O(log n)-Approximation Algorithm for Grammar-Based Compression

Sakamoto, Hiroshi
Department of Informatics, Kyushu University

Shimozono, Shinichi
Department of Artificial Intelligence, Kyushu Institute of Technology

Shinohara, Ayumi
Department of Informatics, Kyushu University

Takeda, Masayuki
Department of Informatics, Kyushu University

出版情報：DOI Technical Report. 201, 2002-01. Department of Informatics, Kyushu University
バージョン：
権利関係：

$O(\log n)$-Approximation Algorithm for Grammar-Based Compression

(This paper is submitted to track A)

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

[†]Department of Informatics, Kyushu University
Fukuoka 812-8581, Japan
[‡]Department of Artificial Intelligence, Kyushu Institute of Technology
Iizuka 820-8502, Japan
{hiroshi, ayumi, takeda}@i.kyushu-u.ac.jp
sin@ai.kyutech.ac.jp

Abstract. In the grammar-based compression scheme, a given text string is transformed into a context-free grammar G such that $L(G) = \{w\}$ and then encoded in an appropriate manner. The compression is thus regarded as an optimization problem of minimizing the size of G for a given string w of length n. For the APX-hard problem an $O(\log n)$-approximation algorithm with arbitrary string is presented. The previously known best approximation ratio was $O(\sqrt{n/\log n})$ [8].

Keywords: *grammar-based compression, approximation algorithm, data compression, string algorithm.*

Correspondence:
Dr. Hiroshi Sakamoto
Department of Informatics, Kyushu Univ.
hiroshi@i.kyushu-u.ac.jp
phone: +81-92-642-2693, fax: +81-92-642-2698

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 [20, 21], often referred to as LZ77 and LZ78, and their variants (e.g., [18, 15]) are most widely-used universal lossless compression algorithms, together with the arithmetic coding algorithms (see, e.g., [13]).

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 [16] 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-hard. Also, De Agostino and Storer [2] generalized the LZ78 scheme and then proved that the computation of the optimum parsing is NP-hard.

Recently, there have emerged new compression algorithms, such as Sequitur [11, 10] and Re-Pair [7], 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 [19] presented a grammar transform algorithm, and showed that the algorithm is asymptotically optimum (i.e. universal) for a broad class of sources, including stationary, ergodic sources.

Very recently, Lehman and Shelat [8] showed that even approximating the size of the smallest grammar within a small constant factor is NP-hard, and established upper and lower bounds on the approximation ratios of four existing grammar-based compression algorithms. In particular, the BISECTION algorithm [6] has the upper bound $O(\sqrt{n/\log n})$ and the lower bound $\Omega(\sqrt{n}/\log n)$ for the size n of an instance string.

It has not been known the technique for the relative approximation of this problem. In fact all the upper bounds of known algorithms were proved by the absolute error. So in this paper we introduce the method for the relative comparison which enable us to analyze the performance ratio of an optimum solution and an algorithm for a same instance. Consequently, for the problem MIN ADMISSIBLE GRAMMAR, we present a polynomial-time algorithm that approximates

the problem within $9(1 + 2\ln|w|)$ for every string w. This ratio is exponentially smaller than the previously known result.

The rest of this paper is organized as follows. In Section 2 we introduce some standard definitions for strings and context-free grammars, and then define our minimization problem. In this section MIN ADMISSIBLE GRAMMAR is reduced to a sub-problem for finding a restricted grammar. The reduced problem is equivalent to MIN ADMISSIBLE GRAMMAR with respect to a constant factor. In Section 3 the problem MIN INTERVAL COVER is defined, which is closely related to the reduced MIN ADMISSIBLE GRAMMAR. We present a polynomial-time $O(\log n)$-approximation algorithm for MIN INTERVAL COVER and show that the size of an optimum solution for it is smaller than that for the reduced MIN ADMISSIBLE GRAMMAR. Moreover we present an algorithm that constructs an admissible grammar from a solution for MIN INTERVAL COVER whose size is at most the size of the solution. Thus our algorithm achieves the performance ratio $O(\log n)$. Section 4 contains final remarks and states open problems.

2 Notions and Definitions

Let A be a finite alphabet. The set of all the strings formed from symbols in A, including the empty string ε, is denoted by A^*. The length of a string $w \in A^*$ is denoted by $|w|$, and $w[i]$ with $1 \leq i \leq |w|$ refers to the ith symbol of w. A string $w[i, j]$ formed from the symbols from the ith to the jth of w, with $1 \leq i \leq j \leq |w|$, is said to be a *substring of* w.

Let Σ and N be finite alphabets, and we assume a fixed order for each alphabet. In the following we distinguish these notions and say a symbol in Σ a *terminal* (*symbol*) and a symbol in N a *non-terminal* (*symbol*). A *context-free grammar* (CFG) $G = (\Sigma, N, P)$ is a triple where P is a relation between N and $(\Sigma \cup N)^+$. An element of P is said to be a *production rule of* G, which represents a rewriting rule of symbols in strings over $\Sigma \cup N$, and is denoted in the form $\alpha \to \beta_1 \cdots \beta_k$. The size $size(G)$ of a grammar G is the total lengths of production rules in P. A context-free language $L(G)$ for G is a possibly infinite set of all the strings derived from the first symbol in N with respect to P. A context-free grammar is said to be an *admissible grammar* if the language $L(G)$ produced by G is a singleton set, i.e. $|L(G)| = 1$. We say an admissible grammar G is a *straight-line program* if every rule in P implies from a non-terminal to a pair of symbols, i.e. is of the form $\alpha \to X \cdot Y$ with $X, Y \in \Sigma \cup N$.

Our target is the following minimization problem.

Problem 1. (MIN ADMISSIBLE GRAMMAR)
INSTANCE: A string $w \in \Sigma^*$.
SOLUTION: An admissible grammar G such that $L(G) = \{w\}$.
MEASURE: The size $size(G)$ of G.

From now on, we only consider the case where the target grammars are straight-line program. The following lemma simply tells us that in contrast to the hardness of this problem that will be shown it is powerful enough without loss of generality.

Lemma 1. *For any string w, the size $size(G)$ of a smallest admissible grammar G for w and the size of a smallest straight-line program G' for w satisfies the inequality $size(G') \leq 2 \cdot size(G)$.*

This can be verified by the fact that every rule with k symbols in its right-hand side can be replaced with at most $2k$ rules implied from new non-terminal symbols introduced for each rules.

3 Approximation Algorithm for Min Admissible Grammar Problem

In this section we present a polynomial-time approximation algorithm for MIN ADMISSIBLE GRAMMAR consisting of two algorithms $A1$ and $A2$. The algorithm $A1$ solves *minimum interval cover problem*, which is a relaxed version of the target problem MIN ADMISSIBLE GRAMMAR. The algorithm $A2$ receives an output of $A1$ and constructs an SLP G.

3.1 Minimum interval cover problem

First we use the following nonstandard notions for any string $w \in \Sigma^*$.

- $Sub_2(w)$ refers the set of all the substrings of w whose lengths are at least two, i.e. $Sub_2(w) = \{w[i,j] \mid 1 \leq i \leq j \leq |w| \text{ and } j \geq i+1\}$.
- $w_{(k)}$ refers the kth string in $Sub_2(w)$ with respect to the lexicographic order on Σ^*.
- $Int(w)$ refers the set of all the intervals longer than two, attached with the corresponding substrings in $Sub_2(w)$, i.e. $Int(w) = \{\langle i,j;k\rangle \mid w[i,j] = w_{(k)} \in Sub_2(w)\}$.

Two integers i and j of $\langle i,j;k\rangle$ are said to be the *start-point* and *end-point* of the interval, respectively. An integer i' is said to be *contained by an interval* $\langle i,j;k\rangle$ if $i \leq i' \leq j$.

Definition 1. *Two intervals are said to be well-parenthesized if (1) the start-point and end-point of one are both contained by the other, or (2) neither the start-point nor end-point of one are contained by the other. Moreover we say that two intervals overlap if they are not well-parenthesized.*

Example 1. If the intervals $t_1 = \langle 2,4;k_1\rangle$, $t_2 = \langle 1,5;k_2\rangle$, and $t_3 = \langle 5,6;k_3\rangle$ are defined for some integers k_1, k_2, and k_3, then t_1 and t_2 are well-parenthesized and so is t_1 and t_3, but t_2 and t_3 overlap each other.

Definition 2. *Let S and S' be sets of intervals. We say that S covers S' if for each interval $t' \in S'$, there exists an interval $t \in S$ that either coincides t' or overlaps t'.*

We define in spite of MIN ADMISSIBLE GRAMMAR the following relaxed problem with an idea of intervals of a string.

Problem 2. (MIN INTERVAL COVER)
INSTANCE: A string $w \in \Sigma^*$.
SOLUTION: An *interval cover* C of $Int(w)$, i.e. a subset $C \subseteq Int(w)$ of well-parenthesized intervals which covers $Int(w)$.
MEASURE: The cardinality of $\{k \mid \langle i, j; k\rangle \in C\}$, which is denoted by $size(C)$.

Now we explain how MIN INTERVAL COVER relates to MIN ADMISSIBLE GRAMMAR by an example below.

Example 2. Let $w = ababb$. Then all $w_{(k)}$'s are the following nine substrings.

$$Sub_2(w) = \left\{ \begin{array}{l} w_{(1)} = ab, \quad w_{(2)} = ba, \quad w_{(3)} = bb, \quad w_{(4)} = aba, w_{(5)} = abb, \\ w_{(6)} = bab, w_{(7)} = abab, w_{(8)} = babb, \quad w_{(9)} = w \end{array} \right\}.$$

The set $Int(w)$ of intervals is defined as

$$Int(w) = \left\{ \begin{array}{l} \langle 1, 2; 1\rangle, \langle 3, 4; 1\rangle, \langle 2, 3; 2\rangle, \langle 4, 5; 3\rangle, \\ \langle 1, 3; 4\rangle, \langle 3, 5; 5\rangle, \langle 2, 4; 6\rangle, \langle 1, 4; 7\rangle, \langle 2, 5; 8\rangle, \langle 1, 5; 9\rangle \end{array} \right\}.$$

The following two sets C_1 and C_2 both cover all intervals of $Int(w)$. However the first two elements $\langle 2, 3; 2\rangle$ and $\langle 3, 5; 5\rangle$ of C_1 overlap each other. Thus C_1 is not a solution for the instance. On the other hand, since all intervals in C_2 are well-parenthesized, C_2 is one of solutions.

$$C_1 = \{\langle 2, 3; 2\rangle, \langle 3, 5; 5\rangle, \langle 1, 4; 7\rangle, \langle 1, 5; 9\rangle\},$$
$$C_2 = \{\langle 1, 2; 1\rangle, \langle 3, 4; 1\rangle, \langle 1, 4; 7\rangle, \langle 1, 5; 9\rangle\}.$$

The size of an interval cover is the number of different indexes k of substrings $w_{(k)}$ attached to the intervals. In this case $size(C_2) = 3$. Any pair of intervals in C_2 are well-parenthesized. Therefore, from C_2 we can straightforwardly construct a set $P = \{S \to Ab, A \to BB, B \to ab\}$ of production rules of an SLP G. This grammar is fortunately a smallest SLP for w.

It is easy to see that MIN INTERVAL COVER is well-defined as follows. For any SLP $G = (\Sigma, N, P)$ for $w \in \Sigma^*$, let $Int_G(w)$ be the set of intervals with respect to G such that

$$Int_G(w) = \{\langle i, j; k\rangle \mid \alpha \in N \text{ derives } w[i, j] = w_{(k)}\}.$$

All intervals in $Int_G(w)$ is well-parenthesized and $Int_G(w)$ covers $Int(w)$, that is, $Int_G(w)$ is an interval cover of $Int(w)$. Thus MIN INTERVAL COVER has at least one solution of size at most $size(Int_G(w)) \leq |Int_G(w)| = |w| - 1$.

We next estimate for a same instance the sizes of optimum solutions for MIN ADMISSIBLE GRAMMAR and for MIN INTERVAL COVER.

Lemma 2. *Let G_{opt} be a smallest SLP for $w \in \Sigma^*$, and let C_{opt} be a minimum interval cover for $Int(w)$. Then the inequality $size(C_{opt}) \leq \frac{1}{2} size(G_{opt})$ holds.*

Proof. Let $G = (\Sigma, N, P)$ be an SLP for w, then obviously $Int_G(w)$ is an interval cover of $Int(w)$ and $size(Int_G(w))$ is at most $|N|$. Since every rule has size two, generally $size(Int_G(w)) \leq |N| = \frac{1}{2} size(G)$ holds. □

The aim of the remained parts of this paper is to show that MIN INTERVAL COVER is approximable within $O(\log n)$ and a small interval cover produces no more than two times SLP for the same string. We present two algorithms $A1$ and $A2$ dealing with the difficulty of according to the following scenario.

The algorithm $A1$ computes a solution C for MIN INTERVAL COVER defined by a string of length n. Let C_{opt} be an optimum solution. Then for the solution C, we show that (1) $size(C)$ is at most $O(\log n)$ times greater than $size(C_{opt})$.

Next we present the algorithm $A2$ which computes an SLP $G = (\Sigma, N, P)$ from the set C of well-parenthesized intervals and show (2) $\frac{1}{2} size(G) = |N| \leq size(C)$.

Let $G'_{opt} = (\Sigma, N'_{opt}, P'_{opt})$ and $G_{opt} = (\Sigma, N_{opt}, P_{opt})$ be an optimum SLP and admissible grammar for w, respectively. Then, by (1) and (2), and using (3) $size(C_{opt}) \leq |N|'_{opt}$, lemma 1 and lemma 2, we can conclude

$$\begin{aligned} \frac{size(G)}{size(G_{opt})} &\leq \frac{2 \cdot size(G)}{size(G'_{opt})} = \frac{2|N|}{|N'_{opt}|} \leq \frac{2 \cdot size(C)}{|N'_{opt}|} \\ &\leq \frac{2 \cdot size(C)}{size(C_{opt})} = O(\log n). \end{aligned}$$

3.2 A greedy algorithm for finding interval covers

The algorithm $A1$ is presented in Fig. 1 and the function f called in line 4 of $A1$ is also presented in Fig. 2. The algorithm $A1$ greedily finds a set $S \subset I_{(k)}$ of well-parenthesized intervals for a fixed index k so that S cover as many remained intervals as possible. In the following we show that $A1$ finds an interval cover for any instance w whose size is no greater than $O(\log n)$ times an optimum.

Let m be the size of x_k found in Step 2 (b) at the first iteration of Step 2, i.e. the largest number $|x_k|$ during an execution of the algorithm. We say, for $1 \leq h \leq m$, all the iterations of Step 2 where the size $|x_k|$ equals h the hth phase of the algorithm. The notion ℓ_h refers to the number of iterations in the hth phase, i.e. the number of indices $k \in Int(w)$ that are chosen in the hth phase. Let X_h be the union of all x_k's subtracted from I in the hth phase. Clearly, $|X_h| = h \cdot \ell_h$ holds.

The function $f(k, I)$ shown in the following computes a maximal set of intervals of $w_{(k)}$ in I that covers points in P as many as possible and are not overlapping each other. Then its result N_k is adopted into S and intervals in x_k that cannot be used in an SLP with them are wasted.

Now let us see that the following lemma holds.

```
1  Algorithm A1 for minimum interval cover problem of input string w.
2     Step 1. Let S ← ∅, I ← Int(w), and K ← {1, ..., |Sub_2(w)|}.
3     Step 2. While I ≠ ∅ do the following:
4        (a) For every k ∈ K, compute the set N_k = f(k, I) and
5           calculate the set x_k = {t ∈ I | ∃u ∈ N_k that overlaps t}.
6              /* The function f(k, I) is defined in Fig. 2. */
7        (b) Find k that maximizes |x_k|.
8        (c) Let S ← S ∪ N_k, I ← I \ x_k, and K ← K \ {k}.
9     Step 3. Output S.
```

Fig. 1. Algorithm for MIN INTERVAL COVER.

```
1  Function f(k, I) for an index k and a set I ⊆ Int(w) of uncovered intervals.
2     Step 1. Let N ← ∅, let I_(k) be the set of intervals of w_(k) in I, and
3        let P ⊂ Z^+ be the set of all start-points and end-points of intervals in I.
4     Step 2. While I_(k) ≠ ∅ do the following:
5        (a) Let t be the left-most interval in I_(k) and T ← {u ∈ I_(k) | t overlaps u}.
6        (b) Choose an interval t' ∈ T that maximizes |P ∩ t'|, then let N ∪ {t'} and
7           I_(k) ← I_(k) \ T.
8     Step 3. Output N.
```

Fig. 2. Function called by $A1$ for finding a set of well-parenthesized intervals so that it covers as many as remained intervals possible.

Lemma 3. *Let S be the output of $A1$ for an input string w. Then the following three conditions hold:*

1. *Algorithm $A1$ always terminates.*
2. *S covers $I(w)$ and all intervals in S are well-parenthesized.*
3. *For any interval $\langle i, j; k\rangle \in S$ with $|w_k| \geq 3$, either*
   - **(a)** *$\langle i, j-1; k'\rangle \in S$,*
   - **(b)** *$\langle i+1, j; k'\rangle \in S$, or*
   - **(c)** *$\langle i, i'; k'\rangle, \langle i'+1, j; k'\rangle \in S$ with $i < i' < j$ exist.*

Proof. The function f in $A1$ produces a set of intervals that do not overlap each other. The algorithm $A1$ wastes all intervals that overlap at least one of them, so at any iteration of Step 2 the intervals remained in I are guaranteed not to overlap any interval chosen in S. It is sufficient to the conditions 1 and 2.

The condition 3 claims that any interval $\langle i, j; k\rangle \in S$ longer than three is exactly divided into two left-aligned and right-aligned intervals either in S or of length one. Any interval strictly contained in $\langle i, j; k\rangle$ coincides to or can be contained in either a left-aligned or a right-aligned interval which can be as long as possible. Also an interval cover for $\langle i, j; k\rangle$ must include all possible intervals

that are longer than one and do not overlap each other. So an interval cover has two left-aligned and right-aligned intervals that exactly divides $\langle i, j; k\rangle$ into two intervals. □

The next lemma shows that the function f guarantees an output contain points no less than $\frac{1}{3}$ times the best.

Lemma 4. *Let I be the set of uncovered intervals in $I(w)$ and P be the set of all the start-points and end-points of intervals in I. Then $p(f(k, I)) \geq \frac{1}{3} \cdot p(I')$ for any $I' \subseteq I_{(k)}$, where $p(S)$ for $S \subseteq I$ refers the cardinality of the set $\{i \in P \mid \exists t \in S\ [\ t$ contains $i\ ]\}$.*

Proof. Each repetition of the function chose an interval that contains the largest number of points among intervals that overlap (and is identical) to the left-most one. Let t_i and t_{i+1} be non-overlapping intervals of $w_{(k)}$ that have been chosen at ith and $i+1$th iterations, respectively. Then the number of points that are in between t_i and t_{i+1} and can be contained by intervals of $w_{(k)}$ is at most the total number of points contained by t_i and t_{i+1}, because all of those remained points must be contained in two intervals t'_i and t'_{i+1} that overlap to t_i and t_{i+1}, respectively. Therefore the number of start-points and end-points of remained intervals which are contained in intervals of $w_{(k)}$ is at most three times the number of points contained in the output of $f(k, I)$. □

Using the above lemma we can show the next lemma which derives the upper bound of the number of intervals covered by intervals of $w_{(k)}$ at one iteration of each hth phase.

Lemma 5. *For each hth phase and each set $I_{(k)}$ of intervals of $w_{(k)}$, the number of intervals in $X_h \cup \cdots \cup X_1$ covered by the intervals in $I_{(k)}$ is less than $9h$.*

Proof. We set $I = X_h \cup \cdots \cup X_1$ for short, which is the set of all intervals not covered yet. Let P be the set of all the start-points and end-points of intervals in I.

For any i and j with $i < j$, it is clear that $i, j \in P$ iff there is an interval $\langle i, j; k'\rangle$ in I. An interval $\langle i, j; k'\rangle$ is removed from I only if either it is chosen or an interval overlapping it is chosen. Thus the number of intervals in I covered by $I_{(k)}$ is at most $max_k = p(I_{(k)}) \cdot (p(I) - p(I_{(k)}))$.

On the other hand, by using Lemma 4, the number of intervals in I covered by the set returned by $f(k, I)$ on hth phase is

$$h \geq p(f(k, I)) \cdot (P(I) - p(f(k, I))) \geq \tfrac{1}{3} p(I_{(k)}) \cdot (p(I) - \tfrac{1}{3} p(f(k, I))) \geq \tfrac{1}{9} max_k.$$

Hence the number of intervals in I covered by intervals in $I_{(k)}$ is at most $9h$ even if we chose all the intervals in $I_{(k)}$. □

Theorem 1. *Let C_{opt} be an optimum solution for an instance of* Min Interval Cover *and C_{A1} be the output of algorithm $A1$ for the same instance for a string w. Then it holds that $size(C_{A1})/size(C_{opt}) \leq 9(1 + 2 \ln |w|)$.*

Proof. Again let $I = X_h \cup \cdots \cup X_1$ be the set of all intervals not covered at hth phase yet. By lemma 5, at any iteration of hth phase, no more than $9h$ intervals are covered by intervals of $I_{(k)}$ for any k. Thus in order to cover all the intervals in I, the intervals for at least $|I|/9h$ different indexes k are needed. Thus we can show the upper bound of $size(C_{A1})/size(C_{opt})$ by the following analysis like [3]. For each $h = 1, \ldots, m$ and $m = |w|^2$,

$$size(C_{opt}) \geq \frac{|I|}{9h} = \frac{h\ell_h + \cdots + \ell_1}{9h}.$$

From this relation,

$$(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m}) \cdot size(C_{opt}) \geq \frac{1}{9} \sum_{g=1}^{m-1} g\ell_g (\frac{1}{g} - \frac{1}{m}).$$

By using Lemma 5 again, we obtain

$$\begin{aligned}(1 + \frac{1}{2} + \cdots + \frac{1}{m}) \cdot size(C_{opt}) &\geq \frac{1}{9m} \sum_{g=1}^{m} g\ell_g + \frac{1}{9} \sum_{g=1}^{m-1} g\ell_g (\frac{1}{g} - \frac{1}{m}) \\ &= \frac{1}{9} \sum_{g=1}^{m} \ell_g = \frac{1}{9} \cdot size(C_{A1}).\end{aligned}$$

Therefore $size(C_{A1})/size(C_{opt}) = 9(1 + \ln m) = 9(1 + 2\ln |w|)$. □

3.3 Construction of grammars

In this subsection we present the algorithm $A2$ which computes a small SLP from an output of $A2$ for input string w. By Lemma 3, we can regard any output of C_{A1} as a binary tree T_{A1} such that the *yield* of it is w, where the yield of a tree is the concatenation of all leaves from the left to right order.

Let V be the set of all the nodes of T_{A1} and $T_{A1}(v)$ be the subtree rooted by $v \in V$. Let $yield(v)$ be the yield of $T_{A1}(v)$ and $\sharp T_{A1} = |\{yield(v) \mid v \in V\}|$. In particular $yield(T)$ denotes $yield(v)$ for the root v of T.

Lemma 6. *For each input string w for $A1$, it holds that $\sharp T_{A1} = size(C_{A1})$.*

Proof. Let $w[i, j] = w_{(k)}$ be a substring of w. If there is a node $v \in V$ such that $yield(v) = w_{(k)}$ and the left most and right most leaves of $T_{A1}(v)$ is i and j, respectively, then it holds that $\langle i, j; k\rangle \in C_{A1}$, and clearly, the converse direction is also true. □

The aim of $A2$ is to construct a tree T' such that the number of different subtrees is at most $size(C_{A1})$. This condition is equivalent to the following claim. Such a tree T' satisfying the claim directly derives the derivation tree of the SLP $G = (\Sigma, N, P)$ for $|N| \leq size(C_{A1})$, and thus, G is at most $O(\log n)$ times greater than G_{opt} by Theorem 1.

Claim. For any nodes u and v of T', $yield(u) = yield(v)$ iff $yield(u_1) = yield(v_1)$ and $yield(u_2) = yield(v_2)$ for the left children u_1 of u and v_1 of v, and the right children u_2 of u and v_2 of v.

Example 3. In this example we illustrate the above claim by the string $w = ababaaba$. The following parenthesized strings w_1 and w_2 correspond to the derivation trees of grammars for w.

$$w_1 = (((a(ba))(ba))((ab)a)), \quad w_2 = (((a(ba))(ba))(a(ba))).$$

The number of different yields in w_1 and in w_2 is six and four, respectively. The tree w_2 is obtained by replacing the subtree $((ab)a)$ of w_1 with $(a(ba))$. The tree w_2 satisfies the claim and it corresponds to the smallest SLP.

---

1 **Algorithm** $A2$ for a binary tree T_{A1} returned by $A1$ for a string w.
2 **Step 1**. Let $T \leftarrow T_{A1}$ and let V be the set of nodes in T.
3 **Step 2**. Select each $w_{(k)}$ in lexicographically small order and do the following:
4 **(a)** Compute $V' = \{v \in V \mid yield(v) = w_{(k)}\}$.
5 **(b)** Find a smallest $\sharp T(v)$ and replace $T(v')$ by $T(v)$ for all $v' \in V'$.
6 **Step 3**. Output the tree $T_{A2} \leftarrow T$.

Fig. 3. Algorithm for minimizing the number of different subtrees of input tree by replacing subtrees.

Lemma 7. *The algorithm $A2$ always terminates and the output tree T satisfies the claim.*

Proof. Since Step 2 in $A2$ is executed at most $|w|$ times, $A2$ terminates for any input. For the output tree T of $A2$, we show that T satisfies the claim by induction on ith iteration of Step 2. For $i = 1$, there are only two types of binary trees such that their yield are $w_{(k)}$ for selected substring $w_{(k)}$ of length three. Then all such threes are replaced by one of them. Thus the claim is true for $i = 1$. We assume the hypothesis on ith iteration of Step 2 and let $w_{(k)}$ be the substring selected $i + 1$th iteration of Step 2. If $yield(v) = w_{(k)}$ for $v \in V$, then v has exactly two children v_1 and v_2 such that $w_{(k)} = w_1 \cdot w_2$, $yield(v_1) = w_1$, and $yield(v_2) = w_2$. Since w_1 and w_2 are selected in Step 2 before $w_{(k)}$, by the assumption, the claim holds for all internal nodes in $T(v_1)$ and $T(v_1)$. In the $i + 1$th iteration, a node v' which minimizes $\sharp T(v')$ is found and all $T(v)$ is replaced if $yield(v) = w_{(k)}$. Then all replaced $T(v)$ also satisfy the claim. Hence the proof of this induction is completed. □

Theorem 2. MIN ADMISSIBLE GRAMMAR *is approximable within $O(\log |w|)$ for every string w.*

Proof. Let C_{A1} be the output of $A1$ for an input string w and T_{A1} be the corresponding tree. Let T_{A2} be the output of $A2$ for input T_{A1} and $G = (\Sigma, N, P)$ be the SLP equivalent to T_{A2}. By Lemma 7, $|N| = \sharp(T_{A2})$. Thus, since $\sharp(T_{A2}) \leq \sharp(T_{A1})$ and $\sharp(T_{A1}) = size(C_{A1})$, we obtain $|N| \leq size(C_{A1})$. Therefore by Theorem 1 and Lemma 2, we can show that $size(G)/size(G_{opt}) \leq 9(1 + 2\ln|w|) = O(\log|w|)$ for a optimum G_{opt}. □

4 Conclusion

We have investigated the approximability of MIN ADMISSIBLE GRAMMAR known to be APX-hard. For this problem, we have presented the polynomial-time $O(\log n)$-approximation algorithm. Its performance ratio is exponentially smaller than the previously known results. However it still remains open whether MIN ADMISSIBLE GRAMMAR is in APX, i.e. it is approximable within a constant.

More general compression scheme *collage system* is presented in [4] in which k-times repetition of a nonterminal and *affix* truncation of a string represented by a nonterminal are allowed as well as concatenation of symbols. Collage system covers LZ77 scheme while grammar-based compression does not. So one of our future works is to investigate the approximability of the optimization problem with respect to collage system.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: combinatorial optimization problems and their approximability properties.* 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. D. S. Johnson. Approximation algorithms for combinatorial problems. J. Computer and System Science, 9:256–278, 1974.
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. J. C. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inform. Theory*, IT-46(4), 1227–1245, 2000.
7. N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. Data Compression Conference (DCC'99)*, pages 296–305. IEEE Computer Society, 1999.
8. E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *Proc. 13th Ann. ACM-SIAM Sympo. on Discrete Algorithms*, 2002. (to appear).
9. M. Lothaire. *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and Its Applications.* Addison-Wesley, 1983.

10. C. Nevill-Manning and I. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 40(2/3):103–116, 1997.
11. C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *J. Artificial Intelligence Research*, 7:67–82, 1997.
12. C. H. Papadimitriou. *Computational complexity*. Addison Wesley, 1993.
13. D. Salomon. *Data compression: the complete reference*. Springer, second edition, 1998.
14. D. Sieling and I. Wegener. Reduction of obdds in linear time. *Inf. Process. Lett.*, 48:139–144, 1993.
15. J. Storer and T. Szymanski. Data compression via textual substitution. *J. Assoc. Comput. Mach.*, 29(4):928–951, 1982.
16. 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.
17. V. V. Vazirani. Approximation algorithm. Springer, 2001.
18. T. A. Welch. A technique for high performance data compression. *IEEE Comput.*, 17:8–19, 1984.
19. 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.
20. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Inform. Theory*, IT-23(3):337–349, 1977.
21. J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. on Inform. Theory*, 24(5):530–536, 1978.