MR-RePair: Grammar Compression based on Maximal Repeats

Isamu Furuya*, Takuya Takagi*, Yuto Nakashima†, Shunsuke Inenaga†,
Hideo Bannai†, Takuya Kida*

* Graduate School of IST,	† Department of Informatics,
Hokkaido University, Japan	Kyushu University, Japan
furuya@ist.hokudai.ac.jp	yuto.nakashima@inf.kyushu-u.ac.jp
tkg@ist.hokudai.ac.jp	inenaga@inf.kyushu-u.ac.jp
kida@ist.hokudai.ac.jp	bannai@inf.kyushu-u.ac.jp

Abstract

We analyze the grammar generation algorithm of the RePair compression algorithm, and show the relation between a grammar generated by RePair and maximal repeats. We reveal that RePair replaces step by step the most frequent pairs within the corresponding most frequent maximal repeats. Then, we design a novel variant of RePair, called MR-RePair, which substitutes the most frequent maximal repeats at once instead of substituting the most frequent pairs consecutively. We implemented MR-RePair and compared the size of the grammar generated by MR-RePair to that by RePair on several text corpora. Our experiments show that MR-RePair generates more compact grammars than RePair does, especially for highly repetitive texts.

1 Introduction

Grammar compression is a method of lossless data compression that reduces the size of a given text by constructing a small context free grammar that uniquely derives the text. While the problem of generating the smallest such grammar is NP-hard [6], several approximation techniques have been proposed. Among them, RePair [11] is known as an off-line method that achieves a high compression ratio in practice [7, 9, 20], despite its simple scheme. There have been many studies concerning RePair, such as extending it to an online algorithm [13], improving its practical working time or space [5, 17], applications to other fields [7, 12, 18], and analyzing the generated grammar size theoretically [6, 15, 16].

Recently, maximal repeats have been considered as a measure for estimating how repetitive a given string is: Belazzougui et al. [4] showed that the number of extensions of maximal repeats is an upper bound on the number of runs in the Burrows-Wheeler transform and the number of factors in the Lempel-Ziv parsing. Also, several index structures whose size is bounded by the number of extensions of maximal repeats have been proposed [2, 3, 19].

In this paper, we analyze the properties of RePair with regard to its relationship to maximal repeats. As stated above, several works have studied RePair, but, to the best of our knowledge, none of them associate RePair with maximal repeats. Moreover, we propose a grammar compression algorithm, called MR-RePair, that focuses on the property of maximal repeats. Ahead of this work, several off-line grammar compression schemes focusing on (non-maximal) repeats have been proposed [1, 10, 14]. Very recently, Gańczorz and Jeż addressed to heuristically improve the compression ratio of RePair with regard to the grammar size [8]. However, none of these techniques use the properties of maximal repeats. We show that, under a specific condition, there is a theoretical guarantee that the size of the grammar generated by MR-RePair is smaller than or equal to that generated by RePair. We also confirmed the effectiveness of MR-RePair compared to RePair through computational experiments.

Contributions: The primary contributions of this study are as follows.

1. We analyze RePair and show the relation between a grammar generated by RePair and maximal repeats.

2. We design a novel variant of RePair called MR-RePair, which is based on substituting the most frequent maximal repeats.

3. We implemented our MR-RePair algorithm and experimentally confirmed that MR-RePair reduces the size of the generated grammar compared to RePair; in particular, the size decreased to about 55% for a highly repetitive text that we used in our experiment.

The remainder of this paper is organized as follows. In Section 2, we review the notations of strings and the definitions of maximal repeats, grammar compression, and RePair. In Section 3, we analyze RePair and show the relation between RePair and maximal repeats. In Section 4, we define MR-RePair, compare it with RePair, and describe the implementation of it. In Section 5, we present experimental results of comparison to RePair. Finally, we conclude the paper in Section 6.

2 Preliminaries

In this sections, we provide some notations and definitions to be used in the following sections. In addition, we recall grammar compression and review the RePair.

2.1 Basic notations and terms

Let Σ be an *alphabet*, which is an ordered finite set of symbols. An element $T = t_1 \cdots t_n$ of Σ^* is called a *string*, where $|T| = n$ denotes its length. We denote the empty string by ϵ which is the string of length 0, namely, $|\epsilon| = 0$. Let $\Sigma^+ = \Sigma^* \backslash \{\epsilon\}$. A string is also called a *text*. Let $T = t_1 \cdots t_n \in \Sigma^n$ be any text of length n. If $T = usw$ with $u, s, w \in \Sigma^*$, then s is called a *substring* of T. Then, for any $1 \leq i \leq j \leq n$, let $T[i..j] = t_i \cdots t_j$ denote the substring of T that begins and ends at positions i and j in T, and let $T[i] = t_i$ denote the ith symbol of T. For a finite set S of texts, text T is said a *superstring* of S, if T contains all texts of S as substrings. We call the number of occurrences of s in a text as a substring, the *frequency* of s, and denote it by $\#\mathrm{occ}(s)$. Texts Σ^* and $\hat{\Sigma}^*$ are said to be isomorphic for alphabet Σ and $\hat{\Sigma}$, if there exists an isomorphism from Σ to $\hat{\Sigma}$.

2.2 Maximal repeats

Let s be a substring of text T. If the frequency of s is greater than 1, s is called a *repeat*. A *left* (or *right*) *extension* of s is any substring of T with the form ws (or sw), where $w \in \Sigma^*$. We say that s is *left* (or *right*) *maximal* if left (or right) extensions of s occur strictly fewer times in T than s, and call s a *maximal repeat* of T if s is left and right maximal. We call s a *maximal repeatg* of T if both left- and right-extensions of s occur strictly fewer times in T than s. In this thesis, we consider only such strings with length more than 1 as maximal repeats. For example, substring `abra` of $T =$`abracadabra` is a maximal repeat, while `br` is not.

2.3 Grammar compression

A *context free grammar* (CFG or *grammar*, simply) G is defined as a 4-tuple $G = \{V, \Sigma, S, R\}$, where V is an ordered finite set of *variables*, Σ is an ordered finite alphabet, R is a finite set of binary relations called *production rules* (or *rules*) between V and $(V \cup \Sigma)^*$, and $S \in V$ is a special variable called *start variable*. A production rule represents the manner in which a variable is substituted and written in a form $v \rightarrow w$ with $v \in V$ and $w \in (V \cup \Sigma)^*$. Let $X, Y \in V \cup \Sigma^*$. If there are $x_l, x, x_r, y \in (V \cup \Sigma)^*$ such that $X = x_l x x_r$, $Y = x_l y x_r$, and $x \rightarrow y \in R$, we write $X \Rightarrow Y$, and denote the reflexive transitive closure

	a	b	r	a	c	a	d	a	b	r	a
$v_\alpha \to \alpha$ ($\alpha = \mathtt{a},\mathtt{b},\mathtt{r},\mathtt{c},\mathtt{d}$)	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$
$v_1 \to v_\mathtt{a}v_\mathtt{b}$	v_1		$v_\mathtt{r}$	$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_1		$v_\mathtt{r}$	$v_\mathtt{a}$
$v_2 \to v_1 v_\mathtt{r}$	v_2			$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_2			$v_\mathtt{a}$
$v_3 \to v_2 v_\mathtt{a}$	v_3				$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_3			
$S \to v_3 v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_3$	S										

Figure 1: An example of the grammar generation process of RePair for text `abracadabra`. The generated grammar is $\{\{v_\mathtt{a}, v_\mathtt{b}, v_\mathtt{r}, v_\mathtt{c}, v_\mathtt{d}, v_1, v_2, v_3, S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}\}, S, \{v_\mathtt{a} \to \mathtt{a}, v_\mathtt{b} \to \mathtt{b}, v_\mathtt{r} \to \mathtt{r}, v_\mathtt{c} \to \mathtt{c}, v_\mathtt{d} \to \mathtt{d}, v_1 \to v_\mathtt{a}v_\mathtt{b}, v_2 \to v_1v_\mathtt{c}, v_3 \to v_2v_\mathtt{d}, S \to v_3v_\mathtt{c}v_\mathtt{a}v_\mathtt{d}v_3\}\}$, and the grammar size is 16.

of $\Rightarrow$ by $\overset{*}{\Rightarrow}$. Let $val(v)$ be the string derived from v, i.e., $v \overset{*}{\Rightarrow} val(v)$, and let $[\![w]\!]$ denote a variable that derives w, i.e. $[\![w]\!] \overset{*}{\Rightarrow} w$. Note that $[\![w]\!]$ is not necessarily unique. We define grammar $\hat{G} = \{\hat{V}, \hat{\Sigma}, \hat{S}, \hat{R}\}$ as a *subgrammar* of G if $\hat{V} \subseteq V$, $\hat{\Sigma} \subseteq V \cup \Sigma$, and $\hat{R} \subseteq R$.

Given a text T, *grammar compression* is a method of lossless text data compression that constructs a restricted CFG, which uniquely derives the given text T. For G to be deterministic, a production rule for each variable $v \in V$ must be unique. In what follows, we assume that every grammar is deterministic and each production rule is $v_i \to expr_i$, where $expr_i$ is an expression either $expr_i = a$ ($a \in \Sigma$) or $expr_i = v_{j_1}v_{j_2}\cdots v_{j_n}$ ($i > j_k$ for all $1 \le k \le j_n$).

We estimate the effectiveness of compression by the size of generated grammar, which is counted by the total length of the right-hand-side of all production rules of the generated grammar.

2.4 RePair

RePair is a grammar compression algorithm proposed by Larsson and Moffat [11]. For input text T, let $G = \{V, \Sigma, S, R\}$ be the grammar generated by RePair. RePair constructs G by the following steps:

Step 1. Replace each symbol $a \in \Sigma$ with a new variable v_a and add $v_a \to a$ to R.
Step 2. Find the most frequent pair p in T.
Step 3. Replace every occurrence (or, as many occurrences as possible, when p is a pair consisting of the same symbol) of p with a new variable v, then add $v \to p$ to R.
Step 4. Re-evaluate the frequencies of pairs for the renewed text generated in **Step 3**. If the maximum frequency is 1, add $S \to$ (current text) to R, and terminate. Otherwise, return to **Step 2**.

Figure 1 shows an example of the grammar generation process of RePair.

Lemma 1 ([11]). *RePair works in $\mathcal{O}(n)$ expected time and $5n + 4k^2 + 4k' + \lceil\sqrt{n+1}\rceil - 1$ words of space, where n is the length of the source text, k is the cardinality of the source alphabet, and k' is the cardinality of the final dictionary.*

3 Analyzing RePair

In this section, we analyze RePair with regard to its relationship to maximal repeats, and introduce an important concept, called MR-order.

3.1 RePair and maximal repeats

The following theorem shows an essential property of RePair. That is, RePair recursively replaces the most frequent maximal repeats.

Theorem 1. *Let T be a given text, and assume that every most frequent maximal repeat of T does not appear with overlaps with itself. Let f be the frequency of the most frequent pairs of T, and t be a text obtained after all pairs with frequency f in T are replaced by variables. Then, there is a text s such that s is obtained after all maximal repeats with frequency f in T are replaced by variables, and s and t are isomorphic to each other.*

We need two lemmas and a corollary to prove Theorem 1. The following lemma shows a fundamental relation between the most frequent maximal repeats and the most frequent pairs in a text.

Lemma 2. *A pair p of variables is most frequent in text T if and only if p occurs once in exactly one of the most frequent maximal repeats of T.*

Proof. ($\Rightarrow$) Let r be a most frequent maximal repeat that contains p as a substring. It is clear that p can only occur once in r, since otherwise, $\#\mathrm{occ}(p) > \#\mathrm{occ}(r)$ would hold, implying the existence of a frequent maximal repeat that is more frequent than r, contradicting the assumption that r is most frequent. Suppose there exists a different most frequent maximal repeat r' that contains p as a substring. Similarly, p occurs only once in r'. Furthermore, since r and r' can be obtained by left and right extensions to p, $\#\mathrm{occ}(r) = \#\mathrm{occ}(r') = \#\mathrm{occ}(p)$, and any occurrence of p is contained in an occurrence of both r and r'. Since r' cannot be a substring of r, there exists some string w that is a superstring of r and r', such that $\#\mathrm{occ}(w) = \#\mathrm{occ}(r) = \#\mathrm{occ}(r') = \#\mathrm{occ}(p)$. However, this contradicts that r and r' are maximal repeats.

($\Leftarrow$) Let r be the most frequent maximal repeat such that p occurs once in. By definition, $\#\mathrm{occ}(r) = \#\mathrm{occ}(p)$. If p is not the most frequent symbol pair in T, there exists some symbol pair p' in T such that $\#\mathrm{occ}(p') > \#\mathrm{occ}(p) = \#\mathrm{occ}(r)$. However, this implies that there is a maximal repeat r' with $\#\mathrm{occ}(r') = \#\mathrm{occ}(p') > \#\mathrm{occ}(r)$, contradicting that r is most frequent. $\qquad\square$

The following corollary is directly derived from Lemma 2.

Corollary 1. *For a given text, the frequency of the most frequent pairs and that of the most frequent maximal repeats are the same.*

The following lemma shows an important property of the most frequent maximal repeats.

Lemma 3. *The length of overlap between any two occurrences of most frequent maximal repeats is at most 1.*

Proof. Let xw and wy be most frequent maximal repeats that have an overlapping occurrence xwy, where $x, y, w \in \Sigma^+$. If $|w| \geq 2$, then, since xw and wy are most frequent maximal repeats, it must be that $\#\mathrm{occ}(w) = \#\mathrm{occ}(xw) = \#\mathrm{occ}(wy)$, i.e., every occurrence of w is preceded by x and followed by y. This implies that $\#\mathrm{occ}(xwy) = \#\mathrm{occ}(xw) = \#\mathrm{occ}(wy)$ as well, but contradicts that xw and wy are maximal repeats. $\qquad\square$

From the above lemmas and a corollary, now we can prove Theorem 1.

Proof of Theorem 1. By Corollary 1, the frequency of the most frequent maximal repeats in T is f. Let p be one of the most frequent pairs in T. By Lemma 2, there is a unique maximal repeat that is most frequent and contains p once. We denote such maximal repeat by r. Assume that there is a substring $zxpyw$ in T, where $z, w \in \Sigma$, $x, y \in \Sigma^*$, and $xpy = r$. We denote $r[1]$ and $r[|r|]$ by $\dot{x}$ and $\dot{y}$, respectively. There are 2 cases to consider:

(i) $\#\mathrm{occ}(z\dot{x}) < f$ **and** $\#\mathrm{occ}(\dot{y}w) < f$**.** If $|r| = 2$, the replacement of p directly corresponds to the replacement of the most frequent maximal repeat, since $p = r$. If $|r| > 2$, after p is replaced with a variable v, r is changed to xvy. This occurs f times in the renewed text, and by Lemma 2, the frequency of every pair occurring in xvy is still f. Because the maximum frequency of pairs does not increase, f is still the maximum frequency. Therefore, we replace all pairs contained in xvy in the following steps,

4

and $z\dot{x}$ and $\dot{y}w$ are not replaced. This holds for every occurrence of p, implying that replacing the most frequent pairs while the maximum frequency does not change, corresponds to replacing all pairs contained (old and new) in most frequent maximal repeats of the same frequency until they are replaced by a single variable. Then, we can generate s by replacement of r.

(ii) $\#\mathrm{occ}(z\dot{x}) = f$ **or** $\#\mathrm{occ}(\dot{y}w) = f$**.** We consider the case where $\#\mathrm{occ}(z\dot{x}) = f$. Note that $\#\mathrm{occ}(zxpy) < f$ by assumption that xpy is a maximal repeat. Suppose RePair replaces $z\dot{x}$ by a variable v before p is replaced. Note that by Lemma 2, there is a maximal repeat occurring f times and including $z\dot{x}$ once (we denote the maximal repeat by r'), and $r' \neq r$ by assumption. By Lemma 3, the length of overlap of r and r' is at most 1, then only $\dot{x}$ is a symbol contained both r and r'. After that, $xpy = r$ is no longer the most frequent maximal repeat because some of its occurrences are changed to $vr[2..|r|]$. However, $r[2..|r|]$ still occurs f times in the renewed text. Since $\#\mathrm{occ}(zxpy) < f$ and $\#\mathrm{occ}(xpy) = f$, $\#\mathrm{occ}(vr[2]) < f$ and $r[2..|r|]$ is a maximal repeat. Then, $r[2..|r|]$ will become a variable in subsequent steps, similarly to **(i)**. Here, r' would also become a variable. Thus, we can generate s in the way that we replace r' first, then we replace $r[2..|r|]$. This holds similarly for $\dot{y}w$ when $\#\mathrm{occ}(\dot{y}w) = f$, and when $\#\mathrm{occ}(z\dot{x}) = \#\mathrm{occ}(\dot{y}w) = f$. $\qquad\square$

3.2 MR-order

From Theorem 1, if the most frequent maximal repeat is unique in the current text, then all the occurrences of it are replaced step by step by RePair. However, it is a problem if there are two or more most frequent maximal repeats and some of them overlap. In this case, which maximal repeat is first summarized up depends on the order in which the most frequent pairs are selected. Note, however, if there are multiple most frequent pairs, which pair is first replaced depends on the implementation of RePair. We call this order of selecting (summarizing) maximal repeats *maximal repeat selection order* (or *MR-order*, simply).

For instance, consider a text `abcdeabccde`. `abc` and `cde` are the most frequent maximal repeats occurring 2 times. There are 2 MR-order, depending on which one is attached priority to the other. The results after replacement by RePair with the MR-order are (i) $xyxcx$ with variables x and y such that $x \overset{*}{\Rightarrow} \mathtt{abc}$ and $y \overset{*}{\Rightarrow} \mathtt{de}$, and (ii) $zwzcw$ with variables z and w such that $z \overset{*}{\Rightarrow} \mathtt{ab}$ and $w \overset{*}{\Rightarrow} \mathtt{cde}$. More precisely, there are 12 possible ways in which RePair can compress the text, and the generated rule sets are:

1. $\{v_1 \to \mathtt{ab}, v_2 \to v_1\mathtt{c}, v_3 \to \mathtt{de}, S \to v_2v_3v_2\mathtt{c}v_3\}$,

2. $\{v_1 \to \mathtt{ab}, v_2 \to \mathtt{de}, v_3 \to v_1\mathtt{c}, S \to v_3v_2v_3\mathtt{c}v_2\}$,

3. $\{v_1 \to \mathtt{bc}, v_2 \to \mathtt{a}v_1, v_3 \to \mathtt{de}, S \to v_2v_3v_2\mathtt{c}v_3\}$,

4. $\{v_1 \to \mathtt{bc}, v_2 \to \mathtt{de}, v_3 \to \mathtt{a}v_1, S \to v_3v_2v_3\mathtt{c}v_2\}$,

5. $\{v_1 \to \mathtt{ed}, v_2 \to \mathtt{ab}, v_3 \to v_2\mathtt{c}, S \to v_3v_1v_3\mathtt{c}v_1\}$,

6. $\{v_1 \to \mathtt{ed}, v_2 \to \mathtt{bc}, v_3 \to \mathtt{a}v_2, S \to v_3v_1v_3\mathtt{c}v_1\}$,

7. $\{v_1 \to \mathtt{ab}, v_2 \to \mathtt{cd}, v_3 \to v_2\mathtt{e}, S \to v_1v_3v_1\mathtt{c}v_3\}$,

8. $\{v_1 \to \mathtt{ab}, v_2 \to \mathtt{de}, v_3 \to \mathtt{c}v_2, S \to v_1v_3v_1\mathtt{c}v_3\}$,

9. $\{v_1 \to \mathtt{cd}, v_2 \to \mathtt{ab}, v_3 \to v_1\mathtt{e}, S \to v_2v_3v_2\mathtt{c}v_3\}$,

10. $\{v_1 \to \mathtt{cd}, v_2 \to v_1\mathtt{e}, v_3 \to \mathtt{ab}, S \to v_3v_2v_3\mathtt{c}v_2\}$,

11. $\{v_1 \to \mathtt{ed}, v_2 \to \mathtt{ab}, v_3 \to \mathtt{c}v_1, S \to v_2v_3v_2\mathtt{c}v_3\}$,

12. $\{v_1 \rightarrow \texttt{ed}, v_2 \rightarrow \texttt{c}v_1, v_3 \rightarrow \texttt{ab}, S \rightarrow v_3 v_2 v_3 \texttt{c} v_2\}$.

Here, 1 - 6 have the same MR-order, because $\texttt{abc}$ is prior to $\texttt{cde}$ in all of them. On the other hand, 7 - 12 have the same MR-order for similar reason; $\texttt{cde}$ is prior to $\texttt{abc}$.

The size of the grammar generated by RePair varies according to how to select a pair when there are several distinct most frequent pairs that overlap. For instance, consider the text $\texttt{bcxdabcyabzdabvbcuda}$. There are 3 most frequent pairs, $\texttt{ab}$, $\texttt{bc}$, and $\texttt{da}$ with 3 occurrences. If RePair takes $\texttt{ab}$ first, the rule set of generated grammar may become $\{v_1 \rightarrow \texttt{ab},\ v_2 \rightarrow \texttt{bc},\ v_3 \rightarrow \texttt{d}v_1,\ S \rightarrow v_2 \texttt{x} v_3 \texttt{cy} v_1 \texttt{z} v_3 \texttt{vv}_2 \texttt{uda}\}$ and the size of it is 19. On the other hand, if RePair takes $\texttt{da}$ first, the rule set of generated grammar may become $\{v_1 \rightarrow \texttt{da},\ v_2 \rightarrow \texttt{bc},\ S \rightarrow v_2 \texttt{x} v_1 v_2 \texttt{yabz} v_1 \texttt{bv} v_2 \texttt{u} v_1\}$ and the size of it is 18.

Remark 1. *If there are several distinct pairs with the same maximum frequency, the size of the grammar generated by RePair depends on the replacement order of them.*

However, the following theorem states that MR-order rather than the replacement order of pairs is essentially important for the size of the grammar generated by RePair.

Theorem 2. *The sizes of the grammars generated by RePair are the same if they are generated in the same MR-order.*

Proof. Let T be a variable sequence appearing in the grammar generation process of RePair, and f be the maximum frequency of pairs in T. Suppose that T' is a variable sequence generated after RePair replaces every pair occurring f times. By Theorem 1, all generated T' are isomorphic to one another, then the length of all of them is the same, regardless of the replacement order of pairs. Let r_1 be the most frequent maximal repeats of T such that r_1 is prior to all other ones in this MR-order. r_1 is converted into a variable as a result, and by Lemma 2, all pairs included in r_1 are distinct. Then, the size of the subgrammar which exactly derives r_1 is $2(|r_1| - 1) + 1 = 2|r_1| - 1$. This holds for the next prioritized maximal repeat (we denote it by r_2) with a little difference; the pattern actually replaced would be a substring of r_2 excluding the beginning or the end of it, if there are occurrences of overlap with r_1. However, these strings are common in the same MR-order, then the sizes of generated subgrammars are the same, regardless of the selecting order of pairs. This similarly holds for all of the most frequent maximal repeats, for every maximum frequency of pairs, through the whole process of RePair. $\square$

3.3 Greatest size difference of RePair

We consider the problem of how large the difference between possible outcomes of RePair can be.

Definition 1 (Greatest Size Difference). *Let g and g' be sizes of any two possible grammars that can be generated by RePair for a given text. Then, the Greatest Size Difference of RePair (GSDRP) is $\max(|g - g'|)$.*

We show a lower bound of above GSDRP in the following theorem.

Theorem 3. *Given a text with length n, a lower bound of GSDRP is $\frac{1}{6}(\sqrt{6n + 1} + 13)$.*

Proof. Let B, L, and R be strings such that

$$B = l_1 x y r_1 l_2 x y r_2 \cdots l_{f-1} x y r_{f-1} l_f x y r_f,$$
$$L = \Diamond l_1 x \Diamond l_2 x \cdots \Diamond l_f x,$$
$$R = \Diamond y r_1 \Diamond y r_2 \cdots \Diamond y r_f,$$

where $x, y, l_1, \ldots, l_f, r_1, \ldots, r_f$ denote distinct symbols, and each occurrence of $\Diamond$ denotes a distinct symbol. Consider text $T = B L^{f-1} R^{f-1}$. Here, xy, $l_1 x$, $\cdots$, $l_f x$, $y r_1$, $\cdots$, $y r_f$ are the most frequent

maximal repeats with frequency f in T. Let G and G' be grammars generated by RePair for T in different MR-order, such that (i) xy is prior to all other maximal repeats, and (ii) xy is posterior to all other maximal repeats, respectively. We denote the sizes of G and G' by g and g', respectively.

First, we consider G and how RePair generates it. The first rule generated by replacement is $v_1 \rightarrow xy$ because of the MR-order. After replacement, L and R is unchanged but B becomes the following text:

$$B_1 = l_1 v_1 r_1 l_2 v_1 r_2 \cdots l_{f-1} v_1 r_{f-1} l_f v_1 r_f.$$

Each pair in B_1 occurs only once in the whole text $B_1 L^{f-1} R^{f-1}$. This means that B_1 is never be shortened from the current length, $3f$. In the remaining steps, each $l_i x$ and $y r_i$ (for $i = 1, \cdots, f$) is replaced, and that is all. L and R changed to texts whose length are both $2f$. Hence, the following holds:

$$g = 3f + 2 \cdot 2f + 2(1 + 2f) = 11f + 2. \tag{1}$$

Next, we consider G' and how RePair generates it. By its MR-order, $l_1 x, \cdots, l_f x, y r_1, \cdots, y r_f$ are replaced before xy is selected. They do not overlap with each other, and after they are replaced, xy does not occur in the generated text. Therefore, in G', there are $2f$ rules which derive each $l_i x$ and $y r_i$ (for $i = 1, \cdots, f$), respectively, but the rule which derives xy is absent. L and R changed to texts whose length are both $2f$, and B changed to a text with length $2f$. Hence, the following holds:

$$g' = 2f + 2 \cdot 2f + 2 \cdot 2f = 10f. \tag{2}$$

Let us denote the length of the original text $T = BL^{f-1}R^{f-1}$ by n. Then, the following holds:

$$n = 4f + 2(3f)(f - 1) = 6f^2 - 2f.$$

Therefore,

$$f = \frac{1}{6}(\sqrt{6n + 1} + 1) \tag{3}$$

holds. By (1), (2), and (3),

$$g - g' = 11f + 2 - 10f = f + 2$$
$$= \frac{1}{6}(\sqrt{6n + 1} + 13)$$

holds and the theorem follows. $\qquad\square$

4 MR-RePair

The main strategy of our proposed method is to recursively replace the most frequent maximal repeats, instead of the most frequent pairs.

In this section, first, we explain the naïve version of our method called Naïve-MR-RePair. While it can have bad performance in specific cases, it is simple and helpful to understand our main result. Then, we describe our proposed MR-RePair.

4.1 Naïve-MR-RePair

Definition 2 (Naïve-MR-RePair)**.** *For input text T, let $G = \{V, \Sigma, S, R\}$ be the grammar generated by Naïve-MR-RePair. Naïve-MR-RePair constructs G by the following steps:*

	a	b	r	a	c	a	d	a	b	r	a
$v_\alpha \to \alpha$ ($\alpha = \mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}$)	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$
$v_1 \to v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a}$	v_1				$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_1			
$S \to v_1 v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_1$	S										

Figure 2: An example of the grammar generation process of Naïve-MR-RePair for text `abracadabra`. The generated grammar is $\{\{v_\mathtt{a}, v_\mathtt{b}, v_\mathtt{r}, v_\mathtt{c}, v_\mathtt{d}, v_1, S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}\}, S, \{v_\mathtt{a} \to \mathtt{a}, v_\mathtt{b} \to \mathtt{b}, v_\mathtt{r} \to \mathtt{r}, v_\mathtt{c} \to \mathtt{c}, v_\mathtt{d} \to \mathtt{d}, v_1 \to v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a}, S \to v_1 v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_1\}\}$, and the grammar size is 14.

Step 1. *Replace each symbol $a \in \Sigma$ with a new variable v_a and add $v_a \to a$ to R.*
Step 2. *Find the most frequent maximal repeat r in T.*
Step 3. *Replace every occurrence (or, as many occurrences as possible, when there are overlaps) of r in T with a new variable v, and then add $v \to r$ to R.*
Step 4. *Re-evaluate the frequencies of maximal repeats for the renewed text generated in **Step 3**. If the maximum frequency is 1, add $S \to$ (current text) to R, and terminate. Otherwise, return to **Step 2**.*

We can easily extend the concept of MR-order to this Naïve-MR-RePair.

We show an example of the grammar generation process of Naïve-MR-RePair in Figure 2. Figure 1 and 2 illustrates the intuitive reason why the strategy using maximal repeats is effective compared to that using pairs. When compressing text $v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a} v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a}$, RePair and Naïve-MR-RePair both generate subgrammars which derive the most frequent maximal repeat $v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a}$. The rule set of the subgrammar of RePair is $\{v_1 \to v_\mathtt{a} v_\mathtt{b}, v_2 \to v_1 v_\mathtt{r}, v_3 \to v_2 v_\mathtt{a}\}$, and the size is 6. On the other hand, the rule set of subgrammar of Naïve-MR-RePair is $\{v_1 \to v_\mathtt{a} v_\mathtt{b} v_\mathtt{r} v_\mathtt{a}\}$, and the size is 4.

However, the following theorem indicates that the size of the grammar generated by Naïve-MR-RePair is larger than that by RePair in particular cases, even when they work in the same MR-order.

Theorem 4. *Given a text T with length n, and assume that RePair and Naïve-MR-RePair work in the same MR-order. Let g_{rp} and g_{nmr} be sizes of grammars generated by RePair and Naïve-MR-RePair for T, respectively. Then, there is a case where $g_{nmr} = g_{rp} + \mathcal{O}(\log n)$ holds.*[1]

Proof. Assume that $G_{rp} = \{V_{rp}, \Sigma_{rp}, S_{rp}, R_{rp}\}$ and $G_{nmr} = \{V_{nmr}, \Sigma_{nmr}, S_{nmr}, R_{nmr}\}$ are grammars generated by RePair and Naïve-MR-RePair, respectively. Let $T' = v_1 \cdots v_n$ such that $v_i \in V_{rp} \cap V_{nmr}$ and $v_i \to T[i] \in R_{rp} \cap R_{nmr}$ (for $i = 1, \cdots, n$), and $\hat{G}_{rp} = \{\hat{V}_{rp}, \hat{\Sigma}_{rp}, \hat{S}_{rp}, \hat{R}_{rp}\}$ (or $\hat{G}_{nmr} = \{\hat{V}_{nmr}, \hat{\Sigma}_{nmr}, \hat{S}_{nmr}, \hat{R}_{nmr}\}$) be a subgrammar of G_{rp} (or G_{nmr}) which derives T'. Assume that $T' = (uw)^{2^{m+1}-1}u$, where $u \in V_{rp} \cap V_{nmr}$, $w \in (V_{rp} \cap V_{nmr})^+$ such that uwu is the most frequent maximal repeat of T', and $m \in \mathbb{N}^+$. Note that $2^{m+1} - 1 = \sum_{i=0}^{m} 2^i$. Here $\hat{R}_{rp}$ and $\hat{R}_{nmr}$ consist as follows:

$\hat{R}_{rp}$: Assume that $x_i \in \hat{V}_{rp}$ for $1 \leq i \leq m$ and $y_j \in \hat{V}_{rp} \cup \hat{\Sigma}_{rp}$ for $1 \leq j \leq |w|$, then

- $|w|$ rules $y_j \to y_l y_r$ with $val(y_{|w|}) = uw$.
- One rule $x_1 \to y_{|w|} y_{|w|}$ and $\log_2 \lfloor 2^{m+1} - 1 \rfloor - 1 = m - 1$ rules $x_i \to x_{i-1} x_{i-1}$ for $2 \leq i \leq m$.
- One rule $\hat{S}_{rp} \to x_m x_{m-1} \cdots x_1 y_{|w|}$.

$\hat{R}_{nmr}$: Assume that $d = |\hat{V}_{nmr}| = |\hat{R}_{nmr}|$ and $z_i \in \hat{V}_{nmr}$ for $1 \leq i \leq d$, then

- One rule $z_1 \to uwu$.
- $d - 1$ rules $z_i \to z_{i-1} w z_{i-1}$ for $2 \leq i \leq d$ and $z_d = \hat{S}_{nmr}$.

---
[1] We show a concrete example of this theorem in Appendix.

8

	a	b	r	a	c	a	d	a	b	r	a
$v_\alpha \to \alpha$ ($\alpha = \mathtt{a,b,r,c,d}$)	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	$v_\mathtt{a}$	$v_\mathtt{b}$	$v_\mathtt{r}$	$v_\mathtt{a}$
$v_1 \to v_\mathtt{a} v_\mathtt{b} v_\mathtt{r}$	v_1			$v_\mathtt{a}$	$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_1			$v_\mathtt{a}$
$v_2 \to v_1 v_\mathtt{a}$	v_2				$v_\mathtt{c}$	$v_\mathtt{a}$	$v_\mathtt{d}$	v_2			
$S \to v_2 v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_2$	S										

Figure 3: An example of the grammar generation process of MR-RePair for text `abracadabra`. The generated grammar is $\{\{v_\mathtt{a}, v_\mathtt{b}, v_\mathtt{r}, v_\mathtt{c}, v_\mathtt{d}, v_1, S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}\}, S, \{v_\mathtt{a} \to \mathtt{a}, v_\mathtt{b} \to \mathtt{b}, v_\mathtt{r} \to \mathtt{r}, v_\mathtt{c} \to \mathtt{c}, v_\mathtt{d} \to \mathtt{d}, v_1 \to v_\mathtt{a} v_\mathtt{b} v_\mathtt{r}, v_2 \to v_1 v_\mathtt{a}, S \to v_2 v_\mathtt{c} v_\mathtt{a} v_\mathtt{d} v_2\}\}$, and the grammar size is 15.

Let $\hat{g}_{rp}$ and $\hat{g}_{nmr}$ be sizes of $\hat{G}_{rp}$ and $\hat{G}_{nmr}$, respectively. Then, the following holds:

$$\hat{g}_{rp} = 2|w| + 2m + (m+2) = 3m + 2|w| + 2 \qquad (4)$$
$$\hat{g}_{nmr} = |w| + 2 + (|w| + 2)(d - 1) = (|w| + 2)d \qquad (5)$$

Here, with regard to the length of T', the following holds:

$$(2^d - 1)|w| + 2^d = n = (2(2^m - 1) + 1)(|w| + 1) + 1.$$

Since the right-side results in $2^{m+1}|w| + 2^{m+1}$, $d = m + 1$ follows it. Hence, by equation (4) and (5), the following holds:
$$\hat{g}_{nmr} - \hat{g}_{rp} = (m - 1)(|w| - 1) - 1.$$

Therefore, $\hat{g}_{nmr} > \hat{g}_{rp}$ holds for some $(m, |w|)$, and the proposition holds. $\qquad \square$

4.2 MR-RePair

The reason why the grammar size of Naïve-MR-RePair becomes larger than that of RePair as seen in Theorem 4 is that Naïve-MR-RePair cannot replace all occurrences of the most frequent maximal repeats if it overlaps with another occurrence of itself. In the remainder of this section, we describe MR-RePair, which is an improved version of the above Naïve-MR-RePair.

Definition 3 (MR-RePair). *For input text T, let $G = \{V, \Sigma, S, R\}$ be the grammar generated by MR-RePair. MR-RePair constructs T by the following steps:*
Step 1. *Replace each symbol $a \in \Sigma$ with a new variable v_a and add $v_a \to a$ to R.*
Step 2. *Find the most frequent maximal repeat r in T.*
Step 3. *Check if $|r| > 2$ and $r[1] = r[|r|]$, and if so, replace r with $r[2..|r|]$.*
Step 4. *Replace every occurrence of r with a new variable v, then add $v \to r$ to R.*
Step 5. *Re-evaluate the frequencies of maximal repeats for the renewed text generated in* **Step 4**. *If the maximum frequency is 1, add $S \to$ (current text) to R, and terminate. Otherwise, return to* **Step 2**.

We show an example of the grammar generation process of MR-RePair in Figure 3. We can easily extend the concept of MR-order to this MR-RePair. We do not care if it uses $r[1..|r - 1|]$ in **Step 3**, instead of $r[2..|r|]$. MR-RePair can replace all occurrences of r even if it overlaps with itself in some occurrences, since by Lemma 3, the length of overlaps of the most frequent maximal repeats is at most 1. If $r[1] = r[|r|]$ but r does not overlap with itself, then $r[1]v$ becomes the most frequent maximal repeat after $r[2..|r|]$ is replaced by v, and $r[1]v$ would be replaced immediately. MR-RePair still cannot replace all of them if $|r| = 2$, but the same is said to RePair.

We show an example of the grammar generation process of MR-RePair in Figure 3. Although the size of the generated grammar in Figure 3 is larger than that of Naïve-MR-RePair shown in Figure 2, it is still smaller than that of RePair shown in Figure 1.

Theorem 5. *Assume that RePair and MR-RePair work based on the same MR-order for a given text. Let g_{rp} and g_{mr} be sizes of grammars generated by RePair and MR-RePair, respectively. Then, $\frac{1}{2}g_{rp} < g_{mr} \le g_{rp}$ holds.*

Proof. Assume that $G_{rp} = \{V_{rp}, \Sigma_{rp}, S_{rp}, R_{rp}\}$ and $G_{mr} = \{V_{mr}, \Sigma_{mr}, S_{mr}, R_{mr}\}$ are grammars generated by RePair and MR-RePair, respectively, for a given text T with length n. Let $T' = v_1 \cdots v_n$ such that $v_i \in V_{rp} \cap V_{mr}$ and $v_i \to T[i] \in R_{rp} \cap R_{mr}$ (for $i = 1, \cdots, n$).

We start with T'. Let f_1 be the maximum frequency of maximal repeats in T'. By Corollary 1, the maximum frequency of pairs in T' is also f_1. Let $\hat{G}_{rp}^{(f_1)}$ (or $\hat{G}_{mr}^{(f_1)}$) be a subgrammar of G_{rp} (or G_{mr}) which is generated while RePair (or MR-RePair) replaces pairs (or maximal repeats) with frequency f_1, $\hat{g}_{rp}^{(f_1)}$ (or $\hat{g}_{mr}^{(f_1)}$) be the size of it, and $T_{rp}^{(f_1)}$ (or $T_{mr}^{(f_1)}$) be the renewed text after all pairs (or maximal repeats) with frequency f_1 are replaced. Let $r_1^{(f_1)}, \cdots, r_{m_1}^{(f_1)}$ be maximal repeats with frequency f_1 in T', and assume that they are prioritized in this order by the MR-order. Let each $l_i^{(f_1)}$ (for $i = 1, \cdots, m_1$) be the length of the longest substring of $r_i^{(f_1)}$ such that there are variables that derive the substring in both $\hat{G}_{rp}^{(f_1)}$ and $\hat{G}_{mr}^{(f_1)}$. Note that this substring is common to RePair and MR-RePair, and each $l_i^{(f_1)}$ is at least 2. Then, by Lemma 2, the following holds:

$$\hat{g}_{rp}^{(f_1)} = \sum_{i=1}^{m_1} 2(l_i^{(f_1)} - 1) \, ,$$

$$\hat{g}_{mr}^{(f_1)} = \sum_{i=1}^{m_1} l_i^{(f_1)}.$$

Therefore,

$$\therefore \ \frac{1}{2}\hat{g}_{rp}^{(f_1)} < \hat{g}_{mr}^{(f_1)} \le \hat{g}_{rp}^{(f_1)} \tag{6}$$

holds. The renewed texts $T_{rp}^{(f_1)}$ and $T_{mr}^{(f_1)}$ are isomorphic for V_{rp} and V_{mr}. Let f_2 be the maximum frequency of maximal repeats in $T_{rp}^{(f_1)}$ (and this is the same in $T_{mr}^{(f_1)}$). Then, the similar discussion holds for $\hat{G}_{rp}^{(f_2)}$ and $\hat{G}_{mr}^{(f_2)}$. Hence, $\frac{1}{2}\hat{g}_{rp}^{(f_2)} < \hat{g}_{mr}^{(f_2)} \le \hat{g}_{rp}^{(f_2)}$ holds similarly to (6), and the renewed texts $T_{rp}^{(f_2)}$ and $T_{mr}^{(f_2)}$ are isomorphic. Inductively, for every maximum frequency of maximal repeats f_i, $\frac{1}{2}\hat{g}_{rp}^{(f_i)} < \hat{g}_{mr}^{(f_i)} \le \hat{g}_{rp}^{(f_i)}$ holds and the renewed texts $T_{rp}^{(f_i)}$ and $T_{mr}^{(f_i)}$ are isomorphic. Let k be a natural number such that $f_k > 1$ and $f_{k+1} = 1$, which is the number of decreasing of maximum frequency through the whole process of RePair and MR-RePair. Then,

$$g_{rp} = \sum_{j=1}^{k} \hat{g}_{rp}^{(f_j)} + |\Sigma| + |T_{rp}^{(f_k)}| = \sum_{j=1}^{k}\sum_{i=1}^{m_j} 2(l_i^{(f_j)} - 1) + |\Sigma| + |T_{rp}^{(f_k)}| \, , \tag{7}$$

$$g_{mr} = \sum_{j=1}^{k} \hat{g}_{mr}^{(f_j)} + |\Sigma| + |T_{mr}^{(f_k)}| = \sum_{j=1}^{k}\sum_{i=1}^{m_j} l_i^{(f_j)} + |\Sigma| + |T_{mr}^{(f_k)}| \tag{8}$$

holds. Because every $l_i^{(f_j)} \ge 2$ and $|T_{rp}^{(f_k)}| = |T_{mr}^{(f_k)}|$, $\frac{1}{2}g_{rp} < g_{mr} \le g_{rp}$ follows (7) and (8) and the proposition holds. $g_{mr} = g_{rp}$ holds when every length of $l_i^{(f_j)}$ is 2. $\square$

The following theorem shows that unless the MR-order of RePair and MR-RePair are the same, the grammar generated by MR-RePair might be larger than that by RePair.

Theorem 6. *Unless the MR-order of RePair and MR-RePair are the same, there is a case where the size of the generated grammar by MR-RePair becomes larger than that by RePair.*

Proof. As shown in Theorem 5, the size of grammar generated by MR-RePair would be strictly equal to that by RePair with the same MR-order. Thus, we can reduce this problem to the problem that there is a difference between sizes of possible grammars generated by RePair as stated in Remark 1. Hence, there are the cases stated in the proposition if the MR-order of MR-RePair matches with a MR-order of RePair which does not generate the smallest RePair grammar. □

We can implement MR-RePair by extending the original implementation of RePair stated in [11], holding the same complexity.

Theorem 7. *Let $G = \{V, \Sigma, S, R\}$ be the generated grammar by MR-RePair for a given text with length n. Then, MR-RePair works in $\mathcal{O}(n)$ expected time and $5n + 4k^2 + 4k' + \lceil \sqrt{n+1} \rceil - 1$ word space, where k and k' are the cardinalities of Σ and V, respectively.*

Proof. Compared with RePair, the additional operations which MR-RePair does in our implementation are (i) it extends the selected pair to left and right until it becomes a maximal repeat, and (ii) it checks and excludes either of the beginning or the end of the obtained maximal repeat if they are the same. They can be realized by only using the same data structures as that of RePair. Then, the space complexity of MR-RePair follows Lemma 1.

We can clearly execute operation (ii) in constant time. So we consider how the time complexity is affected by operation (i). Let l be the length of the maximal repeat containing the focused pair, and f be the frequency of the pair. Then, when MR-RePair checks the left- and right-extensions for all occurrences of the focused pair, $\mathcal{O}(fl)$ excessive time is required compared with RePair. However, the length of entire text is shortened at least $f(l-1)$ by the replacement. Therefore, according to possible counts of replacement through the entire steps of the algorithm, MR-RePair works in $\mathcal{O}(n)$ expected time. □

Remark 2. *We can convert a grammar of RePair to that of MR-RePair by repeating the following transform: If a variable v appears only once on the right-hand side of other rules, remove the rule for v and replace the one occurrence of v with the right-hand side of the removed rule. However, time and space complexity stated in Theorem 7 cannot be achieved in this manner, since additional operations and memory for searching and storing such variables are required.*

5 Experiments

We implemented MR-RePair and measured the number of generated rules and the execution time in order to compare it to existing RePair implementations and Re-PairImp[2] proposed by Gańczorz and Jeż [8].

As stated in Remark 1, the size of a generated grammar depends on the MR-order. In practice, the MR-order varies how we implement the priority queue managing symbol pairs. To see this, we used five RePair implementations in the comparison; they were implemented by Maruyama[3], Navarro[4], Prezza[5] [5], Wan[6], and Yoshida[7].

Table 1 summarizes the details of the texts we used in the comparison. We used three texts as highly repetitive texts; one is a randomly generated text (rand77.txt), and the others are a Fibonacci string (fib41) and a German text (einstein.de.txt) which were selected from Repetitive Corpus of Pizza&Chili Corpus[8]. The randomly generated text, rand77.txt, consists of alphanumeric symbols and some special

---

[2] `https://bitbucket.org/IguanaBen/repairimproved`
[3] `https://code.google.com/archive/p/re-pair/`
[4] `https://www.dcc.uchile.cl/~gnavarro/software/index.html`
[5] `https://github.com/nicolaprezza/Re-Pair`
[6] `https://github.com/rwanwork/Re-Pair`; We ran it with level 0 (no heuristic option).
[7] `https://github.com/syoshid/Re-Pair-VF`; We removed a routine to find the best rule set.
[8] `http://pizzachili.dcc.uchile.cl/repcorpus.html`

Table 1: Text files used in our experiments.

texts	size (bytes)	$\vert\Sigma\vert$	contents
rand77.txt	2,097,152	77	32 copies of 1024 random patterns of length 64
fib41	267,914,296	2	Fibonacci string from Pizza&Chili Corpus
einstein.de.txt	92,758,441	117	Edit history of Wikipedia for Albert Einstein
E.coli	4,638,690	4	Complete genome of the E. Coli bacterium
bible.txt	4,047,392	63	The King James version of the bible
world192.txt	2,473,400	94	The CIA world fact book

symbols; and it is generated by concatenating 32 copies of a block that includes 1024 random patterns of length 64, i.e., the size is $64 \times 1024 \times 32 = 2,097,152$ byte. In addition, we used three texts (E.coli, bible.txt, world192.txt) for real data selected from Large Corpus[9]. We executed each program seven times for each text and measured the elapsed CPU time only for grammar generation process. We calculated the average time of the five results excluding the minimum and maximum values among seven. We ran our experiments on a workstation equipped with an Intel(R) Xeon(R) E5-2670 2.30GHz dual CPU with 64GB RAM, running on Ubuntu 16.04LTS on Windows 10. All the programs are compiled by gcc version 7.3.0 with "-O3" option.

Table 2 lists the experimental results. Here, we excluded the number of rules that generate a single terminal symbol from the number of rules because it is the same between MR-RePair and RePair. As shown in the table, for all texts except for fib41, the size of rules generated by each RePair implementation differs from each other.[10] In any case, MR-RePair is not inferior to RePair in the size of rules. For rand77.txt in particular, the number of rules decreased to about 11% and the size of rules decreased to about 55%. For einstein.de.txt, moreover, the number of rules decreased to about 44% and the size of rules decreased to about 72%. On the other hand, for the texts of Large Corpus, which are not highly repetitive, it turned out that the effect of improvement was limited. Note that fib41 does not contain any maximal repeats longer than 2 without overlaps. Therefore, MR-RePair generates the same rules as RePair. Also note that MR-RePair runs at a speed comparable to the fastest implementation of RePair.

6 Conclusion

In this thesis, we analyzed RePair and showed that RePair replaces step by step the most frequent pairs within the corresponding most frequent maximal repeats. Motivated by this analysis, we designed a novel variant of RePair, called MR-RePair, which is based on substituting the most frequent maximal repeats at once instead of substituting the most frequent pairs consecutively. Moreover, we implemented MR-RePair and compared the grammar generated by it to that by RePair for several texts, and confirmed the effectiveness of MR-RePair experimentally especially for highly repetitive texts.

We defined the greatest size difference of any two possible grammars that can be generated by RePair for a given text, and named it as GSDRP. Moreover, we showed that a lower bound of GSDRP is $\frac{1}{6}(\sqrt{6n+1}+13)$ for a given text of length n. Improving the lower bound and showing a upper bound of GSDRP are our future works.

Although we did not discuss how to encode grammars, it is a very important issue from a practical point of view. For MR-RePair, if we simply use delimiters to store the rule set, the number of rules may drastically affect the compressed data size. To develop an efficient encoding method for MR-RePair is one of our future works.

---

[9] http://corpus.canterbury.ac.nz/descriptions/#large

[10] We found that the results of Yoshida were the same as those of Maruyama because Yoshida utilized the code of Maruyama.

Acknowledgments

Table 2: The sizes of generated grammars and the execution times. Each cell in the table represents the number of generated rules, the total lengths of the right side of all the rules except for the start variable, the length of the right side of the start variable, and the total grammar size in order from the top row. The fifth row separated by a line represents the execution time with seconds.

| text file | RePair | | | | | Re-PairImp | MR-RePair |
	Maruyama	Navarro	Prezza	Wan	Yoshida		
rand77.txt	41,651	41,642	41,632	41,675	41,651	41,661	**4,492**
	83,302	83,284	83,264	83,350	83,302	83,322	**46,143**
	9	**2**	7	**2**	9	**2**	9
	83,311	83,286	83,271	83,352	83,311	83,324	**46,152**
	0.41	**0.37**	4.76	4.27	0.40	3.95	0.42
fib41	38	38	38	38	38	**37**	38
	76	76	76	76	76	**74**	76
	3	**3**	**3**	**3**	**3**	23	**3**
	79	**79**	**79**	**79**	**79**	97	**79**
	26.75	**23.94**	96.05	483.86	25.04	1360.40	33.62
einstein.de.txt	49,968	49,949	50,218	50,057	49,968	49,933	**21,787**
	99,936	99,898	100,436	100,114	99,936	99,866	**71,709**
	12,734	12,665	13,419	**12,610**	12,734	12,672	12,683
	112,670	112,563	113,855	112,724	112,670	112,538	**84,392**
	30.08	43.45	216.74	213.15	30.76	452.56	**29.63**
E.coli	66,664	66,757	66,660	67,368	66,664	66,739	**62,363**
	133,328	133,514	133,320	134,736	133,328	133,478	**129,138**
	651,875	**649,660**	650,538	652,664	651,875	650,209	650,174
	785,203	783,174	783,858	787,400	785,203	783,687	**779,312**
	1.20	**1.02**	14.67	10.37	1.56	27.04	1.33
bible.txt	81,193	81,169	80,999	81,229	81,193	81,282	**72,082**
	162,386	162,338	161,998	162,458	162,386	162,564	**153,266**
	386,514	386,381	386,992	386,094	386,514	**385,989**	386,516
	548,900	548,719	548,990	548,552	548,900	548,553	**539,782**
	1.33	**1.21**	13.00	9.12	1.47	24.38	1.27
world192.txt	55,552	55,798	55,409	55,473	55,552	55,437	**48,601**
	111,104	111,596	110,812	110,946	111,104	110,874	**104,060**
	213,131	213,962	213,245	**212,647**	213,131	212,857	212,940
	324,235	325,558	324,057	323,593	324,235	323,731	**317,000**
	0.59	0.80	7.57	4.89	**0.56**	12.35	0.66

References

[1] Alberto Apostolico and Stefano Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000.

[2] Djamal Belazzougui and Fabio Cunial. Fast label extraction in the CDAWG. In *Proceedings of the 24th International Symposium on String Processing and Information Retrieval (SPIRE 2017)*, volume 10508 of Lecture Notes in Computer Science, pages 161–175. Springer, 2017.

[3] Djamal Belazzougui and Fabio Cunial. Representing the suffix tree with the CDAWG. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of Leibniz International Processings in Informatics, pages 7:1–7:13, 2017.

[4] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite repetition-aware data structures. In *Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching (CPM 2015)*, volume 9133 of Lecture Notes in Computer Science, pages 26–39. Springer, 2015.

[5] Philip Bille, Inge Li Grtz, and Nicola Prezza. Space-efficient Re-Pair compression. In *Proceedings of Data Compression Conference (DCC 2017)*, pages 171–180. IEEE Press, 2017.

[6] Moses Charikar, Eric Lehman, Ding Liu, Panigrahy Ring, Manoj Prabhakaran, Amit Sahai, and abhi shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[7] Francisco Claude and Gonzalo Navarro. Fast and compact web graph representations. *ACM Transactions on the Web*, 4(4):16:1–16–31, 2010.

[8] Michał Gańczorz and Artur Jeż. Improvements on Re-Pair grammar compressor. In *Proceedings of Data Compression Conference (DCC 2017)*, pages 181–190. IEEE Press, 2017.

[9] Rodrigo González and Gonzalo Navarro. Compressed text indexes with fast locate. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*, volume 4580 of Lecture Notes in Computer Science, pages 216–227.

[10] Shunsuke Inenaga, Takashi Funamoto, Masayuki Takeda, and Ayumi Shinohara. Linear-time off-line text compression by longest-first substitution. In *Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE 2003)*, volume 2857 of Lecture Notes in Computer Science, pages 137–152. Springer, 2003.

[11] N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000.

[12] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. Xml tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013.

[13] Takuya Masaki and Takuya Kida. Online grammar transformation based on Re-Pair algorithm. In *Proceedings of Data Compression Conference (DCC 2016)*, pages 349–358. IEEE Press, 2016.

[14] Ryosuke Nakamura, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Simple linear-time off-line text compression by longest-first substitution. In *Proceedings of Data Compression Conference (DCC 2007)*, pages 123–132. IEEE Press, 2007.

[15] Gonzalo Navarro and Luís MS Russo. Re-pair achieves high-order entropy. In *Proceedings of the Data Compression Conference (DCC 2008)*, page 537. IEEE Press, 2008.

[16] Carlos Ochoa and Gonzalo Navarro. RePair and all irreducible grammars are upper bounded by high-order empirical entropy. *IEEE Transactions on Information Theory*, pages 1–5, 2018.

[17] Kei Sekine, Hirohito Sasakawa, Satoshi Yoshida, and Takuya Kida. Adaptive dictionary sharing method for Re-Pair algorithm. In *Proceedings of Data Compression Conference (DCC 2014)*, pages 425–425. IEEE Press, 2014.

[18] Yasuo Tabei, Hiroto Saigo, Yoshihiro Yamanishi, and Simon J. Puglisi. Scalable partial least squares regression on grammar-compressed data matrices. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, pages 1875–1884. ACM, 2016.

[19] Takuya Takagi, Keisuke Goto, Yuta Fujishige, Shunsuke Inenaga, and Hiroki Arimura. Linear-size CDAWG: New repetition-aware indexing and grammar compression. In *Proceedings of the 24th International Symposium on String Processing and Information Retrieval (SPIRE 2017)*, volume 10508 of Lecture Notes in Computer Science, pages 304–316. Springer, 2017.

[20] Raymond Wan. *Browsing and searching compressed documents*. PhD thesis, The University of Melbourne, 2003.

A Appendix

We show Figures 4, 5, and 6 to help for understanding Proof of Theorem 4.

Let G_{rp}, G_{nmr}, and G_{mr} be the grammars generated by RePair, Naïve-MR-RePair, and MR-RePair, respectively. For a given text $T = a_1 \cdots a_n$ ($a_i \in \Sigma$, $1 \le i \le n$) of length $|T| = n$, let g_{rp}, g_{nmr}, and g_{mr} be the sizes of G_{rp}, G_{nmr}, and G_{mr}, respectively. Here, assume that $T = (aw)^{2(2^m-1)+1}a$, where $w \in \Sigma^+$ such that awa is the most frequent maximal repeat of T, and $m \in \mathbb{N}^+$. Then, by Proof of Theorem 4, $g_{nmr} > g_{rp}$ holds with some m and w such that $(m-1)(|w|-1) > 1$.

We show a concrete example of the grammar generation process of RePair and G_{rp} for $T = (\texttt{abcd})^7\texttt{a}$ with $m = 2$ and $|w| = 3$ in Figure 4. The size g_{rp} is 18 for this example. We also show an example of the process of Naïve-MR-RePair and G_{nmr} for the same T in Figure 5. As we see, the size g_{nmr} is 19, and thus $g_{nmr} > g_{rp}$ holds. As shown in Figure 5, in particular cases, Naïve-MR-RePair may fail to extract repetitive patterns (like $\texttt{abcd}$ of $(\texttt{abcd})^7\texttt{a}$ for the running example). However, this problem is solved by using MR-RePair. We show an example of the process of MR-RePair and G_{mr} for the same $T = (\texttt{abcd})^7\texttt{a}$ in Figure 6. The size g_{mr} is 16 and this is smaller than $g_{rp} = 18$. While the most frequent maximal repeat at the second replacement step is $v_\texttt{a}v_\texttt{b}v_\texttt{c}v_\texttt{d}v_\texttt{a}$, MR-RePair replaces $v_\texttt{a}v_\texttt{b}v_\texttt{c}v_\texttt{d}$ with new variable v_1 because of the additional **Step 3** of Definition 3.

Figure 4

	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a
$v_\alpha \to \alpha$ ($\alpha = a, b, c, d$)	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a
$y_1 \to v_a v_b$	y_1		v_c	v_d	y_1		v_c	v_d	y_1		v_c	v_d	y_1		v_c	v_d	y_1		v_c	v_d	y_1		v_c	v_d	y_1		v_c	v_d	v_a
$y_2 \to y_1 v_c$	y_2			v_d	y_2			v_d	y_2			v_d	y_2			v_d	y_2			v_d	y_2			v_d	y_2			v_d	v_a
$y_3 \to y_2 v_d$	y_3				y_3				y_3				y_3				y_3				y_3				y_3				v_a
$x_1 \to y_3 y_3$	x_1								x_1								x_1								y_3				v_a
$x_2 \to x_1 x_1$	x_2																x_1								y_3				v_a
$\hat{S}_{rp} \to x_2 x_1 y_3 v_a$																													

Figure 4: Grammar generation process of RePair and its generated grammar for text $(abcd)^7 a$. The grammar size is 18.

Figure 5

	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a
$v_\alpha \to \alpha$ ($\alpha = a, b, c, d$)	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a
$z_1 \to v_a v_b v_c v_d v_a$	z_1					v_b	v_c	v_d	z_1					v_b	v_c	v_d	z_1					v_b	v_c	v_d	z_1				
$z_2 \to z_1 v_b v_c v_d z_1$	z_2																z_2					v_b	v_c	v_d	z_1				
$\hat{S}_{nmr} \to z_2 v_b v_c v_d z_2$																													

Figure 5: Grammar generation process of Naïve-MR-RePair and its generated grammar for text $(abcd)^7 a$. The grammar size is 19.

Figure 6

	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a	b	c	d	a
$v_\alpha \to \alpha$ ($\alpha = a, b, c, d$)	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a	v_b	v_c	v_d	v_a
$v_1 \to v_a v_b v_c v_d$	v_1				v_1				v_1				v_1				v_1				v_1				v_1				v_a
$v_2 \to v_1 v_1$	v_2								v_2								v_2								v_1				v_a
$v_3 \to v_2 v_2$	v_3																v_2								v_1				v_a
$\hat{S}_{mr} \to v_3 v_2 v_1 v_a$																													

Figure 6: Grammar generation process of MR-RePair and its generated grammar for text $(abcd)^7 a$. The grammar size is 16.