Variation of Boyer-Moore String Matching
Algorithm: A Comparative Analysis

Ramshankar Choudhary
Department Of Computer Science
And Engineering
Maulana Azad National Institute of
Technology
Bhopal-462051,India
ramshankar.choudhary2010@gmail.com

Prof. Akhtar Rasool
Asst. Professor Department Of
Computer Science And Engineering
Maulana Azad National Institute of
Technology
Bhopal-462051,India
akki262@yahoo.co.in

Dr. Nilay Khare
Associate Professor, HOD
Department Of Computer Science
And Engineering
Maulana Azad National Institute of
Technology Bhopal-462051,India
nilay_khare@yahoo.co.in

Abstract- String matching plays an important role in field of
Computer Science and there are many algorithm of String
matching, the important aspect is that which algorithm is to be
used in which condition. BM(Boyer-Moore) algorithm is
standard benchmark of string matching algorithm so here we
explain the BM(Boyer-Moore) algorithm and then explain its
improvement as BMH (Boyer-Moore-Horspool), BMHS
(Boyer-Moore-Horspool-Sundays), BMHS2 (Boyer-Moore-
Horspool-Sundays 2), improved BMHS( improved Boyer-
Moore-Horspool-Sundays), BMI (Boyer-Moore improvement)
and CBM (composite Boyer-Moore).And also analyze and
compare them using a example and find which one is better in
which conditions.

Keywords-String Matching: BM; BMH; BMHS; BMHS2;
improved BMHS; BMI; CBM

I. INTRODUCTION

In computer science, the Boyer-Moore string search
algorithm is a particularly efficient string searching
algorithm, and it has been the standard benchmark for the
practical string search literature.

It was developed by Bob Boyer and J Strother Moore in
1977. The algorithm preprocesses the pattern string that is
being searched in text string.[5]

Before BM algorithm was proposed, the direction of
character comparison was consistent to the moving direction
of the pattern i.e. both are from left to the right. But in BM
the direction of character comparison is different from the
moving direction of the pattern i.e. from right to left in
pattern.[4]

After BM algorithm was proposed there were some
algorithms are proposed to improve it. In 1980, Horspool
simplified BM algorithm and proposed BMH algorithm.
Although it only used the information of the table Right,
BMH algorithm acquired no bad efficiency. In 1990 Sunday
proposed BMHS algorithm that improved the BMH
algorithm.[6]

In 2010, Lin quan Xie, Xiao ming liu proposed BMHS2,
which is strictly based on the analysis of BMHS algorithm
to improve is in the match fails, the text string matches last
bit characters to participate in the next match, a character
string in the case appear to increase the last bit character and
appear in the character string matching the first characters of
a position if there is consideration.[3]

In 2010 BMI algorithm is proposed by Jingbo Yuan,
Jisen Zheng, Shunli Ding which is improvement of BM
algorithm. The BMI algorithm combines with the
good-suffix function and the advantages of BMH and BMHS.
At the same time the BMI algorithm also takes into account
the singleness and combination features of the Next-Character
and the Last- Character. [8, 9]

There are two important factors which influence the
efficiency and speed of pattern matching and they are the
cost to find the mismatching character in the text string
and the shift distance to right. On basis of the two factors,
an improved algorithm called Improved BMHS algorithm
which is given by Yuting Han, Guoai Xu in 2010.[7]

Another improved algorithm called composite Boyer-
Moore was proposed in 2010 by Zhengda Xiong. The key
issue of the composite Boyer-Moore algorithm is how to
utilize the history comparison information achieved at
previous iteration. So a new concept of two-dimensional
table Jump[m][m] is introduced.[4]

II. BM ALGORITHM

The BM algorithm scans the characters of the pattern
from right to left beginning with the rightmost one and
performs the comparisons from right to left. In case of a
mismatch (or a complete match of the whole pattern) it uses
two pre-computed functions to shift the window to the right.
These two shift functions are called the good-suffix shift
(also called matching shift and the bad-character shift (also
called the occurrence shift).

Assume that a mismatch occurs between the character
$P[i]=b$ of the pattern and the character $T[i+j]=a$ of the
text during an attempt at position $j$. Then, $P[i+1..m-1]=T[i+j+1..j+m-1]=u$ and $P[i]≠T[i+j]$.The good-suffix shift consists in aligning the segment $T[i+j+1..j+m-1]$ with its rightmost occurrence in $P$ that is preceded by a character different from $P[i]$.
BM algorithm will carry through shift computing as follow.

(1) good-suffix function
The algorithm looks up string \( u \) leader character is not \( b \) in \( P \) from right to left. If there exist such segment, shift right \( P \) to get a new attempt window. If there exists no such segment, the shift consists in aligning the longest suffix \( v \) of \( T[i+j+1..j+m-1] \) with a matching prefix of \( P \).

(2) bad-char function
The bad-character shift consists in aligning the text character \( T[i+j] \) with its rightmost occurrence in \( P[0..m-2] \).

If \( T[i+j] \) does not occur in the pattern \( P \), no occurrence of \( P \) in \( T \) can include \( T[i+j] \), and the left end of the window is aligned with the character immediately after \( T[i+j] \), namely \( T[i+j+1] \).

BM algorithm uses good-suffix function and bad-char function to calculate the new comparing position, shifting rightward \( P \) by taking maximum of these two values.\[1\]

Practice shows that BM Algorithm is fast in the case of larger alphabet. In preprocessing phase, time and space complexity is \( O(m+\sigma) \), where \( \sigma \) is the size of the finite character set relevant with pattern and text. In searching phase time complexity is in \( O(mn) \). There are \( 3n \) text character comparisons in the worst case when searching for a non periodic pattern. Under best performance time complexity is \( O(n/m) \). Under the worst time complexity is \( O(mn) \).\[1\]

Advantages
- The both good-suffix and bad-char combined provides a good shift value as maximum of two is taken as shift value.

Disadvantages
- The preprocessing of good-suffix is complex to implement and understand.
- Bad-char of mismatch character may give small shift, if mismatch after many matches.

Example: We have a text string "STRINGMATCHINGISTOFINDTHEPATTERN". And a pattern "PATTERN" which is to find in a text string, so we apply all above algorithm as discussed below to solve this example. Example of BM is shown in Table 1.

<table>
<thead>
<tr>
<th>STRINGMATCHINGISTOFINDTHEPATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
</tbody>
</table>

**III. IMPROVEMENT OF BM ALGORITHM**

A. BMH Algorithm
The preprocessing of good suffix is hard to be understood and implemented; BMH algorithm only uses the bad characters shift. 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 one of pattern string.\[7\]

In preprocessing phase, time complexity is \( O(m+s) \). In searching phase, time complexity is \( O(mn) \). In the best performance, time complexity is \( O(n/m) \). Practical applications show that BMH algorithm is much more efficient than BM algorithm.\[2\]\[10\]

Example: shown in Table 2.

Advantages
- The concept of Good-suffix is removed so easy to implement.
- In case of mismatch , the shift value is determined by the bad char value of last character instead of character that caused mismatch so more jump is archived using bad char than in BM.

Disadvantages
- The removal of good-suffix sometime may not give shift as much as in BM.
B. BMHS Algorithm

The core idea is in the calculation of Bad char function; consider the situation of the next character, namely the use of the next character T[m] to determine the right offset. If the character does not appear in the matching string is skip that step by pattern length + 1; otherwise, the mobile step= match strings in the far right of the character to the end of the range+1. In the matching process, the mode string must not be asked to compare, it does not match is found, the algorithm can skip as many characters to match the next step to improve the matching efficiency. [3]

BMHS algorithm worst case time complexity is O (mn), the best case time complexity is O (n/m+1). For a short pattern string matching problem, the algorithm is faster. [3]

Example: shown in Table 3

Advantages
- In BMH the maximum shift achieved is equal to pattern length but in BMHS the maximum shift that can be achieved is equal to one more than pattern length.

Disadvantages
- Suppose last character is not in pattern but next-to-last character is in pattern so In state of mismatch less shift is achieved as compared to BMH.

C. BMHS2 Algorithm

The idea of algorithm is when mismatch occur at any position then the Right Shift value is determined by Next-to-Last character and Last character of Text corresponding to Pattern that is T[i+m] and T[i+m-1] where m is length of Pattern.

Now matching start from Last character of Pattern, if mismatch at any position than consider Next-to-Last character (T[i+m]) of Text and find its position in pattern

(1) If not in pattern than right shift by m+1.
(2) If occur at first position than right shift by m.
(3) If occur other than first position than shift calculated is X than
   - Consider Last character of Text corresponding to pattern and calculate shift, if shift calculated by this is X than shift by X.
   - Otherwise shift by m+1.

BMHS2 algorithm worst case time complexity is O(mn), the best case time complexity is O(n), where n is length of text and the maximum moving distance of m+1. [3]

Example: shown in Table 4

Advantages
- This algorithm considers last character and next-to-last character both so it combined advantages of both BMH and BMHS.

Disadvantages
- Searching overhead increases as we have to take care of two characters for calculation of shift.
D. Improved BMHS Algorithm

The improved algorithm uses the comparative order from right to left. Supposing that the pattern string \(P_0P_1\ldots P_{m-1}\) aligns with the part of the text string \(T_{k-m+1}\ldots T_k\).

The preprocessing phase is as follows: construct the array \(Skip[x]\) according to the bad-character rules, in the conditions of \(x \in \Sigma\). In addition, improved algorithm needs to construct \(Num[y]\) which records the times of each character appearing in the pattern string.

The searching phase is as follows: compare the character \(P_{m-1}\) with \(T_k\). When mismatch occurs between \(P_{m-1}\) and \(T_k\), calculate \(Skip[T_{k+1}]\) and \(Skip[T_{k+2}]\). If \(Skip[T_{k+1}]\) is equal with one, the pattern string will shift one point to right. Otherwise, the movement will be determined by the larger one between \(Skip[T_{k+1}]\) and \(Skip[T_{k+2}]\).

When \(P_{m-1}\) and \(T_k\) match successfully, compare the character \(P_{m-2}\) with character \(T_{k-1}\). If the match is successful, continue to comparing \(P_{m-2}\) and \(T_{k-2}\), \(P_{m-3}\) and \(T_{k-3}\), and so on, until the text string is matched completely. If mismatch occurs at \(P_{m-j} \neq T_{k-j}\), calculate \(Skip[T_{k+1}]\) and \(Skip[T_{k+2}]\). If \(Skip[T_{k+1}]\) is equal with one, check \(Num[P_{m-j}]\) whether it is equal with one, if \(Num[P_{m-j}]\) is equal with one, change \(Skip[T_{k+1}]\) to \(m+1\). Then compare between \(Skip[T_{k+1}]\) and \(Skip[T_{k+2}]\), select the larger one as the movement of the Pattern shift. [2]

In preprocessing phase, time complexity is \(O(m+s)\). In searching phase, if the successful match takes place in \(T_i\), it is compared \((i-1)*m\) times before successful matching, and \(m\) times during article \(i\) time of comparison. So it is compared \(i*m\) times.

The time complexity is \(O(mn)\). In the best case, if successful match takes place in \(T_i\), it is compared \(i/(m+2)\) times before successful matching, and \(m\) times during article \(i\) time of comparison. So it is compared \(m+i/(m+2)\) times. The best time complexity is \(O(n/m+2)\). [2]

Example: shown in Table 5

**Advantages**

- Maximum shift that can be achieved using this algorithm is pattern length + 2.

**Disadvantages**

- Calculation of shift using Next-to-Last and Next-to-Next-to-Last character increase searching over head and for that preprocessing of Num[ ] is done which increases preprocessing overhead.

<table>
<thead>
<tr>
<th>STRING MATCHING CIST OFINDTHEPATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PATTERN</td>
</tr>
<tr>
<td>2 PATTERN</td>
</tr>
<tr>
<td>3 PATTERN</td>
</tr>
<tr>
<td>4 PATTERN</td>
</tr>
<tr>
<td>5 PATTERN</td>
</tr>
</tbody>
</table>

E. BMI Algorithm

The BMI algorithm combines with the good-suffix function and the advantages of BMH and BMHS [8][9]. At the same time the BMI algorithm also takes into account the singleness and combination features of the Next-Character and the Last-Character.

<table>
<thead>
<tr>
<th>STRING MATCHING CIST OFINDTHEPATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PATTERN</td>
</tr>
<tr>
<td>2 PATTERN</td>
</tr>
<tr>
<td>3 PATTERN</td>
</tr>
<tr>
<td>4 PATTERN</td>
</tr>
<tr>
<td>5 PATTERN</td>
</tr>
</tbody>
</table>
Define: The Last-Character refers to the rightmost character of each attempting window in text \( T \). The Next-Character refers to the first character on right side of attempting window in text \( T \).

Now if \( m \) comparisons have completed and \( T_i T_{i+1} \ldots T_{i+m-1} = P_1 P_2 \ldots P_{m-1} \), the matching is successful. If \( (P_j = a) \neq (T_{i+j} = b) \) in \((m-j)\)-th comparison, the BMI algorithm calculates the jump shift as below methods. Denote \( T_{i+1} T_{i+2} \ldots T_{i+m-1} = P_1 P_2 \ldots P_{m-1} = u \) and \( T_{i+j} \neq P_j \).

1) Calculate the jump shift using the Last-Character \( d \) in pattern and OneChar function. The algorithm looks up the position of the first occurrence of the Last-Character \( d \) from right to left in \( P_0 P_1 \ldots P_{m-2} \). If found the position, the pattern \( P \) right shifts to align with character \( d \). If not found the position, the pattern \( P \) right shifts to align with right side of character \( d \). Then the algorithm begins to compare in new attempt window.

2) Calculate the jump shift using the Next-Character \( c \) and OneChar function. The algorithm look up the position of the first occurrence of Next-Character \( c \) from right to left in \( P_0 P_1 \ldots P_{m-1} \). If found the position, the pattern \( P \) right shifts to align with character \( c \). If not found the position, the pattern \( P \) right shifts to align with right side of character \( c \). Then the algorithm begins to compare in new attempt window.

3) Calculate the jump shift using the Last-Character \( d \), the Next-Character \( c \) and TwoChar function. Denote \( X \) as the combination of character \( b \) and \( c \), that is, \( X = bc \). The algorithm look up the position of the first occurrence of \( X \) from right to left in \( P_0 P_1 \ldots P_{m-1} \). If found the position, the pattern \( P \) right shifts to align with character \( b \). If not found the position, the pattern \( P \) right shifts to align with right side of character \( b \). Then the algorithm begins to compare in new attempt window.

In the case of mismatch, the BMI algorithm combines three different shift functions to optimize the number of characters that can be skipped during the skip process.

If the Last-Character \( d \) is matching with the rightmost character of Pattern, the algorithm calculates the jump shift using above three methods and takes the maximum value of its results as final jump shift. If failed, the algorithm calculates the jump shift using above method (1) and method (2) and takes the maximum value as final jump shift. [1]

Under best performance the time complexity of BM and BMH algorithm all are \( O(n/m) \), the time complexity of BMHS and BMI algorithm all are \( O(n/m+1) \), but the average time complexity of BMI algorithm is better. [1]

Example: shown in Table 6

**Advantages**
- BMI uses last character, Next-to-Last character and combination of these two characters for calculation of shift means BMI Takes advantages of BMH, BMHS and good-suffix feature of BM for combination of last character and Next-to-Last character.

**Disadvantages**
- In calculation of shift using three different methods and taking maximum of these increases overhead in searching.

**TABLE 6. BMI Example (4 Shift and 11 Comparisons)**

<table>
<thead>
<tr>
<th>STRING MATCHING IST OF FIND THE PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
<tr>
<td>PATTERN</td>
</tr>
</tbody>
</table>

F. CBM Algorithm

The key issue of the CBM algorithm is how to utilize the history comparison information achieved at previous iteration. So we construct a two-dimensional table \( \text{Jump}[m][m] \). \( \text{Jump}[i][j] \) denotes the shift distance of pattern \( P \), when the mismatch at previous iteration appears at \( p[i] \), and the mismatch at current iteration appears at \( p[j] \). This table is only related to pattern \( P \). Once \( \text{Jump}[m][m] \) is constructed, it can be utilized for searching \( P \) in different texts.
The comparison principle of algorithm CBM is shown in Figure 1. Suppose P is at place P₀ at previous iteration, and the mismatch appears at index i of P₀; and suppose P is at place P₁ at current iteration, the mismatch appears at index j of P₁; then P₂, P₀’s new position, must meet following conditions: its substring at B matches with P₁’s substring at B; its character at b does not match P₀’s character at j; its substring at A matches P₀’s substring at A; and its character at a does not matches with P₁’s character at i. Above four matching conditions make a large shift distance Jump[i][j] for pattern P. [4]

In the procedure, the initial values of Jump[i][j] is set to Jump[j] for every i. Then the values increased gradually by test, until it satisfies above four matching conditions. After generating table Jump[m][m], the specific matching process is similar to the BM algorithm.

In the case of small alphabet and long pattern, values in Jump[m][m] that is close to the right column are usually larger than the corresponding values in Jump[m], and the matching efficiency are improved. Binary searching in Computer Science and DNA sequence tests in genetic engineering are such kind of applications. [4]

### IV. COMPARISON AND ANALYSIS

BMH algorithm is more efficient when last character does not occur in pattern. BMHS is more effective than BMH when last character occurs in pattern but next to last character does not occur in pattern. Improved BMHS algorithm is efficient when next to last character and next to next to last character does not occur in pattern. BMHS2 perform better when next to last character does not occur in pattern or occur at first position in pattern. BMI algorithm perform better when Next to Last character does not occur in pattern; Or when Last character does not occur in pattern; Or when combination of Last character with Next to Last character does not occur in pattern. CBM is effective in case of small alphabet and long pattern such as Binary Searching.

**Analysis Based on Example:**

- In our example BM and BMH performance was equal as SHIFT=5 and Comparison=13.
- In case of BMHS SHIFT decreases to 4 but Comparison remains to 13, so we can say that BMHS always perform better than BMH, it totally depends on Input.
- Improved BMHS performance is better than BM, BMH and BMHS as SHIFT=4 and Comparison=12.
- Performance of BMI and BMHS2 is even better than Improved BMHS as SHIFT=4 and Comparison=11.
- In example performance of BMI and BMHS2 is equal but we also can say that there performance remains always same, it is also depends on Input.

### Table 7. Comparison

<table>
<thead>
<tr>
<th>Para/Algo</th>
<th>BM</th>
<th>BMH</th>
<th>BMHS</th>
<th>Improved BMHS</th>
<th>BMHS2</th>
<th>BMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>COMPARISON</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>BEST CASE TIME COMPLEXITY</td>
<td>O(n/m)</td>
<td>O(n/m)</td>
<td>O(n/m+1)</td>
<td>O(n/m+2)</td>
<td>O(n/m+1)</td>
<td>O(n/m+1)</td>
</tr>
<tr>
<td>WORST CASE TIME COMPLEXITY</td>
<td>O(mm)</td>
<td>O(mm)</td>
<td>O(mm)</td>
<td>O(mm)</td>
<td>O(mm)</td>
<td>O(mm)</td>
</tr>
</tbody>
</table>

**Analysis Based on Experiment:**

**Experimental Environment**

Processor: i7
RAM: 8 GB
OS: windows 7
Language: visual C++ runs on visual studios 2008

**Experimental Data**

Text File: of size 2, 68,196 KB in which large number of occurrence of pattern.
Pattern of length 15

**Experiment**

In the experiment we have search a pattern in text and calculated number of comparison which is how many times we compare pattern character with text character and search time is also calculated in milliseconds. The results as search time and number of comparison, corresponding to different algorithm are shown in table 8.

**Table 8: Experimental Results**

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Para/Algo</th>
<th>No. of Comparison</th>
<th>Search Time (millisc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BM</td>
<td>26112769</td>
<td>108</td>
</tr>
<tr>
<td>2</td>
<td>BMH</td>
<td>26220229</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>BMHS</td>
<td>26041122</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>Improved BMHS</td>
<td>20238289</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>BMI</td>
<td>20023363</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>BMHS2</td>
<td>19199507</td>
<td>73</td>
</tr>
</tbody>
</table>
On the basis of experimental results we plot bar graphs for comparison and search time as shown in graph1 and graph2.

Graph1: Number of Comparison of Different Algorithm

Graph2: Searching Time of Different Algorithm

V. CONCLUSION

The comparison of BM and its relative algorithm is performed on the basis two factors; one is number of comparison performed and second is search time. In example and in experiment we present a comparison on the basis of number of comparison performed that performance of BM, BMH and BMHS are almost equal as number of comparison is almost same. Improved BMHS perform better than BMHS as number of comparison decreases. BMI and BMHS2 perform even better than Improved BMHS as number of Comparison decreases. In Experiment we also present a comparison on the basis of search time in which BM and BMH perform almost same but BMHS search time increases. Improved BMHS search time is less in comparison to BM, BMH and BMHS. In BMI searching is faster than above four and BMHS2 search time is even less than BMI. So finally we can say that BMHS2 is best of all six algorithms as search time and number of comparison both are less than in all other algorithm.

Composite Boyer-Moore algorithm is efficient in case of binary searching where small varieties of alphabet and long pattern.

The performance of algorithm depends on two factors, first on Input, number of inputs and type of inputs, Second is Methodology of algorithm, so there may be possible that some variation in performance occur as input changes.

VI. FUTURE WORK

The focus of future work is to improve existing algorithm and finding the efficient string searching algorithm so that searching speed can be increased and performance as well.

REFERENCES

[2]. Yuting Han, Guoai Xu. Improved Algorithm of Pattern Matching based on BMHS. 978-1-4244-6943-7/10 2010 IEEE.

AUTHORS PROFILE


Dr. Nilay Khare, Associate Prof. and Head in Department of Computer Science in Maulana Azad National Institute of Technology,Bhopal. Reviewer of Journal Elsevier