule are restricted to the form $\sigma \to Xa$, where σ is a nonterminal symbol, a is a terminal symbol, and X is either a terminal symbol or a nonterminal symbol.

Our proof employs a reduction from MONOTONE 3-SAT to the decision version of the problem, and is considerably complicated. This settles partially, but as the first hardness result, the open problem in text compression by a reduced grammar transform. The string reduced from a monotone 3-CNF formula is constructed over a finite alphabet, whose size depends on the size of the given formula. However, it exhibits a potential intractability of the problem, and may suggest difficulties in developing efficient algorithms for the problem over a large alphabet.

We then restrict ourselves to the case of binary alphabet, and present two polynomial-time approximation algorithms. One is a general algorithm that finds an SLP for an arbitrary binary string, and achieves the worst-case approximation ratio $3n/\log^2 n$. Interestingly, this algorithm stands on a simple idea that makes pairs of symbols in strings from left to right, and is considered as an extension of the algorithms for obtaining the minimum ordered-binary decision diagram from a complete truth table [12].

The other is an algorithm that produces an SLP whose size is at most 6 times the optimum, for a class of binary strings that contain no overlapping substrings. This algorithm computes an SLP by involving the former algorithm as subroutine, with finding the longest substring of the input whose binary symbols are alternating.

The rest of this paper is organized as follows. In Section 2 we introduce some notation and definitions. In Section 3 we prove the NP-hardness of the problem by showing a reduction from MONOTONE 3-SAT. Then, we present two approximation algorithms for the case of binary strings in Section 4. Section 5 contains final remarks and states open problems.

2 Preliminaries

A set Σ denotes a finite alphabet through this paper. The set of all finite strings over Σ is denoted by Σ^*, and Σ^+ denotes $\Sigma^* \setminus \{\varepsilon\}$ where ε is the empty string. The length of a string $w \in \Sigma^*$, i.e. the number of symbols consisting of w, is denoted by $|w|$, and the ith symbol of w with $1 \le i \le |w|$ is denoted by $w[i]$.

A string $x \in \Sigma^*$ is a *substring of* $w \in \Sigma^*$ if there exist $y, z \in \Sigma^*$ such that $w = yxz$. In this case, y is a *prefix of* w, z is a *suffix of* w, and if $|x| < |w|$ then x is a *proper substring of* w. The notion $w[i, j]$ for $1 \le i \le j \le |w|$ refers to the substring $w[i] \cdots w[j]$. An *occurrence of* $w[i, j]$ *in* w refers to a position $1 \le k \le |w|$ such that $w[i, j] = w[k, k + (j - i)]$. By $\sharp(x, w)$, we denote the number of occurrences of x in w.

A context-free grammar $G = (\Sigma, N, P, S)$ is said to be *admissible* if it generates a singleton set of a nonempty string s, that is, $L(G) = \{s\}$. The production rules of any admissible grammar can be written as

$$\begin{aligned} \sigma_1 &\to \alpha_1; \\ \sigma_2 &\to \alpha_2; \\ &\vdots \\ \sigma_m &\to \alpha_m \end{aligned}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_m$ are the ordered nonterminal symbols in N, $\sigma_m = S$ is the start symbol, and α_i is a nonempty string over the alphabet $\Sigma \cup \{\sigma_1, \ldots, \sigma_{i-1}\}$ for $i = 1, 2, \ldots, m$. We assume G contains no useless symbols. The *size* of an admissible grammar G, denoted by $\|G\|$, is the total length of the right hand sides of production rules. That is, $\|G\| = |\alpha_1| + |\alpha_2| + \cdots + |\alpha_m|$.

In this paper, we consider the minimization problem for the admissible grammars in Chomsky normal form, called *straight-line programs* (*SLPs*). For simplicity of discussion, we replace all occurrences of σ within the right hand sides of the production rules with a for every production rule of the form $\sigma \to a$,

nal symbol. The grammar size is then exactly twice the cardinality of N. We note that this replacement is possible unless the string to be expressed by a grammar is of length one. Thus, we deal with the following minimization problem.

Definition 1. MIN STRAIGHT-LINE PROGRAM
Given a string $s \in \Sigma^+$, find the smallest straight-line program over Σ for s, i.e., a straight-line program that generates s and the cardinality of its set N of nonterminal symbols is minimum.

The decision problem with respect to MIN STRAIGHT-LINE PROGRAM is formulated by a positive integer threshold k in an instance and the question asking whether an SLP smaller than k exists.

Here we define some notions for minimization problems and their approximation algorithms (see e.g. [1, 10] for details.) An algorithm A for a minimization problem Π *approximates* Π *within ratio* α, or *achieves a worst-case approximation ratio* α if for any instance of Π the algorithm A produces its solution whose measure (cost) is no more than α times the measure of an optimum solution. In this case, we also say that A is an α-*approximation algorithm for* Π.

The straight-line program G is also represented by the term *yield* of a tree, i.e. the concatenation of all labels of the leaves of the tree; For a string w, G is a tree t over $N \cup \Sigma$ that yields w and each internal node is labeled by a nonterminal. The set P of rules represented by t is the set of all production rules $\sigma_i \to XY$ such that t contains a node labeled by $\sigma_i \in N$ whose children are labeled by $X, Y \in N \cup \Sigma$. The tree of w with respect to an SLP P is said to be a *derivation tree of* P *for* w, and we say P *derives* w.

From a view point of binary derivation trees, the following observation is rather immediate.

Lemma 1. The upper bound and lower bound of the sizes of minimum straight-line programs are $n-1$ and $\log n$, respectively, where n is the length of the input string.

Proof. Let w be an input string and $|w| = n$. Any derivation tree for w contains exactly $n-1$ internal nodes. Thus, the maximum number of rules derivation tree. Let P be a set of rules and $|P| = k$. If any rule is of the form $\sigma \to XX$ for some $\sigma \in N$ and $X \in N \cup \Sigma$, then the derivation tree is complete, i.e., all paths from the root to leaves are length k. If $|P| = k$ and the derivation tree is not complete, then there exists a path p which is longer than k. Since any path does not contain a nonterminal twice, p contains at least $k+1$ nonterminals. Thus, the minimum number of rules is bounded by $k = \log n$. □

Thus, a trivial approximation ratio achievable by a polynomial-time algorithm for MIN STRAIGHT-LINE PROGRAM is $(n-1)/\log n$.

3 MIN SLP is computationally intractable

In this section we show the following theorem:

Theorem 1. MIN STRAIGHT-LINE PROGRAM is NP-hard.

We prove this by a reduction from MONOTONE 3-SAT to a decision problem version of MIN STRAIGHT-LINE PROGRAM. We firstly define MONOTONE 3-SAT, then describe a translation algorithm from a monotone 3CNF to a pair of a string and a nonnegative integer, that is, an instance of decision version of MIN SLP. Then we show a series of lemmas and prove that the translation is a reduction.

Let $X = \{x_1, \ldots, x_n\}$ be a set of n Boolean variables. A monotone 3-CNF formula $F = (X, C)$ is defined by a set $C = \{c_1, \ldots, c_M\}$ of 3-literal monotone conjunctive clauses over X such that each clause c_i is either a positive clause $(x \vee y \vee z)$ or a negative clause $(\neg x \vee \neg y \vee \neg z)$. Without loss of generality, we assume that clauses in C are (i) ordered collections of 3-literals with respect to their variable indices, and (ii) indexed so that all the first $m \leq M$ clauses $c_1, \ldots, c_m$ are positive clauses and the remained $c_{m+1}, \ldots, c_M$ are negative clauses.

The translation algorithm constructs a string $s = s_C s_X s_\alpha s_\beta$ consisting of the four parts and formed from symbols in Σ. The alphabet Σ is finite but whose size depends on F; we define its symbols along with the description of the algorithm. Let $\alpha = (\mathtt{ab})^4$

$\mathtt{a}_i\mathtt{b}$. We define for all positive clauses and for all negative clauses the string $\pi^+ = \Pi_{1\le\ell\le m}(\alpha_{4\ell-2}\,\alpha_{4\ell-1}\,\mathtt{z})$ and $\pi^- = \Pi_{m+1\le\ell\le M}(\alpha_{4\ell-3}\,\alpha_{4\ell-2}\,\mathtt{z})$, where $\mathtt{z}$ is a new symbol for each ℓ. Then we define the string $s_C = \pi^+\,\pi^-$.

The string s_X is constructed as follows. Let $\pi = (\alpha\beta)^{12M-1} = (\beta\alpha)^{12M-1}$. We employ this as the 'template,' and with a variable index $1 \le i \le n$ define $\pi(i,1) = (\alpha\beta)^{4M-1}$, $\pi(i,2) = \pi(i,3) = (\alpha\beta)^{4M}$, and the gadget $\pi_i = \pi(i,1)\,\pi(i,2)\,\pi(i,3)$. We consider that $\pi(i,1)$ is formed from $4M-1$ segments of $\alpha\beta$, and $\pi(i,2), \pi(i,3)$ are formed from M segments of $\Delta = (\alpha\beta)^4$. We denote by $\Delta(i,2)_\ell$ the ℓth Δ of $\pi(i,2)$ and by $\Delta(i,3)_\ell$ the ℓth Δ of $\pi(i,3)$. According to the occurrences of the literals x_i and $\neg x_i$ in clauses, we transform each π_i.

For each $1 \le i \le n$, we transform $\pi(i,1)$ as follows: (i) If c_ℓ contains x_i or $\neg x_i$, replace $(4\ell-1)$th $\alpha\beta$ by $\alpha_{4\ell-1}\beta_{4\ell-1}$, $(4\ell-2)$th $\alpha\beta$ by $\alpha_{4\ell-2}\beta_{4\ell-2}$, and $(4\ell-3)$th $\alpha\beta$ by $\alpha_{4\ell-3}\beta_{4\ell-3}$. (ii) for each remained $\alpha\beta$, introduce a new index i^* and replace $\alpha\beta$ with $\alpha_{i^*}\beta_{i^*}$. We note that each 4ℓth $\alpha\beta$ with $1 \le \ell \le M-1$ is replaced by $\alpha_{i^*}\beta_{i^*}$ with the unique index i^*.

Next, for each clause c_ℓ with $1 \le \ell \le M$ by assuming which is either $(x_i \vee x_j \vee x_k)$ or $(\neg x_i \vee \neg x_j \vee \neg x_k)$, we transform $\pi(i,2)$,$\pi(j,2)$, and $\pi(k,2)$ by replacing

(i) $\Delta(i,2)_\ell$ with $\alpha_{\ell_0}\beta_{\ell_0}\alpha_{\ell_2}\beta_{\ell_2}\alpha_{\ell_1}\beta_{\ell_1}\alpha_{\ell_3}\beta_{\ell_3}$,

(ii) $\Delta(j,2)_\ell$ with $\alpha_{\ell_0}\beta_{\ell_0}\alpha_{\ell_3}\beta_{\ell_3}\alpha_{\ell_2}\beta_{\ell_2}\alpha_{\ell_1}\beta_{\ell_1}$,

(iii) $\Delta(k,2)_\ell$ with $\alpha_{\ell_0}\beta_{\ell_0}\alpha_{\ell_1}\beta_{\ell_1}\alpha_{\ell_3}\beta_{\ell_3}\alpha_{\ell_2}\beta_{\ell_2}$,

where $\ell_0, \ell_1, \ell_2, \ell_3$ are new indices introduced for each clause. Remained Δ's of each $\pi(i,2)$ are replaced with $(\alpha_{i^*}\beta_{i^*})^4$ with introducing four new indices for each segment. Similarly we transform $\pi(i,3)$,$\pi(j,3)$, and $\pi(k,3)$ by introducing new indices $\ell'_0, \ell'_1, \ell'_2, \ell'_3$, and i^*. Then, construct $s_X = \pi_1\,\mathtt{z}\cdots\pi_n\,\mathtt{z}$ for the modified string $\pi_i = \pi(i,1)\pi(i,3)\pi(i,3)$.

Finally, we define s_α and s_β. The string s_α is the concatenation of all $\alpha_{i_1}\,\beta_{i_1}\,\mathtt{z}\cdot\alpha_{i_2}\,\beta_{i_2}\,\mathtt{z}$, where $\alpha_{i_1}\,\beta_{i_1}$ and $\alpha_{i_2}\,\beta_{i_2}$ are the first and the last $(\alpha\beta)$ segments of π_i (the prefix and the suffix of π_i of length 10), respectively. The string s_β is the concatenation of all $\beta_i\,\beta_j\,\mathtt{z}$ such that $\beta_i\,\beta_j = \mathtt{a}_i\mathtt{b}\,\mathtt{a}_j\mathtt{b}$ with $i \ne j$ appears in s_X.

The nonnegative integer threshold for transforming Min SLP to the decision problem is $f_C + 2M$, function for computing this number will be specified later.

Now we show that the algorithm given above is a reduction by the lemmas in the following. At first, we introduce some notions and definitions.

Let w be a string in Σ^+, P an SLP for w over N and Σ, and u a substring of w such that a nonterminal $\sigma \in N$ that derives u exists. Then, a sequence of substrings $u_1, \ldots, u_k$ of u is said to be a *partitioning of u by P* if $u = u_1 \cdot \cdots \cdot u_k$ and for each u_i with $1 \le i \le k$ a nonterminal that derives u_i exists in P. We denote a partitioning of u by the notion $\langle u_1 \cdots u_k\rangle_P$, and by $\langle u_1 \cdots u_k\rangle$ if it is not necessary to specify P. Let w be $\alpha_1\beta_1\cdots\alpha_k\beta_k$ for an odd number k. This can be also represented as

$$\begin{aligned} w &= \alpha_1 \cdot \beta_1\beta_2 \cdot \alpha_2\alpha_3 \cdot \cdots \cdot \alpha_{k-1}\alpha_k \cdot \beta_k \\ &= \beta_1 \cdot \alpha_1\alpha_2 \cdot \beta_2\beta_3 \cdot \cdots \cdot \beta_{k-1}\beta_k \cdot \alpha_k. \end{aligned}$$

Each $\alpha_i\beta_i$ in w is said to be an *$\alpha\beta$-segment* of w, and each $\alpha_i\alpha_{i'}$ and $\beta_i\beta_{i'}$ with $i \ne i'$ are said to be an *α-segment* and a *β-segment*, respectively. By assuming an SLP that contains nonterminals deriving those segments, we say the partitioning of w according to the former representation the *left-aligned partitioning*, and the later the *right-aligned partitioning*. We say a partitioning of s_X is *regular* if each π_i is left-aligned or right-aligned.

Lemma 2. Let $w_i = \pi_i s_\alpha s_\beta$ and let P_i be a smallest SLP for w_i, for each $1 \le i \le n$. Then, P_i is a regular partitioning of π_i.

Proof. The string π_i is the concatenation of $12M-1$ different $\alpha\beta$-segments. The string s_α contains only two of them, and s_β contains at most one β-segment for each pair of different indices. Thus, if the 3-CNF monotone formula is sufficiently large, then the size of an SLP for w_i depends on how many nonterminals deriving substrings of π_i can be shared with those for $s_\alpha s_\beta$.

Both the left- and the right-aligned partitioning contain $6M-1$ different β-segments that can be shared with s_β. Both the first segment of the left-aligned partitioning and the last segment of the right-aligned partitioning can be shared with s_α. Thus, the left- and the right-aligned partitioning for π_i requires the same number of rules.

Then, it must contain at least one isolated β such as $\langle \cdots \alpha_j \beta_j \alpha_{j'} \cdots \rangle_{P_i}$ or $\langle \cdots \alpha_j \beta_{j'} \alpha_{j'} \cdots \rangle_{P_i}$ for some $j \neq j'$. Since neither $\alpha_j \beta_j$ nor $\beta_{j'} \alpha_{j'}$ is contained in s_α and s_β, any SLP having such a partitioning is not smallest. □

Lemma 3. Let c_ℓ be a clause formed from literals over x_i, x_j and x_k. Let γ_i be either the left-aligned or the right-aligned partitioning of the string $\Delta(i,2)_\ell$ defined for c_ℓ. Let γ_j and γ_k be the partitioning for $\Delta(j,2)_\ell$ and $\Delta(k,2)_\ell$, whose alignments are same with that of γ_i, respectively. Then, if both a left- and a right-aligned partitioning exist, there is exactly one α-segment appears in exactly two of $\gamma_i, \gamma_j, \gamma_k$; otherwise all α-segments are different.

Proof. The left-aligned partitioning of $\Delta(i,2)_\ell$, $\Delta(j,2)_\ell$, $\Delta(k,2)_\ell$ are, respectively,

$$\langle \alpha_{\ell_0} \beta_{\ell_0} \beta_{\ell_2} \alpha_{\ell_2} \alpha_{\ell_1} \beta_{\ell_1} \beta_{\ell_3} \alpha_{\ell_3} \rangle,$$
$$\langle \alpha_{\ell_0} \beta_{\ell_0} \beta_{\ell_3} \alpha_{\ell_3} \alpha_{\ell_2} \beta_{\ell_2} \beta_{\ell_1} \alpha_{\ell_1} \rangle,$$
$$\langle \alpha_{\ell_0} \beta_{\ell_0} \beta_{\ell_1} \alpha_{\ell_1} \alpha_{\ell_3} \beta_{\ell_3} \beta_{\ell_2} \alpha_{\ell_2} \rangle.$$

There is no α-segment appearing twice above. For the right-aligned partitioning, it can be check in the same way.

We can also check the converse direction by enumerating a set of partitioning. For instance, $\alpha_{\ell_1} \alpha_{\ell_3}$ appears twice in $\langle \beta_{\ell_0} \alpha_{\ell_0} \alpha_{\ell_2} \beta_{\ell_2} \beta_{\ell_1} \alpha_{\ell_1} \alpha_{\ell_3} \beta_{\ell_3} \rangle$ and $\langle \alpha_{\ell_0} \beta_{\ell_0} \beta_{\ell_1} \alpha_{\ell_1} \alpha_{\ell_3} \beta_{\ell_3} \beta_{\ell_2} \alpha_{\ell_2} \rangle$. □

Lemma 4. Let $w = s_X s_\alpha s_\beta$ and $s_X = \pi_1 \mathtt{z} \cdots \pi_n \mathtt{z}$. For any regular partitioning p of s_X, there exists a smallest SLP P for w that includes p.

Proof. By Lemma 2, an admissible partitioning for each π_i is either a left- or a right-aligned partitioning. Thus, there are 2^n candidates for a smallest SLP. We show that for any combination of n partitioning, there is a smallest SLP that includes them.

Let p_1 be the partitioning of w such that each partitioning for π_i is left-aligned. Let p be any partitioning of w. We assume for s_X both p_1 and p is regular. We estimate the number of rules required by these partitioning. Since every β-segment appears in a candidate partitioning of any π_i and is shared with that of s_β, the number of rules for π_i's by p_1 is equal bers of rules for p_1 and for p depends only on the strings of the form α-segments.

Recall that $\pi_i = \pi(i,1)\,\pi(i,2)\,\pi(i,3)$. $\pi(i,1)$ consists of $4M-1$ $\alpha\beta$-segments, and $\pi(i,2)$ and $\pi(i,3)$ consist of $4M$ $\alpha\beta$-segments, respectively. We first focus on $\pi(i,1)$. By the definition of π_i, for each $1 \leq \ell \leq M$, $(4\ell-2)$th segment of π_i is $\alpha_{4\ell-2}\beta_{4\ell-2}$ if the clause c_ℓ has x_i or $\neg x_i$, and $\alpha_{i^*}\beta_{i^*}$ otherwise, where i^* is the unique index for this segment. Since the formula is a 3-CNF, for each $1 \leq \ell \leq M$, there are $\pi(i,1), \pi(j,1), \pi(k,1)$ having $\alpha_{4\ell-1}\beta_{4\ell-1}$. In the case of p_1, all partitioning of them are of the form $\langle \cdots \beta_{4\ell-3}\beta_{4\ell-2}\alpha_{4\ell-2}\alpha_{4\ell-1} \cdots \rangle$. Since $\beta_{4\ell-3}\beta_{4\ell-2}$ is shared with s_β, we need only two rules $\sigma \to \sigma_1\sigma_2$ and $\sigma_2 \to \sigma_3\sigma_4$ to compress $\beta_{4\ell-3}\beta_{4\ell-2}\alpha_{4\ell-2}\alpha_{4\ell-1}$ to a nonterminal such that σ_1 to $\beta_{4\ell-3}\beta_{4\ell-2}$, σ_3 to $\alpha_{4\ell-2}$, and σ_4 to $\alpha_{4\ell-1}$.

Assume that p has at least one left-aligned partitioning and at least one right-aligned partitioning for $\pi(i,1), \pi(j,1), \pi(k,1)$. Then, there are both partitioning of $\langle \cdots \beta_{4\ell-3}\beta_{4\ell-2}\alpha_{4\ell-2}\alpha_{4\ell-1} \cdots \rangle$ and $\langle \cdots \alpha_{4\ell-3}\alpha_{4\ell-2}\beta_{4\ell-2}\beta_{4\ell-1} \cdots \rangle$. In this case, we need four rules to compress these segments to nonterminals.

We show that this difference of the numbers of rules between p_1 and p is canceled by the partitioning of $\{\pi(i,2), \pi(j,2), \pi(k,2)\}$ and $\{\pi(i,3), \pi(j,3), \pi(k,3)\}$. By Lemma 3, in the case of p_1, the partitioning of $\pi(i,2), \pi(j,2), \pi(k,2)$ contains no α-segments appearing twice. On the other hand, in the case of p, it contains one of the three left-aligned partitioning and one of the three right-aligned partitioning. It follows that there is exactly one α-segments appearing twice. It is easy to see that this condition is satisfied for the combination of $\pi(i,3), \pi(j,3), \pi(k,3)$ by their definition. As a consequence, the increase of rules is always balanced between $\{\pi(i,1), \pi(j,1), \pi(k,1)\}$ and $\{\pi(i,2), \pi(j,2), \pi(k,2)\} \cup \{\pi(i,3), \pi(j,3), \pi(k,3)\}$. Thus, we obtain that the numbers of rules required by p_1 and p is equal. Therefore, each partitioning of w among 2^n distinct ones is a partitioning of w by a smallest SLP for w. □

Proof of Theorem 1. We show that, for each monotone 3-CNF $F = (X, C)$, we can define a polynomial $f(M,n)$ such that F is satisfiable if and only if an SLP P for s satisfying $|P| \leq f(M,n)$ exists.

s that simulates the assignment of 0/1-value for the variables $x_1, \ldots, x_n$. By Lemma 2 and Lemma 4, any smallest SLP for w must contain a regular partition of s_X, and for any regular partition p of s_X, there exists a smallest SLP for w that implies p. Thus, every regular partitioning of s_X is associated with the assignments of 0/1-value for n variables: the left-aligned partitioning of π_i of s_X is associated with $x_i = 1$ and the right-aligned partitioning of π_i is associated with $x_i = 0$. By Lemma 4, for any regular partitioning p, there is a smallest SLP P_p for w that implies p. Then we can select the left partitioning for all π_i of s_X to compute an smallest SLP for $s_X s_\alpha s_\beta$. We can find all distinct $\beta_i \beta_{i'}$'s in s_X in polynomial time. Using these pairs, we can compute a smallest rules which derive the string inside of the left partitioning. By Lemma 4, the number of rules in the outside of the partitioning is invariable. Thus, we can deterministically compute f_C depending on w such that $|P_p| = f_C(M, n)$ for any regular partitioning p.

Next let us consider the part $s_C = \pi^+ \pi^-$. The string π^+ is the concatenation of all $\alpha_{4\ell-2}\alpha_{4\ell-1}$ with $1 \leq \ell \leq m$ for m positive clauses. Let x_i be a variable occurring in c_ℓ. Then, π_i contains the substring $\alpha_{4\ell-2}\alpha_{4\ell-1}$. Thus, the string $\alpha_{4\ell-2}\alpha_{4\ell-1}$ of s_C can be shared with the α-segment of π_i if a partition for s_X contains the left partition of π_i. There is the correspondence between (1) $x_i = 1$ and c_ℓ contains x_i, and (2) the left-aligned partitioning is chosen for π_i and s_C contains $\alpha_{4\ell-2}\alpha_{4\ell-1}$. For π^-, we can also construct the correspondence between (3) $x_i = 0$ and c_ℓ contains $\neg x_i$, and (4) the right-aligned partitioning is chosen for π_i and s_C contains $\alpha_{4\ell-3}\alpha_{4\ell-2}$. Thus, C is satisfiable if and only if there is a smallest SLP P_p for w with a regular partitioning p that compresses s to the string $\sigma_1 \mathtt{z} \cdots \mathtt{z}\, \sigma_M\, \mathtt{z}\, \sigma_w$ and derives $\alpha_{4\ell-2}\alpha_{4\ell-1}$ from σ_ℓ if c_ℓ is positive, and $\alpha_{4\ell-3}\alpha_{4\ell-2}$ if c_ℓ is negative. Hence, C is satisfiable if and only if an SLP P for s such that $|P| \leq f_C(M, n) + 2M$ exists. □

4 Approximation algorithms for Min SLP

In this section, we deal with strings over a binary alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and we present two algorithms finds an SLP for an arbitrary string in Σ^*. This achieves a worst-case approximation ratio which is slightly and strictly smaller than the straightforward upper bound $(n-1)/\log n$ of approximation ratios. This algorithm gives a basic idea of the second one, and is applicable for alphabets whose sizes are greater than two. The second one is designated to achieve a much better ratio for a restricted class of strings. It finds an SLP whose size is at most 6 times the optimum.

4.1 A general algorithm by semi-complete binary diagram strategy

The algorithm '$A1$' for Min SLP is presented in Figure 1. It simply repeats transforming the input and the intermediately produced strings to the sequence of new rules each of which concatenates two contiguous symbols of the original from the left to right. So the compression is achieved only by avoiding creating redundant rules.

Algorithm $A1$ (*Input:* a string $w \in \Sigma^*$; *Output:* an SLP P which derives w.)

Step 1. Let $i := 1$ and $P = \emptyset$.

Step 2. Repeat while $|w| > 1$ the following:

(a) Add a rule $z^i_\ell \to w[2\ell, 2\ell - 1]$ to P for each $1 \leq \ell \leq \lfloor |w|/2 \rfloor$; nonterminals z^i_ℓ and $z^i_{\ell'}$ with $\ell \neq \ell'$ are the same symbols if and only if their right-hand side are the same. Then let w be the string $z^i_1 \cdots z^i_{\lfloor |w|/2 \rfloor}$ if $|w|$ is even, otherwise let $w := z^i_1 \cdots z^i_{\lfloor |w|/2 \rfloor} \cdot w[|w|]$.

(b) Let $i := i + 1$.

Step 3. Output P.

Figure 1: Algorithm $A1$.

Theorem 2. Algorithm $A1$ runs in $O(n \log^2 n)$ time, and approximates Min SLP over a binary alphabet within $3n/\log^2 n$.

Proof. Let w be the input string and let $n = |w|$, and let $m = \lceil \log n \rceil$. Then Step 2 loops at most m

rithms, $A1$ can be implemented as an $O(n \log nm)$-time algorithm, i.e. runs in $O(n \log^2 n)$ time.

Let P be the SLP for w produced by $A1$, and T_P the derivation tree of P. Then T_P is, say, a left-aligned complete binary tree whose height is m. The size of P is the number of different labels of internal nodes of T_P, thus $|P| = \sum_{1 \le i \le m} N_i$, where N_i is the number of labels of internal nodes at the depth i of T_P.

Now we show that $|P| < 3n/\log n$. Since the leaves of T_P is either `a` or `b`, at most four different rules which possibly map to `aa`, `ab`, `ba` and `bb` could be made at the first iteration of Step 2. This implies that any internal node at depth $m-1$ is labeled by one of four nonterminals, and thus $N_m \le 4$.

Similarly, by the induction, any node at the depth $m-i$ must be labeled by one of at most 2^{2^i} nonterminals, thus $N_{m-i} \le 2^{2^i}$. On the other hand, the number of nodes in depth $m-i$ cannot exceed $n/2^i$, thus the above inequality holds for every i that satisfies $2^{2^i} \le n/2^i$. This is satisfied by at most $k = \log\log n - 1$. For each ith iteration of Step 2 with $i \ge k+1$, the number of labels is no more than the number of nodes at depth $m-i$. Thus in the worst case, $N_{m-i} = 2^{m-i}$ for $1 \le i \le k+1$. Therefore,

$$\begin{aligned} |P| = \sum_{1 \le i \le m} N_i &\le \sum_{1 \le i \le k} 2^{2^i} + \sum_{1 \le i \le k+1} 2^{k+1-i} \\ &\le 2\sqrt{n} + \frac{2n}{\log n} \le \frac{3n}{\log n}. \end{aligned}$$

Since obviously the size of SLP $|P|$ is no less than $\log n$ for any string of length n, the worst-case approximation ratio of $|P|$ to the optimum is $3n/\log^2 n$. □

4.2 An algorithm for overlap-free strings

A string $w \in \Sigma^*$ has an *overlapping factor* if w contains two overlapping occurrences $s \cdot t \cdot u$ of a string $s \cdot t = t \cdot u$. A string w is *overlap-free* if w has no overlapping factors. The following lemma gives a more precise description of this case.

Lemma 5 (Lothaire [7]). A string w has an overlapping factor if and only if w contains a substring $\mathtt{a}v\mathtt{a}v\mathtt{a}$ with some symbol `a` and a (possibly empty) string v.

Algorithm A2 (*Input:* A string w, *Output:* An SLP P for w.)

Step 1. Let $i := 1$, let s_1, s_2 be the empty string ε, and $P = \emptyset$. Additionally l and r are used to store the detached prefix and suffix.

Step 2. Repeat while $|w| > 1$ the following:

(a) Determine the longest prefix x and the longest suffix y of w that contain neither `aa` nor `bb`.

(b) If $|x|$ is odd then $l := \varepsilon$, if x is empty (a prefix of w is `aabaa` or `bbabb`) then $l := w[1,2]$, otherwise $l := w[1]$.

(c) If $|z|$ is odd then $r := \varepsilon$, if z is empty then $r := w[|w|-1, |w|]$, otherwise $r := w[|w|]$.

(d) Let $s_1 := s_1\, l$, $s_2 := r\, s_2$, $w := pairwise(w[|l|+1, |w| - |r|], i)$, and $i := i+1$.

Step 3. Let $w := s_1\, w\, s_2$, and repeat $w := pairwise(w, i)$ and $i := i+1$ while $|w| > 1$.

Step 4. Output P.

Figure 2: The approximation algorithm for MIN SLP on overlap-free strings.

There exist infinitely many overlap-free strings even over a binary alphabet. The following is well known overlap-free infinite string: the *Thue-Morse* [7] string $\mathbf{t} = \mu^\infty(\mathtt{a}) = \mathtt{abbabaabbaab}\cdots$ is obtained by applying a morphism μ to `a` infinitely many time, where $\mu : \Sigma^* \to \Sigma^*$ is defined by $\mu(\mathtt{a}) = \mathtt{ab}$ and $\mu(\mathtt{b}) = \mathtt{ba}$. Since $\mathbf{t}$ is overlap-free, any substring of $\mathbf{t}$ is also overlap-free.

We present the algorithm $A2$ for MIN SLP over the overlap-free strings in Fig. 2. The procedure $pairwise(w, i)$ invokes Step 2-(a) of the algorithm $A1$. The Step 2 repeats, similarly to the algorithm $A1$, approximately $\log n$ time with the size n of input string. On each iteration of Step 2, the algorithm searches for a square $w[j, j+1] = xx$, contiguous two same symbols. If a square has been found, the pairs $\cdots w[j-1, j], w[j+1, j+2] \cdots$ are chosen to make new rules. Therefore the positions of the first and the last pairs depend on whether j and $|w|$ are

For example, assume that abbab is given. Then $A2$ adds the rules $\sigma_1^1 \to \mathtt{ab}$ and $\sigma_2^1 \to \mathtt{ba}$ to P at the first iteration of Step 2, and at the second iteration, $A2$ adds $\sigma_1^2 \to \sigma_1^1\sigma_2^1$ and $\sigma_1^3 \to \sigma_1^2\mathtt{b}$. Then, there are totally four rules. It is not optimal because abbab can be expressed by three rules. Thus, unfortunately, outputs of the algorithm $A2$ are not always optimal. However, $A2$ is an approximation algorithm that achieves a constant worst-case factor.

First, we prove the following lemma.

Lemma 6. Let w be a sufficiently large, overlap-free string on $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Then, there exists a substring v of w that satisfies $v \in \{\mathtt{ab}, \mathtt{ba}\}^*$ and $|v| \geq |w| - 4$.

Proof. An overlap-free string w contains no three (or longer) contiguous same symbols such as aaa. Therefore, w can be uniquely expressed as $w = x_1y_1 \cdots x_ny_nx_{n+1}$ where every y_j is a square either aa or bb, and each x_i contains no squares, i.e. is an alternating string. If all the lengths of x_i are even, then the substring $w[2, |w-1|]$ is in $\{\mathtt{ab}, \mathtt{ba}\}^*$ and the statement holds. So we consider the cases such that for some $1 \leq i \leq n$ the length of x_i is odd. Note that in the following discussion obviously we can swap the symbols a with b.

Assume that $y_i = y_{i+1} = \mathtt{aa}$ and $|x_{i+1}|$ is odd. In the case $|x_{i+1}| = 1$, $y_i\, x_{i+1}\, y_{i+1} = \mathtt{aabaa}$. If $y_i\, x_{i+1}\, y_{i+1}$ is a proper substring of w, then the last symbol of x_i and the first symbol of x_{i+2} are b. It implies that w contains an overlapping string baabaab, a contradiction. In the case $|x_{i+1}| = 2\ell + 1$ for some $\ell > 0$, $y_i\, x_{i+1}\, y_{i+1} = \mathtt{aa}(\mathtt{ba})^\ell\mathtt{b\,aa}$. This falls into the fact that w contains an overlapping string aba. Thus, w is overlap-free only if $x_{i+1} = \mathtt{b}$ and either (i) $i = 1$ and $x_1 = \varepsilon$, or (ii) $i = n-1$ and $x_{n+1} = \varepsilon$. Note that, if $y_i = \mathtt{aa}$ and $y_{i+1} = \mathtt{bb}$, then since w is overlap-free, x_{i+1} must be $(\mathtt{ba})^\ell$ for some $\ell \geq 0$ and thus $|x_{i+1}|$ is even.

As a consequence, $|x_i|$ can be odd only if i is either $1, 2, n$, or $n+1$. Moreover, either $|x_1|$ or $|x_2|$ can be odd, and either $|x_{n+1}|$ or $|x_n|$ can be odd. The length of squares $y_1, \ldots, y_n$ is always two. Summarizing these facts, if $|x_1|$ and $|x_{n+1}|$ are odd, then we can choose $v = w \in \{\mathtt{ab}, \mathtt{ba}\}^+$. If $|x_1|$ and $|x_n|$ are odd, $w[1, |w|-2] \in \{\mathtt{ab}, \mathtt{ba}\}^+$. If $|x_2|$ and $|x_{n+1}|$ are odd, $w[3, |w|-2] \in \{\mathtt{ab}, \mathtt{ba}\}^+$. □

Theorem 3. The algorithm $A2$ for MIN SLP achieves the worst-case performance ratio 6 for overlap-free strings over a binary alphabet.

Proof. Let w_i be the string w produced by the ith iteration of Step 2. By the definition of the procedure $pairwise(w, i)$, $|w_{i+1}| \leq |w_i|/2$. It follows that Step 2 is repeated at most $\log n$ time with $|w| = n$. By Lemma 6, at the ith iteration, $A2$ find the longest substring w_i of w_{i-1} such that $w_{i-1} = l\, w_i\, r$ and $w_i \in \{a_{i-1}b_{i-1}, b_{i-1}a_{i-1}\}^+$, where a_{i-1} and b_{i-1} are nonterminals added at the $(i-1)$th iteration. Thus, the number of rules added to P in whole the iterations of Step 2 is at most $2\log n$.

At each iteration of Step 2, two strings l and r are added to s_1 and s_2. Again, by Lemma 6, $|l|, |r| \leq 2$. Thus, the length of the string $s_1\, w\, s_2$ at Step 3 is at most $4\log n + 1$, and the number of rules added in Step 3 is at most $4\log n$. Therefore, the total number of rules is at most $6\log n$ and by Lemma 1, the algorithm $A2$ is a 6-approximation algorithm. □

5 Conclusion

Text compression is as fertile an area for research now as it was 50 years ago when computing resources were scarce. Recently, compression scheme based on grammar transform attracts special concerns. In this scheme, a given text string is converted into a context-free grammar that derives strictly the string and then encoded. Such a grammar is called an admissible grammar. In this paper, we have proved the NP-hardness of the minimization problem for the class of admissible grammars in Chomsky normal form. Unfortunately, the intractability of the problem for general admissible grammars remains an open problem. The two approximation algorithms presented in this paper are limited to the case of a binary alphabet. The interesting task for further research is to develop an approximation algorithm for a general alphabet.

Kida et al. [4] introduced a more general framework than admissible grammars, called collage systems, in which the repetition and the affix truncation operations for strings are introduced in addition

problem of string matching for text strings described in terms of collage system, but they did not discuss the complexity of computing a collage system that represents a given text string. To analyze the complexity of the minimization problem for collage systems will be future work.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, 1999.

[2] S. De Agostino and J. A. Storer. On-line versus off-line computation in dynamic text compression. *Inform. Process. Lett.*, 59:169–174, 1996.

[3] E. hui Yang and J. C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform–part one: Without context models. *IEEE Trans. on Inform. Theory*, 46(3):755–777, 2000.

[4] T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theoret. Comput. Sci.*, 2001. (to appear).

[5] J. C. Kieffer and E. hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. on Inform. Theory*, 46(3):737–754, 2000.

[6] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. Data Compression Conference (DCC'99)*, pages 296–305. IEEE Computer Society, 1999.

[7] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications.* Addison-Wesley, 1983.

[8] C. Nevill-Manning and I. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40(2/3):103–116, 1997.

[9] C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artificial Intelligence Research*, 7:67–82, 1997.

ity. Addison Wesley, 1993.

[11] D. Salomon. *Data Compression: the complete reference.* Springer, second edition, 1998.

[12] D. Sieling and I. Wegener. Reduction of obdds in linear time. *Inf. Process. Lett.*, 48:139–144, 1993.

[13] J. Storer and T. Szymanski. Data compression via textual substitution. *J. Assoc. Comput. Mach.*, 29(4):928–951, 1982.

[14] J. A. Storer and T. G. Szymanski. The macro model for data compressoin. In *Proc. 10th Ann. Sympo. on Theory of Computing*, pages 30–39, San Diego, California, 1978. ACM Press.

[15] T. A. Welch. A technique for high performance data compression. *IEEE Comput.*, 17:8–19, 1984.

[16] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Inform. Theory*, IT-23(3):337–349, 1977.

[17] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. on Inform. Theory*, 24(5):530–536, 1978.