An Efficient Exact Single Pattern Matching Algorithm

Urmila Patel[1], Mr. Mitesh Thakkar[2]
[1]M.E. Scholar, [2]Assistant Professor, [1,2]computer Engineering Department,
[1,2]L J Institute of Engineering and Technology Ahmedabad, Gujarat, India

Abstract –Exact Pattern Matching Problem defined as a finding Pattern P [1..m] in to long Text String T [1..n] with its all occurrence of exact match where n>>m. It is much studied and classical problem of computer science area. We propose an efficient version of Bidirectional Pattern is An Efficient Exact Single Pattern Matching (EESP) algorithm in which try to reduce pre-processing time and also find its all occurrences of the Pattern in to long Text String. All the previous Pattern Matching algorithms are with novel idea or variations of previous idea or hybrid idea of multiple algorithms. In this thesis, we provide the efficient version of Bidirectional Pattern Matching Algorithm. However, no Exact Pattern matching algorithm is efficient one for all kind of short as well as long Pattern.

Keyword: Algorithm, Complexity, Pre-processing phase, Pattern window, Text window.

I. INTRODUCTION

Pattern matching problem broadly categorized in to Exact Pattern Matching and Approximate Pattern Matching. Exact Pattern match defined as finding exact match of Pattern in to Text String and its applicability in Intrusion Detection System, Text Editor, plagiarism and many other areas. Approximate pattern matching defined as finding Pattern window into text string with edit distance and its applicability in Bioinformatics, Video Retrieval. Pattern matching problem also divide based on the application like single pattern match or multi pattern match.

III. LITERATURE STUDY

There are many algorithm for finding Pattern P[1..m] into given Text String with its all occurrence of exact match where n>>m.
Following are Exact Pattern Matching algorithm with their detailed description of algorithm logic, complexity of both phase (Searching phase and Pre-processing phase) and also the order of comparison in which the algorithm works

A. Brute Force (BF) Algorithm

From the study of [1,2,3,6] , Brute Force (BF) Algorithm is the basic algorithm for Exact String Matching in which first letter of pattern window will match with given text window, if match then check next character of pattern with text otherwise shift the pattern window with one.
This algorithm does not having any pre-processing phase for shifting logic of window so every time window will shift with one character only and also not require any extra space. The time complexity of this algorithm is O(mn).

B. Morris-Pratt (MP) Algorithm

From the study of [2], Morris-pratt (MP) algorithm matches the pattern with text window from left to right character by character. It performs at most 2n-1 text character comparisons during searching phase. Pre-processing phase calculate the shift logic of pattern character which knowledge used in searching phase of algorithm.
The extra space require for that is O(m) and the time complexity of pre-processing is O(m) and searching complexity is O(m+n).

C. Knuth Morris-Pratt (KMP) Algorithm

From the study of [1,2,3,6,7], Knuth Morris Pratt (KMP) algorithm is proposed in year 1977.This algorithm match the pattern with text window from left to right character by character. This algorithm has the Pre-processing logic for taking the decision for shift, so it uses the knowledge of shift in the searching phase.
The extra space require for that is O(m) and the time complexity of pre-processing is O(m) and searching complexity is O(mn).

D. Boyer-Moore (BM) Algorithm

From the study of [2,3,6,8], Boyer-Moore (BM) algorithm is a practically efficient string matching algorithm in usual applications. This algorithm match the pattern with text window from right to left character by character and starts with right most character. This algorithm has two logic functions for shift window to the right which used when complete match or mismatch occur. Good suffix shift and Bad character shift both used in this algorithm.
Its time and space complexity of pre-processing phase is O(m+|Σ|).Its Searching phase complexity is O(mn).Best case Performance is O(n/m).

E. Boyer-Moore Horspool (BMH) Algorithm

From the study [2,3,6,10], Boyer-Moore Horspool (BMH) is simplified version of Boyer-Moore algorithm and easy to implement. This algorithm only use bad-character shift for computing shift. Pre-processing phase prepare the bad character shift value and that table used during the searching phase of algorithm.
Its pre-processing time complexity is O(m+|Σ|) and space complexity is O(|Σ|).Searching phase complexity is O(mn).

F. Raita Algorithm

From the study of [2,11], Raita algorithm which first compare last character of pattern window to pattern, then if they match compare the first character of pattern window, then if they match compare middle character of

pattern to text window. And finally if they match compare other character of pattern window from second last character but possibly compare middle character again.
Its pre-processing time complexity is O(m+|Σ|) and space complexity is O(|Σ|).Searching phase complexity is O(mn).

G. Quick Search (QS) Algorithm
From the study of [2], Quick Search algorithm (QS) is simplified version of Boyer-Moore algorithm and only uses the bad-character shift. This algorithm matches the pattern with text window in any order character by character. Its best performance in short pattern and large alphabets.
Its pre-processing time complexity is O(m+|Σ|) and space complexity is O(|Σ|).Searching phase complexity is O(mn).

H. A Fast String Matching (VS) Algorithm
From the Study of [12], A Fast string matching algorithm (VS) is proposed in 2011. It has been improved using the shift provided by the Horspool search bad-character and by defining a fixed order of comparison. This algorithm firstly match right most character if match then secondly left most character and it also match then thirdly middle character, if match then start from the second last character and go towards left side.
Its time complexity of pre-processing phase is O(|Σ|). Its matching phase complexity is O(m(n-m+1)).

I. Fastest Bidirectional (FBD) Algorithm
From the study of [13, 14], Fastest Bidirectional algorithm (FBD) is proposed in year 2010.This algorithm match the pattern with text window from both sides simultaneously. In case of complete match or mismatch of pattern, scan for the mismatched and right most characters of the text window to the left of the related text characters in pattern.
Its pre-processing time complexity is O(m) and space complexity is O(m).Searching phase complexity is O(mn/2).

J. Improved Bidirectional (IBD) Algorithm
From the study of [15], Improved Bidirectional algorithm (IBD) is proposed in year 2013. This algorithm matches the pattern with text window from both sides simultaneously as BD. In case of complete match or mismatch of pattern, scans pattern from second last character to leftmost character of pattern. If both character occurs at equal distance then window shift at that location.
Its pre-processing time complexity is O(m) and space complexity is O(m).Searching phase complexity is O(mn/2).

IV. COMPARSION

TABLE-I ALGORITHM ANALYSIS TABLE

Algo Name	Comparison Order	Pre Processing complexity	Searching complexity
BF	Not Relevant	NO	O(mn)
MP	Left to Right	O(m)	O(m+n)
KMP	Left to Right	O(m)	O(m+n)
BM	Right to Left	O(m+|Σ|)	O(mn)
BMH	Right to Left	O(m+|Σ|)	O(mn)
Raita	Specific Order	O(m+|Σ|)	O(mn)
QS	Any Order	O(m+|Σ|)	O(mn)
VS	Specific Order	O(|Σ|)	O(m(n-m+1))
FBD	Both Side Simultaneously	O(m)	O(mn/2)
IBD	Both Side Simultaneously	O(m)	O(mn/2)

V. PROPOSED METHOD

Proposed Algorithm is based on to the Improved Bidirectional Algorithm (IBD). This algorithm working in two phase pre-processing phase and actual searching phase.
Detail description in provided below.

A. Pre-processing Phase
Pre-processing phase defined the work done to calculated next location where pattern will likely to happen and start searching at that location with out compare with each and every position of the Text string. Pre-processing phase gives better shift decision in searching phase and give faster execution of algorithm.

Case-1: Full Match of Pattern window with Text Window.
If first character present in pattern string else then first position then record that next location. At the second occurrence of first character location is next shift location, because at their likely to have pattern. And if the second occurrence of first character is zero then there is no possibility to happen string in between so we move at the total length of string, so maximum shift is equal to the pattern length.

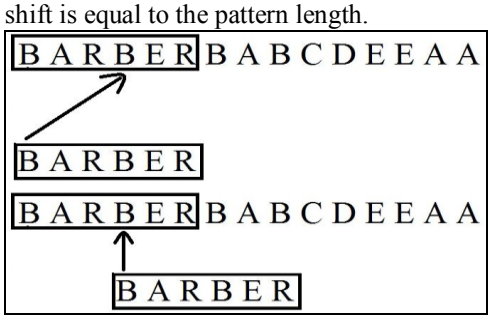

Fig 1: Second occurrence of first character happen in the pattern

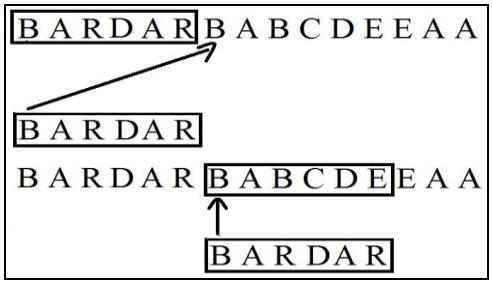

Fig 2: No second occurrence of first character in the pattern

Case-2: Mismatch in comparison of Pattern window character versus of Text window.

When the mismatch occurs then pre-processing phase calculates the best next shift where pattern will likely to happen. In the pre-processing check for the pattern character pair occur at equal length in text window if its matches then shift at that location otherwise shift with pattern length.

```
T: ABCDAEFDAECHT
P:ABCDAEF

5678910
EFDAEC
100001
ABCDAE

EFDAEC
 10001
 ABCDAE

EFDAEC
  1001
  ABCDAE

5678910
EFDAEC
   101
   ABCDAE

Next i=8
```

```
T: ABCDAEFDFEGHT
P:ABCDAEF

5678910
EFDFEG
100001
ABCDAE

EFDFEG
 10001
 ABCDAE

EFDFEG
  1001
  ABCDAE

EFDFEG
   101
   ABCDAE

EFDFEG
    11
    ABCDAE
Next i=10
```

Fig 3: pre-processing –Case 2

B. Searching Phase

Searching phase is modified version of Improved Bidirectional (IBD) Algorithm. In the Searching phase actual comparison of Text window and Pattern window made. Comparison work like both side character simultaneously but first compare right side then left side. Comparison start with right most character of pattern window with right most character of text window and left most character of pattern window with left most character of text window and after continue toward middle of pattern window.

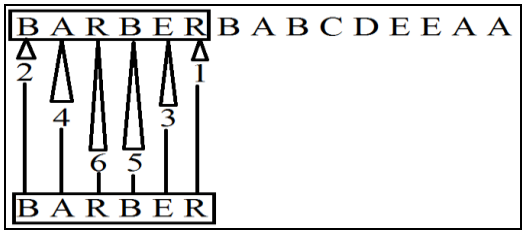

Fig. 4: Comparison order of Pattern window with Text window

II. EXPERIMENTS AND RESULT

The performance of Efficient Exact Single Pattern Matching (EESP) algorithm is measured and compared with BF, MP, KMP, BM, BMH, RAITA, QS, VS, FBD and IBD. For experiments we have used Text of fix English random character and Pattern of different length of random character which is substring of Text. A Text length of 1000 English random character and Pattern lengths are {10, 20, 30, 40, 50, 60, 70, 80, 90, 100} which are chosen randomly from substring inside Text.

Performance measured with calculating number of character comparison takes for finding pattern with in text and result compared with original as well as existing algorithm which shown in Figure 5. The Result in this figure shows that EESP take less character comparison than the other algorithm for different length.

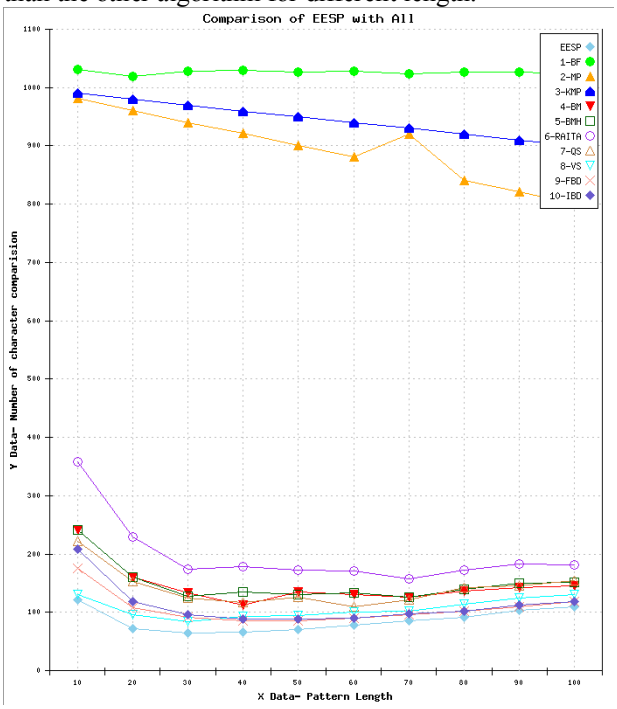

Fig 5: Comparison of EESP with all

V.COCLUSION

In this paper various Exact Pattern Matching Algorithm has been described with their time and space complexity. Comparative study of various algorithm result in applicability of algorithm based on the requirement of given problem, so rather then applying each Pattern Matching algorithm in every problem, we choose optimal algorithm for that problem. Comparative result `shows that Efficient Exact Single Pattern matching algorithm is quiet efficient then the other algorithm for short as well as long pattern.

REFERENCE

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L., "Introduction to Algorithms", Chapter 34, MIT Press, 1990.

[2] Christian Charras, Thierry Lecroq, "Handbook of Exact String-Matching Algorithm".

[3] Nimisha Singla, Deepak Garg,,"String Matching Algorithms and their Applicability in various

Application",IJSCE,Vol.1,Issue-6,Jan-2012,pp.218-222.

[4] Jonathan Leidig and Christian Trefftz,"A Comparison of the Performance of four Exact String Matching Algorithms",IEEE EIT,May-2007,pp.333-336.

[5] Vidya SaiKrishna, Prof. Akhtar Rasool and Dr. Nilay Khare, "String Matching and its Applications in Diversified Fields", IJCSI, Vol. 9, Issue 1, No 1, January 2012,pp.219-226.

[6] Kamal M. H. Alhendawi, Ahmad Suhaimi Baharudin, "String Matching Algorithms (SMAs): Survey & Empirical analysis", International Journal of Computer Science and Management Research, Vol 2 Issue 5, May 2013 , pp.2637-2644.

[7] 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.

[8] R.S. Boyer, J.S. Moore, "A fast string searching algorithm," Communi-cation of the ACM, Vol. 20, No. 10, 1977, pp.762–772.

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

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

[11] TIMO RAITA,"Tuning the Boyer–Moore–Horspool String Searching Algorithm",SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 22(10), OCT-1992, pp.879–884.

[12] H N Verma and Ravendra Singh, "A Fast String Matching Algorithm" ,IJCTA ,Vol.2(6) ,NovDec-2011 ,pp.1877-1883.

[13] Iftikhar Hussain, Muhammad Zubair, Jamil Ahmed and Junaid Zaf-far, "Bidirectional Exact Pattern Matching Algorithm," TCSET'2010, Feb. 2010, pp.293.

[14] Iftikhar Hussain, Imran Ali, Muhammad Zubair and Nazarat Bibi, "Fastest Approach to Exact Pattern Matching", ICIET, June-2010 ,pp.1-5.

[15] Iftikhar Hussain, Syed Zaki Hassan Kazmi, Israr Ali Khan,Rashid Mehmood, "Improved-Bidirectional Exact Pattern Matching", IJSER,Vol. 4,Issue 5,May- 2013,pp.659-663.

