International Journal of Advanced Computer Science and Information Technology (IJACSIT)
Vol. 3, No. 4, 2014, Page: 344-353, ISSN: 2296-1739

www.elvedit.com

Analysis of Multiple String Pattern Matching Algorithms

Author

Akinul Islam Jony
Department of Informatics,
Technical University of Munich

akinul@mytum.de
Munich, Germany

Abstract

Multiple string pattern matching is a basic problem in computer science and is used to locate all the appearances of a finite set of patterns inside an input text. It is widely used in many applications for searching, matching, filtering, and detecting a set of pattern. In this paper, to illustrate and for the better understanding of this particular problem, the widely used multiple string patterns matching algorithms have been analyzed and discussed. A theoretical and experimental result along with the analysis and discussion of the algorithms is presented as well in this paper. An extensive reference list is also included at the end of the paper.

Key Words

Algorithms, Multiple Pattern, String Matching, String Searching.

I. INTRODUCTION

String pattern matching or searching is the act of checking for the presence of the constituents of a given pattern in a given text where the pattern and the text are strings over some alphabet. It is an important component of many problems and it is used in many application such as text editing, data retrieval, data filtering (also called data mining) to find selected patterns, DNA sequence matching, detecting certain suspicious keywords in security applications, and of course, many other applications. The string searching or string matching problem consists of finding all occurrences of a set of pattern in a text, where the pattern and the text are strings over some alphabet.

Multiple string pattern matching problems has been a topic of intensive research that has resulted in several approaches for the solution such as multiple keyword generalization of Boyer-

Moore algorithm [4], Boyer-Moore-Horspool algorithm [5] (which is simplified version of Boyer-Moore algorithm), Aho-Corasick algorithm [1], Commentz-Walter algorithm [2], Fan-Su algorithm [11] (which is a combination of Aho-Corasick and Boyer-Moore algorithms), Wu-Manber algorithm [3], and Set Backward Oracle Matching (SBOM) algorithm [12], [13] (which is the extension of the Backward Oracle Matching (BOM) algorithm [13], [14]). This paper mainly presents the analysis of mostly used algorithms for multiple string pattern matching problems: the Aho-Corasick algorithm, the Commentz-Walter algorithm, and the Wu-Manber algorithm. Experimental results of these algorithms are included for the analysis and discussion about multiple pattern matching problems. This paper also discusses the main theoretical results for each of the algorithm. The performance of each algorithm is shown against the length of pattern and the number of pattern in a pattern set. An extensive list of references is also presented at the end of this paper.

This paper structures as follows: Section II briefly describes the multiple pattern matching algorithms specifically Aho-Corasick, Commentz-Walter, and the Wu-Manber algorithm, Section III outline the experiment methodology, Section IV presents the experimental results on the multiple pattern matching algorithms, Section V presents the analysis and discussion on pattern matching problem based on the experimental results and existing works, and Section VI gives the conclusion of this paper.

II. MULTIPLE PATTERN MATCHING ALGORITHMS

This section presents the overview of most popular solutions for the multi-pattern matching problem: Aho-Corasick algorithm [1], Commentz-Walter algorithm [2], and Wu-Manber Algorithm [3].

A. Aho-Corasick algorithm

Aho-Corasick algorithm, a variant of the Knuth-Morris-Pratt algorithm [7], was the first algorithm to solve the multiple string pattern matching problems in linear time based on automata approach. This algorithm serves as the basis for the UNIX tool *fgrep*.

Aho-Corasick algorithm consists of two parts. In the first part they constructed a finite state pattern matching machine from the set of keywords and in the second part, the text string as input is applied to the pattern matching machine. The machine signals whenever it has found a match for a keyword (pattern). The pattern matching machine consists of a set of states and each state is represented by a number. The behavior of the pattern matching machine is dictated by three functions: a *goto* function *g*, a *failure* function *f*, and an *output* function *output*. Figure 1 shows a sample pattern matching machine for the set of patterns, p = {he, she, his, hers} [1].

The *goto* function *g* maps a pair consisting of a state and an input symbol into a state or the message *fail*. The *failure* function *f* maps a state into a state. The *failure* function is consulted whenever the *goto* function reports *fail*. The *output* function of certain states indicates that a set of keywords has been found.

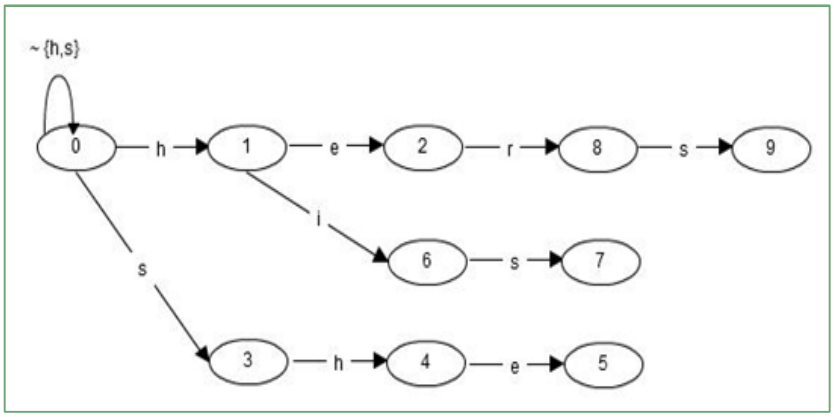


FIGURE 1: A SAMPLE PATTERN MATCHING MACHINE.

The construction of Aho-Corasick automaton machine takes running time linear in the sum of the lengths of all patterns/ keywords. This involves building the pattern tree (keyword tree) for the set of pattern and then converting the tree to an automaton (pattern matching machine) by defining the functions *g (goto* function*), f* (*failure* function), and *output* function for labeling states with the keyword(s) matched. The space or memory requirements of the Aho-Corasick algorithm can be quite large depending on the pattern set and also the length of each pattern in a pattern set. The matching process is simply stepping through the input characters one at a time and checks if there is any matching. Each step in pattern matching machine happens in constant time. So, the Aho-Corasick matcher always operates in O(n) running time.

B. Commentz-Walter algorithm

Commentz-Walter presented an algorithm for the multi-pattern matching problem that combines the Boyer-Moore technique with the Aho-Corasick algorithm. Commentz-Walter combines the filtering functions of the single pattern matching Boyer-Moore algorithm and a suffix automaton to search for the occurrence of multiple patterns in an input string. The tree used by Commentz-Walter is similar to that of Aho-Corasick' pattern matching machine but is created from the reversed patterns. A sample reversed pattern tree is shown in figure 2.

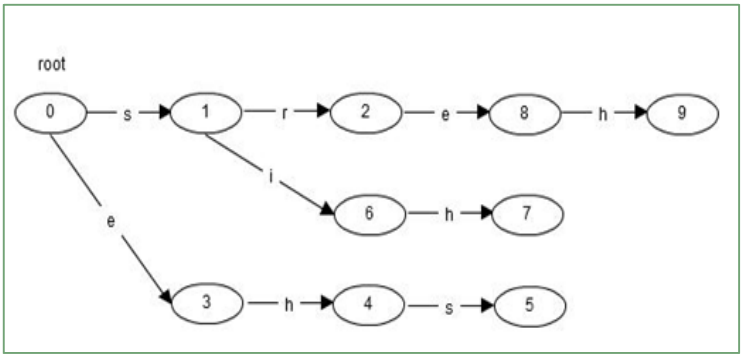


FIGURE 2: A SAMPLE REVERSED PATTERN TREE

The original paper presented two versions of the algorithm. In all version of the algorithm a common program skeleton is used with different shift function. The Commentz-Walter algorithm is also consists of two phase: pre-computing phase and matching phase. The pre-computation phase of algorithm is responsible for creating a pattern tree by using the reversed pattern (see figure 2). The matching phase of the Commentz-Walter algorithm is combination of two ideas. One is from the ideas of Aho-Corasick' finite automata technique (in pattern tree) and another one is from the Boyer-Moore shifting techniques (in right-to-left matching). In this algorithm a match is conducted by scanning backwards through the input string. At the point of mismatch some number of characters about the input string is known (that is, the number of characters that ware matched before the mismatch) and this information then is used as an index. The index is used in a pre-computed table to determine a distance which is later helps to shifting before commencing the next match attempt.

C. Wu-Manber Algorithm

Wu-Manber algorithm is a simple variant of the Boyer-Moore algorithm that uses the bad-character shift for multiple pattern matching. They actually come to the idea of this algorithm after making a UNIX based tool *agrep* [10] which was for searching many patterns in files. To improve the performance, they come through a unique idea, that is, their algorithms looks at block of text instead of single character. So, they consider both pattern and text as blocks of size B instead of single characters. As recommended in their paper [3], in practice B could be equal to 2 for a small pattern set size or to 3 otherwise.

The operational process of the WM algorithm includes two phases. In first phase which is called preprocessing phase, three tables are constructed by the patterns, the SHIFT, the PREFIX and the HASH tables. The SHIFT table stores the shift values of the block characters that determine the safe shifting of characters during the searching phase. If a block of B characters does not occur in any pattern, then the shift value for that block assigns to the maximum value, which is $m - B + 1$. The HASH table stores hashed values (h) of B characters suffix of each pattern while the PREFIX stores hashed values (h′) of B' characters prefix of a list of patterns that they have the same suffix.

The second part of the algorithm is the searching phase. During this phase, the algorithm is searching for the occurrences of all patterns in the input text with the assistant of the three tables that have been created by the previous state. Firstly, a hash value (h) for the block of B characters is calculated into the current search window and the shift value for that is checked (SHIFT[h]). If the shift value is greater than zero, then the current search window is shifted by SHIFT[h] positions, or else there is a potential matching and the tables HASH and PREFIX should be considered in order to validate the matching.

The first thing is to compute the minimum length of a pattern, call it *m*, and consider only the first m characters of each pattern. This requirement is crucial to the efficiency of the algorithm. If one of the patterns is very short, say of length 2, then it is not possible to shift by more than 2, so having short patterns inherently makes this approach less efficient. The expected running time

complexity of the main matching phase/ searching phase is shown by Wu-Manber to be less than linear in *n* (the length of the input text) [3].

III. Experiment Methodology

To evaluate the performance of the multiple pattern matching algorithms, the practical running time is used as a measure. Practical running time is the total time an algorithm needs to find all occurrences of a pattern set in an input text including any preprocessing time. In the experiment, English text is considered as an input text where the pattern set is chosen randomly for the searching/ matching process. The input text used in this experiment is consists of 477,048 characters excluding spaces and it contains 99,449 words in total. For the implementation of these algorithms JAVA is used as a programming language. All the experiments are run on a computer which has a 2.20 GHz Intel Core 2 Due processor, 4 GB RAM, and 64-bit Windows 7 Operating System.

IV. Experimental Results

In this section of the paper the experimental results of the algorithms are presented. The performance of the algorithms is shown by measuring the running time against the number of pattern and length of pattern (pattern size) in a pattern set. The table 1 and figure 3 shows the running time of Aho-Corasick algorithm with different number of patterns (10 to 500 patterns) but the minimum length of pattern is 2. In Aho-Corasick algorithm, if the number of pattern is increases, the running time is also increase.

TABLE 1: RUNNING TIME OF AHO-CORASICK ALGORITHM DEPENDING ON NUMBER OF PATTERNS

Number of pattern	Minimum length of pattern	Running time (ms)
10	2	32
50	2	39
100	2	48
150	2	48
200	2	51
250	2	60
300	2	63
350	2	63
400	2	64
450	2	69
500	2	73

FIGURE 3: RUNNING TIME OF AHO-CORASICK ALGORITHM DEPENDING ON NUMBER OF PATTERNS

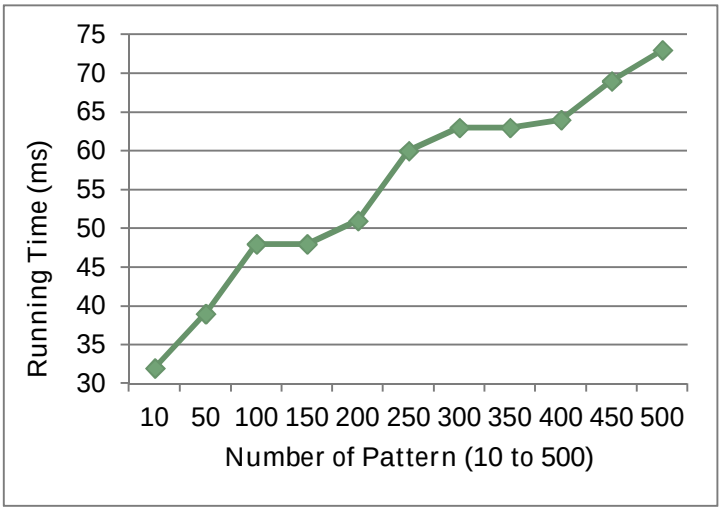


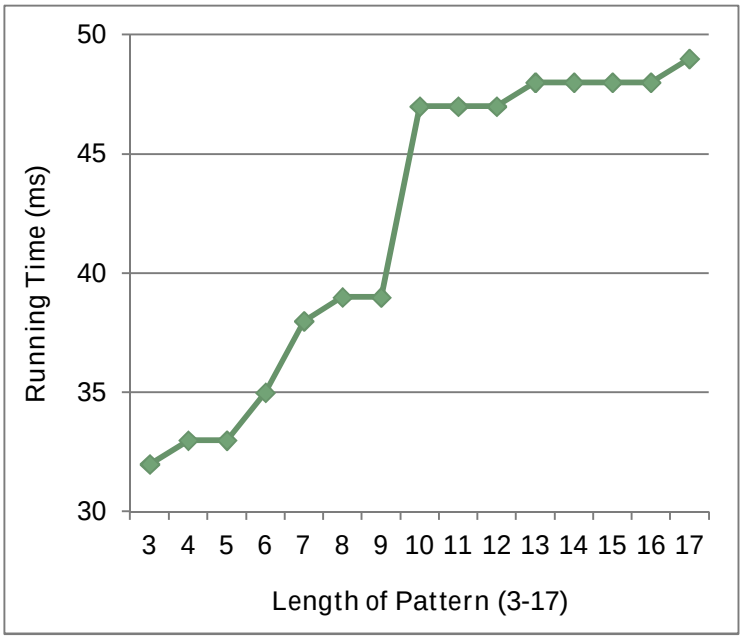
Table 2 and figure 4 also shows the running time of Aho-Corasick algorithm but in this time with different length of pattern (length 3 to 17) and fixed number of pattern (in this case 10). In Aho-Corasick algorithm, if the length of pattern is increases, the running time is also increase.

But it has better performance with the larger length of pattern. As we can see that running time in Aho-Corasick algorithm does not change too much with the larger length of pattern.

TABLE 2: RUNNING TIME OF AHO-CORASICK ALGORITHM DEPENDING ON LENGTH OF PATTERNS

Number of pattern	Length of pattern	Running time (ms)
10	3	32
10	4	33
10	5	33
10	6	35
10	7	39
10	8	39
10	9	39
10	10	47
10	11	47
10	12	47
10	13	48
10	14	48
10	15	48
10	16	48
10	17	49

FIGURE 4: RUNNING TIME OF AHO-CORASICK ALGORITHM DEPENDING ON LENGTH OF PATTERNS



The Table 3 and Figure 5 shows the running time of Commentz-Walter algorithms with different number of patterns (10 to 500 patterns) but the minimum length of pattern is 2. Like in the Aho-Corasick algorithm, the running time of Commentz-Walter algorithm is also increasing with the number of pattern increases.

TABLE 3: RUNNING TIME OF COMMENTZ-WALTER ALGORITHM DEPENDING ON NUMBER OF PATTERNS

Number of pattern	Minimum length of pattern	Running time (ms)
10	2	30
50	2	38
100	2	46
150	2	48
200	2	49
250	2	54
300	2	58
350	2	60
400	2	62
450	2	67
500	2	70

FIGURE 5: RUNNING TIME OF COMMENTZ-WALTER ALGORITHM DEPENDING ON NUMBER OF PATTERNS

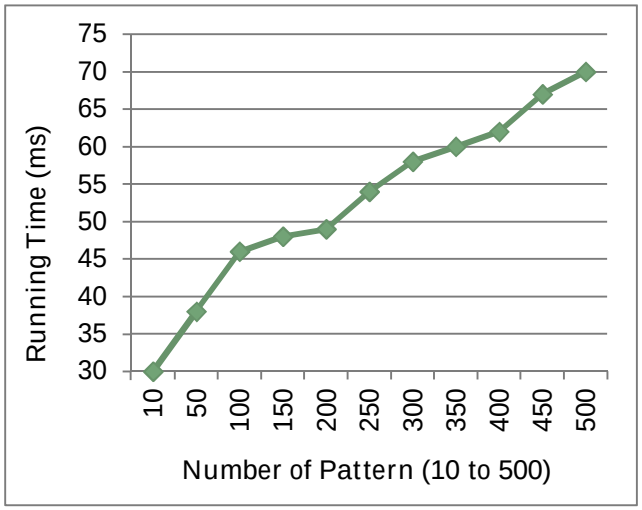


Table 4 and figure 6 shows the running time of Commentz-Walter algorithm with different length of pattern (length 3 to 17) but fixed number of pattern (in this case 10). In Commentz-Walter algorithm, if the length of pattern is increases, the running time is also increase. But the running time of Commentz-Walter algorithms improved approximately linearly with increasing length of the shortest pattern in the pattern set.

TABLE 4: RUNNING TIME OF COMMENTZ-WALTER ALGORITHM DEPENDING ON LENGTH OF PATTERNS

Number of pattern	Length of pattern	Running time (ms)
10	3	31
10	4	33
10	5	33
10	6	34
10	7	36
10	8	37
10	9	39
10	10	41
10	11	41
10	12	43
10	13	44
10	14	44
10	15	46
10	16	47
10	17	47

FIGURE 6: RUNNING TIME OF COMMENTZ-WALTER ALGORITHM DEPENDING ON LENGTH OF PATTERNS

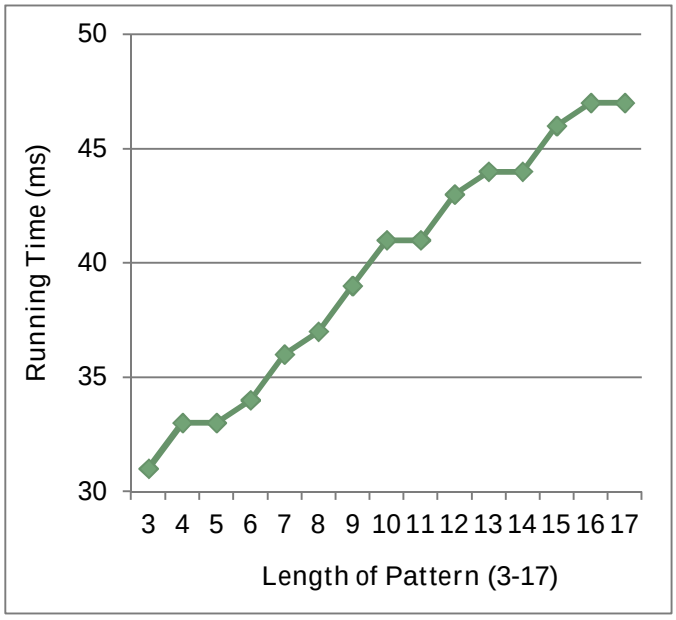
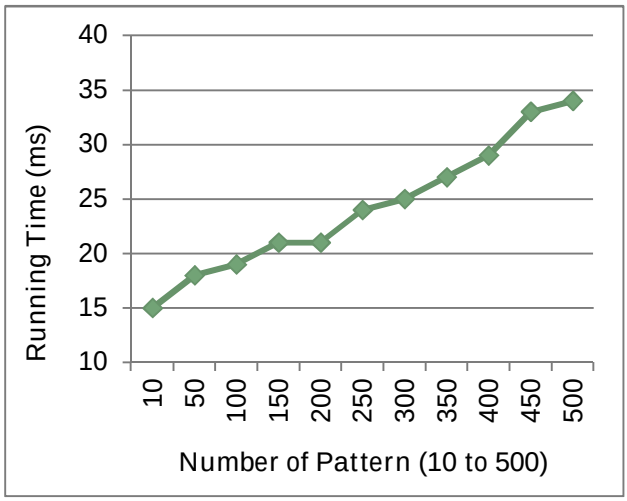


TABLE 5: RUNNING TIME OF WU-MANBER ALGORITHM DEPENDING ON NUMBER OF PATTERNS

Number of pattern	Minimum length of pattern	Running time (ms)
10	3	15
50	3	18
100	3	19
150	3	21
200	3	21
250	3	24
300	3	25
350	3	27
400	3	29
450	3	33
500	3	34

FIGURE 7: RUNNING TIME OF WU-MANBER ALGORITHM DEPENDING ON NUMBER OF PATTERNS



The table 5 and figure 7 shows the running time of Wu-Manber algorithm with different number of patterns (10 to 500 patterns) but the minimum length of pattern is 3. In Wu-Manber algorithm, if the number of pattern is increases, the running time is also increase. But the performance of this algorithm is better than the Aho-Corasick algorithm because, Wu-Manber algorithm use block of character shifting while searching for a set of pattern in a given text. Furthermore, as Aho-Corasick and Commentz-Walter algorithms are based on automata approach, and hence, they consume more memory than Wu-Manber algorithm.

V. ANALYSIS & DISCUSSION

A linear time algorithm for multiple patterns matching problem proposed by Aho and Corasick [1] is optimal in worst case but Boyer and Moore [4] demonstrated an algorithm where they showed that it is possible to skip a large portion of the text while searching for certain patterns. Eventually, this is working faster than linear algorithm in the average case. The Commentz-Walter algorithm [2] combines the idea of Boyer and Moore technique with Aho-Corasick algorithm for multiple patterns matching problem which is substantially faster than the Aho-Corasick algorithm in practice. It uses the idea of Boyer Moore technique to skip a large portion of the text while searching and as a result leading to faster than linear time algorithms in the average case. There has another algorithm proposed by Baeza-Yates [6] which also combines the idea of Boyer-Moore-Horspool algorithm [5] (which is a slight variation of the classical Boyer-Moore algorithm) with the Aho-Corasick algorithm. Whereas, Wu-Manber algorithm is the most efficient algorithm under some scenarios such as, long random patterns, low matching rate, and low memory requirement. However, Aho-Corasick performance does not suffer great decline when comparing with others as it is a linear time searching algorithm in worst case.

Independent from the pattern set size, searching time complexity for Aho-Corasick algorithm is O(n) but when pattern set size increase, the memory consumption increased drastically and also the time consumption increased. The performance of Commentz-Walter algorithms declined with increasing number of pattern in a pattern set (pattern set size). But the performance of Commentz-Walter algorithms improved approximately linearly with increasing length of the shortest keyword/ pattern in the pattern set. Moreover, in [8, pp. 281], [9], A.V. Aho states that,

> *"In practice, with small numbers of keywords, the Boyer-Moore aspects of the Commentz-Walter algorithm can make it faster than the Aho-Corasick algorithm, but with larger numbers of keywords the Aho-Corasick algorithm has a slight edge."*

This paper also found the above statement correct with the presented experimental result and analysis. But the Aho-Corasick and Commentz-Walter algorithms consume lots of memory because in the preprocessing stage both these algorithms use the automata data structure whereas Wu-Manber algorithm consume much less memory than these two algorithms.

VI. CONCLUSION

The algorithms that offer the solution for the multi-pattern matching problem, among them Aho-Corasick, Commentz-Walter, and Wu-Manber algorithms are very popular solutions. A comprehensive analysis and discussion of these selected algorithms as a state-of-the-art along with some experimental results is covered in this article. This paper has presented the analysis and discussion of the algorithms in order to understand the multiple pattern matching problem in an easier manner.

The Aho-Corasick algorithm considers as a classic solution and core element of many other pattern matching algorithms. Also it has been used in many other applications. As it is a linear time algorithm, it is considered very useful solution for multiple pattern matching problems. On the other side, Commentz-Walter algorithm seems to be the first sub-linear running time algorithm for multiple-pattern matching problems in average case by using a sifting technique where a large portion of the text is skipped while searching. The Wu-Manber algorithm has excellent average case performance because of the successful use of shifting operation as a block of characters. However, Wu-Manber algorithm has minimum length problem. If the minimum length of pattern is less, then it is not as efficient as it should be. So it would be an optimization area for Wu-Manber algorithm. For the Aho-Corasick and Commentz-Walter algorithms, memory compression would be a good optimizing area as they consume lots of memory. Also subset division of pattern set could be another way for the optimization of Aho-Corasick algorithm.

This paper mainly covers the analysis and discussion among Aho-Corasick, Commentz-Walter, and Wu-Manber algorithms for multiple string pattern matching problems. A comprehensive study on all the existing algorithms of multiple pattern matching problems would be a very demanding material in the research area of multiple pattern matching problems.

REFERENCES

[1] Aho, Alfred V. & Corasick, Margaret J. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18, 333-340.

[2] Commentz-Walter, Beate. (1979). A string matching algorithm fast on the average. *Automata Languages and Programming*, 6, 118-132.

[3] Wu, Sun & Manber, Udi. (1994). A fast algorithm for multi-pattern searching. *Technical Report TR-94-17, University of Arizona*.

[4] Boyer, R. S. & Moore, J. S. (1977). A fast string searching algorithm. *Communications of the ACM*, 20, 762-772.

[5] Horspool, N. (1980). Practical fast searching in strings. *Software: Practice and Experience*, 10, 501-506.

[6] Baeza-Yates, R. A. (1989). 'Improved string searching, *Software - Practice and Experience, 19,* 257-271.

[7] Knuth, Donald E., Morris, James H., Pratt, Vaughan R. (1974). Fast pattern matching in strings. *Technical Report CS440, Computer Science Department, Stanford University, Stanford, California*.

[8] Aho, A. V. (1990). Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science* In J. van Leeuwen (Ed.), (pp. 257-300). North-Holland, Amsterdam.

[9] Waston, B. W. (1994). The performance of single-keyword and multiple keyword pattern matching algorithms.

[10] Wu, S. & Manber, U. (1992). Agrep – a fast approximate pattern-matching tool. In *Proceedings USENIX Winter 1992 Technical Conference*, (pp. 153–162), San Francisco, CA.

[11] Fan, J.-J. & Su, K.-Y. (1993). An efficient algorithm for matching multiple patterns. *IEEE Transactions on Knowledge and Data Engineering*, 5(2), 339–351.

[12] Navarro, G. & Raffinot, M. (2002). Flexible Pattern Matching in Strings, Practical Online Search Algorithms for Texts and Biological Sequences. *Cambridge University Press*, Cambridge, UK.

[13] Allauzen, C. & Raffinot, M. (1999). Oracle des facteurs d'un ensemble de mots. *Technical Report IGM 99-11*, Institut Gaspard Monge, Universit´e de Marne-la-Vall´ee,France.

[14] Allauzen, C., Crochemore, M., Raffinot, M. (1999). Factor oracle: A new structure for pattern matching. In J. Pavelka, G. Tel, and M. Bartosek, (Ed.), Theory and Practice of Informatics (Brno, 1999), volume 1725 of Lecture Notes in Computer Science, pp. 291–306. Springer-Verlag. In Proceedings *of the 26th Seminar on Current Trends in Theory and Practice of Informatics*, Milovy, Czech Republic.

AUTHOR'S BIOGRAPHY

Akinul Islam Jony born in Dhaka, Bangladesh. He received his M.Sc. degree in Informatics at Technical University of Munich (TUM) in Germany. Previously he completed his B.Sc. degree in Computer Science at American International University – Bangladesh (AIUB) and Master degree in Information Technology at University of Dhaka (DU) in Bangladesh. His current research interest includes algorithms, service-oriented computing, distributed middleware system, and ubiquitous computing.