数理解析研究所講究録
第586巻 1986年 18-34

COMPARISON AND IMPROVEMENT OF STRING MATCHING ALGORITHMS FOR JAPANESE TEXTS

Jeehee YOON, Toshihisa TAKAGI, and Kazuo USHIJIMA

尹志熙　　高木利久　　牛島和夫

Department of Computer Science and Communication Engineering
Kyushu University, Fukuoka, 812, Japan

九州大学 工学部

This paper considers the problem of string matching for Japanese texts. We apply three string matching algorithms which are well known in the Roman alphabetic world to the Japanese texts and compare their performances experimentally. Because of a large character set, Japanese characters are coded into two bytes, and this feature has a great influence on the performance of the algorithms. The Boyer-Moore algorithm which is extremely efficient on Roman alphabetic texts shows the worst performance on the Japanese texts. This is because the preprocessing time of the algorithm is proportional to the size of the character set. We present an efficient method to improve the preprocessing time. We also consider an implementation method of the algorithms on the normalized Japanese texts. By considering the Japanese texts as a series of one byte codes, we scan the texts in one byte units as is done in the Roman alphabetic texts. The performances of the algorithms which use this method are evaluated and compared with the other ones.

1. Introduction

Unlike English and other European languages, Japanese has a number of features[10] that give rise to problems concerning text

processing. The most significant feature lies in the number of characters.

In this paper we study the problem of string matching for Japanese text with a large character set. String matching is one of the basic operations of text processing and several techniques have been proposed. The most important techniques are the Knuth-Morris-Pratt algorithm[1], the Boyer-Moore algorithm[2] and the Quadratic algorithm[8]. We applied these three algorithms to Japanese texts and compared their performances experimentally.

For the large number Japanese characters, we need at least two bytes to encode them. The Japanese text is composed of a free mixture of passages in two kinds of character sets : traditional alphabetic set coded into one byte units and Japanese character set coded into two byte units. In our case, shift codes are used to distinguish these two character sets. On the Japanese texts that have the above features, it is not so easy to implement the algorithms which are generally developed for Roman alphabetic texts. In this experiment, for the simple implementation of the algorithms, we adopted the normalization method[9] which allows us to treat both character sets uniformly without using shift codes. Only by scanning the texts in Japanese character units (two byte units and not one byte units), we can apply the algorithms to the normalized Japanese texts without any modifications to the algorithms.

The performances of the algorithms implemented by this method are evaluated and compared experimentally. This empirical study shows several interesting results which are different from those exhibited on the Roman alphabetic texts. For example, the Boyer-Moore algorithm, commonly known as the most efficient algorithm on Roman alphabetic texts, shows the worst performance on Japanese

texts. This is because the preprocessing time of the Boyer-Moore algorithm is proportional to the size of character set. We present an efficient method to improve the preprocessing time.

We also consider an implementation method on the normalized Japanese texts. The method is : by considering the Japanese texts as only a series of one byte codes, scanning is done in one byte units as is done in the Roman alphabetic texts. We call this method one byte unit scanning. In this case, the problem of mis-detection that occurs from one byte unit scanning is considered and its setting methods are also given. The performances of algorithms which use the one byte unit scanning are also evaluated and compared. In comparison with the others, the Boyer-Moore algorithm adopting the one byte unit scanning method shows the best performance.

2. Algorithms

We start with a brief review of the three string matching algorithms. We define the cost of the algorithm as the number of references made to the text string [2] and represent the pattern and text length by m and n.

the Quadratic Algorithm (QUADRATIC)

This algorithm[8] is a simple one and is given as follows : the pattern is placed on the top of the left-hand end of the text string so that the first character of the pattern and that of the text string are aligned. And then the pattern is scanned through the text string. If a mismatch occurs, the pattern is shifted one position to the right and the scan is restarted from the new position of the pattern's first character. The cost of this algorithm ranges from a minimum of n to a maximum of $m*(n-m+1)$[8].

the Knuth-Morris-Pratt Algorithm (KMP)

In the worst case, the cost of the Quadratic algorithm is of order of m*n[8]. Knuth, Morris and Pratt present an algorithm[1] proportional to m+n. This algorithm can be described as follows : like Quadratic, the pattern and the text string are aligned and the scanning is done to the right. However, when a mismatch occurs, the pattern is shifted to the right such that the scanning can be started at the point where mismatch occurred in the text string.When using this algorithm, we must preprocess the pattern to prepare a table that is called "next table". This tells us how far to slide the pattern when we detect a mismatch. This algorithm has a cost ranging from n to 2n-m and it additionally needs O(m) comparisons and O(m) memory to implement the next table.

The Boyer-Moore Algorithm (BM)

In every case, Quadratic and KMP inspect each character of the text string at least once. Boyer and Moore, on the other hand , present a new string matching algorithm[2] where the number of the inspected characters is approximately c*(n+m), where c<1.

Unlike Quadratic and KMP, BM compares the pattern with the text from the right end of the pattern. If a mismatch should occur, the pattern is shifted according to the value of the precomputed tables (delta1 and delta2). For example, if we find that the text character positioned against the last character in the pattern does not appear in the pattern at all, we immediately shift the pattern m places. Thus, when the alphabet size is large, we need to inspect only about n/m characters of the text, on average. Even in the worst case[3,4,5,6], the cost is proportional to 4n-m comparisons. However, to prepare two

tables (delta1 and delta2), the algorithm requires O(m + the size of the alphabet) comparisons and O(m + the size of the alphabet) memory in addition.

Smit [8] reports on the average performances of the three algorithms on Roman alphabetic texts. The conclusions are :

- BM is extremely efficient in most cases.
- Contrary to the impression one might get from the analytical results, KMP is not significantly better on average than Quadratic.

3. Environment for Japanese Text Processing

As mentioned in Section 1, the large number Japanese characters cannot be represented in one byte, and at least two bytes are needed to encode them. For a Japanese text file, coding methods and file formats differ from one computing system to another. Our experiment was done on a FACOM M382 system. The operating system is FACOM OS IV/F4 with JEF (Japanese Extended Features). In JEF environment, we have to use two character sets. One is the set of traditional alphabetic sets which are coded into one byte, which we call 1-byte character. The other one consists of Kanji, Hiragana, Katakana, and other Japanese characters. These are coded into two bytes, which we call 2-byte characters.

As a Japanese text is composed of a free mixture of passages in these two character sets, we may need shift codes to distinguish them. However these shift codes make it difficult to process Japanese texts[9]. For the simple implementation of algorithms, we adopted the normalization method[9]. This involves coding even the 1-byte characters into two bytes. That is, we normalized each 1-byte character into two bytes by appending a leading byte whose value is 0.

This method allows us to treat both character sets uniformly. In Figure 1 we show the representation of character codes in both the JEF and the normalized texts.

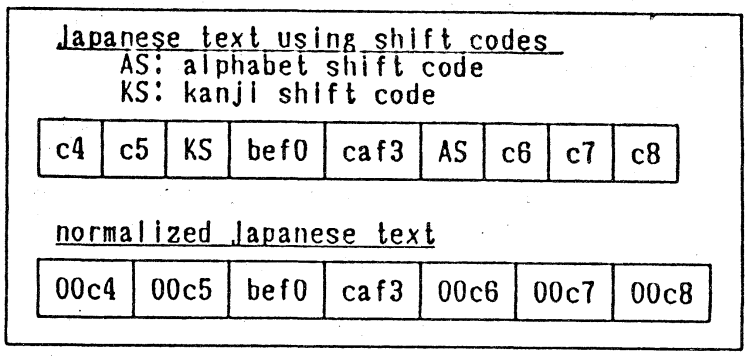

Figure 1. Internal representation of Japanese texts

4. Experimental comparison of the three algorithms

For the simple implementation of algorithms, we used the Japanese texts normalized by the method given in Section 3. That is, only by scanning the texts in two byte units (Japanese character units), not in one byte units, we could apply the algorithms to the Japanese texts very simply. From the normalized Japanese text of length 5000 2-byte characters, we randomly selected 20 patterns of length m, from 1 to 14. For each pattern length, one pattern not occurring in the text string was generated.

To determine the cost of each algorithm, we first count the number of references made to text string to locate every occurrence of a pattern. We will call this the scanning cost. Additionally, for the algorithm which require preprocessing, we count the number of comparisons between pattern characters (for KMP) and the number of assignment and test times (for BM). By adding these preprocessing costs to the scanning costs, we obtained the total cost that represent

the performances of the algorithms. This cost is divided by the number of text length and a cost factor is obtained. We then average these cost factors over the 20 patterns of each pattern length.

In Figure 2 the cost of the algorithm is plotted against pattern length for each of the three algorithms, Quadratic, KMP and BM.The costs of the Quadratic and KMP are each respectively, slightly more than one, and almost independent of the pattern length. This reflects the fact that Quadratic and KMP reference the text string almost once per character. The most obvious feature to be noted in the figure is that the cost of BM is much higher than those of Quadratic and KMP.

In Figure 3, the costs of the algorithms without consideration for preprocessing are plotted against the pattern length for KMP and BM. Note that for the BM the number of references to text string per character is less than 1. That is, the majority of the cost of the BM is in the preprocessing , not in the scanning. However from Figures 2 and 3 we can say that the preprocessing cost of KMP is negligible in comparison with the KMP scanning cost.

In Figure 4 the cost of the BM algorithm is plotted against the pattern length for four different lengths of text strings : A, B, C, D, constituted of 5000, 20000, 40000, 80000 2-byte characters respectively. This figure exhibits the phenomenon that the cost of BM decreases as the text length gets longer. For example, the cost is lower than 1.0 for a pattern length 2 or more in a text of length C.

5. Improving the preprocessing time of BM

In BM, it is necessary to construct two tables, delta1 and delta2, during the preprocessing. However, the purpose of the delta2 table is only to optimize the handling of repetitive patterns. As the repetitive patterns seldom appear in English text, Horspool deletes delta2 table

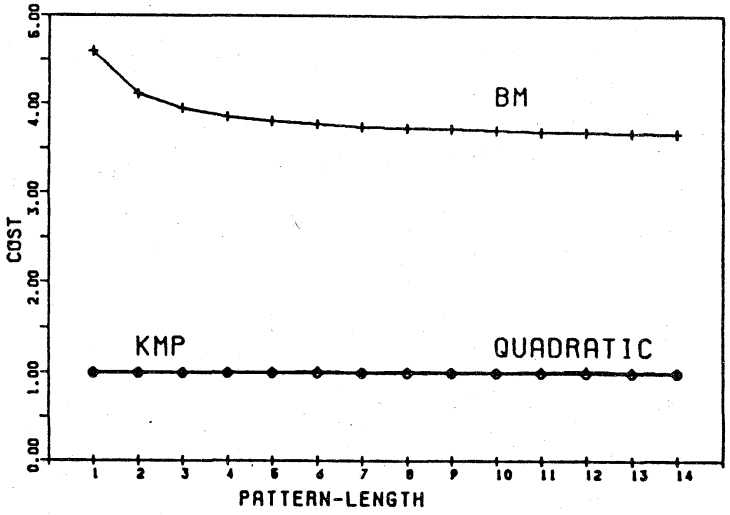

Figure 2. Comparison of costs vs. the pattern length

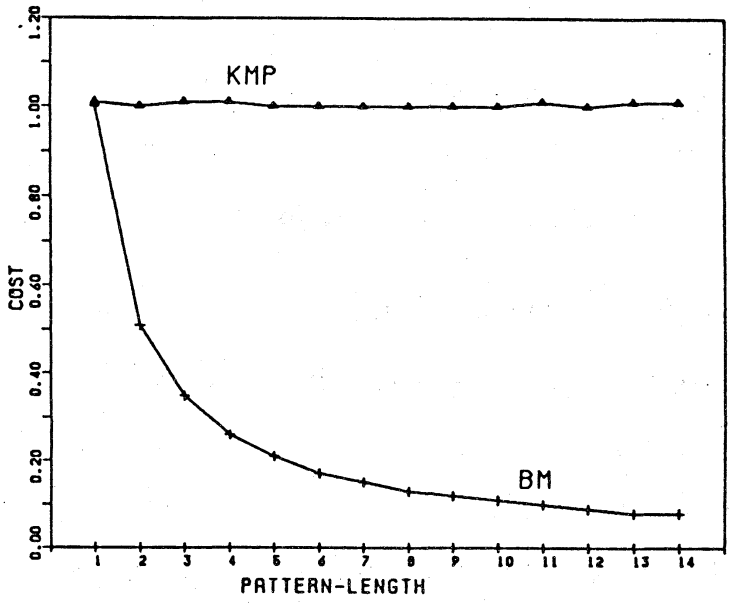

Figure 3. Comparison of costs vs. the pattern length without considering the preprocessing

in the BM algorithm[7]. The same thing can be said for Japanese texts and we can also delete it. In the preprocessing we construct the delta1 table in the following form :

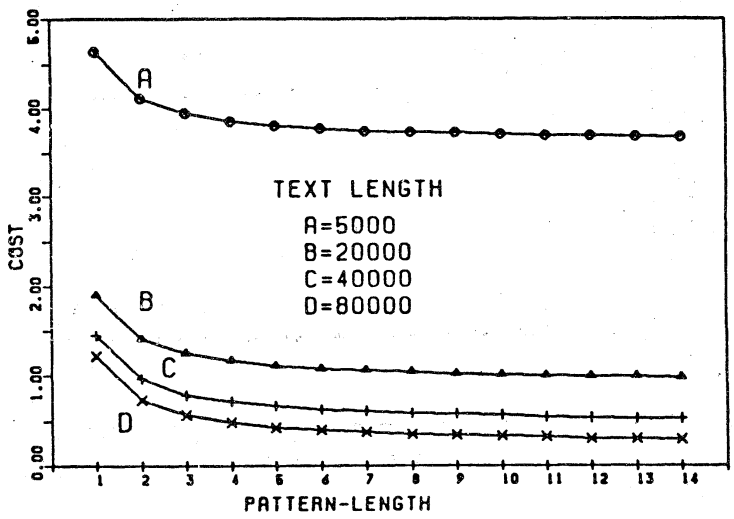

Figure 4. Costs of BM algorithm for different lengths of texts

Note : here the notation pat(j) refers to the j-th character in the pattern (counting from 1 on the left). The entry in the delta1 table for some character **ch** is denoted by delta1(**ch**).

Algorithm BM { computing delta1}

for every character **ch** in the input alphabet do

delta1(**ch**) := m ;

for j := m downto 1 do

if delta1(pat(j)) = m then delta1(pat(j)) := m - j ;

As given above, we must first initialize all the entries of this table and set up the values in a linear scan through the pattern. Thus this preprocessing time for the delta1 table is proportional to (the pattern length + the size of the character set). Here, for Japanese text processing, we must execute a large number of assignment statements to initialize all entries and this make the performance of the BM so bad as that given in Section 4.

To improve the preprocessing time, we have reconstructed the structure of the delta1 table. We call the improved BM algorithm using our delta1 table the NEW-BM algorithm. In Figure 5-(A) we show the configuration of the delta1 table implemented by the conventional method. This table has 65,536($=2^{16}$) entries and among them at most m entries have the value different from m for the pattern length m. In the NEW-BM, we implement the delta1 table in a double-layered manner as sketched in Figure 5 -(B). This table is composed of the main table with 256 entries and several subtables with each 256 entries also. The main table corresponds to the upper half of 16-bit characters, and the subtables corresponds to the lower half.

In table(B), the entry value of table(A) which is different from m is represented as follows : we insert a pointer to a subtable in the entry of the main table which corresponds to the upper half of that entry character, and the value is inserted in the entry of the subtable which corresponds to the lower half of that entry character. For example, the value 1 of the character entry of "ベ" (coded in "A4D9") in the table(A) corresponds to the value of the subtable's entry of "D9" pointed to by the main table's entry of "A4".

We present the preprocessing of NEW-BM using our compressed delta1 table in the following way :

Note : here the notation pat(j_1) refers to the upper half of the j-th character in the pattern and pat(j_2) refers to the lower half. The i-th entry of main table of delta1 is denoted by delta1(main,i) and that of j-th subtable is denoted by delta1(main+j, i).

Algorithm NEW-BM { computing delta1}

for i:=0 to 255 do

```
    delta1(main,i) := m ;
for j := m downto 1 do
    if delta1(main , pat(j1)) = m then
        begin
            delta1(main , pat(j1)) := main + j ;
            for i := 0 to 255 do
                begin
                    delta1(main+j , i) := m ;
                    delta1(main+j , pat(j2)) := m - j
                end
        end
    else
        if delta1(delta1(main,pat(j1)),pat(j2)) = m then
            delta1(delta1(main , pat(j1)),pat(j2)) := m - j ;
```

Now we show the performance of our NEW-BM algorithm from the view-point of space and time. If a table entry is represented by 4 byte, the size of table(A) is 256Kbytes and that of table(B) is at most (m+1) Kbytes. Thus, the space to represent the delta1 table is reduced by (m+1)/256. In this case m is always much less than 256.

By representing the delta1 table as above, the preprocessing cost needed to initialize the table is reduced. In Figure 6, the cost of NEW-BM is plotted against the pattern length and is compared with the other algorithms. It is clear that the performance of the algorithm is significantly improved in comparison with that of BM given in Figure 2.

However, by transforming the structure of the table, accessing the table during the search itself requires more instructions than before. Therefore, the speed of the search algorithm is impaired slightly. In

Table 1 the running times of the string matching algorithms are compared. NEW-BM is superior to the other algorithms, especially as the pattern length becomes longer.

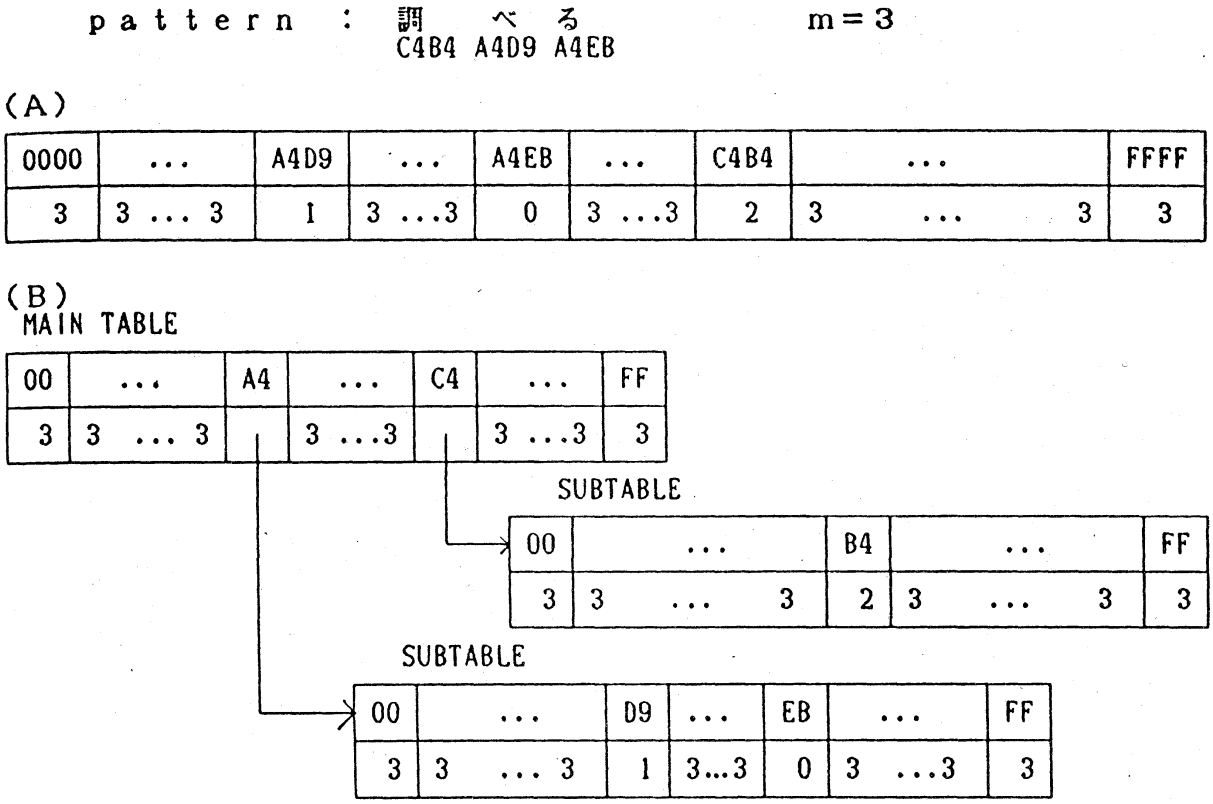

Figure 5. Structure of delta1 table

6. One byte unit scanning of the Japanese texts

Until now, the scanning has been in Japanese character units (two byte units). In this Section, we show some different implementation method of the algorithms.

By considering the normalized Japanese texts as a series of one byte codes, we can scan the texts in one byte units as is done in the Roman alphabetic texts. But if we directly apply this method to the

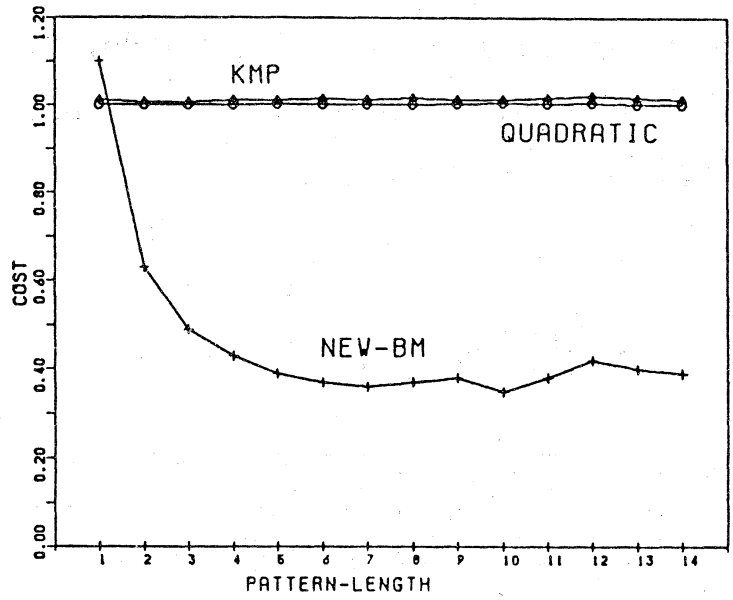

Figure 6. Comparison of costs vs. the pattern length

implementation of these algorithms, Quadratic, KMP, and BM, we will detect the patterns to which no original patterns corresponds. For example, if we let the Japanese characters A, B, C, D be represented as follows : A =aa, B=ab, C=ba, D=bb. Then the algorithms will detect a pattern abab=BB in a given text ACD=aababb. To avoid such a mis-detection, some modifications of the algorithms are required. But when the pattern scanning is done on the normalized Japanese texts, the modification is very simple. As the Japanese characters are represented by uniform two byte codes, the point where the pattern is detected must be at the odd-numbered position only. By checking whether the point is odd-numbered or not, we can select the real occurrences of a pattern which corresponds to the original one.

We applied this methods to the implementation of the three string matching algorithms and compared their performances. In Table 1, the running times of the algorithms are given. Here QUADRATIC*, KMP* and BM* respectively represents the modified QUADRATIC, KMP and BM in which the scanning is done in one byte units. The

most obvious feature in this table is that BM* is extremely efficient in most cases. The BM* is more efficient than the NEW-BM algorithm. This is because in the BM* algorithm the preprocessing time to construct the delta1 table and access time of the delta1 table are much less in comparison with the NEW-BM algorithm, even though the number of references made to the texts in BM* is a little more than that of the NEW-BM.

The number of times of mis-detection is counted for each of the pattern length and the percents that it occupy in the total number of times of detection are given in Table 2. As the pattern length becomes longer, the possibility of making the mis-detection is almost 0.

7. Conclusions

From the results of the comparisons of the string matching algorithms, we can readily say that in processing Japanese texts, except when the pattern length is 1, the best solution for a string matching program is to use the BM* algorithm which scans the text in one byte units. For the pattern length 1, the QUADRATIC is more efficient than the other algorithms.

We have observed the performance and improvement of string matching algorithms on normalized Japanese texts[9]. As both of the 1-byte and 2-byte characters are coded into two bytes uniformly, we can implement the algorithms for Japanese texts with a few modifications. However, most of the Japanese computer manufacturers use their own character sets and shift codes for their Japanese text files. And there is no compatibility[9] among them. In processing these text files which are not normalized, considerable modifications of string matching algorithms will be needed.

Table 1. Running times of the algorithms (CPU time : msec)

m	QUADRATIC	KMP	BM	NEW-BM	QUADRATIC*	KMP*	BM*
1	1.90	2.00	7.10	5.25	4.94	6.25	3.00
2	1.85	1.95	4.85	2.90	4.69	5.75	1.54
3	1.85	1.90	4.25	2.10	4.69	5.59	1.04
4	1.95	1.95	3.70	1.65	5.00	6.29	0.79
5	1.95	1.95	3.50	1.35	4.84	5.94	0.64
6	1.90	1.95	3.35	1.15	4.69	5.54	0.59
7	1.85	1.90	3.25	1.10	4.44	5.00	0.50
8	1.85	1.95	3.35	1.05	5.00	6.39	0.44
9	1.85	1.95	3.10	1.05	4.64	5.50	0.39
10	1.85	1.95	3.10	0.95	4.84	5.84	0.39
11	1.95	2.00	3.10	0.90	4.94	6.09	0.39
12	1.95	2.00	3.10	0.90	5.39	7.19	0.34
13	1.90	1.95	3.00	0.80	4.89	6.14	0.39
14	1.90	2.00	2.95	0.65	5.00	6.44	0.34

Most of the problems that occur in processing Japanese texts are common to processing other languages that have large character sets such as Chinese and Korean[10].

Acknowledgement

This work was partly supported by Grant-in-Aid for Scientific Research, Ministry of Education, Science and Culture, No. 60460228.

We are very grateful to Mr. T. Shinohara of Kyushu University Computer Center for his advice on this research.

Table 2. The percents of mis-detection in one byte unit scanning

m	MIS-DETECTION (%)	m	MIS-DETECTION (%)
1	11.30	8	0.00
2	0.00	9	0.00
3	18.30	10	0.00
4	21.99	11	0.00
5	0.00	11	0.00
6	0.00	13	0.00
7	0.00	14	0.00

References

[1]Knuth, D. E., Morris, J. H. Jr., and Pratt, V. R., "Fast pattern matching in strings," *SIAM J. Comput.*, Vol. 6, No. 2, pp. 323-350, 1977.

[2] Boyer, R. S., and Moore, J. S., "A fast string searching algorithm," *Comm. ACM*, Vol. 20, No. 10, pp. 762-772, 1977.

[3] Rivest, R. L., "On the worst case behavior of string-searching algorithms," *SIAM J. Comput.*, Vol. 6, No. 4, pp. 669-674, 1977.

[4] Galil, Z., "On improving the worst-case running time of the Boyer-Moore string matching algorithms," *Comm.ACM*, Vol. 22, No. 9, pp. 505-508, 1979.

[5] Rytter, W., "A correct preprocessing algorithm for Boyer-Moore string-searching," *SIAM J. Comput.*, Vol. 9, No. 3, pp. 509-512, 1980.

[6] Guibas, L. J. and Odlyzko, A. R., "A new proof of the linearity of the Boyer-Moore string searching algorithm," *SIAM, J. Comput.*, Vol. 9, No. 4, pp. 672-682, 1980.

[7] Horspool, R. N., "Practical fast searching in strings ," *Softw. Pract. Expr.*, Vol. 10, pp. 501-506, 1980.

[8] Smit, G. "A comparison of three string matching algorithms," *Softw. Pract. Expr.*, Vol. 12, pp. 57-66, 1982.

[9] Ushijima, K., Kurosaka, T., and Yoshida, K., "SNOBOL4 with Japanese text processing facility," *Proc. ICTP'83*, pp. 235-240, 1983.

[10] Becker, J. D., "Typing Chinese, Japanese, and Korean," *Computer*, pp. 27-34, January, 1985.