Journal of Software Engineering and Applications, 2016, 9, 399-411
Published Online August 2016 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2016.98026

A Fast Pattern Matching Algorithm Using Changing Consecutive Characters

Amjad Hudaib[1], Dima Suleiman[1], Arafat Awajan[2]

[1]Department of Computer Information Systems, King Abdullah II for Information Technology, The University of Jorda, Amman, Jordan
[2]Department of Computer Science, King Hussein Faculty of Computer Sciences, Princess Sumaya University for Technology, Amman, Jordan
Email: Ahudaib@ju.edu.jo, d.suleiman@psut.edu.jo, awajan@psut.edu.jo

Received 20 June 2016; accepted 5 August 2016; published 8 August 2016

Abstract

Pattern matching is a very important algorithm used in many applications such as search engine and DNA analysis. They are aiming to find a pattern in a text. This paper proposes a Pattern Matching Algorithm Using Changing Consecutive Characters (PMCCC) to make the searching process of the algorithm faster. PMCCC enhances the shift process that determines how the pattern moves in case of the occurrence of the mismatch between the pattern and the text. It enhances the Berry Ravindran (BR) shift function by using *m* consecutive characters where *m* is the pattern length. The formal basis and the algorithms are presented. The experimental results show that PMCCC made enhancements in searching process by reducing the number of comparisons and the number of attempts. Comparing the results of PMCCC with other related algorithms has shown significant enhancements in average number of comparisons and average number of attempts.

Keywords

Pattern, Pattern Matching Algorithms, String Matching, Berry Ravindran, EBR, RS-A, Fast Pattern Matching Algorithms

1. Introduction

Pattern matching is considered a very important algorithm in various applications such as search engine and DNA analysis [1]-[4].

The main purpose of pattern matching algorithms is to find a pattern (which is relatively small) in a text (which is very relatively large). They are aiming to enhance the search process and make it faster. This can be

How to cite this paper: Hudaib, A., Suleiman, D. and Awajan, A. (2016) A Fast Pattern Matching Algorithm Using Changing Consecutive Characters. *Journal of Software Engineering and Applications*, **9**, 399-411.
http://dx.doi.org/10.4236/jsea.2016.98026

done either by decreasing the number of comparisons, attempts or both. Some algorithms made enhancements on searching process and the way comparisons are made between the text and the pattern [1] [5]-[7], others made enhancements on the preprocessing phase [3] [8]-[15] which used to determine the amount of shift in case a mismatch happened between the pattern and the text.

In some algorithms, comparisons occur between the text and the pattern from only one side of the text either the left side [13] or the right one [4] by aligning one pattern with the text. Other algorithms use two sliding windows to make comparisons by using two patterns, one pattern aligned with the text from left side of the text, and the second pattern aligned with the text from the right side, comparisons occur in parallel [5] [6] [11] [16].

Another type of enhancements is made on the shift values. The shift value is the amount of shift that the pattern will move in case of a mismatch between the text and the pattern occurs. This enhancement is very important since it affects the efficiency of the searching process. Most of algorithms determine the shift value according to the number of consecutive characters on the text after aligning the pattern with the text. Some algorithms take one consecutive character [7] [17] some take two [6] [13] [16] others use three and few use four [8] [9].

This paper proposes a new Pattern Matching algorithm Using Changing Consecutive Characters (PMCCC) to make the searching process of the algorithm faster. PMCCC made enhancements on the shifting values, which maximize the amount of shift that the pattern will move once a mismatch between the text and the pattern occurs. The new algorithm uses only one sliding window so the pattern will be aligned with text from the left side. The formal basis of the algorithm, algorithm steps and analysis is presented. Comparisons are made with existing algorithms such as BR [12], EBR [11] and RS-A [4]. The results of the comparisons show that the new algorithm is better than the other algorithms in terms of comparisons and number of attempts.

The reminder of this paper is organized as follows: Section 2 presents the related work. Section 3 describes PMCCC algorithm and analyzes its performance. Section 4 presents a working example to show how the algorithm works. Section 5 shows the experiment and the comparisons with other algorithms. Finally, Section 6 covers the conclusions and future work.

2. Related Works

Pattern matching is one of the main topics in research since it is very important and can be used in different applications [1] [18]-[22].

Enhancements have been made to make the searching process faster either by using different technique for the search process itself or increasing the shift value that the pattern must move in case there is a mismatch between the pattern and the text [10] [15]. Other enhancements were performed to reduce the memory used to deal with the shift values in preprocessing phase.

Boyer Moore [17] made enhancement in a searching process, if a mismatch occurs between the text and the pattern after aligning the pattern with the text the pattern will be shifted, the amount of shift is determined by aligning the pattern with the text and taking one character in the text immediately to the right of the pattern if the pattern already aligned with left side of the text.

Berry-Ravindran (BR) [12] made enhancement on Boyer Moore such that instead of taking one character to determine the shift value, BR takes two consecutive characters which makes the value of the shift greater and then the searching process faster.

Shift technique used in Berry-Ravindran algorithm (BR) [13] form basis for many algorithms to determine the amount of shift in case a mismatch occurs between the text and the pattern. Some algorithms used the same shift technique used in BR by using two consecutive characters but made enhancement in a searching process such as Two Sliding Windows algorithm [6] and Enhanced Two Sliding Windows algorithm [16]. Others made enhancement on number of characters used for shift value, some used three characters as in Enhanced Berry Ravindran [11] and others used four characters such as RS-A [4] and ERS-A [5].

Two Sliding Windows algorithm (TSW) [6] used two sliding windows pattern to scan the text from the left and right side at the same time. TSW uses two windows the left window aligned with the text from the left and the right window aligned with the text from the right, the comparisons between the text and the pattern of both sides occurs at the same time. In case a mismatch occurs, the two windows will be shifted, the right window will be shifted to left and the left window will be shifted to right. The amount of shift in the two cases was determined according to BR [13] shift value by taking two consecutive characters.

Enhanced Two Sliding Window algorithm (ETSW) [16] made enhancements on TSW. ETSW used the same

searching technique used in TSW; it also used two windows to scan the text from both sides at the same time, in addition it used the same shift function used in BR and TSW by using two consecutive characters. The main difference between ETSW and TSW is that ETSW made comparisons between the text and the pattern from both sides of the pattern in parallel in this case number of comparisons between the text and the pattern will be minimized. The complexity of best case, worst case and preprocessing time are $O(m/2)$, $O(((n/2 - m/2 + 1))(m/2$, $O(2(m - 1))$ respectively.

To determine the amount of shift instead of using two consecutive characters, in case a mismatch occurs between the text and the pattern as in TSW, Enhanced Berry Ravindran (EBR) [11] used three characters. EBR also used two sliding windows to scan the text from both sides simultaneously.

ERS-A [5] algorithm made enhancements on RS-A [4]. RS-A algorithms used four consecutive characters immediately after the aligning the text with the pattern to determine the amount of shift that the pattern will move in case a mismatch occurs between the text and the pattern. RS-A searches the text from the right side only. On the other hand ERS-A uses four consecutive character and scans the text from the left and right side at the same time. As in TSW, ERS-A used two sliding windows aligned with the text, the left window aligned with the text from the left and the right window aligned with the text from right, in case a mismatch occurs the two windows will be shifted to the opposite side.

Enhancing ERS-A algorithm for pattern matching EERS-A [9] used the same technique used in ETSW [16]. Both of them made comparisons between the text and the pattern from both sides of the pattern, they also used two sliding windows to scan the text. But while ETSW using BR shift function, EERS-A uses RS-A shift to determine the amount of shift which depends on using four consecutive characters.

Four sliding windows pattern matching algorithm (FSW) [8] used four consecutive characters to determine the shift value as in RS-A, ERS-A and EERS-A. FSW used four sliding windows. The text was partitioned into two parts, each part scanned by using two windows, one aligned with the left side of the part and the other with the right side of the part.

A Performance Study of the Running Times of well-known Pattern Matching Algorithms for Signature-based Intrusion Detection Systems [20] was done and comparisons between different algorithms using multiple sliding windows approach were made.

PMCCC uses only one pattern window to search for a pattern p in text t from the left side of the text. In case a mismatch occurs between the pattern and the text the pattern will be shifted to right according to the shift value. The shift value is determined according to the pattern length m. Instead of taking two consecutive characters to determine the amount of shift as in TSW, three characters as in EBR or four characters in EERS-A, the new algorithm takes m (pattern length) consecutive characters immediately of the text after aligning the pattern with the left side of the text. Additional two algorithms shift 5 and shift 6 are developed to make comparisons. These two algorithms use one window to scan the text from the left side. They use five and six consecutive characters respectively to determine the amount of shift.

Comparing the results of the proposed algorithm with those of Br, EBR and RS-A has shown significant enhancements in average number of attempts and comparisons. The new algorithm makes the searching process faster and more efficient.

3. Description of the (PMCCC) Algorithm

PMCCC enhanced the searching process by increasing the amount of shift that occurs every time there is a mismatch between the text and the pattern. PMCCC depends on using m consecutive characters of the text immediately after aligning the text with the pattern, where m is the same as the pattern length. PMCCC algorithm determines the number of the consecutive characters according to the length of pattern.

The main difference between BR [13], EBR [11], RS-A [4] and PMCCC is that each one of the previous algorithms used a fixed length number of consecutive characters after the text to determine the amount of shift, BR uses two consecutive characters, EBR uses three and RS-A uses four.

3.1. Pre-Processing Phase

PMCCC calculates the value of the shift using m consecutive text characters $x_1, x_2, x_3, \cdots, x_m$, where m is the pattern length which are aligned immediately to the right of text.

Initially, m consecutive characters in the text have the indexes $(m+1),(m+2),(m+3),\cdots,(2m)$ for

$x_1, x_2, x_3, \cdots, x_m$. After a mismatch occurs between the text and the pattern, the pattern will be shifted to the right according to the *shift value*, which will be calculated by PMCCC algorithm (see **Figure 1**).

After shifting *m* consecutive characters immediately to right of the text will be (*leftindex* + 1), (*leftindex* + 2), (*leftindex* + 3), …., (*leftindex* + *m*) for $x_1, x_2, x_3, \cdots, x_m$. **Figure 1** shows the code of PMCCC algorithm where the shifting process is presented.

The main idea of the proposed algorithm is to use shift function used in BR [13], Berry Ravindran used two consecutive characters in the text that follow the pattern window to determine the shift value. It depends on using equation (1) to determine the amount of shift, x_1 and x_2 are the two consecutive characters, *p* is the pattern, and *m* the pattern length.

$$\text{shift value}\left[x_1, x_2\right] = \min \begin{cases} 1 & \text{if } p\left[m-1\right] = x_1 \\ m-i & \text{if } p\left[i\right] p\left[i+1\right] = x_1 x_2 \\ m+1 & \text{if } p\left[0\right] = x_2 \\ m+2 & \text{otherwise} \end{cases} \tag{1}$$

Other algorithms also used BR shift function such as EBR [11] and RS-A [4], but instead of using two characters they used three and four characters respectively. The shifting techniques used in EBR and RS-A depended on equation (2) and equation (3) respectively.

$$\text{shift value}\left[x_1, x_2, x_3\right] = \min \begin{cases} 1 & \text{if } p\left[m-1\right] = x_1 \\ 2 & \text{if } p\left[m-2\right]\left[m-1\right] = x_1 x_2 \\ m-i & \text{if } p\left[i\right]\left[i+1\right]\left[i+2\right] = x_1 x_2 x_3 \\ m+1 & \text{if } p\left[0\right] = x_2 \\ m+2 & \text{if } p\left[0\right] = x_3 \\ m+3 & \text{otherwise} \end{cases} \tag{2}$$

Shifting Process of PMCCC Algorithm

```
shift_value=m*2              /// m is the pattern length
                             /// shift_value contains the ///amount of shift p pattern text

 A:  {   for (int i=1;i<m;i++)
           { boolean found = true; int k=1;
             for (int j=i;j>=1;j--)
             { found = found &&
               (p.charAt(m- 1*k)==t.charAt(leftindex+j));
                 k++;}
                 if (found)
                 {  shift_value =i; break A; }
          }
       }
for (int i=0;i<m;i++)
    {      value1 +=t.charAt(leftindex+i+1);
           value2 +=p.charAt(i);
    }
if (value1.equals(value2))
   if(Left_shift_value > m)
        Left_shift_value = m
for (int i=2;i<=m;i++)
      { if (p.charAt(0)==t.charAt(leftindex+i))
          if(shift_value >(m+i-1))
                    shift_value =m+i-1;     }
```

Figure 1. PMCCC algorithm-shifting process.

$$\text{shift value}\left[x_1,x_2,x_3,x_4\right]=\min\begin{cases}1 & \text{if } p\left[m-1\right]=x_1\\ 2 & \text{if } p\left[m-2\right]\left[m-1\right]=x_1x_2\\ 3 & \text{if } p\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3\\ m-i & \text{if } p\left[i\right]\left[i+1\right]\left[i+2\right]\left[i+3\right]=x_1x_2x_3x_4\\ m+1 & \text{if } p\left[0\right]=x_2\\ m+2 & \text{if } p\left[0\right]=x_3\\ m+3 & \text{if } p\left[0\right]=x_4\\ m+4 & \text{otherwise}\end{cases} \tag{3}$$

In order to make a good comparisons, we implemented two new algorithms and called them shift 5 and shift 6. Shift 5 and shift 6 also used Berry Ravindran shift but they used 5 and 6 characters respectively to determine the amount of shift. The shifting process of algorithm shift 5 depends on equation (4) and the shifting process of shift 6 depends on equation (5).

$$\text{Left shift value}\left[x_1,x_2,x_3,x_4,x_5\right]=\min\begin{cases}1 & \text{if } p\left[m-1\right]=x_1\\ 2 & \text{if } p\left[m-2\right]\left[m-1\right]=x_1x_2\\ 3 & \text{if } p\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3\\ 4 & \text{if } p\left[m-4\right]\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3x_4\\ m-i & \text{if } p\left[i\right]\left[i+1\right]\left[i+2\right]\left[i+3\right]p\left[i+3\right]=x_1x_2x_3x_4x_5\\ m+1 & \text{if } p\left[0\right]=x_2\\ m+2 & \text{if } p\left[0\right]=x_3\\ m+3 & \text{if } p\left[0\right]=x_4\\ m+4 & \text{if } p\left[0\right]=x_5\\ m+5 & \text{otherwise}\end{cases} \tag{4}$$

$$\text{shift value}\left[x_1,x_2,x_3,x_4,x_5,x_6\right]=\min\begin{cases}1 & \text{if } p\left[m-1\right]=x_1\\ 2 & \text{if } p\left[m-2\right]\left[m-1\right]=x_1x_2\\ 3 & \text{if } p\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3\\ 4 & \text{if } p\left[m-4\right]\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3x_4\\ 5 & \text{if } p\left[m-5\right]\left[m-4\right]\left[m-3\right]\left[m-2\right]\left[m-1\right]=x_1x_2x_3x_4x_5x_6\\ m-i & \text{if } p\left[i\right]\left[i+1\right]\left[i+2\right]\left[i+3\right]p\left[i+3\right]p\left[i+4\right]=x_1x_2x_3x_4x_5x_6\\ m+1 & \text{if } p\left[0\right]=x_2\\ m+2 & \text{if } p\left[0\right]=x_3\\ m+3 & \text{if } p\left[0\right]=x_4\\ m+4 & \text{if } p\left[0\right]=x_5\\ m+5 & \text{if } p\left[0\right]=x_6\\ m+6 & \text{otherwise}\end{cases} \tag{5}$$

PMCCC algorithm used m characters, where m is the pattern length, to determine the amount of shift and depends on equation (6).

Figure 2 presents how PMCCC algorithm calculates the shift value based on equation (6).

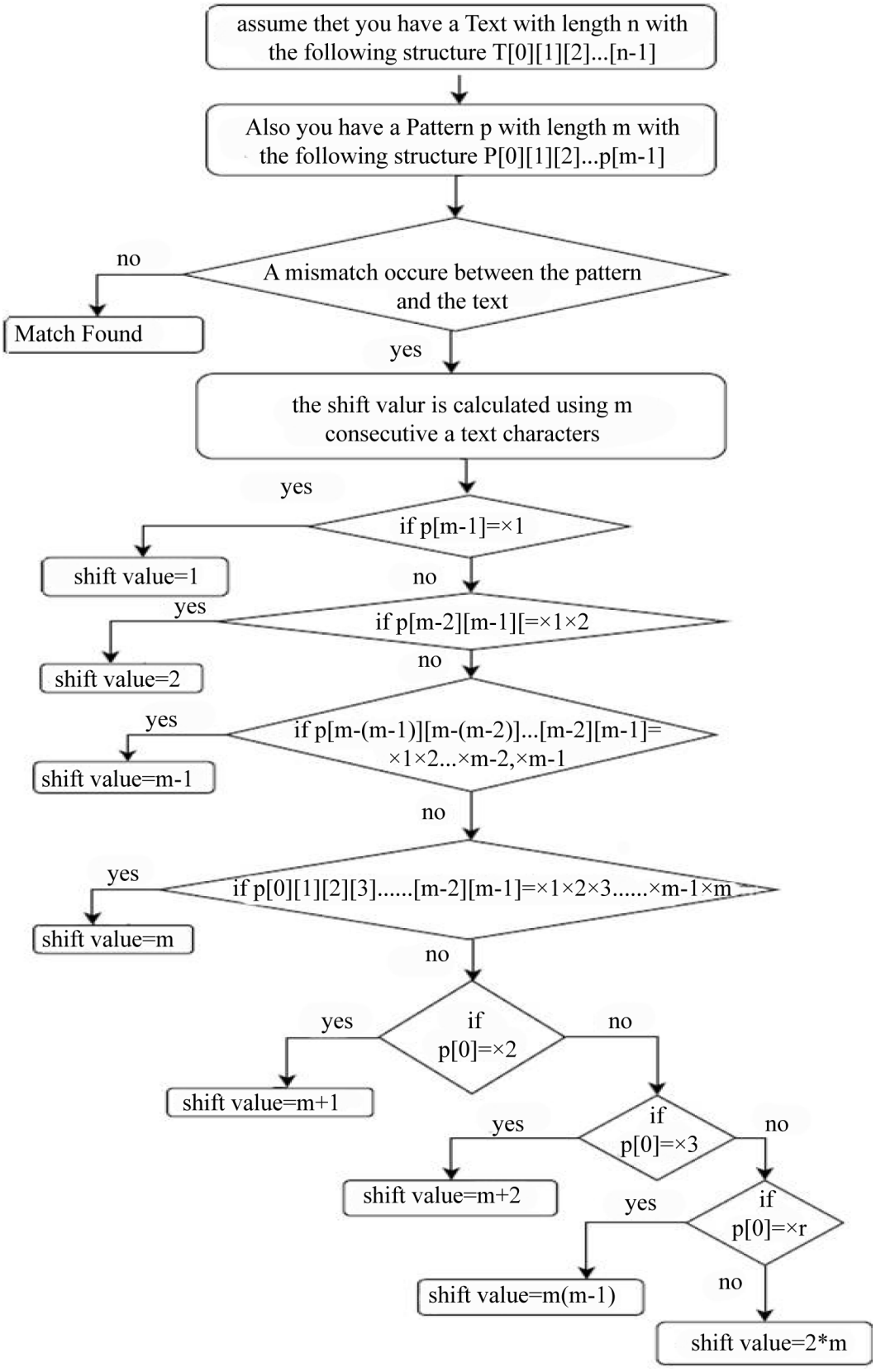

Figure 2. The shifting process of PMCCC algorithm.

$$\text{shift value}\left[x_1, x_2, x_3, \cdots, x_m\right] = \min \begin{cases} 1 & \text{if } p\left[m-1\right] = x_1 \\ 2 & \text{if } p\left[m-2\right]\left[m-1\right] = x_1 x_2 \\ 3 & \text{if } p\left[m-3\right]\left[m-2\right]\left[m-1\right] = x_1 x_2 x_3 \\ \quad \vdots & \\ m-1 & \text{if } p\left[m-(m-1)\right]\left[m-(m-2)\right]\cdots\left[m-2\right]\left[m-1\right] = x_1 x_2 \cdots x_{m-2} x_{m-1} \\ m & \text{if } p\left[0\right]\left[1\right]\left[2\right]\left[3\right]\cdots\left[m-2\right]\left[m-1\right] = x_1 x_2 x_3 \cdots x_{m-1} x_m \\ m+1 & \text{if } p\left[0\right] = x_2 \\ m+2 & \text{if } p\left[0\right] = x_3 \\ m+3 & \text{if } p\left[0\right] = x_4 \\ \quad \vdots & \\ m+(m-2) & \text{if } p\left[0\right] = x_{m-1} \\ m+(m-1) & \text{if } p\left[0\right] = x_m \\ 2m & \text{otherwise} \end{cases} \tag{6}$$

3.2. Searching Phase

PMCCC algorithm starts the searching from the left side of the pattern using one window, in case of a mismatch between the text and the window; the window will be shifted to the right according to the shift value in equation (6).

One pointer (*leftindex*) in the text and one pointer in the pattern (L) will be used to make comparison between the text and the pattern. The first character of the pattern and the corresponding text character (*leftindex*) will be compared. If a match occurs the *leftindex* and L will be incremented by one and if a mismatch occurs the *leftindex* will be incremented by the shift value and L index value will be zero.

One pointer (*leftindex*) in the text and one pointer in the pattern (L) will be used to make comparison between the text and the pattern. The first character of the pattern and the corresponding text character (*leftindex*) will be compared. If a match occurs the *leftindex* and L will be incremented by one and if a mismatch occurs the *leftindex* will be incremented by the shift value and L index value will be zero.

3.3. Analysis

Lemma 1: The time complexity in worst case is $O(((n - m + 1))(m))$.

Proof: The worst case occurs when a match between the pattern and text occurs in all characters (*i.e.* The shift value is equal to one), until we reach the last character of the pattern and a mismatch between the pattern and text occurs. This case is reputed until we reach the index $(n - (m + 1))$ of the text.

Lemma 2: The time complexity in best case is $O(m)$.

Proof: The best case occurs when a match occurs between the pattern and the text in the leftmost of the text.

Lemma 3: The time complexity in average case is $O(n/(2*m))$.

Proof: The average case occurs when the m consecutive characters immediately to the right of the text after aligning it with the pattern are not found in the pattern. In this case, time complexity is $O(n/(2*m))$ and the shift value will be $(2*m)$.

4. Working Example

In this section, we will give an example to explain the new algorithm.

Given a text T with $n = 50$.

T = "ABECABACBAFECABAEEBEBEABACBEECABACCCBAEEBABEBEBABA", with index from 0 to 49.

And a pattern P with $m = 9$:
P = "ABACCCBAE", with index from 0 to 9.

Pre-processing phase

Initially, shift value = $2*m = 18$.

Searching phase

The searching process for the pattern p is explained through the following steps:

First attempt:

In the first attempt (**Figure 3(a)**), we align the pattern window with the text from the left, a comparison is made between and first character in the pattern (P [0] = A) and the first character of the text (T [0] = A). Since the two characters are the same, another comparison is made between the second character in the pattern (P [1] = B) and the second character of the text (T [1] = B), and again a match occurs. Comparisons continue until either a complete match occurs or a mismatch occurs. In this example a mismatch occurs between the third character of the pattern (P [2] = A), and of the text (T [2] = E), therefore we take 9 consecutive characters of the text at indexes: 9, 10, 11, 12, 13, 14, 15, 16, 17 which are A, F, E, C, A, B, A, E, E respectively.

To determine the shift value we have to make the following comparison according to equation (6) as in **Table 1**. Then the pattern will be shifted to right 13 steps.

Second attempt:

In the second attempt, after shifting the pattern 13 steps to the right (**Figure 3(b)**), the first character of the pattern is found and is aligned with the text character at index 13. After making comparison between the text and the pattern a mismatch occurs between the text character at index 16 (character E) and the fourth index in the pattern (character C). To determine the shift value we take 9 consecutive characters at indexes: 22, 23, 24, 25, 26, 27, 28, 29, 30 which are A, B, A, C, B, E, E, C, A respectively. So that the pattern must be shifted using equation 6 as shown in **Table 2**. Again to determine amount of shift we apply equation (6) and shown in **Table 2**. Then the pattern will be shifted to right 11 steps.

Third attempt:

In the third attempt, a mismatch occurs between text character at index 25 (character C) and the second pattern text (character B), (see **Figure 3(c)**). The nine consecutive characters at indexes: 33, 34, 35, 36, 37, 38, 39, 40, 41, which are C, C, C, B, A, E, E, B, A, respectively.

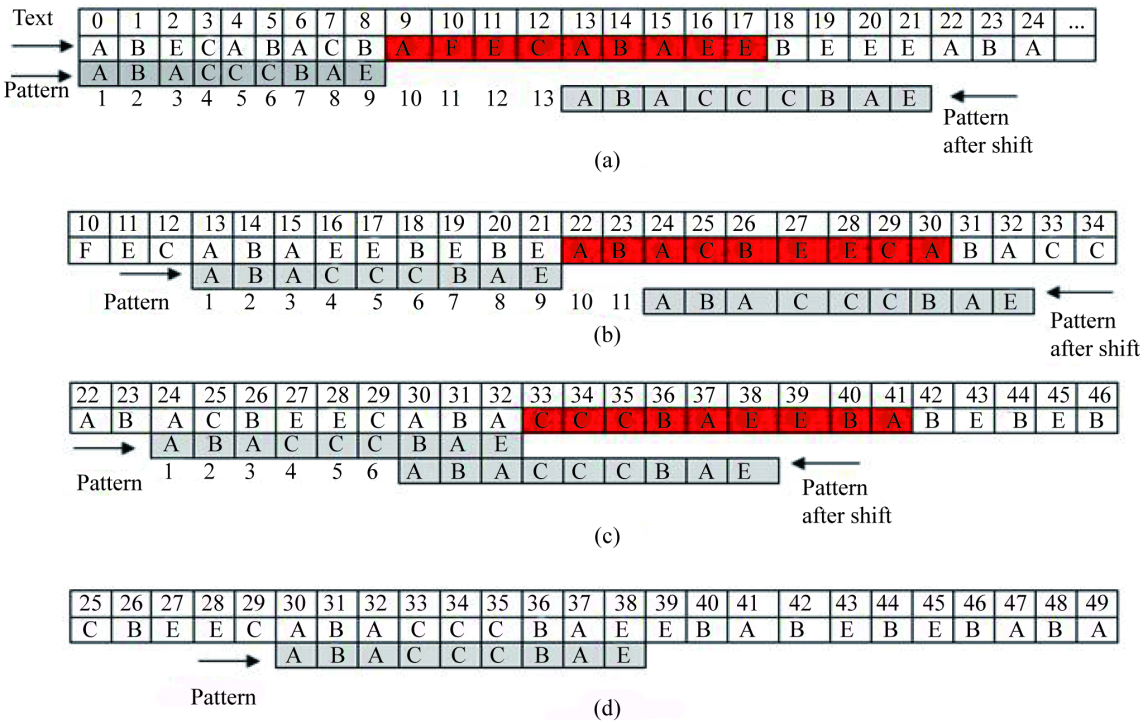

Figure 3. Working example.

Table 1. Determine the shift value for the first attempt.

Shift value	Comparisons
1	if p[8] = A?, Yes stop. No, continue
2	if p[7] [8] = AF?, Yes stop. No, continue
3	if p[6] [7] [8] = AFE?, Yes stop. No, continue
4	if p[5] [6] [7][8] = AFEC?, Yes stop. No, continue
5	if p [4] [5] [6] [7] [8] = AFECA?, Yes stop. No, continue
6	if p[3] [4] [5] [6] [7] [8] = AFECAB?, Yes stop. No, continue
7	if p[2] [3] [4] [5] [6] [7] [8] = AFECABA?, Yes stop. No, continue
8	if p[1] [2] [3] [4] [5] [6] [7] [8] = AFECABAE?, Yes stop. No, continue
9	if p[0] [1] [2] [3] [4] [5] [6] [7] [8] = AFECABAEE?, Yes stop. No, continue
10	if p[0] = F?, Yes stop. No, continue
11	if p[0] = E?, Yes stop. No, continue
12	if p[0] = C, Yes stop. No, continue
13	if p[0] = A?, **Yes stop the shift value will be 13**

Table 2. Determine the shift value for the second attempt.

Shift value	Comparisons
1	if p[8] = A?, Yes stop. No, continue
2	if p[7] [8] = AB?, Yes stop. No, continue
3	if p[6] [7] [8] = ABA?, Yes stop. No, continue
4	if p[5] [6] [7] [8] = ABAC?, Yes stop. No, continue
5	if p[4] [5] [6] [7] [8] = ABACB?, Yes stop. No, continue
6	if p[3] [4] [5] [6] [7] [8] = ABACBE?, Yes stop. No, continue
7	if p[2] [3] [4] [5] [6] [7] [8] = ABACBEE?, Yes stop. No, continue
8	if p[1] [2] [3] [4] [5] [6] [7] [8] = ABACBEEC?, Yes stop. No, continue
9	if p[0] [1] [2] [3] [4] [5] [6] [7] [8] = ABACBEEECA?, Yes stop. No, continue
10	if p[0] = B?, Yes stop. No, continue
11	if p[0] = A**?, Yes stop the shift value will be 11**

Now, to determine amount of shift we apply equation (6) and shown in **Table 3**, so that, the pattern will be shifted to the right 6 steps.

Fourth attempt:

We align the first character of the pattern (character A) with text character at index 30 (character A). A complete match between the text and pattern occurs at index 30, (see **Figure 3(d)**).

5. Experimental Results and Discussion

Multiple experiments have been done to make comparisons between PMCCC and other algorithms. In addition to implementing the PMCCC, we implemented two other algorithms shift 5 and shift 6 to make good comparisons. Shift 5 and shift 6 algorithms used the same shift functions used in BR [13], but instead of taking two consecutive characters to determine the shift value, shift 5 and shift 6 algorithms used 5 and 6 consecutive characters respectively. Also we used the well-known algorithms to make comparisons, which are BR [13], EBR [11], RS-A [4] which uses 2, 3, 4 consecutive characters respectively to determine the shift value. But since EBR used two sliding windows we implement it using only the left side equations to determine the amount of shift.

Table 3. Determine the shift value for the third attempt.

Shift value	Comparisons
1	if p[8] = C?, Yes stop. No, continue
2	if p[7] [8] = CC?, Yes stop. No, continue
3	if p[6] [7] [8] = CCC?, Yes stop. No, continue
4	if p[5] [6] [7] [8] = CCCB?, Yes stop. No, continue
5	if p[4] [5] [6] [7] [8] = CCCBA?, Yes stop. No, continue
6	if p[3] [4] [5] [6] [7] [8] = CCCBAE?, Yes stop the shift value will be 6

Table 4. The average number of comparisons for patterns with different lengths.

Pattern length	Number of words	BR	EBR	RS-A	Shift 5	Shift 6	PMCCC
7	1988	13345	11749	10638	9783	9100	8056
8	1167	14807	13092	11922	11033	10320	9217
9	681	15892	14095	12911	12004	11273	10141
10	382	15799	14070	12927	12064	11362	10289
11	191	14243	12675	11672	10910	10298	9367
12	69	10923	9774	9030	8508	8074	7401
13	55	11370	10191	9422	8895	8466	7786
14	139	21673	19255	17845	16843	16008	14734
15	32	22384	19747	18318	17261	16435	15107
16	10	28644	25452	23531	22163	21080	19381
17	3	28148	25169	23119	21922	20895	19252

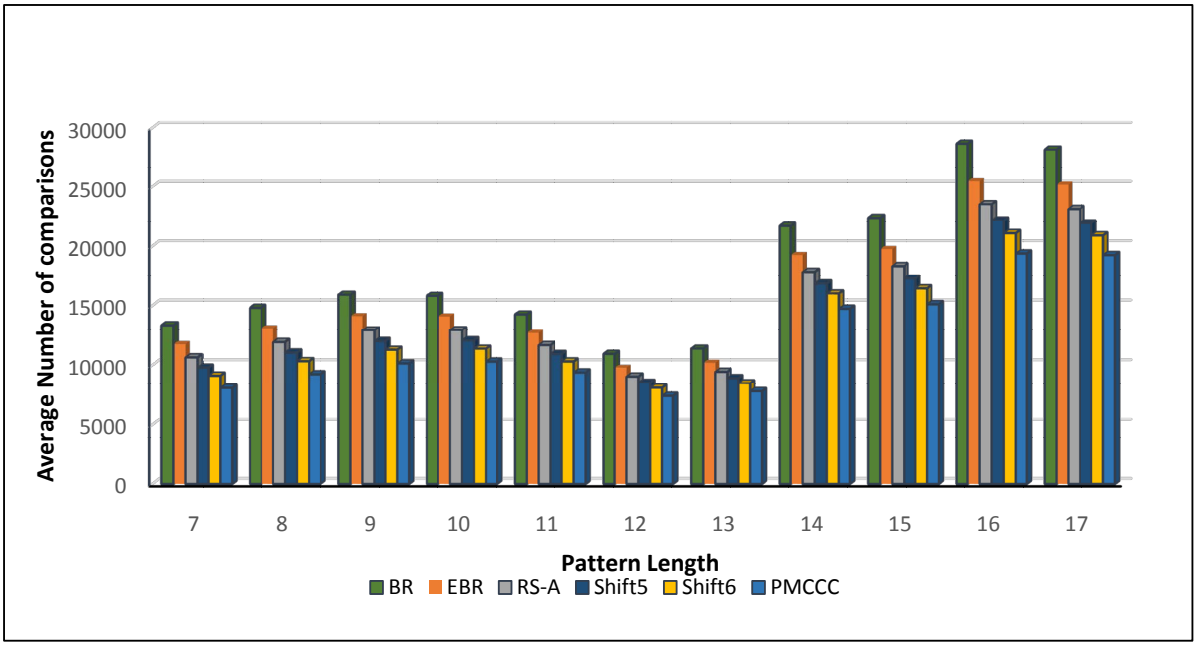

Figure 4. The average number of comparisons of BR, EBR, RS-A, Shift 5, Shift 6 and PMCCC Algorithms.

Table 5. The average number of attempts for patterns with different lengths.

Pattern length	Number of words	BR	EBR	RS-A	Shift 5	Shift 6	PMCCC
7	1988	11953	10505	9512	8749	8139	7203
8	1167	13256	11704	10660	9866	9226	8241
9	681	14149	12532	11477	10673	10024	9019
10	382	14127	12567	11543	10774	10148	9188
11	191	12808	11378	10480	9798	9252	8410
12	69	9598	8584	7927	7472	7091	6507
13	55	10334	9250	8560	8069	7677	7065
14	139	19548	17356	16086	15172	14423	13273
15	32	19817	17454	16176	15249	14517	13362
16	10	26086	23176	21411	20176	19194	17662
17	3	22554	20138	18551	17570	16747	15458

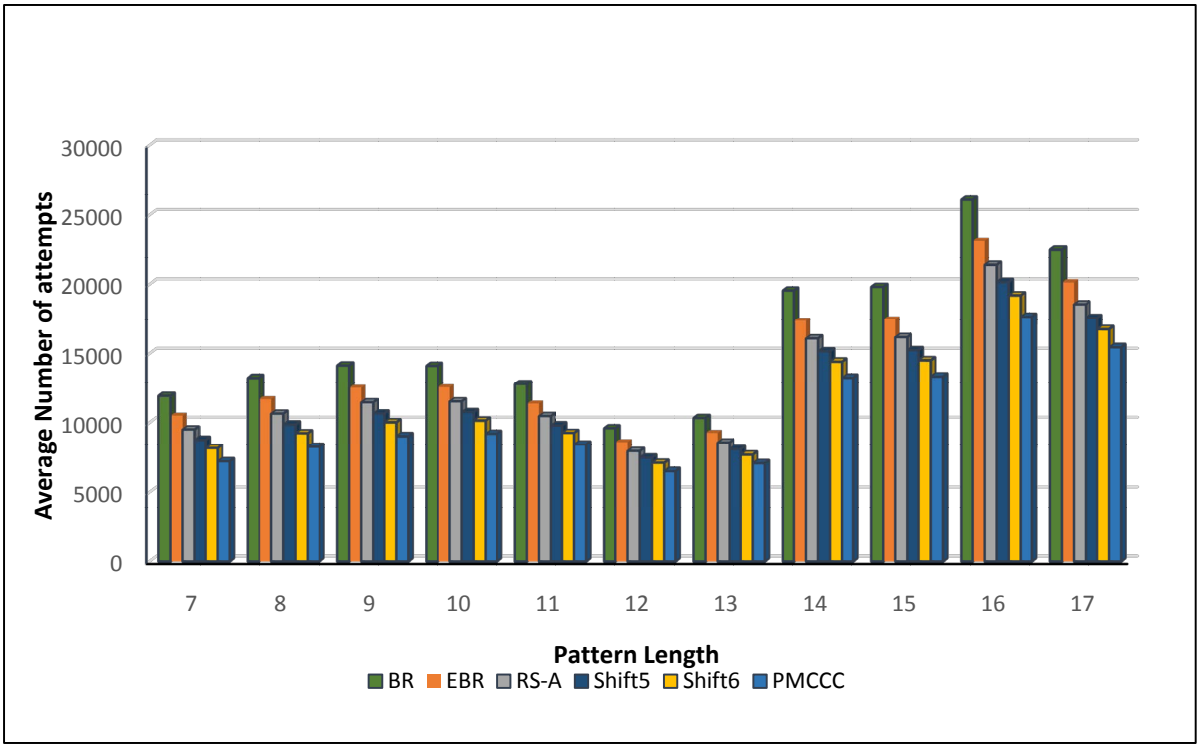

Figure 5. The average number of attempts of BR, EBR, RS-A, Shift 5, Shift 6 and PMCCC algorithms.

- Dataset:

All the six algorithms used Book1 from the Calgary corpus to be the text [23]. Book1 consists of 141,274 words (752,149 characters). Patterns of different lengths are also taken randomly from Book1.

Table 4 shows the results of comparing the algorithms BR, EBR, RS-A, shift 5, shift 6 and PMCCC based on the average number of comparisons using different patterns with different lengths. The pattern length is in the first column and it is from 7 characters to 17 characters. The number of patterns with a certain length is presented in the second column. The results are represented in **Figure 4**. **Table 4** and **Figure 4** show that PMCCC results are better than all other algorithms results in term of average number of comparisons. For example, **Table 4** shows that for 191 words of length 11, the average number of comparisons is 9367, which is better than all other algorithms. Average number of comparisons for the same pattern length are 14243, 12675, 11672, 10910

and 10298 in BR, EBR, RS-A, shift 5, shift 6 algorithms respectively. The main reason for that is PMCCC always use m consecutive characters to determine the shift value. That means the PMCCC algorithm is faster than the other algorithm in finding pattern.

Table 5 shows the comparison between PMCCC algorithm and the other algorithms in terms of the average number of attempts using different patterns with different lengths. The results show that PMCCC algorithm needs less number of attempts than the other algorithm. For example, **Table 5** shows for 191 words of length 11, the average number of attempts of PMCCC algorithm is 8410, which is better than the results of all other algorithms. The average number of attempts for the same pattern length are 12808, 11378, 10480, 9798, and 9252 in BR, EBR, RS-A, shift 5, shift 6 algorithms respectively. The main reason for that is PMCCC always use m consecutive characters to determine the shift value. That means the PMCCC algorithm is faster than the other algorithm in finding pattern. **Figure 5** represents the results.

6. Conclusion and Future Work

This paper proposed a new pattern matching algorithm based on Changing Consecutive Characters (PMCCC) to make the searching process of the algorithm faster. PMCCC algorithm uses *m* consecutive characters where *m* is the pattern length to determine the shift value in case a mismatch occurs between the text and pattern. This process made the PMCCC faster than many other algorithms that used shift function. Comparisons made between PMCCC and already existing algorithms BR, EBR, RS-A and also comparisons made with new algorithms implemented for the purpose of comparison, are called Shift 5 and Shift 6, respectively. The experimental results show that PMCCC is the faster than other algorithms in terms of number of comparison and attempts.

References

[1] Chao, Y. (2012) An Improved BM Pattern Matching Algorithm in Intrusion Detection System. *Applied Mechanics and Materials*, **148-149**, 1145-1148.

[2] Diwate, R. and Alaspurkar, S. (2013) Study of Different Algorithms for Pattern Matching. *International Journal of Advanced Research in Computer Science and Software Engineering*, **3**, 615-620.

[3] Bhukya, R. and Somayajulu, D. (2010) An Index Based Forward Backward Multiple Pattern Matching Algorithm. *World Academy of Science, Engineering and Technology*, **4**, 1513-1521.

[4] Senapati, K.K., Mal, S. and Sahoo, G. (2012) RS-A Fast Pattern Matching Algorithm for Biological Sequences. *International Journal of Engineering and Innovative Technology* (*IJEIT*), **1**, 116-118.

[5] Suleiman, D., Hudaib, A., Al-Anani, A., Al-Khalid, R. and Itriq, M. (2013) ERS-A Algorithm for Pattern Matching. *Middle East Journal of Scientific Research*, **15**, 1067-1075.

[6] Hudaib, A., Al-Khalid, R., Suleiman, D., Itriq, M. and Al-Anani, A. (2008) A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW). *Journal of Computer Science*, **4**, 393-401. http://dx.doi.org/10.3844/jcssp.2008.393.401

[7] Pendlimarri, D. and Petlu, P.B.B. (2010) Novel Pattern Matching Algorithm for Single Pattern Matching. *International Journal on Computer Science and Engineering*, **2**, 2698-2704.

[8] Hudaib, A., Al-Kalid, R., Al-Anani, A., Itriq, M. and Suleiman, D. (2015) Four Sliding Windows Pattern Matching Algorithm (FSW). *Journal of Software Engineering and Applications*, **8**, 154-165. http://dx.doi.org/10.4236/jsea.2015.83016

[9] Suleiman, D., Itriq, M., Al-Anani, A., Al-Khalid, R. and Hudaib, A. (2015) Enhancing ERS-A Algorithm for Pattern Matching (EERS-A). *Journal of Software Engineering and Applications*, **8**, 143-153. http://dx.doi.org/10.4236/jsea.2015.83015

[10] Salmela, L., Tarhio, J. and Kalsi, P. (2010) Approximate Boyer-Moore String Matching for Small Alphabets. *Algorithmica*, **58**, 591-609. http://dx.doi.org/10.1007/s00453-009-9286-3

[11] Suleiman, D. (2014) Enhanced Berry Ravindran Pattern Matching Algorithm (EBR). *Life Science Journal*, **11**, 395-402.

[12] Al-Mazroi, A. and Rashid, N. (2011) A Fast Hybrid Algorithm for the Exact String Matching Problem. *American Journal of Engineering and Applied Sciences*, **4**, 102-107. http://dx.doi.org/10.3844/ajeassp.2011.102.107

[13] Berry, T. and Ravindran, S. (2001) A Fast String Matching Algorithm and Experimental Results. *Proceedings of the Prague Stringology Club Workshop* '99, Collaborative Report DC-99-05, Czech Technical University, Prague, 16-26.

[14] Faro, S. (2009) Efficient Variants of the Backward-Oracle-Matching Algorithm. *International Journal of Foundations of Computer Science*, **20**, 967-984. http://dx.doi.org/10.1142/S0129054109006991

[15] Faro, S. and Külekci, M.O. (2012) Fast Packed String Matching for Short Patterns. arXiv:1209.6449v1 [cs.IR]

[16] Itriq, M., Hudaib, A., Al-Anani, A., Al-Khalid, R. and Suleiman, D. (2012) Enhanced Two Sliding Windows Algorithm for Pattern Matching (ETSW). *Journal of American Science*, **8**, 607-616.

[17] Boyer, R.S. and Moore, J.S. (1977) A Fast String Searching Algorithm. *Communications of the Association for Computing Machinery*, **20**, 762-772. http://dx.doi.org/10.1145/359842.359859

[18] Hlayel, A.A. and Hnaif, A.A. (2014) A New Exact Pattern Matching Algorithm (WEMA). *Journal of Applied Science*, **14**, 193-196. http://dx.doi.org/10.3923/jas.2014.193.196

[19] Hussain, I., Kausar, S., Hussain, L. and Khan, M. (2013) Improved Approach for Exact Pattern Matching (Bidirectional Exact Pattern Matching). *IJCSI International Journal of Computer Science Issues*, **10**, 59-65.

[20] Naik, S.M. and Geethanjali, N. (2015) Performance Study of the Running Times of Well Known Pattern Matching Algorithms for Signature-Based Intrusion Detection Systems. *International Journal on Recent and Innovation Trends in Computing and Communication*, **3**, 4177-4180.

[21] Vangipuram, R.K., Sandeep, S.J. and Reddy, A. (2011) Text Segmentation Based Pattern Search Algorithm. *International Journal of Wisdom Based Computing*, **1**.

[22] Hussain, I., Kazmi, S., Khan, I. and Mehmood, R. (2013) Improved-Bidirectional Exact Pattern Matching. *International Journal of Scientific & Engineering Research*, **4**, 659-663.

[23] Calgary Corpus. ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus