International Journal of Computer Applications (0975 – 8887)
Volume 72– No.1, May 2013

Performance Improvement of BMH and BMHS using PDJ (Possible Double Jump) and MValue (Match Value)

Akhtar Rasool
Department of Computer Science Engineering
Maulana Azad National Institute of Technology,
Bhopal-462051, India

Nilay Khare
Department of Computer Science Engineering
Maulana Azad National Institute of Technology,
Bhopal-462051, India

ABSTRACT

BM (Boyer-Moore) string matching algorithm and its enhanced versions like BMH and BMHS are a standard benchmark for single pattern string matching. Based on BMH and BMHS algorithms we have developed improved algorithms *EBMH(Enhanced BMH*)and *EBMHS(Enhanced BMHS)*. These algorithms uses the newly introduced PDJ (possible double jump) and MValue(Match Value) concepts. While searching these concepts helps to provide longer jump of characters. The algorithm *EBMH* emphasize on BMH algorithm with the inclusion of PDJ and MValue and the algorithm *EBMHS* emphasize on BMHS algorithm with the inclusion of PDJ and MValue. Through these algorithm the number of comparison of characters between text and pattern are reduced to a significant amount. In this paper PDJ, MValue, EBMH and EBMHS are described and analyzed. Experimental results show that in the algorithms searching time is reduced as compared to BM, BMH, and BMHS. The algorithms are analyzed on the basis of time requirement in best, worst and average case.

General Terms

String Matching Algorithms, Boyer-Moore.

Keywords

String Matching; BM; BMH; BMHS; Improved BMHS; PDJ; MValue; EBMH; EBMHS.

1. INTRODUCTION

String matching is to find the specific pattern string in the large text string . String matching plays an important role in many algorithms of computer science such as in intrusion detection algorithms, plagiarism detection algorithms, bioinformatics, digital forensics, text mining and video retrieval etc.[1,2,3,4]

Boyer Moore & its variant compare the characters from right to left in the pattern. In BM algorithm the concept of good-suffix and bad character is used to calculate jump for skipping of characters in case of mismatch. In BMH algorithm a character of text corresponding to last character of pattern is used to calculate jump[5,6,7,10,11]. The BMHS algorithm uses next-to-last character for calculating jump as a bad character rule. There are many algorithms which uses a combination of both for calculation of jump. In BMHS2 algorithm last character and next-to-last character are used for calculation of jump. In improved BMHS algorithm next-to-last and next-to-next-to-last character calculates jump as a bad character rule[12,13,14,15,16]. In this paper EBMH (Enhanced BMH) and EBMHS (Enhanced BMHS) algorithms are described. These algorithms uses newly introduced concept PDJ (Possible double jump) and MValue (Match Value).

2. BM STRING MATCHING ALGORITHM AND ITS IMPROVEMENT AS BMH AND BMHS

2.1 BM Algorithm

The BM algorithm scans the characters of the pattern from right to left. In case of a mismatch (or a complete match of the whole pattern) it uses two pre-computed functions, Good suffix and Bad character to shift the matching window to the right. The window is shifted to right by maximum of these two values .[5,8]

	P	A	T	T	E	R	N	M	A	T	C	H	T	O	F	I	N	D	T	E	M	P	T	E	X	T
0	T	E	X	T																						
1				T	E	X	T																			
2								T	E	X	T															
3												T	E	X	T											
4																T	E	X	T							
5																			T	E	X	T				
6																							T	E	X	T

Fig 1.Example BM: 6 Shifts and 12 Comparisons

2.2 BMH Algorithm

In BMH algorithm only bad character is consider to calculate shift or jump. Here the position of mismatch is not important. The shift is determined by the character in text string which is aligned to the last character of the pattern string[6,9].

	P	A	T	T	E	R	N	M	A	T	C	H	T	O	F	I	N	D	T	E	M	P	T	E	X	T
0	T	E	X	T																						
1				T	E	X	T																			
2								T	E	X	T															
3												T	E	X	T											
4																T	E	X	T							
5																			T	E	X	T				
6																							T	E	X	T

Fig 2.Example BMH: 6 Shifts and 12 Comparisons

2.3 BMHS Algorithm: In BMHS Algorithm Next-to-Last character of text corresponding to patterns last character is considered for calculation of shift using bad Character[10,11]. BMHS example is given below

	P	A	T	T	E	R	N	M	A	T	C	H	T	O	F	I	N	D	T	E	M	P	T	E	X	T
0	T	E	X	T																						
1				T	E	X	T																			
2									T	E	X	T														
3										T	E	X	T													
4															T	E	X	T								
5																T	E	X	T							
6																			T	E	X	T				
7																				T	E	X	T			
8																							T	E	X	T

Fig 3.Example BMHS: 8 Shifts and 16 Comparisons

3. PDJ(Possible Double Jump) AND MValue(Match Value)

3.1 PDJ

3.1 PDJ : In PDJ two jumps are calculated. First jump is calculated in the same manner as in BMH. The next jump is calculated using the next-to-last bad character value of the character obtained as a result of first jump. For example consider the text "ABCEDFRWXY" and the pattern "TEXT". In the figure 4 below there is a mismatch between E and T, therefore the shift calculated using E is 2. Than the shift on F is 4. Aggregative the shift calculated using two characters E and F is 6.which is the concept of PDJ.

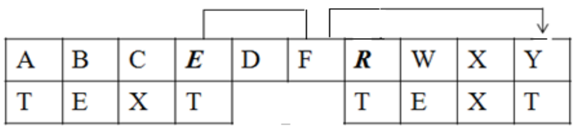


Fig. 4: PDJ Logic

3.2 MValue

MValue is the position of last character of pattern from right other than last position. If last character is not found again in the pattern than it is equal to the pattern length. MValue concept is used when mismatch occurs in between or fully matched. The double jump is calculated without additional processing of shifts as compared to BMH, as we know that the last character is matched in this condition. MValue is calculated once and used in the condition of a in-between mismatch or fully matched whenever required to obtain double jump in single calculation. We have Text: *ABCTDFRWXYZ* and Pattern: *TEXT* In this mismatch is occur in between so calculate shift using a character which is three character apart from last character of pattern because for "TEXT" MValue is 3. In figure 5 we explain PDJ Logic using MValue.

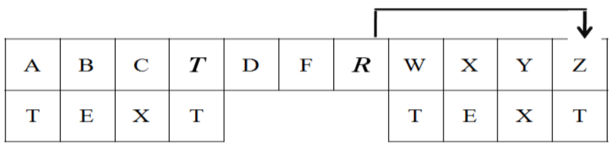


Fig. 5: PDJ Logic using Mvalue

4. IMPROVED ALGORITHM: EBMH

The algorithm is improvement of BMH algorithm using PDJ and MValue. The main concept of algorithm is to reduce the number of comparisons by calculating possible double jump in one shift with the help of bad character of BMH algorithm. EBMH algorithm contains two phases preprocessing phase and searching phase.

4.1 Preprocessing Phase

Calculate last-bad-character, next-to-last bad character and MValue of the pattern. For example suppose Text String: “PATTERNMATCHTOFINDTEMPTEXT” and Pattern String: “TEXT”. Than values after preprocessing phase is

Table 1: Last-bad character

Last Bad Character	
Character	**Jump**
X	1
E	2
T	3
Others	4

Table 2: Next-to-last bad character

Next-to-Last bad character	
Character	**Jump**
T	0
X	1
E	2
Others	4

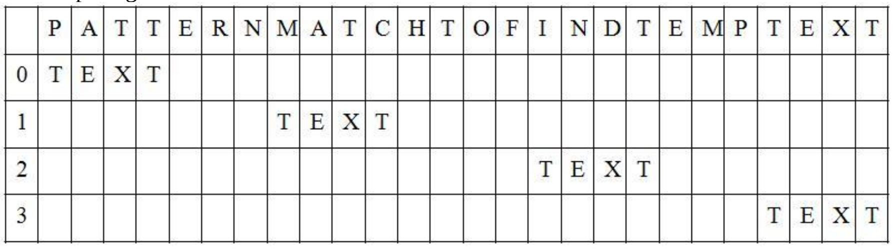
MValue = 3

4.2 Searching Phase

Matching logic is divided in to two parts. First mismatch at last or second mismatch in-between or fully match condition. In the first matching logic while comparing from the last character if there is a mismatch at last character, we calculate first jump value corresponding to last character by bad character. The next jump is calculated using the next-to-last bad character value of the character obtained as a result of first jump. Than the matching window is shifted to right by the sum of these two values. This will provides double shifting value if possibility is there. In the second matching logic if mismatch occurs in-between or fully matched the jump is calculated using a character which is MValue apart from last character of pattern. The window is now shifted to right by the sum of MValue and calculated jump by MValue.

***EBMH* algorithm matching logic:**

```
If( Mismatch at the last character in pattern )
{    j= corresponding character in text with respond to last character of the pattern.
        jump1 = Last-Bad-Character[j];
        jump2 = Next-to-Last-Bad-Character[character at "current mismatch location+jump1"];
        Total Shift = jump1+jump2;
}
Else{   // case of in-between mismatch or fully matched case
   jump1 = MValue of Pattern;
   jump2 = Next-to-Last-Bad-Character[character at "current mismatch location+jump1"];
   Total Shift = jump1+jump2;
}
```



	P	A	T	T	E	R	N	M	A	T	C	H	T	O	F	I	N	D	T	E	M	P	T	E	X	T
0	T	E	X	T																						
1								T	E	X	T															
2																T	E	X	T							
3																							T	E	X	T

Fig 6.Example EBMH: 3 Shifts and 9 Comparisons

Here Compare P [3] =T with S [3] =T. A match is found, therefore we compare P [2] =X with S [2] =T. A mismatch is encountered, therefore jump value is calculated using a character which is MValue distant from last character that is ‘N’. For N we calculate jump using Next-to-last bad character table which 4. Therefore PDJ is 3+4 = 7. After shifting by 7, ‘C’ of text string is aligned with last character of pattern that is ‘T’. Compare P [3] =T with S [10] =C. A mismatch is encountered therefore first jump value corresponding to ‘C’ is 4 and second jump value using ‘F’ is 4, therefore final jump is 4+4=8. After shifting by 8, ‘T’ of text string is aligned with last character of pattern that is ‘T’. Comparing P [3] =T with S [18] =T. A match is reported, therefore compare P[2]=X with S[17]=D. A mismatch is found. Therefore jump value is calculated using a character which is MValue distant from last character that ‘P’. For ‘P’ we calculate jump using Next-to-last bad character table which 4. Therefore PDJ is 3+4 = 7. After shifting by 7, ‘T’ of text string is aligned with last character of pattern that is ‘T’. Comparing P [3] =T with S [25] =T. Matches repeatedly occur till the beginning of the window, reporting the occurrence of pattern in the text. In Figure 6 example of EBMH is shown.

5. IMPROVED ALGORITHM: EBMHS

The EBMHS is further improvement of EBMHS using the concept of BMHS algorithm. EBMHS is the same as EBMH except that in the condition of mismatch at last character, it uses next-to-last character instead of last character for calculation of jump as in BMHS algorithm. If the shift value is 1, then we shift by that value otherwise calculate second jump using the jump value obtained from first jump and then the final shift is calculated by sum of these two values. This algorithm performs better than EBMH when Next-to-last character is not found in pattern but last character is present in pattern.

EBMHS Algorithm Matching logic:

```
If( Mismatch at the last character in pattern )
{
 j= next character to the corresponding character in text with respond to last character of the pattern.
  shift =1+Next-to-Lastbadcharacter[j];
   If (shift ==1)
   {
        Total Shift = 1;
   }
   Else {
        Total Shift= Shift + Next-to-Last-Bad-Character[character at "current mismatch location + Shift"];
        }
}
Else{// case of in-between mismatch or fully matched case
   jump1 = MValue of Pattern
   jump2 = Next-to-Last-Bad-Character[character at "current mismatch location+jump1"];
   Total Shift = jump1+jump2;
}
```

	P	A	T	T	E	R	N	M	A	T	C	H	T	O	F	I	N	D	T	E	M	P	T	E	X	T
0	T	E	X	T																						
1								T	E	X	T															
2																	T	E	X	T						
3																							T	E	X	T

Fig 7. Example EBMHS: 3 Shifts and 8 Comparisons

6. Analysis And Experimental Results

6.1 Analysis Based on Example

For the above example number of shifts and comparisons of different algorithms are shown in table 8. BM and BMH require 6 shifts and 12 comparisons, BMHS require 8 shifts and 16 comparison. *EBMH* requires 3 shift 9 comparisons. *EBMHS* requires 3 shifts and 8 comparisons.

Table 3: Experimental Results

Algorithm\Parameter	Number of Shifts	Number of Comparisons
BM	6	12
BMH	6	12
BMHS	8	16
EBMH	3	9
EBMHS	3	8

6.2 Analysis of Time Complexity

Suppose we have text set Q_t, $Q_t = \{x_0x_1x_2.......x_{n-1}\}$, where n is number of distinct character in text.

And pattern set Q_p,$Q_p = \{y_0,y_1,y_2..........Y_{m-1}\}$, where m is number of distinct character in pattern.

EBMH: ***Best Case I:*** Conditions for best case are always mismatch at last character, PDJ= 2 Pattern Length. In this best case the characters of the pattern and characters of the text are drawn from two disjoint sets. Here $Q_t \cap Q_p$=NULL, So at each iteration a jump of twice of pattern length is achieved. Therefore in this best case algorithm has complexity of O (n/2m +C_{PDJ}) Where, C_{PDJ} is a constant of overhead of calculating PDJ.

Best Case II: Conditions for second best case are mismatch at second last character, MValue = pattern length and PDJ = 2 Pattern Length. For this best case only one character of pattern at each iteration matches with text character, which is last character of pattern. Here $Q_t \cap Q_p$ = singleton set .So at each iteration double jump is achieved. Therefore this algorithm has complexity O (n / 2m),Here C_{PDJ} = 0, because of the use of MValue, PDJ is achieved in single calculation.

Average Case: Conditions for average case are mismatch other than last and second last character, MValue lies between 1 and pattern length and PDJ $_{second}$ is less than pattern length. Here $Q_t \cap Q_p$= finite set, means it is condition of mismatch in between, therefore at each iteration a jump of (MValue + PDJ_{second}) is achieved. The algorithm has average case complexity of O (n/average jump), where average jump is $^{k}\sum_{i=1}(MValue_i + PDJ_i)$ / k, where k is number of total jumps.

Worst Case: Conditions for worst case are mismatch at first character, MValue = 1 and PDJ $_{second}$ = 0. Here all character of pattern matches except the character present in beginning of window. $Q_t \cap Q_p=Q_p$ So at each iteration only single jump is achieved. The algorithm has worst case complexity of O (nm).

EBMHS: ***Best Case I:*** Conditions for best case are mismatch at last character and PDJ = 2 pattern length**.** Here, we use BMHS for first jump in PDJ and that in best case is pattern length +1. Therefore at each iteration a jump of one more than twice of pattern length is achieved. The algorithm has best case time complexity of O(n/2m+1 + C_{PDJ}). Where, C_{PDJ} is a constant of overhead of calculating PDJ.

Best Case II: Conditions for best case are mismatch at second last character, MValue = pattern length and PDJ = 2 Pattern Length. complexity of EBMHS is same as EBMH in this case, which is O (n / 2m).

Average Case: Complexity of EBMHS is same as EBMH in average case, which is O (n/average jump), where average jump is $^{k}\sum_{i=1}$(MValue$_i$ + PDJ$_i$) / k, where k is number of total jumps.

Worst Case: Complexity of EBMHS is same as EBMH in worst case, which is O (nm).

6.3 Analysis Based on Experiment

Experimental Environment: Processor: i7**,** RAM: 8 GB**,** OS: windows 7**,**Language: visual C++ runs on visual studios 2008**.**

Experimental Data: 50 different text files of size 767 MB (80, 52, 83,893 bytes) approximately, having large number of occurrences of pattern and pattern of length 10.

Experiment: Based on the experiment performed on above text and pattern using BM, BMH, BMHS, *EBMH* and *EBMHS*, the results are shown in table 4.

Table 4: Experimental Results

Algorithm\Parameter	Number of Comparisons	Time(milisecond)
BM	184416143	395
BMH	172121737	400
BMHS	153680129	470
EBMH	135238505	290
EBMHS	129091301	295

Graphical analysis for number of comparison performed by different algorithms and searching time taken by different algorithm is shown in figure 8 and figure 9 below.

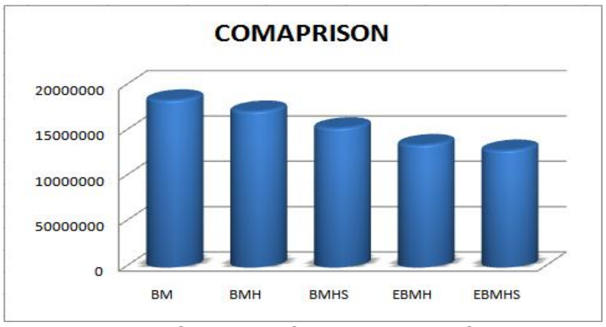


Fig.8: graph of number of comparison performed by different algorithms

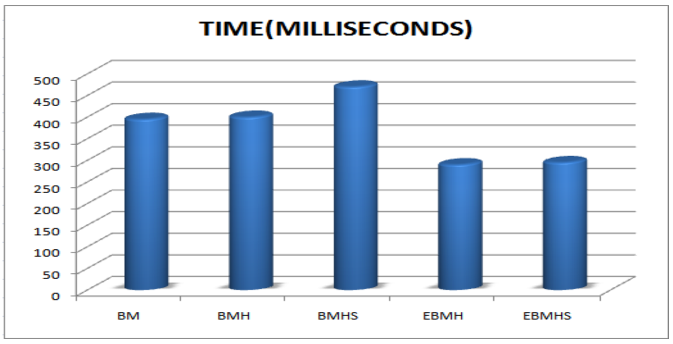


Fig.9: Graph of searching time taken by different algorithm

6.4 Experimental comparison of complexity of Proposed Algorithms

Here we take experimental results for best case, average case and worst case conditions of EBMH and EBMHS.

Experimental Data: Experiments are performed on text and pattern file for four different conditions of performance as best case I, best case II, average case and worst case. The size of each of all four cases text files to be search is 251 MB and pattern length of all pattern are 10.

Experimental Results: Experimental results of all four cases for EBMH and EBMHS are shown in table 10 below.

Table 5: Experimental Results

ALGORITHM	SEARCH TIME(Millisecond)			
	BEST CASE 1	BEST CASE 2	AVERAGE CASE	WORST CASE
EBMH	110	95	140	780
EBMHS	100	93	170	670

Graphical analysis of four cases for EBMH and EBMHS are shown in figure 10.

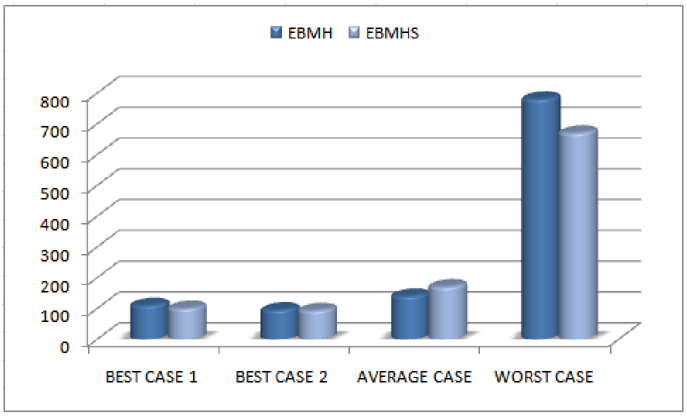


Fig. 10: graph of Comparison of different complexity case using experiment

7.CONCLUSION

BMG and BMHS provide better performance in string matching. Experimental results and mathematical analysis shows that EBMH performs better than BM, BMH and BMHS as number of comparison performed and search time both are less. Another improved algorithm EBMHS is also better than BM, BMH and BMHS and performs almost similar to EBMH. The performance improvement is the consequence of the use of PDJ and MValue concept. MValue makes PDJ more efficient in case of mismatch in-between or fully matched. Using MValue and PDJ performance of BMH and BMHS algorithm has been drastically improved.

8.REFERENCES

[1] Pei-fei Wu, “The Research and Amelioration of Pattern-matching Algorithm in Intrusion Detection System”, In the proc. IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), June 2012, pp. 1712-1715.

[2] Qingduan Meng, Xiaoling Zhang, Dongwei Lv, “Research on Detection Speed Improvement of Snort”, In the proc. Internet Technology and Applications, Aug. 2010, pp. 1-5.

[3] Lin, Pi-yuan, Shaopeng Liu, Lixia Zhang, Peijie Huang, “Compressed Pattern Matching in DNA Sequences Using Multithreaded Technology”, In the proc. 3rd International Conference on Bioinformatics and Biomedical Engineering ,ICBBE 2009, June 2009, pp. 1-4.

[4] Pei-fei Wu, “The Research and Amelioration of Pattern-matching Algorithm in Intrusion Detection System” , In the proc. 9th International Conference on Embedded Software and Systems (HPCC-ICESS), June 2012, pp. 1712-1715

[5] Jingbo Yuan, Jinsong Yang, Shunli Ding, “An Improved Pattern Matching Algorithm Based on BMHS”, In the proc. 11th International Conference on Distributed Computing and Applications to Business, Engineering & Science (DCABES), Oct. 2012, pp. 441-445.

[6] Yuting Han, Guoai Xu, “Improved algorithm of pattern matching based on BMHS”, In proc. IEEE International Conference on Information Theory and Information Security (ICITIS), Dec. 2010, pp 238-241.

[7] Liu Zhen, Xu Su, Zhang Jue,“Improved Algorithm of Pattern Matching for Intrusion Detection” , In proc. International Conference on Multimedia Information Networking and Security, Nov. 2009, pp. 446-449.

[8] Yuting Han, Guoai Xu, ”Improved Algorithm of Pattern Matching based on BMHS”, In proc. IEEE International Conference on Information Theory and Information Security (ICITIS), Dec. 2010, pp. 238-241.

[9] Lin quan Xie, Xiao ming Liu, Guangxue Yue, “Improved Pattern Matching Algorithm of BMHS”, In proc. International Conference on Information Science and Engineering (ISISE), Dec. 2010, pp. 616-619.

[10] R. S. Boyer, and J. S. Moore, “A Fast String Searching Algorithm”, Communications of the ACM, 1977, 20(10):762-772.

[11] R. N. Horspool, “Practical Fast Searching in Strings”, Software Practice and Experience, 1980, 10(6):501-506.

[12] Yuan, Lingling, “An improved algorithm for boyer-moore string matching in chinese information processing”, In proc. International Conference on Computer Science and Service System (CSSS), June 2011, pp. 182-184.

[13] Xiong, Zhengda, “A Composite Boyer-Moore Algorithm for the String Matching Problem ”, In proc. International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Dec. 2010, pp. 492-496.

[14] Qingduan Meng, Xiaoling Zhang, Dongwei Lv, “Improved AC_BMH Algorithm for String Matching”, In the proc. Internet Technology and Applications, Aug. 2010, pp. 1-5.

[15] Yong-qiang, Zhu, “Two Enhanced BM Algorithm in Pattern Matching”, In proc Conference on Digital Media and Digital Content Management (DMDCM) , May 2011, pp. 341-346.