Middle-East Journal of Scientific Research 15 (7): 1067-1075, 2013
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2013.15.7.11109

ERS-A Algorithm for Pattern Matching

[1]Dima Suleiman, [2]Amjad Hudaib, [2]Aseel Al-Anani, [2]Rola Al-Khalid and [3]Mariam Itriq

[1]Department of Business Information Technology, The University of Jordan, Jordan
[2]Department of Computer Information System, The University of Jordan, Jordan
[3]Department of Business Information Technology, The University of Jordan, Jordan

Abstract: Pattern matching algorithms have many applications that cover a wide range including information retrieval, text processing, DNA sequence analysis and pattern recognition. In this paper, we propose a new algorithm ERS-A, that made enhancements on both two sliding windows (TSW) and Fast Pattern Matching (RS-A) algorithms. In ERS-A and TSW algorithms two sliding windows are used to scan the text from the left and right simultaneously, but while TSW utilizes the idea of Berry Ravindran bad character shift function (BR), ERS-A adds an improvement by using the shift technique provided by RS-A algorithm. RS-A algorithm uses four consecutive characters in the text immediately following the pattern window, instead of using two consecutive characters as in BR. The experimental results show that the ERS-A has enhanced the searching process significantly.

Key words: Pattern matching · Berry-Ravindran algorithm · Two Sliding Windows algorithm · RS-A Fast Pattern Matching Algorithm

INTRODUCTION

Pattern matching algorithms are used in many applications vary from search engines to more complex systems such as biological applications, especially that related to DNA [1-4]. Such algorithms differ from each other according to different criteria, some of them focus on altering the searching process [5, 6], shifting values [7-10] and preprocessing techniques [9, 11], but in all cases the main goal is to make the searching process faster to accommodate the needed purposes.

Pattern matching algorithms search for a certain pattern p of length m in a text t of length n. Some algorithms use one window [7, 12-18]; whose size is equal to the pattern length m, for searching process. Others using two windows [6, 5, 19], one from the left and the other from the right each of length $\boldsymbol{m}$, in this case, the comparisons between the text and two windows happened at the same time. Other Algorithms depend on making some modification on the shifting values. After aligning the pattern with the text a mismatch may occur and the pattern will be shifted. The shifting value vary from one algorithm to other, such variations depend on the number of consecutive characters in the text immediately after the pattern window, some algorithms using one, two, three and others may use four characters.

In this paper, we propose a new pattern matching algorithm which made an enhancement on RS-A so we called it ERS-A. The algorithm uses two sliding windows such that used in TSW algorithm [6], but made an enhancement on the shift values by using RS-A algorithm [7]. In this case the Preprocessing phase of TSW algorithm will be changed to take into account four consecutive characters instead of two consecutive characters.

Comparisons are made between the new algorithm and three others algorithms, TSW [6], ETSW [5] and RS-A [7]. The experimental results section showed that the new algorithm is better than the others in case of the number of comparisons and the number of attempts. The rest of this paper is organized as follows: The next section introduces some literature review about the topic; it is followed by a section that covers the ERS-A algorithm. We then present the analysis. The conclusion and future work is drawn in the last section.

Corresponding Author: Dima Suleiman, Department of Business Information Technology, The University of Jordan, Jordan.

Related Works: Many Pattern matching algorithms have been developed to cover the diversity of needed applications [1-3, 17, 20]. Most of these algorithms depend on two phases: preprocessing phase and searching phase. Some algorithms try to make enhancement in either preprocessing or searching phase, while others improve both [9]. The needed improvement was to minimize the execution time which can be achieved by reducing the number of comparisons between the text and patterns. Other improvement needed to minimize the memory capacity needed in the preprocessing phase [9, 11].

The Berry-Ravindran algorithm (BR) [10] made some adjustment in a preprocessing phase of the searching algorithm. The shifting values depend on the bad character shift for two consecutive characters in the text immediately to the right of the pattern window. The pattern is aligned with the text from the left, in case of a mismatch the pattern will be shifted to the right. The pre-processing and searching time complexities of BR algorithm are O(σ2) and O(nm) respectively.

An enhancement on BR algorithm was improved by using Two Sliding Window algorithm (TSW). TSW uses two windows instead of one, each of them equal to the length of the pattern *m*. One window aligned with the text from the left and the other aligned from the right. Scanning the text happened at the same time from both sides, in case of a mismatch the two windows will be shifted. The left window will be shifted to the right and the right window will be shifted to left. The process will stop either if there is a match or if the pattern is not found at all. 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)).

In order to minimize the number of comparisons, Enhanced Two Sliding Window algorithm (ETSW) [5] made some modification on TSW. The preprocessing phase remains the same; however the modifications happened on the comparison process between the text and the two windows in searching phase. Instead of comparing one character at a time, parallel comparisons happened between the text and the pattern by using two pointers one from the left of the pattern and the other from the right of the same pattern. The same process applied to the two 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)).

RS-A [7] maximizes the efficiency of BR [10]. While BR uses two consecutive characters in the text immediately to the right of the pattern window; RS-A uses four characters. The modifications happened on preprocessing phase, other issue is that BR scanning the text from the left to the right while RS-A scanning it from the right to the left.

ERS-A uses two sliding windows in the searching process, the same as in TSW [6]. In addition to using RS-A algorithm to calculate the shift values of the right pattern, we made some enhancement to calculate the shifting values for the left pattern depending on RS-A algorithm. The result is a new algorithm (ERS-A) that maximizes the efficiency of the searching process.

The Enhanced RS-A Algorithm (ERS-A): ERS-A algorithm uses two sliding windows to search for a certain pattern *p* of size *m* in a text *t* of size *n*, where each window is of size *m*. As with TSW algorithm the two windows aligned with the text one from the left and the other from the right, where the two windows scan the text at the same time. In case of a mismatch in both windows the two windows will be shifted. The shifting values depend on RS-A algorithm to get better shift values. The searching process will stop either when the pattern does not exist at all in the text or when the pattern is found by the left or the right window.

The main differences between ERS-A and RS_A algorithms are: ERS-A uses two sliding windows rather than one to scan all the text characters while RS_A uses only one window to scan the text from the right. Another difference is that the ERS-A uses two arrays; each array is a one dimensional array of size *m-3*, the arrays are used to store the calculated shift values for the two sliding windows, while the original RS_A algorithm uses two variables to store the shift values. In both algorithms the shift values are calculated only for the pattern characters.

The main difference between ERS-A algorithm and TSW algorithm is: The ERS-A uses two arrays; each array is a one dimensional array of size *m-3* while TSW uses two arrays each array is a one dimensional array of size *m-1*. The main reason for this reduction is related to using four consecutive characters instead of two.

Pre-Processing Phase: The pre-processing phase is used to generate two arrays *nextl* and *nextr*, each array is a one-dimensional array. The values of the *nextl* array are calculated according to our proposed shift function.

nextl contains the shift values needed to search the text from the left side. To calculate the shift values, the algorithm considers four consecutive text characters *a*, *b*, *c* and *d* which are aligned immediately to the right of the sliding window. Initially, the indexes of these four consecutive characters (*a,b,c,d*) in the text string from the left are (*m+1*), (*m+2*), (*m+3*) and (*m+4*) respectively as in Equation (1) where *m* is the pattern length.

ERS-A left shift value [a,b,c,d]

$$= \min \begin{cases} 1 & \text{if } p[m-1]=a \\ 2 & \text{if } p[m-2][m-1]=ab \\ 3 & \text{if } p[m-3]\ [m-2][m-1]=abc \\ m+1 & \text{if } p[0]=b \\ m+2 & \text{if } p[0]=c \\ m+3 & \text{if } p[0]=d \\ m-i & \text{if } p[i][i+1][i+2][i+3]=abcd \\ m+4 & \text{otherwise} \end{cases} \quad ...(1)$$

On the other hand, the values of the *nextr* array are calculated according to RS-A algorithm. *nextr* contains the shift values needed to search in the text from the right side. Initially the indexes of the four consecutive characters in the text string to the left of the right window are *(n-m-4), (n-m-3), (n-m-2) and (n-m-1)* for *a*, *b*, *c* and *d* respectively, which are used to calculate the shift values as in Equation (2).

ERS-A right shift value [a,b,c,d]

$$= \min \begin{cases} m+3 & \text{if } p[m-1]=a \\ m+2 & \text{if } p[m-1]=b \\ m+1 & \text{if } p[m-1]=c \\ 1 & \text{if } p[0]=d \\ 2 & \text{if } p[0][1]=cd \\ 3 & \text{if } p[0][1][2]=bcd \\ m-((m-4)-i) & \text{if } p[i][i+1][i+2][i+3]=abcd \\ m+4 & \text{otherwise} \end{cases} \quad ...(2)$$

The two arrays will not be changed during the searching process. Figure 1 illustrates the steps of the pre-processing algorithm.

Searching Phase: In this phase, the text is scanned from both sides using the left and the right windows.

The searching process is stopped when the pattern is found either from the beginning or from the end of the text. In case of a mismatch, the left window will be shifted to the right using the values in *nextl*, while the right window will be shifted to the left using the values in *nextr* array.

```
1. Begin
2. shiftl=shiftr=m+4
3. for (each character p_i ∈ P_{i=0,.....m-4} )
4.       {nextl[i]=m-i,nextr[i]=m-((m-4)-i)}
5. if P[m-1]=a {shiftl=1}
6. else if p[m-2][m-1]=ab { shiftl=2}
7. else if p[m-3][m-2][m-1]=abc { shiftl=3}
8. else if p[0]=b { shiftl=m+1}
9. else if p[0]=c { shiftl=m+2}
10. else if p[0]=d { shiftl=m+3}
11. else if p[i][i+1][i+2][i+3]=abcd { shiftl=nextl[i]}
12. if p[0]=d { shiftr=1}
13. else if p[0][1]=cd  { shiftr=2}
14. else if p[0][1][2]=bcd { shiftr=3}
15. else  if P[m-1]=a{shiftr=m+3}
16. else if P[m-1]=b {shiftr=m+2}
17. else if P[m-1]=c {shiftr=m+1}
18. else if p[i][i+1][i+2][i+3]=abcd { shiftr=nextr[i]}
19. End
```

Fig. 1: The pre-processing algorithm

The two main steps of the ERS-A searching phase algorithm are:

Step1: After aligning the text with the two windows, comparisons are made between the first character of the text from the left and the first character of the left window and also between the last character of the text from the right and the last character of the right window. If there is a mismatch go to step2; otherwise the comparison process between the text and the patterns continues until a complete match is found.

Step 2: In this step, we use the shift values from the *nextl* and *nextr* arrays depending on the four text characters placed immediately after the pattern windows. The four characters located to the right side of the left window and four characters located to the left side of the right window. The corresponding windows are shifted to the correct positions based on the shift values, the left window is shifted to the right and the right window is shifted to the left. Both steps are repeated until the first occurrence of the pattern is found from any sides or until both windows are positioned beyond $\lfloor n/2 \rfloor$.

Working Example: In this section, we will present an example to clarify the new algorithm. Given:
Pattern(P)=”ABBCEDAB”, m=8,
Text(T)=”ACDABABBACEDABBCEDAABDABABBCEDABAABABBCEDCDABEDAB”, n=50

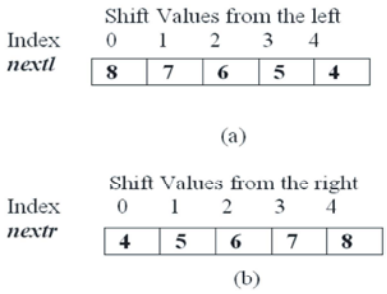

Fig. 2: The nextl and nextr arrays

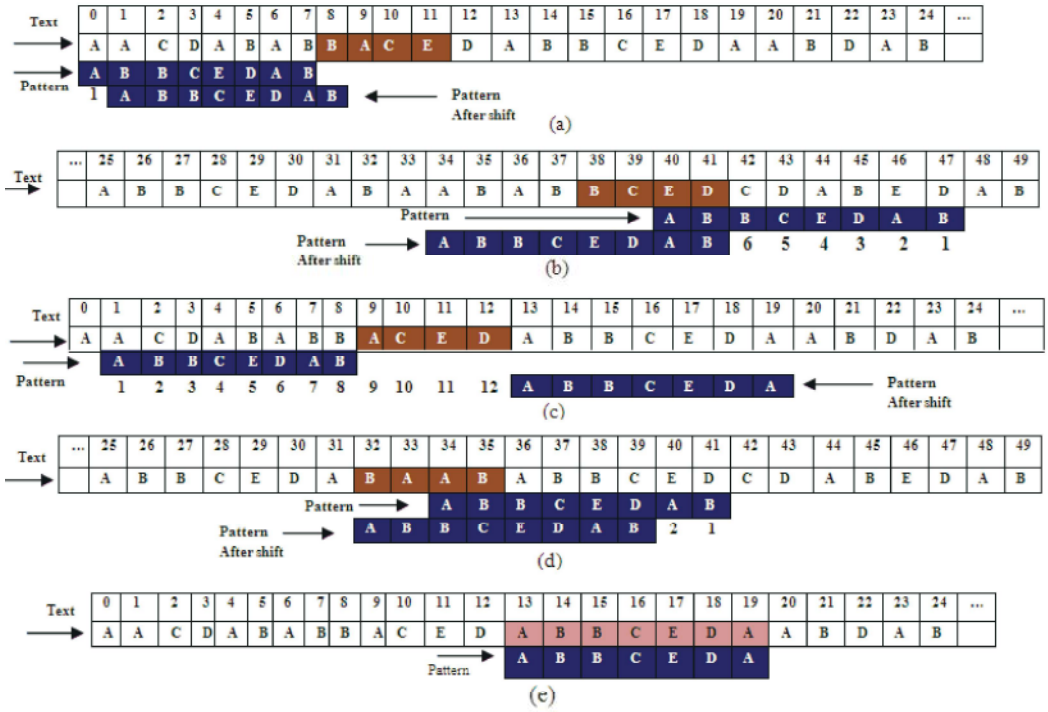

Fig. 3: Working Example

Pre-Processing Phase: Initially, we have to build the two next arrays (*nextl* and *nextr*). We can use the for loop in line number 3 in Figure 1, in this case the shift values are stored in two arrays *nextl* and *nextr* as shown in Figure 2(a) and Figure 2 (b) respectively.

After creating *nextl* and *nextr* arrays, we have to determine the shift values from the left and right. The left shift values stored in *shiftl* which depends on Equation (1) and the right shift values stored in *shiftr* which depends on Equation (2).

According to the line number 2 in preprocessing algorithm the initial values of shiftl = shiftr = m+4 = 12 which may be changed.

To determine the values of *shiftl* and *shiftr* depending on *nextl* and *nextr* arrays, we must take each four consecutive characters of the pattern and give it an index starting from 0. For example for the pattern structure ABBCEDAB, the consecutive characters ABBC, BBCE, BCED, CEDA and EDAB are given the indexes 0,1,2,3 and 4 respectively.

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 (Figure 3(a)), we align the first sliding window with the text from the left. In this case, a mismatch occurs between text character(A) at T[1] and pattern character (B) at P[1]; therefore we take the four consecutive characters of the text at index 8, 9, 10 and 11 which are (B, A, C and E) respectively to determine the left shift value (*shiftl*).

Initially *shiftl* =12, according to Figure 1 the if statement in line number 5 is true that is p[m-1]=*a*.
p[m-1]=P[7]=B and *a*=T[8]=B then according to a preprocessing algorithm the shift value will be 1.

Second attempt: In the second attempt (Figure 3(b)), we align the second sliding window with the text from the right. In this case, a mismatch occurs between text character (B) at T[45] and pattern character (C) at P[3]; therefore, we take the four consecutive characters of the text at index 38, 39, 40 and 41 which are (B, C, E and D) respectively. To determine the amount of shift (*shiftr*), we have to do the following two steps:

We find the index of BCED in the pattern which is 2. Since we search from the right side, we use *nextr* array of index (2): *nextr*[2]=6, then the shift value will be 6. Therefore the right window will be shifted to the left 6 steps.

Third Attempt: In the third attempt (Figure 3(c)), a mismatch occurs from the left between text character (C) at T[2] and pattern character (B) at P[1]; therefore we take the four consecutive characters from the text at indexes 9, 10, 11 and 12 which are (A, C, E and D) respectively. Since ACED is not found in the pattern, so the window will be shifted to the right 12 steps.

Fourth Attempt: In the fourth attempt (Figure 3(d)), a mismatch occurs from the right between text character (D) at T[43] and pattern character (B) at P[7]. According to line number 13 in Figure 1, the if statement is true since P[0][1] = *cd*=AB, so the right shift value *shiftr* will be 2 then the pattern will be shifted two steps to the left.

Fifth Attempt: The fifth attempt (Figure 3(e)), we align the left most character of the pattern P[0]with T[13]. A comparison between the pattern and the text characters leads to a complete match at index 13. In this case, the occurrence of the pattern is found using the left window.

Analysis

Preposition 1: The space complexity is O(2(m-3)) where *m* is the pattern length.

Preposition 2: The pre-process time complexity is O(2(m-3)).

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

Proof: The worst case occurs when at each attempt, all the characters of both the pattern and the text are matched except the last character and at the same time the shift value is equal to 1. If the pattern is aligned from the left then shift by one occurs when the first character of the two consecutive characters is matched with the last pattern character, while if the pattern is aligned from the right then shift by one occurs when the second character of the two consecutive characters is matched with the first pattern character.

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+4)))

Proof: The Average case occurs when the four 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+4) and hence the time complexity is O([n/(2*(m+4))]).

RESULTS AND DISCUSSION

As many pattern matching algorithm, many experiments have been done in ERS-A algorithm using Book1 from the Calgary corpus to be the text [21]. Book1 consists of 141,274 words (752,149 characters). Patterns of different lengths are also taken from Book1.

Table1 and Figure 4 show the results of comparing the algorithms TSW, RS-A and ERS-A. Figure 4(a) and Figure 4(b) represent the average number of attempts and comparisons respectively.

In Table 1, the first column represents the pattern length; the second column is the number of words of a certain length. According to the result, the number of comparisons and the number of attempts of ERS-A algorithm is always better than TSW and RS-A. For example, as shown in Table 1, 681 words of length 9, the average number of comparisons in TSW is 10538, in RS-A is 12911 and in the new algorithms is 8957, which is the minimum value among the others values. The same result can be shown about the average number of attempts. Although ERS-A algorithm uses the same shift function of RS-A algorithm the ERS-A algorithm searches the text from both sides simultaneously, since RS-A algorithm only searches the text from the right side, so

Table 1: The average number of attempts and comparisons of TSW, RS-A and ERS-A algorithms

		TSW		RS-A		ERS-A	
Pattern length	Number of words	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	4535	4456	4896	7361	8191	3533	3880
6	2896	7596	8311	8545	9556	6166	6750
7	1988	9341	10263	9512	10638	7737	8506
8	1167	10056	11087	10660	11922	8451	9319
9	681	9538	10538	11477	12911	8106	8957
10	382	9283	10272	11543	12927	7970	8830
11	191	5451	5967	10480	11672	4701	5146
12	69	6384	7168	7927	9030	5589	6286
13	55	7947	8673	8560	9422	6955	7587
14	139	19437	21319	16086	17845	17115	18776
15	32	19682	21739	16176	18318	17385	19198
16	10	20029	21596	21411	23531	17722	19147
17	3	21897	25404	18551	23119	19521	22669

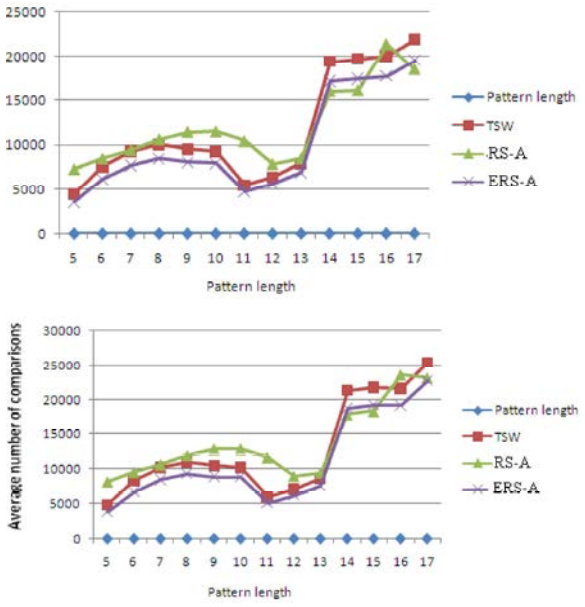

Fig. 4: The average number of attempts and comparisons of TSW, RS-A and ERS-A algorithms

the average number of comparisons and attempts in ERS-A algorithm are less than that of the RS-A algorithm. The same conclusion results when comparing new algorithm with TSW, ERS-A algorithm uses four consecutive characters instead of two, so that the number of comparisons and attempts is lower and the searching process is faster.

Table 2 shows the average number of attempts and comparisons for 100 words taken from the right side of Book1. Clearly, we can see that RS-A is the best among the others since it scan the text only from the right hand side. But we can see the different results on Table 3 and Table 4, where ERS-A algorithm is the best.

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.

Performance of ERS-A algorithm is observed in Table 5, which contains the number of attempts and comparisons performed to search for a set of patterns that

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

		TSW		RS-A		ERS-A	
Pattern length	Number of words	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	185	206	89	76	146	163
6	100	227	255	95	112	182	205
7	100	347	388	154	181	286	324
8	100	504	568	217	250	424	476
9	100	670	750	296	341	571	640
10	100	1160	1290	523	599	999	1117
11	100	622	705	269	319	529	597
12	100	865	972	390	457	756	860

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

		TSW		RS-A		ERS-A	
Pattern length	Number of words	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	13965	15140	8871	9833	11038	11970
6	100	16682	18317	10135	11294	13536	14870
7	100	27267	30095	15497	17487	22607	24971
8	100	27830	30915	16748	18783	23385	25976
9	100	33929	37200	18822	20929	28764	31541
10	100	29676	32817	16766	18783	25471	28193
11	100	23195	24646	16587	18028	19886	21119
12	100	26806	30222	17706	20313	23484	26507

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

		TSW		RS-A		ERS-A	
Pattern length	Number of words	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	271	297	9942	11087	216	238
6	100	364	402	16685	18783	295	326
7	100	402	447	17682	19682	333	372
8	100	536	592	17198	19078	451	499
9	100	776	859	20214	22954	660	730
10	100	1579	1756	25880	29000	1361	1517
11	100	619	669	26554	29572	531	573
12	100	1667	1872	25333	28768	1459	1641

Table 5: The number of attempts and comparisons performed to search for a set of patterns that do not exist in the text

	TSW		RS-A		ERS-A	
Pattern length	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	888	978	709	793	703	771
6	778	852	638	712	635	694
7	684	739	575	642	570	616
8	598	612	500	512	499	500
9	545	561	468	480	464	477
10	512	552	454	509	445	479
11	472	507	424	474	415	445
12	870	974	789	890	776	869

Table 6: The average number of attempts and comparisons for patterns with different lengths

		TSW		BR		ETSW		RS-A		ERS-A	
Pattern	Number										
length	of words	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	4535	4456	4896	9577	10645	4456	3549	7361	8191	3533	3880
6	2896	7596	8311	10898	12173	7596	7633	8545	9556	6166	6750
7	1988	9341	10263	11953	13345	9341	9118	9512	10638	7737	8506
8	1167	10056	11087	13256	14807	10056	10115	10660	11922	8451	9319
9	681	9538	10538	14149	15892	9538	9590	11477	12911	8106	8957
10	382	9283	10272	14127	15799	9283	9339	11543	12927	7970	8830
11	191	5451	5967	12808	14243	5451	5482	10480	11672	4701	5146
12	69	6384	7168	9598	10923	6384	6433	7927	9030	5589	6286
13	55	7947	8673	10334	11370	7947	7986	8560	9422	6955	7587
14	139	19437	21319	19548	21673	19437	19535	16086	17845	17115	18776
15	32	19682	21739	19817	22384	19682	19782	16176	18318	17385	19198
16	10	20029	21596	26086	28644	20029	20092	21411	23531	17722	19147
17	3	21897	25404	22554	28148	21897	22147	18551	23119	19521	22669

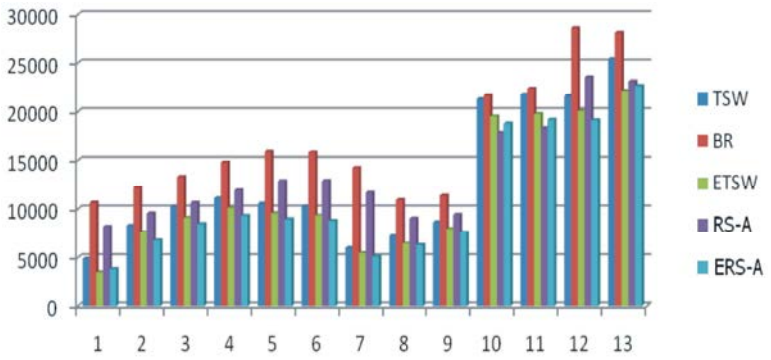

Fig. 5: The average number of attempts and comparisons of TSW, BR, ETSW, RS-A and ERS-A algorithms

do not exist in the text. Table 6 and Figure 5 show the results of comparing ERS-A algorithm with other algorithms. ERS-A algorithm has the minimum average number of comparisons and attempts among all other algorithms. The results are reasonable since ERS-A algorithm scan the text from both side simultaneously using two pattern window. In addition to that, the shift value is maximized by using RS-A shift algorithm. RS-A depends on using four consecutive characters, which means that the shifting values range from 1 up to m+4 characters.

CONCLUSION

In this paper, a new pattern matching algorithm ERS-A is implemented. ERS-A improves the performance by reducing the number of comparisons and the number of attempts needed to search for a particular pattern. The new algorithm has two aspects, the first one using two sliding windows to scan the text from both sides at the same time. The second aspect is related to using RS shifting algorithm to maximize the shift values; therefore an enhancements are made on two algorithms TSW and RS-A. While TSW uses BR bad character shift algorithm, ERS-A uses RS-A shifting algorithm. ERS-A can be used in many applications specially the ones related to Biological sequence such as DNA.

We evaluated ERS-A performance by using a text string and various set of patterns. In addition the ERS-A reduced the memory required in a preprocessing phase by using two one-dimensional arrays each of (m-3) length only. In future research, we intend to implement the ERS-A algorithm on real parallel processors to minimize the number of comparisons and attempts. Also we intend to implement the idea of the two sliding windows on other algorithms such as KMP and BM.

REFERENCES

1. Liu Z., X.Chen, J.Borneman and T. Jiang, 2005. A Fast Algorithm for Approximate String Matching on Gene Sequences A. Apostolico, M. Crochemore and K. Park, Springer-Verlag Berlin Heidelberg, pp: 79-90.
2. Bhukya, R. and D. Somayajulu, 2011. Multiple Pattern Matching Algorithm using Pair-count. IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 2, July 2011 ISSN (Online): pp. 1694-0814.
3. Bhukya, R. and D. Somayajulu, 2010. An Index based Forward Backward Multiple Pattern Matching Algorithm. World Academy of Science, Engineering and Technology, pp: 1513- 1521.
4. Senapati, K.K., G. Sahoo and S. Sahana, 2010. 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.
5. Itriq, M., A. Hudaib, A. Al-Anani, R. Al-Khalid and D. Suleiman, 2012. Enhanced Two Sliding Windows Algorithm For Pattern Matching (ETSW). Journal of American Science, 8(5): 607- 616.
6. Hudaib, A., R. Al-Khalid, D. Suleiman, M. Itriq and A. Al-Anani, 2008. A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW). Journal of Computer Science, 4(5): 393-401.
7. Senapati, K.K., S. Mal and G. Sahoo, 2012. RS-A Fast Pattern Matching Algorithm for Biological Sequences.International Journal of Engineering and Innovative Technology (IJEIT), 1(3): 116-118.
8. Sheik, S.S., Aggarwal, K. Sumit, Poddar, N. Anindya, Balakrishnan and K. Sekar, 2004. A FAST Pattern Matching Algorithm. Journal of Chemical Information and Computer Sciences, 44(4): 1251-1256.

9. Salmela, L., J. Tarhio and P. Kalsi, 2010. Approximate Boyer-Moore String Matching for Small Alphabets. 58(3): 591-609(19)
10. Berry, T. and S. Ravindran, 2001. 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, pp: 16-26.
11. Vangipuram, R.K., S.J. Sandeep and A. Reddy, 2011. Text Segmentation Based Pattern Search Algorithm. International Journal of Wisdom Based Computing, 1(3).
12. Boyer, R.S. and J.S. Moore, 1977. A Fast String Searching Algorithm. Commun. ACM, 20: 762-772.
13. Pendlimarri, D. and P.B.B. Petlu, 2010. Novel Pattern Matching Algorithm for Single Pattern Matching. International Journal on Computer Science and Engineering, 02(08): 2698-2704.
14. Franek, F., C.G. Jennings and W.F. Smyth, 2007. A simple fast hybrid pattern-matching algorithm. Journal of Discrete Algorithms, 5(4): 682-695.
15. Yang Wang, 2009. On the shift-table in Boyer-Moore's String Matching Algorithm. JDCTA; 3(4): pp. 10-20, doi: 10.4156/jdcta.
16. Knuth, D.E., J.H. Morris and V.R. Pratt, 1977. Fast pattern matching in strings. SIAM J. Comput., 6(2): 323-350.
17. Faro, S. and M.O. K¨ulekci‡ 2012. Fast Packed String Matching for Short Patterns.arXiv:1209.6449v1 [cs.IR] 28 Sep 2012.
18. Chao, Y., 2012. An Improved BM Pattern Matching Algorithm in Intrusion Detection System. Applied Mechanics and Materials 148-149: 1145-1148.
19. El Emary, I.M.M. and M.S.M. Jaber, 2008. A New Approach for Solving String Matching Problem through Splitting the Unchangeable Text. World Applied Sciences Journal, 4(5): 626-633.
20. Johnson, M., 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, pp: 136-143.
21. Calgary Corpus available at: ftp:// ftp.cpsc.ucalgary .ca/pub/projects/text.compression.corpus/