TCS-TR-A-12-61

TCS Technical Report

Effective Variable-Length-to-Fixed-Length Coding via a Re-Pair Algorithm

by

Satoshi Yoshida and Takuya Kida

Division of Computer Science

Report Series A

November 12, 2012

Hokkaido University
Graduate School of
Information Science and Technology

Email: kida@ist.hokudai.ac.jp

Phone: +81-011-706-7679
Fax: +81-011-706-7680

Effective Variable-Length-to-Fixed-Length Coding via a Re-Pair Algorithm

Satoshi Yoshida* Takuya Kida*

November 12, 2012

Abstract

We address the problem of improving variable-length-to-fixed-length codes (VF codes). A VF code is an encoding scheme that uses a fixed-length code, and thus, one can easily access the compressed data. However, conventional VF codes usually have an inferior compression ratio to that of variable-length codes. Although a method proposed by T. Uemura *et al.* in 2010 achieves a good compression ratio comparable to that of gzip, it is very time consuming. In this study, we propose a new VF coding method that applies a fixed-length code to the set of rules extracted by the Re-Pair algorithm, proposed by N. J. Larsson and A. Moffat in 1999. The Re-Pair algorithm is a simple off-line grammar-based compression method that has good compression-ratio performance with moderate compression speed. Moreover, we present several experimental results to show that the proposed coding is superior to the existing VF coding.

1 Introduction

Our objective is to develop an effective *variable-length-to-fixed-length code* (VF code). A VF code is a coding scheme that parses an input text into a consecutive sequence of substrings, and then, it assigns a fixed length codeword to each parsed substring. Such a code enables us to access any block randomly because the codeword boundaries are clear, which is a valuable feature from an engineering viewpoint. For example, VF codes have been reevaluated to speed up the search for compressed texts [KS08,Kid09].

Early VF codes [TW87, Ziv90, SG97, YY01], typified by *Tunstall code* [Tun67], usually have inferior compression ratios than those of other well-known compression tools such as gzip and bzip because all the codewords have equal length; therefore it is difficult to eliminate data redundancy effectively. For example, the compression ratio of the Tunstall code is usually 60% or less. Consequently, for practical applications, less attention has been paid to early VF codes.

*Graduate School of Information Science and Technology, Hokkaido University. Kita 14-jo, Nishi 9-chome, Kita-ku, 060-0814 Sapporo, Japan. {syoshid,kida}@ist.hokudai.ac.jp.

The compression ratio of a conventional VF code depends on a parse tree, which is a dictionary tree for parsing input text. To improve the compression ratios of VF codes, Uemura *et al.* [UYK+10] proposed a method that trains the parse tree by scanning the input text repeatedly. Although it achieves a good compression ratio compared to that of gzip, it requires excessive computational time. In fact, this method is approximately 100 times slower than Tunstall coding. An alternative approach is required to achieve both rapid compression and a high compression ratio.

The key to improvement lies in how rapidly we can construct a better dictionary, i.e., the method of parsing text is an important issue. This issue is common to *grammar-based compression* [KY00], a compression scheme that translates an input text into a set of grammar rules, and then encodes the rules. Each rule corresponds to a substring in the text, i.e., grammar-based compression uses a rule set as a dictionary for coding. In such a compression scheme, a smaller set of rules gives a better compression ratio. Extracting the rule set from the text is related to the method of parsing the text. Although finding the optimal grammar is NP-hard [CLL+05], several excellent heuristic algorithms have been proposed [LM99, NMWM94, MTST08]. Combining such algorithms with VF coding is a promising idea.

In this study, we propose a method to apply fixed-length coding to the rules extracted by the *Re-Pair algorithm*, proposed by Larsson and Moffat [LM99]. The Re-Pair algorithm is a simple offline grammar-based compression algorithm that iteratively replaces the most frequent bigrams in an input text into nonterminal symbols until all the bigrams become unique. Our method encodes the rules extracted by the Re-Pair algorithm with fixed-length codewords, whereas the original algorithm utilizes variable-length codewords to achieve an extremely good compression ratio. To minimize the decrease in the compression ratio compared to the original algorithm, we exploit a simple characteristic of the algorithm; the minimum output size frequently occurs in the process of repeated bigram replacement. Because all the codewords have equal length in our method, we can easily estimate the final output size for each intermediate rule set of the Re-Pair algorithm. Therefore, by preserving the best point and rewinding the rule set back to this point, we can obtain the minimum output with a reasonable cost.

The performance of the proposed method is explained by evaluation experiments for some corpus. The experimental results show that the compression ratio of the proposed method is approximately equal to that of bzip even though it uses fixed-length codewords. The compression speed is approximately the same as that of the original Re-Pair algorithm. Pattern-matching performance is also demonstrated on compressed texts, and it is confirmed that the compressed pattern matching with our VF code is faster than UNIX zgrep, which is a typical decompress-then-search method, i.e., gunzip-then-grep.

Our contributions can be summarized as follows:

- We developed a new VF coding that has superior compression ratio and compression time compared with those of the existing VF coding. The proposed

method is based on a general concept. However, it was not so obvious whether the method was really effective.

- We demonstrated experimentally that pattern matching can be performed faster on a text compressed by our method than that on the text compressed by the decompress-then-search method, which is one of the advantages of VF codes over other high-compression methods using variable-length codewords.

2 Related Studies

The *compressed matching problem* was first defined in the study of Amir and Benson [AB92] as the task of performing string matching in a compressed text without decompressing it. Many pattern matching algorithms have been proposed for each compression method [KTS+99, NR99, NT00]. However, almost all of them are not faster than *the decompress-then-search method*.

From late 90's to the beginning of 2000, practical and effective methods were proposed [SMT+00, RTT02]. These methods increased search speed approximately linearly with the compression ratio, i.e., they can perform pattern matching on compressed texts faster than that by an ordinary-search algorithm on uncompressed texts. From 2000 onward, researchers began to develop a new compression method suitable for searching. Therefore, Brisaboa *et al.* proposed a series of *Dense Codes* [BINP03, BFNE03, BFnNP10, BFL+10]. Klein and Ben-Nissan [KBN08] devised a variation of the Dense Code using the Fibonacci codes for text compression.

For VF codes, Klein and Shapira [KS08] and Kida [Kid09] independently presented VF code based on a suffix tree (*STVF code*). In the STVF code, a frequency-base-pruned suffix tree is used as a parse tree. Although the compression ratio of the STVF code is superior to that of classical VF codes such as *Tunstall code* [Tun67], it is still inferior to state-of-the-art compression methods.

On the other hand, various practical algorithms for grammar-based compression have also been devised. Bisection [KYNC00] is considered as a grammar-based compression algorithm in which the grammar belongs to the class of a straight-line program. In addition, algorithms for restricted context-free grammars (*CFG*) have been presented [KY00, NMWM94, LM99]. Among them, Re-Pair [LM99] and Sequitur [NMWM94] are particularly useful. Maruyama *et al.* [MTST08] presented a compression method based on context-sensitive grammar.

3 Re-Pair Algorithm

The Re-Pair algorithm [LM99] is a simple offline grammar-based compression method, based on CFGs. Formally, a CFG is represented by a quadruple (Σ, V, σ, R), where

Algorithm 1 The Re-Pair algorithm

Input: A text $T = T[1..n]$ and an alphabet $\Sigma = \{a_1, a_2, \cdots, a_{|\Sigma|}\}$.
Output: The binary coded sequence of the rule set R for T.

```
1: s ← |Σ| + 1, R ← ∅;
2: add (σ ⇒ T) to R; // we identify T with the right-side of σ below.
3: for all i ∈ {1, ··· , |Σ|} add (α_i ⇒ a_i) to R;
4: replace every a_i in T with α_i;
5: while the frequency of the most frequent bigram in T is not equal to 1 do
6:     (β, γ) ← the most frequent bigram;
7:     add (α_s ⇒ βγ) to R;
8:     replace all the bigrams βγ in T with α_s by the left-to-right manner;
9:     s ← s + 1
10: end while
11: encode R with an entropy encoding.
```

$\Sigma = \{a_1, a_2, \cdots, a_{|\Sigma|}\}$ is the terminal alphabet, $V = \{\alpha_1, \alpha_2, \cdots, \alpha_{|V|}\}$ is the non-terminal alphabet, $\sigma \in V$ is the start symbol, and R is a finite relation from V to $(\Sigma \cup V)^*$. Note that Σ and V are disjoint sets.

The CFG constructed by the Re-Pair algorithm consists of rules in which

$$\sigma \Rightarrow \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_m} \quad (\forall i_k \in \{1, \cdots, |\Sigma| + |V| - 1\}),$$
$$\alpha_i \Rightarrow \begin{cases} a_i & \text{if } 1 \le i \le |\Sigma|, \\ \alpha_j\alpha_k \quad (1 \le j, k < i) & \text{if } i > |\Sigma|, \end{cases}$$

and all the bigrams on the right sides of the rules are unique. Algorithm 1 shows the Re-Pair algorithm. The algorithm replaces the most frequent bigrams in the sequence with a new non-terminal symbol and adds the replacement into R as a rule. The algorithm repeats this procedure until there are no repeated bigrams, i.e., the frequencies of all bigrams are equal to one. The start symbol σ generates the obtained sequence after the repetition. Finally, the algorithm encodes the set of rules with a proper entropy encoding.

4 Proposed Method

We can easily encode the rule set with a fixed-length code so that α_i is coded by a $\lceil \lg s \rceil$-bits integer, where s denotes the number of non-terminal symbols except for the start symbol σ. However, the compression ratio is usually inferior to the original.

Our concept for improving the compression ratio is based on the observation that adding a new rule does not always improve the ratio. Although the sequence always becomes shorter by replacing bigrams with a new rule, the rule set becomes larger. Therefore, the codeword length is increased, and thus, the final output becomes larger. If we find the best value of s, we can obtain minimum output in this framework. Note that s monotonically increases by one for each repetition.

The final output is obtained as a sum of encoded rules. For the original Re-Pair algorithm, it is difficult to predetermine whether the output will become shorter prior to replacing bigrams because the algorithm employs a variable-length code.

By using a fixed-length code, we can easily estimate the output size. Each non-terminal symbol α_i is encoded into a $\lceil \lg s \rceil$-bits integer. We have to output $s - |\Sigma|$ bigrams in addition to the information of Σ as the dictionary, that is, the dictionary size is $2(s - |\Sigma|)\lceil \lg s \rceil$ bits plus some auxiliary bits for Σ. The right-hand side of the start symbol σ is encoded in $|\sigma|\lceil \lg s \rceil$ bits as the encoded sequence, where $|\sigma|$ is the length of the right-hand side of σ. Therefore, the estimated output size $f(s, |\sigma|)$ is given as follows:

$$f(s, |\sigma|) = [2(s - |\Sigma|) + |\sigma|] \cdot \lceil \lg s \rceil.$$

The term $|\Sigma|$ is an invariant factor and $|\sigma|$ depends on the number of repetitions, which corresponds to the size of the rule set R. This means that f depends only on s. In other words, the value of s controls the final output size.

By computing $f(s, |\sigma|)$ for each intermediate rule set, we can find the best value of s for f after all repetitions are completed. We denote this value s by $\hat{s}$. It is not sufficient only to compare the current value of f with the next value after replacement, because the value may fall into a local minimum.

There are two approaches to output σ with $\alpha_1, \cdots, \alpha_{\hat{s}}$. The first approach is to rewind the rule set, constructed by the Re-Pair algorithm, to the intermediate set for $\hat{s}$ and replace T. The second approach is to preserve s and T when the current minimum value of f is updated during repetitions. In the first approach, we can reduce the memory consumption required for encoding; however, we need to partially expand σ when we output it. In the second approach, we require a lot of memory; however, the output procedure is simple. Algorithm 2 shows the first approach.

The function $R(i)$ in Algorithm 2 denotes the bigram of the right-hand side of the i-th rule α_i. For example, for $(\alpha_i \Rightarrow \beta\gamma) \in R$, $R(i) = (\beta, \gamma)$. In this algorithm, we identify the rule α_i with its subscript i, and $\sigma[i]$ denotes the i-th non-terminal symbol of the right-hand side of σ.

5 Experiments

5.1 Compression Performance

We implemented our proposed algorithm, Re-pair-VF, and compared it to STVF algorithm (STVF) [Kid09], the original Re-Pair algorithm[1], gzip, and bzip2. We measured compression ratios, and compression and decompression times. We used the default options for gzip and bzip2. Re-pair-VF and STVF are variable-to-fixed

[1]We used the program implemented by S. Maruyama (`http://code.google.com/p/re-pair/`). He states it as an implementation of the original Re-Pair. However, its performance in compression ratio seems slightly inferior because a simpler binary encoding is employed.

Algorithm 2 Proposed method

Input: A text $T = T[1..n]$ and an alphabet $\Sigma = \{a_1, a_2, \cdots, a_{|\Sigma|}\}$.
Output: The binary coded sequence of the rule set R for T.

$s \leftarrow |\Sigma| + 1$, $R \leftarrow \emptyset$;
$b \leftarrow \infty$, $\hat{s} \leftarrow s$;
add $(\sigma \Rightarrow T)$ to R; // we identify T with the right-hand side of σ below.
for all $i \in \{1, \cdots, |\Sigma|\}$ add $(\alpha_i \Rightarrow a_i)$ to R;
replace every a_i in T with α_i;
while the frequency of the most frequent bigram in T is not equal to 1 **do**
    $(\beta, \gamma) \leftarrow$ the most frequent bigram;
    add $(\alpha_s \Rightarrow \beta\gamma)$ to R;
    replace all the bigrams $\beta\gamma$ in T with α_s by the left-to-right manner;
    if $f(s, |\sigma|) < b$ **then**
        $b \leftarrow f(s, |\sigma|)$;
        $\hat{s} \leftarrow s$;
    end if
    $s \leftarrow s + 1$;
end while
output $\hat{s}$ and the information of Σ;
for $i \leftarrow |\Sigma| + 1$ **to** $\hat{s}$ **do**
    output $R(i)$ with $\lceil \lg \hat{s} \rceil$ bits for each symbol;
end for
for $i \leftarrow 1$ **to** the size of the right-hand side of σ **do**
    call procedure REWIND-OUTPUT($\sigma[i]$, $\hat{s}$, R);
end for

procedure REWIND-OUTPUT(s, $\hat{s}$, R)
    if $s \leq \hat{s}$ **then**
        output s with $\lceil \lg \hat{s} \rceil$ bits;
    else
        $(\beta, \gamma) \leftarrow R(s)$;
        call procedure REWIND-OUTPUT(β, $\hat{s}$, R);
        call procedure REWIND-OUTPUT(γ, $\hat{s}$, R);
    end if
end procedure

Table 1: Text files used in our experiments.

Texts	size (byte)	$\|\Sigma\|$	Contents
Dazai.utf.txt	7,268,943	141	Japanese texts (encoded by UTF-8)
DBLP2003.xml	90,510,236	97	XML data
GBHTG119.dna	87,173,787	4	DNA sequences
Reuters21578.txt	18,805,335	103	English texts

length encodings, whereas Re-Pair, gzip, and bzip2 are variable-to-variable length encodings. Our program is written in C++ and compiled by g++ version 3.4. We performed the experiments on a workstation equipped with an Intel Xeon (R) 3GHz CPU with 12GB RAM, operating Red Hat Enterprise Linux ES Release 4.

We used XML data, DNA data, English texts, and Japanese texts in our experiments (see Table 1 for details). "Dazai.utf.txt" is the complete works of Osamu Dazai[2]; the text is written in Japanese and encoded by UTF-8. "DBLP2003.xml" consists of all the 2003 data from dblp20040213.xml[3]. "GBHTG119.dna" is a collection of DNA sequences from GenBank[4]; we eliminated all meta data, spaces, and line feeds. "Reuters21578.txt" (distribution 1.0)[5] is a sample collection of English texts.

Table 2 lists the compression ratios for each file and the compression method measured as (compressed file size)/(original file size). As shown in the table, Re-pair-VF is better than STVF and gzip for natural language texts. In particular, Re-pair-VF is approximately 1.3 times better than gzip, which is almost the same as Re-Pair.

Table 3 lists the compression times. The results show that Re-pair-VF is approximately two times faster than STVF, which is almost the same as Re-Pair. This means that $\hat{s}$ defined in Section 4, can be selected with almost no increase in time.

Table 4 lists the decompression times. Re-pair-VF is as fast as STVF and is approximately three times faster than bzip2.

5.2 Pattern Matching Performance

We implemented pattern matching algorithms for Re-pair-VF and STVF according to the way of Kida *et al.* in 2003 [KMS+03]. to demonstrate Re-pair-VF's pattern matching performance on compressed texts. We used UNIX zgrep for pattern matching on the text compressed by gzip. We chose patterns with lengths 5–50 characters from the text. We measured pattern matching times for 50 patterns for each length and calculated the average. In this study, we only mention the results on Reuters21578.txt because of space constraints. Table 5 lists the results of matching throughput performance averaged, i.e., (original text length)/(pattern matching time). Re-pair-VF is the fastest among all the methods, and notably, it is 1.7–2.1 times faster than zgrep.

[2] `http://j-texts.com/`

[3] `http://www.informatik.uni-trier.de/~ley/db/`

[4] `http://www.ncbi.nlm.nih.gov/genbank/`

[5] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

Table 2: Compression ratios in percentage.

	Re-pair-VF	Re-Pair	STVF	gzip	bzip2
Dazai.utf.txt	25.86	26.02	39.88	33.41	22.93
DBLP2003.xml	13.67	12.97	28.30	17.30	11.26
GBHTG119.dna	28.02	31.72	24.93	28.23	26.00
Reuters21578.txt	27.96	27.84	46.26	36.98	25.80

Table 3: Compression times in seconds for each data.

	Re-pair-VF	Re-Pair	STVF	gzip	bzip2
Dazai.utf.txt	4.444	4.408	8.886	0.79	1.134
DBLP2003.xml	52.976	54.468	121.192	2.82	23.221
GBHTG119.dna	57.272	57.134	131.924	18.19	16.773
Reuters21578.txt	13.536	13.534	24.444	1.372	3.344

Table 4: Decompression times in seconds for each data.

	Re-pair-VF	Re-Pair	STVF	gzip	bzip2
Dazai.utf.txt	0.128	0.132	0.568	0.079	0.374
DBLP2003.xml	2.112	1.932	1.944	0.666	3.342
GBHTG119.dna	1.800	2.200	1.456	0.910	5.513
Reuters21578.txt	0.392	0.438	0.750	0.203	1.012

6 Conclusions

In this study, we proposed a new VF coding based on the Re-Pair algorithm. The experimental results demonstrated that the proposed coding is superior to the existing VF codes with respect to compression ratio and compression time. Even though the Re-pair-VF algorithm uses fixed-length codewords, it shows a good compression performance, similar to bzip2. Moreover, we showed that pattern matching on a text compressed by the proposed coding can be performed much faster than an ordinary decompress-then-search approach such as zgrep.

Although compared to STVF, the proposed coding improves compression speed, it is much slower than gzip and bzip2. Therefore, further improvements are necessary. In the future, we intend to develop VF codes that employ online grammar-based compression methods such as SEQUITUR [NMWM94].

References

[AB92] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *DCC*, pages 279–288, 1992.

Table 5: Pattern matching throughput on Reuters21578 in mebibyte per second.

Pattern Length	Re-pair-VF	STVF	gzip
5	182.63	124.37	85.73
10	181.89	123.68	88.00
15	169.51	124.37	88.35
20	173.78	123.68	88.61
25	171.13	123.34	89.49
30	171.78	122.50	88.35
35	170.48	121.01	87.91
40	164.84	119.24	89.05
45	157.59	116.15	88.35
50	150.45	112.79	87.91

[BFL+10] N. R. Brisaboa, A. Fariña, J.-R. López, G. Navarro, and E. R. Lopez. A new searchable variable-to-variable compressor. In *DCC*, pages 199–208, 2010.

[BFNE03] N. R. Brisaboa, A. Fariña, G. Navarro, and M. F. Esteller. (S, C)-dense coding: An optimized compression code for natural language text databases. In *SPIRE*, pages 122–136, 2003.

[BFnNP10] N. R. Brisaboa, A. Fariña, G. Navarro, and J. Paramá. Dynamic lightweight text compression. *ACM Trans. Inf. Syst.*, 28:10:1–10:32, July 2010.

[BINP03] N. R. Brisaboa, E. L. Iglesias, G. Navarro, and J. R. Paramá. An efficient compression code for text databases. In *ECIR*, pages 468–481, 2003.

[CLL+05] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *Information Theory, IEEE Transactions on*, 51(7):2554–2576, 2005.

[KBN08] S. T. Klein and M. K. Ben-Nissan. Using fibonacci compression codes as alternatives to dense codes. In *DCC*, pages 472–481, 2008.

[Kid09] T. Kida. Suffix tree based VF-coding for compressed pattern matching. In *DCC*, page 449, Mar. 2009.

[KMS+03] T. Kida, T. Matsumoto, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003.

[KS08] S. T. Klein and D. Shapira. Improved variable-to-fixed length codes. In *SPIRE*, pages 39–50, 2008.

[KTS+99] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. In *CPM*, LNCS 1645, pages 1–13, 1999.

[KY00] J. C. Kieffer and E.-H. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Trans. on Inform. Theory*, 46(3):737–754, 2000.

[KYNC00] J. C. Kieffer, E.-H. Yang, G. Nelson, and P. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Trans. Inform. Theory*, 46(4):1227–1245, 2000.

[LM99] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *DCC*, pages 296–305. IEEE Computer Society, 1999.

[MTST08] S. Maruyama, Y. Tanaka, H. Sakamoto, and M. Takeda. Context-sensitive grammar transform: Compression and pattern matching. In *SPIRE*, pages 27–38, Nov. 2008.

[NMWM94] C. Nevill-Manning, I. Witten, and D. Maulsby. Compression by induction of hierarchical grammars. In *DCC*, pages 244–253. IEEE, 1994.

[NR99] G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *CPM*, LNCS 1645, pages 14–36, 1999.

[NT00] G. Navarro and J. Tarhio. Boyer-Moore string matching over Ziv-Lempel compressed text. In *CPM*, LNCS 1848, pages 166–180, 2000.

[RTT02] J. Rautio, J. Tanninen, and J. Tarhio. String matching with stopper encoding and code splitting. In *CPM*, LNCS 2373, pages 42–52, 2002.

[SG97] S. A. Savari and R. G. Gallager. Generalized tunstall codes for sources with memory. *IEEE Transactions on Information Theory*, 43(2):658–668, Mar. 1997.

[SMT+00] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa. A Boyer-Moore type algorithm for compressed pattern matching. In *CPM*, LNCS 1848, pages 181–194, 2000.

[Tun67] B. P. Tunstall. *Synthesis of noiseless compression codes.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1967.

[TW87] T. J. Tjalkens and F. M. J. Willems. Variable to fixed-length codes for markov sources. *IEEE Transactions on Information Theory*, IT-33(2):246–257, March 1987.

[UYK+10] T. Uemura, S. Yoshida, T. Kida, T. Asai, and S. Okamoto. Training parse trees for efficient VF coding. In *SPIRE*, pages 179–184, 2010.

[YY01] H. Yamamoto and H. Yokoo. Average-sense optimality and competitive optimality for almost instantaneous VF codes. *IEEE Transactions on Information Theory*, 47(6):2174–2184, Sep. 2001.

[Ziv90] J. Ziv. Variable-to-fixed length codes are better than fixed-to-variable length codes for markov sources. *IEEE Transactions on Information Theory*, 36(4):861–863, July 1990.