Improved-Bidirectional Exact Pattern Matching

Iftikhar Hussain, Syed Zaki Hassan Kazmi, Israr Ali Khan, Rashid Mehmood

Abstract—In this research, we present an improved version of Bidirectional (BD) exact pattern matching (EPM) algorithm to solve the problem of exact pattern matching. Improved-Bidirectional (IBD) exact pattern matching algorithm introduced a new idea of scanning partial text window (PTW) as well with the pattern to take decision of moving pattern to the right of partial text window. IBD algorithm compares the characters of pattern to selected text window (STW) from both sides simultaneously as BD. The time complexity of preprocessing phase of IBD algorithm is *O(2m)* and searching phase takes *O(mn/2)*.

Index Terms—Algorithm, window sliding, scanning text window, string matching, exact pattern matching, Improved-Bidirectional, Bidirectional, boyer-moore

1 Introduction

EXACT pattern matching algorithms, are a dominant class of the string algorithms which aim to find one or all occurrences of string within a larger group of text string [1]. In exact pattern matching, pattern is fully compared with selected text window of text string.

In this paper, we proposed an Improved-Bidirectional (IBD) exact pattern matching algorithm based on window sliding method of exact pattern matching which will be helpful in various needs of pattern matching and searching.

Literature review of previous exact string matching algorithms used to complete this research. After the publications of Knuth-Morris-Pratt and Boyer-Moore exact pattern matching algorithms, so for there have hundreds of papers published related to exact pattern matching [19]. According to literature survey, all the authors focus to reduce the number of character comparisons as [8] and processing time as [14, 9, 10] in both worst/average cases. In this paper we compare Improved-Bidirectional algorithm's results with Bad Character Boyer-Moore, BM Horspool, Quick Search, Berry Ravindran, BM Smith, Raita and Bidirectional exact pattern matching algorithms which considered efficient in terms of number of character comparisons and attempts take to complete the processing of selected text.

In this paper, we present a brief literature review of some existing exact string matching algorithms. Section 2 describes the basic concept and working of Improved-Bidirectional algorithm with brief example. Then we compare the IBD algorithm with some existing algorithms in terms of their comparison order, preprocessing space complexity, preprocessing time complexity and searching time complexity (best, average and worst). In Section 3, we present results and discussions that compare the IBD algorithm with existing algorithms. Finally, in Section 4, we draw conclusions from the experiments.

1.1 Literature Survey

Brute force (BF) [1] or Naïve algorithm is the logical place to begin the review of exact string matching algorithms. It compares a given pattern with all substrings of the given text in any case of a complete match or a mismatch. It has no preprocessing phase and did not require extra space. The time complexity of the searching phase of brute force algorithm is *O(mn)*.

Knuth-Morris-Pratt (KMP) [2] algorithm is proposed in 1977 to speed up the procedure of exact pattern matching by improving the lengths of the shifts. It compares the characters from left to right of the pattern. In case of match or mismatch it uses the previous knowledge of comparisons to compute the next position of the pattern with the text. The time complexity of preprocessing phase of KMP is *O(m)* and of searching phase is *O(nm)*.

Boyer-Moore (BM) [3] algorithm published in 1977 and at that time it considered as the most efficient string matching algorithm. It performed character comparisons in reverse order from right to the left of the pattern and did not require the whole pattern to be searched in case of a mismatch. In case of a match or mismatch, it used two shifting rules to shift the pattern right. The time and space complexity of preprocessing phase is $O(m+|\sum|)$ and the worst case running time of searching phase is $O(nm + |\sum|)$. The best case of Boyer-Moore algorithm is *O(n/m)*.

Boyer-Moore Horspool (BMH) [9] did not use the shifting heuristics as Boyer-Moore algorithm used. It used only the occurrence heuristic to maximize the length of the shifts for text characters corresponding to right most character of the pattern. It's preprocessing time complexity is $O(m+|\sum|)$ and searching time complexity is *O(mn)*.

Quick Search (QS) [10] algorithm perform comparisons from left to right order, it's shifting criteria is by looking at one character right to the pattern and by applying bad character shifting rule. The worst case time complexity of QS is same as Horspool algorithm but it can take more steps in practice.

- ***Iftikhar Hussain**, lecturer at Faculty of Administrative Sciences, Kotli University of Azad Jammu and Kashmir, Muzaffarabad,AJK Pakistan. Cell No.-0092-3235356089. E-mail: iftikhar.iftikhar786@gmail.com*
- ***Syed Zaki Hassan Kazmi,** lecturer at Faculty of Sciences, Muzaffarabad University of Azad Jammu and Kashmir, Muzaffarabad,AJK Pakistan Cell No.-0092-3465881738. E-mail: zaki.mzd@gmail.com*
- ***Dr. Israr Ali Khan**, Chairman, Department of Mathematics ,FASK, University of Azad Jammu and Kashmir, Muzaffarabad,AJK Pakistan Cell No.-0092-3006091742. E-mail: israrkhan81@gmail.com*
- ***Rashid Mehmood**, lecturer at Faculty of Administrative Sciences, Kotli University of Azad Jammu and Kashmir, Muzaffarabad,AJK Pakistan. Cell No.-0092-3435634099. E-mail: rashid.mehmood@ajku.edu.pk*

International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
ISSN 2229-5518

Boyer-Moore Smith (MBS) [11] noticed that by computing the BMH shift, sometimes maximize the shifts than QS shifts. It uses the bad character shifting rule of BMH and QS bad character rule to shift the pattern. It's preprocessing time complexity is $O(m+|\sum|)$ and searching time complexity is $O(mn)$.

Turbo Boyer Moore (TBM) [14] is variation of the Boyer-Moore algorithm, which remembers the substring of the text string which matched with suffix of pattern during last comparisons. It does not compare the matched substring again; it just compares the other characters of the pattern with text string.

In Reverse Colussi (RC) [15] algorithm comparisons are done in specific order given by the preprocessed phase. The time complexity of preprocessing phase is $O(m2)$ and searching phase is $O(n)$.

Two Way algorithm (TW) [17] proposed by Crochemore and Rytter in 2002. The Two Way algorithm uses an idea related to the short maximal suffix of the pattern to calculate the shifting lengths of pattern in text string. The Two Way algorithm's time complexity with the short maximal suffix is $O(n)$.

Berry Ravindran (BR) [18] algorithm proposed by Berry and Ravindran in 1999, it performs shifts by using bad character shifting rule for two consecutive characters to the right of the partial text window of text string. The preprocessing time complexity is $O(m+(|\sum|)2)$ and the searching time complexity is $O(mn)$.

2 Improved-Bidirectional Exact Pattern Matching Algorithm

2.1 Basic Idea

Improved-Bidirectional exact pattern matching algorithm compares a given pattern character wise from both sides simultaneously as Bidirectional algorithm. The difference between BD and IBD is that, in case of mismatch or a complete match of the pattern, IBD scans pattern for the rightmost character of the partial text window and scan partial text window for the leftmost character of the pattern. If both characters found on equal shifts then align pattern to the position where the characters found otherwise, shift whole pattern to the right of the partial text window of text string. A complete match will be found when left pointer cross right pointer at middle of the pattern. The comparison order of pattern's characters with the characters of selected text window is shown in Figure 1.

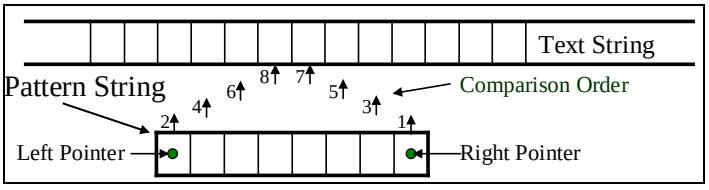


Figure 1: Comparison order of pattern's character with STW.

Improved-Bidirectional algorithm has two cases to shift the pattern to right of partial text window. Suppose $T[1\ldots n]$ is the text string and $P[1\ldots m]$ is the pattern and we compare pattern $P[1\ldots m]$ with the selected text window $T[i\ldots i+m-1]$ from both sides of the pattern simultaneously. If mismatch cause by any pointer either right or left at any position of the pattern then scan pattern $P[m-1\ldots 1]$ for the rightmost character of partial text window $T[i+m-1]$ and scan partial text window $T[i+1 \ldots i+m-1]$ for the leftmost character of the pattern $P[1]$;

Case 1: If rightmost character $T[i+m-1]$ of partial text window and leftmost character $P[1]$ of pattern are found in the pattern and selected text window respectively on equal distances then align pattern to the position where the characters found on equal shifts as in Figure 2.

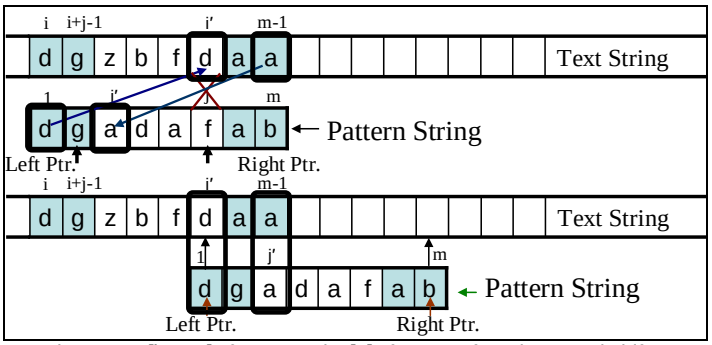


Figure 2: $T[i+m-1]$ of PTW and $P[1]$ of pattern found at equal shifts.

Case 2: Otherwise; If rightmost character of selected text window $T[i+m-1]$ and leftmost character of pattern $P[1]$ did not find in the pattern and selected text window on equal shifts. Or at least one character did not find in the pattern and selected text window then shift whole pattern to the right of the selected text window as shown in Figure 3.

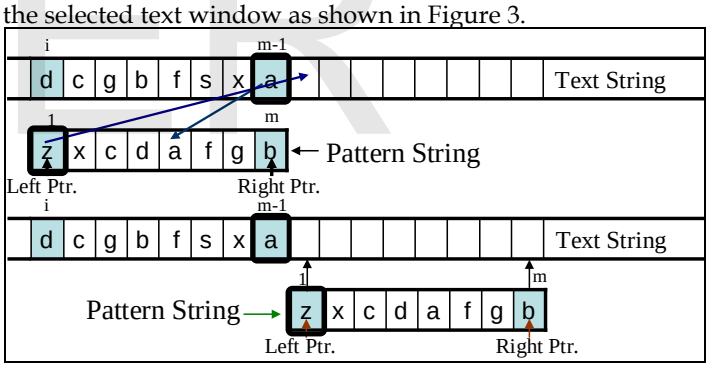


Figure 3: Maximum shift taken by Improved-Bidirectional algorithm.

2.2 Example of Improved-Bidirectional EPM

Here we present a short example of Improved-Bidirectional exact pattern matching algorithm, where T="A B C D G H D B D H A B A B B A D B H B B B D A B H A B A B D A B B A D B H" and P="A B D C G H D B".

Attempt 1:

A B C D G H D B D H A B A B B A D B H B B B … … …
A B D C G H D B

In first attempt Improved-Bidirectional EPM algorithm takes 6 comparisons, after a mismatch, the preprocessing phase scans for the rightmost character 'B' of partial text window in pattern and leftmost character 'A' of pattern in partial text window at equal shift's length. Here both characters did not find at same shifts, so case 2 should be applied to perform shift to the right of text window.

International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
ISSN 2229-5518

Attempt 2:

Mismatch occurred at first comparison by right pointer in second attempt, after scanning pattern and partial text window both characters found at same shifting positions. So, here Case 1 should be applied to perform shift by considering relevant characters.

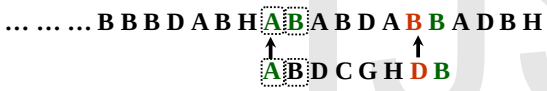
Attempt 3:

Mismatch caused by right pointer at third comparison in third attempt and after preprocessing phase both characters did not find at same shift's length so case 2 should be applied to move pattern to the right of the partial text window of text string.

Attempt 4:

Here Case 1 should be applied because both characters were found at same shifts in preprocessing phase. In fourth attempt, it takes two comparisons.

Attempt 5:



IBD takes four comparisons in fifth attempt.

By applying IBD algorithm, this example takes 16 comparisons in 5 attempts to complete the task.

2.3 Implementation of Improved-Bidirectional EPM

2.3.1 Preprocessing Phase

Preprocessing phase of Improved-Bidirectional EPM algorithm finds occurrences of rightmost character of partial text window and leftmost character of pattern, in pattern and partial text window respectively. This phase helps to take decision of moving pattern to the right of the partial text window of text string. The pseudocode of preprocessing phase in Figure 4 shows that the *preprocess()* function pass a pattern *P[1 … m]*, partial text window *T[i … i+m-1]* and starting index of partial text window *'i'* as input and return shifts's length to move pattern to the right of partial text window of text string. For loop, of preprocessing phase scans pattern from second last character to leftmost character of pattern by decrementing the indexes of pattern. It also scan partial text window from second to rightmost character of partial text window by incrementing index *'i'*.

Inside for loop, if both characters (rightmost of partial text window and leftmost of pattern) found in the pattern and partial text window respectively on equal shift's length then *preprocess()* returns the shift's distance. If both characters did not find in pattern and partial text window at same shift's length, then *preprocess ()* returns the maximum shift's length to move the whole pattern to the right of partial text window of text string.

```
Input:  Pattern P[1…m], partial text window T[i…i+m-1]
and starting index 'i' of partial text window of text string.
Output: Return shift's length.
1       k ← 1, shift ← P.length;
2       for j ← P.length-2 to 0  do
3             if P[j] = T[i+m-1] and T[i+1] = P[0] then
4                   shift ← k;     Break;
5             End if;
6             k ← k+1, i ← i+1;
7       End For;
8       return shift;
```

Figure 4: Pseudocode of preprocessing phase of Improved-Bidirectional.

2.3.2 Searching Phase

Character wise comparison will be performed between the pattern and the selected text window of the text string. Figure 5 shows the pseudocode of searching phase of Improved-Bidirectional exact pattern matching algorithm in which it takes a text string of length *'n'* and a pattern of length 'm' as input and display one or all occurences of pattern from text string as output. The external while loop is used to shift the pattern to the right of each partial text window whose index is calculated by adding shift's length returned by *preprocess ()* (described in preprocessing phase) to the index of previous partial text window of text string.

```
Input:  Text string 'T' of length 'n' and Pattern 'P' of
length 'm'.
Output: All occurrences of pattern in text string.
1   n ← T.length, m ← P.length;
2   i ← 0, j ← 0;
3   while i ≤ n-m do
4       left ← 0, right ← m-1;
5       while j < (m+1)/2  do
6           if P[right] = T[i+right] and P[left] = T[i+left]
7               if j = (m+1)/2 -1 then
8                   "Pattern matched at:", i;
9                   i ← i + preprocess (P, T[i…i+m-1], i);
10              left ← left+1;
11              right ← right-1;
12          Else begin
13              i ← i + preprocess (P, T[i…i+m-1], i);
14              Break Inner-While;
```

Figure 5: Pseudocode of searching phase of Improved-Bidirectional.

Two pointers (left and right) are used to compare pattern with the selected text window within second while loop. A complete match will be found, if left pointer cross right pointer at middle of the pattern then *preprocess ()* function will be called to calculate the length of the shift. Else, if mismatch caused by left or right pointer, then *preprocess ()* function will be executed to calculate the shift's length to move pattern to right of partial text window where next attempt will be performed.

International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
ISSN 2229-5518

2.4 Analysis of Improved-Bidirectional EPM

2.4.1 Analysis of Preprocessing Phase

The worst case time complexity of preprocessing phase of Improved-Bidirectional EPM algorithm is *O(m)*, because only one loop is used to scan the pattern and partial text window to find the rightmost character of partial text window and leftmost character of pattern.

2.4.2 Analysis of Searching Phase

The inner while loop of searching phase of Improved-Bidirectional EPM algorithm runs at most '*m/2*' times so, its worst case time-complexity is *O(m/2)* because two pointers are used. The worst case time-complexity to shift pattern to right of the text is *O(n)* because the external while loop runs 'n' times at most. The total time complexity of searching phase is *O(m/2).O(n)*, because the inner loop runs within external while loop. It can be written as *O(mn/2)*.

Improved-Bidirectional EPM algorithm requires *O(m)* extra memory space in worst case to execute in addition with the text and pattern string.

TABLE 1
COMPARISON OF IMPROVED-BIDIRECTIONAL EPM WITH EXISTING ALGORITHMS

Algos.	Preproces. phase	Searching phase	Extra Space	Comparison Order
BF	---	O(mn)	---	Left to right
KMP	O(m)	O(n)	O(m)	Left to right
BM	O(m+|∑|)	O(mn)	O(m+|∑|)	Right to left
BMH	O(m+|∑|)	O(mn)	O(|∑|)	1st right then left to right
QS	O(m+|∑|)	O(mn)	O(|∑|)	Left to right
BMS	O(m+|∑|)	O(mn)	O(|∑|)	Left to right
TBM	O(m+|∑|)	O(n)	O(m+|∑|)	Right to left
TW	---	O(n)	O(n+n)	Middle-right middle –left
BR	O(m+(|∑|)²)	O(mn)	O(m+(|∑|)²)	Left to right
RC	O(m²)	O(n)	O(m+|∑|)	Specific order
BD	O(m)	O(mn)/2	O(m)	Both sides simultaneously
IBD	O(m)	O(mn)/2	O(m)	Both sides simultaneously

Table 1 show the worst case time and space complexities of Improved-Bidirectional EPM algorithm with some of existing algorithms asymptotically. The comparison order is also shown in the table. The searching phase of Improved-Bidirectional EPM algorithm takes *O(mn/2)* time to execute which is considered efficient than existing algorithms. The comparison order of Improved-Bidirectional EPM algorithm is from both sides of the pattern simultaneously.

3 RESULTS AND DISCUSSIONS

The efficiency of Improved-Bidirectional exact pattern matching algorithm is measured and compared with existing; Bad Character Boyer-Moore, BM Horspool, Quick Search, Berry Ravindran, BM Smith, Raita and Bidirectional exact pattern matching algorithms. Improved-Bidirectional algorithm is compared with existing algorithms by using characters compare-base and attempts/shifts-base comparison. A text string of length 60,000 of four *{A, C, G, T}* random characters and part of text of different lengths *{6, 12, 18, 24, 30, 36, 42, 48, 54, and 60}* is used as pattern for the experiments. The experiments calculate the number of characters comparison and attempts/shifts each algorithm takes to perform the task; the results are shown by using graphs in figs. 6 and 7.

3.1 Attempts Base Comparison

Total numbers of attempts taken by each algorithm using different pattern's lengths are shown in graphical form in Figure 6. As results in the graph shows that Improved-Bidirectional algorithm took minimum shifts as compare to other Naïve, Not So Naive, Bad Character Boyer-Moore, BM Horspool, Quick Search, Berry Ravindran, BM Smith and Raita algorithms. Results showed that using short pattern's length shifts of Improved-Bidirectional EPM algorithm are closer to other algorithms but when pattern's length increased Improved-Bidirectional exact pattern matching algorithm becomes more and more efficient.

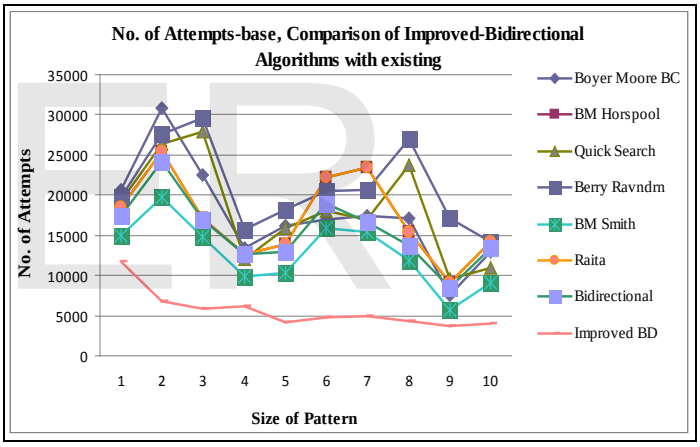


Figure 6: Shift Base Comparison of IBD with existing algorithms.

3.2 No. of Characters compare base Comparison

Total numbers of characters comparisons taken by each algorithm using different pattern lengths are shown graphically in Figure 7.

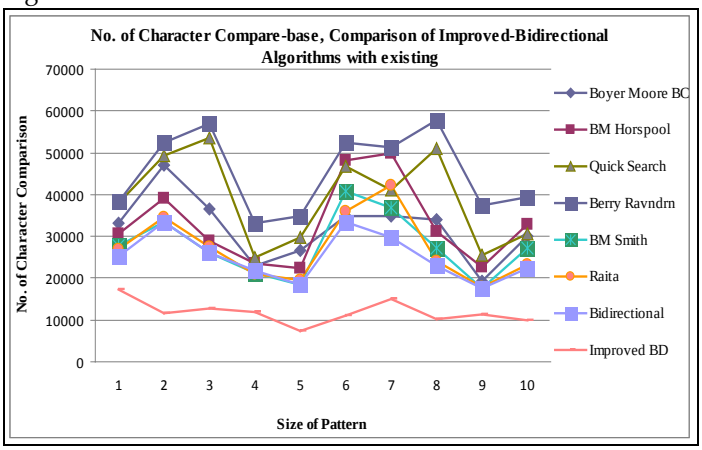


Figure 7: No. of characters compare base Comparison.

International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
ISSN 2229-5518

Results in the Figure shows that Improved-Bidirectional EPM algorithm takes less number of character comparisons than the existing algorithms when the pattern's length short as well as long.

According to both comparing criterion Improved-Bidirectional EPM exact pattern matching algorithm is quite efficient than the existing algorithms.

4 Conclusion

This research presents a new version of Bidirectional exact pattern matching algorithm. The basic idea of Improved-Bidirectional EPM algorithm is, it scans partial text window of the text string for leftmost character of pattern and pattern for the rightmost character of the partial text window. Improved-Bidirectional EPM algorithm compares a given pattern character wise with selected text window from both sides simultaneously as Bidirectional algorithm does. The worst case time-complexity of Improved-Bidirectional EPM algorithm is *O(mn/2)* in searching phase and *O(m)* in preprocessing phase. Comparison results show that the Improved-Bidirectional EPM algorithm is quite efficient than the existing algorithms, when the pattern length is short as well as on long pattern's lengths. There is still need to improve the shift decisions of preprocessing phase of exact pattern matching problem, to move pattern to the right of selected text window with long distance.

References

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L., *Introduction to Algorithms*, Chapter 34, MIT Press, 1990, pp 853-885.

[2] Knuth, D., Morris, J. H., Pratt, V., "Fast pattern matching in strings," SIAM Journal on Computing, Vol. 6, No. 2, doi: 10.1137/0206024, 1977, pp.323–350.

[3] R.S. Boyer, J.S. Moore, "A fast string searching algorithm," *Communication of the ACM*, Vol. 20, No. 10, 1977, pp.762–772.

[4] Rami H. Mansi, and Jehad Q. Odeh, "On Improving the Naïve String Matching Algorithm," *Asian Journal of Information Technology*, Vol. 8, No. 1, ISSN 1682-3915, 2009, pp. 14-23.

[5] Ziad A.A. Alqadi, Musbah Aqel, & Ibrahiem M. M. El Emary, "Multiple Skip Multiple Pattern Matching Algorithm," *IAENG International Journal of Computer Science*, Vol. 34, No. 2, IJCS_34_2_03, 2007.

[6] Ababneh Mohammad, Oqeili Saleh and Rawan A. Abdeen, "Occurrences Algorithm for String Searching Based on Brute-force Algorithm," *Journal of Computer Science*, Vol. 2, No. 1, ISSN 1549-3636, 2006, pp.82-85.

[7] A. Apostolic and R. Giancarlo, "The Boyer-Moore-Galil string searching strategies revisited," *SIAM J. Computer*. Vol. 15, No. l, 1986, pp.98–105.

[8] L. Colussi, Z. Galil, and R. Giancarlo, 'On the exact complexity of string matching,' *31st Symposium an Foundations of Computer Science* I, IEEE (October 22-24 1990), pp.135–143.

[9] R. N. Horspool, "Practical fast searching in strings," *Software – Practice and Experience,* Vol. 10, No. 3, 1980, 501–506.

[10] Sunday, D.M., "A very fast substring search algorithm," *Communications of the ACM*, Vol. 33, No. 8, 1990, pp. 132-142.

[11] Smith, P.D., "Experiments with a very fast substring search algorithm," *Software-Practice and Experience*, Vol. 21, No. 10, pp.1065-1074.

[12] Karp, R.M., Rabin, M.O., "Efficient randomized pattern matching algorithms," *IBM Journal on Research Development*, Vol. 31, No. 2, 1987, pp. 249-260.

[13] Apostolico, A. Crochemore, M., "Optimal canonization of all substrings of a string," *Information and Computation*, Vol. 95, No. 1, 1991, pp.76-95.

[14] Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski. W., Rytter, W., "Speeding up two string matching algorithms," *Algorithmica*, Vol. 12, No. 4/5, 1994, pp.247-267.

[15] Colussi, L., "Fastest pattern matching in strings," *Journal of Algorithms*, Vol. 16, No. 2, 1994, pp.163-189.

[16] Hume, A., Sunday, D. M., "Fast string searching," *Software Practice & Experience*, Vol. 21, No. 11, 1991, pp.1221-1248.

[17] Crochemore, M. and Rytter, W., "Jewels of Stringology," *World Scientific, Singapore*, 2002.

[18] Berry, T. Ravindran, S., "A fast string matching algorithm and experimental results, in proceeding of the Prague Stringology," *Club Workshop-99*, Collaborative report DC-99-5, Czech Technical University, Prague, Czech Republic, 1999, pp.16-26.

[19] Frantisek Franek, Christopher G. Jennings, W. F. Smyth, "A simple fast hybrid matching algorithm," *Journal of Discrete Algorithms*, Vol. 5, 2007, pp. 682-695.

[20] Iftikhar Hussain, Muhammad Zubair, Jamil Ahmed and Junaid Zaffar, "Bidirectional Exact Pattern Matching Algorithm," *TCSET'2010*, Feb. 2010, pp. 293 (*Abstract*).

[21] Charras, C. and T. Lecroq, *Hand Book of Exact String-Matching Algorithms*, Publication 2004, First Edition, ISBN: 978-0-7546-64.

[22] T. Lecroq, "Experimental Results on Exact String Matching," *Software-Practice & Experience,* Vol. 25, pp. 727-765, 1995.

[23] A. Sleit, W. AlMobaideen, A. H. Baarah and A. H. Abusitta, "An Efficient Pattern Matching Algorithm," *Journal of Applied Sciences,* Vol. 7, no. 18, pp. 269-2695, 2007.

[24] M. Ahmed, M. Kaykobad and R. A. Chowdhury, "A New String Matching Algorithm," *International Journal Computer and Maths,* Vol. 80, no. 7, July 2003, pp. 825-834.

[25] A. Hudaib, R. Al-Khalid, D. Suleiman, M. Itriq and A. Al-Anani, "A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW)," *Journal of Computer Science,* Vol. 4, no. 5, pp. 393-401, 2008.