テキスト圧縮に対する効率よい可変長-固定長符号化アルゴリズム

Efficient Variable-to-Fixed Length Coding Algorithms for Text Compression

吉田 諭史 ♥

Satoshi YOSHIDA

Abstract

Data compression is a technique for reducing the storage space and the cost of transferring a large amount of data, using redundancy hidden in the data. There are two categories for data compression: *lossless compression* and *lossy compression*. The former guarantees that the original data can be completely restored from the compressed data. While in the latter category, due to distortion, the original data can not be reconstructed precisely from the compressed data. We generally use lossless compression methods for string data, often referred to as *texts*, because we need to read them without error. We therefore focus on lossless compression in this thesis.

Performance of data compression methods was classically evaluated from their compression ratio, because to reduce the space was the most significant matter in an era when hard disk drives were extremely expensive. Compressing data also allows us to reduce transfer costs. A number of data compression algorithms are devised to gain better compression ratio [24, 25].

The significant criteria have changed into speeds for compression and decompression because hard disk drives with large capacity and high speed data transfer have been available in low price[1]. Therefore, compression methods that emphasize high speed processing rather than compression ratio have attracted attention. Particularly, gzip, which is an elder compression tool based on Lempel-Ziv method [32, 33], is still used due to its well-balancedness among the three criteria, which are compression ratio, compression speed, and decompression speed, in spite of its milder compression ratio compared to the state of the art compression methods.

Today, said to be the era of big data, a huge amount of data that were difficult to manage in past is available. Such data include contents of social networking services such as Twitter and Facebook, access logs of web servers, and sensor data generated by global positioning system receivers. Performing fast data analysis on such massive data is strongly required. Since such data are massive but individual data are not significant, they are usually discarded after performing event processing or saving summarized meta information at present. In this case, we have to determine what information is necessary in advance. In other words, we can not perform data analysis on past data, which is previously unexpected. To reuse past data, all the data must be preserved.

To reuse a huge amount of data stored in secondary storage, I/O speeds are bottlenecks. Such a communication-speed problem can be relieved if we transfer only compressed data through the communication channel and furthermore can perform every necessary processes, such as string search and access to any position, on the compressed data itself without decompression. From this viewpoint, pattern matching and data mining on compressed data have been gathering a great deal of attention since the late 1990s [1, 2, 8, 10, 12, 28].

Development of compression algorithm is currently in the mainstream of data compression field but many of them are not adequate for that criterion. The algorithms employing variable length codewords succeeded to achieve an extremely good compression ratio, but the boundaries between codewords are not obvious without a special processing. To treat such compressed data, extracting codewords with dictionary from the beginning of them is required because all the codewords do not have the same length.

On the contrary, *Variable-to-Fixed-length coding*, which is abbreviated as *VF coding*, is promising for our demand. VF coding is a coding scheme that segments an input text into a consecutive sequence of substrings, called *phrases*, and then assigns a fixed length codeword to each substring. Boundaries between codewords of VF coding are obvious because all of them have the same length. Therefore, we can realize "accessible data compression" by VF coding. However, they were seldom used in practice. Although it is theoretically known that VF coding achieves the same compression ratios as compression methods that employ variable length codes, a VF coding algorithm that achieves the compression ratio did not exist so far. For example, Shibata *et al.* [26] evaluated

♥ 北海道大学大学院情報科学研究科．平成２６年３月 博士課程修了，博士（情報科学）． s.yoshida1044@gmail.com

[1] From 1980s to 2010s, the cost of hard disk drives per unit capacity became 10 millionth[2]. In 1980s, computers were connected to the network via telephone line with 56 kbps or leased line. In 2010s, high speed data transfer with 100 Mbps is available.

[2] http://www.mkomo.com/cost-per-gigabyte

compression methods from the viewpoint of faster searching on the compressed data, and they rediscovered Byte Pair Encoding (BPE) [7], which is a kind of VF coding, but it has mild compression ratio about 50% on natural language text at most, while gzip usually achieves better than 40%.

The objective of this study is to improve performances of VF coding methods. We developed high-level and well-balanced VF coding algorithms for the four criteria of compression ratio, compression speed, decompression speed, and processing on the compressed data. It is achieved by improving compression ratio, compression speed, and decompression speed of VF coding beyond the level of typical ones. As mentioned above, gzip is a popular compression method for its well-balancedness. A goal of this study is therefore to design a VF coding algorithm that achieves good compression performance as gzip.

To improve the performance of VF coding is a difficult problem because they employ fixed length codewords. The compression ratio of a data compression method generally depends on the set of strings, called *dictionary*, used during compression. Therefore, the problem of improving the compression ratio of it is the one of how to generate the optimal dictionary. However, this problem can not be solved in practical time because constructing the optimal dictionary is known to be an NP-Hard problem. Hence, the disputed point is how to construct a better dictionary with a greedy method.

The organization of this thesis as follows. In Chapter 2, we focus on basic notion and definition. In Chapter 3, we discuss an improvement of *Almost Instantaneous VF coding* (*AIVF coding*) [30] proposed by Yamamoto and Yokoo in 2001. Conventional VF coding methods originated by *Tunstall coding* [29] use tree structures called parse trees as dictionaries. Although AIVF coding achieves better compression ratio than Tunstall coding by using multiple parse trees, it is known that it requires large amount of time and space during compression. We propose an improved method by constructing an integrated parse tree used in AIVF coding and then simulate encoding of AIVF coding on it. Moreover, we give theoretical analysis of upper and lower bounds of the number of nodes in the integrated parse tree and the number of nodes reduced by the integration.

In Chapter 4, we discuss a method of brushing up a parse tree constructed by existent VF coding methods. We propose a method that repeatedly deletes useless nodes that are in the parse tree to add nodes that are expected to be useful but not in the parse tree with reading the input text. The method constructs a parse tree that achieves a fairly good compression ratio. We experimentally show that application of this method to a parse tree generated by *Suffix Tree-based VF coding* (*STVF coding*) [31] yields better compression ratios than those of gzip.

In Chapter 5, we show how to realize a VF coding method by combining grammar-based compression [13] and fixed length codeword. Grammar-based compression is a compression method that models the input text by a grammar that generates it to encode the grammar. We give a VF coding method that combines *Re-Pair algorithm* [17] proposed by Larsson and Moffat in 2000 and fixed length coding. Re-Pair algorithm is a compression method based on a simple grammar which achieves extremely good compression ratio. We introduce a numerical formula calculating the total amount of dictionary and compressed data in order to determine which rule in the grammar generated by Re-Pair algorithm should be in the dictionary. This method is beyond the framework of conventional VF coding methods with parse trees. It achieves better compression ratio than gzip by 20% or more and faster compression than STVF coding by a factor of 40 on natural language text.

In Chapter 6, we give practical techniques to apply VF coding methods to large texts. We propose two methods: (i) compressing large texts by block division and dictionary sharing and (ii) faster access to arbitrary position specified in the original text on compressed text. The former is a technique to reduce memory usage by dividing the input text into fixed length blocks and then compress each blocks. We improve the compression ratio by sharing a part of dictionaries for all blocks. The latter is a problem of identifying the position on compressed data corresponding to specified one on original text. To solve this, decompressing the compressed data from the beginning is generally required. We propose a faster method for this problem by having a bit sequence of n bits and its fully indexable dictionary where n denotes the length of the input text. We experimentally show that the proposed method works faster than FOLCA [18] proposed by Maruyama *et al.* in 2013 by a factor of 10.

Finally, we conclude this thesis and discuss future works in Chapter 7.

Related studies

We aimed to develop a data compression scheme, which would allow us to process compressed data with ease. This issue arose from studies of the *compressed pattern matching problem*.

The compressed pattern matching problem was first defined in a study by Amir and Benson [1] as the task of performing string matching in a compressed text without its de-

compression. Many pattern matching algorithms have been proposed for each specific compression method [11, 20, 21]. However, most of them are no faster than *the decompress-then-search method*.

Practical and effective methods were proposed from late 1990s until the beginning of 2000 [23, 27]. These methods increased the search speed and they had an approximately linear relationship to the compression ratio, i.e., they could perform pattern matching in compressed texts faster than ordinary search algorithms using uncompressed texts.

After 2000, researchers began to develop a new compression method that was suitable for searching. Thus, Brisaboa *et al.* proposed a series of *Dense Codes* [3–6]. Dense codes parse an input text using a morphological analysis tool before encoding it with byte-oriented codewords. Klein and Ben-Nissan [15] devised a variation of the Dense Code by using Fibonacci codes for text compression. Although Dense Codes work well for natural language texts that all words are separated by spaces such as English texts, they are not effective on texts that each word can not be easily extracted such as Japanese texts or DNA data.

For VF coding methods, Klein and Shapira [16] and Kida [9] independently presented a VF coding method based on a suffix tree (STVF coding[3]). A frequency-base-pruned suffix tree is used as a parse tree in the STVF coding. STVF coding is also suitable for searching because it uses a static dictionary and the codeword boundaries are obvious (see Section 2.5 for compressed pattern matching on VF coding). The compression ratio of STVF coding is superior to that of classical VF coding methods such as Tunstall coding, but it is still inferior to state-of-the-art compression methods. Some experimental comparisons of Dense Codes, VF codes, and gzip were presented in [31].

Various practical algorithms have also been developed for grammar-based compression. Bisection [14] is a grammar-based compression algorithm where the grammar belongs to the class of a straight-line program. Compression algorithms have also been presented for restricted context-free grammars [13, 17, 22]. For example, Re-Pair [17] and Sequitur [22] are particularly useful because of their good compression ratios.

Maruyama *et al.* [19] presented an excellent compression method based on context-sensitive grammar, known as BPEX[4]. This method can be viewed as an extension of Byte Pair Encoding (BPE) [7], which is a restricted version of the Re-Pair algorithm. BPEX improves the compression ratio compared with BPE and its pattern matching performance is extremely good. However, the compression speed of BPEX is slow and it is difficult to decode or perform pattern matching directly from the middle of the compressed data because any codeword in BPEX-compressed data depends on the preceding codeword.

[3] The method of [16] is referred to as DynC in their paper, where the encoding algorithm is slightly different from that used by [9].

[4] "BPEX" is simply the name of the program written by Maruyama but we refer to it as the name of their method.

[文献]

[1] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In James A. Storer and Martin Cohn, editors, *Proceedings, Data Compression Conference, March 24–27, 1992, Snowbird, Utah*, pages 279–288. IEEE Computer Society, 1992.

[2] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Computer and System Sciences*, 52(2):299–307, 1996.

[3] N. Brisaboa, A. Fariña, J.-R. López, G. Navarro, and E. Lopez. A new searchable variable-to-variable compressor. In James A. Storer and Michael W. Marcellin, editors, *Proceedings, Data Compression Conference, 24–26 March 2010, Snowbird, Utah*, pages 199–208. IEEE Computer Society, 2010.

[4] N. Brisaboa, A. Fariña, G. Navarro, and M. Esteller. (S, C)-dense coding: An optimized compression code for natural language text databases. In Mario A. Nascimento, Edleno S. de Moura, and Arlindo L. Oliveira, editors, *String Processing and Information Retrieval, 10th International Symposium, SPIRE 2003, Manaus, Brazil, October 2003, Proceedings*, volume 2857 of *Lecture Notes in Computer Science*, pages 122–136. Springer-Verlag, 2003.

[5] N. Brisaboa, A. Fariña, G. Navarro, and J. Paramá. Dynamic lightweight text compression. *ACM Transactions on Information Systems*, 28:10:1–10:32, July 2010.

[6] N. Brisaboa, E. Iglesias, G. Navarro, and J. Paramá. An efficient compression code for text databases. In Fabrisio Sebastiani, editor, *Advances in Information Retrieval, 25th European Conference on IR Research, ECIR 2003, Pisa, Italy, April 2003, Proceedings*, volume 2633 of *Lecture Notes in Computer Science*, pages 468–481. Springer-Verlag, 2003.

[7] P. Gage. A new algorithm for data compression. *C Users Journal*, 12:23–38, February 1994.

[8] K. Goto, H. Bannai, S. Inenaga, and M. Takeda. Fast q-gram mining on SLP compressed strings. *Journal of Discrete Algorithms*, 18:89–99, January 2013.

[9] T. Kida. Suffix tree based VF-coding for compressed pattern matching. In James A. Storer and Michael W. Marcellin, editors, *Proceedings, Data Compression Conference, 15–19 March 2009, Snowbird, Utah*, page 449. IEEE Computer Society, Mar. 2009.

[10] 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.

[11] T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. In Maxime Crochemore and Mike Paterson, editors, *Combinatorial Pattern Matching, 10th Annual Symposium, CPM99, Warwick University, UK, July 1999, Proceedings*, volume 1645 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1999.

[12] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In James A. Storer and Martin Cohn, editors, *Proceedings, Data Compression Conference, March 30–April 1, 1998, Snowbird, Utah*, pages 103–112. IEEE Computer Society, 1998.

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

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

[15] S. Klein and M. Ben-Nissan. Using Fibonacci compression codes as alternatives to dense codes. In James A. Storer and Michael W. Marcellin, editors, *Proceedings, Data Compression Conference, 25–27 March 2008, Snowbird, Utah*, pages 472–481. IEEE Computer Society, 2008.

[16] S. Klein and D. Shapira. Improved variable-to-fixed length codes. In Amihood Amir, Andrew Turpin, and Alistair Moffat, editors, *String Processing and Information Retrieval, 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 2008, Proceedings*, volume 5280 of *Lecture Notes in Computer Science*, pages 39–50. Springer-Verlag, 2008.

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

[18] S. Maruyama, Y. Tabei, H. Sakamoto, and K. Sadakane. Fully-online grammar compression. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval, 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 218–229. Springer International Publishing, 2013.

[19] S. Maruyama, Y. Tanaka, H. Sakamoto, and M. Takeda. Context-sensitive grammar transform: Compression and pattern matching. In Amihood Amir, Andrew Turpin, and Alistair Moffat, editors, *String Processing and Information Retrieval, 15th International Symposium, SPIRE 2008, Melbourne, Australia, November 2008, Proceedings*, volume 5280 of *Lecture Notes in Computer Science*, pages 27–38. Springer-Verlag, 2008.

[20] G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In Maxime Crochemore and Mike Paterson, editors, *Combinatorial Pattern Matching, 10th Annual Symposium, CPM99, Warwick University, UK, July 1999, Proceedings*, volume 1645 of *Lecture Notes in Computer Science*, pages 14–36. Springer-Verlag, 1999.

[21] G. Navarro and J. Tarhio. Boyer-Moore string matching over Ziv-Lempel compressed text. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching, 11th Annual Symbosium, CPM 2000, Montreal, Canada, June 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 166–180. Springer-Verlag, 2000.

[22] C. Nevill-Manning, I. Witten, and D. Maulsby. Compression by induction of hierarchical grammars. In James A. Storer and Martin Cohn, editors, *Proceedings, Data Compression Conference, March 29–31, 1994, Snowbird, Utah*, pages 244–253. IEEE Computer Society, 1994.

[23] J. Rautio, J. Tanninen, and J. Tarhio. String matching with stopper encoding and code splitting. In Alberto Apostolico and Masayuki Takeda, editors, *Combinatorial Pattern Matching, 13th Annual Symposium, CPM 2002, Fukuoka, Japan, July 2002, Proceedings*, volume 2373 of *Lecture Notes in Computer Science*, pages 42–52. Springer-Verlag, 2002.

[24] D. Salomon and G. Motta. *Handbook of data compression*. Springer-Verlag, 5th edition, 2010.

[25] K. Sayood. *Introduction to data compression*. Morgan Kaufmann, 4th edition, 2012.

[26] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Speeding up pattern matching by text compression. In Giancarlo Bongiovanni, Giorgio Gambosi, and Rossella Petreschi, editors, *Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, Proceed-*

ings, volume 1767 of *Lecture Notes in Computer Science*, pages 306–315. Springer-Verlag, 2000.

[27] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa. A Boyer-Moore type algorithm for compressed pattern matching. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching, 11th Annual Symbosium, CPM 2000, Montreal, Canada, June 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 181–194. Springer-Verlag, 2000.

[28] M. Takeda, Y. Shibata, T. Matsumoto, T. Kida, A. Shinohara, S. Fukamachi, T. Shinohara, and S. Arikawa. Speeding up string pattern matching by text compression: The dawn of a new era. *Transactions of Information Processing Society of Japan*, 42(3):370–384, 2001.

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

[30] 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.

[31] S. Yoshida, T. Uemura, T. Kida, T. Asai, and S. Okamoto. Improving parse trees for efficient variable-to-fixed length codes. *Journal of Information Processing*, 20(1):238–249, 2012.

[32] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.

[33] J. Ziv and A. Lempel. Compression of individual sequences via variable length coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.