Efficient Wu Manber String Matching Algorithm for Large Number of Patterns

Vasudha Bhardwaj
LNCTE, Bhopal

Vikram Garg
LNCTE, Bhopal

ABSTRACT

String matching is one of the most important concept used in computer science in various real life applications like as Intrusion detection system, Data mining, Plagiarism detection system. There are many string matching algorithms which help to find pattern from the text. These algorithms are categorized in single string matching and multiple string matching. The Wu-Manber (WM) algorithm is multiple patterns algorithm which is the finest string matching algorithm. The performance of WM depends on various table build in pre processing phase these are prefix table, shift table and hase table. We introduce a new algorithm namely the Efficient Wu Manber algorithm (EWM) algorithm which is advance version of Wu Manber algorithm with respect to time. Efficient Wu-Manber Algorithm eliminate the prefix table which is unused most of the cases in wu manber, construct two shift table instead of single shift table and uses nonlinear data structure i.e. AVL tree instead of linear data structure i.e. linked list used in WM in Hash table, which reduce the traversed number of nodes to find exact match. The experimental results and analysis show that EWM algorithm has better performance as compare to WM and its existing improved algorithm and also better from various string matching tools.

Keywords

Wu-Manber, String Matching, Single pattern matching, Multiple pattern matching, Boyer Moore, KMP, Advance Wu Manber.

1. INTRODUCTION

Sting matching algorithms [1] plays important role in the real life applications of the computer science world. There are lots of algorithms for string matching. Morris Pratt Algorithm[2], Knuth-Morris Pratt Algorithm[3], Boyer-Moore Algorithm[4], Boyer Moore Horspool algorithm[5] and Robin Karp [10] are some benchmark algorithms of single pattern matching while Aho-Corasick [6] algorithm, Commentz-Walter [7] algorithm and Wu- Manber [8] algorithm are some benchmark algorithms of multi-pattern matching. These algorithms are further classified into two categories (a) automata-based algorithms and, (b) hashing based algorithms.

Automata based algorithm carried out into two phase in first phase automata is created based on patterns while in second phase searching is done based on automata. Aho-Corasick algorithm and its all variants are representatives of automata based algorithms and the Wu-Manber algorithm and its all variants are comes under hashing based algorithms. Automata approach has a linear-time complexity. As we have to make an automata for each and every pattern memory requirement as well as complexity of the algorithm is also increases.

Wu Manber [8] string matching algorithm is implemented by the Sun Wu and Udi Manber which works on multiple patterns in 1994. Wu manber algorithm works in two steps pre-processing and scanning. In pre-processing phase various tables are constructed these ideas is taken from Boyer-Moore (BM) [4]. Other hand in scanning phase searching is done based on the various tables. This algorithm gives the better results and takes less space with compares to automata based algorithm.

Based on Wu Manber many varients are proposed to overcome limitation of Wu manber algorithm. Quick Wu-Manber (QWM) [9] is implemented after the wu manber in 2006 by Yang Dong Hong, Xu Ke and Cui Yong which is based on Quick Search [12]. In wu manber algorithm when suffix are same so we have to find out the all pattern which ahs same suffix. To overcome this limitation QWM construct another table named as Head table which keep the details of first two characters of the patterns. When suffix of pattern is same it checks the head table value and reaches the maximum shift distance in comparison to WM algorithm.

Improved Wu Manber is the another variant of wu manber algorithm. Improved WM [11] is implemented by Chen Zhen and Wu Di in 2008, in improved WM algorithm implement two shift table instead of single shift table. First shift table is constructed based on Boyer more algorithm [4] while second shift table is created based on Wu Mnaber algorithm[8]. By doing so the shift distance is increases which improve the performance of algorithm.. But when the number of patterns reaches 40,000 or more the introduced extra shift table still cannot improve the performance.

In 2009 an improved WM algorithm based on address filtering named as AFWM [13] was proposed by Baojun Zhang and others. Based on the address pointers of the patterns the Prefix table in AFWM is utilized to filter the link list of possible matching patterns. The patterns in the link list are sorted in ascending order according to the address pointers [13]. Advantages of address filtering algorithm is that it avoids traversing the whole link list [13].

In 2013 Yoon Ho proposed B-LAyered bad-character Shift Tables (BLAST) [14] algorithm. The idea of BLAST is use multiple shift tables of single character (gives us the advantage of maximum shift distance) instead of block of character. Here the memory size of shift table is also reduced. But if every character in the rightmost position of the patterns is present the performance of BLAST algorithm decreases because the Hash table will be compared for every character in the text [14].

We have proposed here an algorithm named Efficient Wu-Manber or EWM based on Wu-Manber. It improves the performance of Wu-Manber when numbers of patterns are very large. The Main change made to EWM is that it uses nonlinear structure i.e. AVL tree which reduce the traversed number of nodes to find exact match. In the following section we have given an overview of WM algorithm then described EWM algorithm. Next section includes experimental results and analysis and in the last we conclude this paper.

2. WU-MANBER ALGORITHM (WM)

The Wu-Manber algorithm is an hasing based algorithm which uses the concept of bad character shift from boyer more algorithm. These bad character shift value helps to get the maximum shift distance in case of mismatch of pattern. Wu-Manber algorithm calculates the hash value of the suffix block of a pattern using the Hashing technique and links all the patterns with the same suffix block in a list. This list is stored in an entry of the hash table. The hash value is calculated for the character block inside the match window and is checked in the hash table to get the entry having the same suffix patterns as the character block [8].

The Wu-Manber algorithm is divided into two stages which are described as follows:

1. Pre-processing Stage
The pre-processing stage comprises of construction of various tables which are further used in the scanning stage of the algorithm. WM consist of three tables named as Prefix table, the Hash table and the Shift table. First of all minimum pattern length (say m) is is search among the all patterns. WM uses only first m characters (known as pattern representative or PR) of the each pattern to build these tables.WM take characters in a block instead of taking them one by one and represent this block with a symbol B, 2 or 3 is a good value for B [8]. Let BL is a string of size B. WM build entries in all tables for all possible BL.

The shift table consists of the maximum safe shift distance for the matching window used in scanning stage. Each string of size B is mapped to an index to the SHIFT table. Hash table value is a pointer to a list of pattern with PR having same suffix. Prefix table is similar to hash table but the difference lies in the key being the prefix instead of suffix of the PR. Hash Table and Prefix table are used when the shift value is zero.

The above process can be understood for the pattern set P = {honey, funeist, list, money}; here m is 4. Table 1 and Figure 1 represent the shift table and hash table for the pattern set respectively.

Table 1: Shift Table for WM

Key	Ho	on	ne	fu	un	ne	li	is	st	mo	*
shift	2	1	0	2	1	0	2	1	0	2	3

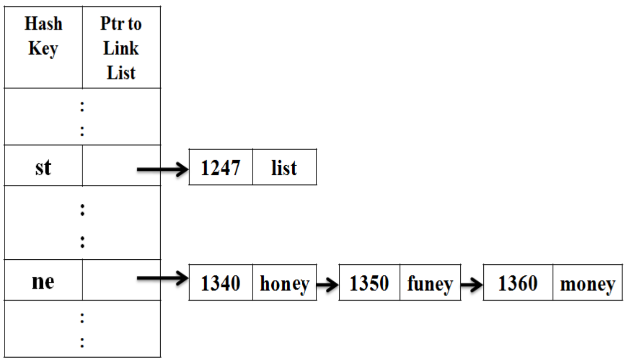

Figure-1 Hash table for WM

2. Scanning Stage
Scanning step in WM algorithm comprises the use of a sliding window of size m which starts from initial character of text. First hash value of the window is computed for suffix of window. If hash value is greater than zero then window is shifted with the same value forward in text character stream. When shift value is zero then Hash table is checked and a list of possible matched candidates is found using suffix hash value. Whole of the retrieved list is traversed in this procedure. The exact comparison to find a match is done when the prefix of pattern in the list is same as prefix of the window. After traversing whole list window is slided by one and whole process repeated till window reached the end of text. For text t= "funeyneedmoney" scanning stage of Wu-Manber is described in Table 2:

Table 2: Scanning stage in Wu-Manber

Step	Text	Shift	Node traversed	Output
1	fu neyneedmoney	0	1	funey, shift by 1
2	funeyneedmoney	3	-	
3	funeyneedmoney	3	-	
4	funeyneedmoney	2	-	
5	funeyneedmoney	0	3	money, shift by 1
6	funeyneedmoney	3	-	

3. EFFICIENT WU-MANBER ALGORITHM
This algorithm is an efficient version of WM Algorithm with some changes in hash table data structure which leads to a better running time for scanning stage. WM Algorithm uses linear data structure i.e. Linked List. While in EWM Algorithm we use nonlinear data structure i.e. AVL tree which reduces the number of possible matching patterns to find exact matching patterns with current window. EWM has two stages similar to WM:

1. Pre-processing Stage

In pre-processing stage various tables are created which helps in searching the pattern in scanning stage. Here in EWM algorithm there is no prefix table and instead of single shift table it construct two shift table. One is constructed on the basis of the single character while other is based on the block of character. While hash table is differ with respect to data structure used. The Pre-processing Stage is done in following steps:

1. ***Find out the minimum pattern length among all patterns & denoted as m(min pattern length).***
2. ***Take only first m letters of each patterns for further process & denoted as PR(Pattern Representative) to which they belong.***
3. ***Shift table Creation using PRs.***
4. ***Hash Table Creation using PRs.***

Constructing Shift table in EWM is same as used in WM there is only difference in Hash table which is described below.

Constructing HASH Table
The last B characters of each PR are used to determine index of pattern in HASH table. Hash table value points to an AVL tree. The structure of an AVL Node contains a left pointer, right pointer, pointer to a linked list and key value of node. The key of AVL node is determined by 1st and 2nd character of PR. The value of an AVL node is the Linked List of patterns with the same node key. The method to form a HASH table is as follows:

```
1.  Consider last Bcharacter hash value for each PR.
2.  Construct Hash table based on this.
        1.  Add PR to AVL tree if hash table already
            has an entry.
        2.  Else PR as a root node Create new entry
            in Hash Table.
```

For previous example HASH Table can be form as follows: for pattern = “honey”; suffix value = “ne” HASH [ne].insert (“honey”); The final HASH table will be as follows:

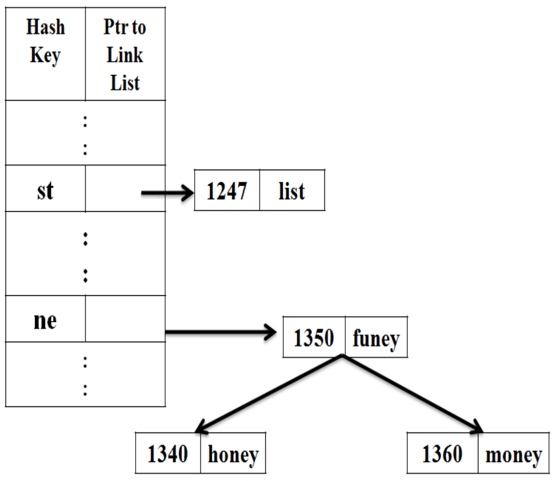

Figure 2: HASH Table for Efficient Wu-Manber

3. Scanning Stage

In scanning step Efficient Wu-Manber Algorithm maintains a window of size m and starts from the beginning of Text. This stage consists of the following steps:

```
1.  Start
2.  Start from left most place of text
3.  Consider window with min pattern length
4.  Take last B characters of window WB
5.  dot
6.  shift [WB] is greater than zero
        a.  Shift the window by value shift[WB]
        b.  Continue;
7.  Otherwise
        a.  Compute the hash value from hash table
            which is in form of AVL tree.
        b.  Compute node value
        c.  Find the node by traversing the AVL tree
            with same key.
                i.   Successful
                     Match the pattern store at that
                     place of node.
                ii.  Unsuccessful
                     Shift window by 1.
8.  while(WB<=Length of text)
9.  End
```

Figure 3: Scanning steps for Efficient Wu-Manber

For the patterns taken in previous example the scanning step for the Text “honeyneedmoney” is shown as follows:

Table 3: Scanning stage in Efficient Wu-Manber

Step	Text	Shift	Node traversed	Output
1	fu neyneedmoney	0	1	funey, shift by 1
2	funeyneedmoney	3	-	
3	funeyneedmoney	3	-	
4	funeyneedmoney	2	-	
5	funeyneedmoney	0	2	money, shift by 1
6	funeyneedmoney	3	-	

4. PERFORMANCE ANALYSIS

The performance of EWM algorithm can be measured in terms of the performance of scanning stage as this process is repeated again and again for different Texts which have same set of patterns. Pre-processing can be done offline just once for a fixed set of patterns. Scanning stage performance depends on two factors one is the shifting of window and other one is finding exact matching patterns.

In WM and EWM number of entries are same in hash table as we are using same suffix and prefix technique. Let there are ‘n’ Pattern in the structure(linked list in WM and AVL tree in EWM) retrieved for current window suffix then finding exact matching patterns in best case(when all prefix are different for these PR’s) takes just O(log(n)) which is O(n) in WM . But in worst case (when all the prefixes are same for all PR’s) it will be O(n).So except worst case in all other cases time taken to find exact matching pattern is less in EWM.

Here we described an another example to compare EWM and WM with respect to the number of node traversed and number of exact matching is done in both algorithms.

For a pattern set P= {abdication, aberration, abjuration, abnegation, absolution, abstention, abreaction, absorption, unconscionable, undulation, unquestionable, unillusioned, unsanctioned, unsynchronized, recitation, recreation, redemption, redivision, reelection, reemission, reflection, refraction, regulation, repetition, reposition} and Text = “try absorption repetition and reposition”. The performance comparison is shown in Table 4. Nodes traversed in EWM are always less than or equal to node traversed in WM.

Table 4: Comparison of WM and EWM

Shift	WM		EWM	
	Node Traversed	Exact Matching	Node Traversed	Exact Matching
4	-	-	-	-
0	25	8	6	3
9	-	-	-	-
1	-	-	-	-
0	25	11	5	2
9	-	-	-	-
5	-	-	-	-
0	25	11	5	2
9	-	-	-	-

5. EXPERIMENTAL RESULTS

In this section we compare our algorithm’s performance with WM and other tools such as agrep, egrep and fgrep.

Experimental results show that EWM is faster than WM and other tools. In our experiment alphabet set and text file is constant for all test cases. Text file used in all test cases is taken from Bible whose size is 101 MB. All the patterns used in our experiments are words taken from English Dictionary. We have tested all these tools on different variants of patterns.

We analyzed the performance with minimum pattern length, number of patterns and block size (B). All experiments were performed on a system Ubuntu, with Linux 3.2.6 (64bit) on 2nd Gen Intel CORE i3-2310M 2.10 GHz, 4GB RAM. WM and EWM were implemented in C++ using Microsoft Visual Studio 2010 as IDE. Each experiment was performed 10 times and the average was taken.

Table 5 shows relationship among EWM (B=2), WM. Figure 4 shows the graphical comparision of table 5 where X-axis represents the number of patterns and Y-axis represents running times taken in scanning step given in seconds. The results clearly show that EWM performs better than WM algorithms with the increase in number of patterns.

Table 5: comparison of EWM with WM timing (in sec.) with various pattern lengths

No. of Patterns	WM	EWM
50	0.995	0.482
100	1.146	0.547
150	1.586	0.673
250	1.794	0.785
500	1.919	0.873

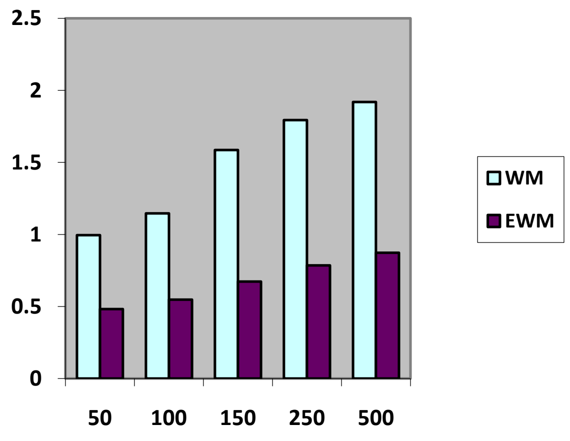

Figure 4: Graphical comparison of EWM with WM timing (in sec.) with various pattern lengths

Table 6 shows the comparison of proposed algorithm EWM with the various tools like as EGRAP, FGRAP, AGRAP with different pattern lengths. On comparison of various tools our algorithm gives a better result which is shown in figure 5 graphically.

Table 6: comparison of EWM with various tools timing (in sec.) with various pattern lengths

No. of patterns	EGREP	FGREP	AGREP	EWM
50	3.052	3.119	2.990	0.482
100	4.957	5.529	4.859	0.547
150	6.369	7.197	5.965	0.673
250	9.086	9.799	8.280	0.785
500	35.077	13.585	11.62	0.873

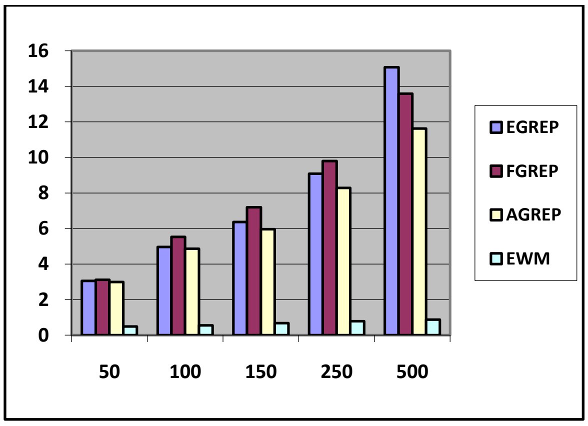

Figure 5: Graphical comparison of EWM with various tools timing (in sec.) with various pattern lengths

6. CONCLUSIONS

We have introduced here a new algorithm EWM for multi pattern exact matching which is efficient version of the WM algorithm. In this algorithm we eliminate the prefix table which is used in the WM. Instead of single shift table we use the two shift table one for single character and another for block of character. While in hash table we use AVL tree instead of linear data structure. By doing all of these changes the proposed algorithm gives us better result as compares to WM algorithm and some other tools of pattern matching. Our experimental result and shows that proposed algorithm is efficient in both respect (time as well as memory).

7. REFERENCES

[1]. Christian Charras and Thierry Lecroq," Handbook of Exact String_Matching Algorithms", Published in King's college publication, Feb 2004.

[2]. Knuth D E, Morris Jr J. H and Pratt V. R," Fast pattern matching in strings", In the procd. Of SIAM J.Comput., Vol. 6, 1, pp. 323–350, 1977.

[3]. Jingbo Yuan, Jisen Zheng and Shunli Ding, "An Improved Pattern Matching Algorithm", In the proc. of Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI), pp. 599-603, 2-4 April 2010.

[4]. Boyer R S and Moore J S,”A fast string searching algorithm”, Communication of ACM 20, Vol. 10, pp. 762–772, 1977.

[5]. Horspool R N,”Practical fast searching in strings”, In proc. Of Software Practical Exp, Vol. 10, 6, pp. 501–506, 1980.

[6]. Alfred v aho and Margaret j corasick,”efficient string matching: an aid to bibliographic search” communication of acm, vol. 18, June 1975.

[7]. Commentz-Walter, “A string matching algorithm fast on the average,” In the Proc. of 6th International Colloquium on Automata, Languages, and Programming, pp. 118–132,1979.

[8]. Wu S. and U.Manber, “A Fast Algorithm for Multi-Pattern Searching,” Technical Report TR-94-17 Department of Computer Science, University of Arizona, Tucson, AZ (May 1994).

[9]. Yang Dong hong, XuKe and Cui Yong,”An improved Wu-Manber multiple patterns matching algorithm”, In the proc. Of 25th IEEE International Performance, Computing, and Communications Conference, IPCCC, pp. 680, 10-12 April 2006.

[10]. R. M. Karp and M. O. Rabin, “Efficient randomized pattern-matching algorithms,” In: (2nd ed.), Tech. Rept. 31-81, Aiken Computer Lab, Harvard University, Cambridge, MA, 1981.

[11]. Chen Zhen and Wu Di, “Improving Wu-Manber: A Multi-pattern Matching Algorithm”, In the proc. of 2008 IEEE International Conference on Networking, Sensing and control (ICNSC), pp. 812 – 817, 6-8 April 2008.

[12]. D.M. Sunday, “A Very Fast Substring Search Algorithm”, Communications of the ACM, Vol. 33, 8, pp. 132-142, 1990.

[13]. Baojun Zhang, Xiaoping Chen, Xuezeng Pan, and Zhaohui Wu “High concurrence Wu-Manber Multiple Patterns Matching Algorithm”, Proceedings of the International Symposium on Information Proces, p.404,August 2009.

[14]. Yoon-Ho,Seung-Woo,”BLAST: B-LAyered bad-character SHIFT tables for high-speed pattern matching”, Journal of Information Security, Institution of Engineering and Technology (IET), Volume 7, pp.195-202,sept. 2013.