algorithms

MDPI

Article

Practical Grammar Compression Based on Maximal Repeats †

Isamu Furuya [1,*], Takuya Takagi [1,‡], Yuto Nakashima [2], Shunsuke Inenaga [2,3], Hideo Bannai [2,§] and Takuya Kida [1,‖]

1 Graduate School of IST, Hokkaido University, N14-W9, Kitaku, Sapporo 060-0814, Japan; tkg@ist.hokudai.ac.jp (T.T.); kida@ist.hokudai.ac.jp (T.K.)
2 Department of Informatics, Kyushu University, 744, Motooka, Nishiku, Fukuoka 819-0395, Japan; yuto.nakashima@inf.kyushu-u.ac.jp (Y.N.); inenaga@inf.kyushu-u.ac.jp (S.I.); bannai@inf.kyushu-u.ac.jp (H.B.)
3 PRESTO, Japan Science and Technology Agency, 4-1-8, Honcho, Kawaguchi, Saitama 332-0012, Japan
* Correspondence: furuya@ist.hokudai.ac.jp; Tel.: +81-11-706-7680
† This paper is an extended version of our paper published in the Proceedings of the Data Compression Conference (DCC 2019), Snowbird, UT, USA, 26–29 March 2019.
‡ Current affiliation: Fujitsu Laboratories LTD, 4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan.
§ Current affiliation: M&D Data Science Center, Tokyo Medical and Dental University, 2-3-10 Kanda Surugadai, Chiyoda-ku, Tokyo 101-0062, Japan.
‖ Current affiliation: Department of Life Science and Technology, Faculty of Engineering, Hokkai-Gakuen University, 1-1, Nishi 11, Minami 26, Chuo-ku, Sapporo 064-0926, Japan.

Received: 18 March 2020; Accepted: 20 April 2020; Published: 23 April 2020

Abstract: This study presents an analysis of RePair, which is a grammar compression algorithm known for its simple scheme, while also being practically effective. First, we show that the main process of RePair, that is, the step by step substitution of the most frequent symbol pairs, works within the corresponding most frequent maximal repeats. Then, we reveal the relation between maximal repeats and grammars constructed by RePair. On the basis of this analysis, we further propose a novel variant of RePair, called MR-RePair, which considers the one-time substitution of the most frequent maximal repeats instead of the consecutive substitution of the most frequent pairs. The results of the experiments comparing the size of constructed grammars and execution time of RePair and MR-RePair on several text corpora demonstrate that MR-RePair constructs more compact grammars than RePair does, especially for highly repetitive texts.

Keywords: lossless data compression; RePair; maximal repeat

1. Introduction

Grammar compression is one of the lossless data compression methods. For a given text, grammar compression constructs a small size context free grammar that derives only the given text. While the problem of constructing the smallest such grammar for a given text is known to be NP-hard [1], several approximation algorithms have been proposed. One of them is RePair [2], which is an off-line grammar compression algorithm. Despite its simple scheme, RePair is known for its high compression in practice [3–5], and hence, it has been comprehensively studied. Some examples of studies on the RePair algorithm include its extension to an online algorithm [6], practical working time/space improvements [7,8], applications to various fields [3,9,10], and theoretical analysis of generated grammar sizes [1,11,12].

In the field of text processing, the repetitiveness of a text is considered an important property. Furthermore, it has been suggested that the extent of the repetitiveness of a given text directly relates to the number of maximal repeats in the text. Belazzougui et al. [13] demonstrated theoretical relations between the number of extensions of maximal repeats and famous other properties of text such as the number of factors in the Lempel–Ziv parsing and the number of runs in the Burrows–Wheeler transform. Also, several text indexing data structures, whose sizes are bounded by the number of extensions of maximal repeats, have been proposed in the literature [14–16].

In this study, we analyzed the properties of RePair, focusing on their relationship to maximal repeats. Although RePair has been extensively studied, to the best of our knowledge, no previous study has associated RePair with maximal repeats. Furthermore, we propose MR-RePair, which is a novel grammar compression algorithm based on the property of maximal repeats. Several off-line grammar compression techniques based on the properties of (non-maximal) repeats have been proposed previously [17–19]. Recently, Gańczorz and Jeż introduced a heuristic method that improves the practical compression ratio of RePair in terms of the grammar size [20]. However, none of the previously proposed methods use the properties of maximal repeats. In this study, we aim to demonstrate that there is a theoretical guarantee for the size of constructed grammars; under a specific condition, the size of the grammar constructed by MR-RePair is smaller than or equal to that constructed by RePair. Our experiments show that MR-RePair constructs smaller grammars compared to RePair. We emphasize that generating a grammar of small size is of great importance since most, if not all, existing algorithms/data structures that work on grammar-compressed texts have running time dependent on the grammar sizes (see e.g., [21–28] and the references therein) and not directly on the encoded sizes.

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

1. We show interesting relations between maximal repeats and grammars constructed by RePair.
2. We propose MR-RePair, which is a novel variant of RePair based on replacing the most frequent maximal repeats.
3. We implement MR-RePair and experimentally demonstrate that MR-RePair produces smaller grammars than all tested implementations of RePair. For a highly repetitive text used in the experiments, MR-RePair decreased the size of the constructed grammar to about 55% of that of RePair.

The rest of this paper is organized as follows. In Section 2, we introduce basic notations and review maximal repeats, grammar compression, and RePair. In Section 3, we present an analysis of the properties of RePair and demonstrate its relationship to maximal repeats. The definition and implementation of MR-RePair and its comparison with RePair are provided in Section 4. In Section 5, we report the experimental results of comparing RePair and MR-RePair. Finally, in Section 6, we conclude the paper.

A preliminary version of this work was previously and partially presented at the Data Compression Conference (DCC 2019) [29]. Compared to the preliminary version, the additional and extended parts of this paper are as follows.

1. We discuss how the sizes of generated grammars differ depending on the implementation of RePair, and prove that a lower bound of the maximum difference of the sizes is $\frac{1}{6}(\sqrt{6n+1}+13)$ (Definition 1 and Theorem 3).
2. We describe Naïve-MR-RePair, which is a naïve version of our MR-RePair. Furthermore, we prove that there is the case where the grammar size of Naïve-MR-RePair becomes larger than that of RePair in logarithmic order of the length of input string (Theorem 4).
3. We performed our experiment again following the advice of a DCC reviewer.
4. For some of the lemmas we omitted the proofs in the previous version, in this version, we show them all.

2. Preliminaries

In this section, 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*, that is, an ordered finite set of symbols. An element $T = t_1 \cdots t_n$ of Σ^* is called a *string* or a *text*, where $|T| = n$ denotes its length. Let ϵ be an empty string of length 0, that is, $|\epsilon| = 0$. Let $\Sigma^+ = \Sigma^* \backslash \{\epsilon\}$ and $T = t_1 \cdots t_n \in \Sigma^n$ be any text of length n. If $T = usw$ with $u, s, w \in \Sigma^*$, then s is called a *substring* of T. Let $T[i..j] = t_i \cdots t_j$ for any $1 \leq i \leq j \leq n$ denote a substring of T beginning at i and ending at 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 to be a *superstring* of S if T contains all texts of S as substrings. Let $\#\mathrm{occ}(s)$ denote the *frequency* of s, i.e., the number of occurrences of s in a text as a substring. If there exists an isomorphism from an alphabet Σ to another alphabet $\hat{\Sigma}$, texts Σ^* and $\hat{\Sigma}^*$ are said to be *isomorphic* for Σ and $\hat{\Sigma}$.

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 in a form of ws (or sw), where $w \in \Sigma^*$. We define s as a *left* (or *right*) *maximal* if left (or right) extensions of s occur a strictly less number of times in T than s. Accordingly, s is a *maximal repeat* of T if s is both left and right maximal. In this paper, we only consider strings with a length of more than 1 as maximal repeats. For example, substring `abra` of $T =$ `abracadabra` is a maximal repeat, whereas `br` is not.

2.3. Grammar Compression

A *context free grammar* (CFG or simply *grammar*) G is defined as a four-tuple $G = \{V, \Sigma, S, R\}$, where V denotes an ordered finite set of *variables*, Σ denotes an ordered finite alphabet, R denotes a finite set of binary relations called *production rules* (or *rules*) between V and $(V \cup \Sigma)^*$, and $S \in V$ denotes a special variable called *start variable*. A production rule refers to the situation, where a variable is substituted and written in a form of $v \to w$, 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 $x \to y \in R$, we write $X \Rightarrow Y$, and denote the reflexive transitive closure of $\Rightarrow$ as $\overset{*}{\Rightarrow}$. Let $val(v)$ be a string derived from v, i.e., $v \overset{*}{\Rightarrow} val(v)$. 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 for lossless text data compression that constructs a restricted CFG uniquely deriving the 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 $1 \leq k \leq j_n$).

For estimating the effectiveness of compression, we use the size of the constructed grammar, which is defined as the total length of the right-hand side of all production rules of the grammar.

2.4. RePair

RePair is a grammar compression algorithm proposed by Larsson and Moffat [2]. For input text T, let $G = \{V, \Sigma, S, R\}$ be the grammar constructed by RePair. Then, the RePair procedure can be described with 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, and then, add $v \to p$ to R.
Step 4. Re-evaluate the frequencies of pairs for the updated 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 illustrates an example of the grammar generation process of RePair.

	a	b	r	a	c	a	d	a	b	r	a
$v_\alpha \to \alpha$ ($\alpha =$ 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_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 the text `abracadabra`. The generated grammar is $\{\{v_{\mathtt{a}}, v_{\mathtt{b}}, v_{\mathtt{r}}, v_{\mathtt{c}}, v_{\mathtt{d}}, v_1, v_2, v_3, S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}\}, S, \{v_{\mathtt{a}} \to \mathtt{a}, v_{\mathtt{b}} \to \mathtt{b}, v_{\mathtt{r}} \to \mathtt{r}, v_{\mathtt{c}} \to \mathtt{c}, v_{\mathtt{d}} \to \mathtt{d}, v_1 \to v_{\mathtt{a}} v_{\mathtt{b}}, v_2 \to v_1 v_{\mathtt{c}}, v_3 \to v_2 v_{\mathtt{d}}, S \to v_3 v_{\mathtt{c}} v_{\mathtt{a}} v_{\mathtt{d}} v_3\}\}$ with a size of 16.

Lemma 1 ([2]). *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 denotes the cardinality of the source alphabet, and k' denotes the cardinality of the final dictionary.*

3. Analysis of RePair

This section presents an analysis of RePair with respect to its relationship to maximal repeats and introduces an important concept, called MR-order.

3.1. RePair and Maximal Repeats

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

Theorem 1. *Let T be a given text, assuming 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 a 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 containing 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 that there exists a different most frequent maximal repeat r' containing 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 a 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 it. By definition, $\#\mathrm{occ}(r) = \#\mathrm{occ}(p)$. If p is not the most frequent symbol pair in T, there exists a 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. □

The following corollary is derived directly 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 the overlap between any two occurrences of most frequent maximal repeats is at most 1.*

Proof. Let xw and wy be the most frequent maximal repeats that have an overlapping occurrence xwy, where $x, y, w \in \Sigma^+$. If we assume that $|w| \geq 2$, since xw and wy are most frequent maximal repeats, it holds 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. □

Theorem 1 can now be proved based on the above lemmas and corollary.

Proof of Theorem 1. According to 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. According to Lemma 2, there is a unique maximal repeat that is most frequent and contains p once. We denote such maximal repeat as r. Let us 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 $r[|r|]$ as $\dot{x}$ and $\dot{y}$, respectively. There are the following two 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 updated text, and according to 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, whereas $z\dot{x}$ 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 (old and new) contained in the most frequent maximal repeats of the same frequency until they are replaced by a single variable. Then, s can be generated by replacing 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 $\#\mathrm{occ}(zxpy) < f$ according to the assumption that xpy is a maximal repeat. Suppose RePair replaces $z\dot{x}$ by a variable v before p is replaced. Note that according to Lemma 2, there is a maximal repeat occurring f times and including $z\dot{x}$ once (we denote the maximal repeat as r'), and $r' \neq r$ by assumption. According to Lemma 3, the length of the overlap of r and r' is at most 1, and then, only $\dot{x}$ is a symbol present in 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 updated 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 by first replacing r' and then replacing $r[2..|r|]$. Similarly, this holds for $\dot{y}w$ when $\#\mathrm{occ}(\dot{y}w) = f$ and $\#\mathrm{occ}(z\dot{x}) = \#\mathrm{occ}(\dot{y}w) = f$. □

3.2. MR-Order

According to Theorem 1, if there is just one most frequent maximal repeat in the current text, then RePair replaces its all occurrences step by step. However, a problem arises if there are two or more most frequent maximal repeats, with some of them overlapping. In this case, the selection order of

pairs (of course, they are most frequent) affects the priority of maximal repeats. We call this order of selecting (summarizing) maximal repeats as the *maximal repeat selection order* (or simply *MR-order*). Note that, the selection order of pairs actually depends on the implementation of RePair.

For instance, consider the text `abcdeabccde`, where `abc` and `cde` are the most frequent maximal repeats occurring twice. There are two MR-orders, depending on which of the two maximal repeats `abc` or `cde` is given priority. The results of the replacement using RePair with the MR-order are (i) $xyxcx$ with variables x and y such that $x \stackrel{*}{\Rightarrow}$ `abc` and $y \stackrel{*}{\Rightarrow}$ `de`, and (ii) $zwzcw$ with variables z and w such that $z \stackrel{*}{\Rightarrow}$ `ab` and $w \stackrel{*}{\Rightarrow}$ `cde`. More precisely, there are 12 possible ways in which RePair can compress the text, with the following generated rule sets:

1. $\{v_1 \to \texttt{ab}, v_2 \to v_1\texttt{c}, v_3 \to \texttt{de}, S \to v_2v_3v_2\texttt{c}v_3\}$,
2. $\{v_1 \to \texttt{ab}, v_2 \to \texttt{de}, v_3 \to v_1\texttt{c}, S \to v_3v_2v_3\texttt{c}v_2\}$,
3. $\{v_1 \to \texttt{bc}, v_2 \to \texttt{a}v_1, v_3 \to \texttt{de}, S \to v_2v_3v_2\texttt{c}v_3\}$,
4. $\{v_1 \to \texttt{bc}, v_2 \to \texttt{de}, v_3 \to \texttt{a}v_1, S \to v_3v_2v_3\texttt{c}v_2\}$,
5. $\{v_1 \to \texttt{ed}, v_2 \to \texttt{ab}, v_3 \to v_2\texttt{c}, S \to v_3v_1v_3\texttt{c}v_1\}$,
6. $\{v_1 \to \texttt{ed}, v_2 \to \texttt{bc}, v_3 \to \texttt{a}v_2, S \to v_3v_1v_3\texttt{c}v_1\}$,
7. $\{v_1 \to \texttt{ab}, v_2 \to \texttt{cd}, v_3 \to v_2\texttt{e}, S \to v_1v_3v_1\texttt{c}v_3\}$,
8. $\{v_1 \to \texttt{ab}, v_2 \to \texttt{de}, v_3 \to \texttt{c}v_2, S \to v_1v_3v_1\texttt{c}v_3\}$,
9. $\{v_1 \to \texttt{cd}, v_2 \to \texttt{ab}, v_3 \to v_1\texttt{e}, S \to v_2v_3v_2\texttt{c}v_3\}$,
10. $\{v_1 \to \texttt{cd}, v_2 \to v_1\texttt{e}, v_3 \to \texttt{ab}, S \to v_3v_2v_3\texttt{c}v_2\}$,
11. $\{v_1 \to \texttt{ed}, v_2 \to \texttt{ab}, v_3 \to \texttt{c}v_1, S \to v_2v_3v_2\texttt{c}v_3\}$,
12. $\{v_1 \to \texttt{ed}, v_2 \to \texttt{c}v_1, v_3 \to \texttt{ab}, S \to v_3v_2v_3\texttt{c}v_2\}$.

Here, 1–6 have the same MR-order because `abc` precedes `cde` in all of them. At the same time, 7–12 have the same MR-order for the same reason: `cde` precedes `abc`.

If there are several distinct most frequent pairs with overlaps, RePair constructs grammars with different sizes according to the selection order of the pairs. For example, consider the text `bcxdabcyabzdabvbcuda`. There are three most frequent pairs, namely, `ab`, `bc`, and `da`, occurring three times each. If RePair takes `ab` first, the rule set of the generated grammar may become $\{v_1 \to \texttt{ab},\ v_2 \to \texttt{bc},\ v_3 \to \texttt{d}v_1,\ S \to v_2\texttt{x}v_3\texttt{cy}v_1\texttt{z}v_3\texttt{v}v_2\texttt{uda}\}$ and its size is 19. If RePair takes `da` first, the rule set of the generated grammar may become $\{v_1 \to \texttt{da},\ v_2 \to \texttt{bc},\ S \to v_2\texttt{x}v_1v_2\texttt{yabz}v_1\texttt{bv}v_2\texttt{u}v_1\}$ and its size 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 their replacement order.*

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

Theorem 2. *The sizes of 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. According to 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 with r_1 preceding all other maximal repeats in this MR-order. As a result, r_1 is converted into a variable, and according to 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 as r_2) with the following slight difference: the pattern actually replaced would be a substring of r_2 excluding its beginning or end 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 order of selecting pairs. Similarly, this holds for all most frequent maximal repeats and every maximum frequency of pairs through the entire process of RePair. □

3.3. Greatest Size Difference of RePair

We consider the problem of determining the greatest size difference between possible outcomes of RePair.

Definition 1 (Greatest size difference)**.** *Let g and g' be the 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* $\max(|g - g'|)$.

A lower bound of the GSDRP can be established according to the following theorem.

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

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

$$\begin{aligned} B &= l_1xyr_1l_2xyr_2\cdots l_{f-1}xyr_{f-1}l_fxyr_f,\\ L &= \diamondsuit l_1x\diamondsuit l_2x\cdots\diamondsuit l_fx,\\ R &= \diamondsuit yr_1\diamondsuit yr_2\cdots\diamondsuit yr_f, \end{aligned}$$

where $x, y, l_1, \ldots, l_f, r_1, \ldots, r_f$ denote distinct symbols, and each occurrence of $\diamondsuit$ denotes a distinct symbol. Consider text $T = BL^{f-1}R^{f-1}$. Here, xy, l_1x, $\cdots$, l_fx, yr_1, $\cdots$, yr_f are the most frequent maximal repeats with a frequency f in T. Let G and G' be grammars generated by RePair for T in different MR-order, such that (i) xy precedes all other maximal repeats and (ii) xy follows all other maximal repeats, respectively. We denote the sizes of G and G' as g and g', respectively.

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

$$B_1 = l_1v_1r_1l_2v_1r_2\cdots l_{f-1}v_1r_{f-1}l_fv_1r_f.$$

Each pair in B_1 occurs only once in the entire text $B_1L^{f-1}R^{f-1}$. This means that B_1 can never be shortened from the current length of $3f$. In the remaining steps, l_ix and yr_i (for $i = 1, \cdots, f$) are replaced. L and R are changed to texts with a length of $2f$ each. 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. According to their MR-order, l_1x, $\cdots$, l_fx, yr_1, $\cdots$, yr_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, there are $2f$ rules in G' deriving l_ix and yr_i (for $i = 1, \cdots, f$), whereas the rule deriving xy is absent. L and R are changed to texts with a length of $2f$ each, and B is changed to a text with a 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. According to Equations (1)–(3),

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

holds and the theorem follows. □

4. MR-RePair

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

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

4.1. Naïve-MR-RePair

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

Figure 2 illustrates an example of the grammar generation process of Naïve-MR-RePair. Figures 1 and 2 explain why the strategy of using maximal repeats is more effective compared to that using pairs. When compressing the text $v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}v_{\mathtt{a}}v_{\mathtt{c}}v_{\mathtt{a}}v_{\mathtt{d}}v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}v_{\mathtt{a}}$, both RePair and Naïve-MR-RePair generate subgrammars deriving the most frequent maximal repeat $v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}v_{\mathtt{a}}$. The rule set of the subgrammar generated by RePair is $\{v_1 \to v_{\mathtt{a}}v_{\mathtt{b}}, v_2 \to v_1v_{\mathtt{r}}, v_3 \to v_2v_{\mathtt{a}}\}$ with a size of 6. At the same time, the rule set of the subgrammar generated by Naïve-MR-RePair is $\{v_1 \to v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}v_{\mathtt{a}}\}$ with a size of 4.

	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			
$S \to v_1v_{\mathtt{c}}v_{\mathtt{a}}v_{\mathtt{d}}v_1$	S										

Figure 2. An example of the grammar generation process of Naïve-MR-RePair for the text `abracadabra`. The generated grammar is $\{\{v_{\mathtt{a}}, v_{\mathtt{b}}, v_{\mathtt{r}}, v_{\mathtt{c}}, v_{\mathtt{d}}, v_1, S\}, \{\mathtt{a},\mathtt{b},\mathtt{r},\mathtt{c},\mathtt{d}\}, S, \{v_{\mathtt{a}} \to \mathtt{a}, v_{\mathtt{b}} \to \mathtt{b}, v_{\mathtt{r}} \to \mathtt{r}, v_{\mathtt{c}} \to \mathtt{c}, v_{\mathtt{d}} \to \mathtt{d}, v_1 \to v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}v_{\mathtt{a}}, S \to v_1v_{\mathtt{c}}v_{\mathtt{a}}v_{\mathtt{d}}v_1\}\}$ with a size of 14.

However, the following theorem indicates that the size of the grammar generated by Naïve-MR-RePair is larger than that generated by RePair in certain cases, even when they work in the same MR-order. Roughly speaking, this is caused by the overlaps of maximal repeats. When there is an occurrence of the most frequent maximal repeat that overlaps with its occurrence, little difference would arise in grammar constructing processes of RePair and Naïve-MR-RePair from the view point of maximal repeats, that is, the targeted maximal repeats would vary in RePair and in Naïve-MR-RePair (RePair replaces the targeted maximal repeat step by step and Naïve-MR-RePair replaces it at once). Indeed, if maximal repeats are carefully embedded in a text for increasing the difference, the case presented in the following theorem occurs.

Theorem 4. *Given a text T with a length of n, let g_{rp} and g_{nmr} be the sizes of the grammars generated by RePair and Naïve-MR-RePair for T, respectively, assuming that RePair and Naïve-MR-RePair work in the same MR-order. Then, there is a case when $g_{nmr} = g_{rp} + \Omega(\log n)$ holds. (We show a concrete example of this theorem in Appendix A.)*

Proof. Let $G_{rp} = \{V_{rp}, \Sigma_{rp}, S_{rp}, R_{rp}\}$ and $G_{nmr} = \{V_{nmr}, \Sigma_{nmr}, S_{nmr}, R_{nmr}\}$ be the grammars generated by RePair and Naïve-MR-RePair, respectively. Let T' be the text generated just after **Step 1** of RePair or Naïve-MR-RePair (the **Step 1** is common in both algorithms), that is, $T' = v_1 \cdots v_n$ 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}) deriving T'. Let $T' = (uw)^{2^{m+1}-1}u$, 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 $\hat{R}_{nmr}$ are defined as follows:

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

- $|w|$ 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 $2 \le i \le m$, and
- 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 $1 \le i \le d$, then $\hat{R}_{nmr}$ consists of

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

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

$$\hat{g}_{rp} = 2|w| + 2m + (m + 2) = 3m + 2|w| + 2, \tag{4}$$

$$\hat{g}_{nmr} = |w| + 2 + (|w| + 2)(d - 1) = (|w| + 2)d. \tag{5}$$

Here, with regard to the length of T', we have

$$n = (2(2^m - 1) + 1)(|w| + 1) + 1, \text{ and}$$
$$n = (2^d - 1)|w| + 2^d.$$

From these, $d = m + 1$ holds. Hence, according to Equations (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. □

4.2. MR-RePair

The grammar size of Naïve-MR-RePair becomes larger than that of RePair as shown in Theorem 4 because Naïve-MR-RePair cannot replace all occurrences of 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 Naïve-MR-RePair.

Definition 3 (MR-RePair)**.** *For an input text T, let $G = \{V, \Sigma, S, R\}$ be the grammar generated by MR-RePair. MR-RePair constructs T through 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, use* $r[1..|r|-1]$ *instead of* r *in* **Step 4**.
Step 4. *Replace every occurrence of* r *with a new variable* v *and then add* $v \to r$ *to* R.
Step 5. *Re-evaluate the frequencies of maximal repeats for the updated 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 can easily extend the concept of the MR-order to MR-RePair. We do not care if it uses $r[2..|r|]$ in **Step 3**, instead of $r[1..|r|-1]$. MR-RePair can replace all occurrences of r even if it overlaps with itself in some occurrences since, according to Lemma 3, the length of the overlaps of the most frequent maximal repeats is at most 1. If $r[1] = r[|r|]$ but r does not overlap with itself, then $vr[|r|]$ becomes the most frequent maximal repeat after $r[1..|r|-1]$ is replaced by v and $vr[|r|]$ is replaced immediately. Similar to RePair, MR-RePair still cannot replace all of them if $|r| = 2$.

Figure 3 illustrates an example of the grammar generation process of MR-RePair. Although the size of the grammar generated by MR-RePair as shown in Figure 3 is larger than that generated by Naïve-MR-RePair as shown in Figure 2, it is still smaller than that generated by RePair as 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 the sizes of the grammars generated by RePair and MR-RePair, respectively. Then,* $\frac{1}{2}g_{rp} < g_{mr} \leq g_{rp}$ *holds.*

	a	b	r	a	c	a	d	a	b	r	a
$v_\alpha \to \alpha$ ($\alpha =$ 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 the text `abracadabra`. The generated grammar is $\{\{v_{\mathtt{a}}, v_{\mathtt{b}}, v_{\mathtt{r}}, v_{\mathtt{c}}, v_{\mathtt{d}}, v_1, S\}, \{\mathtt{a}, \mathtt{b}, \mathtt{r}, \mathtt{c}, \mathtt{d}\}, S, \{v_{\mathtt{a}} \to \mathtt{a}, v_{\mathtt{b}} \to \mathtt{b}, v_{\mathtt{r}} \to \mathtt{r}, v_{\mathtt{c}} \to \mathtt{c}, v_{\mathtt{d}} \to \mathtt{d}, v_1 \to v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{r}}, v_2 \to v_1 v_{\mathtt{a}}, S \to v_2 v_{\mathtt{c}} v_{\mathtt{a}} v_{\mathtt{d}} v_2\}\}$ with a size of 15.

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 a length of n. Let T' be the text generated just after **Step 1** of RePair or Naïve-MR-RePair (the **Step 1** is common in both algorithms), that is, $T' = v_1 \cdots v_n$ such that $v_i \in V_{rp} \cap V_{mr}$ and $v_i \to T[i] \in R_{rp} \cap R_{mr}$ (for $i = 1, \cdots, n$).

Let f_1 be the maximum frequency of the maximal repeats in T'. According to Corollary 1, the maximum frequency of the 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}) generated while RePair (or MR-RePair) replaces pairs (or maximal repeats) with the frequency f_1, $\hat{g}_{rp}^{(f_1)}$ (or $\hat{g}_{mr}^{(f_1)}$) be the size of this subgrammar, and $T_{rp}^{(f_1)}$ (or $T_{mr}^{(f_1)}$) be the updated text after all pairs (or maximal repeats) with the 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' assuming that they are prioritized in this order by the MR-order. Let $l_i^{(f_1)}$ (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. Since RePair replaces such substring step by step and MR-RePair replaces it at once, the following holds:

$$\hat{g}_{rp}^{(f_1)} = \sum_{i=1}^{m_1} 2(l_i^{(f_1)} - 1)\,, \tag{6}$$

$$\hat{g}_{mr}^{(f_1)} = \sum_{i=1}^{m_1} l_i^{(f_1)}. \tag{7}$$

From these,

$$\begin{aligned}\hat{g}_{rp}^{(f_1)} - \hat{g}_{mr}^{(f_1)} &= 2\sum_{i=1}^{m_1} l_i^{(f_1)} - 2m_1 - \sum_{i=1}^{m_1} l_i^{(f_1)} \\ &= \sum_{i=1}^{m_1} l_i^{(f_1)} - 2m_1 \\ &\geq 2m_1 - 2m_1 = 0 \ \ (\because l_i \geq 2 \ \text{ holds for each } i).\end{aligned}$$

Hence,

$$\hat{g}_{mr}^{(f_1)} \leq \hat{g}_{rp}^{(f_1)} \tag{8}$$

holds. According to Equation (6),

$$\begin{aligned}\hat{g}_{rp}^{(f_1)} &= 2\sum_{i=1}^{m_1} l_i^{(f_1)} - 2m_1 \\ &= 2\hat{g}_{mr}^{(f_1)} - 2m_1 \quad \text{(by Equation (7))}.\end{aligned}$$

Hence,

$$\frac{1}{2}\hat{g}_{rp}^{(f_1)} < \hat{g}_{mr}^{(f_1)} \tag{9}$$

holds. Therefore, according to Equations (8) and (9),

$$\frac{1}{2}\hat{g}_{rp}^{(f_1)} < \hat{g}_{mr}^{(f_1)} \leq \hat{g}_{rp}^{(f_1)} \tag{10}$$

holds. The updated 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 the maximal repeats in $T_{rp}^{(f_1)}$ (and $T_{mr}^{(f_1)}$). Then, a similar analysis holds for $\hat{G}_{rp}^{(f_2)}$ and $\hat{G}_{mr}^{(f_2)}$. Hence, $\frac{1}{2}\hat{g}_{rp}^{(f_2)} < \hat{g}_{mr}^{(f_2)} \leq \hat{g}_{rp}^{(f_2)}$ holds similarly to Equation (10), and the updated texts $T_{rp}^{(f_2)}$ and $T_{mr}^{(f_2)}$ are isomorphic. Inductively, for every maximum frequency of maximal repeats f_i, $\frac{1}{2}\hat{g}_{rp}^{(f_i)} < \hat{g}_{mr}^{(f_i)} \leq \hat{g}_{rp}^{(f_i)}$ holds and the updated texts $T_{rp}^{(f_i)}$ and $T_{mr}^{(f_i)}$ are isomorphic. Let k be a natural number such that $f_k > 1$ and $f_{k+1} = 1$, that is, k is the number of times that the maximum frequency decreases through the entire process of RePair and MR-RePair. Then,

$$\begin{aligned}g_{rp} &= \sum_{j=1}^{k} \hat{g}_{rp}^{(f_j)} + |\Sigma| + |T_{rp}^{(f_k)}| \\ &= \sum_{j=1}^{k}\sum_{i=1}^{m_j} 2(l_i^{(f_j)} - 1) + |\Sigma| + |T_{rp}^{(f_k)}|\,, \ \text{and} \qquad (11) \\ g_{mr} &= \sum_{j=1}^{k} \hat{g}_{mr}^{(f_j)} + |\Sigma| + |T_{mr}^{(f_k)}| \\ &= \sum_{j=1}^{k}\sum_{i=1}^{m_j} l_i^{(f_j)} + |\Sigma| + |T_{mr}^{(f_k)}| \qquad (12)\end{aligned}$$

hold. Recall that each symbol $a \in \Sigma$ is replaced with a new variable in the first step both in RePair and in MR-RePair. $|\Sigma|$ is the size of the subgrammar consisting of the rules generated in the first step.

Since every $l_i^{(f_j)} \geq 2$ and $|T_{rp}^{(f_k)}| = |T_{mr}^{(f_k)}|$, $\frac{1}{2}g_{rp} < g_{mr} \leq g_{rp}$ follows Equations (11) and (12), and thus, the proposition holds. $g_{mr} = g_{rp}$ holds when every length $l_i^{(f_j)}$ is 2. □

However, when the MR-orders of RePair and MR-RePair are different, then the grammar generated by MR-RePair can be larger than that generated by RePair, as the following theorem indicates:

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

Proof. We show a concrete example of the case stated in the proposition. Consider text `abcxabcyab czcxcxcycyczcz`. There are four most-frequent maximal repeats, `abc`, `cx`, `cy`, and `cz`. Let A and B be two different MR-orders such that A prioritizes the maximal repeats in order of `abc`, `cx`, `cy`, `cz` and B prioritizes the maximal repeats in order of `cx`, `cy`, `cz`, `abc`, respectively. MR-RePair working in A generates a grammar whose rules are $\{v_{\mathtt{a}} \to \mathtt{a}, v_{\mathtt{b}} \to \mathtt{b}, v_{\mathtt{c}} \to \mathtt{c}, v_{\mathtt{x}} \to \mathtt{x}, v_{\mathtt{y}} \to \mathtt{y}, v_{\mathtt{z}} \to \mathtt{z}, v_1 \to v_{\mathtt{a}}v_{\mathtt{b}}v_{\mathtt{c}}, v_2 \to v_{\mathtt{c}}v_{\mathtt{x}}, v_3 \to v_{\mathtt{c}}v_{\mathtt{y}}, v_4 \to v_{\mathtt{c}}v_{\mathtt{z}}, S \to v_1xv_1yv_1zv_2v_2v_3v_3v_4v_4\}$, where S is the start variable. Meanwhile, RePair working in B generates a grammar whose rules are $\{v_{\mathtt{a}} \to \mathtt{a}, v_{\mathtt{b}} \to \mathtt{b}, v_{\mathtt{c}} \to \mathtt{c}, v_{\mathtt{x}} \to \mathtt{x}, v_{\mathtt{y}} \to \mathtt{y}, v_{\mathtt{z}} \to \mathtt{z}, v_1 \to v_{\mathtt{c}}v_{\mathtt{x}}, v_2 \to v_{\mathtt{c}}v_{\mathtt{y}}, v_3 \to v_{\mathtt{c}}v_{\mathtt{z}}, v_4 \to v_{\mathtt{a}}v_{\mathtt{b}}, S \to v_4v_1v_4v_2v_4v_3v_1v_1v_2v_2v_3v_3\}$, where S is the start variable. The size of the grammar generated by MR-RePair working in A is 27, whereas the size of that generated by RePair working in B is 26. □

While Theorem 6 indicates that the grammar can be larger in MR-RePair than in RePair, in Section 5 we demonstrate that MR-RePair outperforms RePair in practice.

We can implement MR-RePair by extending the original implementation of RePair stated in [2] and holding the same complexity.

Theorem 7. *Let $G = \{V, \Sigma, S, R\}$ be the grammar generated by MR-RePair for a given text with a length of 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' denote the cardinalities of Σ and V, respectively.*

Proof. Compared to RePair, the additional operations performed by MR-RePair are (i) extending the selected pair to left and right until it becomes a maximal repeat and (ii) checking and excluding either the beginning or the end of the obtained maximal repeat if they are the same. These additional operations can be realized using the same data structures as those employed in RePair. Then, the space complexity of MR-RePair follows Lemma 1.

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

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, the rule can be removed for v, and the one occurrence of v can be replaced with the right-hand side of the removed rule. However, the 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 and conducted some comparative experiments. In particular, we compared the sizes of constructed grammars and execution times of the proposed MR-RePair, several existing RePair implementations, and Re-PairImp (https://bitbucket.org/IguanaBen/repairimproved), which was recently proposed in [20] as an improvement of RePair.

As stated in Remark 1, the MR-order affects the size of a constructed grammar. In practice, the MR-order varies depending on the implementation of the priority queue that manages pairs. For this reason, we used four different implementations of RePair in the comparative analysis, and they were implemented by Maruyama (https://code.google.com/archive/p/re-pair/), Navarro (https://www.dcc.uchile.cl/~gnavarro/software/index.html), Prezza (https://github.com/nicolaprezza/Re-Pair) [7], and Wan (https://github.com/rwanwork/Re-Pair); we ran it with level 0 (no heuristic option), respectively.

Table 1 lists the details of the texts that we used in the experiments. In particular, we employed three texts as highly repetitive texts: one is a randomly generated text (rand77.txt), and the other two are a Fibonacci string (fib41) and a German text (einstein.de.txt) selected from the Repetitive Corpus of the Pizza&Chili Corpus (http://pizzachili.dcc.uchile.cl/repcorpus.html). The randomly generated text, rand77.txt, consists of alphanumeric symbols and some special symbols. It was generated by concatenating 32 copies of a block that includes 1024 random strings of length 64 each, i.e., the size of the randomly generated text is $64 \times 1024 \times 32 = 2,097,152$ byte. In addition, we used three texts (E.coli, bible.txt, and world192.txt) selected from the Large Corpus (http://corpus.canterbury.ac.nz/descriptions/#large) to consider a real-data case. We executed each program seven times for each text and measured the elapsed CPU time only for the grammar generation process. We calculated the average time across five results, excluding the minimum and maximum values among the seven runs. The experiments were run on a computer equipped with an Intel(R) Core i7-8700 3.2–4.6 GHz 6core, 32GB RAM, and using Ubuntu 16.04. All of the programs were compiled using gcc version 7.4 with the "-O3" option.

Table 1. Text files used in our experiments.

Text	Size (bytes)	$\|\Sigma\|$	Content
rand77.txt	2,097,152	77	32 copies of 1024 random patterns with a length of 64
fib41	267,914,296	2	Fibonacci string from the Pizza&Chili Corpus
einstein.de.txt	92,758,441	117	Edit history of the 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

Table 2 summarizes the experimental results. Unfortunately, Re-PairImp was unable to process fib41 in our experimental environment because of a lack of memory. Here, we excluded the number of rules generating a single terminal symbol from the number of rules since they are the same between RePair and MR-RePair. As shown in the table, the sizes of grammars constructed by each RePair implementation differ from each other for all texts except fib41. In any case, MR-RePair is not inferior to RePair in terms of the size of grammars while in Theorem 6 we show that the grammar can be larger in MR-RePair than in RePair if their MR-orders are different. For rand77.txt, the number of rules and size of the grammars for MR-RePair decreased to about 11% and 55% of those for RePair, respectively. Long maximal repeats occur more frequently in rand77.txt than in other texts and we consider this is a main reason of the remarkable effectiveness of MR-RePair for the text.

For einstein.de.txt, the number of rules and size of the grammar decreased to about 44% and 72% of those for RePair, respectively. By contrast, it turned out that the effect of the improvement was limited for the texts from the Large Corpus, which are not highly repetitive. Note that fib41 does not contain any maximal repeats longer than 2 without overlaps. Therefore, MR-RePair generated the

same rules as RePair in this case. It should be also be noted that MR-RePair runs at a speed comparable to the fastest implementation of RePair.

Table 2. Sizes of generated grammars and execution times of the considered algorithms. Each cell in the table represents the number of generated rules, total lengths of the right side of all of the rules except for the start variable, length of the right side of the start variable, and the total grammar size in the order from the top row. The total grammar size presented in the fourth row is the total of the values presented in the second row and the third row. The fifth row separated by a line represents the execution time for compression in seconds. The best results are highlighted in bold.

Text File		RePair Maruyama	Navarro	Prezza	Wan	Re-Pair Imp	MR-RePair
rand77.txt	Rules	41,651	41,642	41,632	41,675	41,661	**4,492**
	Total length	83,302	83,284	83,264	83,350	83,322	**46,143**
	Start variable	9	**2**	7	**2**	**2**	9
	Grammar size	83,311	83,286	83,271	83,352	83,324	**46,152**
	Execution time	0.22	0.34	2.94	0.94	2.48	**0.20**
fib41	Rules	38	38	38	38	-	38
	Total length	76	76	76	76	-	76
	Start variable	**3**	**3**	**3**	**3**	-	**3**
	Grammar size	**79**	**79**	**79**	**79**	-	**79**
	Execution time	**9.99**	14.38	48.85	85.39	-	14.88
einstein.de.txt	Rules	49,968	49,949	50,218	50,057	49,933	**21,787**
	Total length	99,936	99,898	100,436	100,114	99,866	**71,709**
	Start variable	12,734	12,665	13,419	**12,610**	12,672	12,683
	Grammar size	112,670	112,563	113,855	112,724	112,538	**84,392**
	Execution time	**9.04**	13.74	136.49	40.24	213.73	9.73
E.coli	Rules	66,664	66,757	66,660	67,368	66,739	**62,363**
	Total length	133,328	133,514	133,320	134,736	133,478	**129,138**
	Start variable	651,875	**649,660**	650,538	652,664	650,209	650,174
	Grammar size	785,203	783,174	783,858	787,400	783,687	**779,312**
	Execution time	**0.52**	0.65	9.82	2.00	11.29	0.58
bible.txt	Rules	81,193	81,169	80,999	81,229	81,282	**72,082**
	Total length	162,386	162,338	161,998	162,458	162,564	**153,266**
	Start variable	386,514	386,381	386,992	386,094	**385,989**	386,516
	Grammar size	548,900	548,719	548,990	548,552	548,553	**539,782**
	Execution time	**0.51**	0.65	8.41	1.85	11.32	0.57
world192.txt	Rules	55,552	55,798	55,409	55,473	55,437	**48,601**
	Total length	111,104	111,596	110,812	110,946	110,874	**104,060**
	Start variable	213,131	213,962	213,245	**212,647**	212,857	212,940
	Grammar size	324,235	325,558	324,057	323,593	323,731	**317,000**
	Execution time	**0.32**	0.55	4.92	1.09	6.81	0.36

6. Conclusions

In this study, we analyzed the process of RePair and revealed that the RePair algorithm replaces the most frequent pairs step by step within their corresponding most frequent maximal repeats. On the basis of this analysis, we designed MR-RePair, a novel variant of RePair. Instead of consecutively substituting the most frequent pairs, MR-RePair performs one-time substitution of the most frequent maximal repeats. Furthermore, we implemented MR-RePair and compared the sizes of its constructed grammars to those of the grammars constructed by several implementations of RePair. Through the experiments, we confirmed the effectiveness of MR-RePair 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, naming it GSDRP. We demonstrated that a lower bound of GSDRP is $\frac{1}{6}(\sqrt{6n+1}+13)$ for a given text of length n. We left improving the lower bound and showing an upper bound of GSDRP as our future work.

We estimated the effectiveness of the compression using the size of the generated grammars instead of the length of the output bits. Reducing the grammar size has important implications since the majority of the existing text algorithms applied to grammar-compressed texts, including grammar-based self indexes [21,22], edit distance computation [23], q-gram mining [24,25], and pattern matching [26–28], have time/space complexities that are dependent on the input grammar size. For instance, the compressed indexes proposed by Claude and Navarro [21,22] can be directly built on MR-RePair grammar-compressed texts. Algorithms specifically designed for *straight-line programs* (*SLPs*), which are text compressions with grammars in Chomsky normal form, can also be easily modified to work on grammars that are not in Chomsky normal form similar to MR-RePair grammars. Hence, MR-RePair serves as a base for practical improvements of these algorithms.

From the viewpoint of storing data more compactly, developing a method for encoding constructed grammars is another important issue. Implementing an efficient encoding method for MR-RePair is one of our future tasks.

Author Contributions: Conceptualization, I.F. and T.T.; methodology, I.F., T.T., Y.N., S.I., and H.B.; software, T.K.; writing—original draft preparation, I.F. and T.K.; writing—review and editing, I.F., T.T., Y.N., S.I., H.B., and T.K.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by JSPS KAKENHI Grant Numbers JP18K18002, JP17H01697, JP16H02783, JP18H04098, and JP18K11149, JST CREST Grant Number JPMJCR1402, and JST PRESTO Grant Number JPMJPR1922.

Acknowledgments: The authors would like to thank the people who provided the source codes.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Figures A1–A3 are provided to help in understanding the 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 $T = a_1 \cdots a_n$ ($a_i \in \Sigma$, $1 \leq i \leq n$) of length $|T| = n$, let g_{rp}, g_{nmr}, and g_{mr} be the sizes of G_{rp}, G_{nmr}, and G_{mr}, respectively. Let us 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, according to the proof of Theorem 4, $g_{nmr} > g_{rp}$ holds for some m and w such that $(m-1)(|w|-1) > 1$.

Figure A1 illustrates a specific example of the grammar generation process of RePair and G_{rp} for $T = (\mathtt{abcd})^7\mathtt{a}$ with $m = 2$ and $|w| = 3$. The size g_{rp} is 18 in this example. Figure A2 illustrates an example of the process of Naïve-MR-RePair and G_{nmr} for the same T. It can be noticed from the figures that the size g_{nmr} is 19, and thus $g_{nmr} > g_{rp}$ holds. As shown in Figure A2, Naïve-MR-RePair may fail to extract repetitive patterns in particular cases (such as $\mathtt{abcd}$ of $(\mathtt{abcd})^7\mathtt{a}$ in the running example). However, this problem can be solved using MR-RePair. Figure A3 illustrates an example of the process of MR-RePair and G_{mr} for the same $T = (\mathtt{abcd})^7\mathtt{a}$. The size g_{mr} is 16, which is smaller than $g_{rp} = 18$. Although the most frequent maximal repeat at the second replacement step is $v_\mathtt{a}v_\mathtt{b}v_\mathtt{c}v_\mathtt{d}v_\mathtt{a}$, MR-RePair replaces $v_\mathtt{a}v_\mathtt{b}v_\mathtt{c}v_\mathtt{d}$ with a new variable v_1, providing the additional **Step 3** in Definition 3.

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

Figure A1. Grammar generation process of RePair and its generated grammar for the text $(\mathtt{abcd})^7\mathtt{a}$. The grammar size is 18.

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

Figure A2. Grammar generation process of Naïve-MR-RePair and its generated grammar for the text $(\mathtt{abcd})^7\mathtt{a}$. The grammar size is 19.

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

Figure A3. Grammar generation process of MR-RePair and its generated grammar for the text $(\mathtt{abcd})^7\mathtt{a}$. The grammar size is 16.

References

1. Charikar, M.; Lehman, E.; Liu, D.; Ring, P.; Prabhakaran, M.; Sahai, A.; Shelat, A. The smallest grammar problem. *IEEE Trans. Inf. Theory* **2005**, *51*, 2554–2576. [CrossRef]
2. Larsson, N.J.; Moffat, A. Off-line dictionary-based compression. *Proc. IEEE* **2000**, *88*, 1722–1732. [CrossRef]
3. Claude, F.; Navarro, G. Fast and compact web graph representations. *ACM Trans. Web* **2010**, *4*, 16:1–16–31. [CrossRef]
4. González, R.; Navarro, G. Compressed text indexes with fast locate. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4580, pp. 216–227.
5. Wan, R. Browsing and Searching Compressed Documents. Ph.D. Thesis, The University of Melbourne, Melbourne, Australia, 2003.
6. Masaki, T.; Kida, T. Online Grammar Transformation Based on Re-Pair Algorithm. In Proceedings of the Data Compression Conference (2016 DCC), Snowbird, UT, USA, 29 March–1 April 2016; pp. 349–358.
7. Bille, P.; Gørtz, I.L.; Prezza, N. Space-Efficient Re-Pair Compression. In Proceedings of the Data Compression Conference (DCC 2017), Snowbird, UT, USA, 4–7 April 2017; pp. 171–180.
8. Sekine, K.; Sasakawa, H.; Yoshida, S.; Kida, T. Adaptive dictionary sharing method for Re-Pair algorithm. In Proceedings of the Data Compression Conference (DCC 2014), Snowbird, UT, USA, 26–28 March 2014; p. 425.
9. Lohrey, M.; Maneth, S.; Mennicke, R. XML tree structure compression using RePair. *Inf. Syst.* **2013**, *38*, 1150–1167. [CrossRef]
10. Tabei, Y.; Saigo, H.; Yamanishi, Y.; Puglisi, S.J. 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), San Francisco, CA, USA, 13–17 August 2016; pp. 1875–1884.
11. Navarro, G.; Russo, L.M. Re-pair Achieves High-Order Entropy. In Proceedings of the Data Compression Conference (DCC 2008), Snowbird, UT, USA, 25–27 March 2008; p. 537.
12. Ochoa, C.; Navarro, G. RePair and All Irreducible Grammars are Upper Bounded by High-Order Empirical Entropy. *IEEE Trans. Inf. Theory* **2018**, 1–5. [CrossRef]
13. Belazzougui, D.; Cunial, F.; Gagie, T.; Prezza, N.; Raffinot, M. Composite Repetition-Aware Data Structures. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2015)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9133, pp. 26–39.
14. Belazzougui, D.; Cunial, F. Fast label extraction in the CDAWG. In *International Symposium on String Processing and Information Retrieval (SPIRE 2017)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10508, pp. 161–175.
15. Belazzougui, D.; Cunial, F. Representing the Suffix Tree with the CDAWG. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*; Leibniz International Processings in Informatics; Kärkkäinen, J., Radoszewski, J., Rytter, W., Eds.; Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik: Wadern, Germany, 2017; Volume 78, pp. 7:1–7:13. [CrossRef]
16. Takagi, T.; Goto, K.; Fujishige, Y.; Inenaga, S.; Arimura, H. Linear-Size CDAWG: New Repetition-Aware Indexing and Grammar Compression. In *24th International Symposium on String Processing and Information Retrieval (SPIRE 2017)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; Volume 10508, pp. 304–316.
17. Apostolico, A.; Lonardi, S. Off-line compression by greedy textual substitution. *Proc. IEEE* **2000**, *88*, 1733–1744. [CrossRef]
18. Inenaga, S.; Funamoto, T.; Takeda, M.; Shinohara, A. Linear-time off-line text compression by longest-first substitution. In *10th International Symposium on String Processing and Information Retrieval (SPIRE 2003)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2857, pp. 137–152.
19. Nakamura, R.; Bannai, H.; Inenaga, S.; Takeda, M. Simple Linear-Time Off-Line Text Compression by Longest-First Substitution. In Proceedings of the Data Compression Conference (DCC 2007), Snowbird, UT, USA, 27–29 March 2007; pp. 123–132.
20. Gańczorz, M.; Jeż, A. Improvements on Re-Pair Grammar Compressor. In Proceedings of the Data Compression Conference (DCC 2017), Snowbird, UT, USA, 4–7 April 2017; pp. 181–190.
21. Claude, F.; Navarro, G. Self-Indexed Grammar-Based Compression. *Fundam. Inform.* **2011**, *111*, 313–337. [CrossRef]

22. Claude, F.; Navarro, G. Improved Grammar-Based Compressed Indexes. In Proceedings of the 19th International Symposium on String Processing and Information Retrieval (SPIRE 2012), Cartagena de Indias, Colombia, 21–25 October 2012; pp. 180–192. [CrossRef]
23. Gawrychowski, P. Faster Algorithm for Computing the Edit Distance between SLP-Compressed Strings. In Proceedings of the 19th International Symposium on String Processing and Information Retrieval (SPIRE 2012), Cartagena de Indias, Colombia, 21–25 October 2012; pp. 229–236.
24. Goto, K.; Bannai, H.; Inenaga, S.; Takeda, M. Fast q-gram mining on SLP compressed strings. *J. Discret. Algorithms* **2013**, *18*, 89–99. [CrossRef]
25. Bille, P.; Cording, P.H.; Gørtz, I.L. Compact Q-gram Profiling of Compressed Strings. *Theor. Comput. Sci.* **2014**, *550*, 51–58, [CrossRef]
26. Tomohiro, I.; Nishimoto, T.; Inenaga, S.; Bannai, H.; Takeda, M. Compressed automata for dictionary matching. *Theor. Comput. Sci.* **2015**, *578*, 30–41. [CrossRef]
27. Jeż, A. Faster Fully Compressed Pattern Matching by Recompression. *ACM Trans. Algorithms* **2015**, *11*, 20:1–20:43. [CrossRef]
28. Bille, P.; Cording, P.H.; Gørtz, I.L. Compressed Subsequence Matching and Packed Tree Coloring. *Algorithmica* **2017**, *77*, 336–348. [CrossRef]
29. Furuya, I.; Takagi, T.; Nakashima, Y.; Inenaga, S.; Bannai, H.; Kida, T. MR-RePair: Grammar Compression based on Maximal Repeats. In Proceedings of the Data Compression Conference (DCC 2019), Snowbird, UT, USA, 26–29 March 2019; pp. 508–517.