High Concurrence Wu-Manber Multiple Patterns Matching Algorithm

Baojun Zhang, Xiaoping Chen, Xuezeng Pan, and Zhaohui Wu
Computer Science, Zhejiang University, Hangzhou, China
Email: zbjhover@zju.edu.cn, xzpan@cs.zju.edu.cn

Abstract—Short patterns affect the performance of Wu-Manber algorithm greatly. In order to solve this problem, an improved Wu-Manber algorithm is proposed. It divides all the patterns into different sets according to their length. For each set, independent data structures are established and different process methods are used. Because there are few resources shared among these sets, high concurrence is obtained when doing pattern matching. Experimental results demonstrate that the improved algorithm represents a high performance than the original one.

Index Terms—multiple pattern matching, hash operation, prefix, suffix, match window, block character, concurrence

I. INTRODUCTION

Multiple pattern matching can search multiple patterns in a text at the same time. It has a high performance and good practicability, and is more useful than the single pattern matching.

Classical multiple pattern matching algorithms include Aho-Corasick algorithm (AC) [1], AC-BM algorithm [2] and Wu-Manber algorithm (WM) [3].

AC is based on FSA, before the pattern matching, it preprocesses the pattern set and builds a FSA tree, then after one times of text scan, all the matched patterns can be searched. For an “n” size text, the time complexity of pattern matching of AC is \(O(n)\) and has nothing to do with the number and length of patterns. Think about the time of preprocessing, the time complexity of AC is \(O(n+M)\). “M” is the total length of patterns.

AC-BM combines AC and BM [4] algorithms. Similar to AC, it changes the pattern set to a FSA tree. The difference is that AC uses the suffix of patterns, but AC-BM algorithm uses the prefix. When doing pattern matching, it compares string from back to front and uses the technology of bad character shift and good prefix shift of BM algorithm. The time complexity of AC-BM is \(O(nM)\).

WM algorithm was proposed by Sun Wu and Udi Manber in 1994. It uses the jump idea of BM and hash function. WM algorithm is one of the most quickly multiple pattern matching algorithms. In addition, WM algorithm is not sensitive to the character set and can be used for Chinese.

WM algorithm proposes a very good idea in pattern matching, but in actual application, many problems can be improved further. At present, there are many improved WM algorithms. A modified WM algorithm (MWM) is used in Snort [5]. It uses the SHIFT table and prefix hash table. Add the function of prefix filter to the original WM algorithm and obtain a good effect. In [6], a prefix table is added to quicken the pattern matching. In [7], the idea of QS algorithm [8] is adopted to enhance the shift distance. In [9], HASH table is modified, and the link list of suffix patterns is added to avoid the missing of matching, so when a pattern in the link list is matched, the subsequent patterns need not to be matched any more. In [10], think about the effect of short patterns, divide all patterns into two groups, the short pattern group and the long pattern group, and then do two times of pattern matching with these two pattern groups in turn. In [11], a table contains the end characters in all patterns is created, by comparing current block character with the end characters in table, avoid the hash calculation, enhance the speed of pattern matching.

In this paper, the effect caused by short patterns is considered and a high concurrence WM algorithm (HCWM) is proposed. The paper is organized as follows. Section 2 introduces the WM algorithm in detail, and indicates the limitations of the algorithm. Section 3 gives our resolvents for these problems, and introduces HCWM algorithm detailedly. Section 4 is the experimental results and performance analysis. Section 5 is the conclusion.

II. WU-MANBER ALGORITHM

A. Introduce to Wu-Manber Algorithm

WM algorithm has two core mechanisms, the filter mechanism based on hash technology and the block character shift mechanism based on the bad character shift technology from BM algorithm.

By calculating the hash value of the suffix block character in patterns, the patterns with same suffix are linked in a list. The list entry is stored in a hash table. When searching a text, we calculate the hash value of the block character inside current match window, and lookup the hash table to get the entry of the patterns that contain the same block character as their suffix. Obviously, these patterns are possible matching strings. Symbol “\(B\)” is used to represent the size of the block character. It usually is 2 or 3. In this paper, we suppose “\(B = 2\)”.

When a matching is finished, the match window should right shift. The size of match window is determined by the length of the shortest pattern. Here we suppose it is “\(m\)”. The bad character shift technology is used for the match window shift. The only difference is
that the single character is replaced by the block character. The bad character shift technology insures the match window can shift to the best of its abilities. It quickens the pattern matching very much.

The operational process of WM algorithm includes two stages, the preprocessing and the pattern search.

1) Preprocessing

During the preprocessing, the size of the match window should be determined first and three important tables should be established, a SHIFT table, a HASH table and a PREFIX table. The SHIFT table stores the shift distance of the block character occurred in text. The HASH table stores the entry of the link list which links all the patterns with the same suffix inside the match window. The PREFIX table stores the entry of the link list which links all the patterns with the same prefix inside the match window.

Suppose text “T = t1, t2, ..., tn”, “n” is the length of “T”, pattern set “P = {P1, P2, ..., P m}”, match window size is “m”, block character size “B = 2 B”.

(1) SHIFT table establishment

Suppose “X” is a text block character inside current match window, and by hash calculation, “X” is mapped to position “i” in SHIFT table, i.e. \( hash(X) = i \).

The shift distance in SHIFT table is calculated as follows.

\[
SHIFT[i] = \begin{cases} 
m - B + 1, & \text{if “X” does not occur in any pattern strings.} \\
\min\{m-q|P_q=B+k|, 1 \leq k \leq B, P_q \in P\} & \text{otherwise}
\end{cases}
\]

In equation (1), “q” is the rightmost place that “X” occurred in the patterns.

(2) HASH table establishment

When establishing a HASH table, the size of match window should be determined first. It is the length of the shortest pattern strings. Here we suppose it is “m”. Second, get “m” characters in the front of patterns and calculate the hash value of the block character that the suffix of the “m” characters. Last, link all the patterns with the same hash value together in a list and store the entry of the list in the HASH table.

(3) PREFIX table establishment

The establishment of PREFIX table is similar to the HASH table. Instead of suffix of the “m” characters, the prefix is used for hash calculation.

2) Pattern search

The process of pattern search in WM is as follows.

(1) Locate match window at the beginning of text “T”. Make the text pointer “T_p” points to the suffix of the match window.

(2) If \( T_p \geq T_{end} \), it means current text pointer beyond the end of the text, and the operation should be finished. Or else, calculate the hash value of the block character at the suffix, such as \( hash(sbc) \). “sbc” means the suffix block character.

(3) Use the hash value to lookup the SHIFT table for shift distance of the block character. It is \( SHIFT[hash(sbc)] \). If the value is “0”, means that the current block character is equal to the suffix block character of some patterns inside the match window, then goto step (4). Or else, \( T_p = T_p + SHIFT[hash(sbc)] \), and goto step (2).

(4) Use the hash value “hash(sbc)” to lookup the HASH table, and find the entry of the patterns which have the same suffix block character as current text inside the match window. The entry is a pointer which points to the head of the link list and is stored in \( HASH[hash(sbc)] \).

(5) Calculate the hash value of the prefix block character of text “T” inside current match window, such as \( hash(pbc) \). “pbc” means the prefix block character.

(6) Calculate the hash value of the prefix block character for each pattern in the link list, only when the value equal to \( hash(pbc) \) does this pattern need to be matched. If the pattern is matched with current text string, record the result. Whether it matched or not, \( T_p = T_p + 1 \), and goto step (2).

B. Testing and Analysis to Wu-Manber Algorithm

1) Testing environment

Testing data: File “email.pst” in English exported from outlook, size 4, 576KB. The patterns are extracted from the file random.

2) Testing result

The length of short patterns is equal or less than three characters. In order to observe the effect of short patterns on the performance of WM, two groups of testing are made.

(1) Set the number of patterns to be “100”. Observe the effect on search time as the length of patterns increasing.

![Figure 1. Relations between search time and the length of patterns in WM.](image1.png)

(2) Generate 100 patterns with the length 6. Observe the effect on search time as the ratio of short patterns increasing.

![Figure 2. Relations between search time and the ratio of short patterns to long patterns in WM.](image2.png)
3) **Algorithm analysis**

From Figure 1, when the length of patterns is short, the search time is long. As the length of the patterns increasing, the search time shortens obviously. It shows that the search time of double-character patterns is a little longer than the single-character patterns. That is because these two kinds of patterns have the same shift distance “1”. When the shift distance is same, the matching time of double-characters is longer than the single-characters.

From Figure 2, when all the length of patterns is long, the search time is short. When one short pattern is added in the pattern set, the search time increases rapidly. As the ratio of short patterns increasing, the search time will fall gradually, but longer than the original situation all along.

It is obvious that short patterns have a great effect on the performance of WM. Through in-depth analysis, the following reasons were found.

(1) The bad character shift technology is meaningless to the short patterns. The bad character shift technology used in WM guarantees the performance of WM. But when the match window is small, the bad character shift technology will lose its advantages. The size of match window is determined by the length of the shortest pattern. Once there exist single-character or double-character pattern, the shift distance will always be “1”, and the performance will fall greatly.

(2) The filter mechanism based on hash technology has little effect on the short patterns. Hash technology is used to filter the block characters in text, through suffix and prefix the two layers of filtering, the speed of pattern matching is highly enhanced. But for short patterns, the filter mechanism loses its functions. In addition, for some long patterns which contain the short patterns, the partial matching is often occurred. It affects the performance of WM seriously.

So, for the short patterns, special processing should be taken in WM.

**III. HCWM ALGORITHM**

HCWM is the improved version to WM. In HCWM, the short patterns get good processing. When the long patterns and short patterns mix together, it shows good performance.

**A. Improvement in HCWM**

In HCWM, the following improvements are made.

(1) Divided all patterns into four pattern sets “PS1-PS4” according to their length, the single-character patterns, the double-character patterns, the tri-character patterns, and the long-character patterns whose length are equal or greater than four characters.

(2) For “PS1” and “PS2”, the shift distance is always “1”, the SHIFT table is not required. For “PS3” and “PS4”, “SHIFT3” and “SHIFT4” are established respectively. The method is the same as WM.

(3) Four HASH tables “HASH1-HASH4” are established for “PS1-PS4”. Except “HASH4”, the other HASH tables are established by the hash value of the whole pattern. For “HASH4”, the method is the same as WM. Use the suffix block character to calculate the hash value.

(4) When doing pattern matching, different operations are taken for different pattern sets. For “PS1”, the shift distance is always “1”. The process of matching is to calculate the hash value of the text character in current match window and lookup the “HASH1” table. For “PS2”, the operation is similar to “PS1”, the only difference is that the size of match window and block character are both “2”. For “PS3”, lookup the “SHIFT3” table to obtain the shift distance. The match process is the same as “PS1” and “PS2”, that is, lookup “HASH3” with the hash values of the text characters inside match window. For “PS4”, the shift and search process is the same as WM.

(5) Concurrent execution. Since each pattern set has independent data structures, and the common text file is read only, the search operation can execute concurrently. Multiple threads are used for the pattern matching.

**B. Flow of HCWM**

HCWM has two stages, the preprocessing and the pattern search.

1) **Preprocessing**

The preprocessing stage includes the following works.

(1) Divide all patterns into four pattern sets “PS1-PS4”.

(2) Determine the size of match window for “PS4”.

(3) Establish HASH table “HASH1-HASH4” for each pattern set.

(4) Establish SHIFT table “SHIFT3”, “SHIFT4” for “PS3”, “PS4”.

(5) Establish PREFIX table “PREFIX4” for “PS4”.

2) **Pattern search**

HCWM can search pattern sets “PS1-PS4” concurrently. For every pattern sets, the search process is different.

a) **Search “PS4”**

The search process of “PS4” is the same as WM.

b) **Search “PS3”**

Search “PS3”, the size of match window is “m = 3”, the length of block character is “B = 2”.

(1) Locate match window at the beginning of text “T”.

Make the text pointer “T_p” points to the suffix of the match window.

(2) If \( T_p < T_{end} \) end the search for “PS3”. Or else, goto step (3).

(3) Calculate the hash value “sh” of the suffix of the text characters inside the match window and lookup “SHIFT3”, if “SHIFT3[sh] > 0”, \( T_p = T_p + \text{SHIFT3[sh]} \), return to step (2). Or else, goto step (4).

(4) Calculate the hash value “h” of the text characters inside the match window, lookup “HASH3”, if “HASH3[h] != null”, the pattern is matched, record the result. \( T_p = T_p + I \), goto step (2).

c) **Search “PS2”**

Search “PS2”, the size of match window is “m = 2”, the length of block character is “B = 2”.

(1) Locate match window at the beginning of text “T”.

Make the text pointer “T_p” points to the suffix of the match window.
(2) If $T_p > T_{end}$, end the search for “PS2”. Or else, goto step (3).

(3) Calculate the hash value “$h$” of the text characters inside the match window, lookup “HASH2”, if “HASH2[ $h$] != null”, the pattern is matched, record the result, $T_p = T_p + 1$, goto step (2).

d) Search “PS1”

Search “PS1”, the size of match window is “$m = 1$”, the length of block character is “$B = 1$”. The process is the same as “PS2”.

C. Example for HCWM

An example is taken to explain the flow of HCWM in this chapter. Suppose that we have the following data.

Text “$T$” = “The mild teacher always told her students to obey the rules of school.”

Patterns, “$P$” = {“mild”, “tea”, “told”, “micky”, “ld”, “tu”, “t”, “m”}

1) Preprocessing

(1) Classify the patterns.

“PS1” = {“t”, “m”}, “PS2” = {“ld”, “tu”}, “PS3” = {“tea”}, “PS4” = {“mild”, “told”, “micky”}

(2) Determine the size of match window for “PS4”, it is “$m = 4$”.

(3) Establish HASH table, refer to Figure 3.

(4) Establish SHIFT table, refer to Figure 4.

(5) Establish PREFIX table, refer to Figure 5.

<table>
<thead>
<tr>
<th>hash(mi)</th>
<th>micky</th>
<th>mild</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash(to)</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

2) Pattern search

(1) Search “PS4”

The size of match window is “4” and is located at the beginning of the text.

The mild teacher always told her students to obey the rules of school.

The suffix block character inside the match window is “e ”. Lookup “SHIFT4”. Since the block character doesn’t occur in any patterns in “PS4”, the shift distance is “3”.

The mild teacher always told her students to obey the rules of school.

Use the hash value of “il” to lookup “SHIFT4”, match window right shift “1” character.

The mild teacher always told her students to obey the rules of school.

Similarly, pattern “told” is matched also. When the match window reaches “scho”, the shift distance is “3”, it is beyond the end of the text, so end the search.

(2) Search “PS3”

The size of match window is “3” and the match window locates at the beginning of the text.

The mild teacher always told her students to obey the rules of school.

Similarly, pattern “told” is matched also. When the match window reaches “scho”, the shift distance is “3”, it is beyond the end of the text, so end the search.

(3) Search “PS2”

The size of match window is “2”. The shift distance is always “1”. Calculate the hash value of the text characters inside the match window and lookup “HASH2”. If the table value is not null, the pattern is matched. Continue the search till the end of the text.

(4) Search “PS1”

It is similar to “PS2”.

Figure 3. HASH table.

Figure 4. SHIFT table.

Figure 5. PREFIX table.
IV. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

Under the same test environment in Chapter II, we compare the performance of WM and HCWM by the following three groups of testing.

(1) The length of patterns is “6” characters all the time. Observe the effect on search time as the number of patterns increasing.

From Figure 6, we can draw the following conclusions. First, as the number of patterns increasing, the search time will increasing. It is normal and accord with the logic. Second, for long patterns, the search time of HCWM and WM is similar. It is because that they take the same process methods for long patterns.

(2) The number of patterns is “100” all the time. Observe the effect on search time as the length of patterns increasing.

From Figure 7, we can draw the following conclusions. First, the performance of short patterns is worse than the long patterns obviously. The reason is that the bad character shift and hash filtering technologies are meaningless to short patterns. Second, for short patterns, the performance of HCWM is better than WM. The reason is that when all the patterns are short, the shift distance in WM and HCWM is the same. But WM need two steps to match a pattern, first, use the hash value of text character inside match window to find the entry of the link list. Second, compare the patterns. HCWM only need to judge whether the list entry is null or not. It needs not to compare the patterns.

(3) Generate 100 patterns with the length 6. Observe the effect on search time as the ratio of short patterns increasing.

From Figure 8, we can draw the following conclusions. First, when there are short patterns mixed in long patterns, the performance will fall seriously. The reason is that the size of match window is determined by the shortest patterns, which affects the shift distance directly. And when long patterns and short patterns are mixed together, lots of partial matching for long patterns will occur, it causes many invalid matching. Second, as the ratio of short patterns increasing, the performance will become better. The reason is that the effect caused by partial matching is weakened as the number of long patterns decreasing. Third, when short patterns and long patterns mix together, the performance of HCWM is better than WM obviously. The reason is that HCWM processes long patterns and short patterns in different ways. Avoid the effect to long patterns caused by short patterns. In addition, independent data structures are used for each type of patterns, a high concurrence is obtained. The use of multiple threads makes the HCWM a good performance.

According to above analysis, HCWM debases the effect caused by short patterns, and shows good performance than WM.

V. CONCLUSION

This paper researches the WM algorithm in-depth. Aiming at the limitations of WM when dealing with the short patterns, a high concurrence WM algorithm, the HCWM, is proposed. By dividing all patterns into different pattern sets according to their length and processing these sets respectively, it debases the effect on performance caused short patterns. In addition, independent data structures are used for different pattern sets, the high concurrence is obtained, which enhances the matching speed of HCWM greatly. Experimental result shows that HCWM represents a high performance than WM.

REFERENCES


