Study of Selected Shifting based String Matching Algorithms

G.L. Prajapati,
PhD
Dept. of Comp. Engg.
IET-Devi Ahilya University, Indore

Mohd. Sharique
Dept. of Comp. Engg.
IET-Devi Ahilya University, Indore

Piyush Nagani
Dept. of Comp. Engg.
IET-Devi Ahilya University, Indore

Adarsh V.
Dept. of Comp. Engg.
IET-Devi Ahilya University, Indore

ABSTRACT
This paper presents detailed comparisons and analysis of shift-based exact string matching algorithms. The paper proposes comparison among these algorithms on the basis of execution time taken by the algorithms to completely match a given pattern on a given text. The algorithms have been analyzed on the following parameters: length of pattern, length of text, and number of characters in the text. This study will help in selecting the appropriate algorithm to be used in solving a particular real-life problem.

General Terms
Pattern Matching, Algorithms, Theoretical Computer Science.

Keywords
String Matching, Exact String Matching, Shifting Based, Execution Time, Performance Analysis.

1. INTRODUCTION
Without the use of string matching algorithms, the naïve approach [13] to exactly match two strings would be to match character by character i.e. brute force. However, the wider domain applications of string matching would require matching of copious data on which brute force will not prove to be feasible and hence require more optimal solution i.e. the string matching algorithms. String matching algorithms aim to find one or all occurrences of the string within a larger group of the text. String matching is further divided into two classes exact and approximate string matching. In exact String matching, pattern is fully compared with the selected text window (STW) of text string and display the starting index position. In approximate string matching, if some portion of the pattern matched with STW then it displays the results. This paper focuses on exact string matching algorithms. Further, exact string matching algorithms can be: shifting-based [13], automaton-based [13] or bit-parallel-processing based [13]. We will carry out the study on shifting based string matching algorithms. In such algorithms, optimization is based on the number of shifts performed during the execution and the length of each shift. Various applications of string matching are: Computational Molecular Biology [6], Voice Recognition [7], Intrusion detection in network [8], Object Recognition [9], Sequence/Sub-sequence and Image Detection, Plagiarism detection [10], Information security [11], Screen scrapers, Digital libraries, Word processors and natural language processing [12]. We will compare a set of eight such algorithms among themselves on the basis of the execution time on datasets of different types and pattern of different lengths.

2. SET OF STRING MATCHING ALGORITHMS UNDER ANALYSIS

2.1 Brute Force Algorithm (BF)
The brute force algorithm [13] starts with matching the first character of the pattern with that of the text, shifting one character forward at a time for the pattern as well as the text until the entire pattern matches or a mismatch is found. In case of a mismatch, we shift forward ahead by one character of text and start matching it with the first character of the pattern. When the complete pattern matches with the text, its starting location is returned and it continues matching with the next character.

2.2 Knuth Morris Pratt Algorithm (KMP)
The KMP string matching algorithm [3] uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves the worst case complexity to O(n). The basic idea behind KMP algorithm is: whenever it detects a mismatch (after some matches), it already knows some of the characters in the text (since they matched the pattern characters prior to the mismatch) and uses this information to avoid matching the characters that we know will anyway match.

2.3 Raita Algorithm
At each attempt Raita algorithm [4] first compares the last character of the pattern with the rightmost text character of the window, if match found, it compares the first character of the pattern with the leftmost text character of the window, again if match found, it compares the middle character of the pattern with the middle text character of the window. And finally if they match it actually compares the other characters from the second to the last but one, possibly comparing again the middle character. In case of any mismatch found above, pattern is shifted by the pre-computed bad character value of the last character of last character of the current text character window.

2.4 Skip Search Algorithm
The preprocessing phase of the Skip Search algorithm [1] pre-processes the pattern by computing the buckets for all distinct characters in the pattern i.e. index of those characters in the pattern. The search phase checks what is the k^{th} symbol in the text string, where k+1 is the length of the pattern. According to this symbol, align every identical symbol in the pattern and execute matching. If match found, return its location, else shift the pattern by pattern length.

2.5 Knuth Morris Pratt Skip Search Algorithm (KMP Skip Search)

The preprocessing phase of KMP Skip Search algorithm [1] computes the buckets for all characters of the alphabet, list table, MP table and KMP table.

2.6 Alpha Skip Search Algorithm(ASKIP)

Alpha skip search algorithm [1] during the preprocessing phase first determines the length of sub-string (L) in the pattern and then find all sub-strings in the pattern whose length matches that of the sub-string (L). The information about where the sub-strings are located in the pattern is stored in a trie. In the searching phase, it uses this information to compare the text with the pattern.

2.7 Two Way Algorithm

The Two Way algorithm [3] conceptually divides the search into two successive phases. The first phase consists in matching the right part of pattern against the given text, the letters of which are scanned from left to right. When a mismatch is found during the first phase, there is no second phase and the pattern is shifted to the right which brings the critical position to the right of the letter of the text that caused the mismatch else the second phase starts. The left part of the pattern is matched against the text where it is scanned from right to left. If a mismatch occurs here during the scanning, the pattern is shifted a number of places equal to the period of pattern. The resulting coinciding prefix is memorized in order to possibly increase the length of the next shift.

2.8 Shift-Or Algorithm

The shift-or algorithm [5] for exact string matching works by encoding the pattern to be matched in a bit matrix whose dimensions are the length of a 32 bit by the length of the alphabet. We then iterate through the characters of our text where the bitwise operations are used for pattern matching.

3. ANALYSIS

3.1 Experiment Specifications

The tests were run on 2194.922 MHz CPU (CPU family 6, Model 61, Stepping 4) with 4389.84 BogoMIPs. Number of CPU(s) and socket is 1 having 1 Thread(s) per core and 1 Core(s) per socket. The algorithms were tested in presence of 32K L1d cache, 32K of L1i cache,256K of L2 cache, and 3072K of L3 cache. Byte Order of CPU used is Little Endian. The computer was running Ubuntu 12.14 LTS. All programs were written in C programming language with gcc compiler 4.8.4 producing x86_64 "64-bit" code. Memory mapped files were used to remove the overhead of I/O operations using Hypervisor vendor "KVM" with "Full" virtualization type.

3.2 Analysis Specification

The types of datasets used for the analysis of algorithms were: genome, protein, english, and some random datasets rand2, rand4, rand8, rand16, rand32, rand64 and rand128 consist of 2, 4, 8, 16, 32, 64, 128 distinct characters respectively of varying length(s). Analysis is carried out on the basis of pattern length i.e. keeping the text of constant length. The pattern length has varied from 2^1 to 2^{12} and 500 instance of each pattern length was taken into account for each and every given text. To understand the effect of character set on pattern, analysis is carried out by explicitly restricting number of distinct characters used to generate the pattern. The datasets used are: genome, protein, random2 and random4. The above process is repeated. Some algorithms preprocess the pattern before they start matching. To normalize preprocessing time, pattern(s) are generated before the text generation and study is carried out on following datasets: english, genome, protein, random2, random4, random8, random16, random32, random64 and random128. Here, text length has been varied from pattern length to 2^{17} (131072) and 500 instance have been created for each text length whereas pattern length has been varied from 2^0 (1) to 21^{12} (2048). This analysis report is based on: the execution time taken by algorithms on all the above mentioned text(s) and pattern(s) working with the above mentioned datasets. All patterns and text were generated from same file to cover the average case i.e. Some of the pattern(s) and text(s) matches directly while some not.

4. RESULTS AND DISCUSSIONS

This analysis report is based on: the execution time taken by algorithms on all the above mentioned text(s) and pattern(s) working with the above mentioned datasets.The resultant execution time is calculated as shown in Table 2 along with its graphical representation in Figure 1. All patterns and text were generated from same file to cover the average case i.e. Some of the pattern(s) and text(s) matches directly while some not.

4.1 Pattern size based

Each cell(i , j) of the table 2 shows the best algorithm as per the execution time on i^{th} dataset and j^{th} pattern size.

4.2 On the basis of character set used to generate the pattern

Table 1 shows the algorithms which performed best when character set for pattern is explicitly defined. The columns represent the pattern sizes (j) and the rows represent the type of pattern(i).

Table 1: Showing the best algorithm in each category corresponding to character set for pattern

	2	4	8	16	32	64	128	256
Genome	shiftOr	shiftOr	shiftOr	shiftOr	shiftOr	shiftOr	shiftOr	shiftOr
Protein	bf	shiftOr	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	shiftOr
Rand2	askip	tw	skip	kmpskip	tw	kmpskip	tw	kmpskip
Rand4	raita	raita	raita	raita	raita	tw	tw	tw

4.3 On The Basis Of Text Length W.R.T Pattern Length

Table 4 shows the algorithms which performed best when pattern is fixed and text is generated from the same file of variable length such that we have 500 instance corresponding to each text length. The columns represent the text length and the rows represent the pattern length.

Table 2: Showing the execution time of the algorithms on English corresponding to length of pattern

Set of algorithms _____

Execution Time (in msec) _____

	bf	kmp	tw	so	raita	skip	kmpskip	askip
1024	0.000526	0.023157	0.00256	0.026014	0.001297	0.015739	0.001944	0.014336
2048	0.017653	0.041487	0.016864	0.048531	0.002657	0.022483	0.009408	0.015489
4096	0.051868	0.084182	0.044232	0.094883	0.004554	0.030824	0.018669	0.021557
8192	0.117088	0.15718	0.094816	0.185077	0.007917	0.040696	0.03214	0.023323
16384	0.245842	0.303809	0.201604	0.368955	0.015887	0.063581	0.064309	0.034348
32768	0.587071	0.643545	0.769876	0.801313	0.033352	0.091742	0.121166	0.047939
65534	1.106956	1.274676	0.948728	1.560118	0.062254	0.144584	0.22328	0.071812
131072	2.313597	2.743237	1.676007	4.807973	0.334975	0.760237	1.018269	0.215029

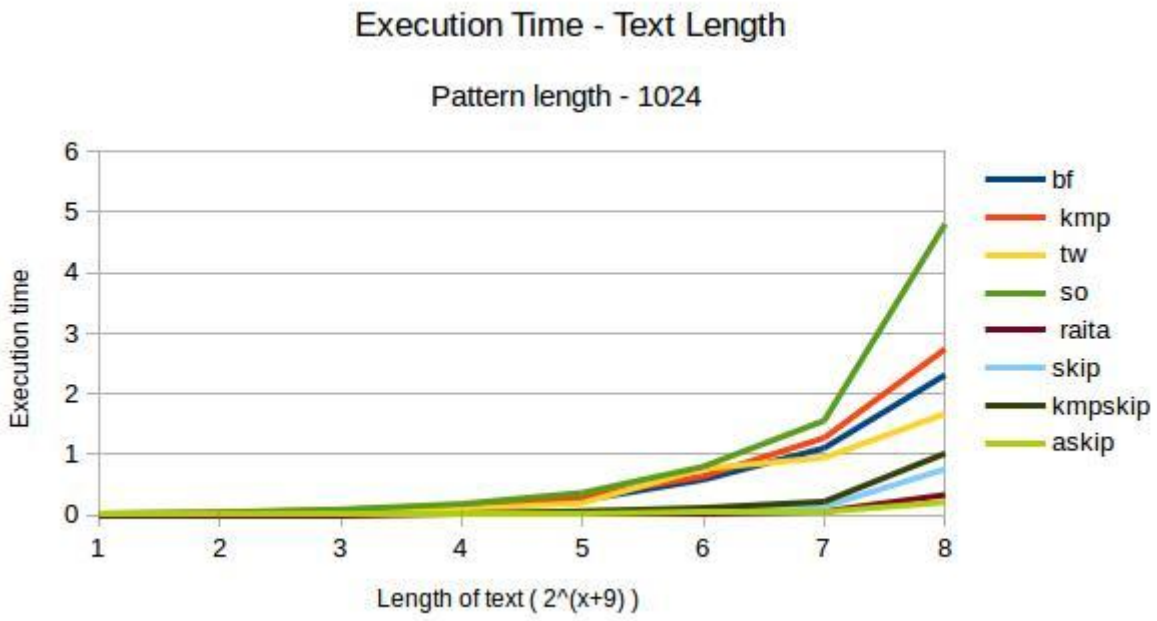

Figure 1. Graph showing execution time vs text length relation corresponding to Table 2

Table 3: Showing the best algorithm in each category corresponding to length of pattern

	2	4	8	16	32	64	128	256	512	1024	2048	4096
English	KMp Skip	ShiftOR	Raita	Raita	Askip	Askip	Askip	Raita	Askip	Raita	Raita	Raita
Genome	ShiftOR	ShiftOR	ShiftOR	Askip	Askip	Askip	Askip	Askip	Askip	Askip	Askip	ShiftOR
Midimusic	KMp Skip	KMP	Raita	KMPSkip	ShiftOR	KMP	ShiftOR	KMPSKip	ShiftOR	ShiftOR	Raita	ShiftOR
Protien	ShiftOR	ShiftOR	Raita	Raita	Raita	ASkip	Askip	Askip	Skip	Raita	Raita	Raita
Rand2	ShiftOR	ShiftOR	ShiftOR	ShiftOR	ShiftOR	ShiftOR	Askip	Askip	ShiftOR	ShiftOR	ShiftOR	ShiftOR
Rand4	ShiftOR	ShiftOR	Askip	Askip	Askip	Askip	Askip	ShiftOR	ShiftOR	ShiftOR	ShiftOR	ShiftOR
Rand8	ShiftOR	ShiftOR	ShiftOR	Askip	Askip	Askip	Skip	Skip	Skip	ShiftOR	ShiftOR	KMPSkip
Rand16	ShiftOR	ShiftOR	Raita	Raita	Raita	Skip	Skip	Skip	KMPSkip	Raita	Raita	KMPSkip
Rand32	KMp Skip	KMPSkip	KMPSkip	Raita	Raita	Raita	Skip	Skip	KMPSkip	Raita	Raita	KMPSkip
Rand64	KMp Skip	KMPSkip	KMPSkip	Skip	Raita	Skip	Skip	KMPSKip	KMPSkip	Raita	Raita	Raita
Rand128	KMp Skip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	Skip	Skip	Skip	Skip	Raita
Rand250	KMp Skip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	KMPSkip	KMPSKip	Skip	Skip	KMPSkip	KMPSkip

Table 4: Showing the best algorithm in Bible (The Holy Book) corresponding to given text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	so
2		bf	bf	bf	bf	bf	bf	bf	skip	so	so	skip	so	skip	so	skip	skip	so
4			bf	bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	skip	skip	askip/	skip	so
8				bf	bf	bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	raita
16					bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
32						bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip
64							bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip
128								bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
256									bf	raita	raita	raita	raita	raita	raita	raita	raita	raita
512										bf	raita	raita	raita	raita	raita	raita	raita	askip
1024											bf	raita	raita	raita	raita	raita	raita	raita
2048												bf	raita	raita	raita	raita	raita	raita

4.3.1 Information obtained from Table 5:

Each cell (i,j) of the following table shows the algorithm that performs best on genome dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.2 Information obtained from Table 6:

Each cell (i,j) of the following table shows the algorithm that performs best on protein dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

Table 5: Showing the best algorithm in Genome corresponding to given text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	so	bf	bf
2		bf	bf	bf	bf	bf	so	so	so	so	so	so	so	skip	so	so	so	so
4			bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so	so
8				bf	bf	bf	so	so	so	so	so	so	so	so	askip	askip	so	askip
16					bf	bf	so	so	raita	raita	so	raita	askip	askip	raita	askip	raita	askip
32						bf	so	raita	so	so	so	so	askip	askip	askip	askip	askip	askip
64							bf	bf	raita	raita	raita	askip	askip	askip	askip	askip	askip	askip
128								bf	raita	raita	raita	raita	raita	raita	askip	askip	askip	askip
256									bf	bf	raita	askip	askip	askip	askip	askip	askip	askip
512										bf	raita	askip	askip	askip	askip	askip	askip	askip
1024											bf	raita	askip	askip	askip	askip	askip	askip
2048												bf	askip	askip	askip	askip	askip	askip

Table 6: Showing the best algorithm in protein corresponding to given text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	so	so	so	so	bf	so	bf
2		bf	bf	bf	bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	askip	kmpskip	askip	so
4			bf	bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	skip	skip	skip	askip	skip
8				bf	bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	raita	raita
16					bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip	raita
32						bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
64							bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
128								bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
256									bf	raita	raita	raita	raita	raita	raita	askip	askip	askip
512										bf	raita	raita	raita	raita	raita	askip	askip	askip
1024											bf	raita	raita	raita	raita	raita	raita	askip
2048												bf	raita	raita	raita	raita	askip	askip

Table 7: Showing the best algorithm in rand2 (text containing 2 characters) corresponding to given text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	kmp	bf	bf
2		bf	bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so	kmp
4			bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so	so
8				bf	bf	so/	so	so	so	so	so	so	so	so	so	so	so	so
16					bf	bf	so	so	so	so	so	so	so	so	so	so	so	so
32						bf	so	so	so	so	so	so	so	askip	so	so	so	askip
64							bf	bf	tw	askip	askip	askip	askip	askip	askip	askip	askip	askip
128								bf	tw	tw	askip	askip	askip	askip	askip	askip	askip	askip
256									bf	tw	askip	askip	askip	askip	askip	askip	askip	askip
512										bf	raita	askip	askip	askip	askip	askip	askip	askip
1024											bf	askip	askip	askip	askip	askip	askip	askip
2048												bf	askip	askip	askip	askip	askip	askip

4.3.3 Information obtained from Table 7:

Each cell (i,j) of the following table shows the algorithm that performs best on rand2 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.4 Information obtained from Table 8:

Each cell (i,j) of the following table shows the algorithm that performs best on rand4 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.5 Information obtained from Table 9:

Each cell (i,j) of the following table shows the algorithm that performs best on rand8 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.6 Information obtained from Table 10:

Each cell (i,j) of the following table shows the algorithm that performs best on rand16 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.7 Information obtained from Table 11:

Each cell (i,j) of the following table shows the algorithm that performs best on rand32 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.8 Information obtained from Table 12:

Each cell (i,j) of the following table shows the algorithm that performs best on rand64 dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

4.3.9 Information obtained from Table 13:

Each cell (i,j) of the following table shows the algorithm that performs best on rand128dataset as per execution time taken for matching only (excluding pre-processing time). The columns represent text length (j) and the rows represent pattern length (i).

Table 8: Showing the best algorithm in rand4 (text containing 4 characters) corresponding to text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	kmp	bf
2		bf	bf	bf	bf	bf	bf	so	so	so	so	so	so	so	askip	so	so	bf
4			bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so	so
8				bf	bf	bf	so	so	so	so	so	so	askip	askip	askip	so	so	askip
16					bf	bf	so	so	so	so	so	so	askip	askip	askip	askip	askip	askip
32						bf	bf	so raita	so	so	so	so	askip	askip	askip	askip	askip	askip
64							bf	raita	raita	raita	raita	askip	askip	askip	askip	askip	askip	askip
128								bf	raita	raita	raita	askip	askip	askip	askip	askip	raita	askip
256									bf	raita	raita	raita	askip	askip	askip	askip	askip	askip
512										bf	raita	askip	askip	askip	askip	askip	askip	askip
1024											bf	raita	askip	askip	askip	askip	askip	askip
2048												bf	askip	askip	askip	askip	askip	askip

Table 9: Showing the best algorithm in rand8 (text containing 8 characters) corresponding to text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	bf	so	bf	so	bf	bf	so	bf
2		bf	bf	bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so
4			bf	bf	bf	bf	bf	so	so	so	so	so	so	so	raita	so	raita	so
8				bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita	so
16					bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	askip	raita
32						bf	bf	raita	raita	raita	raita	skip	skip	askip	askip	askip	askip	askip
64							bf	raita	raita	raita	raita	raita	raita	askip	askip	askip	askip	askip
128								bf	raita	raita	raita	raita	skip	skip	skip	askip	askip	askip
256									bf	raita	raita	raita	skip	askip	skip	askip	askip	askip
512										bf	raita	raita	raita	raita	askip	askip	askip	askip
1024											bf	raita	raita	askip	askip	askip	askip	askip
2048												bf	raita	askip	askip	askip	askip	askip

Table 10: Showing the best algorithm in rand16 (text containing 16 characters) corresponding to text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	so	so	bf	so	so	bf	so	bf
2		bf	bf	bf	bf	bf	bf	so	so	so	so	so	so	so	so	so	so	so
4			bf	bf	bf	bf	bf	so	so	so	so	so	so	so	so	skip	raita	so
8				bf	bf	bf	bf	so	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
16					bf	bf	bf	so	raita	raita	raita	raita	raita	raita	askip	askip	askip	raita
32						bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	askip	askip	raita
64							bf	raita	raita	raita	raita	raita	raita	raita	skip	askip	askip	askip
128								bf	raita	raita	raita	raita	skip	skip	skip	askip	askip	skip
256									bf	raita	raita	raita	raita	raita	raita	raita	askip	askip
512										bf	raita	raita	raita	askip	askip	askip	askip	askip
1024											bf	raita	raita	raita	askip	askip	askip	askip
2048												bf	raita	askip	askip	askip	askip	askip

Table 11: Showing the best algorithm in rand32 (text containing 32 characters) corresponding to text and pattern length

Text Length ----------

Pattern Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	bf	so	kmpskip	so	so	kmpskip	so	so
2		bf	bf	bf	bf	bf	bf	bf	so	skip	skip	skip	skip	skip	askip	skip	skip	so
4			bf	bf	bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	skip	skip	skip	skip
8				bf	bf	bf	bf	raita	raita	raita	raita	skip	skip	skip	skip	skip	askip	skip
16					bf	bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	raita
32						bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	raita
64							bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	skip	skip
128								bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	raita
256									bf	raita	raita	raita	raita	raita	raita	raita	skip	askip
512										bf	raita	raita	raita	raita	raita	raita	askip	askip
1024											bf	raita	raita	raita	raita	askip	askip	raita
2048												bf	raita	raita	raita	askip	askip	askip

Table 12: Showing the best algorithm in rand64 (text containing 64 characters) corresponding to text and pattern length

Pattern Length \ Text Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	so	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	so
2		bf	bf	bf	bf	bf	bf	bf	bf	skip	skip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	so
4			bf	bf	bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	skip	askip	kmpskip	skip
8				bf	bf	bf	bf	skip	skip	skip	skip	skip	skip	skip	skip	kmpskip	skip	skip
16					bf	bf	bf	bf	raita	raita	raita	raita	skip	skip	skip	skip	skip	kmpskip
32						bf	bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip
64							bf	bf	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip
128								bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	skip
256									bf	raita	raita	raita	raita	raita	raita	raita	skip	skip
512										bf	raita	raita	raita	raita	raita	raita	raita	skip
1024											bf	raita	raita	raita	raita	raita	raita	askip
2048												bf	raita	raita	raita	raita	askip	askip

Table 13: Showing the best algorithm in rand128 (text containing 128 characters) corresponding to text and pattern length

Pattern Length \ Text Length	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
1	bf	bf	bf	bf	bf	bf	bf	bf	bf	bf	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	bf
2		bf	bf	bf	bf	bf	bf	bf	skip	kmpskip	kmpskip	skip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	so
4			bf	bf	bf	bf	bf	bf	skip	skip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	kmpskip	skip
8				bf	bf	bf	bf	skip	skip	skip	kmpskip	skip	kmpskip	kmpskip	skip	kmpskip	kmpskip	skip
16					bf	bf	bf	skip	skip	skip	skip	kmpskip	kmpskip	skip	skip	kmpskip	skip	skip
32						bf	bf	skip	skip	skip	raita	raita	raita	skip	skip	skip	skip	skip
64							bf	skip	raita	raita	raita	raita	raita	raita	raita	raita	raita	skip
128								bf	raita	raita	raita	raita	raita	raita	raita	raita	skip	skip
256									bf	raita	raita	raita	raita	raita	raita	raita	raita	raita
512										bf	raita	raita	raita	raita	raita	raita	raita	raita
1024											bf	raita	raita	raita	raita	raita	raita	raita
2048												bf	raita	raita	raita	raita	raita	kmpskip

5 CONCLUSIONS

It can be seen from the aforementioned analysis that for different dataset, text size and pattern size, all algorithms under analysis perform differently i.e. certain algorithms perform better as compared to others. Irrespective of the datasets under consideration, for smaller text length (fixed pattern size), brute force algorithm outperforms other algorithms. For sufficiently large text and pattern size (fixed pattern size), on datasets bible, protein, rand64 and rand 128, raita algorithm performs better and on genome, rand2, rand4, rand8 and rand16, Alpha Skip Algorithm performs better. Although, it should be duly noted that Alpha Skip Algorithm has very large pre-processing time as compared to other algorithms. However, in case of repeated use of the same on single pattern size, its effect is nullified as pre-processing time is required only once and search time is repeatedly required. The appropriate algorithm can be chosen as per the desired problem domain using the performance analysis of exact string matching algorithms presented in thujs research paper.

6 REFERENCES

[1] C. Charas, T. Lecroq and J. D. Pehoushek. 2005. A very fast string matching algorithm for small alphabets and long patterns, Lecture notes in Computer Scienc, Volume 1448, pp. 55-64.

[2] J. J. McConnell, Analysis of Algorithms,Canisius College, pp 125-128.

[3] M. Crochemore and D. Perrin, 1991, Two Way String Matching, Journal of the Association for Computing Machinery, Vol. 38, No 3, pp.65I-675.

[4] T. Raita, 1992, Tuning the Boyer–Moore–Horspool String, Searching Algorithm, Software Practice and Experience, Vol 22(10),pp 879–884.

[5] R. Baeza-Yates and G. H. Gonnet, 1992, A New Approach To Text Searching, Communication of the ACM,Vol.35, No.10.

[6] James Lee Holloway, 1992, Algorithms for string matching with applications in molecular biology, Oregon State University Corvallis, OR, USA

[7] Luqman Gbadamosi, 2013, VOICE RECOGNITION SYSTEM USING TEMPLATE MATCHING, International Journal of Research in Computer Science eISSN 2249-8265 Volume 3 Issue 5 (2013) pp. 13-17.

[8] Saurabh Tiwari and Deepak Motwani, 2014, Feasible Study on Pattern Matching Algorithms based on Intrusion Detection Systems. *International Journal of Computer Applications* 96(20):13-17.

[9] Kavita Ahuja 1, Preeti Tuli 2, 2013,Object Recognition by Template Matching Using Correlations and Phase Angle Method, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 3.

[10] Prabhudeva S, 2008, Plagiarism Detection by using Karp-Rabin and String Matching Algorithm Together Sonawane Kiran Shivaji Master of Engineering, Computer Engineering, Ahmednagar, Maharashtra, IJCSNS International Journal of Computer Sci 314 ence and Network Security, VOL.8 No.10.

[11] Jamuna Bhandari and Anil Kumar, 2013, International Journal of Computer Science Engineering (IJCSE), Vol. 2 No.05 .

[12] DavidM.Magerman,1994 ,NATURAL LANGUAGE PARSING AS STATISTICAL PATTERN RECOGNITION, of computer science and the committee on graduate studies of stanford university in partial fulfillment of the requirements for the degree of doctor of philosophy.

[13] Koloud Al-Khamaiseh, 2014, A Survey of String Matching Algorithms Int. Journal of Engineering Research and Applications www.ijera.com ISSN: 2248-9622, Vol. 4, Issue 7(Version 2), pp.144-156.