Journal of Software Engineering and Applications, 2015, 8, 143-153
Published Online March 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.83015

Enhancing ERS-A Algorithm for Pattern Matching (EERS-A)

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

[1]Department of Business Information Technology, King Abdullah II School for Information Technology, The University of Jordan, Amman, Jordan
[2]Department of Computer Information Systems, King Abdullah II School for Information Technology, The University of Jordan, Amman, Jordan
Email: dima.suleiman@ju.edu.jo, m.itriq@ju.edu.jo, a.anani@ju.edu.jo, r.khalid@ju.edu.jo, ahudaib@ju.edu.jo

Received 25 February 2015; accepted 18 March 2015; published 20 March 2015

Abstract

Pattern matching is a very important topic in computer science. It has been used in various applications such as information retrieval, virus scanning, DNA sequence analysis, data mining, machine learning, network security and pattern recognition. This paper has presented a new pattern matching algorithm—Enhanced ERS-A, which is an improvement over ERS-S algorithm. In ERS-A, two sliding windows are used to scan the text from the left and the right simultaneously. The proposed algorithm also scans the text from the left and the right simultaneously as well as making comparisons with the pattern from both sides simultaneously. The comparisons done between the text and the pattern are done from both sides in parallel. The shift technique used in the Enhanced ERS-A is the four consecutive characters in the text immediately following the pattern window. The experimental results show that the Enhanced ERS-A has enhanced the process of pattern matching by reducing the number of comparisons performed.

Keywords

Pattern Matching, Enhanced Two Sliding Windows Algorithm, RS-A Fast Pattern Matching Algorithm, Enhanced RS-A

1. Introduction

Many applications use pattern matching algorithms such as search engines, anti-virus and biological applications such as DNA [1]-[5].

Most of the algorithms have been implemented in order to make the searching process faster and more efficient; this can be achieved either by reducing the number of attempts, comparisons or by both. Some of algo-

How to cite this paper: 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

rithms made enhancements on preprocessing phase [6]-[9] and searching phase [10]-[12], and others modified the shifting technique [5] [9] [13]-[15] used in a case of a mismatch between the pattern and the text.

Searching process differs from one algorithm to another: some algorithms scan the text from one side only, either from left [15] or from right [5]; other algorithms use two sliding windows to scan the text from the left and the right sides simultaneously [10]-[13].

Some algorithms made modification on the shifting values. The shift value is the amount of shift that the sliding window will move in a case of a mismatch between the text and the window; shift values depend on the number of consecutive characters immediately after the sliding window. Some algorithms use one consecutive character [16] [17]; others use two [10] [12] [15] and few use three [13].

In this paper, we propose a new pattern matching algorithm—Enhanced ERS-A. The algorithm uses two sliding windows such as TSW [12], ETSW [10], ERS-A [11] and EBR [13] algorithms. It also uses four consecutive characters to compute the shift values such as RS-A [5] and ERS-A [11] algorithms. In addition to that, it uses the same comparison technique between the pattern and the text that was used in ETSW [10].

This paper compares between the new algorithms of EERS-A, ETSW [10] and ERS-A [11]. The results showed that the new algorithm is better than others as explained in Section 5. The reminder of this paper is organized as follows: Section 2 consists of related works; Section 3 explains EERS-A algorithm; Section 4 covers the analysis. Finally, the conclusion and the future work are presented in Section 6.

2. Related Works

Pattern matching algorithms were needed in many applications [2] [14] [18]-[20]. Some algorithms made enhancement in a memory used in preprocessing phase, while others try to make the searching process faster and more efficient [8] [9] [21].

The Berry-Ravindran algorithm (BR) [15] made enhancement in Boyer Moore algorithm [16]. In a case of a mismatch between the text and the pattern window Boyer uses one consecutive character in the text immediately to the right of the pattern window to determine amount of shift the window must move, on the other hand Berry-Ravindran uses two consecutive characters.

Many enhancements made in Berry-Ravindran algorithm (BR) [15], some algorithms changed the searching process but on the other hand they used the same bad character shift values, bad character shift depends on using two consecutive characters immediately that follow the text such as TSW [12] and ETSW [10]. Others made enhancement on the shift values such as RS-A [5] and ERS-A [11] these two algorithms changed the shift values by using four consecutive character instead of two, also EBR [13] algorithm made modifications on Berry-Ravindran bad character shift by depending on using three consecutive characters.

Two Sliding Windows algorithm TSW [12] used the same preprocessing technique used in BR [15] but made enhancements on the searching phase. In a case of a mismatch between the pattern and text, it uses two consecutive characters to determine the amount of shift the pattern window must slide which is the same technique used in BR [15], but instead of using one pattern window to scan the text as in BR [15] it uses two sliding windows that scan it in parallel.

Searching process become faster in Enhanced Two Sliding Window algorithm (ETSW) [10]. ETSW [10] made enhancements on TSW [12] by minimizing the number of comparisons needed but it doesn't make any changes on the number of attempts. The reason for this is that two algorithms used the same preprocessing techniques and also used the two sliding window, the only difference between them is related to the idea that TSW [12] compares the text with the pattern from one side of the pattern while ETSW [10] compare the text with the pattern from the both sides of the pattern at the same time, the best time complexity is $O(m/2)$ and the worst case time complexity is $O(((n/2-m/2+1))(m/2))$. The preprocess time complexity is $O(2(m-1))$.

RS-A [5] algorithm used only one window to search for a pattern p in a text t from the right side of the text. In order to make the searching process faster ERS-A [11] algorithms made enhancements on RS-A by using two windows instead of one. The left window aligned with the text from the left and in a case of a mismatch the window will be shifted to the right, and the right window aligned with text from the right and in a case of a mismatch the window will be shifted to the left. The two windows slide in parallel. The search will stop either when the pattern is found or in a case the pattern not found at all.

The EERS-A made enhancement on ERS-A [11], it uses the same preprocessing technique that depends on using four consecutive characters to determine the amount of shift, it also uses two sliding windows to scan the text from the left and right sides at the same time. Enhancements made on a comparison between the text and the

pattern, in this case it uses the same method used in ETSW [10], which make comparisons between the text and the pattern from both sides of the pattern at the same time. It is obvious from the results that there are no changes in average number of attempts in both ERS-A [11] and EERS-A, on the other hand there are a clear differences in average number of comparisons. The EERS-A is more efficient and faster.

3. The Enhanced ERS-A Algorithm

EERS-A algorithm improved the searching process by scanning the pattern as well as the text from both sides simultaneously. EERS-A used two sliding windows to scan the text from both sides at the same time; also comparisons between the pattern and the text happened from both sides of the pattern.

EERS-A algorithm used the same searching technique used in ERS-A, they uses two sliding windows to search for a pattern *p* in a text *t*. Two sliding windows scan the text from the left and right side at the same time and in a case of a mismatch both algorithms will use RS-A bad character shift function [5]. The searching process will stop either when a pattern is found or in a case a pattern is not found in the text at all.

The main difference between EERS-A and ERS-A [11] algorithms is that in ERS-A [11] comparisons between the pattern and the text happened only from the left side of the pattern while in EERS-A comparisons done from left and right sides of the pattern at the same time.

4. Pre-Processing Phase

Two arrays are generated in this phase *nextl* and *nextr*, each array is one-dimensional array. The values of the *nextr* array are calculated according to RS-A algorithm [5]. Initially the indexes of the four consecutive characters of the text after aligning it with 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 in a case of a mismatch from the right side of the text as in Equation (1).

$$\text{Right shift value}\left[\text{a,b,c,d}\right]=\min\begin{Bmatrix} m+3 & \text{if } p[m-1]=\text{a} \\ m+2 & \text{if } p[m-1]=\text{b} \\ m+1 & \text{if } p[m-1]=\text{c} \\ 1 & \text{if } p[0]=\text{d} \\ 2 & \text{if } p[0][1]=\text{cd} \\ 3 & \text{if } p[0][1][2]=\text{bcd} \\ m-\left((m-4)-i\right) & \text{if } p[i][i+1][i+2][i+3]=\text{abcd} \\ m+4 & \text{otherwise} \end{Bmatrix} \tag{1}$$

On the other hand, the values of the *nextl* array are calculated according ERS-A algorithm [11]. The shift values are calculated by using four consecutive text characters a, b, c and d which are aligned immediately to the right of the left sliding window. Initially, the indexes of the four consecutive characters in the text string needed to search the text from the left side are $(m+1)$, $(m+2)$, $(m+3)$ and $(m+4)$ for a, b, c and d respectively as in Equation (2).

$$\text{Left shift value}\left[\text{a,b,c,d}\right]=\min\begin{Bmatrix} 1 & \text{if } p[m-1]=\text{a} \\ 2 & \text{if } p[m-2][m-1]=\text{ab} \\ 3 & \text{if } p[m-3]\ [m-2][m-1]=\text{abc} \\ m+1 & \text{if } p[0]=\text{b} \\ m+2 & \text{if } p[0]=\text{c} \\ m+3 & \text{if } p[0]=\text{d} \\ m-i & \text{if } p[i][i+1][i+2][i+3]=\text{abcd} \\ m+4 & \text{otherwise} \end{Bmatrix} \tag{2}$$

The pre-processing phase is the same in both ERS-A and EERS-A algorithms but the searching phase is enhanced in EERS-A.

5. Searching Phase

In EERS-A algorithm the searching process started by aligning two windows with the text, the left window aligned with the beginning of the text and the right window aligned with the end of the text. In case of a mismatch the proposed window will be shifted according to next array values calculated in a preprocessing phase; the left window will be shifted to the right and the right window will be shifted to left.

Four pointers will be used in a comparison process between the text and the pattern, two for each window. The left window uses the L and temp_newlindex pointers while the right window uses the R and temp_newrindex pointers.

Comparisons between the text and the pattern from both sides is the same, in each window the first character of the pattern is compared with the corresponding text character and at the same time the last character of the pattern is compared with the corresponding character in the text. Each window uses different pointers as explained below:

6. Left_Window Search Process

After aligning the left window with the text from the left, comparisons between the pattern and the text will be done using two pointers, L and temp_newlindex, L pointer points at the last character of the pattern and the temp_newlindex points at the first character of the pattern, comparisons are made between the pointers and the corresponding characters of the text.

If a mismatch occurs in any pointer a shift will occurs according to the ERS-A bad character algorithm.

In case of a match the two pointers will move.

The L pointer will move to the left and the temp_newlindex will move to the right. A movement of pointers in a case of a match will stop either when two pointers reach the middle of the pattern or when the L pointer is less than or equal the temp_newlindex, in either case the pattern is found.

7. Right_Window Search Process

After aligning the right window with the text from the right, comparisons between the pattern and the text will be done using two pointers, R and temp_newrindex, R pointer points at the first character of the pattern and the temp_newrindex points at the last character of the pattern, comparisons are made between the pointers and the corresponding characters of the text.

If a mismatch occurs in any pointer a shift will occurs according to the ERS-A bad character algorithm. In case of a match the two pointers will move. The R pointer will move to the right and the temp_newrindex will move to the left. A movement of pointers in a case of a match will stop either when two pointers reach the middle of the pattern or when the R pointer is less than or equal the temp_newrindex, in either case the pattern is found.

The proposed EERS-A algorithm is explained in **Figure 1**.

8. Working Example

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

Given:

Pattern (P) = “ACCBCBAC”, $m = 8$,

Text (T) = “ABCDABADBACEDABACCBCBACABADBCEDABADDDBACCBABCBCBAB”, $n = 50$.

9. Pre-Processing Phase

Initially, $shiftl = shiftr = m + 4 = 12$.

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

To build the two next arrays (*nextl* and *nextr*), we take each four consecutive characters of the pattern and

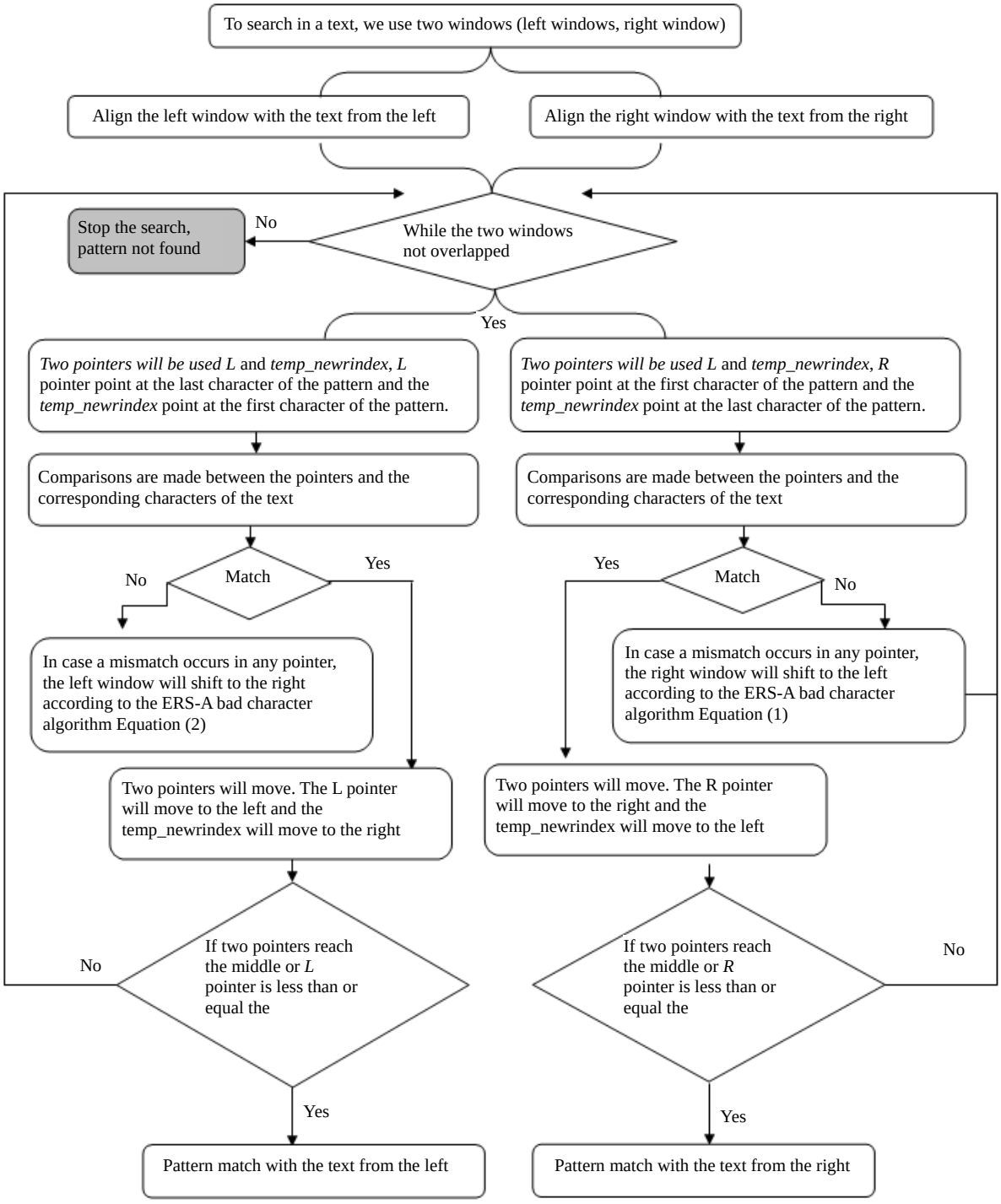

Figure 1. EERS-A flowchart.

Shift Values from the left

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

(a)

Shift Values from the right

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

(b)

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

give it an index starting from 0. For example for the pattern structure ACCBCBAC, the consecutive characters ACCB, CCBC, CBCB, BCBA and CBAC are given the indexes 0, 1, 2, 3 and 4 respectively.

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

10. Searching Phase

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

10.1. First Attempt

In the first attempt (see **Figure 3(a)**), we align the left sliding window with the text from the left. In this case, a comparison is made between the text character located at index 0 (character A) with the leftmost character in the pattern (character A) although a match occurs, the total result is a mismatch since at the same time a comparison must be made between the text character at index 7 (character D) with the rightmost character in the pattern (character C); 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 amount of shift (*shiftl*) we have to do the following:

Since if $p[m-3][m-2][m-1] = abc$; BAC = BAC then according to a preprocessing algorithm the shift value will be 3.

10.2. Second Attempt

In the second attempt (see **Figure 3(b)**), we align the right sliding window with the text from the right. In this case, a match occurs between text character at index 42 (A) and left most character of the pattern character A but a mismatch occurs between the last text character B with the last pattern character C so as a total result,

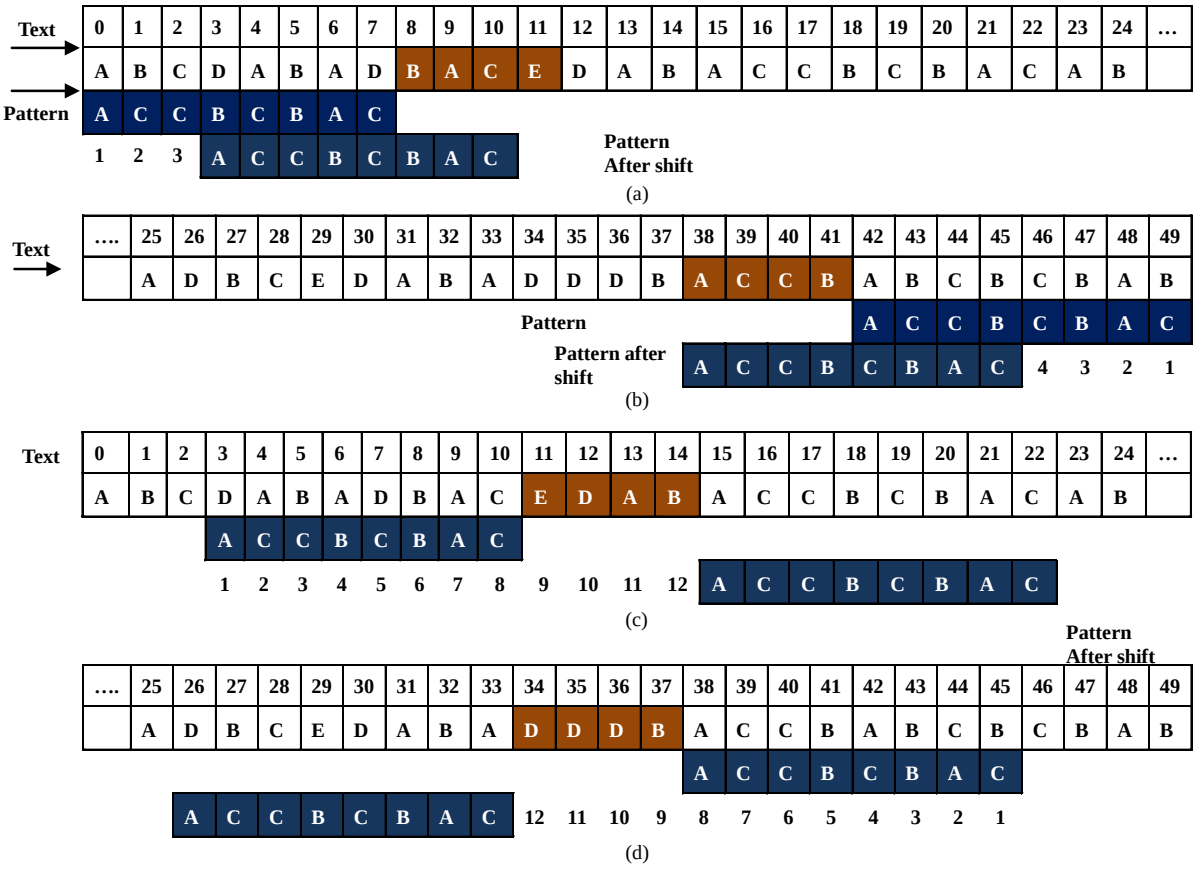

Figure 3. Working example.

there is a mismatch; therefore we take the four consecutive characters from the text at index 38, 39, 40 and 41 which are A, C, C and B respectively. To determine the amount of shift (*shiftr*), we have to do the following:

a) We find the index of ACCB in the pattern which is 0.

b) Since we search the text from the right side we use *nextr* array, and *shiftr* = *nextr*[0] = 4; therefore the window will be shifted to the left 4 steps.

10.3. Third Attempt

In the third attempt (see **Figure 3(c)**), a mismatch occurs from the left between the text character at index 3 (character D) and the leftmost character in the pattern (character A) while there is a match between the text character at index 10 (character C) and the rightmost character in the pattern (character C); therefore we take the four consecutive characters from the text at indexes 11, 12, 13 and 14 which are E, D, A and B respectively, since EDAB is not found in the pattern, so the window will be shifted to the right 12 steps.

10.4. Fourth Attempt

In the fourth attempt (see **Figure 3(d)**), a comparison is made between the text character located at index 38 (character A) with the leftmost character in the pattern (character A) at the same time a comparison must be made between the text character at index 45 (character B) with the rightmost character in the pattern (character C) since there is a mismatch we take the four consecutive characters of the text at index 34, 35, 36 and 37 which are D, D, D and A respectively; since DDDB is not found in the pattern, so the window will be shifted to the left 12 steps.

10.5. Fifth Attempt

We align the left most character of the pattern P[0] with T[15]. A comparison between the pattern and the text characters leads to a complete match at index 15. In this case, the occurrence of the pattern is found using the left window.

11. Analysis

Preposition 1: The space complexity is $O\left(2\left(m-3\right)\right)$ where m is the pattern length.

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

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

Proof: The worst occurs when a match between the text and patterns occurs in all characters except a character at index $m/2$ in the pattern, and if at the same time the shift values in case of a mismatch equal to 1.

Lemma 2: The best case time complexity is $O\left(m/2\right)$.

Proof: The best case occurs when the pattern is found in the leftmost or the rightmost sides of the text.

Lemma 3: The average case time complexity is $O\left(n/\left(2*\left(m/2+4\right)\right)\right)$.

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 $O\left(\left\lceil n/\left(2*\left(m/2+4\right)\right)\right\rceil\right)$.

12. Experimental Results and Discussion

Several experiments have been done using Book 1 from the Calgary corpus [22] to be the text in most of searching algorithms. Book 1 consists of 141,274 words (752,149 characters). Book 1 consists of 141,274 words (752,149 characters). Patterns of different lengths are also taken from Book 1.

Table 1 shows the results of comparing the algorithms ETSW, ERS-A and EERS-A.

In **Table 1**, the first column related to the pattern length; second column is the number of words in a certain length. It can be clearly seen that the number of attempts in EERS-A and ERS-A are the same since the two algorithms use the same shifting techniques in a case of a mismatch, and according to the results it's clear that the number of comparisons in EERS-A are better than all other algorithms. For example, as shown in **Table 1**, 2896 words of length 6, the average number of comparisons in ETSW is 7633, in ERS-A is 6750 and in the new algorithms is 6212, which is the minimum value among the others values.

ETSW algorithm and EERS-A algorithm use the same comparison technique, they use two sliding windows to scan the text, and in addition to that they use the same technique when comparing the text with the pattern where comparisons happened from both sides of the pattern simultaneously. It is clear that the number of comparisons and attempts in EERS-A algorithm are better than ERS-A algorithm and this refers to using different shifting technique. While ETSW uses Berry-Ravindran (BR) [15] bad character shift values which uses two consecutive characters, EERS-A uses ERS-A [11] bad character shift values that uses four consecutive characters.

Table 2 shows the average number of attempts and comparisons for 100 words taken from the right side of Book1. It is clear that EERS-A is the best among all others according to the number of comparisons and the same results can be seen in **Table 3** and **Table 4**.

Table 5 and **Figure 4** show the average number of comparisons needed to search for patterns with different lengths. The results show that EERS-A finds the patterns with minimum number of comparisons compared with

Table 1. The average number of attempts and comparisons of ETSW, ERS-A and EERS-A algorithms.

Pattern length	Number of words	ETSW		ERS-A		EERS-A	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	4535	4456	3549	3533	3880	3533	2813
6	2896	7596	7633	6166	6750	6166	6212
7	1988	9341	9118	7737	8506	7737	7636
8	1167	10056	10115	8451	9319	8451	8525
9	681	9538	9590	8106	8957	8106	8171
10	382	9283	9339	7970	8830	7970	8042
11	191	5451	5482	4701	5146	4701	4742
12	69	6384	6433	5589	6286	5589	5653
13	55	7947	7986	6955	7587	6955	7004
14	139	19437	19535	17115	18776	17115	17242
15	32	19682	19782	17385	19198	17385	17519
16	10	20029	20092	17722	19147	17722	17807
17	3	21897	22147	19521	22669	19521	19855

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	ETSW		ERS-A		EERS-A	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	185	187	146	163	146	148
6	100	227	230	182	205	182	186
7	100	347	351	286	324	286	291
8	100	504	510	424	476	424	431
9	100	670	677	571	640	571	579
10	100	1160	1170	999	1117	999	1011
11	100	622	628	529	597	529	536
12	100	865	878	756	860	756	774

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	ETSW		ERS-A		EERS-A	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	13965	11618	11038	11970	11038	10875
6	100	16682	16771	13536	14870	13536	13648
7	100	27267	26242	22607	24971	22607	22179
8	100	27830	28015	23385	25976	23385	23617
9	100	33929	34069	28764	31541	28764	28943
10	100	29676	29845	25471	28193	25471	25689
11	100	23195	23242	19886	21119	19886	19946
12	100	26806	27009	23484	26507	23484	23761

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	ETSW		ERS-A		EERS-A	
		Attempts	Comparisons	Attempts	Comparisons	Attempts	Comparisons
5	100	271	270	216	238	216	218
6	100	364	368	295	326	295	299
7	100	402	405	333	372	333	337
8	100	536	541	451	499	451	457
9	100	776	783	660	730	660	668
10	100	1579	1593	1361	1517	1361	1378
11	100	619	624	531	573	531	537
12	100	1667	1685	1459	1641	1459	1480

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

Pattern length	Number of words	TSW	ETSW	RS-A	ERS-A	EERS-A
		Comparisons	Comparisons	Comparisons	Comparisons	Comparisons
5	4535	4896	3549	8191	3880	2813
6	2896	8311	7633	9556	6750	6212
7	1988	10263	9118	10638	8506	7636
8	1167	11087	10115	11922	9319	8525
9	681	10538	9590	12911	8957	8171
10	382	10272	9339	12927	8830	8042
11	191	5967	5482	11672	5146	4742
12	69	7168	6433	9030	6286	5653
13	55	8673	7986	9422	7587	7004
14	139	21319	19535	17845	18776	17242
15	32	21739	19782	18318	19198	17519
16	10	21596	20092	23531	19147	17807
17	3	25404	22147	23119	22669	19855

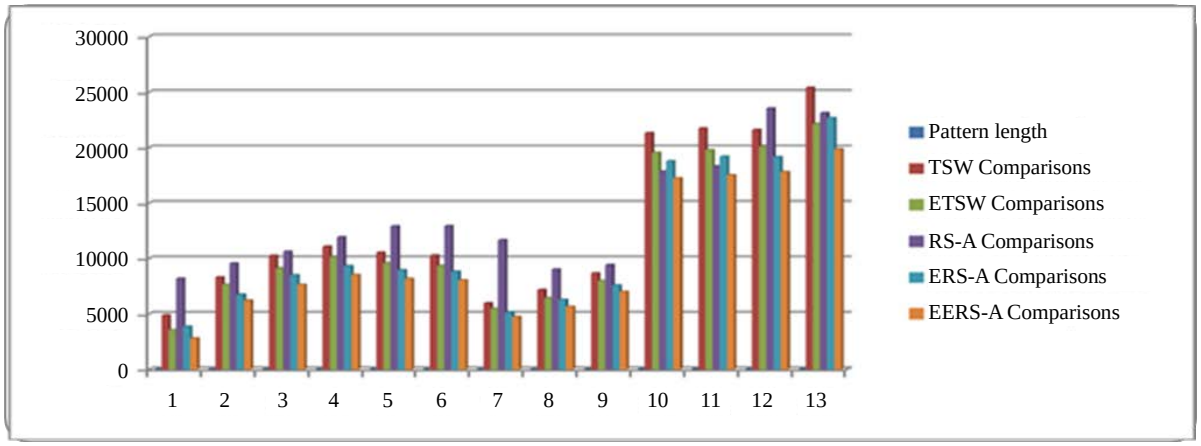

Figure 4. The average number of comparisons of TSW, ETSW, RS-A and ERS-A, EERS-A algorithms.

other algorithms such as TSW, ETSW, RSA and ERS-A. For example, it took EERS-A 8042 comparisons to locate the pattern of length 10 while it took TSW, ETSW, RS-A and ERS-A 10272, 9339, 12927 and 8830 comparisons respectively for the same pattern length. These results show that EERS-A is the best algorithm compared to others.

13. Conclusions and Future Work

In this paper, we presented a new pattern matching algorithm—Enhanced ERS-A algorithm. The Enhanced ERS-A algorithm enhances the ERS-A's process by utilizing the idea of the two sliding windows and by making comparisons with the pattern from both sides simultaneously. The comparisons done between the text and the pattern are done from both sides in parallel. This process gives the proposed algorithm a preference over the ERS-A algorithm.

The Enhanced ERS-A algorithm utilizes the idea of RS shifting algorithm to maximize the shift values. To assess the performance of the proposed algorithm, we considered ETSW and ERS-A algorithms for comparison with the proposed algorithm. The experimental results show that the Enhanced ERS-A shows better results in the number of comparisons performed.

References

[1] El Emary, I.M.M. and Jaber, M.S.M. (2008) A New Approach for Solving String Matching Problem through Splitting the Unchangeable Text. *World Applied Sciences Journal*, **4**, 626-633.

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

[3] 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.

[4] 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.

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

[6] Bhukya, R. and Somayajulu, D. (2011) Multiple Pattern Matching Algorithm Using Pair-Count. *IJCSI International Journal of Computer Science Issues*, **8**, 1694-0814.

[7] 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

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

[9] Salmela, L., Tarhio, J. and Kalsi, P. (2010) Approximate Boyer-Moore String Matching for Small Alphabets. *Algorithmica*, **58**, 591- 609.

[10] 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.

[11] 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.

[12] 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

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

[14] 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.

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

[16] 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

[17] 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.

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

[19] Bhandaru, J. and Kumar, A. (2014) A Survey of Fast Hybrid String Matching Algorithms. *International Journal of Emerging Sciences*, **4**, 24-37.

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

[21] Hlayel Abdallah, A. and Hnaif Adnan, A. (2014) A New Exact Pattern Matching Algorithm (WEMA). *Journal of Applied Sciences*, **14**, 193-196.

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