Barisal University Journal Part 1, 4(2):399-411 (2017) ISSN 2411-247X

A FAST ON-LINE HYBRID MATCHING ALGORITHM FOR EXACT STRING MATCHING

Tania Islam*, Kamrul Hasan Talukder and Rahat Hossain Faisal

Computer Science and Engineering Discipline, Khulna University, Khulna, Bangladesh
Department of Computer Science and Engineering, University of Barisal, Barisal 8200, Bangladesh

Abstract

String matching technique is the procedure of defining one or more existences of a pattern string inside a larger string or text. This process is applied in the numerous fields of computer science for simplifying the pattern searching operations. Several string searching or string identification algorithms have been identified and continuous studies are being conducted for further improvement. The aim of our proposed algorithm is to develop a fast hybrid string matching algorithm which can reduce the amount of character comparisons than that of ABSBMH algorithm. The combination of some features of the Quick Search Algorithm, SSABS Algorithm and ABSBMH algorithms has been used to determine an enriched process of string identification problem, which will enhance speed and reduce cost. The proposed hybrid algorithm which is called Fast On-line Hybrid Matching Algorithm (FOHM) algorithm has been tested by using different types of dataset.

***Keywords*:** On-line string matching, DNA sequence, Protein sequence, Pattern; ABSBMH algorithm, QS algorithm.

Introduction

In theoretical computer science, one of the most studied research problem is pattern matching which play an important role in different fields of science and information is processing. A few of its imperative applications are spell checkers, spam filters, intrusion detection system, search engines, detecting the copy of text from others, analyzing biological data and information saving systems, firewall interception and searching nucleotide or amino acid, identification of pattern in genome, and protein sequence data (Bhukya and Somayajulu, 2011). Generally pattern matching algorithm mainly depends on the shifting value of the pattern according to the given text. The greater shift value

*Corresponding author's e-mail: tania.bd.09@gmail.com

will make the algorithm more efficient because it can minimize the number of comparisons (Senapati et al., 2012).

String searching also known as string matching is mainly aimed to find an occurrence of pattern which length is m, in a given string of length is n, where n≥m (Galil, 1997).

Suppose a text T= {t_1.t_2.t_3……t_n} which contains n number of characters and a pattern P= {p_1. p_2.p_3…….p_m} of length m. The main target of this algorithm is to find an integer s, which is referring as valid shift value where 0≤s≤n-m and T [s+1…….s+m] = P[1…m]. In other word it can describe as to find the pattern P in text T i.e., where P is consider as the substring of T. The element of T and P are consist of finite number of alphabet such as {0, 1} or {A, B…..Z, a, b…….z} (Cormen et al., 1990).

In all pattern matching algorithm a part of given text is selected for matching purpose, this part is called window which is exact size of the given pattern. This pattern mainly aligns with window of text from left side to right side and tries to match the text window character with the pattern character according to any algorithm. This matching process is differing according to any string searching algorithm. In order to know the existence of any pattern in a given text an well and strong algorithm is needed (Abdeen, 2011).

A good numbers of algorithms have already been developed for string searching. Each algorithm uses different technique and strategy for matching. Most of the algorithms use pre-processing phase. Some algorithms are pre-processed the pattern (Sheik et al., 2004). The algorithm which pre-processed the pattern is called on-line string matching algorithm. And whereas some are pre-processed the text (Bhukya and Somayajulu, 2011). The algorithm which pre-processed the text is called off-line string matching algorithm. And then it starts searching phase. Also some algorithms pre-processed both pattern and text before starting searching stage (Bhukya and Somayajulu, 2011). There are some algorithms which do not require any kind of pre-processing phase (Gonzalo, 2001).

In our proposed algorithm, we pre-processed the given pattern. So our algorithm is called on-line string matching algorithm. The proposed name of our algorithm is FOHM algorithm. We pre-process our pattern according to the Quick Search algorithm. In terms of number character comparison and number of attempt, our algorithm gives better result than ABSBMH algorithm.

The rest of the paper is prepared as follows. Section 2 which is gives the analysis of numerous well-organized algorithms in exercise. Section 3 defines the planned algorithm

in part. In section 4, the investigational outcomes with judgment between planned algorithm and other associated algorithm are specified. The section 5 is the description of conclusion.

Related Work

The basic string matching algorithm which considered as a fundamental algorithm is Brute-Force algorithm.

Start-to-End Algorithm (Abdeen, 2011) algorithm is the modification of the BF algorithm. According to this algorithm, here is no need to pre-process the pattern and text. It starts comparing from the beginning of both text and pattern. It compares the starting character of text window with the starting character of pattern. In this condition when they match then it tries to compare the character at the end of the text window and at end of pattern. Remaining characters are matching from right to left direction basis.

Start-End-Mid Algorithm (Abdeen, 2011) is also the modification of the BF algorithm. This algorithm also does not require any kind of pre-processing stage. In this algorithm it is first match with first character of text window with the first character of pattern, then last character of text with the last character of pattern. At last, finally middle character of text with the middle character of pattern.

Another basic algorithm is Boyer Moore algorithm (Boyer and Moore, 1977). In this algorithm the matching is started from the end character of the text window with the end character of given pattern. This algorithm uses good suffix heuristic and bad character heuristic to control the shift of the pattern when any mismatch or match occur with the text and pattern. The Boyer Moore algorithm is considered to be an efficient algorithm for pattern searching. It has the property that the longer the pattern is; the faster it performs the comparison. However the algorithm suffers from the phenomenon that it tends to work incompetently on small alphabets like DNA.

Horspool algorithm (Horspool, 1980) is the simplification of the Boyer-Moore algorithm (Boyer and Moore, 1977). This algorithm uses the same searching procedure like the BM algorithm but it only uses the bad character heuristic because good suffix heuristics is complicated and it is very difficult to implement. Horspool suggested that using only the bad character heuristic would also give the good result for longer pattern.

Raita algorithm (Raita, 1992) is uses Boyer-Moore bad character table in pre-processing phase. In its searching phase, this algorithm starts searching from right character in the text window with the right character of pattern, if it matches then check from left

character in text window with the left character of pattern, if it again matches then starts matching from the middle of both text and pattern and for the remaining character it starts matching as right to left order.

Quick search algorithm (Sunday, 1990) is also the simplification of BM algorithm but it only uses the quick search bad character table (qsBc) for shifting the window of text and pattern. For calculating the value of qsBc table they use the position of the character from the right end. If any character does not occur in pattern, then it values calculated as m+1 and in searching phase, matching starts from left to right.

QSS algorithm (Naser et al., 2012) is a hybrid algorithm with the combination of quick search algorithm and Skip search algorithm. In pre-processing phase it uses quick search bad character table and skip search skip search bucket. And in searching phase it shifted the text window according one of these two processes.

The SSABS algorithm (Sheik, 2004) is the mixture of quick search algorithm and Raita string matching algorithms. The authors use quick search bad character table for shifting the window. Like Raita algorithm, SSABS algorithm starts comparing from the end character of the text window with the end character of pattern, if it makes a match then it compares the starting character of the text window with starting character of pattern. Remaining characters are matching at right to left order basis.

ABSBMH Algorithm (Mahmood et al., 2017) which is a hybrid algorithm with the combination of QS algorithm and SSABS algorithm. In this algorithm they use QS bad character table in pre-processing stage and for shifting the window in searching stage they use modified horspool algorithm. They compare with end two characters instead of one that is in SSABS algorithm. Then compare starting character of text window with starting character of pattern. After that, continue this process from second last to first character. This algorithm shows worst result for DNA sequence.

Proposed Methodology

After listing most of the well-known string matching algorithms, this section discusses the proposed hybrid solution that combines the quick Search algorithm, SSABS algorithm and the ABSBMH algorithms because all of this algorithms are used quick search bad character table which we used in our proposed algorithm.

Like the others current algorithms, the effectiveness of the planned hybrid algorithm lies in two segments which are the pre-processing segment and the searching segment. The

searching phase is depend on the pre-processing phase of the algorithm which is pre-processed the pattern in pre-processing phase in order to reduce the number of character comparisons and number of attempt.

Pre-processing Phase

Pre-processing phase is mainly used to reduce the searching time complexity. Here we use quick search bad character table to pre-process the pattern. This phase is used to determine the shift value in case of occurring any kind of mismatch at either the left or the right side of the text with regard to the pattern. In our proposed hybrid string matching algorithm, we use quick search bad character table (qsBc) in pre-processing phase to determine the shifting value of the text window. The QS algorithm bounces the supreme move value $(m + 1)$ once the character following to the right most character on the text window does not seen at the pattern. On the other hand, as soon as the character next to the rightmost character on the text window matches to the existing character of the pattern, the quick search algorithm provides the lowest shift value. This work is done according to the equation 1.

$$\mathrm{qsBc(x)} = \begin{cases} (\mathrm{i}: 0 \le \mathrm{i} < \mathrm{m} \text{ and } \mathrm{P}[\mathrm{m} - \mathrm{i}] = \mathrm{x}) \text{ if x occurs in P} \\ \mathrm{m} + 1 \text{ otherwise} \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

In quick search algorithm, only one character immediately to the right of the given string window is considered as the shift value.

Searching Phase

The proposed algorithm is the modification of the searching phase of ABSBMH algorithm. In our algorithm the searching phase starts from right end of the both text and pattern window. So that, any mismatch character is occur in end of the both text and pattern window, both windows will jump to the next shift position according to the shifting value of quick search bad character table. The main steps of proposed hybrid algorithm are:

After aligning the window of the text with the pattern, comparisons starts from right side of the window (Sheik et al., 2004). If any mismatch occurs during the comparison of last character of the text window with the last character of pattern window, it goes to step 2, otherwise it again matches with the second last position character of the text with the 2nd last character of the pattern. If it matches then it compares 3rd on and if it makes a matching then it matches first character of the text with the first position character of

pattern and continues this process for next two characters, if they all are matched then start matching from remaining character in the direction of right to left.

In this step when a mismatch occurs in text character with pattern, then the window will shift to the right side of the text according to the value of bad character table. When a complete match is found then it is shifted $(m + 1)$ position.

This procedure is repeated until the window is placed beyond $(n - m + 1)$, that is the last character of the pattern placed beyond the last character of the text.

When any match found then it repeat the procedure until the end of the text to find the repetitions of pattern into the given text.

Working Example

In this section we present an example to clarify our proposed FOHSM algorithm.

Given text, $T = \text{ATCTGAGTTCT}$ Where $n = 11$

Pattern, $P = \text{AGTT}$ Where $m = 4$

Pre-processing Phase

The shifting value for window is calculated by using equation 1. Quick search bad character table value for pattern is given below:

Table 1. Quick search bad character table

	A	**G**	**C**	**T**
qsBc	4	3	5	1

Searching Phase

The searching process of pattern $\boldsymbol{P}$ in given text $\boldsymbol{T}$ is illustrated through the working example is given in Fig. 2. In first attempt (Fig. 2a), we align text $\boldsymbol{T}$ with pattern $\boldsymbol{P}$ in the left side of the text. A comparison starts from right side of the text ($\boldsymbol{T}$) with the right side of the pattern ($\boldsymbol{T}$). It makes a match, then it compares 2nd last character and it's a mismatch, then the window shifts to the right side according to the value of $\boldsymbol{G}$. In 2nd attempt (Fig. 2b), the last character of text ($\boldsymbol{G}$) is making a mismatch with the last character of pattern ($\boldsymbol{T}$). Then it shifts the window. Again in 3rd attempt (Fig. 2c), 2nd last character making a mismatch. At last, in 4th (Fig. 2d) attempt, it makes a match in position of 5 in the text.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1 0**
A	T	C	T	G	A	G	T	T	C	T
A	G	T	T							

2(a)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1 0**
A	T	C	T	G	A	G	T	T	C	T
			A	G	T	T				

2(b)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1 0**
A	T	C	T	G	A	G	T	T	C	T
				A	G	T	T			

2(c)

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1 0**
A	T	C	T	G	A	G	T	T	C	T
				A	G	T	T			
				4	3	2	1			

2(d)

Experimental Result

To evaluate the performance of our proposed algorithm, we compare our algorithm with ABSBMH algorithm that has already been tested with different algorithms.

We use three kinds of data set that is DNA sequence which contains only four characters, Protein sequence which contains 20 character and English text which contains 100 characters over small and capital letters with special symbol and numbers.

We tested our algorithm using C# programming language and using laptop computer with processor Intel (R) core (TM) i5, Operating System name Microsoft Windows 7 professional and RAM 4.00GB.

Our algorithm uses input dataset of 100MB (Mahmood et al., 2017), and different size of pattern that is $12, 24, 36, 48, 60, 72, 84,$ and 96. We randomly choose our pattern from the data set and tested over 10 times for each pattern size.

The efficiency of any algorithm is depending on the number of character comparisons and number of attempt. Our algorithm is able to reduce the number of character comparisons and number of attempt than ABSBMH algorithm.

Table 2 and Table 5 show the number of character comparisons with the proposed algorithm and ABSBMH algorithm using English type dataset. Our proposed algorithm is able to minimize the number of comparisons and also number of attempt than that of ABSBMH algorithm.

By using DNA sequence data types we tested our algorithm and ABSBMH algorithm and our algorithm gives better output in terms of number of character comparison and number of attempt which is described in Table 3 & 6.

We are using three types of dataset. Then for another protein sequence our algorithm can reduce the number of character comparisons and number of attempt than ABSBMH algorithm that is explain in Table 4 & 7.

It noticed that the number of attempt is almost same or sometimes our algorithm can work better with the comparison of ABSBMH algorithm. Our algorithm can make better improvement in terms of number of character comparisons.

Table 2. Number of character comparisons using English type dataset.

Pattern Length	**ABSBMH**	FOH**M**
12	11023806	11023800
24	9046138	9046100
36	6833164	6833100
48	7248270	7246813
60	6467220	6467200
72	5318487	5312962
84	5512485	5505539
96	4984744	4983645

Table 3. Number of character comparisons using DNA sequence.

Pattern Length	ABSBMH	FOHM
12	11023806	11023800
24	9046138	9046100
36	6833164	6833100
48	7248270	7246813
60	6467220	6467200
72	5318487	5312962
84	5512485	5505539
96	4984744	4983645

Table 4. Number of character comparisons using protein sequence.

Pattern Length	ABSBMH	FOHM
12	13637923	13637119
24	9753190	9750735
36	8160677	8160500
48	7509247	7509200
60	6213911	6213739
72	6166346	6165316
84	4921907	4921807
96	5745196	5745096

Table 5. The Number of attempt using English type dataset.

Pattern Length	ABSBMH	FOHM
12	10646545	10646543
24	8061707	8061700
36	6833164	6820907
48	6378687	6378680
60	5863962	5863960
72	4853505	4853504
84	4785077	4785076
96	4654523	4654522

Table 6. The Number of attempt using DNA Sequence.

Pattern Length	ABSBMH	FOHM
12	18757336	18757336
24	42289959	42289958
36	23437604	2343604
48	16739263	16739262
60	38895762	38895760
72	20831745	20831745
84	24919045	24919044
96	26253709	26253709

Table 7. The Number of Attempt using Protein Sequence

Pattern Length	ABSBMH	FOHM
12	11955258	11955257
24	8051802	8051802
36	6741679	6741678
48	6617901	6617900
60	5613590	5613590
72	5686733	5686733
84	4776561	4776560
96	46822223	4682221

Conclusion

In this paper we present an efficient algorithm with the combination of Quick Search algorithm, SSABS algorithm and ABSBMH algorithm. We are depending only Quick Search algorithm in the pre-process phase and we modify the search stage of ABSBMH algorithm. We tested our algorithm using three kinds of data sets that are DNA sequence, Protein sequence and English alphabet. Experimental result shows that our algorithm works better than ABSBMH algorithm. In this algorithm we work with only one sequence or with one long sentence but in future we will try to update our algorithm for working with multiple sequences and multiple sentences at a time.

Acknowledgement

We would like to acknowledge the ICT Division Ministry of Posts, Telecommunications and Information Technology, Government of the People's Republic of Bangladesh and also Computer Science and Engineering Discipline, Khulna University for supporting this work.

References

Abdeen R. A, 2011. Start-to-End algorithm for string searching, International Journal of Computer Science and Network Security. **11**:2-12.

Abdeen R. A. 2011. An algorithm for string searching based on brute-force algorithm, International Journal of Computer Science and Network Security, vol. 11, no.7.

Abdeen R. A. 2011. Start-to-End Algorithm for string searching. International Journal of Computer Science and Network Security. **11:**2-10.

Bhukya R. and Somayajulu. D. 2011. Index based multiple pattern matching algorithm using DNA sequence and pattern count. International Journal of Information Technology and Knowledge Management. **4**:431-441.

Bhukya R. and Somayajulu. D. 2011. An Index based sequential multiple pattern matching algorithm using least count. International Conference on Life Science and Technology, IACSIT Press, Singapore, **3**:10-15.

Bhukya R. and D. Somayajulu. 2011. Exact multiple pattern matching algorithm using DNA sequence and pattern pair. International Journal of Computer Applications, **17**:20-25.

Boyer R. S. and J. S. Moore. 1977. A fast string-searching algorithm, Communication of ACM. **4:**762-772

Charras C. and T. Lecroq. 2012. Handbook of Exact String Matching Algorithms, online, October.

Cormen T. H., C. E. Leiserson and R. L. Rivest. 1990. Introduction to Algorithms, MIT Press, First Edition, pp. 853-885.

Galil Z. 1997. Pattern Matching Algorithms, Oxford University Press.

Gonzalo N.2001. A guided tour to approximate string matching," ACM Computing Surveys. **33**:31-88.

Hasan A. A. and N. A. A. Rashid. 2012. Hash - Boyer-Moore - Horspool string matching algorithm for intrusion detection system. International Conference on Computer Networks and Communication Systems, IPCSIT. **35**:12-16.

Horspool R.1980. Practical fast searching in strings. Software Practice and Experience, **10**:501-506.

Mahmood S. S., A. Dabbagh and N. H. Barnouti. 2017. A new efficient hybrid string matching algorithm to solve the exact string matching problem. British Journal of Mathematics and Computer Science. **8**:1-14.

Michailidis P. D. and K. G. Margaritis. 2007. On-line string matching algorithms: Survey and experimental results. International Journal of Computer Mathematics. **15**:740-741.

Naser M. A. S., Rashid N. A., and M. F. Aboalmaaly. 2012. A quick-skip search hybrid algorithm for the exact string matching problem. International Journal of Computer Theory and Engineering. **4**:2-10.

Raita T. 1992. Tuning the Boyer-Moore-Horspol string matching algorithm. Software-Practice & Experience. **22**:879-884.

Senapati K. K., D. R. D. Adhikary and G. Sahoo. 2012. An application of pattern matching for motif identification. International Journal of Biometrics and Bioinformatics (IJBB). **6**:111-121.

Sheik S. S., S. K. Aggarwal, A. Poddar, N. Balakrishnan and K. Sekar. 2004. A fast pattern matching algorithm. Journal of Chemical Information and Computer Sciences. **44**:1251-1256.

Sunday D. M. 1990. A very fast string-searching algorithm. Communication of ACM **33:**132-142