Efficient Multiple Pattern Matching Algorithm Based on BMH: MP-BMH

Akhtar Rasool[1], Gulfishan Firdose Ahmed[2], Raju Barskar[3], and Nilay Khare[1]
[1]Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, India
[2]Department of Computer Science, College of Agriculture, India
[3]Department of Computer Science and Engineering, University Institute of Technology RGPV, India

Abstract: *String matching is playing a key role in a wide range of applications in the information computing. Due to its importance, large numbers of different algorithms have been developed over the last 50 years. There are various standards of single and multiple pattern string matching algorithms like Boyer-Moore (BM), Boyer-Moore-Horspool (BMH), Aho-Corasick etc. Multiple pattern string matching is very useful in real world applications. Standard benchmark multiple pattern algorithm Aho-Corasick and many other multiple pattern string matching algorithms are not memory and time efficient for wider length and large number of patterns set on large text data set because they are using the concept like DFA and require full scan of text data. Many string matching tools which are currently popular for string matching are based on these algorithms. Using the bad character principle of BMH, a method for multiple pattern string matching is being developed. Using this method a time and memory efficient exact multiple pattern string matching algorithm Multiple Pattern BMH (MP-BMH) is proposed. Unlike Aho-Corasick, the proffered MP-BMH algorithm provides skipping of unnecessary matching of characters in text while searching like BMH Algorithm. It also provides the skipping of characters in patterns using the similarity between patterns. Along with the shifts while searching, this algorithm also provides shrewd switches among the patterns for efficacious matching.In this paper, the aforesaid method, MP-BMH algorithm with its time, memory and experimental analysis are described.*

Keywords: *String matching; multiple pattern string matching, Boyer-Moore BM,BMH, MP-BMH.*

Received January 8, 2017; accepted January 16, 2018

1. Introduction

String matching is one of the important concepts for solving problems in computer science. In the multi-pattern string matching algorithm [9, 35, 39] all occurrences of the desired patterns in the text string are searched. The taxonomy and relative performance measures of multi-pattern matching algorithms are mentioned in [28, 51, 53]. String Matching has many real world applications like intrusion detection system [28, 51, 53], plagiarism detection [3], text mining techniques [35], digital forensics technique [31] and many more. An effective multiple string matching algorithm can boost the performance of these applications as these applications are widely used in real world. An effective selection of multiple string matching algorithms [29] can improve the efficiency of these real world applications. The most important benefit of multiple string matching algorithms is that no additional search structure is needed. Only one pass of searching is sufficient to search multiple patterns or keywords from a given text. Knowledge of single pattern string matching could help in better understanding of the multiple pattern string matching algorithms.

2. Related Work

There are various popular single pattern string matching algorithms such as Boyer-Moore (BM) [45, 47], Knutt Morris Pratt (KMP) [49, 50], Rabin Karp[51], Boyer-Moore-Horspool (BMH) [27, 42, 44, 50], BMHS [30, 34], BMHS2 [47], BMI [53], Improved BMHS [44] and Backward Non-Deterministic Dawg Matching (BNDM) [39] and Two way Non-Deterministic Dawg Matching (TNDM) [49]. These algorithm are simple but time consuming so move to multi-pattern string matching[16, 49] Some of the multiple pattern string matching algorithms subsume Multi-pattern string matching with q-grams[29, 50], a fast algorithm for multi-pattern searching [46], Wu Manber [31, 36, 48],commentz-walter [22] and standard benchmark algorithm Aho-Corasick [11, 12, 13, 15, 17, 18, 19, 20, 21, 22, 23, 24, 30, 32, 33, 34, 38, 40, 41, 43]. Boyer-Moore-Horspool (BMH) [50] algorithm uses the concept of Bad Character [14] of BM algorithm [47] for shifting through text in matching process. In BMH algorithm, no matter the location of mismatching, the distance of shift to right is determined by the character in the text string which is aligned to the last character of pattern string. It means here always the Bad Character value

[14] of a character of text corresponding to last character of pattern is calculated. And then the window is shifted right by that value. BMH always provides the jump at least one and maximum of pattern length [50]. In [50] they applied the BMH algorithm for multiple patterns in a different way using q-grams. They called their proposed filtering algorithm as Horspool with q-Grams (HG) algorithm. The HG algorithm performs faster for less than 2000 patterns and works well for less than 10000 patterns. Here full text is scanned when the pattern set contains more than 50000 patterns. This algorithm is a filtering method and so is not designed for searching texts with large number of matches.

In [46] extended the BM algorithm for handling multiple patterns. They proposed two new algorithms whose operations involves remembrance of previous matches. With N as text string length and maximum pattern length D, the first proposed algorithm remembers maximum $1+ \log_4 D$ previous matches and consults O(N log D) text characters. The second algorithm, being designed for distinct time-space trade-off, consults O (kN log D) text characters and remembers maximum t/k non periodic matches for any given k and t number of patterns.

Aho-Corasick [42] is standard multi-pattern string matching algorithm [9] which is based on finite automata. The complexity of Aho-Corasick (AC) [2] algorithm is in linear time, which scans on character by character basis in the text by creating finite state machine from the patterns given at the beginning. By taking basis of that work, Commentz-Walter [48] have provided almost linear solution by combining Boyer-Moore's [47] matching with the AC machine. In addition to the approaches of automata, researchers have also considered bit parallelism approach (e.g., shift-or algorithm [4], which proved to deliver considerable speed up in case of single pattern string matching but it is also observed that applying bit-parallelism for multiple pattern string matching is tedious task [27]. By utilizing arithmetic and logical operations and assigning numbers to different states, Baeza-Yates and Gonnet [4] proposed searching method, which includes all keywords of the pattern set when computer word size is not large enough. Kim & Kim [35] proposed an algorithm which performs bitwise hashing operations while scanning by encoding the patterns and text at the beginning of the process. Timo Raita [29] improves the BMH algorithm by introducing the dependencies between the characters and achieved significant speed enhancement over the well-known fastest Boyer–Moore–Horspool String Searching Algorithm.

Yuan *et al*. [50] implemented the QWM which provide the better performance by combining the QS algorithm with Wu-manber algorithm. Improved Wu-Manber algorithm performance turns out to be significant as the maximum shift distance of improved version is better than original Wu-Manber algorithm [46]. Wu-Manber and Quadrupled Wu Manber (QWM) algorithm is relatively not better than Aho-Corasick and Commentz-Walter algorithm [8] when binary text is employed rather than increased pattern length and alphabet size. Among all four, QWM algorithm has better performance. That's why QWM algorithm is applied in various fields, such as network content analysis, intrusion detection, and text retrieval systems.

The grep tools contributed a lot to the field of multi pattern string matching. First implementation of grep was given by Thompson using Nondeterministic Finite Automaton (NFA) for regular expression [37]. Then, Alfred Aho made egrep (enhanced grep and later extended grep) using Deterministic Finite Automaton (DFA) [3]. Also fgrep (fixed-string grep) tool is based on Aho-Corasick algorithm [43]. The UNIX grep utility searches the input and outputs the lines which contain any of the input patterns. The UNIX egrep supports additional operators for regular expressions [1]. UNIX fgrep performs like grep but does not recognize any regular expression meta-characters as special. So, fgrep is basically a fixed pattern matching tool, hence performs faster [1]. The original egrep and fgrep could not perform well for more than few hundred patterns [43]. Agrep [44] advances from egrep and fgrep as it searches the patterns in given text approximately, which is based on bitap algorithm. In 1964 Balint Domolki proposed the bitap algorithm for exact string searching [9, 26] and extended by Shyamsundar in 1977 [32], this algorithm is reinvented by Manber and Wu [44] from exact string matching to approximate string matching also known as fuzzy string matching based on work done by Baeza-Yates and Gonnet [4]. Further improvement was done by Sunday [34] and then for long patterns Gene Myers proposed an improved algorithm based on bitap algorithm in 1998 [25].

In this paper, multiple pattern string matching method is designed on the basis of BMH algorithm. By using this method, a multiple pattern string matching algorithm is proposed. This algorithm provides shifting mechanism for unnecessary matched character. Here MP-BMH algorithm, its time and space complexity analysis and comparative analysis with other standard tools and algorithm are described.

3. Multiple Pattern String Matching Using BMH

In the BMH multiple pattern string matching method at a time single pattern is consider for string matching process. The pattern that is in the string matching is referred as "current pattern". Whenever mismatch is occurred then the minimum shift is calculated. This minimum shift is calculated by aligning the character of text corresponding to last character of pattern which

is same in any other patterns. In the next string matching process the pattern through which this minimum shift is calculated set as current pattern and used for string matching. Suppose there are k number of patterns and T is text in which string matching is to be performed. There is a pattern of length m_r which is the smallest length pattern or one of the smallest length pattern among the k patterns. For string matching any position j, ($j \leq i$) or ($j \geq i- m_r$) where "i" is the location of the character in the text which is aligned to the last character of the current pattern. If there is a mismatch at any position j, condition is described in Figure 1.

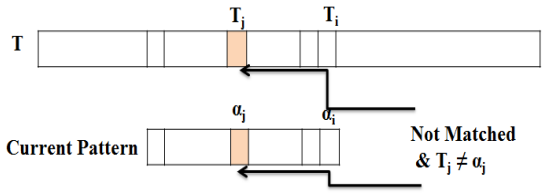

Figure 1. Multiple Pattern Matching Using BMH Method Mismatch Logic.

Here, character T_i is the character of text aligned to the last character of the pattern in the string matching process. Now, for shift search for occurrences of character T_i in all k patterns. Suppose there is a pattern ɤ, this character T_i could be occurred in a pattern or not, by taking all the cases the value of shift will be

$$0 \leq shift \leq m_{ɤ} \quad (1)$$

Shift is zero when the character T_i is at rightmost position or last position from right. Shift is m_r when the character T_i is not in the pattern. If character lies in between than shift is greater than zero and less than $m_ɤ$. Among all the calculated shifts in different patterns, we choose a minimum shift.

$$Minimum\ Shift = min\ \{shift\ (1),\ shift\ (2), shift\ (3).....,sh\ \} \quad (2)$$

OR

$$Minimum\ Shift = min\ \{shift\ (x):\ 1 \leq x \leq k\} \quad (3)$$

Than new string matching process start from the position i_{new}, which is described in Figure 2.

$$i_{new} = i_{old} + Minimum\ Shift \quad (4)$$

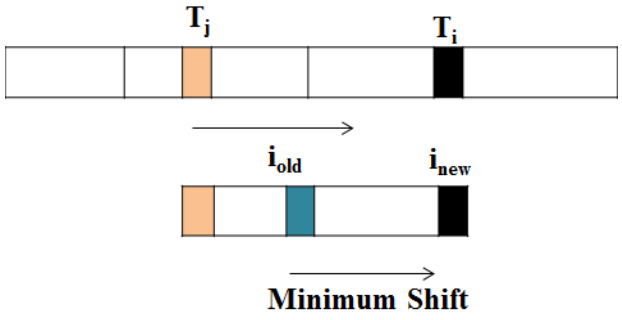

Figure 2. Multiple pattern matching using BMH method shifting Logic.

Here choose a pattern corresponding to this minimum shift is calculated and that pattern is set as "current pattern". Now same process is executed for next iterations. In case of fully matched pattern, same process is executed for calculation of minimum shift but here current pattern is not used for the calculation because it is already having minimum value.

4. Proposed MP-BMH Algorithm

MP-BMH algorithm is based on the concept of the Multiple Pattern Matching using BMH method. Algorithm is a exact string matching algorithm based on above method.

4.1. Terminologies Used:

- **Current Pattern:** It refers to the pattern that is in the string matching process.
- **MValue:** MValue is bad character value of last character of pattern (except last character). If that character does not appear again in pattern then it is equal to pattern length.

4.2. MP-BMH Algorithm

Algorithm has two phases pre-processing and searching. Pre-processing is further divided in two parts phase-1 and phase-2. In pre-processing phase-1 MValue and next–to-last bad character of all the patterns and then in pre-processing phase-2 minimum values tables are calculated. Calculated values in pre-processing Phase-1are not used so these values can be removed after phase-2. These values are only used for the calculation of phase-2. Minimum values tables which are generated in phase-2 will be used in searching phase.

- **Pre-processing Phase-1**: In pre-processing phase-I first sort the patterns according to ascending order of their length. Then MValue and next-to-last bad characters values of all patterns are calculated and stored in the form of tables.
- **Pre-processing Phase-2**: Using the pre-processing phase-I values go for next level of pre-processing in which after sorting calculate three different values for each character for each pattern according to the different situation. These situations are mismatch at last character, mismatch in between and fully match for a pattern. These values are MBCVA (minimum of next-to-last bad character value), MBCVMV (minimum of next-to-last bad character value of all patterns except "current pattern" and MValue of "current pattern") and NMBCVA (Non zero minimum of next-to-last bad character value). MBCVA table used in the case of mismatch at last character. Whereas MBCVMV and NMBCVA are used in case of mismatched in between and fully matched pattern. If the multiple patterns of same

suffixes are present then analogy function is used. Analogy function is used to achieve better performance by skipping unnecessary matching of characters by seeing the similarity between the patterns. This function holds patterns of common suffixes.

Algorithm 1 MP-BMH

1. Start
#Pre-Processing Phase:
2. Pre-Processing Phase I:
- *Sort the pattern in ascending order of length and calculate next to last bad characters table MValue for each pattern.*

3. Pre-Processing Phase II:
- *Calculate MBVCA, MBCVMV and NMBCVA tables and analogy function*

#Search Phase:
4. Start as first pattern as "current pattern" in sorted list, which is smallest length pattern or may be one of the smallest length patterns among all the patterns.
5. Index=Start index of the Text.
6. While (Index<Length of Text) do
(a) If (Index < Length of "current pattern") then
- *Shift by smallest pattern length*
- *Set "current pattern"=smallest length pattern*

(b) Align the "current pattern "with text and start matching from right to left like BMH.
(c) If Mismatch occurs at last character then
- *Shift using shift value of table MBCVA.*
- *Set the "current pattern"=pattern of MBCVA.*
- *Goto step 6.*

(d) If Mismatch occurs in between then
- *Shift using shift value of table MBCVMV*
- *If shift non-zero then Set the "current pattern"=pattern in the pattern field of MBCVMV and goto step 6.*

Else
- *(iii) Search all patterns in the pattern field of MBCVMV using the analogy function for efficient matching, if occurs then report the occurrence.*
- *(iv) Shift using shift value of NMBCVA.*
- *(v) Set the "current pattern"=pattern in the pattern field of NMBCVA.*
- *(vi) Goto step 6.*

(e) If Full match occurs then
- *Report occurrence and (pattern position)*
- *Shift using shift value of Table MVCVMV*
- *If shift is non zero then (set "current pattern"=pattern in the pattern field of MBCVMV and goto Step-6*

Else
- *Search all patterns in the pattern field of MBCVMV using the analogy function for efficient matching,if occurs then report the occurrence.*
- *Shift using shift value of NMBCVA.*
- *Set the "current pattern"=pattern in the pattern field of NMBCVA*
- *Goto step 6.*

END

5. Example of MP-BMH Algorithm

Patterns: ABC, ABCD, ZABC, YZABC, EDE. **Text:** XYZABCDABCCDE

5.1. Pre-processing Phase-1

The patterns are sorted in ascending order of their length shown Table 1.

Table 1. Sorted list of patterns.

Index	Pattern
0	ABC
1	EDE
2	ABCD
3	ZABC
4	YZABC

MValue and next-to-last bad character table for each of the patterns are calculated in first phase of pre-processing shown in Table 2. In below Table 2, the character is represented by "Ch" shift is represented by "Sh", and others is represented by "O" respectively.

Table 2. MValue and next-to-last bad character table for each pattern.

Pattern: ABC MValue: 3

Ch	Sh
C	0
B	1
A	2
Ot	3

Pattern: EDE MValue: 2

Ch	Sh
E	0
D	1
Ot	3

Pattern: ABCD MValue: 4

Ch	Sh
D	0
C	1
B	2
A	3
Ot	4

Pattern: ZABC MValue: 4

Ch	Sh
C	0
B	1
A	2
Z	3
Ot	4

Pattern: YZABC MValue: 5

Ch	Sh
C	0
B	1
A	2
Z	3
Y	4
Ot	5

5.2. Pre-processing Phase-2

The results after pre-processing phase-II for above example are shown in Table 3. In below Table 3, the character is represented by "Ch" shift is represented by "Sh", others is represented by "Ot" and pattern is represented by "Pat" respectively.

Table 3. Pre-processing Phase-II Tables MBCVA, MBCVMV and NMBCVA.

Pattern: ABC MValue: 3

Ch	Sh
C	0
B	1
A	2
Ot	3

Pattern: EDE MValue: 2

Ch	Sh
E	0
D	1
Ot	3

Pattern: ABCD MValue: 4

Ch	Sh
D	0
C	1
B	2
A	3
Ot	4

Pattern: ZABC MValue: 4

Ch	Sh
C	0
B	1
A	2
Z	3
Ot	4

Pattern: YZABC MValue: 5

Ch	Sh
C	0
B	1
A	2
Z	3
Y	4
Ot	5

5.3. Searching Phase

For the above example given text is "XYZABCDABCCDE". Then in searching process, comparison starts from smallest length pattern, which is ABC. Character C of pattern ABC is compared with character Z of text, it mismatches so for Z go to shift MBCVA Table which is 3 here for pattern ABC, then

shift by 3. Now compare C of ABC with C of text, they matches, compare next until mismatch or fully match, here all characters of ABC are matched, means ABC is found. Now go to MBCVMV Table for ABC, here the shift is zero so match all the patterns in the pattern field of this Table with the corresponding characters of text and if match found then report occurrence. After this go to NMBCVA Table for character C and shift by the shift value of this Table and continue searching as follows. The full searching phase is shown in Figure 3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Text	X	Y	Z	A	B	C	D	A	B	C	C	E	D	E
1	A	B	C .											
2				A .	B .	C .								
3			Z .	A	B	C								
4		Y .	Z .	A	B	C								
5				A .	B .	C .	D .							
6						E	D	E .						
7								A .	B .	C .				
8							Z .	A	B	C				
9						Y	Z .	A	B	C				
10								A	B	C	D .			
11									A	B .	C .			
12								Z	A	B	C			
13							Y	Z	A	B	C			
14									A	B	C	D .		
15										E	D .	E .		
16												E .	D .	E .

Figure 3. Example of MP-BMH Searching Phase('.' shows where matching is performed).

6. Time and Memory Analysis

6.1. Time Complexity Analysis

Consider that there is K number of distinct patterns of pattern length {m_1, m_2, ...m_k} and n is the size of text. Suppose smallest length pattern length is m_s. There is a set T = {t_1, t_2, ...,t_t} contains t distinct characters appears in the text and P ={p_1, p_2,...., p_p} contains p distinct characters appears in the K patterns. Now, here the time complexity of algorithm among various stages is explained such as pre-processing phase-1, pre-processing phase-2 and searching phase.

- Pre-processing Phase: In pre-processing phase-1 there are two tasks. First, to sort all K patterns according to their lengths in ascending order and second are to calculate bad character table and MValue for all pattern. For sorting of K patterns we can choose any of the sorting algorithm. Since time complexity of sorting algorithms lies between O (K $\log_2$K) to O (K^2) choose worst case time that is O (K^2).Time required for calculating next-to-last bad character values of K patterns is O (Kt).Calculating MValue for all patterns takes O (m_1+m_2+.....+m_k) time, if m_{avg} is the average length of K patterns then it can be written as O (Km_{avg}). Then the total time required for the pre-processing phase-I is O (K^2+Kt+Km_{avg}). In pre-processing phase-2 MBCVA, MBCVMV and NMBCVA table values are calculated. MBCVA Table contains values for all the distinct characters available in text. For calculating, these values scan each character in all the other patterns. So the time complexity is O (Kt). For MBCVMV table search next-to-last values of last character of each pattern in the other entire K-1 patterns so the time complexity is O (K (K-1)) i.e., O (K^2).As MBCVA table, NMBCVA also contains values for all distinct characters of text. And similarly the time complexity for NMBCVA table is O (Kt).So the total time of phase-II is O(Kt+K^2+Kt) i.e., O (K^2+Kt). Overall time complexity of pre-processing that include phase-I and phase-II is O(Kt+Km_{avg}+K^2+Kt) i.e., O (K^2+Kt+Km_{avg}).
- **Searching**: In searching best case, average case, and worst case is possible based on different type of text irrespective of pattern set.

a) Best Case: It is the case where It is the case where no pattern exists in the text and minimum pattern length is larger. When all the character of T and P are different. This is a no pattern found condition. It means T∩P=Null. Here at each mismatch condition we get a jump of m_s. so time complexity is O(n/m_s). If $m_s$is large then we get more fast search results. This case is shown in the Figure-4. Here Patterns are ABCD,DEFG,FGJIJ and Text is "XYZXWYZXWZXWXYWZ".

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Text	X	Y	Z	X	W	Y	Z	X	W	Z	X	W	X	Y	W	Z
1	A	B	C	D												
2					A	B	C	D								
3									A	B	C	D				
4													A	B	C	D

Figure 4. Best Case of MP-BMH algorithm.

Here MP-BMH always gives maximum shift i.e., minimum pattern length shift while scanning the text.

b) Average Case: It is a case where some patterns may exists in the text and chances for getting zero to pattern length shift. When some of the characters of T and P are same or T is a superset of P. It means T∩P=Q, Q is the subset of P and T both. Here we always get the jump from one to minimum pattern length. So time complexity lies between O (n/m_s) to O (n). In general total jump is (1+2+.....+ m_s)/ m_s i.e., m_s (m_s+1)/ 2$m_s$or (m_s+1)/ 2then the average time is O (2n/(m_s+1)). This case is shown in Figure 5.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Text	X	Y	Z	A	B	C	D	W	X	Y	Z	Y	N	M	W	Y	X	D	E	F	G	X	Y	J	Z	W
1	A	B	C	D																						
2				A	B	C	D																			
3							D	E	F	G																
4											A	B	C	D												
5															A	B	C	D								
6																		D	E	F	G					
7																				F	G	J	I	J		
8																						F	G	J	I	J

Figure 5. Average Case of MP-BMH algorithm.

Here Patterns are ABCD, DEFG, FGJIJ and Text is "XYZABCDWXYZNMWYXDEFGXYJZW".

c) Worst Case: It is the case where maximum pattern exists in text and chances of always getting one shift. When all the characters of T and P are same. It means T∩P=P. In this case for each char in text it is possible to match each and every character of all the patterns. So "n" times $O(Km_{avg})$could be the complexity. This should be $O(m_{avg}nK)$. This case is marginally similar to the Figure 6. This case practically almost impossible in real world. Here patterns are ABCD, XABCD, XYABCD, YXZZX and Text "WXYABCDMNYXZZXNNBXDMNNM".

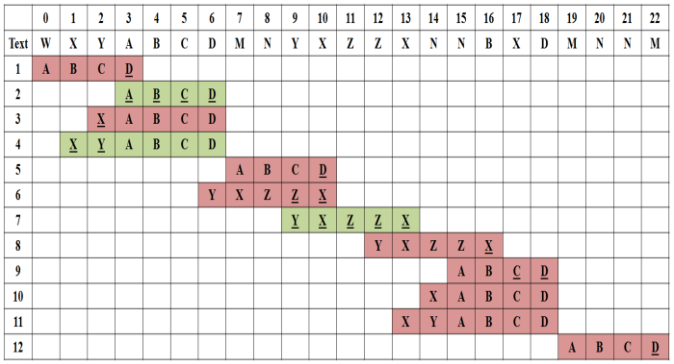

Figure 6. Worst Case of MP-BMH algorithm.

Here MP-BMH always rematch the characters approximately equal to the minimum pattern length or it is similar to getting one shift in each iteration here complexity is just equal to the O (nK).

6.2. Space Complexity Analysis

In MP-BMH memory requirement is only in the pre-processing information storage. In Pre-processing phase-I sorting of patterns requires O(K) space complexity. The space complexity is O (K) for the MValue calculation and O (Kt) for next-to-last bad character values calculations. Then the total space complexity for the pre-processing phase-I is O (K+Kt+K). In the pre-processing phase-II For MBCVA and NMBCVA tables, space complexity for each of them is O(t). For MBCVMV table, in worst case, total K (K-1) values to be saved. So, this will take O (K^2) space complexity for the table. Then total space complexity for pre-processing phase-II is O ($t+t+K^2$) i.e. O (K^2+t). Now total space complexity for the pre-processing phase is O ($K+Kt+K^2+t$)i.e. O(K^2+Kt+t). Whereas DFA based Aho-Corasick algorithm takes O(($m_1+ m_2+.....+ m_k+1$) t) worst case space complexity. If m_{avg} is the average length of K pattern then O (K m_{avg} t) is the space complexity in the worst case.

7. Experimental Results and Analysis of MP-BMH Algorithm

In this section MP-BMH algorithm is compared to various types of string matching tools based on different string matching algorithms. These comparisons are based on number of patterns and size of patterns. MP-BMH algorithm has shown significant good results in compared to other tools for large pattern size as well as on small pattern size. The experiments shown here were tested on Intel(R) Core(TM) i52430M CPU 2.4GHz with installed memory(RAM) 4 GB, running 64-bit version of Windows7 operating system and ubantu12.04. All times are execution times on a lightly loaded system getting more than 90% of the CPU. These times given in seconds; each experiment was performed 20 times and then the averages are given. For all the experiment shown in this paper we have used text file from Bible [36] whose size is 101 MB. All the patterns used in experiments are taken corresponding to the above text file.

Table 4 shows comparison of different types of string searching tools like egrep(DFA based), fgrep (Aho-Corasick based) and agrep (bitap based) with the proposed algorithm MP-BMH. MP-BMH is showing significant relative speed up. We have tested all these tools on different variants of patterns ranging from minimum pattern length=2 to minimum pattern length=10. Table 1 shows the result of running time (assuming average case for all tools) from m=4 pattern length for number of patterns ranging from 100 to 1000.

Table 4. Comparison of different search routines on a 101 MB text.

# OF PATTERNS	EGREP	FGREP	AGREP	MP-BMH
100	6.105	6.239	5.981	1.982
200	9.914	11.059	9.719	3.432
300	12.738	14.395	11.931	5.226
500	18.173	19.599	16.560	9.079
1000	70.154	27.171	23.240	22.979

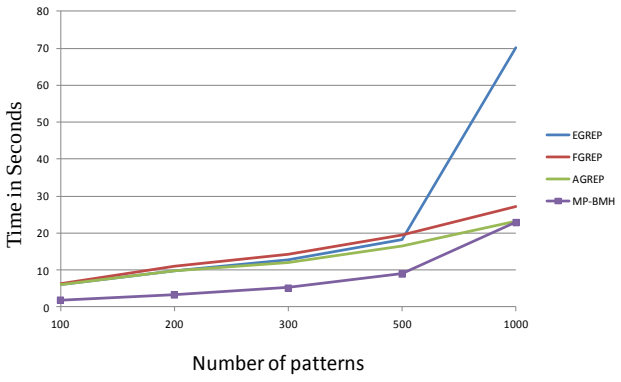

Figure 7. Comparison of running times for different number of patterns.

In Figure 7, we have plotted graph for MP-BMH for different number of patterns ranging from 100 to 1000. Here y-axis shows the running time in seconds and x-axis shows the number of patterns viz.

100,200,300,500 and 1000 in MP-BMH from different string searching tools. It is seen from the Table 4 and graph as shown in Figure 6, above that when number of patterns taken are 100 then percentage increase of MP-BMH from egrep, fgrep and agrep is 67.53%, 68.23% and 66.86% respectively, when number of patterns taken are 300 then percentage increase of MP-BMH from egrep, fgrep and agrep is 58.97%, 63.7% and 56.19% respectively and when number of patterns taken are 1000 then percentage increase of MP-BMH from egrep, fgrep and agrep is 67.24%, 15.42% and 1.12% respectively. So it is obvious from the facts that although percentage improvement of MP-BMH is decreasing with respect to egrep,fgrep and agrep when number of patterns increases gradually but relative running time of MP-BMH in comparison to other tools like agrep, fgrep and egrep is still good.

The Figure 8, shows the graph plotting of average case running time for MP-BMH algorithm. The numbers of patterns are 500 and minimum pattern size ranges from 2 to 10. It is obvious from the graph as shown in Figure 3, that average case of running time is significantly improves from minimum pattern size 2 to minimum pattern size 10 like other tools egrep, fgrep and agrep.

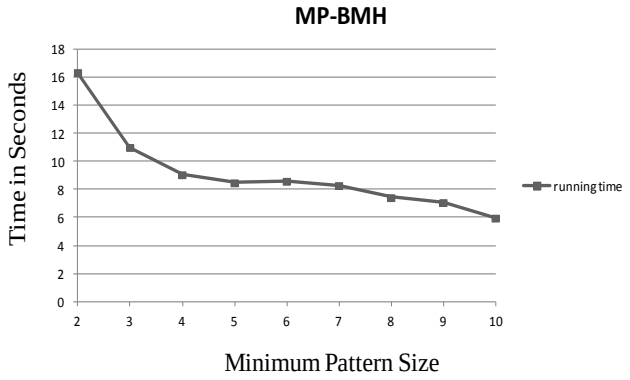

Figure 8. The effect of the minimum pattern length on the running time.

8. Conclusions

MP-BMH is better and more efficient in terms of time and memory requirements while comparing with standard benchmark algorithms like Aho-Corasick and Shift-OR with q-grams. Experiments r showed that MP-BMH is faster than the standard string matching tools like egrep, fgrep and agrep because of its shifting mechanism. It is significantly faster with up to 1000 number of patterns. Its performance improvement is also based on the minimum pattern length. As the minimum pattern length increase algorithm provides larger shift and overall performance will be increased. MP-BMH is more efficient in memory and it requires less memory for providing hashing. In conclusion, MP-BMH can provide great improvement in the string matching process which plays major role in many real world applications.

References

[1] Abou-Assaleh T. and Ai W., "Survey of Global Regular Expression Print (Grep) Tools," *in Proceedings of Citeseer, Topics in Program Comprehension*, Nova Scotia, pp. 1-8, 2004.

[2] Aho A. and Corasick M., "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM*, vol. 18, no. 6, pp. 333-340, 1975.

[3] Alzahrani S., Salim N., and Abraham A., "Understanding Plagiarism Linguistic Patterns, Textual Features, and Detection Methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 133-149, 2012.

[4] Baeza-Yates R. and Gonnet G., "A New Approach to Text Searching," *Communications of the ACM*, vol. 35, no. 10, pp. 74-82, 1992.

[5] Baeza-Yates R. and Navarro G., "A Faster Algorithm for Approximate String Matching," *in Proceedings of Annual Symposium on Combinatorial Pattern Matching*, Berlin, pp. 1-23, 1996.

[6] Boyer R. and Moore J., "A Fast String Searching Algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762-772, 1977.

[7] Brođanac P., Budin L., and Jakobović D., "Parallelized Rabin-Karp Method for Exact String Matching," *in Proceedings of 33rd International Conference on of Information Technology Interfaces*, Dubrovnik, pp. 585-590, 2011.

[8] Commentz-Walter B., "A String Matching Algorithm Fast on the Average," *in Proceedings of 6th International Colloquium on Automata, Languages, and Programming*, Berlin, pp. 118-132, 1979.

[9] Dolmiki B., "A Universal Compiler System Based on Production Rules," *BIT Numerical Mathematics*, vol. 8, no. 4, pp. 262-275, 1968.

[10] Domolki B., *an Algorithm for Syntactical Analysis*, Computational Linguistics, 1964.

[11] Fethallah H. and Amine C., "Automated Retrieval of Semantic Web Services: A Matching Based on Conceptual Indexation," *The International Arab Journal of Information Technology*, vol. 10, no. 1, pp. 61-66, 2013.

[12] Han Y. and Xu G., "Improved Algorithm of Pattern Matching Based on BMHS," *in Proceedings of IEEE International Conference on Information Theory and Information Security*, Beijing, pp. 238-241, 2010.

[13] Horspool R., "Practical Fast Searching in Strings," *Software Practice and Experience*, vol. 10, pp. 501-506, 1980.

[14] Hume A., "A Tale of Two Greps," *Software Practice and Experience*, vol. 18, no. 11, pp. 1063-1072, 1988.

[15] Ke X., Yong C., and Hong Y., "An Improved Wu-Manber Multiple Patterns Matching Algorithm," *in Proceedings of 25th IEEE International Performance, Computing, and Communications Conference*, Phoenix, 2006.

[16] Khancome C. and Boonjing V., "New Hashing-Based Multiple String Pattern Matching Algorithms," *in Proceedings of 9th International Conference on Information Technology: New Generations (ITNG)*, Las Vegas, pp. 195-200, 2012.

[17] Kim B. and Kim I., "Kernel Based Intrusion Detection System," *in Proceedings of 4th Annual ACIS International Conference on Computer and Information Science*, Jeju Island, pp. 13-18, 2005.

[18] Kim S. and Kim Y., "A Fast Multiple String-Pattern Matching Algorithm," *in Proceedings of 17th AoM/IAoM Conference on Computer Science*, pp. 1-8, 1999.

[19] Knuth D., Morris J., and Pratt V., "Fast Pattern Matching in Strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323-350, 1977.

[20] Kouzinopoulos C. and Margaritis K., "A Performance Evaluation of the Pre-processing Phase of Multiple Keyword Matching Algorithms," *in Proceedings of 15th Panhellenic Conference on Informatics*, Kastonia, pp. 85-89, 2011.

[21] Lee T., "Generalized Aho-Corasick Algorithm for Signature Based Anti-Virus Applications," *in Proceedings of 16th International Conference on Computer Communications and Networks*, Honolulu, pp. 792-797, 2007.

[22] Marturana F., Me G., and Tacconi S., "A Case Study on Digital Forensics in the Cloud," *in Proceedings of International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Sanya, pp. 111-116, 2012.

[23] Miao C., Chang G., and Wang X., "Filtering Based Multiple String Matching Algorithm Combining q-Grams and BNDM," *in Proceedings of 4th International Conference on Genetic and Evolutionary Computing*, Shenzhen, pp. 582-585, 2010.

[24] Muth R. and Manber U., "Approximate Multiple String Search," *in Proceedings of Combinatorial Pattern Matching, Lecture Notes in Computer Science*, Berlin, pp. 75-86, 1996.

[25] Myers G., "A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming," *Journal of the ACM*, vol. 46, no. 3, pp. 395-415, 1999.

[26] Navarro G. and Raffinot M., "Fast and Flexible String Matching by Combining Bit-Parallelism and Suffix Automata," *Journal of Experimental Algorithmics*, vol. 5, no. 4, 2000.

[27] Peltola H. and Tarhio J., "Alternative Algorithms for Bit-Parallel String Matching," *in Proceedings of International Symposium on String Processing and Information Retrieval*, Berlin, pp. 80-93, 2003.

[28] Qiang Z., "An Improved Multiple Patterns Matching Algorithm for Intrusion Detection," *in Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems*, Xiamen, pp. 124-127, 2010.

[29] Raita T., "Tuning the Boyer-Moore-Horspool String Searching Algorithm," *Software Practice and Experience*, vol. 22, no. 10, pp. 879-884, 1992.

[30] Salmela L., Tarhio J., and Kytojoki J., "Multi Pattern String Matching with Q-Grams," *Journal of Experimental Algorithmic*, vol. 11, pp. 1-19, 2006.

[31] Sánchez D., Martín-Bautista M., Blanco I., and de la Torre C., "Text Knowledge Mining: An Alternative to Text Data Mining," *in Proceedings of IEEE International Conference on Data Mining Workshops*, Pisa, pp. 664-672, 2008.

[32] Shyamasundar R., "Precedence Parsing Using Domolki's Algorithm," *International Journal of Computer Mathematics*, vol. 6, no. 2, pp. 105-114, 1977.

[33] Sridhar M., Efficient Algorithms for Multiple Pattern Matching, Doctoral Dissertations, the University of Wisconsin, 1986.

[34] Sunday D., "A Very Fast Substring Search Algorithm," *Communications of the ACM*, vol. 33, no. 8, pp. 132-142, 1990.

[35] Tao T. and Mukherjee A., "Multiple-Pattern Matching in LZW Compressed Files Using Aho-Corasick Algorithm," *in Proceedings of International Conference on Information Technology: Coding and Computing*, Las Vegas, pp. 482, 2005.

[36] The Bible-Pdf E-Book Version of the Bible, www.holybooks.com/download-bible, Last Visited, 2017.

[37] Thompson K., "Programming Techniques: Regular Expression Search Algorithm," *Communications of the ACM*, vol. 11, no. 6, pp. 419-422, 1968.

[38] Wang Y., "A New Method to Obtain The Shift-Table in Boyer-Moore's String Matching Algorithm," *in Proceedings of 19th International Conference on Pattern Recognition*, Tampa, pp. 1-4, 2008.

[39] Watson B. and Zwaan G. "A Taxonomy Of Sub Linear Multiple Keyword Pattern Matching Algorithms," *Science of Computer Programming*, vol. 27, no. 2, pp. 85-118, 1996.

[40] Watson B., "Taxonomies and Toolkits of Regular Language Algorithms," Ph.D. Dissertation, Faculty of Computer Science, Eindhoven University of Technology, 1995.

[41] Watson B., "The Performance of Single-Keyword and Multiple-Keyword Pattern Matching Algorithms," Eindhoven University of Technology, Computing Science Section, 1994.

[42] Wu P. and Shen H., "The Research and Amelioration of Pattern-matching Algorithm in Intrusion Detection System," *in Proceedings of IEEE 14th International Conference on High Performance Computing and Communication and IEEE 9th International Conference on Embedded Software and Systems*, Liverpool, pp. 1712-1715, 2012.

[43] Wu S. and Manber U., "Fast Text Searching: Allowing Errors," *Communications of the ACM*, vol. 35, no. 10, pp. 83-91, 1992.

[44] Wu S. and Manber U., "Agrep-A Fast Approximate Pattern- Matching Tool," *in Proceedings of Usenix Winter Technical Conference*, San Francisco, pp.153-162, 1992.

[45] Wu S. and Manber U., "Fast Text Searching With Errors," Technical Report, Department of Computer Science, University of Arizona, Tucson, 1991.

[46] Wu S. and Manber U., "A Fast Algorithm for Multi-Pattern Searching," Technical Report, Department of Computer Science, 1994.

[47] Xiangyan F., Tinggang X., Yidong D., and Youguang Y., "The Research And Improving For Multi-Pattern String Matching Algorithm," *in Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems*, Xiamen, pp. 266-270, 2010.

[48] Xie L., Liu X., and Yue G., "Improved Pattern Matching Algorithm of BMHS," *in Proceedings of 3rd International Symposium on Information Science and Engineering*, Shanghai, pp. 616-619, 2010.

[49] Xiong Z., "A Composite Boyer-Moore Algorithm for the String Matching Problem," *in Proceedings of International Conference on Parallel and Distributed Computing, Applications and Technologies*, Wuhan, pp. 492-496, 2010.

[50] Yuan J., Yang J., and Ding S., "An Improved Pattern Matching Algorithm Based on BMHS," *in Proceedings of 11th International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, Guilin, 2012.

[51] Yuan J., Zheng J., and Ding S., "An Improved Pattern Matching Algorithm," *in Proceedings of 3rd International Symposium on Intelligent Information Technology and Security Informatics*, Jinggangshan, pp. 599-603,2010.

[52] Zha X. and Sahni S., "GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1156-1169, 2013.

[53] Zhang B., Chen X., Ping L., and Wu Z., "Address Filtering Based Wu-Manber Multiple Patterns Matching Algorithm," *in Proceedings of 2nd International Workshop on Computer Science and Engineering*, Qingdao, pp. 406-412, 2009.

Akhtar Rasool is an Assistant Professor, at Maulana Azad National Institute of Technology. He did M.Tech and PhD in Computer Science and Engineering from Maulana Azad National Institute of Technology. He has published more than 20 research papers in international/national journals and conferences. His research areas include String Matching Algorithms, Parallel Computing, Artificial Intelligence, Big Data Analysis, Software Engineering, Analysis and Design of Algorithms, Cluster and Grid Computing.

Gulfishan Firdose Ahmed is an Assistant Professor at College of Agriculture, Powarkheda, affiliated by JNKVV Jabalpur. She did M.Tech from RGPV Bhopal and she received PhD in Computer Science and Engineering from Maulana Azad National Institute of Technology. She has published more than 20 research papers in international/national journals and conferences. Her research areas include wireless network, string Matching Algorithms and high performance computing.

Raju Barskar is an Assistant Professor at UIT RGPV Bhopal. He did M.Tech and pursuing PhD in Computer Science and Engineering from Maulana Azad National Institute of Technology. He has published more than 20 research papers in international/national journals and conferences. His research areas include wireless network, VANET, string Matching Algorithms and image processing.

Nilay Khare is an associate professor at Maulana Azad National Institute of Technology. He did M.Tech. in Computer Science and Engineering from IIT Delhi. He received Ph.D degree in Computer Science and Engineering. He has been professor in Department of Technical Education, Government of Madhya Pradesh, India. He has also worked as head of State Project Facilitation Unit, Government of Madhya Pradesh, India and as head of department of Computer Science and Engineering, Rajiv Gandhi Technological University, Bhopal, Madhya Pradesh, India. He has published more than 50 research papers in national and international journals. He is also member of Indian Society of Technical Education (ISTE) and Computer Society of India (CSI). His research areas include Wireless Networks, High performance computing and Theoretical Computer Science.