Enhanced Berry Ravindran Pattern Matching Algorithm (EBR)

Dima Suleiman

Department of Business Information Technology, King Abdullah II School for Information Technology,
The University of Jordan, Amman 11942 Jordan
dima.suleiman@ju.edu.jo

Abstract: In this paper, a new pattern matching algorithm is proposed. The new algorithm Enhanced Berry Ravindran (EBR) algorithm made enhancements on Berry-Ravindran (BR) algorithm. In BR, the shift value is computed by taking two consecutive characters of the text immediately following the pattern window. The new algorithm maximizes the shift used in BR by computing the shift value using three consecutive characters. EBR uses two sliding windows to scan the text from the left and right simultaneously. In each sliding window the comparisons made between the text and the pattern are made from both sides of the pattern at the same time. The proposed algorithm shows better performance compared with the existing algorithms in terms of number of comparisons and attempts.

[Dima Suleiman. **Enhanced Berry Ravindran Pattern Matching Algorithm (EBR).** *Life Sci J* 2014;11(7):395-402]. (ISSN:1097-8135). http://www.lifesciencesite.com. 47

Keywords: Pattern matching, Berry-Ravindran algorithm, Two Sliding Windows algorithm, Enhanced Two Sliding Windows Fast Pattern Matching Algorithm.

1. Introduction

Pattern matching algorithms have been used in many applications such as anti-viruses, search engines, anti plagiarism and many biological applications (Bhukya et al., 2011 ; Bhukya et al., 2010; Diwate et al., 2013; K.K.Senapati et al., 2010). Most of pattern matching algorithms search for a certain pattern *p* of length *m* in a text *t* of length *n* (Boyer et al.,1977; Chao et al.,2012; Diwate et al.,2013; Pendlimarri et al.,2010; Senapati et al., 2012). The main goal of these algorithms is to make the searching process faster and more efficient by making different enhancements, such enhancements can be made either on shifting value (Al-mazroi et al., 2011; Berry et al., 2001; Salmela et al.,2010;Senapati et al.,2012) during preprocessing phase or in a searching process (Hudaib et al., 2008;Hussain et al.,2013;Hlayel et al., 2014; Itriq et al., 2013). Enhancements on the shift value used to maximize the shift in case of a mismatch between the text and the pattern; in this case the amount of shift depends on the number of consecutive characters immediately after the pattern window (Berry et al., 2001; Suleiman et al., 2013). On the other hand enhancements may be made on the number of windows used in a searching process, some algorithms uses one window others use two or more (Hudaib et al., 2008; Itriq et al., 2013).

In this paper, a new pattern matching algorithm is implemented: EBR. While EBR algorithm uses the same comparisons techniques used in ETSW (Itriq et al., 2013), it uses different amount of shift in case of a mismatch. EBR uses three consecutive characters instead of two to determine the shift value, which make the searching process faster. Comparisons are made between the EBR algorithm, TSW (Hudaib et al., 2008), ETSW (Itriq et al., 2013) and Berry-Ravindran (BR) (Berry et al., 2001) algorithms. The experimental results showed that EBR is faster than the others in case of number of comparisons and number of attempts.

2. Related Works

Applications such as search engines, text processing and many others depend on pattern matching algorithms which make the searching algorithms one of the hot topics in research (Bhukya et al.,2011; Bhukya et al.,2010;Faro et al.,2012; Vangipuram et al., 2011). Most of pattern matching algorithms use two phases: preprocessing phase and searching phase. Most of researches have been made to make enhancements on either execution time or memory usage or both (Salmela et al., 2010).

The Berry-Ravindran algorithm (BR) (Berry et al., 2001) made enhancement on the preprocessing phase. Preprocessing phase used to determine the value of the shift in case of a mismatch between the text and the pattern. In BR (Berry et al., 2001) the shift value depends on using two consecutive characters in the text immediately to the right of the pattern. The pre-processing and searching time complexities of BR algorithm are $O(\sigma^2)$ and O(nm) respectively.

Two Sliding Windows algorithm (TSW) (Hudaib et al.,2008) uses the same preprocessing technique used in BR (Berry et al.,2001). The main difference between TSW (Hudaib et al.,2008) and BR (Berry et al.,2001) is in the searching phase while BR uses only one window, TSW uses two windows to scan the text from the left and right at the same time. By using two windows instead of one the searching process become faster and the number of comparisons and attempts minimized. In TSW, the best time complexity is O(*m*) and the worst case time complexity is O(((*n*/2-*m*+1))(*m*)). The pre-process time complexity is O(2(*m*-1)).

An enhancement has been made on TSW (Hudaib et al.,2008) to make a new algorithm, enhanced two sliding windows ETSW (Itriq et al.,2013). In ETSW algorithm the comparison technique between the text and the pattern improved by making parallel comparisons between the left side and the right side of the pattern at the same time. The same process applied to the two pattern windows, the best time complexity is O(*m/2*) and the worst case time complexity is O(((*n*/2-*m/2*+1))(*m/2*)). The pre-process time complexity is O(2(*m*-1)).

EBR algorithm uses the same searching process used in enhanced two sliding windows algorithm (ETSW). The searching process uses two sliding windows and the comparisons between the pattern and the text are made from both sides simultaneously. EBR made enhancements on Berry-Ravindran (BR) algorithm (Berry et al.,2001); while BR uses two consecutive characters of the text immediately following the pattern window to determine the amount of shift, EBR uses three which maximizes the shift and the efficiency of the searching process.

3. The Enhanced Berry Ravindran (EBR) Algorithm.

The EBR algorithm scan the text to search for a particular pattern *p* in a text *t* from both sides by using two sliding windows such as TSW (Hudaib 2008) algorithm, a comparison made between the patterns and text also happened in parallel from both sides of the pattern such as comparisons made in ETSW (Hudaib et al.,2008) algorithm. In new algorithm, the two widows aligned with text from both sides one from the left and the other from the right; in case of a mismatch the widows shifted according to the modifications of Berry Ravindran algorithm by using three consecutive characters instead of two. The searching process will stop either when the pattern found from either sides or in a case the pattern not found at all.

The main differences between EBR algorithm and ETSW algorithm are:

1. EBR uses new shifting algorithm which is a modification on BR (Berry et al., 2001) algorithm that depends on using three consecutive characters instead of two.
2. The EBR uses two arrays; each array is a one dimensional array of size *m*-2. The arrays are used to store the shift values for the two sliding windows. While TSW and ETSW uses two arrays of size m-1 the main reason for this is using three consecutive characters instead of two.

3.1 Pre-processing Phase

The pre-processing phase is used to determine the shift value in case of a mismatch at either the left or the right side of the text. This phase is used to generate two arrays *nextl* and *nextr*, each one of size m-2. The values of two arrays are calculated according to the modification on BR algorithm. *nextl* contains the shift values needed in case a mismatch happened from the left side. To calculate the shift values, the algorithm considers three consecutive text characters *a, b, c* which are aligned immediately after the left sliding window.

Initially, the indexes of the three consecutive characters in the text string from the left are *(m+1)*, *(m+2)* and *(m+3)* for *a*, *b* and *c* respectively as in Equation (1).

$$\mathrm{EBR}[a,b,c] = \min \left\{ \begin{array}{ll} 1 & \text{if } p[m-1]=a \\ 2 & \text{if } p[m-2][m-1]=ab \\ m+1 & \text{if } p[0]=b \\ m+2 & \text{if } p[0]=c \\ m-i & \text{if } p[i][i+1][i+2]=abc \\ m+3 & \text{otherwise} \end{array} \right\} \quad \ldots(1)$$

On the other hand, *nextr* contains the shift values needed when a mismatch occurs at the right side, initially the indexes of the three consecutive characters in the text string from the right are *(n-m-3), (n-m-2)* and *(n-m-1)* for *a*, *b, c* respectively, which are used to calculate the shift values as in Equation (2).

$$\mathrm{EBR}[a,b,c] = \min \left\{ \begin{array}{ll} m+2 & \text{if } p[m-1]=a \\ m+1 & \text{if } p[m-1]=b \\ 1 & \text{if } p[0]=c \\ 2 & \text{if } p[0][1]=bc \\ \ldots(2) & \\ m-((m-3)-i) & \text{if } p[i][i+1][i+2]=abc \\ m+3 & \text{otherwise} \end{array} \right\}$$

The two arrays will be invariable during the searching process. Figure 1 illustrates the steps of the pre-processing algorithm.

```
Begin
  shiftl=shiftr=m+3
  for (each character $p_i \in P_{i=0,.....m-3}$)
        {nextl[i]=m-i,nextr[i]=m-((m-3)-i)}

if P[m-1]=a {shiftl=1}
else if p[m-2][m-1]=ab { shiftl=2}
else if p[0]=b { shiftl=m+1}
else if p[0]=c { shiftl=m+2}
else if p[i][i+1][i+2]=abc { shiftl=nextl[i]}

if p[0]=c { shiftr=1}
else if p[0][1]=bc   { shiftr=2}
else if P[m-1]=a {shiftr=m+2}
else if P[m-1]=b {shiftr=m+1}
else if p[i][i+1][i+2]=abc { shiftr=nextr[i]}
End
```

Figure 1: The pre-processing algorithm

3.2 Searching Phase

In this phase, the search starts from both sides simultaneously using two windows, the right window aligned with the text from the right and the left window aligned with the text from the left. In case of a mismatch the right window will be shifted to the left and the left window will be shifted to the right using the shift values stored in *nextr* and *nextl* arrays respectively.

The two main steps of the EBR algorithm are:

Step1: After aligning the left and the right windows with text, comparisons between the windows and the text will happen at the same time. Comparisons will start from the beginning and the end of each window with the text by using four pointers, two for each side. The pointers called left pointer and right pointer as in ETSW (Itriq et al.,2013) algorithm. In case of a mismatch in either sides go to step 2; otherwise the left pointer will be shifted one step to right and the right pointer will be shifted one step to the left in each window in parallel, once the pointers intersected a complete match found.

Step2: In this step, a mismatch occurs between the patterns and the text, so that the left window will be shifted to the right according to the values in *nextl* and the right window will be shifted to left a number of steps depending on *nextr* array. In both cases the shift values depend on three consecutive characters in the text which placed immediately after the windows.

Both steps are repeated until the first occurrence of the pattern is found from either sides or until both windows are positioned beyond $\ulcorner n/2 \urcorner$.

3.3 Working Example

In this section I will present an example to clarify the new algorithm.

Given:

Pattern(P)="ACEBCCAB*", m*=8

Text(T)="DDCBCACABCCDACEBCCABCABCA CAACEBCACACCAEBCCDBCAEBCA*",n*=50

3.3.1 Pre-processing phase:

Initially, shiftl = shiftr = m+3 = 11.

The shift values are stored in two arrays *nextl* and *nextr* as shown in Figure 2(a) and Figure 2(b) respectively.

Shift Values from the left

Index	0	1	2	3	4	5	
	8	*7*	*6*	*5*	*4*	*3*	***nextl***

(a)

Shift Values from the right

Index	0	1	2	3	4	5
	3	**4**	**5**	**6**	**7**	**8**

nextr

(b)

Figure 2: The *nextl* and *nextr* arrays

To build the two next arrays (*nextl* and *nextr*), I take each three consecutive characters of the pattern and give it an index starting from 0. For example for the pattern structure ACEBCCAB, the consecutive characters ACE, CEB, EBC, BCC, CCA and CAB are given the indexes 0,1,2,3,4 and 5 respectively.

The shift values for the *nextl* array are calculated according to Equation (1) while the shift values for the *nextr* array are calculated according to Equation (2).

3.3.2 Searching Phase:

The searching process for the pattern *P* is illustrated through the working example as shown in Figure 3.

First attempt:

In the first attempt (see Figure 3(a)), I align the left window with the text from the left. A comparisons are made between the first character of the text from the left (D) with the first character of the window from the left (A) and at the same time character at index (7) from the text (A) with the last character of the window (B) a mismatch occurs; therefore I take three consecutive characters from the text at indexes 8, 9 and 10 which are (B, C and C) respectively. To determine the amount of shift (*shiftl*) I have to do the following:

Since p[m-1]=a; B=B then according to a preprocessing algorithm the shift value will be 1.

Second attempt:

In the second attempt (see Figure 3(b)), I align the right window with the text from the right. A comparisons are made between the character at index (42) from the text (D) with the first character from the right window (A) and at the same time between the last character of the text (A) with the last character of the window (B) a mismatch occurs; therefore I take the three consecutive characters from the text at indexes 39, 40 and 41 which are (B, C and C) respectively. To determine the amount of shift (*shiftr*), I have to do the following two steps:

a) Find the index of BCC in the pattern which is *3*.
b) Since the search occurred from the right side I use *nextr* array for index (3):

nextr[3]=6, then the shift value will be 6.

Therefore the window will be shifted to the left 6 steps.

Third attempt:

In the third attempt (see Figure 3(c)), a match occurs from the left between text character (B) at index (8) and pattern character (B) but since a mismatch occurs between text character (D) at index (1) and pattern character (A) , so comparisons will stop and a mismatch occurs; therefore I take the three consecutive characters from the text at indexes 9, 10 and 11 which are (C, C and D) respectively, since CCD is not found in the pattern, so the window will be shifted to the right 11 steps.

Fourth attempt:

In the fourth attempt (see Figure 3(d)), a match occurs from the left between text character (B) at index (43) and pattern character (B) but since a mismatch occurs between text character (C) at index (36) and pattern character (A), so comparisons will stop and a mismatch occurs; Since P[0][1] =AC =cd according to a preprocessing algorithm equation(2) the shift value will be *2* so the pattern will be shifted two steps to the left.

Fifth attempt:

Align the left most character of the pattern P[0]with T[12]. Comparisons between the pattern and the text characters leads to a complete match at index 12. In this case, the pattern was found using the left window.

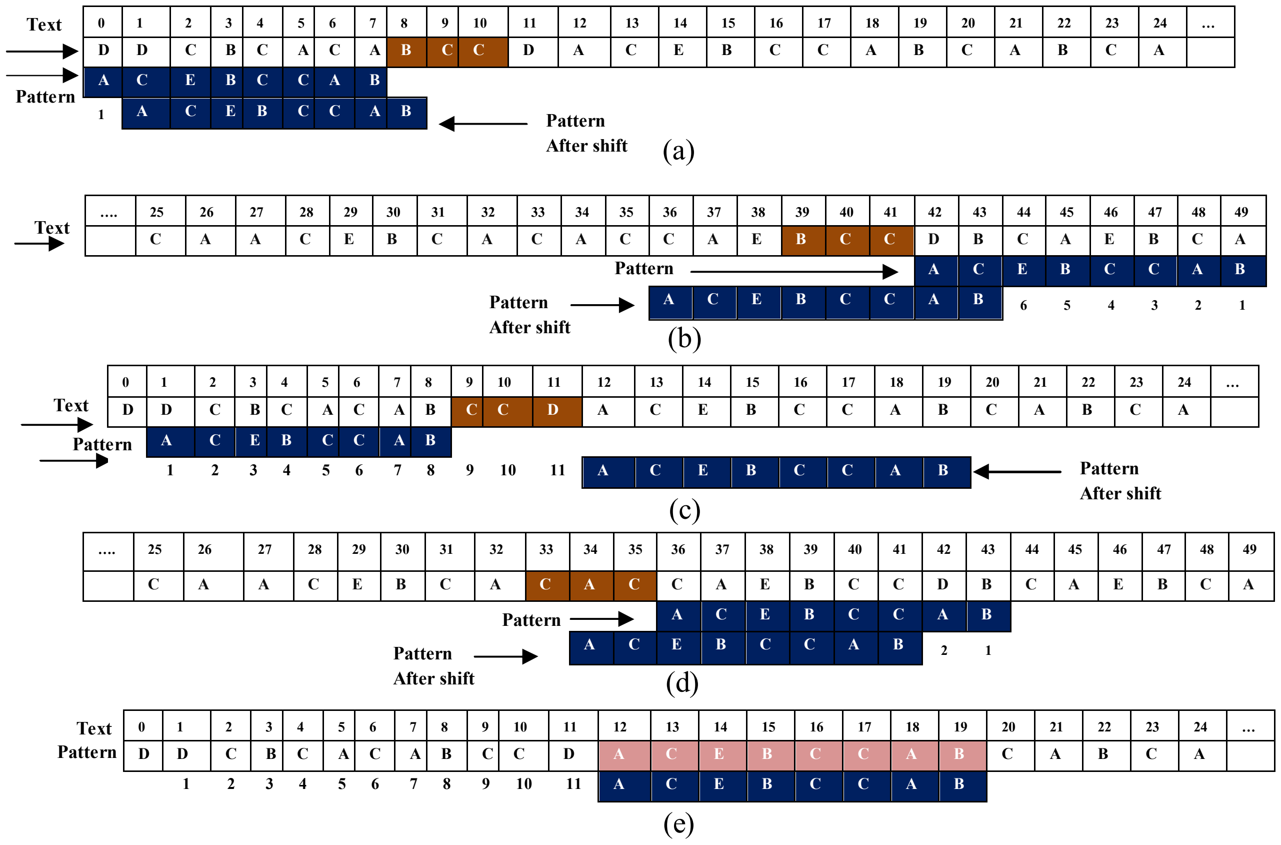

Figure 4: Working Example

4. Analysis

Preposition 1: The space complexity is *O(2(m-2))* where m is the pattern length**.**

Preposition 2: The pre-process time complexity is *O(2(m-2))***.**

Lemma 1: The worst case time complexity is *O(((n/2-m+1))(m))*

Proof: The worst case occurs when at each attempt a match occurs between all pattern character except the one at the middle, and at the same time the shift value is equal to 1.

Lemma 2: The best case occurs when the pattern is found at the first index or at the last index (n-m). In these cases the complexity is *O*(m).

Lemma 3: The Average case time complexity is *O(n/(2*(m+3)))*

Proof: The Average case occurs when the three consecutive characters of the text directly following the sliding window is not found in the pattern. In this case, the shift value will be (*m*+3) and hence the time complexity is *O([n/(2*(m+3))])*.

5. Experimental Results and Discussion

Many experiments have been done in EBR algorithm using Book1 from the Calgary corpus to be the text (Calgary corpus). Book1 consists of 141,274 words (752,149 characters). Patterns of different lengths are also taken from Book1.

Table 1 and Figure 4 show the results of comparing the algorithms TSW, ETSW and EBR and Figure 4(a) and Figure 4(b) represents the average number of attempts and comparisons respectively.

In Table 1 the first column represents the pattern length; second column is the number of words of a certain length. It's obvious from the results that the number of attempts and comparisons in EBR is better than the other algorithms. For example, as shown in Table 1, 1167 words of length 8, the average number of comparisons in TSW is 11087, in ETSW is 10115 and in the new algorithms is 9198 which is the minimum value among the others values. The same results can be shown about the average number of attempts.

Although EBR algorithm uses the same techniques of comparisons used in ETSW it uses different shifting algorithm which depends on using three consecutive characters of the text, while ETSW and TSW algorithm uses only two consecutive characters, so that the average number of comparisons and attempts in ETSW and TSW algorithm are more than that of EBR.

Table 1: The average number of attempts and comparisons of TSW, ETSW and EBR algorithms

Pattern length	Number of words	TSW		ETSW		EBR	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	4535	4456	4896	4456	3549	3174	3193
6	2896	7596	8311	7596	7633	6777	6818
7	1988	9341	10263	9341	9118	8223	8276
8	1167	10056	11087	10056	10115	9131	9198
9	681	9538	10538	9538	9590	8707	8765
10	382	9283	10272	9283	9339	8512	8576
11	191	5451	5967	5451	5482	5009	5045
12	69	6384	7168	6384	6433	5908	5966
13	55	7947	8673	7947	7986	7364	7408
14	139	19437	21319	19437	19535	18031	18144
15	32	19682	21739	19682	19782	18253	18367
16	10	20029	21596	20029	20092	18569	18641
17	3	21897	25404	21897	22147	20340	20639

Table 2 shows the average number of attempts and comparisons for 100 words taken from the right side of Book1. Clearly can be seen that EBR is the best among the others since it maximize the shift value in case of a mismatch.

Table 3 shows the average number of attempts and comparisons for 100 words taken from the middle of Book1 while Table 4 shows the average number of attempts and comparisons for 100 words taken from the left side of Book1 and the same results shown in Table 2 are shown here.

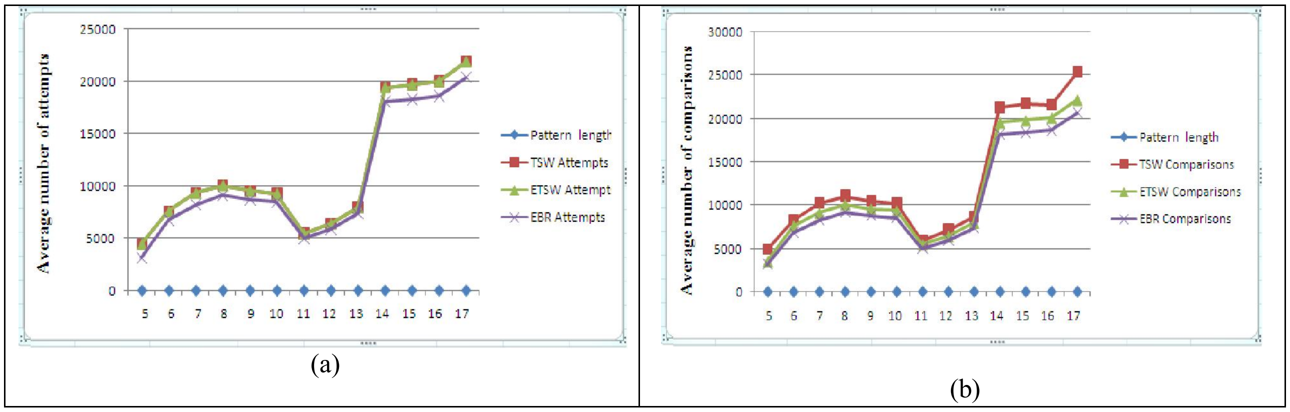

Figure 4: The average number of attempts and comparisons of TSW, ETSW and EBR algorithms

Table 2: The average number of attempts and comparisons performed to search for (100) patterns selected from the right side of the text

Pattern length	**Number of words**	**TSW**		**ETSW**		**EBR**	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	185	206	185	187	165	167
6	100	227	255	227	230	201	205
7	100	347	388	347	351	314	318
8	100	504	568	504	510	458	465
9	100	670	750	670	677	612	619
10	100	1160	1290	1160	1170	1065	1076
11	100	622	705	622	628	568	574
12	100	865	972	865	878	797	811

Table 3: The average number of attempts and comparisons performed to search for (100) patterns selected from the middle of the text

Pattern length	**Number of words**	**TSW**		**ETSW**		**EBR**	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	13965	15140	13965	11618	9113	9145
6	100	16682	18317	16682	16771	14895	14997
7	100	27267	30095	27267	26242	23701	23855
8	100	27830	30915	27830	28015	25262	25470
9	100	33929	37200	33929	34069	30928	31087
10	100	29676	32817	29676	29845	27208	27403
11	100	23195	24646	23195	23242	21234	21283
12	100	26806	30222	26806	27009	24804	25052

Table 5 compares between four algorithms BR, TSW, ETSW and EBR. The results are reasonable EBR is the best among the other algorithms and this is related to three reasons: First: It scans the text from both side simultaneously using two windows. Second it compares the pattern with text using two pointers at the same time. Finally it uses three consecutive characters instead of two so that the shift value will be maximized

Table 4: The average number of attempts and comparisons performed to search for (100) patterns selected from the left side of the text

Pattern length	**Number of words**	**TSW**		**ETSW**		**EBR**	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	271	297	271	270	208	210
6	100	364	402	364	368	325	329
7	100	402	447	402	405	361	365
8	100	536	592	536	541	487	493
9	100	776	859	776	783	710	717
10	100	1579	1756	1579	1593	1451	1466
11	100	619	669	619	624	572	577
12	100	1667	1872	1667	1685	1547	1567

Table 5: The average number of attempts and comparisons for patterns with different lengths

Pattern length	Number of words	TSW		BR		ETSW		EBR	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
4	8103	3904	4213	6409	7039	3904	3875	3174	3193
5	4535	4456	4896	9577	10645	4456	3549	6777	6818
6	2896	7596	8311	10898	12173	7596	7633	8223	8276
7	1988	9341	10263	11953	13345	9341	9118	9131	9198
8	1167	10056	11087	13256	14807	10056	10115	8707	8765
9	681	9538	10538	14149	15892	9538	9590	8512	8576
10	382	9283	10272	14127	15799	9283	9339	5009	5045
11	191	5451	5967	12808	14243	5451	5482	5908	5966
12	69	6384	7168	9598	10923	6384	6433	7364	7408
13	55	7947	8673	10334	11370	7947	7986	18031	18144
14	139	19437	21319	19548	21673	19437	19535	18253	18367
15	32	19682	21739	19817	22384	19682	19782	18569	18641
16	10	20029	21596	26086	28644	20029	20092	20340	20639
17	3	21897	25404	22554	28148	21897	22147	3174	3193

1. Conclusion and Future Work

In this research a new pattern matching algorithm EBR was implemented. While EBR uses the same comparisons techniques used in ETSW (Itriq et al.,2013), it also enhances the techniques of calculating the shift value. The shift value calculated depends on using three consecutive characters instead of two to determine the amount of shift to maximize the shift value and reduce the number of comparisons and attempts. As many searching algorithms, EBR can be used in applications related to Biological sequence such as DNA.

Performance of EBR was evaluated by comparing it with many algorithms such as BR, TSW and ETSW, and in all cases it was considered the best.

References

1. Al-mazroi A, Rashid N. A Fast Hybrid Algorithm for the Exact String Matching Problem. American J. of Engineering and Applied Sciences. 2011, 4 (1): 102-107.
2. Berry, T. and Ravindran, S., "A Fast String Matching Algorithm and Experimental Results". In Proceedings of the Prague Stringology Club Workshop '99 (eds Holub, J.and Simanek, M), Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic, 2001, pp. 16-26.

3. Bhukya R., Somayajulu D. Multiple Pattern Matching Algorithm using Pair-count. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 2, July 2011 ISSN (Online): 1694-0814.
4. Bhukya R., Somayajulu D. An Index based Forward Backward MultiplePattern Matching Algorithm. World Academy of Science, Engineering and Technology 42 2010: 1513-1521.
5. Boyer, R. S. and Moore, J. S., "A Fast String Searching Algorithm". *Commun. ACM*, 1977, 20, 762-772.
6. Calgary Corpus available at: ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/
7. Chao Y. An Improved BM Pattern Matching Algorithm in Intrusion Detection System. Applied Mechanics and Materials (Volumes 148 - 149) 2012.
8. Diwate R, Alaspurkar S. Study of Different Algorithms for Pattern Matching. International Journal of Advanced Research in Computer Science and Software Engineering. 2013; Volume 3, Issue 3, 615-620.
9. FARO S, EFFICIENT VARIANTS OF THEBACKWARD-ORACLE-MATCHING ALGORITHM. International Journal of Foundations of Computer Science, 2009, Vol. 20, No. 6: 967–984.
10. Faro S, K¨ulekci‡M. O. Fast Packed String Matching for Short Patterns.arXiv:1209.6449v1 [cs.IR] 28 Sep 2012.
11. Hudaib A., Al-Khalid R., Suleiman D., Itriq M. and Al-Anani A. A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW). Journal of Computer Science 2008; 4 (5): 393-401.
12. Hussain I, Kausar S, Hussain L, Khan M. Improved Approach for Exact Pattern Matching(Bidirectional Exact Pattern Matching). IJCSI International Journal of Computer Science Issues. 2013; Vol. 10, Issue 3, 59-65
13. Hussain I,Kazmi S,Khan I,Mehmood R. Improved-Bidirectional Exact Pattern Matching. International Journal of Scientific & Engineering Research. 2013; Volume 4, Issue 5, 659-663.
14. Hlayel, Abdallah A.; Hnaif, Adnan A. A New Exact Pattern Matching Algorithm (WEMA). Journal of Applied Sciences . 2014, Vol. 14 Issue 2, p193-196. 4p.
15. Itriq M., Hudaib A., Al-Anani A., Al-Khalid R. and Suleiman D. Enhanced Two Sliding Windows Algorithm For Pattern Matching (ETSW). Journal of American Science 2012;8(5): 607- 616.
16. K.K.Senapati, G.Sahoo, S.Sahana" An Efficient pattern matching algorithm for biological sequence". Proceedings of the International conference on Image processing, Computer Vision and Pattern Recognition (IPCV2010), VOL-II, PP-755-759, LasVegas, USA.
17. Pendlimarri D. and Petlu P. B. B. Novel Pattern Matching Algorithm for Single Pattern Matching.International Journal on Computer Science and EngineeringVol. 02, No. 08, 2010, 2698-2704
18. Suleiman D,Hudaib A, Al-Anani A,Al-Khalid R, Itriq M. ERS-A Algorithm for Pattern Matching. Middle East Journal of Scientific Research.2013. Vol. 15 Issue 7, p 1067-1075
19. Salmela L. Tarhio J. Kalsi P. Approximate Boyer-Moore String Matching for Small Alphabets. Volume 58, Number 3, November 2010 , pp. 591-609(19)
20. Senapati K.K., Mal S., Sahoo G. RS-A Fast Pattern Matching Algorithm for Biological Sequences. International Journal of Engineering and Innovative Technology (IJEIT)Volume 1, Issue 3, March 2012: 116- 118.
21. Vangipuram R. K.,Sandeep S. J., Reddy A. Text Segmentation Based Pattern SearchAlgorithm. International Journal of Wisdom Based Computing, Vol. 1(3), December 2011

4/15/2014