Indian Journal of Science and Technology, *Vol 10(3), DOI: 10.17485/ijst/2017/v10i3/84939, January 2017*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Single and Multiple Pattern String Matching Algorithm

Chinta Someswara Rao[1*] and K. Butchi Raju[2]

[1]Department of CSE, S R K R Engineering College, Bhimavaram, W.G. District - 534 204, A.P. India; chinta. someswararao@gmail.com
[2]Department of CSE, GRIET, Hyderabad - 500090, Telangana, India; butchiraju.katari@gmail.co

Abstract

Background/Objectives: Information Retrieval Systems (IRS) are playing an eminent role in different applications like World Wide Web, DNA sequence retrieval, etc. Basically, the IRS systems use the string matching algorithms. **Methods/ Statistical Analysis**: Since IRS uses string matching algorithms. If string matching algorithms quality is improved then automatically information retrieval system will achieve the most relevant results. For this retrieval purpose in this paper, single pattern and multiple pattern string matching algorithms are proposed. **Findings:** To assess the efficiency of the proposed single pattern and multiple pattern string matching algorithm in this paper, DNA sequences of different monkeys datasets called Saimiri boliviensis (2.46 Gb), Pan paniscus (2.71 Gb), Macaca nemestrina (2.78 Gb), Colobus angolensis palliates(2.8 Gb), Tarsius syrichta (3.31 Gb) are considered and different tetra patterns TAGA, TCTG are searched in this data sets. From the experimental results, it is observed that proposed single pattern and multiple pattern string matching algorithms outperforms compared to other well-known string matching algorithms. **Application/Improvements:** It is also observed that multiple pattern string matching algorithm reduces search time and unnecessary comparisons when compared to single pattern string matching and other existing string matching algorithms. These proposed algorithms very useful when we searching about the multiple patterns.

Keywords: DNA Sequence, Information Retrieval Systems (IRS), Multiple Patterns, String Matching, Single Pattern

1. Introduction

String matching is the process of searching for the occurrence of a specified pattern in a given text. It is one of the important aspects of many applications as said in the abstract. It is divided into single pattern and multiple patterns[1-9] groups.

Single pattern further divided into suffix, prefix and substring matching algorithms. The first string matching algorithm is presented in[10], which searches the single pattern. In this first character of the pattern is compared with the text, if complete match occurs then one position is shifted to right, if mismatch occurs at any place then also the search process is shifted one position to right.

Another algorithm proposed in[11], which also searches the single pattern. In this algorithm first character of the pattern is compared with the text, if complete match/mismatch occurs then the shift position is calculated with shift rule.

Algorithm proposed in[12]also searches the single pattern, in which right most character of the pattern is compared with the text character, if complete match occurs then the shift is calculated with good suffix rule, if mismatch occurs at any place then shift position is calculated with bad character rule. Later many of the researchers revisited these three algorithms and variants are proposed and still research is continued in this single pattern area.

**Author for correspondence*

Another group of the string matching is multiple patterns, in which multiple patterns are searched instead of single pattern. Algorithm proposed in[13] is some other string matching algorithm which is one of the variants of the string matching algorithm presented in[11], which consists of two portions. In the first portion, the finite state machine is constructed with set of possible words, where as in the second portion the text is applied to pattern state machine.

Algorithm proposed in[14], which is another multiple pattern string matching algorithm which is the combination of algorithms in[12] and in[13]. It has pre processing and searching phases. In pre-processing, the finite state machine is constructed with algorithm in[13] whereas in searching, the search process presented in[12] is used to search the pattern. If mismatch/complete match occurs then the shift position is calculated with shift table which is calculated with BM bad character and good suffix rules.

Algorithm presented in[15] is one of the variants of the algorithm presented in[12]. It also consists of pre-processing and search processes. In pre-processing, SHIFT, PREFIX and HASH tables are constructed. In search process, the right most character of the pattern is compared with the character of the text, if complete match/mismatch occurs then the shift position is calculated with SHIFT, PREFIX and HASH table values.

Algorithm presented in[16] is one of the variants of algorithm presented in[10]. This algorithm first calculates the hash values of both the windows of pattern and text, and these values are compared, if they are equal then comparison is performed as like in[6], if the values are not equal then it calculates the hash values for next window.

2. Methodology

The methodology of the proposed approach is divided into four phases namely input, divide, process and output as shown in Figure 1. In input phase, the input file, pattern file and processor count is accepted. In divide phase, the input file is divided into number of sub files based on the target processors count, and the sub files are distributed among the target processors along with the pattern. In process phase, the actual search process is performed with the single and multiple pattern string matching algorithms. In output phase, the occurrence count, line number and occurrence position is returned.

Algorithm 1:	**String Matching Algorithm Process**
/* Input */	
1.	Text file of size n,
2.	Pattern file of size m and
3.	Number of processors (p) available.
/* Divide */	
4.	Undergo text file divided into ' i ' number of subtexts, each i contains (n/p)+m-1 text characters.
5.	The divided sub text files are stored in a directory.
6.	Broadcast these sub text files to each processor in the network along with the pattern.
/* process */	
7.	for i←0 to sub_text file.length do
8.	begin
9.	Each Processor searches the pattern string in the given Sub text file using the single(Algorithm 2)or multiple(Algorithm 3) string matching algorithms
10.	Each processor stores the match count and occurrence position
11.	end for;
/*Output */	
12.	Number of occurrences and occurrence position

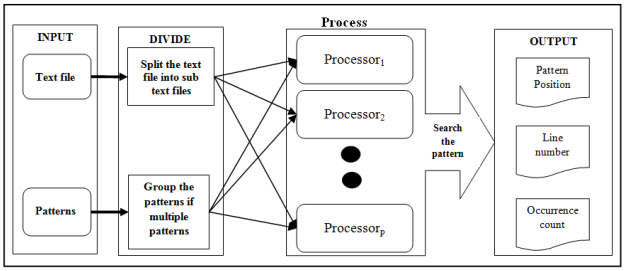

Figure 1. System structure.

Algorithm 2:	**Single Pattern String Matching Algorithm**
/*Input*/	
1.	Text (T), Pattern(P)
/*Initialization:*/	
2.	count=0, position=0, i=0, n=length of the text,
3.	T=" ",P=" ", StringBuffer sb=null
4.	**/* Main Method */**
5.	void main()

```
6.      begin
7.          File.Open()
8.            while ((Line = File.readLine()) != null)
9.                  sb.append(Line);
10.           search(sb,pattern);
11,     end;
/* Search function */
12.     void search(StringBuffer sb,char[] pattern)
13.     begin
14.     T= sb.toCharArray();
15.     P=pattern.toCharArray();
16.     n=T.length(),m=P.length();
17.     initial_check=0;
18.     for i←0 to n-1
19.     begin
20.         if (initial_check==0)
21.         i=shift(i,P);
22.         int j = m-1;
23.         while(j >= 0 && P[j] == T[i+j])
24.             j--;
25.         if (j < 0)
26.         begin
27.         initial_check=1;
28.         count++;
29.         print the occurrence position
30.         i=shift(i,P);
31.         end
32.         else
33.         i=shift(i,P);
34.     end for;
35.     end;
/* shift function */
36.     int shift(int i, char[] Pattern)
37.     begin
38.       while ( (Text[i]!=Pattern[0]) && (i<=n) ) )
39.       i++;
40.     return i;
41.     end;
/*Output*/
42.     The number of occurrences and position of the pattern
```

Algorithm 3:	Multiple Pattern String Matching Algorithm
/* Input */	
1.	Text (T), Pattern_Set
/*Initialization:*/	
2.	count=0, position=0, i=0, n=T.length, Pattern_Set={ Set of Patterns }
3.	T=" ",P=" ", StringBuffer sb=null
/* Main Method */	
4.	void main()
5.	begin
6.	Pattern_Set.Open()
7.	while ((pattern = Pattern_Set. readPattern()) != null)
8.	begin
9.	/* Group the Patterns with Left Most Character and length;*/ $group_i$.add(pattern);
10.	end;
11.	File.Open()
12.	while ((Line = File.readLine()) != null)
13.	sb.append(Line);
14.	while ((pattern=$group_i$.read()) !=null)
15.	begin
16.	search(sb, pattern);
17.	i++;
18.	end;
19.	end;
/* Output */	
20.	The number of occurrences and positions of the patterns

Note: The search and shift functions will be used in the Algorithm 3 as in those of Algorithm 2.

3. Data Set

In general a monkey chromosome contains 10 (TAGA, TCAT, GAAT, AGAT, AGAA, GATA, TATC, CTTT, TCTG and TCTA) Complex DNA Index Structures (CODIS), here these 10 CODIS are considered as search patterns.

To assess the efficiency of the proposed string matching algorithms, all the chromosomes of Saimiri boliviensis (2.46 Gb), Pan paniscus (2.71 Gb), Macaca nemestrina (2.78 Gb), Colobus angolensis palliates(2.8 Gb), Tarsius syrichta (3.31 Gb) and Saimiri boliviensis (2.46 Gb) are considered as data sets[17]. The proposed and AC, CW, WM and RK string matching algorithms are implemented in JAVA on WINDOWS 8.1 Operating System with 4 GB of RAM. The experimental

Table 1. Single pattern, multiple pattern and existing string matching algorithms search results for all datasets

	AC	CW	WM	RK	Proposed Single pattern	Proposed Multiple pattern
Saimiri boliviensis (2.46Gb)	909435	886314	865080	843869	820746	400534
Pan paniscus (2.71Gb)	1030662	1009430	988199	965078	942957	461745
Macaca nemestrina (2.78Gb)	1075516	1054284	1033071	1010950	987829	493868
Colobus angolensis palliates(2.8Gb)	1080144	1057022	1033901	1012690	989545	494868
Tarsius syrichta (3.31Gb)	1157898	1136666	1115435	1092311	1068188	534067

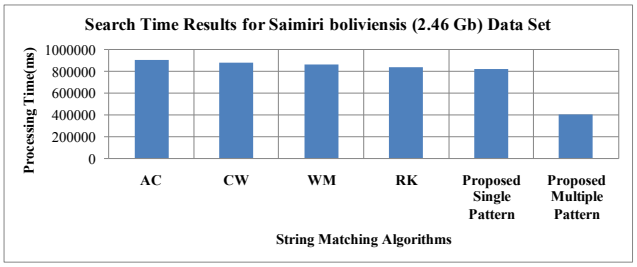

Figure 2. Search Time Results for Saimiri boliviensis (2.46 Gb) Dataset.

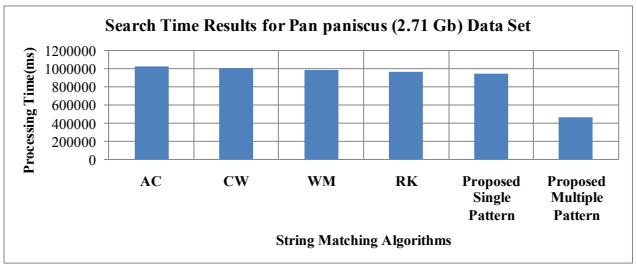

Figure 3. Search Time Results for Pan paniscus (2.71 Gb) Dataset.

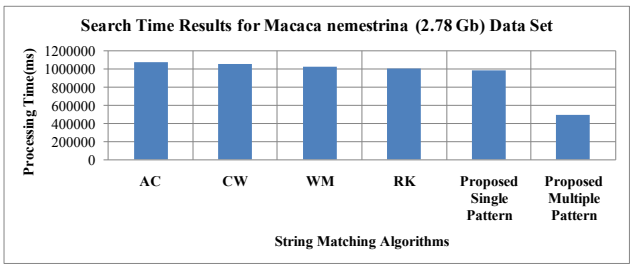

Figure 4. Search Time Results for Macaca nemestrina (2.78 Gb) Dataset.

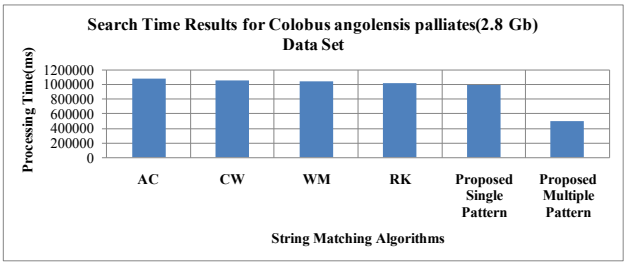

Figure 5. Search Time Results for Colobus angolensis palliates(2.8 Gb) Dataset.

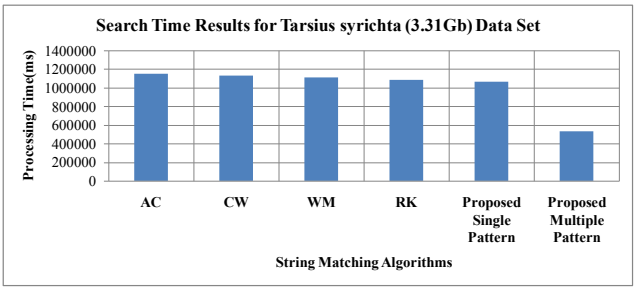

Figure 6. Search Time Results for Tarsius syrichta (3.31Gb) Dataset.

results are shown in Table 1. From these results, graphs are drawn and shown in Figure 2, 3, 4, 5 and 6.

From the Figures 2,3,4,5 and 6, the following observations can be made:

- From the Figure 2 it can be observed that, the proposed multiple pattern string matching algorithms takes 400534 ms to search the Saimiri boliviensis data set of size 2.46 Gb, whereas AC algorithm takes 909435 ms for the same data set which is double the search time when compared to proposed multiple pattern string matching algorithm.
- From the Figure 3 it can be observed that, the proposed single pattern string matching algorithm reduces the search

time 87705 ms than AC algorithm, 66473 ms than CW algorithm, 45242 ms than WM algorithm, 22121 ms than RK algorithm for Pan paniscus dataset of size 2.71 Gb.

- From Figures 4, 5 and 6, it can be observed that, the proposed multiple pattern string matching algorithm takes maximum search time 534067 ms to search the Tarsius syrichta data set of size 3.31 Gb, because it is the largest data set, whereas other algorithms also require maximum search time for this same data set on the other hand with the proposed multiple pattern string matching algorithm the search time is reduced to half of those of the existing algorithms.

3.1 Parallel Search Times of Single, Multiple and Existing String Matching Algorithms

In Algorithm 1, we discussed about the parallel processing, since it has four phases, input, divide, process and output. For example, we consider eight processors in the network, from algorithm 1, the input text file is taken, and divided into eight parts ($\because \frac{n}{p} + (m-1)$, where p is the number (8) of processors), and distribute among the processors along with the pattern. The search process is performed by processors, and later the output is returned. To assess the efficiency of the proposed single, multiple and existing string matching algorithms, all the chromosomes of Saimiri boliviensis (2.46 Gb), Pan paniscus (2.71Gb), Macaca nemestrina (2.78 Gb), Colobus angolensis palliates (2.8 Gb) and Tarsius syrichta (3.31 Gb) are considered as data sets and executed in the parallel processing environment. The experimental results are shown in Table 2. From these results the graph is drawn as shown in Figure 7.

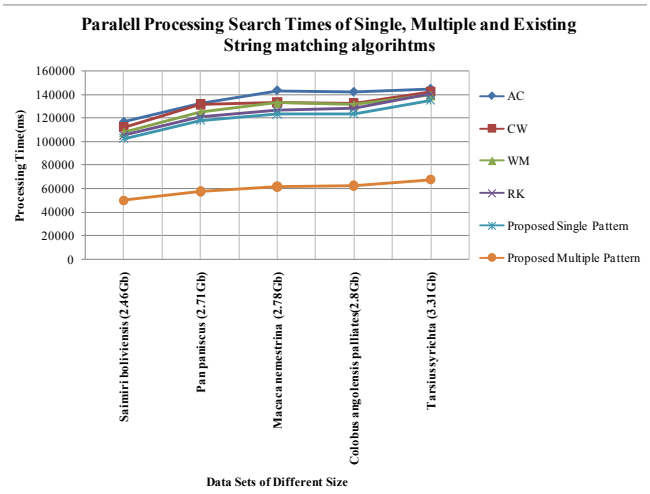

Figure 7. Parallel Processing Search Times of Single, Multiple and Existing String matching Algorithms.

From the Figure 7, it is noticed that the proposed multiple string matching algorithm executed under parallel processing the search time is reduced by seven times that of sequential processing. The search times of the single and existing algorithms are also reduced as like the proposed multiple pattern algorithms.

From Figures 2, 3, 4, 5, 6 and 7, it is concluded that the proposed single and multiple string matching algorithms reduced the search times than those of the existing string matching algorithms in both sequential and parallel processing.

Table 2. Parallel processing search times of single, multiple and existing string matching algorithms

	Saimiri boliviensis (2.46Gb)	Pan paniscus (2.71Gb)	Macaca nemestrina (2.78Gb)	Colobus angolensis palliates(2.8Gb)	Tarsius syrichta (3.31Gb)
AC	116594	132136	143402	142124	144737
CW	112334	131455	133826	132310	142439
WM	108135	125120	132956	131372	139621
RK	105484	120801	126385	128221	140238
Proposed Single Pattern	102595	117870	123494	123755	135248
Proposed Multiple Pattern	50067	57725	61803	62547	67604

4. Conclusions

The proposed single and multiple pattern string matching algorithms are executed with DNA genome data sets. The experimental results have shown that the single pattern string matching algorithm reduced the search time when compared with existing string matching algorithms. Whereas, the multiple string matching algorithms outperform in terms of search time as compared to proposed single pattern and existing string matching algorithms.

5. References

1. Verma A, Kaur I, Singh I. Comparative analysis of data mining tools and techniques for information retrieval. Indian Journal of Science and Technology. 2016 Mar; 9(11). Doi no:10.17485/ijst/2016/v9i11/81658
2. Singhal S, Amit A. Modern information retrieval: A brief overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 2001:35-43.
3. Meenakshi A, Suganthi P, Aghila R, Nirmala S. Information retrieval using dynamic decision Quadtree in soil database. Indian Journal of Science and Technology. 2016 Mar; 9(10). Doi no:10.17485/ijst/2016/v9i10/87954
4. Pavani T, Das RP, Naga Jyothi A, Sampath Dakshina Murthy A. Investigations on array pattern synthesis using nature inspired metaheuristic algorithms. Indian Journal of Science and Technology. 2016 Jan; 9(2). Doi no:10.17485/ijst/2016/v9i2/80642
5. Radhakrishnan S, Neduncheliyan S, Thyagharajan KK. A review of downlink packet scheduling algorithms for real time traffic in LTE-advanced networks. Indian Journal of Science and Technology. 2016 Jan; 9(4). Doi no:10.17485/ijst/2016/v9i4/84061
6. Rao CS, Raju SV. Next Generation Sequencing (NGS) database for tandem repeats with multiple pattern 2°-shaft multicore string matching. Genomics Data. 2016:307-17.
7. Rao CS, Raju SV. Similarity analysis between chromosomes of *Homo sapiens* and monkeys with correlation coefficient, rank correlation coefficient and cosine similarity measures. Genomics Data. 2016:202-9.
8. Rao CS, Balakrishna A, Raju MB, Raju SV. A frame work for XML ontology to STEP-PDM from express entities: A string matching approach. Annals of Data Science. 2016 Dec 1; 3(4):469-507.
9. Rao CS, Raju SV. Concurrent Information Retrieval System (IRS) for large volume of data with multiple patterns multiple (2^N) shaft parallel string matching. Annals of Data Science. 2016:1-29.
10. Aho A, Alfred V, John E, Hopcroft H. Design and Analysis of Computer Algorithms, Pearson Education India; 1974. p.470.
11. Knuth D, James H, Morris Jr, Pratt V. Fast pattern matching in strings. SIAM Journal on Computing. 1977; 6(2):323-50.
12. Boyer RS, Moore JS. A fast string searching algorithm. Association of Computing Machinary. 1977; 20(10): 262–72.
13. Aho AV, Corasick MJ. Efficient string matching: An aid to bibliographic search. Communications of the ACM. 1975 Jun 1; 18(6):333-40.
14. Commentz-Walter B. A string matching algorithm fast on the average. Berlin Heidelberg: Springer; 1979. p. 118-32.
15. Wu W, Sun S, Manber U. A fast algorithm for multi-pattern searching. 1994. p. 1-11.
16. Karp RM, Rabin MO. Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development. 1987 Mar; 31(2):249-60.
17. Nuclei Acid Research. 2016. Available from: http://www.ncbi.nlm.nih.gov