Journal of Computer Science and Cybernetics, V.35, N.3 (2019), 197–216
DOI 10.15625/1813-9663/35/3/13620

A NEW APPROACH TO EXACT PATTERN MATCHING

NGUYEN HUY TRUONG

School of Applied Mathematics and Informatics, Hanoi University of Science and Technology, Vietnam; truong.nguyenhuy@hust.edu.vn

Abstract. In this paper we introduce a flexible approach to design an effective algorithm for exact pattern matching, and compare it with some of the most efficient algorithms, such as AOSO, EBOM, FJS, FSBNDM, HASHq, LBNDM, SA, BMH-SBNDM, SBNDMq, TVSBS. These results are based on the concept of the degree of fuzziness (appearance) presented by P. T. Huy et al. in 2002. Theoretical analyses and experimental results indicate that in practice our proposed algorithm is faster than the above mentioned algorithms in most of the cases of patterns and alphabets.
Keywords. Exact Pattern Matching; String Matching; Automata Approach.

1. INTRODUCTION

Pattern matching is a classic problem in computer science and one of the most cited problems in word processing algorithms. The applications of pattern matching algorithms are used daily to access information, such as in the search engine Google, database queries, search and replace in text editing systems, etc. At present, there are two research directions for the problem that are exact and approximate pattern matching. Moreover, based on the number of patterns found by the algorithm, the pattern matching problem can be divided into single and multiple pattern matching.

Our work addresses the exact single pattern matching problem (hereafter, commonly called the exact pattern matching problem).

Given a pattern string x of length m and a text y of length n on the same alphabet. The exact pattern matching problem is to search for occurrences of the pattern x in the text y.

Since 1977, with the two famous publications of the Boyer-Moore [6] and Knuth-Morris-Pratt [18] exact pattern matching algorithms, there have been about hundreds, if not thousands, of papers published that dealt with the exact pattern matching problem [13].

The worst case lower bound of the pattern matching problem is $O(n)$. The first algorithm to achieve the bound was proposed by Morris and Pratt in 1970, afterwards improved by Knuth et al. in 1977 [12, 18].

In 2013, S. Faro and T. Lecroq reviewed the 85 exact pattern matching algorithms published during the 2000 - 2010 period and presented experimental results to compare them. According to this evaluation, they listed 10 most efficient sequential algorithms in practice, as well as their best results. They were AOSO, EBOM, FJS, FSBNDM, HASHq, LBNDM, SA, BMH-SBNDM, SBNDMq, TVSBS (for short, called 10 algorithms) [12].

Nearly all of these algorithms are concerned with how to slide the window. They scan the text with the help of a window, actually the window is a substring of the text whose size

equals the length of the pattern. For each window of y, they try to find an occurrence of x in y by comparing the letters of the window with the letters of x (FJS [6, 13, 18, 24], HASHq for $q = 3; 5; 8$ [12, 19, 26], TVSBS [3, 23, 25]) or moving the state of an automaton when one letter of window is scanned at a time (AOSO [14], EBOM [1, 7, 11], FSBNDM [1, 7, 11, 12], LBNDM [5, 9, 10, 15, 16, 20, 21], SA [2], BMH-SBNDM [5, 9, 12, 15, 16, 20, 21], SBNDMq for $q = 2; 4; 6; 8$ [10, 12]) in a certain way. After each occurrence of x, or a mismatch occurs at any position being scanned in window, they shift the window to the right by a certain number of positions depending on the approach used by the algorithms. This mechanism is called the sliding window mechanism [12] and repeated until the right end of the window does not belong to y. At the beginning of the matching, the left ends of the window and the text are aligned. The sliding window mechanism is shown in the following figure.

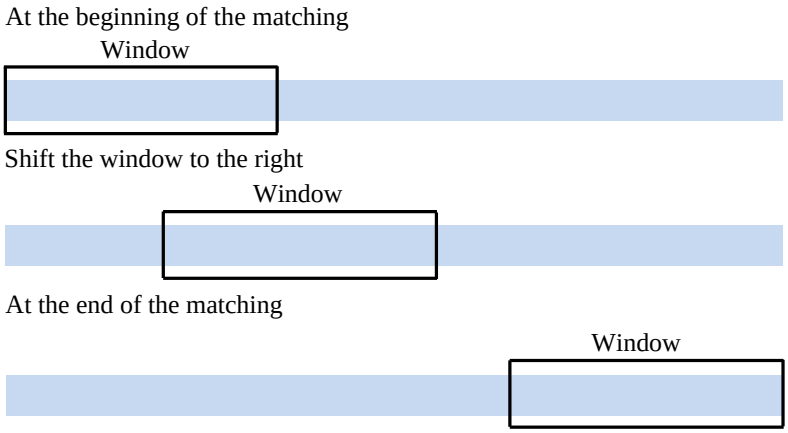

Figure 1. Sliding Window mechanism

According to our knowledge, the 10 algorithms have some common features as follows. They first do not take advantage of the relationship between the size of the pattern and the alphabet, then they are not flexible to shift the window regularly to the right by the most possible maximum number of positions with high probability (except for HASHq, SBNDMq). Secondly, the appearance of a part of the pattern is not immediately reflected or updated at a position being scanned in the text, hence when the windows overlap, the letters of the text could still be scanned more than once (except for SA, but this algorithm does not use the sliding window mechanism).

Our main goal will be to attempt to handle the two above situations. All of the best sequential algorithms in [12] have an $O(n)$ time complexity in the worst case and perform well in practice.

As we know that theoretical analyses in the worst case are not sufficient to predict actual running time accurately for exact pattern matching algorithms [3, 25]. For example, two of the best algorithms listed in [12], EBOM [11] and TVSBS [25], have $O(mn)$ time complexity in the worst case, where m and n are lengths of the pattern and the text. So, for the exact pattern matching problem, authors mainly focus on the efficiency of algorithms in practice.

Theoretically, it is not possible to make the time complexity less than the linear level, but in fact, the running time might decrease by reducing the number of letters of the text accessed. To solve this problem, we need to avoid scanning an arbitrary letter of the text

repeatedly and shift the window regularly to the right by a possible maximum number of letters. Based on the concept of the degree of fuzziness [17], we propose a flexible approach to design an effective algorithm which deals with above analyses successfully in practice.

In our approach, we construct an automaton corresponding to a given pattern to accept the pattern, where a state of the automaton is a degree of fuzziness. This tool will reflect and update the appearance of the longest prefix of the pattern in the text at any position. If the longest prefix of the pattern is the pattern, then a occurrence of the pattern in the text is found. A new window is always scanned from left to right. However, the window scanning process is only started when the right end substring of length c (for short, called c_block) of the window belongs to the pattern, where c is a given positive integer, $1 \leq c \leq m$. Further, the window scanning process will be stopped immediately if the current state of the automaton holds a given condition. These techniques make the window be shifted more regularly to the right by the most possible maximum number of positions with high probability. The first position scanned in each new window is determined based on the last occurrence of the c_block in the pattern. Depending on this position, the initial state of the automaton is also set up newly. These help our algorithm to scan each letter of the text at most once and reflect exactly the appearance of a part of the pattern at any position in the text.

The total number of all letters of y accessed by our algorithm is $n + 2c$ for $1 \leq c \leq m$ in the worst case and is $\frac{cn}{m-c+1}$ for $1 \leq c \leq m$ in the best case. Experimental results show that in practice our algorithm is faster than the above mentioned algorithms in most of the cases of given patterns and alphabets.

The rest of the paper is organized as follows. In Section 2, we recall some terminologies and definitions used in sequel [4, 8, 10, 17, 19, 22]. Section 3 proposes a new approach by using the automaton to design a pattern matching algorithm (the MR_c algorithm). Also some theoretical analyses of the MR_c algorithm are discussed. In Section 4, the experimental results comparing our algorithm with the 10 algorithms [12] are presented in the tables. Finally, we draw some conclusions from our approach and experimental results in Section 5.

2. PRELIMINARIES

Let Σ be a finite set which is called an alphabet. The number of all elements in the alphabet Σ is denoted by $|\Sigma|$. We call an element of Σ a letter. A finite sequence of n letters of Σ is called a string of length n over Σ and a string s can be represented by

$$s = s[1]s[2]..s[n],\ s[i] \in \Sigma,\ 1 \leq i \leq n,$$

where n is a positive integer.

Denote a special string, the empty string having no letters called the empty string, by ϵ. Denote the number of letters in the string s called the length of it by $|s|$. The length of the empty string is 0.

The set of all strings on the alphabet Σ is denoted by Σ^*. The operator of strings is concatenation that writes strings as a compound. Denote the concatenation of the two strings s_1 and s_2 by s_1s_2.

Given strings x, u_1 and u_2, if $x = u_1su_2$, then call the string s a substring of the string x. Call the string s a prefix (resp. suffix) of the string x if $u_1 = \epsilon$ (resp. $u_2 = \epsilon$). Call the

prefix (resp. suffix) s to be proper if $s \neq x$. Note that the prefix or the suffix can be empty.

Let y be a string of length n, the i^{th} element of y is denoted by $y[i]$ and call i a position in y. With $1 \leq i \leq j \leq n$, denote the substring $y[i]y[i+1]..y[j]$ of y by $y[i..j]$. For m is a positive integer, if a string x of length m is a substring of y, then $\exists i, n-m+1 \geq i \geq 1$ such that $y[i..i+m-1] = x$. Then we call that x occurs in y at position i or a match for x in y occurs at position i.

Below we recall the definition of the exact pattern matching problem.

Definition 2.1 ([19]). Let x be a pattern of length m and y be a text of length n over the alphabet Σ. Then the exact pattern matching problem is to find all occurrences of the pattern x in y.

To illustrate the most original and simple way to solve this problem, we use the Brute Force (BF) algorithm [8] in the following example.

Example 2.2. We are given a pattern x = fah, and a text y = dfahfkfaha. Then there are two occurrences of x in y as shown below dfahfkfaha. The BF algorithm is performed by the following steps presented in Table 1 (when comparing the letters of the pattern and the text, the bold letters correspond to the mismatches, the underlined letters represent the matches). We know that many letters scanned will be scanned again by the BF algorithm because each time either a mismatch or a match occurs, the pattern is only moved to the right one position.

Table 1. The performing steps of the BF algorithm

Step	y	d	f	a	h	f	k	f	a	h	a
1	x	**f**	a	h							
2	x		f	a	h						
3	x			**f**	a	h					
4	x				**f**	a	h				
5	x					**f**	a	h			
6	x						**f**	a	h		
7	x							f	a	h	
8	x								**f**	a	h

In our work, the degree of fuzziness in [17] is used to determine the longest prefix of the pattern in the text at any position. However, this terminology can lead to several misunderstandings for the readers. So throughout this paper, we will replace the degree of fuzziness with the degree of appearance. Next, we restate the concept of the degree of appearance.

Definition 2.3 ([17]). Let x be a pattern and y be a text of length n over the alphabet Σ. Then for each $1 \leq i \leq n$, a degree of appearance of x in y at position i is equal to the length of a longest substring of y such that the substring is a prefix of x, where the right end letter of this substring is $y[i]$.

Notice that obviously, if the degree of appearance of x in y at an arbitrary position i equals $|x|$, then a match for x in y occurs at position $i - |x| + 1$. Figure 2 illustrates the concept of the degree of appearance of the pattern x in y.

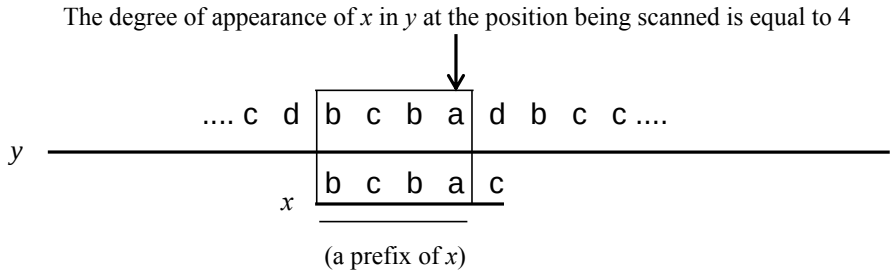

Figure 2. The degree of appearance of the pattern x

3. THE NEW ALGORITHM - THE MR$_c$ ALGORITHM

In this section, based on the degree of appearance recalled in Section 2, we show a way to build an automaton corresponding to a given pattern to accept the pattern (Theorem 3.5), present a new approach by using the automaton to design a pattern matching algorithm (the MR$_c$ algorithm). Some theoretical results which analyze the MR$_c$ algorithm are also considered (Propositions 3.11, 3.12 and 3.13, Theorem 3.14).

According to our design of the automaton, each state of the automaton is a degree of appearance. So, to determine the degree of appearance, we give the following concept.

Definition 3.1. Given a pattern x of length m. Then $Next$ of x is a function such that $Next : \{1, \ldots, m\} \to \{0, \ldots, m-1\}$ defined by

$$Next(l) = \max\{|s| : s \text{ is both a proper prefix and a suffix of } x[1..l]\}$$

for $l \in \{1, \ldots, m\}$.

Example 3.2. Given a pattern $x =$ acbac. Then $Next$ of x is determined as follows.

l	1	2	3	4	5
$Next$	0	0	0	1	2

The following algorithm is used to compute the $Next$ of the given x.

Algorithm 1

Input: An arbitrary pattern x with length m;
Output: The $Next$ of x;

```
1:  Next[1] = 0;
2:  for l = 2 to m do
3:     k = Next[l − 1];
4:     while (k > 0 and x[k + 1] ≠ x[l]) do
5:        k = Next[k];
6:     end while
7:     if (k == 0 and x[k + 1] == x[l]) then
8:        Next[l] = 1;
9:     end if
10:    if (k == 0 and x[k + 1] ≠ x[l]) then
11:       Next[l] = 0;
12:    end if
13:    if (k ≠ 0) then
14:       Next[l] = k + 1;
15:    end if
```

16: **end for**
17: **return** *Next*;

The correctness of the Algorithm 1 is confirmed by the following theorem.

Theorem 3.3. *For any given pattern x of length m, Algorithm 1 correctly computes the Next of x.*

Proof. By an application of Definition 3.1, $Next[1] = 0$, then (1) is true. Consider $l = 2$, then $k = Next[l-1] = Next[1] = 0$. Thus, clearly, either the block of (7), (8) and (9) or the block of (10), (11) and (12) correctly computes $Next[2]$ by Definition 3.1. Assume that Algorithm 1 correctly computes $Next[l]$ for all $2 \leq l < m$. We must prove that it is also true for $l+1$. By (3), $k = Next[l]$, at the end of the block of (4), (5) and (6), then either $k = 0$ or there exists a maximum number k such that $x[1..k+1] = x[l-k..l]$. In the case $k = 0$ the proof is analogous to the case $l = 2$. In the contrary case, by Definition 3.1, therefore, the block of (13), (14) and (15) correctly computes the *Next* of x. ∎

Based on the function *Next* of x, the following lemma offers us a way to compute the degree of appearance of x at any position when the degree of appearance of x at the adjacent position ahead is given.

Lemma 3.4. *Given a pattern x, a text y on Σ and assume that the degree of appearance of x in y at the position i equals $l, 0 \leq l \leq |x|$. Then the degree of appearance l' at the position $i+1$ in y is computed by the formula $l' = Appearance(l, a)$, where $a = y_{i+1}$ and the function Appearance corresponding to x is defined by*

$$\text{Appearance}(l,a) = \begin{cases} 0 & a \notin x; \text{ or } l = 0, a \neq x[1], \quad (3.1a) \\ 1 & a = x[1], l = 0, \quad (3.1b) \\ l+1 & a = x[l+1] \text{ for } 0 < l < |x|, \quad (3.1c) \\ \text{Appearance}(Next(l), a) & l = |x|; \text{ or } 0 < l < |x|, a \neq x[l+1]. \quad (3.1d) \end{cases}$$

Proof. From Definition 2.3, (3.1a), (3.1b) and (3.1c) are true. The equation (3.1d) has two cases. Consider the first case of (3.1d) $0 < l < |x|$ and $a \neq x[l+1]$, then $l = Next(l)$, if $a \neq x[l+1]$ then the statement $l = Next(l)$ is repeated until one of the possibilities (3.1a), (3.1b) and (3.1c) occurs, as the proof above leads to this case is true. In the second case of (3.1d) $l = |x|$, since $Next(l) < l$, then the statement $l = Next(l)$ is done once and it follows $0 < l < |x|$. Then at this time, it means that the second case returns to the first case. So, this case is true. ∎

Next, we construct an automaton corresponding to a given pattern to accept the pattern in the following theorem by Lemma 3.4.

Theorem 3.5. *Given a pattern x of length m and an automaton $M_x = (\Sigma, Q, q_0, \delta, F)$ corresponding to x on the same alphabet Σ, where*

- *The set of states $Q = \{0, 1, \ldots, m\}$,*
- *The initial state $q_0 = 0$,*
- *The set of final states $F = \{m\}$,*
- *The transition function $\delta : Q \times \Sigma \to Q$ such that $\delta(q, a) = Appearance(q, a)$, where the function Appearance corresponding to x defined as in Lemma 3.4,*
- *To accept an input string, the transition function δ is extended as follows*

$$\delta : Q \times \Sigma^* \to Q$$

such that $\forall q \in Q$, $\forall s \in \Sigma^*$, $\forall a \in \Sigma, \delta(q, as) = \delta(\delta(q, a), s)$ and $\delta(q, \epsilon) = q$.

Then M_x accepts the pattern x.

Proof. By Lemma 3.4, clearly Appearance$(i, x[i+1]) = i+1$ for all $0 \le i < m$. Based on the construction of the M_x as above, for any input pattern x,

$$\begin{aligned}\delta(q_0, x) &= \delta(\delta(0, x[1]), x[2..m]) = \delta(\text{Appearance}(0, x[1]), x[2..m]) \\ &= \delta(\text{Appearance}(1, x[2]), x[3..m]) = \ldots = \delta(\text{Appearance}(m-1, x[m]), \epsilon) = \delta(m, \epsilon) = m \in F.\end{aligned}$$

Thus M_x accepts x. ∎

Example 3.6. Consider a pattern $x =$ abcba and the automaton M_x given as in Theorem 3.5, denote an arbitrary letter, which is not in x, by the symbol #. Then transition function δ is represented by the following table.

δ	a	b	c	#
0	1	0	0	0
1	1	2	0	0
2	1	0	3	0
3	1	4	0	0
4	5	0	0	0
5	1	2	0	0

To make the window slide more regularly while the text is being scanned, we use the breaking points defined as follows.

Definition 3.7. Let x be a pattern and y be a text over the alphabet Σ. Consider a position i in y for $1 \le i < |y|$ and the degree of appearance of x in y at positions i and $i+1$ are q and q', respectively. Then the position i is called a breaking point if one of the two following conditions is satisfied.

(a) $q = 0$,

(b) $q' \le q < |x|$.

Notice that if only the condition (a) in Definition 3.7 is satisfied, then breaking points are called type a breaking points.

One of problems that needs to solve in our approach is to avoid scanning each letter of the text repeatedly. So, we give some terminologies and a concept below.

Given a positive integer c, a string of length c is called a c_block. A c_block is called (resp. not) to be in x, denoted by c_block $\in x$ (resp. c_block $\notin x$), if the c_block is (resp. not) a substring of x. For a given positive integer $c, 1 \le c \le m$ and $c \le i \le m$, the substring $x[i-c+1..i]$ is called a c_block of x at position i, denoted by $c\text{\_block}_x^i$. In particular, $c = 1$, then c_block is only a letter.

Definition 3.8. Let x be a pattern and z be a c_block of x, where c is a positive integer for $1 \le c \le |x|$. Let i be some position in x for $c \le i \le |x|$. Then i is called the last position of appearance of z in x, denoted by $Pos(z)$, if $z = \text{c\_block}_x^i$ and $\forall j > i, j \le |x|, z \ne \text{c\_block}_x^j$.

Based on the automaton M_x and the two concepts of the breaking point and Pos, we present the basic idea of our new approach to exact pattern matching as follows:

1. Use the automaton M_x to reflect and update the degree of appearance of x in y at any position. If the degree of appearance of x at some position equals $|x|$, then mark an occurrence of x in y.

2. Take advantage of the relationship between the size of the pattern x and the alphabet Σ flexibly, and use the breaking point. These make our approach flexible and the window be shifted more regularly to the right by the most possible maximum number of positions with high probability.

3. A new window is scanned from left to right. The window scanning process is only started when the right end c_block of the window belongs to x (do the test jump). The first position scanned in the window (called the backtracking position) depends on the Pos value of the c_block and the initial state of M_x will be set up again. These make our approach scan each letter of the text at most once and reflect exactly the appearance of a part of the pattern at any position in the text. The window scanning process will be stopped if the breaking point occurs. At this moment, one next window is determined (slide the window) and immediately do the next test jump. Repeat this mechanism until reach out of y. This mechanism helps our approach to design a pattern matching algorithm effectively.

   Figure 3 shows our approach.

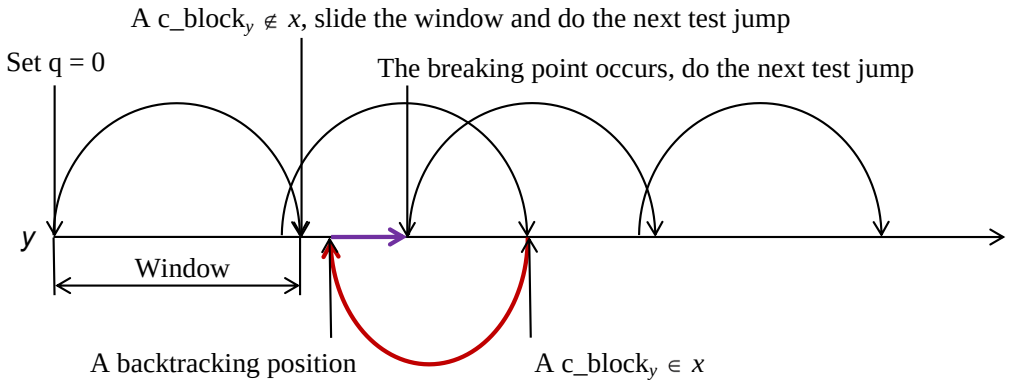

Figure 3. The basic idea of our approach

Based on our approach, we construct a pattern matching algorithm. The pseudo-code for our algorithm (called the MR$_c$ algorithm) is shown as follows.

Algorithm 2 (The MR$_c$ algorithm)

Input: Two arbitrary pattern x and text y;
Output: All occurrences of the pattern x in y;

```
1:  q = 0;;
2:  jump = |x|;
3:  while (jump ≤ |y|) do
4:    if (c_block_y^jump ∈ x) then
5:      bt = Pos(c_block_y^jump);
6:      if (q == 0) then
7:        i = jump − bt + 1;
8:      else if (bt ≤ |x| − q′) then
9:        q = 0;
10:       i = jump − bt + 1;
11:     else
12:       q = q′;
13:       i = jump − |x| + q′ + 1;
14:     end if
15:     q′ = δ(q, y[i]);
16:     do
17:       q = q′;
18:       if (q == |x|) then
19:         Mark an occurrence of x at i − |x| + 1 in y;
20:       end if
```

```
21:          i++;
22:          if (i ≤ |y| and q ≠ 0) then
23:            q' = δ(q', y[i]);
24:          else
25:            break;
26:          end if
27:        while (q ≠ 0 and q' > q or q == |x|);
28:        if (q == 0) then
29:          jump = i − 1;
30:        else
31:          jump = i − q';
32:        end if
33:      else
34:        jump = jump − c + 1;
35:        if (q ≠ 0) then
36:          q = 0;
37:        end if
38:      end if
39:      jump = jump + |x|;
40:  end while
```

The performing steps of the MR_c for $c = 1$ is illustrated in the following example.

Table 2. The performing steps of the MR_1 algorithm

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$y[i]$	a	b	a	b	c	b	a	d	a	b	e	e	g	a	t	k	a	u
Jump Test Back					$jump$=5 c $\in x$ $i = 3$								$jump$=13 g $\notin x$					$jump$=18 u $\notin x$
q=0	Ignore		1	2	3	4	5	0	Ignore									

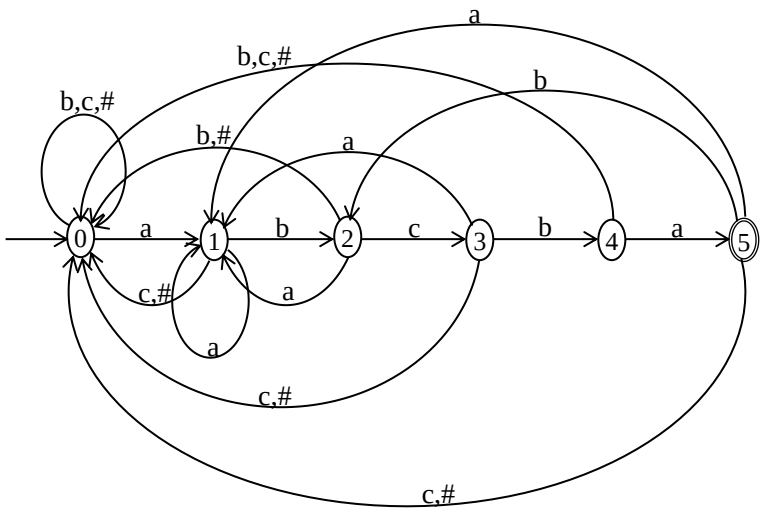

Figure 4. The transition diagram of M_x, x = abcba

Example 3.9. Consider a pattern x = abcba and the automaton M_x given by Theorem 3.5, denote an arbitrary letter, which is not in x, by the symbol #. Then the transition diagram of M_x is shown in Figure 4. Here, we use the MR_1 to find x in y. It is easy to compute Pos values for the given x above, $Pos(a) = 5$, $Pos(b) = 4$, $Pos(c) = 3$. Let a text y = ababcbadabeegatkau. Then the whole process of the pattern matching of the MR_1 is presented in Table 2.

The correctness of the MR_c algorithm is guaranteed by the following theorem.

Theorem 3.10. *For any given pattern x and text y, the MR_c algorithm finds all occurrences of the pattern x in y.*

Proof. Assume that the process of the scanning y starts at the position 1 and i is the position being scanned in y. Algorithm 2 is only correct if the following two cases are true:

(a) The state q of automaton M_x is the degree of appearance of x in y at position i.

(b) The process of the scanning y does not ignore any occurrences of x.

Suppose that if (a) and (b) are true, then when $q = m$ at the position i, this means x occurs at the position $i - m + 1$ in y by Definition 2.3, at this time the occurrence is marked immediately by the block of (18), (19) and (20). So all occurrences of x in y are not never ignored.

Prove (a): The process of the scanning y only occurs if $\text{c\_block}_y^{jump} \in x$. Starting with i determined by (5), the block of (6) and (7), (10) and (13), the state of M_x is always initialized newly before scanning y by (1), (9), (12) and the block of (35), (36) and (37), after each time y is scanned by (15) and the block of (22) and (23), i increases by one unit by (21) when the conditions in (22) and (27) hold. Since δ is defined as in Theorem 3.5, this leads to q is always the degree of appearance of x in y at position i. So (a) is true.

Prove (b): Note that *jump* is a position in y to test whether or not c_block_y^{jump} is x. If $\text{c\_block}_y^{jump} \notin x$ in (4), then the substring $y[jump - m + 1..jump]$ of length m does not contain x, but $\text{c\_block}_y^{jump+1}$ can belong to x. Thus $y[jump - m + 1..jump\text{-}c\text{+}1]$ must not be scanned by (2), (34) and (39), it means that the window is shifted to the right by $m - c + 1$ letters and *jump* is always the left end of this new window. So (b) is true. Conversely, if $\text{c\_block}_y^{jump} \in x$ in (4), then the next process of the scanning y starts with the new current position i determined by (5), the block of (6) and (7), (10) and (13). By the Definitions 2.3, 3.8 and the way to determine i, all occurrences of x are not ignored. Thus (b) is true and the proof is complete. ∎

The theoretical analyses of the MR_c algorithm given in Propositions 3.11, 3.12, 3.13 and Theorem 3.14 are as follows.

Proposition 3.11. *Let x be a pattern of length m and y be a text of length n over the alphabet Σ. Let c be a positive integer constant such that $1 \le c \le m$. Then the MR_c algorithm requires $\dfrac{cn}{m - c + 1}$ letters of y accessed in the best case.*

Proof. In the best case all letters of y are not matched in whole process of the pattern matching, y is only accessed by (4) in the MR_c algorithm. Let $T(n)$ be the number of all letters of y accessed by the MR_c algorithm. Since each time (4) is performed and $\text{c\_block}_y^{jump} \notin x$, the window is shifted by $m - c + 1$ positions and c letters of y are accessed. Thus $T(n) = \dfrac{cn}{m - c + 1}$. ∎

Proposition 3.12. *Let x be a pattern of length m and y be a text of length n over the alphabet Σ. Let c be a positive integer constant such that $1 \le c \le m$. Then MR_c algorithm requires $n + 2c$ letters of y accessed in the worst case.*

Proof. Similarly, as in the proof of Proposition 3.11, $T(n)$ is the number of all letters of y accessed by the MR_c algorithm. The worst case occurs when all letters are matched in whole process of the pattern matching. By the block of (6) and (7), (15), the block of (22) and (23), and (27), y is scanned once for every letter, this implies that n letters are accessed. Clearly, the two statements (4) and (5) in the MR_c algorithm are only performed once in the whole process of the matching, then there are $2c$ letters of y accessed by them. Thus $T(n) = n + 2c$. ∎

We use the notation p to denote the probability of an arbitrary event.

Proposition 3.13. *Let x be a pattern of length m over the alphabet Σ. If $|\Sigma| \geq 4$ and $8 \leq m \leq 2048$, then there exists $c, 1 \leq c \leq 8$ such that for an arbitrary c_block z over the alphabet Σ, $p(z \in x) \leq 2^{-5}$ with a uniform distribution over the alphabet Σ.*

Proof. Set $P = p(z \in x)$. Let d be the number of different *c_blocks* in x. Then for a uniform distribution over Σ, clearly we have

$$P = p(z \in x) = \frac{d}{|\Sigma|^c} \leq \frac{m - c + 1}{|\Sigma|^c}.$$

Consider the case $1 \leq c$, then

$$P \leq \frac{m}{|\Sigma|^c}.$$

To have $P \leq 2^{-5}$, we let

$$\frac{m}{|\Sigma|^c} \leq 2^{-5},$$

or equivalently

$$2^{-5}|\Sigma|^c \geq m. \tag{3.2}$$

For (3.2) hold with $|\Sigma| \geq 4$ and $8 \leq m \leq 2048$, we choose $c, 1 \leq c \leq 8$ such that $2^{2c-5} \geq m$. Let $2^{2c-5} \geq 2048$, then we can choose $c = 8$. ∎

Theorem 3.14. *Let x be a pattern of length m and y be a text of length n over the alphabet Σ. Let $T(n)$ be the number of all letters of y accessed by the MR_c algorithm. If $|\Sigma| \geq 4, 16 \leq m \leq 2048$ or $|\Sigma| \geq 32, 8 \leq m \leq 2048$, then there exists $c, 1 \leq c \leq 8$ such that the two following conditions are satisfied with a uniform distribution over Σ.*

(a) $T(n) < n$,

(b) $p(z \in x) \leq 2^{-5}$, where z is an arbitrary c_block over the alphabet Σ.

Proof. Let S be the set of all letters of y scanned by (15) and the block of (22) and (23). Let I be the set of all unscanned letters of y. Actually, I consists of $(m - c + 1)$_blocks or $(m - q - c + 1)$_blocks, where q is a degree of appearance of x in y at a position after some breaking point. Note that (13) helps the MR_c algorithm to avoid scanning y again, it means that any letter is only scanned once. Thus $n = |S| + |I|$.

Let T_1 (resp. T_2) be the set of all c_blocks of y tested by (4) such that these c_blocks $\notin x$ (resp. c_blocks $\in x$) and BT be the set of all c_blocks of y accessed by (5). Clearly, in the case a c_block $\in x$, this c_block is both always accessed only once by (4) and (5). Thus $T_2 = BT$. Then $T(n) = |S| + |T_1| + |T_2| + |BT| = n - |I| + |T_1| + 2|T_2|$.

Set $P = p(z \in x)$, then $p(z \notin x) = 1 - P$.

Suppose that c is chosen so that (b) satisfies. Then clearly, $P \ll 1 - P$. This implies that $|T_2| \ll |T_1|$. Thus we can remove $2|T_2|$ in the formula of $T(n)$. So the new formula of $T(n)$ is $n - |I| + |T_1|$ given $P \ll 1 - P$. Obviously, by the block of (28) and (29), the block of (30) and (31), (34) and (39), an arbitrary c_block $\in T_1$ corresponds to one to one either (m-c+1)_block or

(m-q-c+1)_block belonging to I. Then to have $T(n) < n$, we need $|T_1| < |I|$, this means $m-c+1 > c$ and $m-q-c+1 > c$. For given $P \ll 1-P$, then $q < c$, thus $m-2c+1 < m-q-c+1$. Therefore

$$c < m-c+1 \Rightarrow c < \frac{m+1}{2}.$$

To have $m-q-c+1 > c$, we let

$$c < m-2c+1 \Rightarrow c < \frac{m+1}{3}.$$

Since $\frac{m+1}{3} < \frac{m+1}{2}$, thus to have $T(n) < n$ with supposing that given $P \ll 1-P$, we need to choose c such that

$$1 \leq c < \frac{m+1}{3}. \tag{3.3}$$

Now, to have (b), similarly, as in the proof of Proposition 3.13, we need to have (3.2). By the hypotheses of Σ and m, we can choose c such that (3.3) and (b) hold as follows.

- The case $|\Sigma| \geq 4$; $16 \leq m \leq 32$: Let $2^{2c-5} \geq 32$ (c holds (b)) and to satisfy (3.3), we can only choose $c = 5$.
- The case $|\Sigma| \geq 4$; $32 < m \leq 2048$: Let $2^{2c-5} \geq 2048$, then choose $c = 8$.
- The case $|\Sigma| \geq 32$; $8 \leq m \leq 16$: Let $2^{5c-5} \geq 16$, then choose $c = 2$.
- The case $|\Sigma| \geq 32$; $16 < m \leq 2048$: Let $2^{5c-5} \geq 2048$, then choose $c = 4$.

We complete the proof. ∎

From the theoretical analyses in Propositions 3.11, 3.12, 3.13 and Theorem 3.14, we can provide an explanation of why the MR_c algorithm seems to be better for large alphabets and the length of the text when comparing with other methods [12] as follows.

- First, according to our knowledge, the 10 algorithms have some common features: they first do not take advantage of the relationship between the size of the pattern and the alphabet, then they are not flexible to shift the window regularly to the right by the most possible maximum number of positions with high probability (except for HASHq, SBNDMq). Next, the appearance of a part of the pattern is not immediately reflected or updated at a position being scanned in the text, hence when the windows overlap, the letters of the text can be scanned more than once (except for SA, but this algorithm does not use the sliding window mechanism).

- Second, the MR_c algorithm never scans any letter of the text more than once (proof of Theorem 3.14).

- Third, in our approach, shifting the window depends on the probability

$$P = \frac{d}{|\Sigma|^c},$$

where d is the number of different *c_blocks* in the pattern, $|\Sigma|$ is the size of the alphabet, c is a positive integer, $1 \leq c \leq m$, m is the length of the pattern (proof of Proposition 3.13). If P is small, then sliding the window will usually occurs. For a given pattern, d is a constant. Hence,

P is only small if $|\Sigma|^c$ is big. Suppose Σ is a large alphabet, then by Proposition 3.13 and Theorem 3.14, the constant c can be small such that $|\Sigma|^c$ is still big. On the other hand, when c is small, the size of a *c_block* will be small. It follows the number of letters of all *c_blocks* accessed will be reduced (proofs of Propositions 3.11, 3.12, 3.13 and Theorem 3.14). Then the total number of letters of the text accessed decreases. In addition, each time the window is shifted, the number of letters of the text which are ignored is $m-c+1$ or $m-q-c+1$, where q is a degree of appearance of the pattern in the text at a position after some breaking point (proof of Theorem 3.14). For c is small, $m-c+1$ or $m-q-c+1$ will be small, then the total number of letters of the text accessed also decreases.

In brief, MR_c will be more efficient for large alphabets. Further, obviously, if the length of the text is large, then MR_c takes the advantage of shifting the window the most effectively.

4. EXPERIMENTAL RESULTS

In this section, we make a number of experiments to compare the MR_c algorithm with 10 algorithms whose codes can be found at http://www.dmi.unict.it/∼faro/smart/ [12]. We used test data of Faro et al. [12] and generated new test data for the alphabet of size 256 to determine the fastest algorithms.

All algorithms are implemented in the C# programming language, compiled Microsoft Visual Studio 2010. We ran experiments in 64-bit Operating System (Win 7), Intel Core I3, 2.20GHz, 4 GB RAM.

We used the MR_c algorithm with the 9 following versions:

- MR_1: $c = 1$, use the type a breaking point.
- MR_1-Pos: $c = 1$, use the type a breaking point and Pos.
- MR_2: $c = 2$, use the type a breaking point.
- MR_2-Pos: $c = 2$, use the type a breaking point and Pos.
- MR_2-Pos-Breaking: $c = 2$, use the breaking point and Pos.
- MR_c: $c = 3, 4, 5, 8$, use the breaking point and Pos.

We used the following test data:

- For each alphabet $\Sigma, |\Sigma| = 4, 8, 16, 32, 64, 128$, we use a text y over the Σ with a uniform distribution of letters consisted in a rand$|\Sigma|$ file of length 5MB. In cases $|\Sigma| = 4, 20$, we additionally use files containing genome and protein sequences. All these files can be downloaded from website http://www.dmi.unict.it/∼faro/smart/corpus.php [12]. Just because the rand256 file corresponding to the alphabet of size 256 is not available now on this website, it follows that we must use a text y which is a file of length about 24MB to ensure randomness of the text and consider the value of each byte of the file to be a code of a letter of the alphabet of size 256.

- For each text y over the alphabet Σ with $|\Sigma| = 4, 8, 16, 20, 32, 64, 128$ (resp. 256), we generate sets of 400 (resp. 83) patterns of fixed length m randomly extracted from the text, for m ranging over the values 8, 16, 32, 64, 128, 256, 512, 1024 and 2048.

- For each set of patterns over the alphabet Σ with $|\Sigma| = 4, 8, 16, 20, 32, 64, 128$ (resp. 256), the mean over the running times of the 400 (resp. 83) runs is reported in a table.

Experimental results are presented in the following tables. Each table corresponds to a size of an alphabet. The rows 3 through 17 of all tables are results of 10 algorithms [12]. The rest of the tables are results of our algorithm. Running times are expressed in seconds. The best results are printed in bold.

According to our experimental results as below, our MR_c algorithm is the fastest in most of the cases of given patterns and alphabets, except for the following cases:

- $|\Sigma| = 4, m = 32$: SBNDM4 is the fastest (for Rand4 and Genome Sequence).
- $|\Sigma| = 8, m = 8$: SBNDM2 is the fastest.
- $|\Sigma| = 20, m = 32$: SBNDM2 is the fastest (for Protein Sequence).
- $|\Sigma| = 64, m = 2048$: TVSBS, SBNDM6, SBNDM8 are similar to MR_2.
- $|\Sigma| = 128, m = 2048$: SBNDM4, SBNDM6 are similar to MR_2 and MR_2-Pos.

Experimental Results on Rand4 Problem									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0694	0.0734	0.0688	0.0670	0.0606	0.0607	0.0587	0.0512	0.0450
TVSBS	0.0453	0.0365	0.0294	0.0270	0.0258	0.0263	0.0244	0.0252	0.0223
SA	0.0695	0.0802	0.0806	-	-	-	-	-	-
BMH-SBNDM	0.0534	0.0423	0.0275	0.0191	0.0134	0.0124	0.0067	0.0037	0.0018
EBOM	0.0405	0.0310	0.0193	0.0116	0.0070	0.0047	0.0024	0.0014	0.0009
AOSO	0.1355	0.0617	0.0155	0.0702	0.1939	0.3102	0.3534	0.3703	0.3680
FSBNDM	0.0334	0.0207	0.0118	0.0093	0.0082	0.0095	0.0049	0.0024	0.0013
HASH3	0.0418	0.0247	0.0175	0.0144	0.0135	0.0126	0.0111	0.0111	0.0115
HASH5	0.0844	0.0332	0.0165	0.0097	0.0069	0.0058	0.0045	0.0044	0.0046
HASH8	0.0334	0.0574	0.0220	0.0107	0.0062	0.0047	0.0032	0.0029	0.0029
SBNDM2	0.0311	0.0203	0.0119	0.0095	0.0085	0.0093	0.0052	0.0025	0.0013
SBNDM4	0.0356	0.0165	**0.0086**	0.0085	0.0079	0.0084	0.0049	0.0025	0.0013
SBNDM6	0.0801	0.0254	0.0109	0.0067	0.0073	0.0082	0.0049	0.0023	0.0012
SBNDM8	0.3083	0.0399	0.0151	0.0073	0.0069	0.0079	0.0046	0.0023	0.0012
LBNDM	0.0385	0.0254	0.0150	0.0131	0.0131	0.0465	0.2750	0.2488	0.2041
MR_3	**0.0301**	0.0210	0.0170	0.0130	0.0110	0.0106	0.0102	0.0103	0.0093
MR_4	**0.0291**	**0.0145**	0.0088	**0.0059**	0.0048	0.0043	0.0038	0.0036	0.0034
MR_8	0.4603	0.0311	0.0123	0.0061	**0.0031**	**0.0021**	**0.0013**	**0.0006**	**0.0004**

Experimental Results on Rand8 Problem									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0379	0.0293	0.0262	0.0269	0.0254	0.0235	0.0228	0.0218	0.0202
TVSBS	0.0256	0.0161	0.0101	0.0069	0.0054	0.0049	0.0050	0.0049	0.0050
SA	0.0679	0.0698	0.0714	-	-	-	-	-	-
BMH-SBNDM	0.0316	0.0231	0.0181	0.0146	0.0105	0.0081	0.0064	0.0042	0.0023
EBOM	0.0207	0.0131	0.0092	0.0063	0.0041	0.0028	0.0017	0.0011	0.0007
AOSO	0.0797	0.0234	0.0130	0.0281	0.1143	0.1276	0.2526	0.3590	0.3639
FSBNDM	0.0202	0.0120	0.0076	0.0053	0.0041	0.0042	0.0043	0.0026	0.0013
HASH3	0.0380	0.0193	0.0112	0.0081	0.0064	0.0060	0.0057	0.0055	0.0053
HASH5	0.0817	0.0299	0.0137	0.0077	0.0046	0.0039	0.0032	0.0029	0.0027
HASH8	0.0202	0.0549	0.0211	0.0104	0.0058	0.0044	0.0036	0.0032	0.0031
SBNDM2	**0.0185**	0.0112	0.0073	0.0052	0.0042	0.0044	0.0044	0.0028	0.0013
SBNDM4	0.0350	0.0142	0.0067	0.0038	0.0038	0.0036	0.0042	0.0027	0.0013
SBNDM6	0.0815	0.0232	0.0097	0.0048	0.0030	0.0034	0.0041	0.0026	0.0012
SBNDM8	0.3151	0.0365	0.0135	0.0064	0.0033	0.0032	0.0041	0.0027	0.0012
LBNDM	0.0280	0.0153	0.0092	0.0069	0.0057	0.0058	0.0177	0.1579	0.1637
MR_3	0.0199	**0.0095**	0.0053	0.0035	0.0027	0.0024	0.0022	0.0020	0.0020
MR_4	0.0266	0.0107	**0.0052**	**0.0029**	**0.0019**	**0.0012**	0.0008	**0.0006**	0.0006
MR_5	0.0402	0.0142	0.0068	0.0034	0.0023	0.0012	**0.0007**	0.0006	**0.0005**

Experimental Results on Rand16 Problem									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0309	0.0188	0.0149	0.0139	0.0135	0.0136	0.0129	0.0121	0.0117
TVSBS	0.0232	0.0127	0.0073	0.0043	0.0030	0.0027	0.0021	0.0019	0.0019
SA	0.0778	0.0716	0.0758	-	-	-	-	-	-
BMH-SBNDM	0.0261	0.0156	0.0121	0.0100	0.0084	0.0071	0.0053	0.0043	0.0031
EBOM	0.0185	0.0091	0.0054	0.0038	0.0031	0.0022	0.0013	0.0009	0.0007
AOSO	0.0542	0.0161	0.0136	0.0177	0.0777	0.0654	0.1640	0.2401	0.3519
FSBNDM	0.0178	0.0090	0.0051	0.0040	0.0029	0.0025	0.0023	0.0020	0.0014
HASH3	0.0396	0.0184	0.0094	0.0058	0.0043	0.0039	0.0035	0.0033	0.0031
HASH5	0.0881	0.0297	0.0135	0.0068	0.0040	0.0033	0.0027	0.0024	0.0024
HASH8	0.0178	0.0539	0.0204	0.0101	0.0056	0.0044	0.0035	0.0031	0.0030
SBNDM2	0.0172	0.0086	0.0048	0.0038	0.0030	0.0026	0.0023	0.0020	0.0014
SBNDM4	0.0391	0.0149	0.0068	0.0035	0.0024	0.0023	0.0021	0.0021	0.0013
SBNDM6	0.0901	0.0249	0.0102	0.0048	0.0027	0.0017	0.0020	0.0020	0.0013
SBNDM8	0.3498	0.0385	0.0141	0.0063	0.0033	0.0018	0.0018	0.0019	0.0013
LBNDM	0.0252	0.0137	0.0070	0.0048	0.0036	0.0034	0.0034	0.0091	0.1012
MR_2	**0.0148**	**0.0075**	0.0050	0.0039	0.0036	0.0037	0.0046	0.0172	0.2119
MR_3	0.0213	0.0087	0.0063	0.0036	**0.0018**	**0.0011**	**0.0007**	**0.0006**	0.0005
MR_4	0.0332	0.0123	**0.0045**	**0.0025**	0.0022	0.0013	**0.0007**	**0.0006**	**0.0004**

Experimental Results on Rand32 **Problem**									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0222	0.0126	0.0087	0.0069	0.0058	0.0057	0.0059	0.0056	0.0055
TVSBS	0.0180	0.0100	0.0055	0.0031	0.0025	0.0018	0.0012	0.0008	0.0006
SA	0.0646	0.0637	0.0638	-	-	-	-	-	-
BMH-SBNDM	0.0189	0.0109	0.0070	0.0053	0.0046	0.0044	0.0038	0.0033	0.0026
EBOM	0.0144	0.0069	0.0038	0.0023	0.0018	0.0012	0.0008	0.0008	0.0006
AOSO	0.0292	0.0123	0.0117	0.0126	0.0433	0.0252	0.0980	0.1044	0.2051
FSBNDM	0.0132	0.0067	0.0036	0.0025	0.0021	0.0014	0.0012	0.0010	0.0006
HASH3	0.0347	0.0153	0.0075	0.0042	0.0029	0.0026	0.0022	0.0020	0.0019
HASH5	0.0735	0.0253	0.0115	0.0059	0.0035	0.0028	0.0023	0.0021	0.0021
HASH8	0.0132	0.0509	0.0189	0.0088	0.0050	0.0039	0.0032	0.0029	0.0028
SBNDM2	0.0140	0.0067	0.0035	0.0023	0.0020	0.0015	0.0013	0.0010	0.0010
SBNDM4	0.0333	0.0128	0.0059	0.0030	0.0020	0.0012	0.0011	0.0009	0.0006
SBNDM6	0.0773	0.0211	0.0087	0.0041	0.0023	0.0013	0.0008	0.0009	0.0006
SBNDM8	0.2988	0.0331	0.0121	0.0055	0.0028	0.0015	0.0008	0.0008	0.0006
LBNDM	0.0162	0.0105	0.0059	0.0033	0.0026	0.0021	0.0017	0.0017	0.0044
MR_2	**0.0097**	**0.0051**	**0.0029**	**0.0021**	**0.0015**	0.0010	0.0008	0.0009	0.0011
MR_2-Pos-Breaking	0.0130	0.0063	0.0035	0.0023	0.0017	0.0013	0.0012	0.0010	0.0006
MR_3	0.0188	0.0083	0.0043	0.0024	0.0017	**0.0009**	**0.0005**	**0.0004**	**0.0003**

Experimental Results on Rand64 **Problem**									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0204	0.0109	0.0065	0.0044	0.0040	0.0038	0.0037	0.0037	0.0038
TVSBS	0.0176	0.0098	0.0054	0.0030	0.0022	0.0014	0.0008	0.0006	**0.0004**
SA	0.0644	0.0639	0.0638	-	-	-	-	-	-
BMH-SBNDM	0.0171	0.0097	0.0056	0.0037	0.0036	0.0034	0.0032	0.0030	0.0025
EBOM	0.0153	0.0069	0.0038	0.0021	0.0016	0.0010	0.0006	0.0006	0.0006
AOSO	0.0202	0.0118	0.0117	0.0120	0.0277	0.0154	0.0691	0.0545	0.1360
FSBNDM	0.0126	0.0064	0.0034	0.0022	0.0016	0.0012	0.0008	0.0006	0.0005
HASH3	0.0375	0.0164	0.0080	0.0044	0.0028	0.0024	0.0020	0.0018	0.0018
HASH5	0.0741	0.0252	0.0113	0.0058	0.0035	0.0028	0.0023	0.0021	0.0021
HASH8	0.0126	0.0481	0.0176	0.0086	0.0049	0.0038	0.0031	0.0027	0.0027
SBNDM2	0.0137	0.0065	0.0033	0.0020	0.0015	0.0011	0.0008	0.0006	0.0005
SBNDM4	0.0331	0.0127	0.0059	0.0030	0.0020	0.0011	0.0006	0.0006	0.0005
SBNDM6	0.0770	0.0211	0.0087	0.0041	0.0024	0.0013	0.0007	0.0005	**0.0004**
SBNDM8	0.3013	0.0335	0.0123	0.0055	0.0029	0.0016	0.0009	0.0006	**0.0004**
LBNDM	0.0132	0.0081	0.0053	0.0032	0.0021	0.0015	0.0013	0.0010	0.0010
MR_1	0.0125	0.0094	0.0087	0.0085	0.0119	0.0743	0.1820	0.1857	0.1852
MR_2	**0.0124**	**0.0060**	**0.0031**	**0.0018**	**0.0013**	**0.0007**	**0.0005**	**0.0004**	**0.0004**
MR_2-Pos-Breaking	0.0132	0.0063	0.0034	0.0020	0.0014	0.0009	0.0006	0.0005	0.0005

Experimental Results on Rand128 **Problem**									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0191	0.0102	0.0056	0.0035	0.0032	0.0029	0.0026	0.0027	0.0026
TVSBS	0.0175	0.0100	0.0055	0.0032	0.0024	0.0013	0.0008	0.0006	0.0004
SA	0.0637	0.0638	0.0637	-	-	-	-	-	-
BMH-SBNDM	0.0164	0.0090	0.0048	0.0030	0.0030	0.0027	0.0024	0.0024	0.0024
EBOM	0.1992	0.0070	0.0037	0.0022	0.0016	0.0009	0.0006	0.0005	0.0005
AOSO	0.0160	0.0118	0.0115	0.0118	0.0199	0.0128	0.0438	0.0255	0.0974
FSBNDM	0.0123	0.0063	0.0033	0.0020	0.0014	0.0008	0.0007	0.0005	0.0004
HASH3	0.0380	0.0166	0.0082	0.0044	0.0028	0.0025	0.0020	0.0018	0.0018
HASH5	0.0759	0.0257	0.0115	0.0059	0.0035	0.0029	0.0023	0.0021	0.0021
HASH8	0.0123	0.0476	0.0177	0.0086	0.0048	0.0038	0.0030	0.0027	0.0027
SBNDM2	0.0136	0.0065	0.0033	0.0019	0.0014	0.0008	0.0006	0.0004	0.0004
SBNDM4	0.0329	0.0128	0.0059	0.0029	0.0020	0.0011	0.0006	0.0004	**0.0003**
SBNDM6	0.0765	0.0211	0.0087	0.0042	0.0024	0.0013	0.0007	0.0005	**0.0003**
SBNDM8	0.2958	0.0331	0.0121	0.0046	0.0028	0.0016	0.0009	0.0006	0.0004
LBNDM	0.0115	0.0067	0.0042	0.0029	0.0020	0.0012	0.0009	0.0007	0.0005
MR_1	**0.0098**	0.0064	0.0049	0.0046	0.0046	0.0066	0.0335	0.1793	0.1893
MR_2	0.0119	**0.0058**	**0.0030**	**0.0018**	**0.0013**	0.0011	**0.0004**	**0.0003**	**0.0003**
MR_2-Pos	0.0123	0.0060	0.0031	**0.0018**	**0.0013**	**0.0007**	**0.0004**	**0.0003**	**0.0003**

Experimental Results on Rand256 **Problem**									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0905	0.0456	0.0246	0.0142	0.0116	0.0103	0.0099	0.0087	0.0085
TVSBS	0.0888	0.0518	0.0306	0.0180	0.0117	0.0064	0.0039	0.0030	0.0019
SA	0.2998	0.2990	0.2988	-	-	-	-	-	-
BMH-SBNDM	0.0753	0.0386	0.0208	0.0124	0.0112	0.0099	0.0093	0.0081	0.0080
EBOM	0.0713	0.0350	0.0196	0.0114	0.0074	0.0045	0.0027	0.0017	0.0012
AOSO	0.0651	0.0549	0.0550	0.0548	0.0739	0.0560	0.1343	0.0737	0.3248
FSBNDM	0.0570	0.0290	0.0152	0.0093	0.0063	0.0037	0.0023	0.0019	0.0013
HASH3	0.2111	0.0756	0.0370	0.0200	0.0131	0.0109	0.0087	0.0082	0.0083
HASH5	0.3354	0.1153	0.0525	0.0271	0.0160	0.0131	0.0106	0.0100	0.0103
HASH8	0.0570	0.1554	0.0848	0.0397	0.0224	0.0176	0.0137	0.0125	0.0124
SBNDM2	0.0637	0.0303	0.0154	0.0088	0.0063	0.0039	0.0022	0.0017	0.0012
SBNDM4	0.1539	0.0596	0.0277	0.0139	0.0094	0.0052	0.0030	0.0021	0.0012
SBNDM6	0.3585	0.0992	0.0410	0.0194	0.0109	0.0062	0.0035	0.0025	0.0016
SBNDM8	1.3897	0.2292	0.0568	0.0256	0.0132	0.0074	0.0042	0.0029	0.0016
LBNDM	0.0506	0.0276	0.0163	0.0112	0.0083	0.0055	0.0034	0.0026	0.0017
MR_1	**0.0394**	**0.0235**	0.0160	0.0134	0.0121	0.0127	0.0159	0.0404	0.6889
MR_1-Pos	0.0407	0.0248	0.0170	0.0131	0.0115	0.0121	0.0101	0.0098	0.0102
MR_2	0.0582	0.0281	**0.0146**	**0.0087**	**0.0062**	**0.0033**	**0.0020**	**0.0013**	**0.0008**
MR_2-Pos	0.0628	0.0309	0.0172	0.0106	0.0067	0.0036	0.0021	0.0014	0.0010

Experimental Results on a Genome Sequence (with $\|\Sigma\| = 4$)									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0611	0.0540	0.0491	0.0459	0.0453	0.0433	0.0430	0.0433	0.0405
TVSBS	0.0388	0.0268	0.0207	0.0182	0.0180	0.0181	0.0181	0.0180	0.0183
SA	0.0594	0.0587	0.0580	-	-	-	-	-	-
BMH-SBNDM	0.0462	0.0314	0.0198	0.0130	0.0097	0.0087	0.0054	0.0029	0.0015
EBOM	0.0355	0.0233	0.0141	0.0082	0.0051	0.0033	0.0020	0.0012	0.0008
AOSO	0.1181	0.0466	0.0112	0.0498	0.1370	0.2139	0.2886	0.2996	0.3001
FSBNDM	0.0291	0.0157	0.0088	0.0067	0.0058	0.0067	0.0041	0.0020	0.0011
HASH3	0.0366	0.0191	0.0126	0.0101	0.0091	0.0090	0.0087	0.0087	0.0089
HASH5	0.0684	0.0241	0.0117	0.0069	0.0047	0.0040	0.0037	0.0036	0.0035
HASH8	0.0291	0.0454	0.0166	0.0081	0.0046	0.0036	0.0028	0.0025	0.0025
SBNDM2	0.0278	0.0159	0.0090	0.0069	0.0059	0.0068	0.0043	0.0021	0.0010
SBNDM4	0.0312	0.0128	**0.0065**	0.0062	0.0055	0.0062	0.0041	0.0021	0.0010
SBNDM6	0.0703	0.0192	0.0080	0.0048	0.0051	0.0060	0.0040	0.0020	0.0010
SBNDM8	0.2723	0.0300	0.0110	0.0051	0.0049	0.0059	0.0039	0.0019	0.0010
LBNDM	0.0350	0.0196	0.0111	0.0091	0.0090	0.0331	0.2007	0.1977	0.1600
MR_3	0.0269	0.0167	0.0126	0.0097	0.0080	0.0074	0.0072	0.0074	0.0073
MR_4	**0.0268**	**0.0117**	0.0068	**0.0047**	0.0038	0.0034	0.0029	0.0027	0.0026
MR_8	0.2044	0.0234	0.0096	0.0087	**0.0029**	**0.0013**	**0.0008**	**0.0005**	**0.0004**

Experimental Results on a Protein Sequence (with $\|\Sigma\| = 20$)									
m	8	16	32	64	128	256	512	1024	2048
FJS	0.0161	0.0100	0.0080	0.0063	0.0054	0.0052	0.0053	0.0051	0.0047
TVSBS	0.0124	0.0069	0.0040	0.0024	0.0017	0.0014	0.0014	0.0009	0.0008
SA	0.0413	0.0406	0.0407	-	-	-	-	-	-
BMH-SBNDM	0.0138	0.0083	0.0058	0.0045	0.0039	0.0033	0.0026	0.0022	0.0016
EBOM	0.0097	0.0050	0.0029	0.0020	0.0016	0.0011	0.0007	0.0006	0.0004
AOSO	0.0284	0.0089	0.0075	0.0094	0.0413	0.0315	0.0811	0.1112	0.1734
FSBNDM	0.0094	0.0049	0.0028	0.0021	0.0016	0.0013	0.0011	0.0009	0.0007
HASH3	0.0221	0.0098	0.0051	0.0031	0.0021	0.0019	0.0018	0.0016	0.0016
HASH5	0.0484	0.0159	0.0073	0.0038	0.0023	0.0018	0.0015	0.0013	0.0014
HASH8	4.7711	0.0320	0.0122	0.0057	0.0032	0.0024	0.0019	0.0017	0.0018
SBNDM2	0.0094	0.0047	**0.0026**	0.0019	0.0017	0.0015	0.0011	0.0009	0.0007
SBNDM4	0.0210	0.0081	0.0038	0.0019	0.0013	0.0011	0.0010	0.0009	0.0007
SBNDM6	0.0488	0.0135	0.0056	0.0027	0.0015	0.0009	0.0009	0.0009	0.0007
SBNDM8	4.7711	0.0210	0.0076	0.0035	0.0018	0.0010	0.0008	0.0008	0.0006
LBNDM	0.0130	0.0076	0.0039	0.0027	0.0020	0.0017	0.0016	0.0021	0.0120
MR_2	**0.0086**	**0.0045**	0.0028	0.0021	0.0018	0.0016	0.0017	0.0022	0.0046
MR_2-Pos-Breaking	0.0092	0.0049	0.0031	0.0022	0.0018	0.0016	0.0013	0.0012	0.0012
MR_3	0.0120	0.0053	0.0027	**0.0015**	**0.0010**	**0.0007**	**0.0005**	**0.0004**	**0.0003**

From the above tables, we know that experiment results are suitable for the theoretical analyses

in Section 3. It means that we can choose at least one appropriate constant c for the MR_c algorithm for the given patterns and texts. This leads to the flexibility of our approach.

5. CONCLUSIONS

By using the automaton M_x to reflect and update the degrees of appearance of x in y at all positions, and taking advantage of the relationship between the size of the pattern x and the alphabet Σ, our approach is flexible to design the MR_c algorithm for the exact pattern matching problem. The effectiveness of MR_c is expressed in two main features as follows. It never scans an arbitrary letter of the text repeatedly. In the cases of $|\Sigma| \geq 4, 16 \leq m \leq 2048$ and $|\Sigma| \geq 32, 8 \leq m \leq 2048$, we always choose a constant c with $1 \leq c \leq 8$ such that $T(n) < n$ and $p(z \in x) \leq 2^{-5}$ hold, where z is an arbitrary c_block over the alphabet Σ. In addition, experimental results comparing with 10 algorithms [12] also show the efficiency of our approach in practice.

The appearance of a part of the pattern is immediately reflected or updated at any position being scanned in the text, so our approach can be applied to secure data environment. This issue will be presented in the next works.

ACKNOWLEDGMENT

The author is extremely grateful to Late Assoc. Prof. Phan Trung Huy, Assoc. Prof. Phan Thi Ha Duong and Dr. Vu Thanh Nam for their valuable suggestions and encouragements.

The author also gives thanks to the reviewers for their worthy comments, which improve the quality of the paper.

REFERENCES

[1] C. Allauzen, M. Crochemore, M. Raffinot, "Factor oracle: A new structure for pattern matching," *SOFSEM 1999: Theory and Practice of Informatics*, Czech Republic, November 27 - December 4, 1999, pp. 295–310. Doi: https://doi.org/10.1007/3-540-47849-3_18

[2] R. Baeza-Yates, G. H. Gonnet, "A new approach to text searching," *Communications of the ACM*, vol. 35, no. 10, pp. 74–82, 1992.

[3] T. Berry, S. Ravindran, "Fast string matching algorithm and experimental results," *A Proceedings of the Prague Stringology Club Workshop '99*, Collaborative Report DC-99-05, Czech Technical University, Prague, pp. 16–26, 2001.

[4] J. Berstel, D. Perrin, *Theory of Codes.* Academic Press, pp. 5–6, 1985.

[5] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, J. I. Seiferas, "The smallest automaton recognizing the subwords of a text," *Theoretical Computer Science*, vol. 40, pp. 31–55, 1985.

[6] R. S. Boyer, J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[7] D. Cantone, S. Faro, "Fast-search algorithms: New efficient variants of the Boyer-Moore pattern-matching algorithm," *J. Autom. Lang. Comb.*, vol. 10, no. 5/6, pp. 589–608, 2005.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms.* (Second edition), MIT Press, Chapter 32, 2001.

[9] M. Crochemore, W. Rytter, *Text Algorithms,* Oxford University Press, pp. 79–142, 1994.

[10] B. Durian, J. Holub, H. Peltola, J. Tarhio, "Tuning BNDM with q-grams," *Proceeding Proceedings of the Meeting on Algorithm Engineering & Expermiments*, New York, January 03 - 03, 2009. Pages 29-37.

[11] S. Faro, T. Lecroq, "Efficient variants of the backward-oracle-matching algorithm," *International Journal of Foundations of Computer Science*, vol. 20, no. 6, pp. 967–984, 2009.

[12] S. Faro, T. Lecroq, "The exact online string matching problem: A review of the most recent results," *ACM Computing Surveys*, vol. 45, no. 2, Article 13, 2013.

[13] F. Franek, C. G. Jennings, W. F. Smyth, "A simple fast hybrid pattern-matching algorithm," *Journal of Discrete Algorithms*, vol. 5, no. 4, pp. 682–695, 2007.

[14] K. Fredriksson, S. Grabowski, "Practical and optimal string matching," *String Processing and Information Retrieval 12th International Conference, SPIRE 2005,* Buenos Aires, Argentina, November 2–4, 2005. Proceedings. Doi: https://doi.org/10.1007/11575832_42

[15] J. Holub, B. Durian, "Fast Variants of Bit Parallel Approach to Suffix Automata," *Stringology Research Workshop'2005*, 2005.

[16] R. N. Horspool, "Practical fast searching in strings," *Software Practice and Experience*, vol. 10, no. 6, pp. 501–506, 1980.

[17] N. T. T. Huyen, P. T. Huy, "Fuzzy approach in some pattern matching algorithms," *J. of Computer Science and Cybernetics*, vol. 18, no. 3, pp. 201–210, 2002.

[18] D. E. Knuth, J. H. Morris, Jr., V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, 1977.

[19] T. Lecroq, "Fast exact string matching algorithms," *Information Processing Letters*, vol. 102, no. 6, pp. 229–235, 2007.

[20] G. Navarro, M. Raffinot, "Fast and flexible string matching by combining bit-parallelism and suffix automata," *ACM Journal of Experimental Algorithmics (JEA)*, vol. 5, no. 4, Article 4, 2000.

[21] H. Peltola, J. Tarhio, "Alternative Algorithms for Bit-Parallel String Matching", *String Processing and Information Retrieval, 10th International Symposium, SPIRE 2003. Proceedings*, Manaus, Brazil, October 8–10, 2003, pp. 80–93.

[22] K. Ruohonen, "Formal Languages," *Tampere University of Technology*, pp. 1–3, 2009.

[23] S. S. Sheik, S. K. Aggarwal, A. Poddar, N. Balakrishnan, K. Sekar, "A fast pattern matching algorithm," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 4, pp. 1251–1256, 2004.

[24] D. M. Sunday, "A very fast substring search algorithm," *Communications of the ACM*, vol. 33, no. 8, pp. 132–142, 1990.

[25] R. Thathoo, A. Virmani, S. S. Lakshmi, N. Balakrishnan, K. Sekar, "TVSBS: A fast exact pattern matching algorithm for biological sequences," *Current Sci.*, vol, 91, no. 1, pp. 47–53, 2006.

[26] S. Wu, U. Manber, "A fast algorithm for multi-pattern searching," *Technical Report TR-94-17*, University of Arizona, Tucson, 1994.

Received on February 19, 2019
Revised on August 08, 2019