Byte pair encoding: a text compression scheme that accelerates pattern matching

Yusuke Shibata[†] Takuya Kida[†] Shuichi Fukamachi[‡]
Masayuki Takeda[†] Ayumi Shinohara[†] Takeshi Shinohara[‡]
Setsuo Arikawa[†]

† Dept. of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
{ yusuke, kida, takeda, ayumi, arikawa} @i.kyushu-u.ac.jp

‡ Dept. of Artificial Intelligence, Kyushu Institute of Technology, Iizuka 320-8520, Japan
{ fukamati, shino} @ai.kyutech.ac.jp

Abstract

Byte pair encoding (BPE) is a simple universal text compression scheme. Decompression is very fast and requires small work space. Moreover, it is easy to decompress an arbitrary part of the original text. However, it has not been so popular since the compression is rather slow and the compression ratio is not as good as other methods such as Lempel-Ziv type compression.

In this paper, we bring out a potential advantage of BPE compression. We show that it is very suitable from a practical view point of *compressed pattern matching*, where the goal is to find a pattern directly in compressed text without decompressing it explicitly. We compare running times to find a pattern in (1) BPE compressed files, (2) Lempel-Ziv-Welch compressed files, and (3) original text files, in various situations. Experimental results show that pattern matching in BPE compressed text is even faster than matching in the original text. Thus the BPE compression reduces not only the disk space but also the searching time.

1 Introduction

Pattern matching is one of the most fundamental operations in string processing. The problem is to find all occurrences of a given pattern in a given text. It becomes more important to find a pattern in text files efficiently, as files become large. A lot of classical or advanced pattern matching algorithms have been proposed (see [13, 8]).

In case that we have enough storage devices and the texts to be stored are almost static, we may construct such data structures as inverted-files [18], suffix-trees [40, 30, 39], or suffix-arrays [29] in order to find a pattern very quickly. On the other hand, if the available storage devices are limited, we cannot exploit such data structures.

Table 1: History of compressed pattern matching.

compression method	compressed pattern matching algorithms
run-length	T. Eilam-Tzoreff and U. Vishkin [16]
run-length (two dim.)	A. Amir, G. M. Landau, and U. Vishkin [7]; A. Amir and G. Benson [3, 4]; A. Amir, G. Benson, and M. Farach [6]
LZ77	M. Farach and M. Thorup [17]; L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter [21]
LZW	A. Amir, G. Benson, and M. Farach [5]; T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa [26]; T. Kida, M. Takeda, A. Shinohara, and S. Arikawa [25]; G. Navarro and M. Raffinot [33]
straight-line program	M. Karpinski, W. Rytter, and A. Shinohara [24]; M. Miyazaki, A. Shinohara, and M. Takeda [32]
Huffman	S. Fukamachi, T. Shinohara, and M. Takeda [19]; M. Miyazaki, S. Fukamachi, M. Takeda, and T. Shinohara [31]
finite state encoding	M. Takeda [38]
word based encoding	E. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates [14, ?]
pattern substitution	U. Manber [28]; Y. Shibata [35]
antidictionary based	Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa [36]

Although the prices of storage devices are coming down year by year, the data to be stored become growing up even more rapidly. Typical examples of such situations are mobile devices such as notebook computers and personal digital assistants (PDAs), where a user is often eager to stuff any available information up to a possible limitation, and the data are frequently rewritten.

Text compression is another old but very important research topic in computer science. The aim is to reduce space requirement of text files. Considerable amount of compression methods have been proposed, and the performance comparisons of these methods have been executed mainly against the following two criteria: the compression ratio and the compression/decompression time (see [34]).

However in general, if a text is stored in a compressed form, the pattern matching problem becomes hard. In order to find a pattern in compressed files, either we once extract the original file and then find a pattern, or we have to develop more clever technique to find a pattern directly in the compressed files. From these backgrounds, *compressed pattern matching* problem attracts special concern recently, where the goal is to find a pattern in a compressed text without decompressing it. Various compressed pattern matching algorithms have been proposed depending on underlying compression methods. See Table 1 for an (incomplete) history of the problem.

From the practical view points, one important goal of compressed pattern matching is to achieve a faster search in compressed file than an ordinary search in original file. Therefore we have to introduce now the third criterion to estimate the performance of compression methods: *the time to find a pattern in compressed files directly.* Since the searching time is the sum of file I/O time and CPU time of compressed pattern matching, it depends on the data transfer ratio. When the data transfer is slow, such as in network environments, the CPU time is negligible compared with the file I/O time. Even a naive method of decompression followed by a search can be faster than

an ordinary search in original file. Thus it is important to improve the compression ratio in such case.

On the other hand, when the data transfer is relatively fast, in such situations as a workstation with local disk storage or a notebook personal computer, the CPU time of compressed pattern matching is also an important factor. In the case of searching in compressed file encoded by adaptive compression methods, such as Lempel-Ziv based algorithms, the CPU time may become a critical factor, since we need to keep track of the compression mechanism.

Byte pair encoding (BPE, in short) [20] is a simple universal text compression scheme based on the pattern-substitution [22]. The basic operation of the compression is to substitute a single character which did not appear in the text for a pair of consecutive two characters which frequently appears in the text. This operation will be repeated until either all characters are used up or no pair of consecutive two characters appears frequently. Thus the compressed text consists of two parts: the substitution table, and the substituted text. Decompression is very fast and requires small work space. Moreover, partial decompression is possible, since the compression depends only on the substitution. This is a big advantage of BPE in comparison with adaptive dictionary methods. Despite such advantages, the BPE method was seldom used until now. The reason is mainly for the following two disadvantages: the compression is terribly slow, and the compression ratio is not as good as other methods such as Lempel-Ziv type compression.

In this paper, we pull out a potential advantage of BPE compression. We show that BPE is very suitable from a practical view point of compressed pattern matching. Manber [28] also introduced a similar idea, based on a little simpler compression scheme. However since its compression ratio is not so good and is about 70% for typical English texts, the improvement of the searching time cannot be better than this rate. The compression ratio of BPE is less than 60%.

We estimated the searching time to find a pattern in (1) BPE compressed files, (2) LZW compressed files, and (3) original text in three different situations: a workstation with remote storage connected via network, a workstation with local disk storage, and a notebook personal computer. Experimental results show that pattern matching in BPE compressed text is approximately 1.6$\sim$1.9 times faster than matching in the original text over all these situations. Searching in BPE compressed text is faster than searching in LZW compressed text except for the situation of remote storage. Thus the BPE compression reduces not only the disk space but also the searching time.

Moreover, we give a faster compression method based on a one-way sequential transducer. It drastically reduces the compression time with minor sacrifices in the compression ratio.

The rest of the paper is organized as follows. In Section 2, we introduce the byte pair encoding scheme, discuss its implementation, and estimate its performance in comparison with `Compress` and `Gzip`. Section 3 is devoted to compressed pattern matching in BPE compressed files, where we have two implementations using the automata and the bit-parallel approaches. In Section 4, we report our experimental results to compare practical behaviors of these algorithms performed in the three

situations. Section 5 concludes the discussion and explains some of future works.

2 Byte pair encoding

In this section we describe the byte pair encoding scheme, discuss its implementation, and then estimate the performance of this compression scheme in comparison with widely-known compression tools `Compress` and `Gzip`.

2.1 Compression algorithm

The BPE compression is a simple version of pattern-substitution method [20]. It utilizes the character codes which did not appear in the text to represent frequently occurring strings. The compression algorithm repeats the following task until all character codes are used up or no frequent pairs appear in the text.

> Find the most frequent pair of consecutive two character codes in the text, and then substitute an unused code for the occurrences of the pair.

For example, suppose that the text to be compressed is

$$T_0 = \texttt{ABABCDEBDEFABDEABC}.$$

Since the most frequent pair is `AB`, we substitute a code `G` for `AB`, and obtain the new text

$$T_1 = \texttt{GGCDEBDEFGDEGC}.$$

Then the most frequent pair is `DE`, and we substitute a code `H` for it to obtain

$$T_2 = \texttt{GGCHBHFGHGC}.$$

By substituting a code `I` for `GC`, we obtain

$$T_3 = \texttt{GIHBHFGHI}.$$

The text length is shorten from $|T_0| = 18$ to $|T_3| = 9$. Instead we have to encode the substitution pairs

$$\texttt{AB} \rightarrow \texttt{G}, \qquad \texttt{DE} \rightarrow \texttt{H}, \qquad \texttt{GC} \rightarrow \texttt{I}.$$

More precisely, we encode a table which stores for every character code what it represents. Note that a character code can represent either (1) the character itself, (2) a code-pair, or (3) nothing. Let us call such table *substitution table*. In practical implementations, an original text file is split into a number of fixed-size blocks, and the compression algorithm is then applied to each block. Therefore a substitution table is encoded for each block.

2.2 Speeding up of compression

In [20] an implementation of BPE compression is presented, which seems quite simple. A text is stored simply in a one-dimensional array, and rewritten in linear time proportional to its size for every iteration that corresponds to one substitution rule. Such naive implementation requires at least $O(\ell N)$ time, where N is the original text length and ℓ is the number of allocated character codes. The set of pairs of consecutive two character codes are stored in a hash table together with their frequencies. Finding the most frequent pair is implemented simply as a linear search through the hash table. The frequencies will be updated during the substitutions. The update requires only linear time in the number of executions of substitutions, assuming that the frequency value of any pair can be obtained and updated in constant time by the hashing technique. Total number of substitutions is at most N. Thus the overall time complexity of the algorithm is $O(\ell N)$.

The time complexity can be reduced to $O(\ell + N)$ if we represent a text as a linked list of characters, and prepare an index to the occurring positions for all byte-pairs. However, this improvement did not reduce the compression time in practice, because the index update operations were relatively heavy.

Thus, we decided to reduce the compression time with minor sacrifices in the compression ratio. One possible approach is to compute a substitution table only from a small part of the text, e.g., the first block, and to use the same table for encoding the whole text. The disadvantage is that the compression ratio decreases when the frequency distribution of character pairs varies depending on parts of the text. The advantage is that a substitution table is encoded only once. This is a desirable property from a practical view point of compressed pattern matching in the sense that we have to perform only once any task which depends on the substitution table as a preprocessing since it never changes.

Fast execution of substitutions according to the table is achieved by an efficient multiple key replacement technique [9, 37], in which a one-way sequential transducer is built from a given collection of replacement pairs which performs the task in only one pass through a text. When the keys have overlaps, it replace the longest possible first occurring key. The running speed is surprisingly fast, and it has been exploited as `REPLACE` command in a general purpose text database management system SIGMA [10].

2.3 Comparison with `Compress` and `Gzip`

We compare the performance of BPE with those of `Compress` and `Gzip`.

We implemented the BPE compression algorithm both in the standard way described in [20] and in the modified way stated in Section 2.2. The `Compress` program has an option to specify in bits the upper bound to the number of strings in a dictionary, and we used `Compress` with specification of 12 bits and 16 bits. Thus we tested five compression programs.

We estimated the compression ratio and the compression/decompression time of the five compression programs on SPARCstation 20 for the following texts.

Brown corpus: A well-known collection of English sentences, which was compiled in the early 1960s at Brown University, USA. The file size is about 6.8 Mbyte.

Medline: A clinically-oriented subset of Medline, consisting of 348,566 references. The file size is about 60.3 Mbyte.

Genbank1: A subset of the GenBank database, an annotated collection of all publicly available DNA sequences. The file size is about 43.3 Mbyte.

Genbank2: The file obtained by removing all fields other than accession number and nucleotide sequence from the above one. The file size is about 17.1 Mbyte.

The results are shown in Table 2. We can see from the table that:

- Compression ratios of BPE are not better than those of `Compress` and `Gzip`. The standard implementation of BPE is competitive with `Compress` with 12bit-option from the viewpoint of the compression ratio.
- Standard implementation of BPE is very slow.
- Modification of BPE stated in Section 2.2 drastically reduces the compression time. In case of English texts, its compression time is better than those of `Compress` and `Gzip`, and its compression ratio is not so worse than that of the standard BPE.
- Decompression is fast. It defeats `Compress` and is competitive with `Gzip`.

The table indicates that BPE is not so good from the traditional criteria. This is the reason why it has received little attentions until now. However, it has the following properties which are quite attractive from the practical view point of compressed pattern matching.

- No bit-wise operations are required since all the codes are of 8 bits.
- Decompression requires very small amount of memory.
- Partial decompression is possible. Namely, we can decompress any portion of compressed text.

In the next section, we will show how we can perform compressed pattern matching efficiently in the case of BPE compression.

3 Pattern matching in BPE compressed texts

This section deals with searching in BPE encoded files. For an uncompressed (raw) text file, many pattern matching algorithms are proposed, such as Knuth-Morris-Pratt (KMP) algorithm [27] and Boyer-Moore [12] for a single pattern, Aho-Corasick (AC) [2] for multiple patterns. The Shift-Or algorithm exploiting *bit parallelism* is

Table 2: Performance of BPE compared with `Compress` and `Gzip`.

(a) Compression ratio (%).

		BPE		Compress		Gzip
		standard	modified	12bit	16bit	
Brown corpus	(6.8Mb)	51.08	59.02	51.67	43.75	39.04
Medline	(60.3Mb)	56.20	59.07	54.32	42.34	33.35
Genbank1	(43.3Mb)	49.36	59.48	43.73	32.55	24.84
Genbank2	(17.1Mb)	36.19	57.20	29.63	26.80	23.15

(b) CPU time of compression/decompression (sec).

		BPE		Compress		Gzip
		standard	modified	12bit	16bit	
Brown corpus	comp.	196.97	8.05	5.63	12.7	37.72
	decomp.	2.15	2.44	4.12	4.02	2.80
Medline	comp.	1699.90	60.71	62.83	73.37	242.25
	decomp.	25.10	21.88	39.17	36.79	22.18
Genbank1	comp.	1119.92	43.73	36.63	41.34	203.96
	decomp.	17.10	16.29	24.53	23.4	14.45
Genbank2	comp.	440.64	16.50	11.17	19.29	100.92
	decomp.	6.15	5.93	7.93	7.93	5.89

practically very effective for a short pattern, and it can find multiple patterns as well as generalized pattern treating *don't cares* and set of symbols [1, 11, 41].

For a compressed text, the most naive approach would be the one which applies any of the above search routines with expanding the original text on the fly. This is quite simple but still useful when the file I/O time is critical. By modifying a given pattern, we can also apply the search routines in order to find the pattern directly without expanding the original text. The problem here is that the encoded pattern is not unique. A solution due to Manber [28] was to devise a way to restrict the number of possible encodings for any string.

We take another two approaches. The first approach is rather straight and brute-force: we expand all corresponding pattern encodings [35]. The number of encodings is $O(\ell^2 m^2)$, where ℓ is the number of pair codes and m is the length of the pattern. We then apply both the AC and the Shift-Or algorithms. The practical behaviors of these approaches are not so bad, as shown in Section 4. Especially for Shift-Or, the number of encodings in terms of the generalized pattern is only $O(m^2)$. Table 3 shows the average and the worst case number of pattern encodings for English words in Unix dictionary of spell checker. Compared with the theoretical estimate above, the number of pattern encodings is quite small. In particular for Shift-Or, only five encodings arise at the worst case.

The second approach is also simple but powerful enough. Suppose we have the KMP automaton for a given pattern. Note that one character code of the BPE compressed text may represent a string of length more than one, which causes a

Table 3: Number of pattern encodings.

	AC		Shift-Or	
	average	max.	average	max.
Brown Corpus	37.11	130	2.73	5
Medline	57.19	156	2.87	5

series of state transitions. Our idea is to substitute just one state transition for each such consecutive state transitions. The transition table that realizes this idea can be implemented as a two dimensional array since the number of different codes is at most 256 in BPE. This is not the case with LZW, in which the number of codes can be the compressed text size. Alternative implementation is the one utilizing the bit parallel paradigm in a similar way that we did for LZW compression [25] (technical details are omitted).

4 Experimental results

We estimated the running time of the algorithms presented in the previous section in searching BPE compressed text, in comparison with those of the algorithm for searching LZW compressed text [25] and ordinary algorithms searching uncompressed text. We used the four text files mentioned in Section 2.

In general, the running time is the sum of the file I/O time and the CPU time for compressed pattern matching. Since the file I/O time is given by

$$\text{file I/O time} = \text{compressed text size/data transfer ratio},$$

the running ratio depends strongly on the data transfer ratio as well as the compression ratio. In case of slow data transfer, the running time is almost dominated by the file I/O time. On the other hand, when the data transfer is relatively fast, the CPU time is an important factor. Thus we performed our experiments in the following different situations.

1. Notebook personal computer (Gateway2000 Solo9100, PentiumII 300MHz). The file transfer ratio is 6.04 Mbyte/sec.

2. Workstation (SPARCstation 20) with local disk storage. The file transfer ratio is 3.27 Mbyte/sec.

3. Workstation (SPARCstation 20) with remote disk storage. The file transfer ratio is 0.96 Mbyte/sec.

The experimental results are shown in Tables 4 and 5. In these tables, (a) and (b) stand for the automata and the bit-parallel implementations stated in the previous section, respectively. In the case of searching in uncompressed text, (a) and (b) mean the KMP and the Shift-Or algorithms, respectively.

Table 4: Elapsed time (sec).

		standard BPE		modified BPE				LZW	uncompressed	
		approach 2		approach 1		approach 2		[25]	(a)	(b)
		(a)	(b)	(a)	(b)	(a)	(b)			
1	Brown Corpus	0.82	0.71	0.76	0.82	0.76	0.66	2.42	1.27	1.43
	Medline	16.26	8.46	5.98	6.27	5.65	6.59	17.47	9.83	10.71
	Genbank1	10.93	5.55	4.73	4.23	4.23	4.23	9.81	6.98	7.86
	Genbank2	5.00	1.59	2.47	2.73	1.70	1.65	3.51	2.80	3.07
2	Brown Corpus	1.45	1.40	1.29	1.32	1.26	1.25	5.16	2.10	2.08
	Medline	40.67	16.06	12.76	11.48	10.63	11.07	42.86	16.81	16.97
	Genbank1	34.27	11.05	8.18	8.32	8.09	8.12	23.55	14.23	13.59
	Genbank2	15.73	3.49	4.07	4.21	3.31	3.13	7.81	5.41	5.49
3	Brown Corpus	3.56	3.48	4.32	4.73	3.96	4.13	5.92	6.84	6.74
	Medline	49.70	34.86	38.07	40.19	34.71	35.91	49.23	60.85	63.96
	Genbank1	38.45	22.07	28.31	27.55	26.29	27.32	26.85	46.21	48.97
	Genbank2	16.13	6.62	12.75	12.81	9.86	9.91	8.93	16.93	16.78

First of all, it can be observed that the algorithms searching in modified BPE compressed files are faster than ordinary search in uncompressed files. In fact the running time is reduced to approximately $54 \sim 63\%$ of uncompressed case. It can also be observed that their performances are better than the algorithm for LZW compressed files in both the situations of notebook and workstation with local disk. Even in the situation of remote disk, the algorithm for LZW compressed files is not better than them except for Genbank2, which almost consists of nucleotide sequences.

In the case of the standard BPE compressed text, since the substitution table varies depending on the block, we need reconstruction of the automaton or the mask bit vector which consumes a considerable amount of CPU time, and therefore the running speed may slow down when the data transfer is relatively fast. Since the reconstruction of the automaton is more expensive than that of the mask bit vector, the running time of (b) is better than (a).

On the other hand, for modified BPE compressed text, the construction of the automaton or the mask bit vector is executed only once. Tables 4 and 5 imply that the automata and the bit-parallel implementations are almost competitive in both the approaches 1 and 2, over the three situations.

5 Conclusion

We have shown potential advantages of BPE compression from a view point of compressed pattern matching. Karp-Rabin algorithm [23] using the fingerprint function is also applicable to BPE compressed method in a natural way. False matches can be detected easily by direct comparisons, because BPE compressed text can be ex-

Table 5: CPU time (sec).

		standard BPE		modified BPE				LZW	uncompressed	
		approach 2		approach 1		approach 2		[25]	(a)	(b)
		(a)	(b)	(a)	(b)	(a)	(b)			
1	Brown Corpus	0.57	0.50	0.41	0.44	0.38	0.46	2.34	0.99	1.05
	Medline	12.85	5.18	2.80	2.99	2.94	3.62	16.18	7.39	7.82
	Genbank1	9.76	3.53	2.16	2.26	2.22	2.69	9.06	4.99	5.32
	Genbank2	4.33	1.36	1.02	1.05	0.97	1.19	3.34	2.18	2.32
2 & 3	Brown Corpus	1.18	1.24	0.65	0.77	0.71	0.93	4.89	1.00	1.25
	Medline	40.28	15.73	6.59	7.76	7.19	9.40	41.32	10.45	12.39
	Genbank1	33.82	10.46	4.57	5.40	5.15	6.56	22.97	7.44	8.80
	Genbank2	15.28	3.28	1.70	2.10	1.79	2.27	7.62	2.87	3.41

tracted partially. This algorithm is also attractive in practice for long pattern. We are preparing the experiments currently.

References

[1] K. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987.

[2] A. V. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18(6):333–340, 1975.

[3] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *Proc. Data Compression Conference*, page 279, 1992.

[4] A. Amir and G. Benson. Two-dimensional periodicity and its application. In *Proc. of the 3rd Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 440–452, 1992.

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

[6] A. Amir, G. Benson, and M. Farach. Optimal two-dimensional compressed matching. *Journal of Algorithms*, 24(2):354–379, 1997.

[7] A. Amir, G. M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.

[8] A. Apostolico and Z. Galil. *Pattern Matching Algorithm.* Oxford University Press, New York, 1997.

[9] S. Arikawa and S. Shiraishi. Pattern matching machines for replacing several character strings. *Bulletin of Informatics and Cybernetics*, 21(1–2):101–111, 1984.

[10] S. Arikawa et al. The text database management syste SIGMA: An improvement of the main engine. In *Proc. of Berliner Informatik-Tage*, pages 72–81, 1989.

[11] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Comm. ACM*, 35(10):74–82, 1992.

[12] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. ACM*, 20(10):62–72, 1977.

[13] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.

[14] E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Direct pattern matching on compressed text. In *Proc. of the 5th International Symp. on String Processing and Information Retrieval*, pages 90–95. IEEE Computer Society, 1998.

[15] E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast sequencial searching on compressed texts allowing errors. In *Proc. of the 21st Ann. International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 298–306. York Press, 1998.

[16] T. Eilam-Tzoreff and U. Vishkin. Matching patterns in strings subject to multilinear transformations. *Theoretical Computer Science*, 60(3):231–254, 1988.

[17] M. Farach and M. Thorup. String-matching in Lempel-Ziv compressed strings. In *27th ACM STOC*, pages 703–713, 1995.

[18] W. B. Frakes and R. Baeza-Yates. *Information Retrieval Data Structures & Algorithms*. Prentice Hall PTR, 1992.

[19] S. Fukamachi, T. Shinohara, and M. Takeda. String pattern matching for compressed data using variable length codes. Submitted, 1998.

[20] P. Gage. A new algorithm for data compression. *The C Users Journal*, 12(2), 1994.

[21] L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, volume 1097 of *Lecture Notes in Computer Science*, pages 392–403. Springer-Verlag, 1996.

[22] G. C. Jewell. Text compaction for information retrieval. *IEEE SMC Newsletter*, 5, 1976.

[23] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Develop*, 31(2):249–260, 1987.

[24] M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic Journal of Computing*, 4:172–186, 1997.

[25] T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Shift-And approach to pattern matching in LZW compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in Computer Science. Springer-Verlag, 1999. to appear.

[26] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa. Multiple pattern matching in LZW compressed text. In J. A. Atorer and M. Cohn, editors, *Proc. of Data Compression Conference '98*, pages 103–112. IEEE Computer Society, 1998.

[27] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput*, 6(2):323–350, 1977.

[28] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. In *Proc. Combinatorial Pattern Matching*, volume 807 of *Lecture Notes in Computer Science*, pages 113–124. Springer-Verlag, 1994.

[29] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Computing*, 22(5):935–948, 1993.

[30] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of ACM*, 23(2):262–272, 1976.

[31] M. Miyazaki, S. Fukamachi, M. Takeda, and T. Shinohara. Speeding up the pattern matching machine for compressed texts. *Transactions of Information Processing Society of Japan*, 39(9):2638–2648, 1998. (in Japanese).

[32] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. 8th Ann. Symp. on Combinatorial Pattern Matching*, volume 1264 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1997.

[33] G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in Computer Science. Springer-Verlag, 1999. to appear.

[34] M. Nelson. *The data compression book*. M&T Publishing, Inc., Redwood City, Calif., 1992.

[35] Y. Shibata. Speeding-up string pattern matching by text compression using byte pair encoding. Diploma thesis (in Japanese), Kyushu Institute of Technology, 1998.

[36] Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. Pattern matching in text compressed by using antidictionaries. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, Lecture Notes in Computer Science. Springer-Verlag, 1999. to appear.

[37] M. Takeda. An efficient multiple string replacing algorithm using patterns with pictures. *Advances in Software Science and Technology*, 2:131–151, 1990.

[38] M. Takeda. Pattern matching machine for text compressed using finite state model. Technical Report DOI-TR-CS-142, Department of Informatics, Kyushu University, October 1997.

[39] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[40] P. Weiner. Linear pattern-matching algorithms. In *Proc. of 14th IEEE Ann. Symp. on Switching and Automata Theory*, pages 1–11. Institute of Electrical Electronics Engineers, New York, 1973.

[41] S. Wu and U. Manber. Fast text searching allowing errors. *Comm. ACM*, 35(10):83–91, October 1992.