Volume 5, No. 1, January 2014
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

STRING MATCHING RULES USED BY VARIANTS OF BOYER-MOORE ALGORITHM

Jamuna Bhandari*, Anil Kumar
IEEE Student member*, IEEE Senior Member
Department of Computer Science and Engineering, Manipal University Jaipur, Rajasthan
bunu17_bhandari@yahoo.com, dahiyaanil@yahoo.com

Abstract: String matching problem is widely studied problem in computer science, mainly due to its large applications used in various fields. In this regards many string matching algorithms have been proposed. Boyer-Moore is most popular algorithm. Hence, maximum variants are proposed from Boyer-Moore (BM) algorithm. This paper addresses the variant of Boyer-Moore algorithm for finding the occurrences of a given pattern *P* within the text *T*.

INTRODUCTION

String matching algorithms [1][2] are one of most commonly used algorithms for Medical science, Network Security, Image Processing etc. The effectives of the string matching algorithms are measures in terms of time and space complexities. Time and space complexity [3] is calculated in two phases preprocessing phase and searching phase of algorithms. Preprocessing phase is calculated for the pattern *P*, which needs to be searched for (a) to check repeated characters within the pattern *P*, (b) to check unique characters within the pattern, (c) to check the position of repeated character. Preprocessing phase calculates the values that are later used in searching phase. It has been found by Lee[4] that most of the string matching algorithms are based on some rules. These rules have been analyzed in details and correlation has been found among each these rules by Jamuna et al [5]. This paper focuses on variants of BM algorithms based on these rules since maximum variant has been proposed from BM algorithms among all the original algorithms.

The organization of paper is as section 2 discussed the overview of BM algorithm. Section 3 discussed the variants of BM algorithms with rules used by them and then finally concludes with section 4.

OVERVIEW OF BOYER-MOORE ALGORITHMS

In 1977, R. S Boyer and J. S Moore [2] designed fast linear string searching algorithm. The comparison of the pattern with the text is done from *right to left*. BM algorithm is very popular and mostly used string matching algorithm and follows the rules of exact string matching techniques [4][5].

Rules of exact string matching algorithms are the mechanisms used to search for the pattern/string within the text. The Boyer-Moore algorithm constructs two preprocessing table. One is for Boyer-Moore Bad characters (BMbc) and another for Boyer-Moore good suffix (BMgs).

Then pattern is shifting based on preprocessing table values for BMbc and BMgs. For pattern searching phase it is found that Boyer-Moore algorithm observations are categorized into two shifting process BMbc and BMgs.

BMbc shifting:

Case 1: If mismatches at the very first character of pattern *P* (m^{th} position's char) with the corresponding character of text *T*. Also mismatch character of text *T* also does not appears in pattern *P* then shift the pattern with pattern length *m.* as shown in figure 1(a).

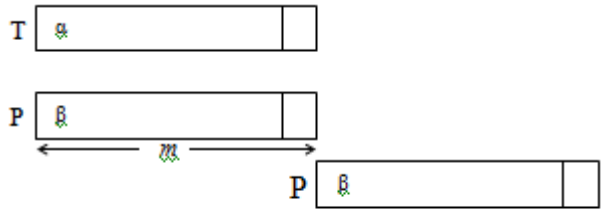

Figure 1(a): No α of text *T* is present at pattern *P*, shift by *M*

This shifting becomes same as shifting in suffix to prefix (Rule-1) Shift as if no suffix found in pattern *P*. So Boyer-Moore's first observation shift is based on Rule-1 of exact string matching algorithm.

Case 2: If the last *m'* chars of pattern *P* matches with the corresponding *m'* characters of text *T* then mismatches occurs after *m'* characters *then* move pattern by length *(m-m')* as shown in figure 1(b).

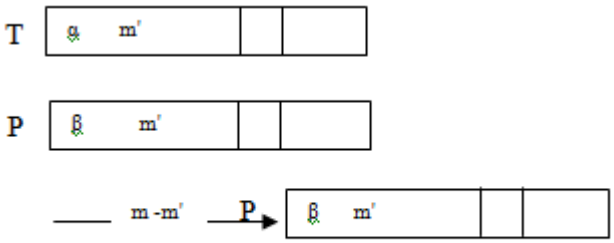

Figure 1(b): Mismatch char m' does not reoccurs in pattern *P*

This shifting becomes same as shifting of substring matching rule (Rule-2) shift as if substring does not found in pattern *P*. So Boyer-Moore's 3a observation shift is based on Rule-2 of exact string matching algorithm.

© JGRCS 2010, All Rights Reserved

BMgs shifting

Case 1: If mismatches at the very first character of pattern *P* (m^{th} position's char) with the corresponding character of text *T* but mismatch character of text *T* also appears in pattern *P*at position i^{th} then shift the pattern by $(m\text{-}i)^{th}$ number. Where, *i* is the position of matched char in pattern *P*.as shown in figure 2(a).

Figure 2(a): α of text *T* is present in pattern *P*, so shift by *m-i*

This shifting becomes same as shifting done as per bad-char rule (Rule-2.1) Shift and 1-suffix rule (Rule-2.2) shift, as if mismatch char found in pattern *P*. So Boyer-Moore's first observation shift is based on Rule-2.1 and 2.2 of exact string matching algorithm.

Case 2: If the last *m'* chars of pattern *P* mismatches with the corresponding *m'* characters of text *T* but *m'* characters reoccurs in pattern *P* then shift the pattern by aligning the matched chars of text and pattern as shown in figure 2(b).

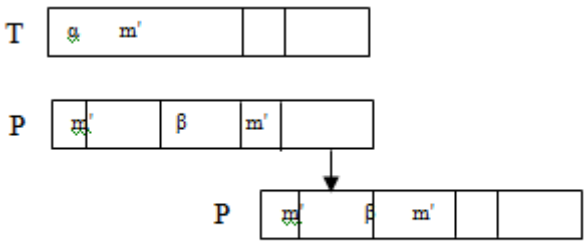

Figure 2(b): Mismatch char m' reoccurs in pattern *P*

This way BM algorithm introduced the four ways for This shifting becomes same as shifting done as per substring matching rule (Rule-2) Shift and 1-suffix rule (Rule-2.2) shift, as if mismatches *m'* characters found in pattern *P*.

VARIANTS OF BOYER-MOORE ALGORITHM

Based on Boyer-Moore pattern searching methods, many variants have been introduced. The most popular and effectively used variants of Boyer-Moore algorithms are Horspool algorithm, Zhu Takaoka algorithm, Quick search algorithm, Tuned BM algorithm, Reverse Cloussi algorithm, Raita Algorithm, Fast search algorithm, Forward-fast-search algorithm.

This section discussed the working procedure of variants algorithms. This includes the rule followed by the variants and complexities of preprocessing and searching phase are also mentioned. (Wherein calculated complexities, *m* is the length of the pattern, *n* is the length of the text *T* and σ is the number of alphabets in pattern *P*).

a. ***Horspool algorithm:*** Horspool[6] proposed variant of Boyer-Moore Practical fast searching in strings in 1980uses Rule-2.2.

Working Principal of Horspool algorithm:

This algorithm implements the concept of bad character. Horspool scan the text *T* with corresponding pattern *P* from right to left. If mismatch occurs (found bad character) at any position of pattern then search the reoccurrence of only the first character of text window (from right) within the pattern *P*. This shows Horspool applies the 1-suffix rule (Rule-2.2) shift. The Horspool algorithm improved the Boyer-Moore algorithm by addressing the occurrence table (preprocessing table) to compute the values of the reoccurrences of characters.

Complexities of Horspool algorithm:

Horspool shows preprocessing phase with $O(m+\sigma)$ time complexity and $O(\sigma)$ space complexity; searching phase in $O(mn)$ time complexity.

a. ***The Zhu Takaoka algorithm:*** Zhu-Takaoka proposed an algorithm [7] "On improving the average case of the Boyer-Moore string matching algorithm, 1987", uses Rule 2.

Working principle of Zhu-Takaoka(ZT):

The comparison in *ZT* algorithm performs scanning of pattern and text in similar manner as Boyer-Moore algorithm from right to left. The *Zhu-Takaoka* uses good suffix rule, since it search 2-substing (2-substring contains two characters, one should be mismatch char followed by a match char). In Zhu-Takaoka algorithm whenever a mismatch or a complete match occurs, it select the 2-substring in *T* and search for the reoccurrence of 2-substring within the pattern *P*. If matched found or not found, shift pattern as per Rule-2.

Complexities of ZT algorithm:

The ZT algorithm construct *2D*bad char preprocessing table since it computes for the pair of characters. The complexities in preprocessing phase of *ZT* is $O(m+\sigma^2)$ time and space complexity whereas searching phase in $O(mn)$ time complexity.

a. ***Quick search algorithm:*** Daniel M. Sunday proposed a new algorithm [8] 'a very fast substring search algorithm', 1990 uses Rule-2.2.

Working Principle of Quick Search algorithm:

The Quick-Search algorithm is also known as Sunday algorithm, this is one of very simple variation of fast Horspool algorithm. After each attempt in Sunday algorithm, the shift is computed according to the character which immediately follows the current window of the text *T*. This algorithm does not depended on scanning order of the pattern like in KMP or BM algorithm. Sunday computes the bad char for the pattern string shift at each stage and follows 1-suffix shift. This is a simple and fast algorithm because it quickly debugged and quickly executed.

Complexities of Quick Search algorithm:

Preprocessing phase in Sunday algorithm as $O(m+\sigma)$ time and $O(\sigma)$ space complexity; searching phase is $O(mn)$ time complexity.

© JGRCS 2010, All Rights Reserved

a. Tuned BM algorithm: Hume A. and Sunday D.M. proposed [9] "Fast string searching", 1991 using Rule-2.2.

Working principle of Tuned BM:

Tuned BM algorithm is also based on bad char concept and shifting is done according to 1-suffix shift. Whenever mismatch occurs it looks for presence 1-suffix character within the pattern and shift is done with the nearest matched found.

Complexities of Quick Search algorithm:

Preprocessing phase in TuBMbc $O(m+\sigma)$ time complexity and $O(\sigma)$is space complexity, searching phase in $O(mn)$ time complexity.

a. Reverse Cloussi: Colussi, L proposed [10] "Fastest pattern matching in strings", 1994 using Rule-1 and Rule-2.

Working principle of Reverse Colussi algorithm:

The Reverse Colussi algorithm is in the spirit of the original Colussi Algorithm, which is one of the variant of KMP algorithm but Reverse Cloussi modified the bad character rule from matching a pair of characters. Reverse Colussi algorithm divides the pattern into two halves with special position and non-special position. Special position allows smaller number of skip. The Reverse Colussi algorithm routes the special position first. Special position allows the smaller number of steps to shift than non-special points. A special position follows theRule-2 for shifting, for any substring (according to rule-2) in Text *T* finds a nearest substring within the pattern *P*. If substring found then shift the pattern *P* in such a way that the two substring chars align correspondingly otherwise, define a new partial text window. For non-special positions, Rule-1 (suffix to prefix rule) is used for shifting of pattern, there must be suffix of the text window matched with the prefix of the pattern *P*.

Complexities of Reverse Cloussi algorithm:

Reverse Cloussi preprocessing time and space complexity is shown as $O(m+\sigma)$. Searching phase time complexity is $O(mn.)$

a. Raita Algorithm: T. Raita proposed an algorithm [11] "Tuning the Boyer-Moore-Horspool string searching algorithm", 1994 uses Rule-2.2.

Working principle of Raita algorithm:

Raita algorithm is a simple modification of Horspool algorithm. Raita first compare the last character of text window *T* with the last character of the pattern *P*, and then compare the first characters of text and pattern and the middle characters. If they match, then compare other characters from left to right. If mismatch occurs, slide the window by the 1-suffix rule.

a. Complexities of Raita algorithm: The preprocessing time complexity of Raita is $O(m+\sigma)$, space complexity is $O(\sigma)$ and searching time complexity is O(mn).

b. Fast search (FS) algorithm: Cantone D and Faro S. designed [12] "A new efficient variant of the BM string matching algorithm", 2003 uses Rule-2.1 and Rule 1.

Working principle of FS algorithm:

The FS algorithm is a fast pattern searching variant of the Boyer-Moore string matching. Fast search applied the bad character rule only if the mismatch char is the last character of the pattern, otherwise the good suffix rule is used. The observations [13] in which the Fast search algorithm carried out shifting of pattern are

(a). Bad character leads to larger shift than good suffix if and only if a mismatch occurs immediately, while comparing the characters of pattern with the text characters.

(b). Repeating the bad char rule until the last char of pattern$[m-1]$ of the pattern is matched correctly against the text, and then apply the good suffix rule, at the end of matching

Fast search is compared with other variants of BM algorithms such as the Horspool algorithm [6], Quick Search algorithm [8], Tuned BM algorithm [9], and Reverse Factor algorithms[13]. Fast Search algorithm meets the very good results, especially if the length of the patterns is short.

Complexities of Fast-Search algorithm:

Fast search algorithm the pre-computed bad-char and good suffix with time complexity O(m) and O(m+σ) since Fast search follows both the bad-char as well as good suffix.

a. Forward-fast-search using rule: Another variant of the BM string matching [14] algorithm is Forward-fast-search.

b. *Working principle of Forward Fast-Search*: The method used in Forward-fast-search is same as the Fast-Search algorithm, but it is based on a modified version of the good suffix rule, forward good suffix refer to look forward character to calculate the larger shift.

In forward good suffix, if first mismatched found at i^{th} position of the pattern *P*, the forward good suffix applied to align sub text *T* with the character of rightmost occurrence in pattern *P* preceded by a char different from *P[i]*. If no such occurrence found then the backward good suffix rule introduced (i.e a shift increment which allow matching the maximum suffix of text *T* with a prefix of *P*.)

Forward fast search algorithm obtained as same as Fast-Search and Tuned BM algorithm. However Forward-Fast-Search [16] implemented using the forward good suffix rule only.

The Forward-fast-search also compared with the Horspool [6], Quick Search algorithm[8], Tuned BM algorithm[8], and Reverse Factor algorithms[13], Berry-Ravindran[15] and also with Fast-Search algorithm[12] itself.

Over all it observed Forward-Fast-Search is one of best algorithm in practice.

***c. Complexity of forward good suffix*:** Forward good suffix needed a preprocessing table of length (m*σ) and gives complexity as O(m*σ).

CONCLUSIONS

This study concludes that each of the variant follows either the concept of bad character shift, good suffix shifts or both. After analyzing the proposed variants of BM algorithms, it is observed that most of the variant uses the concept of bad

© JGRCS 2010, All Rights Reserved

character shifts since they initiate the maximum number of shifting with bad char concept. All the variants perform scanning of pattern from right to left except Quick search algorithm. Quick search algorithm does not depend on scanning order, scanning can be performed in any manner. It is also being studied most of the variants adopt the bad character rule and good suffix rule. The algorithms are compared in terms of efficiency of run-time, number of character compared.

ACKNOWLEDGEMENT

This work is partially supported by HRD, Govt. of Sikkim (India), vide notification no. 166/SCH/EDN 2003, Ref. No 82/SCH/EDN, issued on 20/7/2013.

REFERENCES

[1] Knuth, D. E., Morris, JR, J. H., and Pratt, V. R. 1977. Fast pattern matching in strings. SIAM J. Comput. 6, 1, 323–350, 1977

[2] Boyer, R. S. and Moore, J. S.1977, A fast string searching algorithm. Commun. ACM, 1977, 20, 762–772.

[3] LivioColussi, ZviGalilt, Raffaele Giancarlo. On The Exact Complexity Of String Matching. CH2925-6/90/0000/01, IEEE,1990

[4] C. W. Lu, C. L. Lu, R.C.T. Lee, Exact String matching rules for algorithms", http://alg.csie.ncnu.edu.tw/lecture_notes_stringmatching.php.

[5] Jamuna Bhandari and Anil Kumar, Analysis of Various Rules of Exact String Matching Algorithms, International Journal of Applied Research and Studies, 2278-9480 Volume 2, Issue 10, 2013.

[6] Horspool, R. N. 1980. Practical fast searching in strings. Softw. Pract. Exp. 10, 6, 501–506

[7] Zhu, R. F. and Takaoka, T. 1987. On improving the average case of the Boyer-Moore string matching algorithm. J. Inform. Process. 10, 3, 173–177.

[8] Sunday, D. M. 1990. A very fast substring search algorithm. Commun. ACM 33, 8, 132–142. University, Prague, Czech Republic, 16–28. Collaborative Report DC–99–05

[9] Hume, A. and Sunday, D. M. 1991. Fast string searching. Softw. Pract. Exp. 21, 11, 1221–1248.

[10] Colussi, L. 1994. Fastest pattern matching in strings. J. Algorithms 16, 2, 163–189

[11] Raita, T. 1992. Tuning the Boyer-Moore-Horspool string searching algorithm. Softw. Pract. Exp. 22, 10, 879–884

[12] Cantone, D. and Faro, S. 2003. Fast-Search: a new efficient variant of the Boyer-Moore string matching algorithm. In WEA 2003. Lecture Notes in Computer Science, vol. 2647. Springer-Verlag, Berlin, 247–267

[13] Crochemore, M. et al, Speeding up two string matching algorithms. Algorithmica 12, 4/5, 247–267, 1994.

[14] Cantone, D. and Faro, S. 2003b. Forward-Fast-Search: another fast variant of the Boyer-Moore string matching algorithm. In Proceedings of the Prague Stringology Conference '03, Czech Technical University, Prague, Czech Republic, 10–24.

[15] Berry, T. and Ravindran, S. 1999. A fast string matching algorithm and experimental results. In Proceedings of the Prague Stringology Club '99, J. Holub and M. Sim ´anek, Eds. Czech Technical

[16] Crochemore, M. and Lecroq, T. 2008. A fast implementation of the Boyer-Moore string matching algorithm.

[17] Franek, F., J Ennings, C. G., and Smyth, W. F. 2007. A simple fast hybrid pattern-matching algorithm. J. Discret Algorithms 5, 4, 682–695

© JGRCS 2010, All Rights Reserved