Applying Bi-Directional Search Strategy in Selected String Matching Algorithms

Grishma Pandey
PG-Scholar
IET-Devi Ahilya University
Indore - 452017, India

G. L. Prajapati, PhD
Department of Computer Engg.
IET-Devi Ahilya University
Indore - 452017, India

ABSTRACT
String matching is an important problem in computer science having several practical applications. In this paper, we apply bi-directional searching mechanism in exact string matching algorithms: Boyer Moore, Brute Force, Knuth- Morris Pratt, and Rabin Karp. Experiments show that this strategy leads to better efficiency of these string matching algorithm.

Keywords
Boyer Moore Algorithm, Brute Force Algorithm, Knuth-Morris Pratt Algorithm, Rabin Karp Algorithm

1. INTRODUCTION
String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (patterns) are found within a larger string or text. The purpose of string matching algorithms is to find all occurrences of the pattern in the text string. There are main 2 techniques of string matching one is exact matching. In exact string matching, pattern is fully compared with the selected text window (STW) of text string and display the starting index position, and other is approximate matching. In approximate string matching, if some portion of the pattern matched with STW then it displays the results.

The problem of string matching is that there are two strings one is text T [1.....n] i.e. is main string given and the other is pattern P [1.....m] i.e. is the given string to be matched with the given main string given m<=n.

1.1 Exact String Matching Problem
We are given a text string pattern string we want to find all occurrences of P in T. In Exact string matching problem the pattern is exactly found inside the text [1].

Consider the following example:

T = AGCCTAAGCTCCTAAGTC

P =CCTA

There are two occurrences of p in T as shown below:

AGCCTAAGCTCCTAAGTC

A brute force method for string matching algorithm:

T =ACCACTAGA

P =ACTA

ACTA

ACTA

ACTA

If the brute force method is used, many characters which had been matched will be matched again because each time a mismatch occurs, the pattern is moved only one step.

There are many string matching algorithms. Nearly all of them are concerned with how to slide the pattern. Few of them are listed below.

1.1.1 Boyer Moore Algorithm [2] [8] [9]
- Performs the comparisons from right to left.
- Preprocessing phase in O(m) time and space complexity.
- Searching phase in O(m×n) time complexity.
- n text character comparisons in the worst case when searching for non-periodic pattern.
- O(n/m) best performance.

1.1.2 Brute Force Algorithm [6] [7] [8]
- No preprocessing phase.
- Constant extra space needed.
- Always shifts the window by exactly 1 position
- to the right.
- Comparisons can be done in any order.
- Searching phase in O (m×n) time complexity.
- 2n expected text character comparisons.

1.1.3 Knuth- Morris Pratt Algorithm [5] [8]
- Performs the comparisons from left to right.
- Preprocessing phase in O(m) space and time complexity.
- Searching phase in O(m+n) time complexity independent from the alphabet size.
- Performs at most 2n -1 text character comparisons during the searching phase.
- Delay bounded by log Φ (m) where Φ is the golden ratio $(1+\sqrt{5})/2$.

1.1.4 Rabin Karp Algorithm [3] [9]
- Uses a hashing function.
- Preprocessing phase in O (m) time complexity and constant space.
- Searching phase in O(m× n) time complexity.
- O(m+n) expected running time.

1.1.5 Morris Pratt Algorithm [4]

- Performs the comparisons from left to right.
- Preprocessing phase inO(m) space and time complexity.
- Searching phase in O(m+n) time complexity independent from the alphabet size.
- Performs at most 2n -1 text character comparisons during the searching phase.
- Delay bounded by m.

1.1.6 Quick Search algorithm [7]

- Simplification of the Boyer Moore algorithm.
- Uses only the bad character shift.
- Easy to implement.
- Preprocessing phase in O (m) time and O(σ)space complexity.
- Searching phase in O(m×n) time Complexity.
- Very fast in practice for short patterns and large alphabets.

2. LITERATURE REVIEW

Boyer-Moore (BM) [2] [3] [4] [5] algorithm is proposed 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).

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).

Quick Search (QS) [9] 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.

3. IMPROVED STRING MATCHING

3.1 Basic idea

Proposed Improved String matching algorithm compares a given pattern with selected text window 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 shifts 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.

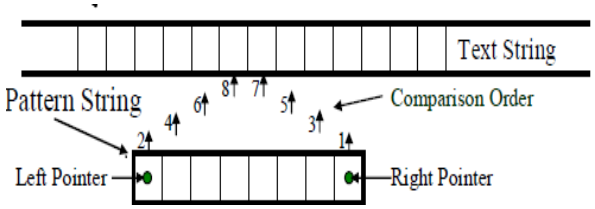

Figure 1: Comparison of pattern

3.2 Working of String Matching algorithm

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

3.3 Implementation of String Matching Algorithm

3.3.1 Preprocessing 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, Preprocessing 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.

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

3.3.2 Finding Phase

Finding will be performed between the pattern and the selected text window of the text string. Algorithm 2 shows, the Finding phase of Improved String Matching algorithm; as external while loop which is used to shift the pattern to right of the text window.

	Algorithm 2: Finding phase

	Input: Text string of length ―n‖ and String of length ―m‖.
	Output: One or all occurrences of pattern in text string.
	ImprovedPatternM (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;

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.

4. EXPERIMENTAL RESULTS

The efficiency of Improved String Matching algorithm is measured and compared with existing techniques we compare proposed technique with existing techniques named as Boyer-Moore, BMH, Knuth-Morris-Pratt, Quick Search and Rabin Karp. We pass a text string of n characters and compare patterns of different sizes as 5, 10, 15, 20 and 25 respectively from all these algorithms. Boyer- Moore, BMH, Knuth-Morris-Pratt, Rabin Karp and Improved implemented in c and results are shown in the form of graphs in figure .the results are shown by using graphs in figs.

We have conducted all the experiments on a Notebook PC with CPU 1.2 GHz using the gcc compiler.

The Table 1 and Figure 1 show the running time of Boyer-Moore algorithm with different number of patterns (5 to 500 patterns) but the minimum length of pattern is 5.

In Boyer-Moore algorithm, if the number of pattern is increases, the running time is also increase.

Table 1: Running Time Of Boyer-Moore Algorithm Depending On Number Of Patterns

Number of pattern	Running time (ms)
5	0.006000
10	0.061000
15	0.114000
20	0.119000
25	0.137000

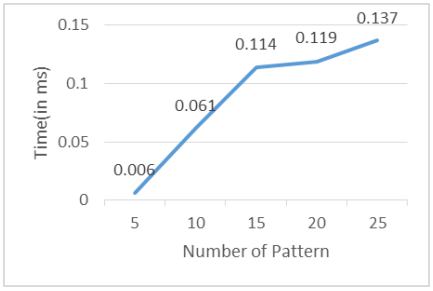

Figure 1: Running Time Of Boyer-Moore Algorithm Depending On Number Of Patterns

The Table 2 and Figure 2 show the running time of Knuth-Morris-Pratt algorithm with different number of patterns (5 to 500 patterns) but the minimum length of pattern is 5.

The Table 3 and Figure 3 show the running time of Improved String Matching algorithm with different number of patterns (5 to 500 patterns) but the minimum length of pattern is 5.

Table 2: Running Time Of Knuth-Morris-Pratt Algorithm Depending On Number Of Patterns

Number of pattern	Running time (ms)
5	0.185000
10	0.201000
15	0.231000
20	0.260000
25	0.287000

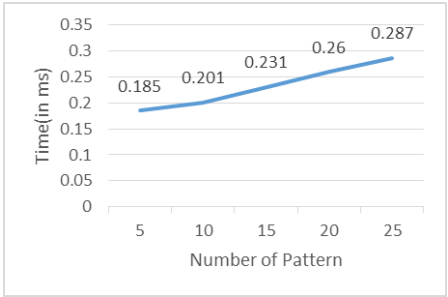

Figure 2: Running Time Of Knuth-Morris-Pratt Algorithm Depending On Number Of Patterns

Table 3: Running Time Of Improved String Matching Algorithm

Number of pattern	Running time (ms)
5	0.04000
10	0.62000
15	0.66000
20	0.79000
25	0.86000

As results in the graph shows that improved 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 Improved String Matching algorithm becomes more and more efficient as compare to other algorithms.

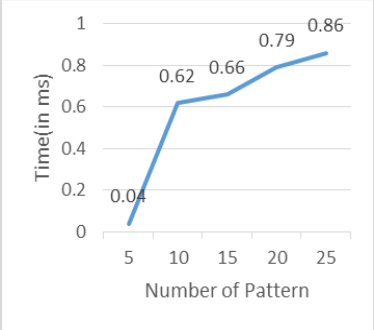

Figure 3: Running Time Of Improved String Matching Algorithm Depending On Number Of Patterns

a) Attempts Base Comparison

Total numbers of attempts taken by each algorithm using different pattern lengths are shown in graph.

b) No. of Characters compare base Comparison

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

Graph.

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 4 shows comparison on the basis of character. In X axis Pattern Length is given and corresponding time(in ms) for searching pattern shown in y axis.

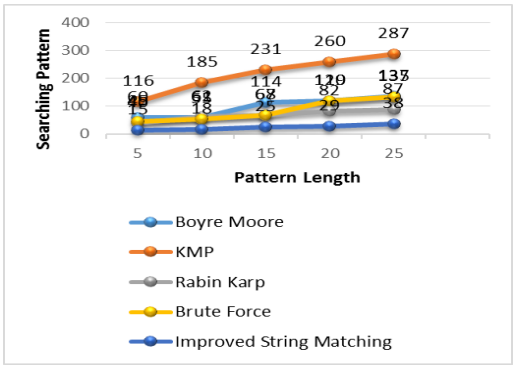

Figure 4: Character Based Comparison

5. CONCLUSIONS

In this paper, comparison is done on various kinds of string matching algorithms. It is analyzed that the Boyer Moore algorithm is extremely fast for large sequences, however improved String Matching Algorithm improves shift decision by scanning rightmost/leftmost character of the selected text window. The analysis shows that the time complexity of Improved String Matching Algorithm is O(mn/2) in searching phase and O(m) in preprocessing phase.

6. REFERENCES

[1] Nimisha Singla, Deepak Garg,"String Matching Algorithms and their Applicability in various Applications", International Journal of Soft Computing and Engineering (IJSCE), Volume-I, Issue-6, January 2012.

[2] Apostolico , A. and Giancarlo, R., The Boyer-Moore-Galil, String searching strategies revisited, SIAM Journal on Computing, Vol. 15, 1986, pp. 98-105.

[3] Charras,C., Lecroq, T. and Pehoushek, J.D., A very fast string matching algorithm for small alphabets and long patterns, in Proceedings of Combinatorial Pattern Matching, 1998, pp. 55-64.

[4] Amir A., Lewenstein M., and Porat E., Faster Algorithms for String Matching with K-Mismatches, Journal of Algorithms 50(2004) 257-275.

[5] Knuth D.E., Morris J.H., and Pratt V.R., Fast Pattern Matching in Strings, Journal of Computing, Vol.6, No.2, 1977.

[6] 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.

[7] Lecroq, T.,A, Variation on the Boyer-Moore algorithm, Theoretical Computer Science, Vol. 92, No. 1, 1992, pp. 119-144.

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

[9] Singla N., Garg D., String Matching Algorithms and their Applicability in various Applications, International Journal of Soft Computing and Engineering, ISSN: 2231-2307, Volume-I, Issue.-6, January 2012.