A Comparative Analysis of Various String Matching Algorithms

DU Vidanagama
Department of Information Technology, Faculty of Computing, General Sir John Kotelawala Defence University, Ratmalana, Sri Lanka
dush85@gmail.com

***Abstract*— *String-matching is a very important subject in the wider domain of text processing. It is used in almost all the software applications starting from text editors to the complex Network Intrusion Detection Systems (NIDS) which uses signature matching based on string matching. There are different string matching algorithms for solving the string matching problem. Such algorithms can greatly reduce the response time of the software applications which uses the string matching. The research problem of this research is to analyse the efficiency of different string matching algorithms. The objective of this research is to evaluate the execution time of the string matching algorithms and compare the efficiency of the algorithms. This study focuses on four selected string matching algorithms which are Naïve algorithm, Brute-Force algorithm, Boyer-Moore algorithm and Knuth-Morris-Pratt algorithm. The algorithms are tested against matching patterns with different lengths and the placement of the pattern (suffix, prefix or middle). The algorithms are written in Java Language and the execution time is measured in nanoseconds. The matching efficiencies of these algorithms are compared by the searching speed. It is observed that the performance of the Brute-Force algorithm and Naive algorithm is comparatively high. When considering the pattern placement, Brute-Force and Knuth-Morris-Pratt algorithms are faster when the pattern is in the prefix than suffix. Naive algorithm has no comparative difference of performance on the pattern placement.*

Keywords*— *String Matching Algorithms, Pattern Length, Pattern Placement

I. INTRODUCTION

String matching is a technique used to find the pattern within a given string. It is used in almost all the software applications starting from text editors to the complex Network Intrusion Detection Systems (NIDS) which uses signature matching based on string matching. String matching algorithms are used to find the matches between the pattern and the specified string. The pattern is denoted by P [1....m]. The text is denoted by T [1...n] *where m<=n.* If P occurs with shift s in T, then s is a valid shift; otherwise, s is an invalid shift.

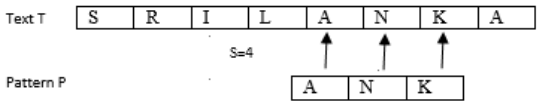

Figure 1.Overview of String Matching

The string matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T (Kumar et al., 2011). Fig. 1 shows this definition.

The string-matching algorithms can be broadly classified into two main categories. Those are Exact String-Matching algorithms and Approximate String-Matching algorithms (Rasool et al., 2012). The objective of this research is to compare the execution time of the string matching algorithms based on the pattern length and the pattern placement within the text. The author considers the problem of selecting the best performed string matching algorithms based on the pattern length and the pattern placement. This paper compares four different exact string matching algorithms based on their execution time. The algorithms that are covered by this paper are: Boyer-Moore Horspool Algorithm, Knuth-Morris-Pratt Algorithm, Brute-Force algorithm and Karp Rabin Algorithm.

II. LITERATURE REVIEW

A. *Exact String matching algorithms*

1) Brute-Force Algorithm (BF):

The BF Algorithm compares the pattern in the text starting from left to right, one character at a time, until a mismatch is found. This algorithm has no pre-processing phase. The algorithm can be designed to stop on either the rest occurrence of the pattern, or upon reaching the end of the text (Pandiselvam et al., 2014). The pattern matching starts with matching the first character of the pattern with the first character of the text. If the match doesn't find then it moves forward to the second character of the text and again compares the first character of the pattern with the second character of the text. In case if the match finds then moves to the second character of the pattern comparing it with the next character of the text. Example for BF is shown in Fig.2.

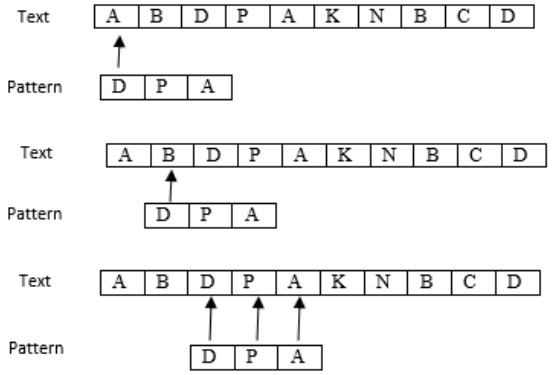

Figure 2. Brute Force Matching Example

2) Boyer-Moore Horspool Algorithm (BMH)

The Boyer-Moore (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 (Rasool et al.,2012). 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 (Fig. 3) (Rasool et al.,2012).These two shift functions are called the good-suffix shift(also called matching shift) and the bad-character shift(also called the occurrence shift) (Rasool et al.,2012).

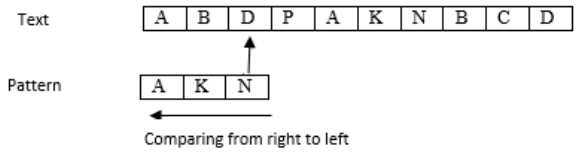

Figure 3. Overview of BMH algorithm

BMH is the simplification of the Boyer-Moore algorithm. The string being searched for is pre-processed to build a table that contains the length to shift when a bad match occurs. The BM algorithm creates the second "good suffix" table. Then the string to find is searched from the last character to the first. The bad match table is used to skip characters when a mismatch occurs. It contains the values for every character in the pattern.

value = length -index -1 ; (1)

where value of every remaining letter=length

Table 1. Bad Match Table

Letter	T	E	A	M	S	*
Value	8	6	2	3	1	8

Example for BMH algorithm is shown in Fig.4 by using the shift values of Table 1. Equation (1) is used to calculate the values.

3) Knuth-Morris-Pratt Algorithm (KMP)

The string being searched for in KMP is pre-processed to build a table of prefixes which is calculated for the chosen substring before the beginning of the matching phase. The matching starts with the left-most character of the pattern. The prefix table is used when a mismatch occurs. As there are two stages, the following algorithm can be used to create the prefix table (Table 2).

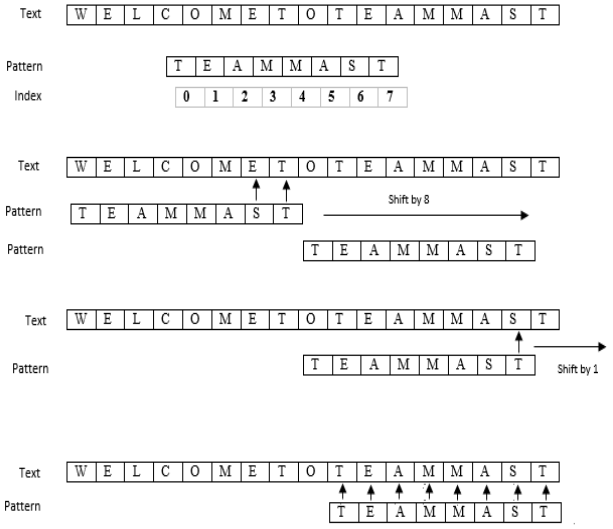

Figure 4. Boyer Moore Horspool Matching Example

Begin
length=Length of P
Prefix[1]=0
a=0
for b=2 upto length step 1 do
while a>0 & P[a+1]≠P[b] do
a=Prefix[a]
if P[a+1]=P[b] then
a=a+1
Prefix[b]=a
return Prefix
End

Table 2.Prefix Table

	1	2	3	4	5	6	7	8
P	C	O	C	A	C	O	L	A
Prefix	0	0	1	0	1	2	0	0

Then for the second stage to match the pattern the following algorithm can be used.

Begin
i=1,j=1,k=1
while n-k>=m do
while j<=m & T[i]=P[j] do
i=i+1
j=j+1
if j>m then output k
if Prefix[j-1]>0 then
k=i- Prefix[j-1]
else
if j=k then i=i+1
k=1
if j>1 then j=Prefix[j-1]+1
End

4) Rabin Karp Algorithm (RK)

This is the augmented version of Naïve approach by applying a powerful programming technique called hash function. At the pre-processing stage it calculates the hash value of pattern P (with m characters) with the hash value for each m-character substring of text T. Then it compares the numerical values instead of comparing the actual symbols. If any match is found, it compares the pattern with the substring by naive approach. Otherwise it shifts to next substring of T to compare with P using the hash values. So the performance of RK algorithm depends on the efficient computation of hash value.

The strings can be treated as the array of characters. Characters can be interpreted as integers, with their exact values depending on what type of encoding is being used (e.g. ASCII, Unicode). So the strings can be treated as array of integers.

Consider an M character sequence as an M-digit number in base R. The Equation (2) is to find the subsequence of string where t^i is the integer at i^{th} position (Sedgewick &Wayne, 2011). Equation (3) is used to determine the hash value of the specified sequence where Q is a large prime number (Sedgewick &Wayne, 2011).

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0 \quad (2)$$

$$H(x_i) = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0 \pmod{Q} \quad (3)$$

Furthermore, given x_i we can compute x_{i+1} for the next subsequence t[i+1 ..i+M] in constant time, as in Equation (4):

$$x_{i+1} = (x_i - t_i * R^{M-1}) R + t_{i+M} \quad (4)$$

Then the Equation (5) can be used to calculate the hash value of the next subsequence (Sedgewick &Wayne, 2011).

$$H(x_{i+1}) = (x_i - t_i * R^{M-1}) R + t_{i+M} \pmod{Q} \quad (5)$$

B. Time Complexity

Table 3 summarizes the time complexity of the selected algorithms in this research (Pandiselvam et al., 2014; <alg.csie.ncnu.edu.tw>). Π is the number of storing characters in BMH algorithm.

Table 3: Time complexity of algorithms

Algorithm	Pre-processing	Searching	Execution Time
BF	No	O(nm)	O(nm)
BMH	O(m+π) time complexity O(π) space complexity	O(mn)	O(mn)
KMP	O(m)	O(n)	O(m+n)
RK	O(m)	O(mn)	O(mn)

C. Previous research studies

The time performance of exact string pattern matching can be greatly improved if an efficient algorithm is used (Lovis and Baud, 2000). According to Pandiselvam et al (2014), the string matching algorithms were studied with biological sequences such as DNA and Proteins. It was analysed that KMP algorithm is relatively easier to implement because it never needs to move backwards in the input sequence and requires extra space. RK algorithm is used to detect the plagiarism which requires additional space for matching. BF algorithm does not require pre-processing of the text or the pattern, but the problem is its slowness and it rarely produces efficient result (Pandiselvam et. al, 2014). Also the BM algorithm is extremely fast for on large sequences, it avoids lots of needless comparisons by significantly pattern relative to text (Pandiselvam et. al, 2014).Based on the study of Kumar et.al (2011), and the best algorithm for usual searching purposes is BM while the best algorithm for long patterns and long payload text is KMP algorithm. This comparison had done for the algorithms for virus-signature detection.

Rasool et.al (2012) had done a research to compare the matching efficiencies of the string matching algorithms by searching speed, pre-processing time, matching time and the key ideas used in those algorithms. It was observed that performance of string matching algorithm was based on selection of algorithms used and also on network bandwidth. It was concluded that Boyer Moore and KMP string matching algorithms are efficient. Also it showed that BM Algorithm is fast in the case of larger alphabet. KMP decreased the time of searching compared to the Brute Force algorithm (Rasool et.al, 2012).

The main drawback of the Boyer-Moore type algorithms is the pre-processing time and the space required, which depends on the alphabet size and/or the pattern size (Baeza-Yates, 2002). So if the pattern is small it is better to use the BF algorithm (Baeza-Yates, 2002). If the alphabet size is large, then the Knuth-Morris-Pratt algorithm is a good choice (Baeza-Yates, 2002). It concluded that in all the other cases, in particular for long texts, the Boyer-Moore algorithm is better. Finally, the Horspool version of the BM algorithm is the best algorithm, according to execution time, for almost all pattern lengths (Baeza-Yates, 2002).

Considering the growing amount of text handled in the electronic patient record it was concluded that The BMH algorithm is a fast and easy-to-implement algorithm and better performed than BF algorithm (Lovis and Baud, 2000). As there were number of researches available for the comparison of different string matching algorithms, this research compares four different widely known algorithms based on the pattern length and pattern placement within the string.

III. RESEARCH MODEL AND HYPOTHESES

The literature identified that the searching speed is affected by the algorithm with the mediation factors of the length of the pattern and the placement of the pattern. The research model which was used for this research is in Fig.5.

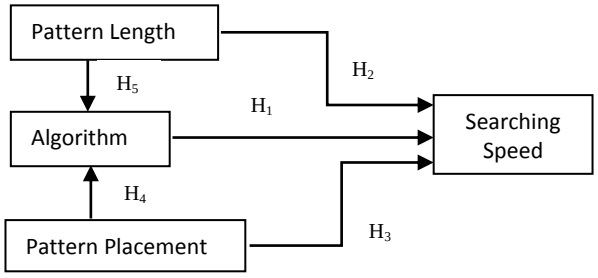

Figure 5. Research Model

Therefore the null hypotheses of the research based on Fig.5 are as follows:

H_1: All the algorithms have equal searching speed on average
H_2: All the pattern lengths have equal searching speed on average
H_3: All the pattern placements have equal searching speed on average
H_4: Algorithm and pattern lengths are independent or that interaction effect is not present
H_5: Algorithm and pattern placements are independent or that interaction effect is not present

IV. RESEARCH METHODOLOGY

All algorithms were implemented in Java language. The tests were conducted on a 2.5GHz Intel(R) Core(TM) i5-2450M processor with 4GB RAM. All the under- going processes are stopped to minimize the system interruptions and operating system processors. All the algorithms are tested in a similar environment.

The target string was an English text consisting with letters, spaces, commas and end-of line separators. The string consisted of 400 words and 2466 characters (including spaces).

The parameters of the each function calls used two string arguments and two integer arguments. All the functions have been implemented using the same function prototype *intSearchAlgorithm (String txt, String pattern, int N, int M)* where *txt* is the target string and *pattern* is the string which was going to match. *N* and *M* are the lengths of target and the pattern respectively. The functions returned -1 when the match was not found or when an error occurred, or else it returned the index of the first occurrence within the target. Pre-processing and computation of hash tables were done within the functions itself and the time cost was allocated to that function. Time for the function execution was measured in *System.nanoTime()*.

The measurements were recorded for the increased size of pattern length and different placements of the pattern within the target string. The each step was repeated for 30 iterations and the arithmetic mean was taken for the analysis. The overview of the method used as in Fig.6.

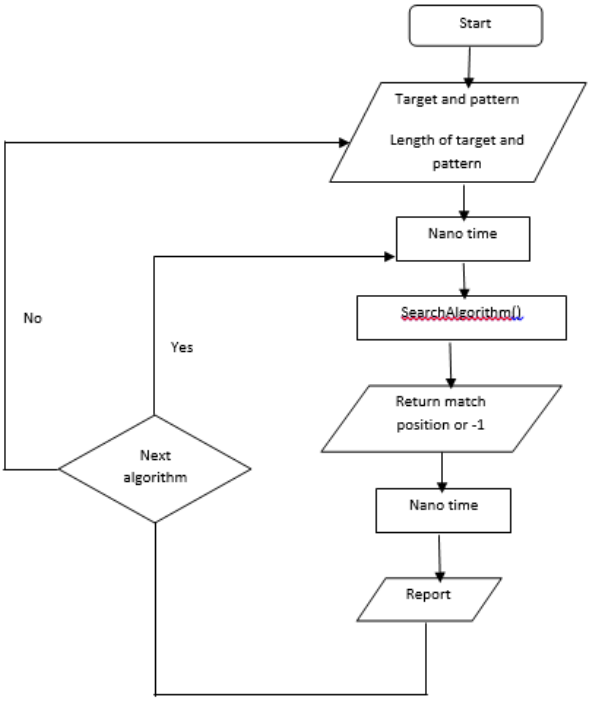

Figure 6. Overview of methodology

V. ANALYSIS

Statistical Analysis was performed using the SPSS 20 software. The statistical method Analysis of Variance (ANOVA) was used to compare the mean differences among and between the four algorithms that were selected. The plot of the data shows the variation among the algorithms with respect to the size of the patterns and the placement of the patterns within the target. The two-way ANOVA was used as there were two independent variables which were going to test the interaction on a continuous dependent variable, time complexity. Turkey HSD test was used to find the comparisons between the independent factors which have a significant difference of mean time.

VI. RESULTS

A. Comparison of algorithm and character length

Fig.7 shows the estimated marginal means of time over the pattern character length over the algorithms. According to Fig.7, the BMH algorithm has the lowest marginal mean time and RK algorithm has the highest marginal mean time over the pattern length. The mean time decreases in KMP and BMH when the pattern length increases. But for the RK algorithm, the marginal mean time increases when the pattern length increases. Also the KMP and BF gets slightly close when the pattern length increases.

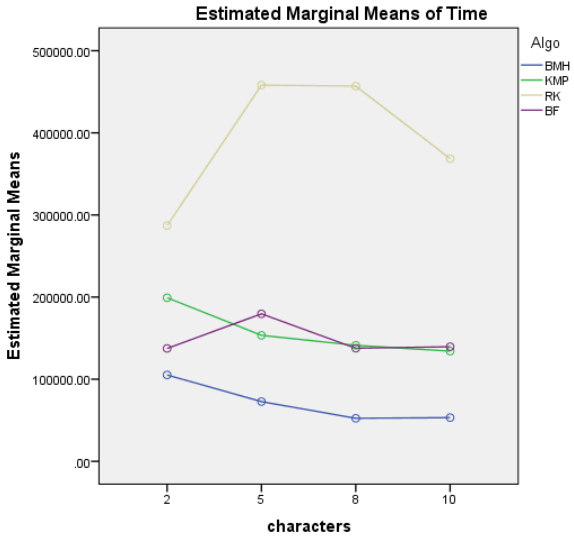

Figure 7. Marginal mean of time over pattern length

According to the significant value (p=0.008) in Table 4, the hypothesis H_4 is rejected. So there is a statistically significant interaction between the algorithm and the pattern length at p=0.008 level. According to the significant values (p=0.000 and p=0.24), the hypothesis H_1 is rejected and hypothesis H_2 is not rejected at 5% significance level. So there is a statistically significant difference in mean time between the algorithms. But there is no statistically significant difference between the pattern length over the mean time (p=0.24).

Table 4.Results of ANOVA

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	3.200E+13[a]	15	2.133E+12	17.656	.000
Intercept	7.335E+13	1	7.335E+13	607.143	.000
Algo	2.880E+13	3	9.598E+12	79.445	.000
characters	5.081E+11	3	1.694E+11	1.402	.240
Algo * characters	2.695E+12	9	2.995E+11	2.479	.008
Error	2.378E+14	1968	1.208E+11		
Total	3.431E+14	1984			
Corrected Total	2.698E+14	1983			

When comparing the mean time of the algorithms there is a statistical significant difference between all the algorithms except between KMP and BF where it shows very small difference at large pattern lengths (Table 5).

Table 5.Multiple Comparisons of Algorithms

(I) Algo	(J) Algo	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
BMH	KMP	-86132.7036*	22072.02200	.001	-142884.9004	-29380.5069
	RK	-321810.159*	22072.02200	.000	-378562.3560	-265057.9625
	BF	-77706.8387*	22072.02200	.002	-134459.0354	-20954.6420
KMP	BMH	86132.7036*	22072.02200	.001	29380.5069	142884.9004
	RK	-235677.456*	22072.02200	.000	-292429.6524	-178925.2589
	BF	8425.8649	22072.02200	.981	-48326.3318	65178.0616
RK	BMH	321810.1593*	22072.02200	.000	265057.9625	378562.3560
	KMP	235677.4556*	22072.02200	.000	178925.2589	292429.6524
	BF	244103.3206*	22072.02200	.000	187351.1238	300855.5173
BF	BMH	77706.8387*	22072.02200	.002	20954.6420	134459.0354
	KMP	-8425.8649	22072.02200	.981	-65178.0616	48326.3318
	RK	-244103.321*	22072.02200	.000	-300855.5173	-187351.1238

B. Comparison of algorithm and pattern placement

Fig.8 shows the estimated marginal means of time over the pattern placement over the algorithms. According to Fig.8, when the pattern is at the beginning of the target string the BF algorithm has the lowest mean time while KMP has the highest mean time. But when the pattern is moving to the end of the target, the mean time is increasing in BF algorithm. RK algorithm has the highest mean time when the pattern is at the end. When the pattern is at the end, BMH shows the lowest mean time.

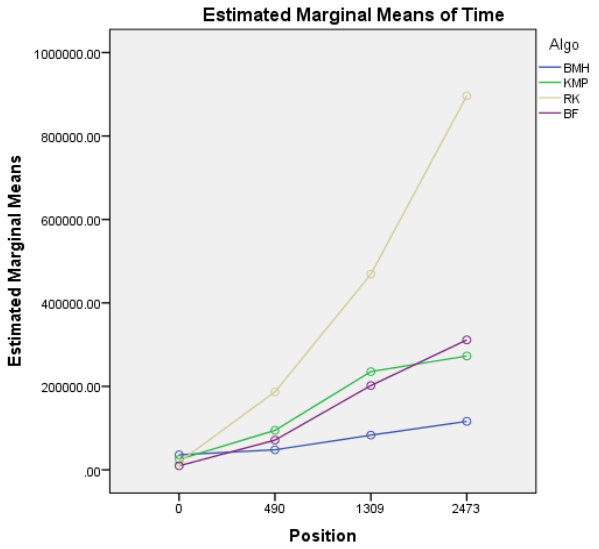

Figure 8. Marginal means over pattern placement

Table 6. Results of ANOVA

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	9.588E+13[a]	15	6.392E+12	72.342	.000
Intercept	7.335E+13	1	7.335E+13	830.189	.000
Algo	2.880E+13	3	9.598E+12	108.630	.000
Position	4.126E+13	3	1.375E+13	155.659	.000
Algo * Position	2.582E+13	9	2.869E+12	32.473	.000
Error	1.739E+14	1968	88358841087		
Total	3.431E+14	1984			
Corrected Total	2.698E+14	1983			

According to the significant value ($p<0.05$) in Table 6, the hypothesis H_5 is rejected. So there is a statistically significant interaction between the algorithm and the pattern placement at $p<0.05$ significant level. Also according to the significant value ($p<0.05$) the hypothesis H_3 is rejected. So there is a statistically significant the mean time difference between the pattern placements.

Table 7. Multiple Comparisons of Pattern placements

(I) Position	(J) Position	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
0	490	-77782.4677*	18875.53044	.000	-126315.7565	-29249.1790
	1309	-224984.617*	18875.53044	.000	-273517.9057	-176451.3282
	2473	-376788.093*	18875.53044	.000	-425321.3815	-328254.8040
490	0	77782.4677*	18875.53044	.000	29249.1790	126315.7565
	1309	-147202.149*	18875.53044	.000	-195735.4379	-98668.8604
	2473	-299005.625*	18875.53044	.000	-347538.9137	-250472.3363
1309	0	224984.6169*	18875.53044	.000	176451.3282	273517.9057
	490	147202.1492*	18875.53044	.000	98668.8604	195735.4379
	2473	-151803.476*	18875.53044	.000	-200336.7646	-103270.1871
2473	0	376788.0927*	18875.53044	.000	328254.8040	425321.3815
	490	299005.6250*	18875.53044	.000	250472.3363	347538.9137
	1309	151803.4758*	18875.53044	.000	103270.1871	200336.7646

VII.CONCLUSION

BMH algorithm is the fast and easy to implement algorithm for string matching. Among the four algorithms BMH is the fastest algorithm without considering the pattern length and pattern placement. RK is the slowest algorithm when increasing the pattern length and the pattern placement. The four algorithms have significant differences of the mean time of algorithm execution. Also there are significant differences of the mean time among pattern placement. Both the KMP and BF algorithms have very small differences of mean time across the pattern length and pattern placement.

If there is a choice between the KMP and BF algorithms for small patterns it is necessary to use BF, but when the pattern is increasing both are possible to use. Also when the pattern places at the end of the target it is possible to use KMP rather than BF.

ACKNOWLEDGEMENT

I would like to acknowledge all my colleagues and academic staff who supported me to fulfil this endeavour. Also I would give my sincere gratitude to my be-loved husband for giving me full support on this time.

REFERENCES

Horspool Algorithm, [Online] <alg.csie.ncnu.edu.tw/course/StringMatching/Horspool.ppt>

Kumar, A., Sharma,V., Kumar,S.(2011),A comparative analysis of various exact string matching algorithms for Virus-Signature Detection, *First International Conference on Emerging Trends in Soft Computing and ICT (SCICT-2011)*

Pandiselvam.P, Marimuthu.T, Lawrance.R.(2014), A Comparative Study on String Matching Algorithm of Biological Sequences, *International Conference on Intelligent Computing*

Rasool, A., Tiwari, A., Singla.G., Khare, N., (2012), String Matching Algorithms: A Comparative Analysis, *International Journal of Computer Science and Information Technologies (IJCSIT)*, 3 (2), pp.3394 – 3397

Sedgewick,R. & Wayne,K.(2011), Algorithms, 4thEdition, Pearson *Education Inc.*

Baeza-Yates, R.A (2002), String Searching Algorithms [Online] *http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap10.htm*

Lovis,C. and Baud, R.H.(2000), Fast Exact String Pattern matching Algorithms Adapted to the Characteristics of the Medical Language, *Journal of the American Medical Informatics Association*, 7(4):pp.378-91

BIOGRAPHY OF AUTHORS

Dushyanthi Udeshika Vidanagama is a probationary lecturer in the Department of IT, Faculty of Computing in Kotelawala Defence University, Sri Lanka. Dushyanthi earned her Master of Science in Management and Information Technology. Her teaching and research interests include Database Management Systems, Web Technologies, E-Learning and XML Databases.