Multiple Sliding Window based Pattern Matching Algorithms: Survey

[1]Anushka Dixit, [2]Rinki Singh
[1]M.Tech Scholar, [2]Assistant Professor
[1,2]Department of Computer Science and Engineering,
[1,2]VIET, G.B. Nagar, INDIA

Abstract : The functional and structural relationship of the biological sequence is determined by similarities in that sequence. Pursuing of similarity among biological sequences is an important research area that can bring insight into the evolutionary and genetic relationships among the genes. This paper will present a critical review of different multiple sliding window based pattern matching algorithms, which are Two Sliding Window (TSW), Enhanced Two Sliding Window (ETSW) and Four Sliding Window (FSW). All these algorithms are variations of Berry-Ravindran algorithm. This review provides insight into determining the efficient sliding window based algorithm. An algorithm with less number of character comparisons and less number of passes in proportion is the one that provides good result.

***IndexTerms* - Character comparison, exact pattern matching, sliding window, Berry-Ravindran, bad character shift.**

I. Introduction

String-matching is important in the wider domain of text processing. They play an important role in theoretical computer science by providing solution to challenging problems. First computer programs to use pattern matching were text editors, which uses Boyer Moore algorithm [1]. Later on searching amino acid sequence patterns in genome and protein sequence databases started using concept of pattern matching [2] [3]. Exact matching [4] finds all the exact occurrences of a pattern inside a large body of text.

In the past, various algorithms have been developed and analyzed [4] . Boyer Moore algorithm was further modified by Berry and Ravindran [5], where shift is performed by two characters immediately to the right of window. Ahead of this, Berry – Ravindran based algorithm was further extended by TSW [6] to get the better shift. This TSW algorithm was further extended by ETSW [7], which inherits property of Berry-Ravindran and at last the algorithm named as FSW [8] was derived.

This paper is organized as follows. Sec. 2 presents related terminologies. Sec. 3 presents the literature review. Sec. 4 presents the working example and their analysis depending on number of character comparisons and number of passes. Finally, we conclude in Sec. 5.

II. Related Terminologies

Exact Pattern Matching

Exact pattern matching finds all occurrences of a single pattern in the text. The pattern is denoted by x=x [0 .. m-1] and its length is equal to m. Text is denoted by y=y [0 .. n-1] and its length is equal to n [1]. Scanning of text is performed by first aligning left ends of the window and the text, then comparing characters of window with the characters of the pattern known as *attempt.* Same procedure is followed until right end of the window goes beyond the right end of the text. Different algorithms work according to changes in their searching phase and shift distance either from left to right or from right to left.

Preprocessing Phase

This is the first phase in every of the pattern matching algorithms. In this phase a table is constructed based on different functions used in different algorithms.

Searching Phase

This is the second phase of algorithm. In this phase searching of the pattern is started in the given text. Shifting of window is done based on the shift values calculated and written within the table. Different algorithms use different techniques for shifting of window.

III. Literature Review

Pattern matching algorithms are used for finding several substrings within text or sequences provided. Many Exact pattern matching algorithms are reported in the literature depending on their number of shift values. Algorithms with large shift values are far better than those having small shift value. Further we have discussed some exact pattern matching algorithms in concise.

Berry and Ravindran Algorithm: Berry and Ravindran [5] designed an *algorithm* which performs shifts by considering the bad character shift. It uses a single window for the scanning of text. It uses two dimensional arrays which are used for calculating shift values in the preprocessing phase. It constructs Berry-Ravindran bad character table using brBc function that works for each pair of character in the pattern. In searching phase, shift value of pattern depends on the two successive characters in the text immediately to the right of window. This reduces number of comparisons between text and pattern. Its main characteristics are:

- Reduces number of comparisons
- It uses Quick search and Zhu Takaoka algorithm

TSW(Two Sliding Window)Algorithm: Name of algorithm states that it uses two sliding windows that scans text from both sides(one from left and one from right). In preprocessing phase, arrays are constructed using two arrays [6] nextl (for left sliding window) and nextr (for right sliding window) using different equations each containing four rules within them depending on which shifting of window is done. Arrays are used in this algorithm to store the calculated shift values. The two windows search the text in parallel as the text is divided into two parts and each part is of size ┌ n/2┐. It uses idea of BR bad character shift function to get better shift values during the searching phase. BR differs by TSW only in number of sliding windows. The TSW algorithm finds either the first occurrence of the pattern in the text through the left window or the last occurrence of the pattern through the right window.

In this Searching phase, the text string is scanned from left to right and right to left. In case of mismatch, the left window is shifted to the right while search starts from left. In case search starts from right window then it is shifted towards left. Both windows are shifted until the pattern is found or the windows reach the middle of the text.

- Uses two sliding windows
- Reduce space and time complexity by using one-dimensional array.
- Algorithm scans only the text to get matches.

ETSW(Enhanced Two Sliding Window)Algorithm: *In preprocessing phase*, text and pattern are both divided into two parts. Each part of the text is of length ┌ n/2┐ while each part of the pattern [7]is of length ┌ m/2┐. As in TSW two arrays *nextl* and *nextr are generated*, each of one-dimension, same procedure is followed here. The shift values of the *nextl* array are calculated according to Berry- Ravindran bad character algorithm. The shift values of the *nextr* array are calculated according to the TSW shift function.

In Searching phase, the text string is scanned from left to right and right to left as same as TSW but with some enhancement. It uses four pointers, two for each window. The left window uses *L* and *temp_newlindex* pointers while the right window uses the *R* and *temp_newrindex* pointers.
In each window, the first character of the pattern is compared with the corresponding character of the text while at the same time the last character of the same pattern is also compared with the corresponding character of the text. This primary step will reduce the number of comparisons done later in the left and the right windows. We will discuss searching by the left and right windows.

- Reduction in number of Comparision

FSW(Four Sliding Window)Algorithm: The pre-processing phase of the FSW algorithm is the same as in ETSW algorithm. Two arrays *nextl* and *nextr* are generated [8]. The shift values are calculated according to Berry-Ravindran bad character algorithm (BR). The shift values needed to search the text from the left side are stored in the *nextl* array and *nextr* array contains the shift values needed to search the text from the right side. Two consecutive characters of pattern are given index starting from 0 for construction of two arrays (nextl and nextr). For eg: Pattern= "dcba", the consecutive characters dc, cb and ba are given the indexes 0, 1 and 2.

In Searching phase, left part is named part 1 and right part is named part 2. Four windows are created out of which two windows are created for each part of the text, to search for the pattern in parallel. The left and right windows of part 1 are named p1L and p1R and of part 2 [15]are given names p2L and p2R respectively. At the beginning of the search, p1L and p2R windows are aligned with the left most and rightmost sides of the text. p2L window is aligned with the text at index $n/2 + m-1$ while p1R window is aligned with the text at index $n/2-1$.

- Good for large alphabet size
- Uses four sliding window.

IV. Working Example

This section shows the functioning of all four algorithms by taking a text and a pattern and applying different techniques used in algorithms. We will get difference in the number of comparisons and number of passes which will specify the better algorithm of all four.

Text: AACATAGAGAAACATAGAGAACATAG
Pattern: ATCTGACG (Length=8)

1) Berry-Ravindran Algorithm: Constructing table using brBc function

brBc(a)	A	T	C	G
A	9	8	3	10
T	9	10	7	5
C	9	6	10	2
G	1	1	1	1

First pass:
[AACATAGA]GAAACATAGAGAACATAG
 12
 ATCTGACG

There is a mismatch on second comparision. Shift is done based on brBc [G] [A] =1

Fourth pass:
 AACATAGAGAA[ACATAGAG]AACATAG
 1 2
 ATCTGACG

Number of comparisons: 8
Number of passes: 4

2) TSW: Constructing first the Berry Ravindran bad character table in preprocessing phase.
Shiftl table

index	0	1	2	3	4	5	6
brBc(a)	8	7	6	5	4	3	2

Shiftr table

Index	0	1	2	3	4	5	6
Tswbc(a)	2	3	4	5	6	7	8

Searching Phase
First pass:
[AACATAGA]GAAAC|ATAGA[GAACATAG]
 12 → ← 21
 ATCTGACG ATCTGACG

There is a mismatch on second comparison on left window and right window so shifting will be done as following:

Shift is done based on GA, for left brBc [G][A] = 4
Shift is done based on GA, for right brBc [G][A] = 4

Third pass:

 AACAT[AGAGAAAC]|[ATAGAGAA]CATAG
 1 1
 ATCTGACG ATCTGACG

Number of comparisons: 7
Number of passes: 3

3) ETSW Algorithm: brBc () and Tsw() function are used. The table in the pre-processing phase will be same as TSW algorithm. Difference is in searching phase.
Shiftl table

Index	0	1	2	3	4	5	6
brBc(a)	8	7	6	5	4	3	2

Shiftr table

Index	0	1	2	3	4	5	6

Tswbc(a)	2	3	4	5	6	7	8

First pass:

```
[AACATAGA]GAAAC|ATAGA[GAACATAG]
 1 2        1 →            ← 1           1
ATCTGACG                          ATCTGACG
```

There is a mismatch on second comparison in left window from left side and on first comparision in right window from left side done parallel, so shifting will be done as following:

Shift is done based on GA, for left brBc [G][A] = 4
Shift is done based on GA, for right brBc [G][A] = 4

Third pass:

```
 AACAT[AGAGAAAC]|[ATAGAGAA]CATAG
          1           1                1
          ATCTGACG   ATCTGACG
```

Number of comparisons: 6
Number of passes: 3

4) FSW Algorithm: Table is constructed using brBc function as in Berry-Ravindran with four sliding windows. Pre-processing phase of this algorithm is also same. Difference is found in searching phase.
First pass:

```
[AACATAGA]GAAAC|ATAGA [GAACATAG]
 1 2        1 →            ← 1           1
ATCTGACG                          ATCTGACG
                       1           21
                      ATCTGACG
                        1
                       ATCTGACG
```

First row is for P_{1L} and $P_{2R.}$ Second row is for $P_{1R.}$ Third row is for $P_{2L.}$

Number of comparisons: 4
Number of passes: 1

V. Results & Comparison

In this section, Table1 shows the number of character comparisons and number of passes that are criteria for defining efficiency of algorithms in this paper. Fig.1 shows graphical representation of number of comparisons and Fig.2 shows graphical representation of number of passes.

Table 1: Comparison among algorithms for pattern length = 8

Algorithms	Number of Comparisons	Number of Passes
Berry- Ravindran	8	4
TSW	7	3
ETSW	6	3
FSW	4	1

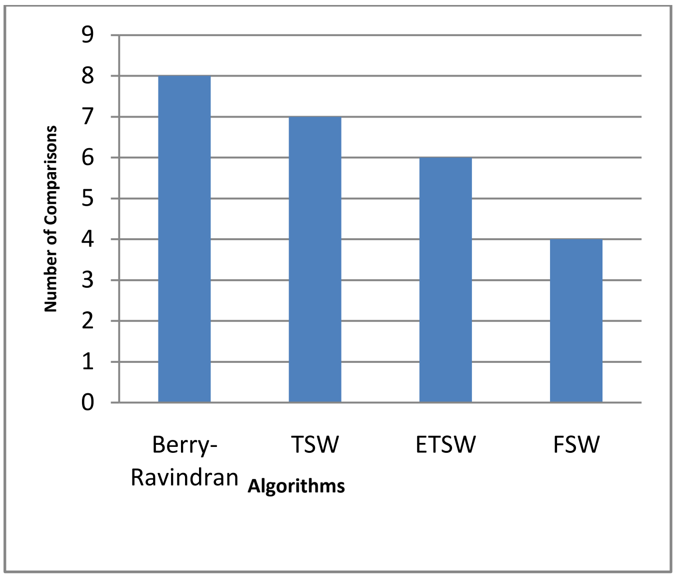

Fig.1: Graph showing Number of Comparisons among algorithms

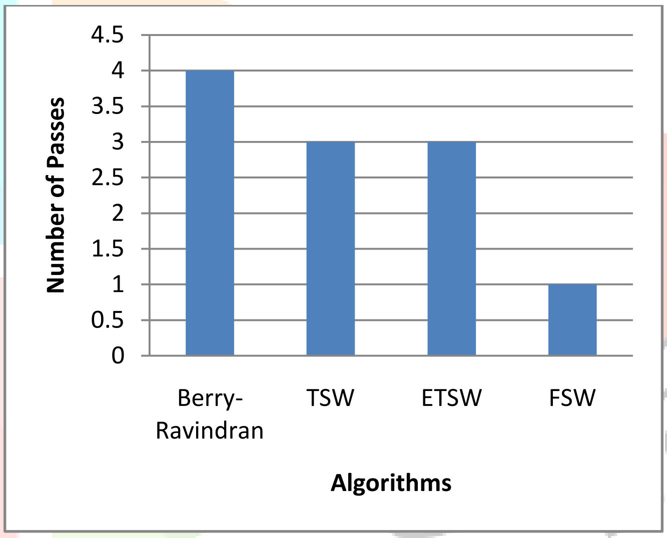

Fig.2: Graph showing Number of passes among algorithms

VI. Conclusion

This paper presents state of art comparison of the very fast algorithms on exact string matching. The algorithms have been analyzed on the basis of their number of comparisons and number of passes that can be called as number of iterations or attempt. Table 1 shows the number of character comparison and number of passes, algorithms are taking. Fig 1. and Fig 2. shows the graph of number of character comparisons and number of passes. The graph shows FSW algorithm outperforms best as it reduces the number of character comparisons and reduces the time to search the matches. In near future, algorithms can be tested on increased size of patterns and text. The algorithm can be implemented on RNA and other huge data sets

References

[1] Boyer, R. S., And Moore, J. S. A fast string-searching algorithm. Communication of ACM 20, 10 (1977), 762-772.
[2] Aoe, Jun-ichi, 1951- *Computer algorithms: string pattern matching strategies*. IEEE Computer Society Press, Los Alamitos, Calif, 1994
[3] Aho, A. V. "Algorithms for finding patterns in strings Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen." (1990).
[4] Knuth, D. E. Fast Pattern Matching in Strings. SIAM Journal of Computing 6, 2 (1977), 323-50.
[5] Berry, Thomas, and S. Ravindran. "A Fast String Matching Algorithm and Experimental Results." *Stringology*. 1999.

[6] Hudaib A., Al-Khalid R., Suleiman D., Itriq M. and Al-Anani A. A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW). Journal of Computer Science 2008; 4 (5): 393-401.

[7] 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.

[8] Itriq, M., Hudaib, A., Al-Anani, A., Al-Khalid, R. and Suleiman, D (2015) Four Sliding Windows Pattern matching Algorithm (FSW). Journal of Software Engineering and Applications,2015,8,154-165.

IJCRT