Improved Approach for Exact Pattern Matching

(Bidirectional Exact Pattern Matching)

Iftikhar Hussain[1], Samina Kausar[2], Liaqat Hussain[3] and Muhammad Asif Khan[4]

[1] Faculty of Administrative Sciences, Kotli, University of Azad Jammu & Kashmir Muzaffarabad, Azad Kashmir, Pakistan

[2] Faculty of Administrative Sciences, Kotli, University of Azad Jammu & Kashmir Muzaffarabad, Azad Kashmir, Pakistan

[3] Faculty of Administrative Sciences, Kotli, University of Azad Jammu & Kashmir Muzaffarabad, Azad Kashmir, Pakistan

[4] Faculty of Commerce, Kotli, University of Azad Jammu & Kashmir Muzaffarabad, Azad Kashmir, Pakistan

Abstract

In this research we present Bidirectional exact pattern matching algorithm [20] in detail. Bidirectional (BD) exact pattern matching (EPM) introduced a new idea to compare pattern with Selected Text Window (STW) of text string by using two pointers (right and left) simultaneously in searching phase. In preprocessing phase Bidirectional EPM algorithm improved the shift decision by comparing rightmost and mismatched character of Partial Text Window (PTW) to the left of pattern at same shift length. The time complexity of preprocessing phase of BD exact pattern matching is O(m) and searching phase takes O(mn/2). The proposed Bidirectional EPM algorithm is effective than the number of existing algorithms in many cases.

Keywords: Algorithm, pattern matching, exact pattern matching, searching, Bidirectional.

1. Introduction

String matching algorithms, also called string searching algorithms are a dominant class of the string algorithms which aim to find one or all occurrences of the string within a larger group of the text [1]. String matching is further divided into two classes exact and approximate string matching. In exact String matching, pattern is fully compared with the selected text window (STW) of text string and display the starting index position. In approximate string matching, if some portion of the pattern matched with STW then it displays the results.

The purpose of string matching algorithms is to find all occurrences of the pattern in the text string [1]. In this paper, we discuss in detail Bidirectional exact pattern matching algorithm based on window sliding method of exact string matching problem which will be helpful in various needs of pattern matching.

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 string 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 Bidirectional EPM algorithm's results with Boyer-Moore [3], BM Horspool [9], Quick Search [10], and Turbo BM [14] algorithms which considered efficient in terms of number of character comparisons and attempts take to complete the processing of selected text string.

In this paper, we present a brief literature review of some existing exact string matching algorithms in Section 2. Section 3 describes the basic concept and working of Bidirectional algorithm with brief example. Then we compare the Bidirectional EPM 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 4, we present experiments that compare the Bidirectional EPM algorithm with existing

algorithms. Finally, in Section 5, we conclude it on the bases of experiments results.

2. Literature Review

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 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+|∑|) and the worst case running time of searching phase is O(nm + |∑|). 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+|∑|) 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.

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+|∑|) 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+(|∑|)2) and the searching time complexity is O(mn).

3. Bidirectional Exact Pattern Matching

3.1 Basic Idea

Proposed Bidirectional exact pattern matching algorithm compares a given pattern with selected text window (STW) from both sides, simultaneously, one character at a time within the text window. It did not require the whole pattern to be searched if a mismatch occurs. In case of a mismatch or a complete match of the pattern, the mismatched and right pointers scan for the mismatched and rightmost characters of the STW to the left of the related text characters in pattern at same shift's length. Then align the pattern to new selected text window of string when rightmost and mismatched characters matched at same shifts in left of pattern. A complete match will be found when the both left and right pointers cross each other at the middle of the pattern. The comparison order of pattern's characters with selected text window can be, as shown in the figure 1.

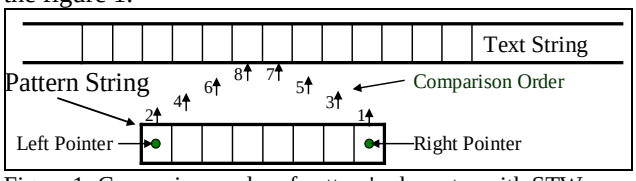


Figure 1: Comparison order of pattern's character with STW.

3.2 Working of Bidirectional EPM

Bidirectional EPM algorithm is basically based on the bad character rule of Boyer-Moore algorithm where only one character is used to identify the shifts.

Bidirectional EPM algorithm has number of cases to shift the pattern maximum to right of text window. Suppose T(1…n) is the text string and P(1…m) is the pattern and we compare P(1…m) with T(i…i+m-1) from both sides of the pattern, one character at a time, start from right side of the pattern.

Case 1: If mismatch cause by right pointer at most right position T(i+j-1) ≠ P(j) or by left pointer at most left position T(i) ≠ P(1) of the pattern here j = m then scan P(j-1…1) for character P(j′) which is equal to T(i+j-1). If character found in the pattern then align character P(j′)=P(j-1…1) with T(i+j-1) as in Figure 2.

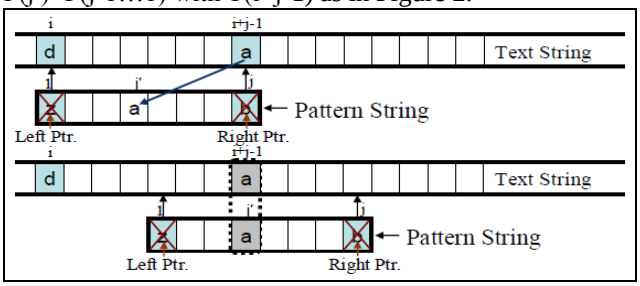


Figure 2: When right most or left most character mismatched.

Case 2: If mismatch cause by right pointer at position T(i+j-1) ≠ P(j) where 1≤j≤m and it is not the right most character of the pattern then scan P(j-1…1) for character P(j″) which is equal to T(i+j-1). And also scan P(m-1…1) for the character P(j′) which is equal to T(i+m-1). If characters found in the pattern then align character P(j″)=P(j-1…1) with T(i+j-1) and P(j′)=P(m-1…1) with T(i+m-1), if shift's length of both characters are equal as shown in Figure 3.

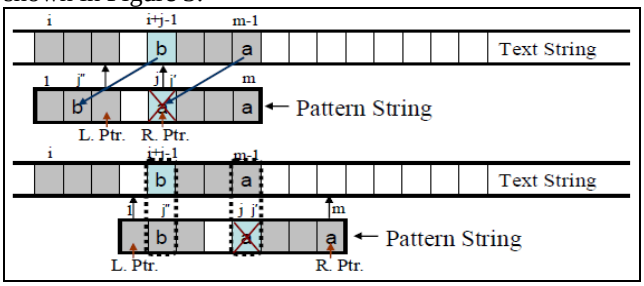


Figure 3: Mismatch by right pointer other than rightmost character.

Case 3: If mismatch cause by left pointer at position T(i+j-1) ≠ P(j) where 1≤j≤m and it is not the left most character of the pattern then scan P(j-1…1) for character P(j″) which is equal to T(i+j-1). And also scan P(m-1…1) for the character P(j′) which is equal to T(i+m-1). If characters found in the pattern then align character P(j″)=P(j-1…1) with T(i+j-1) and P(j′)=P(m-1…1) with T(i+m-1) if shifts of both characters are at equal length as shown in Figure 4.

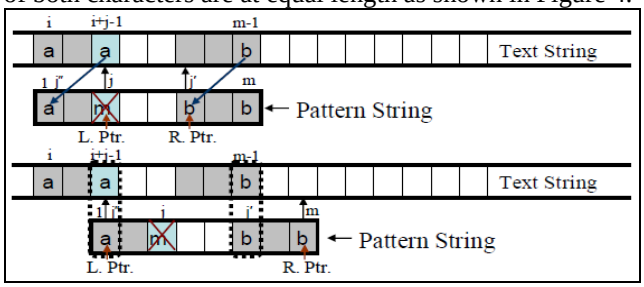


Figure 4: Mismatch by left pointer other than left most character.

Case 4: If equal shifts of mismatched (either cause by right or left pointer) and right most characters are not found, and P(j′)= T(m-1) is found at the left of mismatched character P(j) of the pattern, then align P(j′) with T(m-1) as in Figure 5.

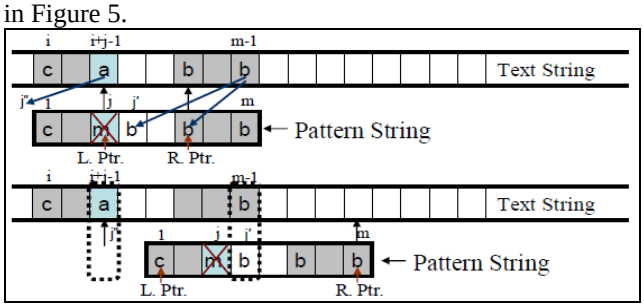


Figure 5: Mismatched character did not find in pattern.

Case 5: If equal shifts of mismatched (either cause by right or left pointer) and right most characters, did not find at the left of mismatched character P(j) of the pattern then align left most character of pattern with T(i+m) as shown in Figure 6.

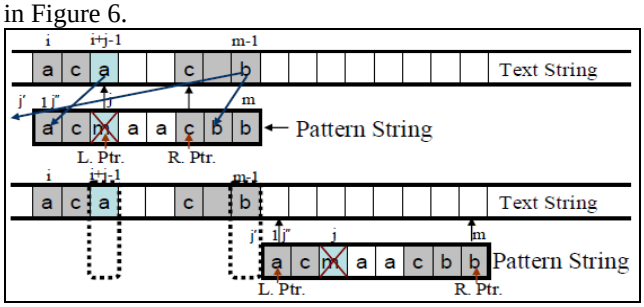
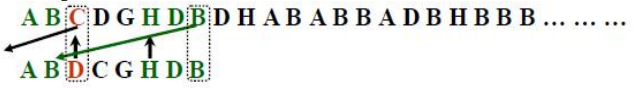


Figure 6: Maximum shift; if rightmost character did not find.

3.3 Example of Bidirectional EPM

Here we give a short example of proposed algorithm, where T="ABCDGHDBDHABABBADBHBBBDABH DBABDABBADBH" and P="ABDCGHDB".

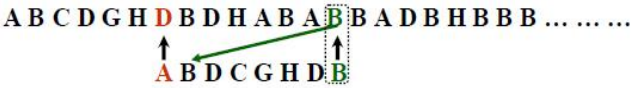
Attempt 1:



6 comparisons

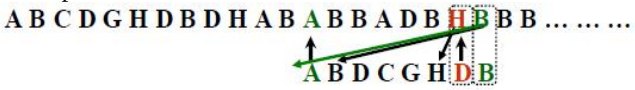
In first attempt, mismatch caused by left pointer and it is not the leftmost character of the pattern. So, Case 3 should be applied to perform shift to the right of text window.

Attempt 2:



Comparisons 2:
Again mismatch caused by left pointer but it is leftmost character of the pattern. Here Case 1 should be applied to perform shift by considering rightmost character of the text window.

Attempt 3:

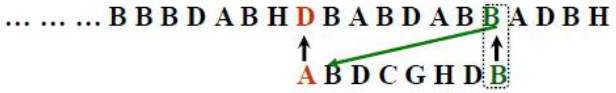


Comparisons 3:
Mismatch caused by right pointer now and it is not the rightmost character. Here Case 4 should be applied because the mismatched and rightmost characters are not found in the pattern on equal shifts.

Attempt 4:

Comparisons 2:
Here Case 1 should be applied because mismatch cause by left pointer at leftmost position of pattern.

Attempt 5:



Comparisons 2:
This example takes 15 comparisons in 5 attempts.

3.4 Implementation of Bidirectional EPM

Comparison Phase

Preprocessing phase finds occurrences of the rightmost and the mismatched characters of text string in the left of the pattern, when a mismatch caused at any position of the pattern. This phase helps to take decision of moving pattern to the right of the selected text window. As algorithm 1 show, preprocess function pass a pattern string, rightmost and the mismatched characters of the text string, and the index of mismatched character. For loop, of this phase scans pattern from second last to leftmost character of the pattern string by decrementing the indexes of pattern. Inside for loop, if rightmost character found in the pattern then check for the mismatched character at same distance as in the selected text window then returns the index of text string where the rightmost character of pattern will align. If mismatched character did not find in the left of pattern at same distance, then return the index value according to rightmost character found otherwise return index where shift of whole pattern take place.

```
Input: Pattern string of length m.
Output: Return index 'k' where last character of 'P' aligns.
Preprocessor ( P[ ], char rm, char mm, int Mindex ) {
k ← -1;
y ← length[P] - 2;
for  i ← y to 0
        if  P[i] = rm
                if mm ≥ 0 AND P[mm] = mm
                        k ← i;
                        i ← -1;
                else if  mm < 0
                        k ← i;
                        i ← -1;
        else Break;
return k;
```

Algorithm 1: Pseudocode of preprocessing phase

Similarly, the flowchart of Algorithm 1 is shown in Fig. 7.

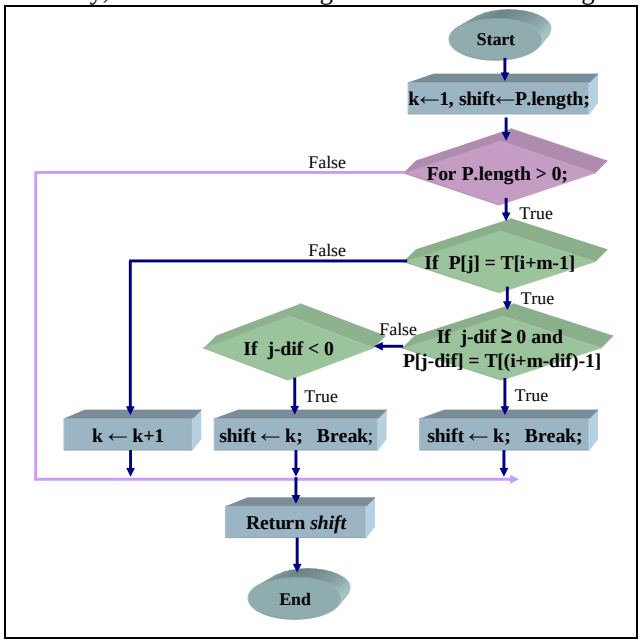


Figure 7: Flowchart of Algorithm 1.

Searching Phase

Searching will be performed between the pattern and the selected text window of the text string. Algorithm 2 shows, the searching phase of bidirectional exact pattern matching algorithm; as external while loop which is used to shift the pattern to right of the text window.

```
Input: Text string of length 'n' and Pattern of length 'm'.
Output: One or all occurrences of pattern in text
string.
BidirectionalPatternM (String T, String P) {
n ← T.length;
m ← P.length;
i ← m-1;
while i <  n
    left ← 0;
    right ← m-1;
    while  left ≤  right
        if  P[right] = P[ i - left] AND P[left] = T[i-right]
                if (left + 1) ≥  right
                    "We have match at:" (i+1) - m;
                     i←i+((m-1) - preprocessor
                index);
                left ← left+1;
                right ← right-1;
        else if  P[left] ≠ T [i-right]
                i ← i+((m-1)-(preprocessor index);
            else
                i ← i+((m-1)-(preprocessor index);
    Break Inner While;
```

Algorithm 2: Pseudocode of searching phase

Two pointers are used to compare pattern with the selected text window within the second while loop. A complete match will be found, if both pointers cross each other at middle of the pattern. Else, if mismatch caused by left or right pointers, then preprocess function will be executed to calculate the shifts where next attempt will be performed.

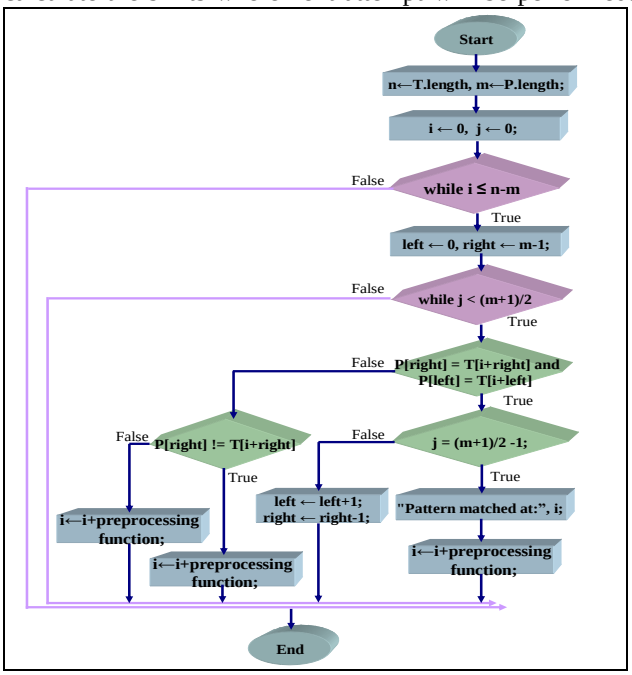


Figure 8: Flowchart of Algorithm 2.

The flowchart of searching phase of bidirectional exact pattern matching is shown above in Figure 8.

3.5 Analysis of Bidirectional EPM

Preprocessing Phase

The worst case time complexity of preprocessing phase of proposed algorithm is O(m), because only one loop is used to scan the pattern to find the rightmost and mismatched characters.

Searching Phase

The inner while loop runs at most 'm/2' times so, its worst case complexity is O(m/2) because two pointers are used within loop. 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).

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

Table 1: Complexities of Bidirectional and other 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	$O(m+(\|\sum\|)^2)$	O(mn)	$O(m+(\|\sum\|)^2)$	Left to right
RC	$O(m^2)$	O(n)	O(m+|∑|)	Specific order
Bidirec-tional	O(m+|∑|)	O(mn)/2	O(m)	Both sides one by one

The table 1 shows the preprocessing, searching and space complexities of some existing and Bidirectional EPM algorithm in asymptotic notations. The searching phase of Bidirectional algorithm takes O(mn/2) time to execute which is considered efficient than existing algorithms. The comparison order of Bidirectional EPM algorithm is from both sides of the pattern string.

4. Results and Discussions

In order to assess the efficiency of the Bidirectional exact pattern matching algorithm with existing techniques, we compare proposed technique with existing four techniques (which are considered fastest in practice) named as Boyer-Moore [3], BM Horspool [9], Quick Search [10], and Turbo BM [14]. We pass a text string of 1125 characters and compare patterns of different sizes as 4, 6, 8, 10, 12, 14 and 16 respectively from all these algorithms. Boyer-Moore [3], BM Horspool [9], Quick Search [10], and Turbo BM [14] and Bidirectional algorithms are implemented in java and results are shown in the form of graphs in figure 7 and 8.

4.1 Shift Based Comparisons

Total numbers of shifts made by each algorithm using different pattern lengths are shown in graph form in figure 9. As results in the graph shows that Bidirectional algorithm took minimum shifts as compare to other four algorithms. Results also shows that in short pattern length, number of shifts is closer to other algorithm but when pattern length is increased Bidirectional algorithm becomes more and more efficient as compare to other algorithms.

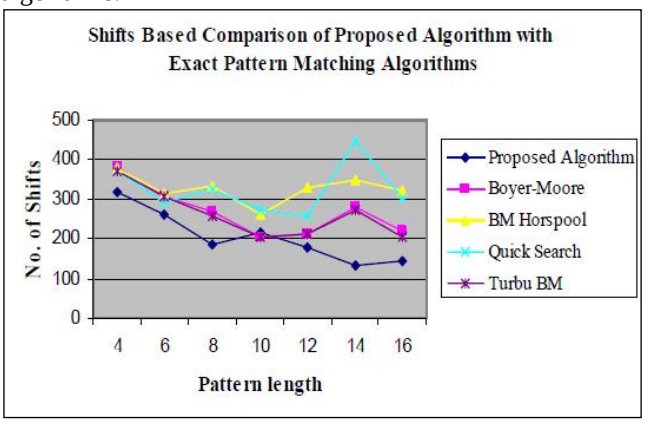


Figure 9: Shift Base Comparison

4.2 No. of Character Based Comparison

Total number of characters compare by each algorithm using different pattern lengths are shown in graph form as in figure 10.

Results in the graph shows that Bidirectional algorithm compares more characters, when pattern length is short. There are two reason first it use two pointers one compare from left and other from right simultaneously and other reason is the prefix and suffix of the pattern string are matched in text string. If prefix or suffix of the pattern early find mismatches in the text then it produce much more efficient result as compare to other algorithms.

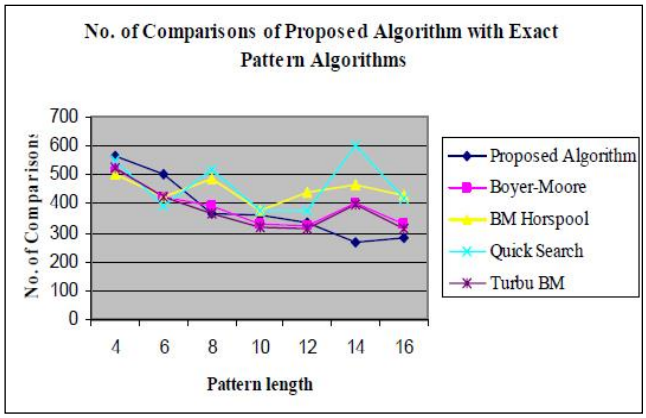


Figure 10: No. of characters compare base Comparison.

Results also shows that when long pattern is used then number of characters compare by Bidirectional EPM algorithm are less than other algorithms.

5. Conclusion

This research presents a new idea to compare pattern with selected text window from both sides of the pattern simultaneously by using right and left pointers. The preprocessing phase of Bidirectional EPM Algorithm improves shift decision by scanning rightmost and mismatched (either caused by right or left pointer) character of the selected text window to the left of pattern at same shift's length. Asymptotic analysis shows that the time complexity of Bidirectional EPM algorithm is O(mn/2) in searching phase and O(m) in preprocessing phase. Shift base comparison results show that Bidirectional algorithm is faster than existing algorithms. Numbers of characters compare by Bidirectional algorithm are also less than existing especially when long pattern is used.

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.

[26] Iftikhar Hussain, Imran Ali, Muhammad Zubair and Nazarat Bibi, "Fastest Approach to Exact Pattern Matching," *ICIET'2010*, 2010.

[27] Muhammad Zubair, Fazal Wahab, Iftikhar Hussain and Muhammad Ikram, "Text Scanning Approach for Exact String Matching," *ICNIT'2010*, PP. 118-122, 2010.

[28] Muhammad Zubair, Fazal Wahab, Iftikhar Hussain and Junaid Zaffar, " Improved Text Scanning Approach for Exact String Matching," *ICIET'2010*, 2010.

Iftikhar Hussain is working as lecturer at Faculty of Administrative Sciences, Kotli, University of Azad Jammu & Kashmir since 2010. He also worked at Savethechildren US, an international NGO. He has completed his MS in computer science, specialized in software engineering from Iqra University, Islamabad Campus in 2009. Before MSCS he has passed his BS in Information Technology from University of Azad Jammu and Kashmir in 2007. He has lot of research experience in software engineering, algorithm and internet banking.

Samina Kausar has been working as lecturer at Faculty of Administrative Sciences, Kotli, UAJK since 2007. She has completed her MS in Computer Science, specialized in database and distributed database systems, from INTERNATIONAL ISLAMIC UNIVERSITY (IIU), Islamabad Campus in 2007. Before MSCS she got the degree of MIT from University of AJ&K in 2004. She has worked as project coordinator in university of Azad Jammu and Kashmir. She has research experience in data mining and algorithms.

Liaqat Hussain is serving as Lecturer Statistics, Faculty of Administrative Sciences Kotli, University of Azad Jammu & Kashmir since 2006.He is an MSc in computer science and Statistics and recently he is performing the duties of Deputy Director Students Affairs (DDSA) at Faculty of Administrative Sciences Kotli. He is the former chairman Department of Computer Science and Information Technology.

Muhammad Asif Khan is serving as Lecturer at Faculty of Commerce Kotli, University of Azad Jammu & Kashmir since 2006.His higher degree is M.Phil and he is planning to go for PhD from abroad. He has first position in M.Sc Commerce and M.Phil. His research interests are in the area of E-banking, finance and IT.