Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Index Based Multiple Pattern Matching Algorithm Using Frequent Character Count in Patterns

S.Nirmala Devi* *Research Scholar, Bharath University, India.*
Dr.S.P Rajagopalan *Dr.M.G.R University India.*
Dr.V.Anuradha *Dept. of Biotechnology Mohammed Satak College, India.*

Abstract— Pattern matching is one of the major issues in the area of computational biology and also in many other areas. Many databases like GenBank were built by researchers for DNA and protein sequences , the string matching problem is the core problem for searching these databases. There are several algorithms in use, in which Index Based Pattern matching using Multithreading yielding good results when compared to the some of the existing popular methods. Biologists search database for important information in different directions. Pattern matching will continue to grow and need changes from time to time. To further raise the performance of pattern matching algorithm An Index based Multiple Pattern Matching Algorithm using Frequent character count in patterns [IPMAFC] is presented. . The proposed method is used to avoid unnecessary comparisons in DNA and Protein sequence. These result in the reduction of the character comparison effort at each attempt. Proposed algorithm is implemented and compared with existing algorithms. Experimental results of applying the technique to DNA/proteins sequences show the effectiveness of the proposed technique.

Keywords— Pattern Matching Algorithms, Pattern Matching, DNA and Protein Sequences, comparison per character

I. INTRODUCTION

Pattern Matching is one of the basic and most important issues, which have been studied in the research area of computer science. Pattern Matching is defined as searching the occurrences of a particular pattern in a large sequence and it has many applications in the modern world which includes text editors, Network Intrusion Detection System, search engine, data mining, molecular medicine and Comparative biology , retrieval of information and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. The analysis of Protein and DNA sequence data has been one of the most active research areas in the field of computational molecular biology. Pattern Discovery is one of the fundamental problems in Bioinformatics. It can be used in Protein structure and function prediction, Multiple Sequence Alignment, Characterization of protein families and other areas. The determination of function and evolutionary relationships among sequences can be done through the discovery of Patterns or Motifs in Protein and Nucleotide sequences. The basic units of DNA are nucleotides and each nucleotide is one of the following four types: adenine (A), guanine (G), cytosine (C) and thymine (T). Proteins are made up of amino acids, which are twenty in count. The amino acid list starts as Alanine,Arginine,Asparagine Acid, etc., and they are denoted as single letter code – A,C,D,E,F,G,H, I, K, L, M, N, P, Q, R, S, T, V, W, Y. The DNA sequence can be viewed as long sequences of 4 Alphabets whereas Protein sequences can be viewed as long sequences of 20 Alphabets. . It is very difficult to retrieve necessary information from the sequence when the size of the database grows. The String Matching can be described as : Given a specific String T generally called Pattern P to search in a large sequence T to locate P in T. If P is in T, the matching is found and indicates the position of P in T, else Pattern is not found in T. Based on the number of Patterns to be matched , a pattern match problem can be classified into single-pattern and multiple-pattern match problems[1]. Single Pattern match algorithm or Single Keyword Match algorithm searches through the Text/Sequence to match only one given pattern at one scan, while the multiple pattern match algorithm searches through the text to match a given set of patterns at one scan.

Existing Pattern Matching algorithms are classified into two categories:

Exact Pattern Matching Algorithms

Inexact Pattern Matching Algorithms

Exact Pattern Matching will find that whether the probability will lead to either successful or unsuccessful search. The problem can be states as: Given a Pattern P of length m and Text/Sequence of length n. Find all the occurrences of P in T. The matching needs to be exact, that is exact pattern is found. More number of algorithms was created to perform string searching. Each algorithm uses a specific strategy to perform the search. Some need to pre-process the Text. Others need to pre-process the Pattern [2][3][4]. There are algorithms that require both the text and pattern to be pre-processed before searching [5] and some do not perform pre-processing of text and pattern. One of the simplest search algorithms is the Brute –Force Algorithm [15]. It is the least efficient way to check whether a pattern occurs in a String. In order to improve the Brute-Force Algorithm various string search algorithms were created. They are Knuth-Morris-Pratt [6], BoyerMoore [7] , and Karp and Rabin algorithms[8]. The application where the algorithm is to be applied determines

which of the algorithms the best is. Inexact or approximate pattern matching is the technique of finding strings that match a pattern approximately. It is defined as LOCATE, COUNT, or EXTRACT all occurrences of pattern P of length m in a text T of size n with at most k errors.

II. BACKGROUND AND RELATED WORK

This section reviews some work related to DNA and protein sequences .An alphabet set Σ = {A, C, G, T} is the set of characters for DNA sequence or Σ = { A,C,D,E,…} of 20 characters for Protein sequence are used in this algorithm.

Brute-force algorithm performs a checking, at all positions in the text between 0 and n-m, whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right. If the match is found then it returns other wise the matching process is continued by shifting one character to the right. If performs n comparisons if there is no match at all in the text for the given pattern. The Boyer-Moore Algorithm works with a backward approach, the target string is aligned with the start of the check string, and the last character of the target string is checked against the corresponding character in the check string. The DP algorithm [10] requires a pre-processing of the given text to prepare a table of the occurrences of the first and the last characters of the given pattern. This table is used to find the probability of having a match of the pattern in the given text, which reduces the number of comparisons, improving the performance of the pattern matching algorithm. The Index based Pattern Matching using Multithreading [11] method performs pre-processing to get the index of the first character of the pattern in the given text. By using this index as the starting point of matching, it compares the Text contents from the defined point with the pattern contents. In IBSPC [12] indexes has been used for the DNA sequence. After creating the index the algorithm will search for the pattern in the string using the index of least occurring character in the string. In IFBMPM[13] to search some pattern P in text S, it start the search from the indexes stored in the row of index table which corresponds to the first character of the pattern P. If any character mismatches in its position, we skip the search and go for the next index which corresponds to the first character of the pattern P according to the indexes stored in index table for matching.

III. AN INDEX BASED MULTIPLE PATTERN MATCHING ALGORITHM USING FREQUENT CHARACTER COUNT IN PATTERNS

The proposed method requires pre-processing phase and search phase. In the pre-processing phase the algorithm find the frequent character in the pattern. Based on that character an index table is created to store the index of the character present in the String/Sequence. In the search phase the frequent character of Pattern P is aligned with the same character in String S. Once the alignment is completed it computes the sum of ascii value of the characters of String S with the sum of the ascii value of Pattern P. If it matches, then it compares the individual characters of string with Pattern. Else it moves to the next index of that character in the String. The algorithm is suitable for DNA and Protein sequence comparison. It requires the pre-processing of the given text to prepare a table of the occurrences of the frequent character of the pattern. This table is used to find the probability of having a match of the pattern in the given text, which reduces the number of comparisons, improving the performance of the pattern matching algorithm.

A. Algorithm

Input : String S of n characters and a pattern P of m characters, where S, P belongs to DNA or protein Alphabet Set.

Output : The number of occurrences of Pattern in String , its location and the number of characters compared.

Step 1 : [Initialization of variables & Index array]
Integer count:=0 and ncmp:=0, StringIdx[n]

Step 2 : The frequently occurred character in pattern P is identified.
frechar=char(mode(pmatrix));

Step 3 : The frequent character of Pattern P is matched with the string and the corresponding index of that character is stored in a table StringIdx[];

```
for a=1:lenx
  if(x(a)~=frechar)
    continue;
  else
    myMatrix(b)=a;
    b=b+1;
  end
end
```

Step 4 : Based on the index value the pattern is aligned with the string .

```
Tstartidx=myMatrix(a) -patternidx;
  Tendidx=Tstartidx+leny-1;

   if Tendidx > lenx
    break;
  end
  outstr= x(Tstartidx : Tendidx);
  if length(outstr) < leny
    exit;
  end
```



```
Step 5          : compare the ascii sum of string and ascii sum of pattern.
if xasc==yasc
Step 6          : If they are equal compare individual characters of string and pattern
        for i=1:leny
           ncmp=ncmp+1;
        if(outstr(i)==y(i))
          found=1;
          continue;
        else
           break;
        end
        end
      if found==1
         count=count+1;
        fprintf('\n Pattern found at location %i ',Tstartidx);
      end
      end
     end
 end
 fprintf('\n Number of Patterns Found in Text is %i ',count);
fprintf('\n No. of characters compared %i ',ncmp);
```

B. Working Example.

1)Example for DNA sequence: To understand the proposed algorithm assume a string S= GCTCGATTTCGATGGCTCGAATCGATCTA of 29 DNA characters and P = TCGAT of 5 characters.

The frequently occurred character of Pattern P is identified. In this example the frequent character of pattern is T.

Based on the character T of Pattern, index table for String S is created to store the index. Table I shows the index table for character T of String S.

TABLE 1

INDEX TABLE FOR STRING S BASED ON FREQUENT CHARACTER OF PATTERN P

3	7	8	9	13	17	22	26	28

The algorithm maps to the first occurrence of T according to the table and aligns the pattern with string.

S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A

P = T C G A T

It finds the sum of Ascii values of substring S and compare with the Ascii value of Pattern.

Pattern Ascii sum is(TCGAT) 84+67+71+65+84=371

Sum of Ascii values of substring (TCGAT) S is 371.

If both the values are equal the algorithm then starts comparing the first element of pattern with the possible match in the string relative to T.

S = G C **T** C G A T T T C G A T G G C T C G A A T C G A T C T A

P = **T** C G A T

The first character matches then it compares the second character of the pattern.

S = G C **T C** G A T T T C G A T G G C T C G A A T C G A T C T A

P = **T C** G A T

The second character also matches then it compares the third character.

S = G C **T C G** A T T T C G A T G G C T C G A A T C G A T C T A

P = **T C G** A T

The third character also matches then it compares the fourth character.

S = G C **T C G A** T T T C G A T G G C T C G A A T C G A T C T A

P = **T C G A** T

The fourth character also matches then it compares the last character of pattern.

S = G C **T C G A T** T T C G A T G G C T C G A A T C G A T C T A

P = **T C G A T**

All the characters are matched, so the pattern is found at position 3.

Next the algorithm aligns the pattern with string based on the second index of T according to the index table.

S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A

P = T C G A T

Ascii sum of Pattern and Ascii sum of substring does not match. Now the pattern is aligned with the third index of T.

S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A

P = T C G A T

Ascii sum of Pattern and Ascii sum of substring does not match. Now the pattern is aligned with the fourth index of T.

S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A

P = T C G A T
Here Ascii sums are equal and then its starts to compare individual characters.
S = G C T C G A T T **T C G A T** G G C T C G A A T C G A T C T A
P = **T C G A T**
All the characters are matched, so the pattern is found at position 9.
Based on the fifth index of T pattern is aligned with String and it compares the ascii value.
S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A
P = T C G A T
The values doesn't match and the pattern is aligned based on the next index
S = G C T C G A T T T C G A T G G C T C G A A T C G A T C T A
P = T C G A T
The Ascii values are not equal and the pattern is aligned to the next index of T.
S = G C T C G A T T T C G A T G G C T C G A A **T C G A T** C T A
P = **T C G A T**
All the characters are matched, so the pattern is found at position 22.

Now the pattern is moved to the next index and the comparison does not occurs because the length of substring is less than the length of the pattern and the algorithm terminates. The Total Numbers of Patterns occurred in the String is 3.
By above example we can conclude that taking frequent count reduces the number of comparisons.

2) Experimental Results

In this section we present several experiments comparing our algorithm to the existing algorithms on different pattern sizes and the comparison results are noted.
The below DNA sequence dataset has been taken from IBSPC[12] for testing IPMAFC algorithm. The DNA biological sequence S of size n = 1024 and pattern P of different sizes are taken. Let S be the following DNA sequence.
AGAACGCAGAGACAAGGTTCTCATTGTGTCTCGCAATAGTGTTACCAACTCGGGTGCCTATTGGCCTCCAA
AAAAGGCTGTTCAACGCTCCAAGCTCGTGACCTCGTCACTACGACGGCGAGTAAGAACGCCGAGAAGGTA
AGGGAACTAATGACGCGTGGTGAATCCTATGGGTTAGGATCGTGTCTACCCCAAATTCTTAATAAAAAACC
TAGGACCCCCTTCGACCTAGACTATCGTATTATGGACAAGCTTTAACTGTCGTACTGTGGAGGCTTCAAAA
CGGAGGGACCAAAAAATTTGCTTCTAGCGTCAATGAAAAGAAGTCGGGTGTATGCCCCAATTCCTTGCTGC
CCGGACGGCCAGGCTTATGTACAATCCACGCGGTACTACATCTTGTCTCTTATGTAGGGTTCAGTTCTTCGC
GCAATCATAGCGGTACTTCATAATGGGACACAACGAATCGCGGCCGGATATCACATCTGCTCCTGTGATGG
AATTGCTGAATGCGCAGGTGTGAATACTGCGGCTCCATTCGTTTTGCCGTGTTGATCGGGAATGCACCTCG
GGGACTGTTCGATACGACCTGGGATTTGGCTATACTCCATTCCTCGCGAGTTTTCGATTGCTCATTAGGCTT
TGCGGTAAGTAAGTTCTGGCCACCCACTTCGAGAAGTGAATGGCTGGCTCCTGAGCGCGTCCTCCGTACAA
TGAAGACCGGTCTCGCGCTAAATTTCCCCCAGCTTGTACAATAGTCCAGTTTATTATCAAAGATGCGACAA
ATAAATTGATCAGCATAATCGAAGATTGCGGAGCATAAGTTTGGAAAACTGGGAGGTTGCCAGAAAACTC
CGCGCCT.

We have implemented and tested the proposed technique using MATLAB and the Experimental Results for the proposed algorithm for different pattern sizes which has been chosen randomly from the above DNA sequence , the number of occurrences, number of comparisons and the number of comparisons per character (i.e) Number of comparisons/DNA sequence size is shown in Table II.

3)Example for Protein Sequence

The below Protein Sequence dataset has taken from NCBI site
(http://www.ncbi.nlm.nih.gov/protein/269849759?report=fasta) Full Sequence in Fasta Format .
>gi|269849759|sp|P04637.4|P53_HUMAN RecName: Full=Cellular tumor antigen p53; AltName: Full=Antigen NY-CO-13; AltName: Full=Phosphoprotein p53; AltName: Full=Tumor suppressor p53
MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLSPDDIEQWFTEDPGPDEAPRMPEAAPPVAP
APAAPTPAAPAPAPSWPLSSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKMFCQLAKTCPVQLWVDS
TPPPGTRVRAMAIYKQSQHMTEVVRRCPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFRHSVVVPYEPP
EVGSDCTTIHYNYMCNSSCMGGMNRRPILTIITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKGEPHHE
LPPGSTKRALPNNTSSSPQPKKKPLDGEYFTLQIRGRERFEMFRELNEALELKDAQAGKEPGGSRAHSSHLKSKK
GQSTSRHKKLMFKTEGPDSD

TABLE II.
EXPERIMENTAL RESULTS OF IPMAFC FOR DNA SEQUENCE

Patterns (P)	Size of P	No. of occurrences	No. of comparisions	CPC
CAT	3	11	51	0.05
AACG	4	5	46	0.04
AAGAA	5	2	26	0.03
AAAAAA	6	3	18	0.02

AGAACGC	7	2	20	0.02
AAAAAAGG	8	1	15	0.01
GCTCATTAG	9	1	32	0.03
CCTTTTCCGG	10	1	18	0.02
TTTTGCCGTGT	11	1	15	0.01
TTCTTAATAAAA	12	1	15	0.01
GGGACCAAAAAAT	13	1	21	0.02
TTTTGCCGTGTTGA	14	1	18	0.02
CCTCCAAAAAAGGCT	15	1	19	0.02
GGCTGTTCAACGCTCC	16	1	23	0.02

Table III shows the experimental result of IPMAFC for the above sequence S of 393 protein characters and different pattern sizes . The number of occurrences, number of comparisons and the number of comparisons per character (i.e) Number of comparisons/Protein sequence size is shown in Table IV.

TABLE III
EXPERIMENTAL RESULTS OF IPMAFC FOR PROTEIN SEQUENCE

Patterns (P)	Size of P	No. of occurrences	No. of comparisions	CPC
PPPG	4	1	7	0.02
PAPAA	5	1	10	0.02
HHELPP	6	1	6	0.02
GTAKSVT	7	1	7	0.02
QETFSDLWKLLPENN	15	1	17	0.04

By using the proposed method different patterns are analysed and the graph is plotted by using these results. In molecular biology this type of large sequences are common to compare with other sequences. To check whether the given pattern presents in the sequences or not we need an efficient algorithm which searches in less time. There are many algorithms which perform the search and each have its advantages and disadvantages. This algorithm will decrease the number of comparisons. The results of IPMAFC for DNA Sequence with different pattern size were compared with Boyer Moore[14] ,Not So Naïve[14] ,Morris-Pratt[14],Quick Search[14],IBSPC and IFBMPM, and are shown in the table 4 and also plotted in the graph as shown in the Fig. 1. The proposed algorithm gives good performance in Number of comparisons and Character per comparison ratio compared with other algorithms. Fig. 1. shows the comparisons of different algorithms with the proposed technique. The current technique gives good performance in reducing the number of character comparisons compared with other popular methods and existing algorithms. The solid line shows the proposed method whereas the other methods are shown by dotted lines. Towards X-axis we have taken randomly different pattern sizes ranges from 4 to 16 whereas towards Y-axis shows the total number of comparisons.

TABLE IV
COMPARISONS OF DIFFERENT ALGORITHMS WITH IPMAFC

Pattern	No. of Occurenc	IPMAFC		IBSPC		IFBMPM		Boyer-Moore		Not So Naïve		Morris-Pratt		Quick Search	
		No. of Com	CPC	No. of Com	CPC	No. of Com	CPC	No. of Com	CPC	No. of Com	CPC	No. of Com	CPC	No. of Com	CPC
CAT	11	51	0.1	320	0	567	0.6	462	0.5	832	0.8	961	0.9	407	0.4
AACG	5	46	0	311	0	614	0.6	381	0.4	586	0.6	922	0.9	368	0.4
AGAACGC	2	20	0	368	0	600	0.6	247	0.2	822	0.8	951	0.9	274	0.3
GCTCATTAG	1	32	0	335	0	582	0.6	258	0.3	824	0.8	925	0.9	354	0.3
CCTTTTCCGG	1	18	0	347	0	562	0.5	187	0.2	581	0.6	967	0.9	203	0.2
TTCTTAATAAAA	1	15	0	332	0	651	0.6	126	0.1	588	0.6	960	0.9	143	0.1
TTTTGCCGTGTTGA	1	18	0	351	0	638	0.6	162	0.2	587	0.6	993	1	304	0.3
GGCTGTTCAACGCTCC	1	23	0	597	1	598	0.6	223	0.2	575	0.6	957	0.9	260	0.3

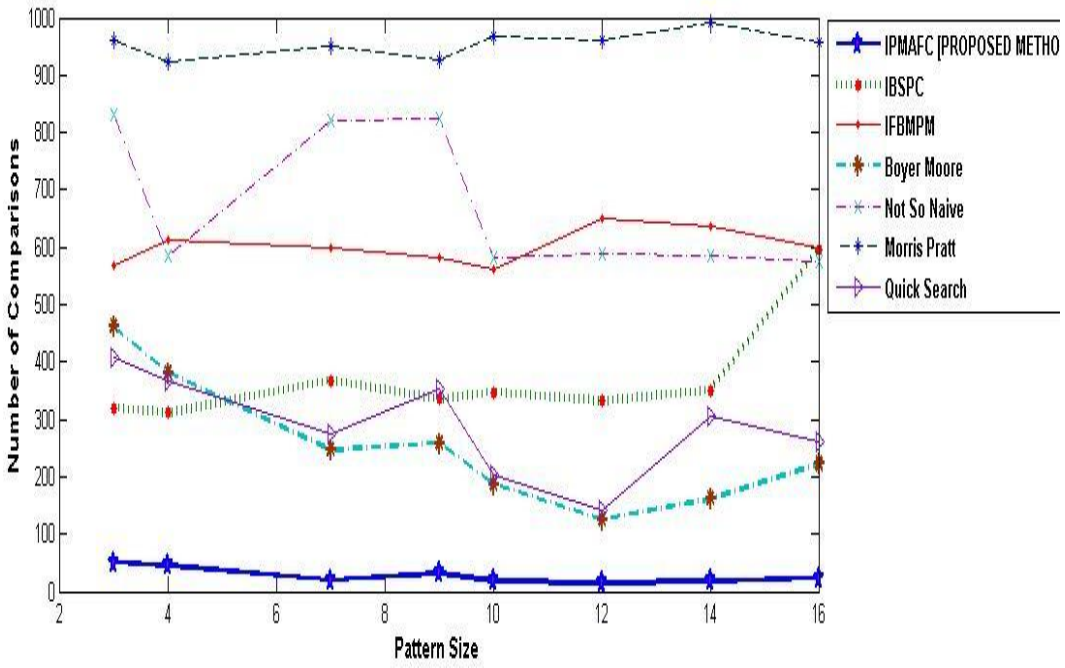

Fig 1. : Comparisons of Different Algorithms with IPMAFC

IV. CONCLUSIONS

We proposed a new algorithm with a simple logic which is very easy to implement. This method can be use for pattern matching in protein sequences and also for English Text. This method depends on the pre-processing phase which retrieves the index based on the frequent character of the pattern and it is suitable for unlimited size of input sequence. The proposed algorithm gives very good performance with the other algorithms. Based on the experimental work our approach provides good performance related to DNA and protein sequence dataset.

REFERENCES

[1] Computer Algorithms String Pattern Matching Strategies Jun-Iche Aoe.
[2] Z, Du X,and Ishii N., An improved adaptive string searching algorithm, Softare Practice and Experience, 1988, 28(2):191–198.
[3] Sunday D., A very fast substring search algorithm, Communications of the ACM, 1990, 33(8):132–142.
[4] Bruce W. and Watson E., A Boyer-Moore-style Algorithm for Regular Expression Pattern Matching, Science of Computer Programming, 2003, 48: 99-117.
[5] Fenwick P., Fast string matching for multiple searches, Software–Practice and Experience, 2001, 31(9):815–833.
[6] Knuth D., Morris.J Pratt.V Fast pattern matching in strings, SIAM journal on computing.
[7] BoyerMoore [7] Boyer R. ande Moore J., "A Fast String Searching Algorithm,"Computer Journals of Communications of the ACM, vol.20, no.10, pp.762-772, 1997.
[8] KARP, R., and M. RABIN. 1987. "Efficient Randomized Pattern-Matching Algorithms. IBM J Res. Development, 31, 249-60.
[9] http://en.wikipedia.org/wiki/Approximate_string_matching#ref_Zob95
[10] Devaki Pendlimarri et al. / (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 08, 2010, 2698-2704 .
[11] S.Nirmala Devi, Dr.S.P Rajagopalan 'An Index Based Pattern Matching using Multithreading' , International Journal of Computer Applications (0975 – 8887) Volume 50 – No.6, July 2012
[12] Raju Bhukya, DVLN Somayajulu," Index Multiple Pattern Matching Algorithm using DNA Sequence and Pattern Count", International Journal of Information Technology and Knowledge Management July-December 2011, Volume 4, No. 2, pp. 431-441
[13] Raju Bhukya, DVLN Somayajulu,"An Index Based Forward Backward Multiple Pattern Matching Algorithm", World Academy of Science and Technology. June 2010, pp347- 355
[14] http://www-igm.univ-mlv.fr/~lecroq/string/node3.html#SECTION0030
[15] Levitin. The Design and Analysis of Algorithms. Addison Wesley, 2003. p.97.

