International Journal of Information Technology and Knowledge Management
July-December 2011, Volume 4, No. 2, pp. 431-441

INDEX BASED MULTIPLE PATTERN MATCHING ALGORITHM USING DNA SEQUENCE AND PATTERN COUNT

Raju Bhukya[1] & DVLN Somayajulu[2]

Executing patterns from a large DNA sequence or protein data is a computationally intensive task. As performance plays a major role in extracting patterns from a given DNA sequence or from a large database independent of the size of the sequence, more efficient approaches are needed for multiple pattern matching techniques. One of the major problems in genomic field is to perform pattern comparison on DNA and protein sequences. In the current approach we explore a new technique which avoids unnecessary comparisons in the DNA sequence and gives the accurate retrieval of the pattern called index based multiple pattern matching algorithm using sequence and pattern count for DNA sequences. The proposed technique gives good performance related to DNA sequence analysis for querying of publicly available genome data. By using the proposed technique the number of comparisons gradually decreases and comparison per character ratio of the proposed algorithm reduces accordingly when compared to the some of the existing popular methods. The results show that there is considerable amount of performance improvement as a result overall performance increases.

Keywords: DNA Sequence, Index, Partition, Pattern Matching.

1. Introduction

The pattern matching problem has attracted a lot of interest throughout the history of computer science, particularly in the present day high performance computing and has used in various computer applications for several decades. These algorithms are applied in most of the operating systems, editors, search engines on the internet, retrieval of information and searching nucleotide or amino acid sequence patterns in genome and protein sequence databases. Bioinformatics is a multi disciplinary science that uses methods and principle from mathematics, computer science and statistics for analyzing biological data. Pattern matching plays a key role in various applications in computational biology for data analysis like feature extraction, searching, disease and structural analysis. The main aspects involved in pattern matching in a sequence is discrimination of diseases evaluated from the gene expression, Sub-cellular localization from experimental data through protein pattern matching. Pattern matching focuses on finding the occurrences of a particular pattern in a given text. The problem in pattern discovery is to determine how often a candidate pattern occurs, as well as possibly of some information on its frequency distribution across the sequence/text. In general, a pattern will be a description of a set of strings, each string being a sequence of symbols. Hence, given a pattern, it is usual to ask for its frequency, as well as to examine its occurrences in a given sequence/text. Many algorithms have been developed each designed for a specific type of search. Although they all serve the same function but they vary in the way they process the search, and second in the methods they use to efficiently achieve the optimal processing time.

Every human has his/her unique genes. Genes are made up of DNA and therefore the DNA sequence of each human is unique. However the DNA sequences of all humans are 99.9% identical, which means there is only 0.1% difference. DNA is contained in each living cell of an organism, and it is the carrier of that organism's genetic code. The genetic code is a set of sequences, which define what proteins to build within the organism. Since organisms must replicate and reproduce tissue for continued life, there should be some means of encoding the unique genetic code for the proteins. The genetic code is the information which will be needed for biological growth and reproductive inheritance. As DNA is the basic blue print of life it can be viewed as a long sequence over the four alphabets A, C, G and T. It contains genetic instructions of an organism and is mainly composed of nucleotides of four types. Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). The pattern itself may not be exactly known, because it may involve insertion, deletion, or replacement of the symbols. The amount of DNA extracted from the organism is increasing exponentially.

The sequence of DNA constitutes the heritable genetic information in nuclei, plasmids, mitochondria, and chloroplasts that forms the basis for the developmental programs of all living organisms. Determining the DNA sequence is therefore useful in basic research studying

[1]Assistant Professor, Department of CSE, National Institute of Technology,Warangal, A.P, India.
[2]Professor, Department of CSE, National Institute of Technology, Warangal, A.P, India.
E-mail: [1]brajuphd@gmail.com, [1]raju@nitw.ac.in, [2]soma@nitw.ac.in [2]somadvlns@gmail.com

fundamental biological processes, as well as in applied fields such as diagnostic or forensic research. Because DNA is key to all living organisms, knowledge of the DNA sequence may be useful in almost any biological subject area. In medicine it can be used to identify, diagnose and potentially develop treatments for genetic diseases. Similarly, genetic research into plant or animal pathogens may lead to treatments of various diseases caused by these pathogens. When we know a particular sequence is the cause for a disease, the trace of the sequence in the DNA and the number of occurrences of the sequence defines the intensity of the disease. As the DNA is a large database we need an efficient algorithms to find out a particular sequence in the given DNA. We have to find the number of repetitions and the start index and end index of the sequence, which can be used for the diagnosis of the disease and also the intensity of the disease by counting the number of pattern matching strings, occurred in a gene database.

The biologists often queries new discoveries against a collection of sequence databases such as GENBANK, EMBL and DDBJ to find the similarity sequences. As the size of the data grows it becomes more difficult for users to retrieve necessary information from the sequences. Hence more efficient and robust methods are needed for fast pattern matching techniques. The string matching can be described as given a specific strings P generally called pattern searching in a large sequence/text 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 does not occurs in the given text. Pattern matching techniques has two categories and is generally divides into single pattern matching and multiple pattern matching algorithms.

- Single pattern matching
- Multiple pattern matching techniques

In a standard problem, we are required to find all occurrences of the pattern in the given input text, known as single pattern matching. Suppose, if more than one pattern are matched against the given input text simultaneously, then it is known as, multiple pattern matching. Single pattern matching algorithm is widely used in network security environments. In network security the pattern is a string indicating a network intrusion, attack, virus, and snort, spam or dirty network information, etc. Multiple pattern matching can search multiple patterns in a text at the same time. It has a high performance and good practicability, and is more useful than the single pattern matching algorithms. To determine the function of specific genes, scientists have learned to read the sequence of nucleotides comprising a DNA sequence in a process called DNA sequencing. DNA comparison, pattern recognition, similarity detection and phylogenetic trees construction in genome sequences are the most popular tasks. The process of sequence alignment allows the insertion, deletion and replacements of symbols that representing the nucleotides or amino acids sequences. From the biological point of view pattern comparison is motivated by the fact that all living organisms are related by evolution. This implies that the genes of species that are closer to each other should show signs of similarities at the DNA level. Moreover, those similarities also extend to gene function. Normally, when a new DNA or protein sequence is determined, it would be compared to all known sequences in the annotated databases such as GenBank, SwissProt and EMBL.

Let $P = \{p_1, p_2, p_3,.., p_m\}$ be a set of patterns of m characters and $T=\{t = t_1,t_2,t_3...t_n\}$ in a text of n characters which are strings of nucleotide sequence characters from a fixed alphabet set called $\Sigma = \{A, C, G, T\}$. Let T be a large text consisting of characters in Σ. In other words T is an element of Σ^*. The problem is to find all the occurrences of pattern P in text T. Many existing pattern matching algorithms are reviewed and classified in two categories.

- Exact string matching algorithm
- Inexact/approximate string matching algorithms

Exact pattern matching algorithm will find that whether the probability will lead to either successful or unsuccessful search. The problem can be stated as: Given a pattern p of length m and a string/Text T of length n($m \leq n$). Find all the occurrences of p in T. The matching needs to be exact, which means that the exact word or pattern is found. Some exact matching algorithms are Naïve Brute force algorithm, Boyer-Moore algorithm [3], KMP Algorithm [7].

Inexact/Approximate pattern matching is sometimes referred as approximate pattern matching or matches with k mismatches/differences. This problem in general can be stated as: Given a pattern P of length m and string/text T of length n. ($m \leq n$). Find all the occurrences of sub string X in T that are similar to P, allowing a limited number, say k different characters in similar matches. The Edit/ transformation operations are insertion, deletion and substitution. Inexact/Approximate string matching algorithms are classified into: Dynamic programming approach, Automata approach, Bit-parallelism approach, Filtering and Automation Algorithms. Inexact sequence data arises in various fields and applications such as computational biology, signal processing and text processing. Pattern matching algorithms have two main objectives.

- Reduce the number of character comparisons required in the worst and average case analysis.
- Reducing the time requirement in the worst and average case analysis.

In many cases most of the algorithm operates in two stages. Depending upon the algorithm some of the algorithm uses pre-processing phase and some algorithm will search

without it. Many Pattern matching algorithms are available with their own merits and demerits based upon the pattern length and the technique they use. Some pattern matching algorithm concentrates on pattern itself. Other algorithm compare the corresponding characters of the patterns and text from the left to right and some other perform the character from the right to left. The performance of the algorithm can be measured based upon the specific order they are compared. Pattern matching algorithms has two different phases.

- Pre-processing phase or study of the pattern.
- Processing phase or searching phase.

The pre-processing phase collects the full information and is used to optimize the number of comparisons. Whereas searching phase finds the pattern by the information collected in pre-processing. Pattern analysis plays a major part for various analysis like discrimination of the cancer from gene expression, mutation evolution, data analysis, feature extraction, searching, disease analysis, structural and functional analysis, e-books, text processing, linguistic translation, data compression, search engine, speech reorganization, information retrieval, genomic data, protein-protein interaction in cellular activities, computer virus detection, network intrusion detection, parsers, spam filters, digital libraries, screen scrapers, word processors, natural language processing and computational biology.

The rest of the paper is organized as follows. We briefly present the background and related work in section 2. Section 3 deals with proposed model i.e., IBSPC algorithm for DNA sequence. Results and discussion are presented in Section 4 and we make some concluding remarks in Section 5.

2. BACKGROUND AND RELATED WORK

This section reviews some work related to DNA sequences. An alphabet set $\Sigma = \{A, C, G, T\}$ is the set of characters for DNA sequence which are used in this algorithm. The following notations are used in this paper:

DNA sequence characters $\Sigma = \{A, C, G, T\}$.

ϕ Denotes the empty string.

$|P|$ Denotes the length of the string P.

S[n] Denotes that a text which is a string of length n.

P[m] Denotes a pattern of length m.

CPC-Character per comparison ratio.

String matching mainly deals with problem of finding all occurrences of a string in a given text. In most of the DNA applications it is necessary for the user and the developer to be able to locate the occurrences of specific pattern in a sequence. In Brute-force algorithm the first character of the pattern P is compared with the first character of the string T. If it matches, then pattern P and string T are matched character by character until a mismatch is found or the end of the pattern P is detected. If mismatch is found, the pattern P is shifted one character to the right and the process continues. The complexity of this algorithm is O(mn). The Bayer-Moore algorithm [3] applies larger shift-increment for each mismatch detection. The main difference the Naïve algorithm had is the matching of pattern P in string T is done from right to left i.e., after aligning P and string T the last character of P will matched to the first of T. If a mismatch is detected, say C in T is not in P then P is shifted right so that C is aligned with the right most occurrence of C in P. The worst case complexity of this algorithm is O(m+n) and the average case complexity is O(n/m). In IFBMPMA [12] the elements in the given patterns are matched one by one in the forward and backward until a mismatch occurs or a complete pattern matches. The KMP algorithm [7] is based on the finite state machine automation. The pattern P is pre-processed to create a finite state machine M that accepts the transition. The finite state machine is usually represented as the transition table. The complexity of the algorithm for the average and the worst case performance is O(m+n).

In IBKPMPM [13] algorithm we first choose the value of k (a fixed value), and divide both the string and pattern into number of substring of length k, each substring is called as a partition. If k value is 3 we call it as 3-partition else if it is 4 then it is 4-partition algorithm. We compare all the first characters of all the partitions, if all the characters are matching while we are searching then we go for the second character match and the process continues till the mismatch occurs or total pattern is matched with the sequence. If all the characters match then the pattern occurs in the sequence and prints the starting index of the pattern or if any character mismatches then we will stop searching and then go to the next index stored in the index table of the same row which corresponds to the first character of the pattern P. In approximate pattern matching method the oldest and most commonly used approach is dynamic programming. In 1996 Kurtz [8] proposed another way to reduce the space requirements of almost O(mn). The idea was to build only the states and transitions which are actually reached in the processing of the text. The automaton starts at just one state and transitions are built as they are needed. The transitions those were not necessary will not be build.

The Deviki-Paul algorithm [5] for multiple pattern matching requires a pre-processing of the given input text to prepare a table of the occurrences of the 256 member ASCII character set. This table is used to find the probability of having a match of the pattern in the given input text, which reduces the number of comparisons, improving the performance of the pattern matching algorithm.

In the MSMPMA [17] technique the algorithm scans the input file to find the all occurrences of the pattern based upon the skip technique. By using this index as the starting point of matching, it compares the file contents from the defined point with the pattern contents, and finds the skip value depending upon the match numbers (ranges from 1 to m – 1). Harspool [6] does not use the good suffix function, instead it uses the bad character shift with right most character .The time complexity of the algorithm is O(mn).

Berry-Ravindran [2] calculates the shift value based on the bad character shift for two consecutive text characters in the text immediately to the right of the window. This will reduce the number of comparisons in the searching phase. The time complexity of the algorithm is O(nm). Sunday [4] designed an algorithm quick search which scans the character of the window in any order and computes its shift with the occurrence shift of the character T immediately after the right end of the window.

The FC-RJ [11] algorithm searches the whole text string for the first character of the pattern and maintains an occurrence list by storing the index of the corresponding character. It uses an array equal to size of the text string for maintaining occurrence list. Time and space complexity of pre-processing is O(n). Ukkonen [15] proposed automation method for finding approximate patterns in strings. He proposed the idea using a DFA for solving the inexact matching problem. Though automata approach doesn't offer time advantage over Boyer-Moore algorithm[3] for exact pattern matching. The complexity of this algorithm in worst and average case is O(m + n). In this every row denotes number of errors and column represents matching a pattern prefix. Deterministic automata approach exhibits O(n) worst case time complexity. The main difficulty with this approach is construction of the DFA from NFA which takes exponential time and space. Wu.S.Manber.U[16] proposed the algorithm for fast text searching allowing errors. The first bit-parallel method is known as "shift-or" which searches a pattern in a text by parallelizing operation of non deterministic finite automation. This automation has m + 1 states and can be simulated in its non deterministic form in O(mn) time. The filtering approach was started in 1990. This approach is based upon the fact it may be much easier to tell that a text position doesn't match. It is used to discard large areas of text that cannot contain a match. The advantage in this approach is the potential for algorithms that do not inspect all text characters.

By using dynamic programming approach especially in DNA sequencing Needleman-Wunsch [9] algorithm and Smith-waterman algorithms [14] are more complex in finding exact pattern matching algorithm. By this method the worst case complexity is O(mn). The major advantage of this method is flexibility in adapting to different edit distance functions. The Raita algorithm [10] utilizes the same approach as Horspool algorithm [6] to obtaining the shift value after an attempt. Instead of comparing each character in the pattern with the sliding window from right to left, the order of comparison in Raita algorithm [10] is carried out by first comparing the rightmost and leftmost characters of the pattern with the sliding window. If they both match, the remaining characters are compared from the right to the left. Intuitively, the initial resemblance can be established by comparing the last and the first characters of the pattern and the sliding window. Therefore, it is anticipated to further decrease the unnecessary comparisons.

The Aho-Corasick algorithm [1] developed at Bell Labs in 1975 by Alfred Aho and Corasick is an extension of the KMP algorithm [7]. The AC algorithm consists of constructing a finite state pattern matching machine from the keyword and then using the machine to process the text in a single pass. It can find an occurrence of several patterns in the order of O(n) time, where n is the length of the text, with pre-processing of the patterns in linear time.

Two dimensional pattern matching methods are commonly used in computer graphics. Takaoka and Zhu proposed using a combination of the KMP[7] and RK methods in an algorithm developed for two dimensional cases. The second approach that runs faster when the row length of the pattern increases and is significantly faster than previous methods proposed. Three dimensional pattern matching is useful in solving protein structures, retinal scans, finger printing, music, OCR and continuous speech. Multi-dimensional matching algorithms are a natural progression of string matching algorithms toward multi-dimensional matching patterns including tree structure, graphs, pictures, and proteins structures.

3. INDEX BASED MULTIPLE PATTERNS MATCHING ALGORITHM USING DNA SEQUENCE AND PATTERN COUNT

Basically IFBMPM[12] checks for each first occurrence of the first character of pattern in the algorithm and compares one character from left and one character from right until all characters are compared, if all characters matches to the pattern then it prints the starting index of the sequence. If any character mismatches it skips the test and continues to check the next occurrence of the first character of the pattern available in the index table. This process continues till the two pointers from left as well as right pointer exchanges their position in opposite direction. In the current algorithm an extra field called count has been added to the IFBMPM [12] which is used to reduce the number of comparisons and CPC ratio. The count has been added to the sequence as well as the pattern. The count value always increments as it scans the DNA sequence and patterns for the characters A, C, G and T. So the count value will be indicating the least count which is present either in A, C, G or T in the DNA sequence. The index values of the character

which is having the least count in the sequence gives the possible places where initial alignment will be done. The least count character should be present in the pattern for alignment. If it is not there then we select the next least frequent character in a decreasing order. Now we align the pattern with sequence at all the occurrences of this character one by one for pattern matching. To match the pattern we make comparisons in decreasing order of count in the pattern. Here we are taking pattern count as the maximum count. As the pattern index table and the sequence index table are built separately for the comparison one needs both the table values for the initial comparison.

The proposed algorithm uses the indexes for the DNA sequence and the pattern to be searched. It scans the DNA sequence from left to right and creates a table of indexes in an increasing order as they appear in the sequence. Similarly an index table is formed for the pattern. It has to search a pattern in a string whose alphabet set $\Sigma = \{A,C,G,T\}$. Let the string be S of n characters represents the DNA sequence and pattern P of m characters is to be searched in string S. It also maintains an array which stores the frequency of each alphabet in Σ^*. After creating the index the algorithm searches for the pattern in the string using the alphabet in P with least count value. Index for pattern is also maintained with their frequencies. The characters with highest count value in pattern are compared first followed by other characters in decreasing order of count. The time complexity of the IBSPC is O(mn).

3.1 Algorithm

Input: String S of n characters and a pattern P of m characters, where S, $P \in \Sigma^*$.

Output: The no. of occurrence and the positions of P in DNA.

Step 1: Integer arrays stab[4][n], ptab[4][n], sidx[4], pidx[4], ssort[4] = {0, 1, 2, 3}, psort = {0, 1, 2, 3}

Integer found:=1, ncmp:= 0, npat:= 0

Step2: FOR i := 0; i < n; i++

stab[(S[i] <<5)>>6][sidx[(S[i]<<5)>>6]++] = i;

END FOR

Step 3: Sort array ssort in ascending order according to values in sidx

Step 4: FOR i := 0; i < n; i++

stab[(P[i] <<5)>>6][pidx[(P[i]<<5)>>6]++] = i;

END FOR

Step 5: Sort array psort in descending order according to values in pidx

Step 6: WHILE pidx[ssort[k++]] = 0 && k < 4

DO

END DO

y:=ssort[k - 1]//y stores the least occurring character in the string which is also part of pattern

Step 7: Store the index of first occurrence of y in the pattern P in dif

Step 8: FOR i := 0; i <=sidx[y]; i++

found = 1;

k = stab[y][i] - dif;

x = k;

b = 0, c = 0;

FOR j = 0; j<= m; j++

If (pidx[psort[b]] == c)

b++, c = 0, j--;

continue;

END IF

ncmp++;

a = ptab[psort[b]][b];

IF((S[k+a] <<5)>>6!= psort[b])

found = 0;

break;

END IF

c++;

END FOR

IF found = 1

npat++;

PRINT "Pattern Found at location x occurrence no is : npat "

END IF

END FOR

The algorithm takes a string as an input, and for given pattern it checks whether the pattern occurs in the Sequence/ text or not. If the pattern occurs in the string it prints the found pattern with its starting index of the string. It first builds up the table called sequence index table (stab[4][n]), which is useful to reduce the number of comparisons. It also stores the count/occurrence of each character found in the sequence, in sidx[4]. The character in pattern which occur least number of times in the sequence is used to further

reduce the number of comparisons. The index based algorithm for multiple pattern matching uses a table called stab[4][n]. This table stores all the indexes of each character in its corresponding vector. It also builds up a table called pattern index table (ptab[4][m]) for storing the indexes for pattern to be searched. After reaching a possible position of pattern in the DNA sequence by using stab in the algorithm. Then we use ptab to first compare the alphabets with maximum count in ptab and then the other alphabets in decreasing order. The algorithm is suitable for DNA pattern matching because the numbers of possible character are less. The characters available in the DNA sequence are Σ = {A, C, G, T}. For each character in Σ we compute its array subscript value in stab by using the following technique.

Table 1
Subscript Values for the DNA Characters

DNA	Binary representation	(binary<<5)	((Binary<<5) >>6	Array Subscript
A	01000001	00100000	00000000	0
C	01000011	01100000	00000001	1
G	01000111	11100000	00000011	3
T	01010100	10000000	00000010	2

In this algorithm [(S[i]<<5)>>6] always returns a subscript value in the range 0, 1, 2, 3 which is needed for subscripting 2D vector of size [4][n]. The subscript values 0, 1, 2, 3 represents characters A, C, T, G respectively. So for each character of string in the function [(S[i]<<5)>>6] directly references to its corresponding position in the 2D vector stab[4][n]. The vector sidx[4] stores the count of each character in the string. Now it uses the character which has least count to find the pattern in the sequence. This will reduce the number of comparisons needed to find the occurrences of the pattern in the given string.

3.2 Trivial Cases in Comparisons

Case i: If S = ϕ i.e., |S| = 0 and P = ϕ i.e.,|P| = 0 then the number of occurrences of P in S is 0.

Case ii: If S = ϕ i.e. |S| = 0 and for any |P| e $\geq$ 0 then the number of occurrences of P in S is 0.

Case iii: If S $\neq \phi$ i.e., |S| $\neq$ 0 and for any |P| = 0 then the number of occurrences of P in S is 0.

Case iv: If S $\neq \phi$ i.e., |S| $\neq$ 0, P $\neq \phi$ i.e.,|P| $\neq$ 0 and |S| $\leq$ |P| then the number of occurrences of P in S is 0.

3.3. This Section Describes Different Examples Using Proposed Approach IBSPC for the DNA Sequences.

Take a string S=GCGTCTCGGACGGTCACGTCAAA AATGGAACTACAACGGT of DNA sequence and P = ACGGT of 5 characters. The following index table (stab) stores all the indexes of each character A,C,G and T in its corresponding row. The 0th row stores the indexes of occurrences of the character A, 1st row for C, 2nd row for T and 3rd row for G. We store the total no of occurrences of each character in separate array (sidx). The comparisons will start from the character in pattern P which is occurring least number of times in the sequence S. For above example the element T, A, G and C are occurring 7, 11, 11, 11 times respectively. Here T is occurring least number of times, so T is used as the initial alignment and if there is a match of pattern T with the sequence character T then rest of the characters are compared in descending order of their frequencies in ptab[4][m] for the pattern matching. We first align T of pattern with the T of the sequence S then we will compare all the other characters. In Pattern P the character G is occurring maximum number of times so we will be comparing all G's first after alignment followed by other characters in decreasing order of their frequencies in ptab.

Table 2
Index Values For A,C,G and T of Sequence Data

DNA	Sequence Indexes											Count
A0	9	15	20	21	22	24	28	29	32	34	35	11
C1	1	4	6	10	13	14	16	19	30	33	36	11
T2	3	5	13	18	25	31	39					7
G3	0	2	7	8	11	12	17	26	27	37	38	11

The count helps us to find the least count character which is available from the sequence as well as the pattern table. So here we directly map the minimum count character so only those many comparisons can be done. So the number of comparisons gradually reduced. The algorithm goes to the first occurrence of T according to the table and then starts comparing using ptab to compare according to frequency of characters in the pattern i.e..first G then A,C and T. Let us take an example and analyze how it actually works. Go to the first occurrence of T using stab then start matching all G's. Also store the first occurrence of T i.e., 4 in variable dif. This is used in alignment to find the starting position of the pattern in sequence.

Table 3
Index Values for A,C,G and T of Pattern

DNA	Pattern Indexes		Count
A	0		1
C	1		1
T	4		1
G	2	3	2

Initially in sequence index table T is minimum. So go to first occurrence of T i.e., 3 now align the pattern with sequence at T's position. For comparison start with G

because it is having maximum count value in ptab. Next comparisons will be made in the decreasing order of count from the pattern index table. This process continues until all the indexes are checked.

S = GCGTCTCGGACGGTCACGTCAAAA ATGGAAC TACAACGGT

P = ACGGT

The first G doesn't matches so we move to next occurrence of T i.e., 5 and align the pattern with the sequence at this position i.e., 5 – 4 = 1 here 4 is value of difference. So 1 is possible starting position of pattern P in sequence S. Now start comparing with G.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAA C TACAACGGT

P = ACGGT

Again the first G doesn't matches so move to next occurrence of T i.e.,13 and align the pattern with the sequence at this position i.e., 13 – 4 = 9. Now again start comparing with G.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAAC TACAACGGT

P = ACGGT

The first character matches so compare next G of patter with sequence.

S = GCGTCTCGGACGGTCACGTCAAAAATGGA A CTACAACGGT

P = ACGGT

Next G is also matched now move to start matching next most frequent character in ptab i.e., A.

S = GCGTCTCGGACGGTCACGTCAAAAATGGA ACTACAACGGT

P = ACGGT

A is matched so we start matching C.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAA CTACAACGGT

P =ACGGT

C is also matched so now start comparing the last T.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAA CTACAACGGT

P = ACGGT

T also matches so all character matches and pattern is found at position 9. Now move to next occurrence of T i.e.,18 and align the pattern with the string 18 – 4 = 14 and start comparing all G's.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAAC TACAACGGT

P = ACGGT

The character doesn't match so move to next occurrence of T i.e., 25 and align the pattern at this position i.e., 25 – 4 = 21 and start comparing all G's.

S = GCGTCTCGGACGGTCACGTCAAAAATGGA ACTACAACGGT

P = ACGGT

The character doesn't match so move to next occurrence of T i.e., 31 and start comparing all G's.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAA CTACAACGGT

P = ACGGT

The character doesn't match so move to next occurrence of T i.e., 39 and align the pattern with the string 39 – 4 = 35 and start comparing all G's.

S = GCGTCTCGGACGGTCACGTCAAAAATGGA A CTACAACGGT

P = ACGGT

The first character matches so compare next character G of pattern with the sequence.

S = GCGTCTCGGACGGTCACGTCAAAA A T G G A ACTACAACGGT

P = ACGGT

Next G also matches so we move to match next most frequent character A.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAAC TACAACGGT

P = ACGGT

A is also matched now compare the next character C.

S = GCGTCTCGGACGGTCACGTCAAAAATGGAAC TACAACGGT

P = ACGGT

C too matches so now compare the final character T.

S = GCGTCTCGGACGTCACGTCAAAAATGGAA CTACAACGGT

P = ACGGT

Character T also matches with the sequence so the pattern is found at position 35. So total number of patterns found are 2 from the DNA sequence S.

4. EXPERIMENTAL RESULTS AND DISCUSSIONS

The below DNA sequence dataset has been taken for the testing of IBSPC algorithm. The DNA biological sequence $S \in \Sigma^*$ of size $n = 1024$ and pattern $P \in \Sigma^*$. Let S be the following DNA sequence.

AGAACGCAGAGACAAGGTTCTCATTGTGTCTCG
CAATAGTGTTACCAACTCGGGTGCCTATTGGCCTCC
AAAAAAGGCTGTTCAACGCTCCAAGCTCGTGACCTC
GTCACTACGACGGCGAGTAAGAACGCCGAGAAGGT
AAGGGAACTAATGACGCGTGGTGAATCCTATGGGTT
AGGATCGTGTCTACCCCAAATTCTTAATAAAAAACC
TAGGACCCCCTTCGACCTAGACTATCGTATTATG GA
CAAGCTTTAACTGTCGTACTGTGGAGGCTTCAAAAC
GGAGGGACCAAAAAATTTGCTTCTAGCGTCAATGAA
AAGAAGTCGGGTGTATGCCCCAATTCCTTGCTGCCC
GGACGGCCAGGCTTATGTACAATCCACGCGGTACTA
CATCTTGTCTCTTATGTAGGGTTCAGTTCTTCGCGC
AATCATAGCGGTACTTCATAATGGGACACAACGAAT
CGCGGCCGGATATCACATCTGCTCCTGTGATGGAAT
TGCTGAATGCGCAGGTGTGAATACTGCGGCTCCATT
CGTTTTGCCGTGTTGATCGGGAATGCACCTCGGbGG
ACTGTTCGATACGACCTGGGATTTGGCTATACTCCA
TTCCTCGCGAGTTTTCGATTGCTCATTAGGCTTTGC
GGTAAGTAAGTTCTGGCCACCCACTTCGAGAAGTGA
ATGGCTGGCTCCTGAGCGCGTCCTCCGTACAATGAA
GACCGGTCTCGCGCTAAATTTCCCCCAGCTTGTACA
ATAGTCCAGTTTATTATCAAAGATGCGACAAATAAA
TTGATCAGCATAATCGAAGATTGCGGAGCATAAGTT
TGGAAAACTGGGAGGTTGCCAGAAAACTCCGCGCCT
ACTTTCGTCAGGATGATTAAGAGTATCGAGGCCCCG
CCGTCAATACCGATGTTCTTCGAGCGAATAAGTACT
GCTATTTTGCAGACCCTTTGCCAGGCCTTGTCTAA
AGGTATGTTACTTAATATTGACAATACATGCGTAT
GGCCTTTTCCGGTTAACTCCCTG.

The index table (index Tab[4][1024]) for sequence S is very large. For different patterns P's the number of occurrences and the number of comparisons is shown in the below Table.4. To check whether the given pattern presents in the sequence or not we need an efficient algorithm with less comparison time and complexity. By the current technique different patterns are analyzed and the graph is plotted by using these results. From the below experimental results, improvement has be seen that IBSPC algorithm gives good performance compared to the some of the popular methods. Different pattern sizes has been taken from the DNA sequence ranging from 1 to 20 randomly and tested by 1024 DNA characters. CPC ratio decreases and is less than 1 in case of the proposed technique where as in other cases CPC is more than 1 in some of the existing methods which is shown in Table.5. From the Table.4 we can see the following fields for comparison of different algorithms like pattern text, number of characters in the pattern, number of occurrences of a pattern, the proposed method and the number of comparisons with comparisons per character. The number of comparisons per character (CPC ratio) which is equal to (Number of comparisons/file size) can be used as a measurement factor, this factor affects the complexity time, and when it is decreased the complicity also decreases.

Table 4
Experimental Results of IBSPC Algorithm with Other Algorithms

S. No.	Pattern(P's)	Noof Char	No. of Occu	IFBMPM	CPC	IBKMPM	CPC	IBSPC	CPC Ratio
1	A	1	259	518	0.50	259	0.25	259	0.25
2	AG	2	53	624	0.60	518	0.50	300	0.29
3	CAT	3	11	567	0.55	542	0.52	320	0.31
4	AACG	4	5	614	0.59	614	0.59	311	0.30
5	AAGAA	5	2	616	0.60	607	0.59	340	0.33
6	AAAAAA	6	3	627	0.61	620	0.60	344	0.33
7	AGAACGC	7	2	600	0.58	613	0.59	368	0.35
8	AAAAAAGG	8	1	634	0.61	623	0.60	346	0.33
9	GCTCATTAG	9	1	582	0.56	590	0.57	335	0.32
10	CCTTTTCCGG	10	1	562	0.54	578	0.56	347	0.33
11	TTTTGCCGTGT	11	1	650	0.63	650	0.63	361	0.35
12	TTCTTAATAAAA	12	1	651	0.63	634	0.61	332	0.32
13	GGGACCAAAAAAT	13	1	579	0.56	582	0.56	350	0.34
14	TTTTGCCGTGTTGA	14	1	638	0.62	654	0.63	351	0.34
15	CCTCCAAAAAAGGCT	15	1	578	0.56	558	0.54	354	0.34
16	GGCTGTTCAACGCTCC	16	1	598	0.58	580	0.56	597	0.58
17	TTTTCGATTGCTCATTA	17	1	643	0.62	633	0.61	359	0.35
18	GGGATTTGGCTATACTCC	18	1	598	0.58	580	0.56	355	0.34
19	GGCCTTGTCTAAAGGTATG	19	1	579	0.56	585	0.57	344	0.33
20	CCTGAGCGCGTCCTCCGTAC	20	1	570	0.55	582	0.56	590	0.57

From Table.5.observations has been made for the following in terms of relative performance of our algorithm with the existing algorithms. The proposed algorithm gives good performance in two parameters like CPC ratio and number of comparisons with the algorithms like MSMPMA, Brute-force, Tri-Match, Naïve string matching, IFBMPM and IBKPMPM algorithms. It is clear that proposed (IBSPC) algorithm outperforms when compared with all other algorithms. From the below Table the total number of occurrences of each pattern, the number of comparison and CPC ratio of each algorithm is given. In each of algorithm we have two fields i,e., number of comparisons and CPC ratio. In some of the cases as the size of the pattern increases the number of comparisons and Comparison per character decreases in case of proposed method. In case of proposed the number of comparisons decreases and is less than half when compared with the existing algorithms. The current technique gives good performance in reducing the number of comparisons compared with other algorithms.

Table 5
Comparisons of Different Algorithms

Pattern	No. of occu	IBSPC		IBKMPM		IFBMPM		MSMPMA		BruteForce		TriMatch		Naïvestring	
		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
A	259	259	0.2	259	0.2	518	0.5	1024	1.0	1024	1.0	1025	1.0	1024	1.0
AG	53	300	0.2	518	0.5	624	0.6	1230	1.2	1282	1.2	1284	1.2	1281	1.2
CAT	11	320	0.3	542	0.5	567	0.5	1298	1.2	1318	1.2	1321	1.2	1310	1.2
AACG	5	311	0.3	614	0.6	614	0.5	1359	1.3	1376	1.3	1380	1.3	1376	1.3
AAGAA	2	340	0.3	607	0.5	616	0.6	1375	1.3	1388	1.3	1393	1.3	1387	1.3
AAAAAAGG	1	346	0.3	623	0.6	634	0.6	1394	1.3	1409	1.3	1417	1.3	1407	1.3
TTCTTAATAAAA	1	332	0.3	634	0.6	651	0.6	1390	1.3	1390	1.3	1402	1.3	1399	1.3
GGCTGTTCAACGCTCC	1	597	0.5	580	0.5	598	0.5	1349	1.3	1349	1.3	1365	1.3	1349	1.3

Fig 1. Shows the graphical comparison of different algorithms with IBSPC. Towards X-axis we have taken different pattern sizes ranging from 1 to 16 in size and towards Y-axis we have taken number of comparisons. The algorithms like MSMPMA, Brute force, tri-match, naïve string search shows more than 1000 comparisons for the same set of input patterns where as the proposed algorithm shows the total number of comparisons between 200 to 597 only from the below graph. It is clear that proposed (IBSPC) algorithm outperforms when compared with all other algorithms. The current technique gives good performance in reducing the number of comparisons compared with other algorithms. The dotted line shows the IBSPC model where as MSMPMA, Brute-Force, Tri-matching, Naïve string searching, IFBMPM and IBKPMPM is shown by solid lines. To compare the proposed algorithm with the existing algorithms we have taken certain parameters like pattern size, number of comparisons and CPC ratio. The CPC ratio almost is less than 1 in all the cases of the proposed technique where as more than one in all the other techniques. So the proposed technique shows efficient than some of the popular techniques which is shown in the below graph. In some of the cases as we increase the size of the pattern the number of comparisons increases where as in our case the number of comparisons decreases.

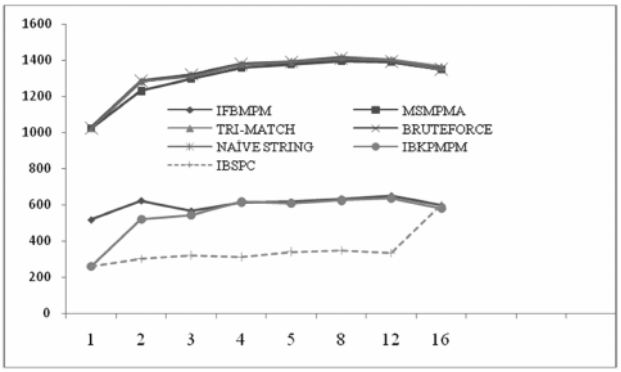

Fig.1: Comparison Graph of Different Algorithms with IBSPC.

The following are observed from the experimental results

- Reduction in number of comparisons.
- The ratio of comparisons per character has gradually reduced and is less than 1.
- Suitable for unlimited size of the input file.
- Once the indexes are created for input sequence we need not create them again.

- For each pattern we start our algorithm from the matching character of the pattern which decreases the unnecessary comparisons of other characters.
- It gives good performance for DNA related sequence applications.

Some of the Applications Related to the Bioinformatics

Different biological problems of bioinformatics involve the study of genes, proteins, nucleic acid structure prediction, and molecular design.

- Alignment and comparison of DNA, RNA, and protein sequences.
- Gene mapping on chromosomes.
- Gene finding and promoter identification from DNA sequences.
- Interpretation of gene expression and micro-array data.
- Gene regulatory network identification.
- Construction of phylogenetic trees for studying evolutionary relationship.
- DNA and RNA structure prediction.
- Protein structure prediction and classification.
- Organize data in a way that allows researchers to access existing information and submit new meaningful entries.
- Develop tools and resources which are used for analysis and management of biological data.
- Use sequence data to analyze and interpret the results in a biologically meaningful manner.
- To help researchers in the pharmaceutical industry in drug design process.
- Finding similarities among strings such as proteins of different organisms.
- Detecting certain patterns within strings.
- Finding similarities among parts of spatial structures.
- Constructing of phylogenetic trees called the evolution of organisms.
- Classifying new data according to previously clustered sets of annotated data.

5. CONCLUSION

We have presented a new model, which is simple technique and yet effective for biological pattern matching algorithm. In this study a new algorithm for improving the performance of indexed multiple pattern matching using sequence and pattern count is proposed. Comparison of proposed algorithm is made with existing algorithms on the basis of the number of comparisons and the attempts made by different pattern sizes of different algorithms to complete the task. Out algorithm outperform in case of number of comparisons related to the DNA sequences. The proposed model which is shown to be very efficient and fast. The analysis illustrate that the IBSPC algorithm is better than the number of existing algorithms. Based on the experimental work carried out with DNA sequence data set, IBSPC approach gives the better performance.

REFERENCES

[1] Aho, A. V., and M. J. Corasick, ‘‘Efficient String Matching: An Aid to Bibliographic Search”, Communications of the ACM (June 1975), pp. 333 340.

[2] Berry, T. and S. Ravindran, 1999. “A Fast String Matching Algorithm and Experimental Results”, In: Proceedings of the Prague Stringology Club Workshop ’99, Liverpool John Moores University, pp: 16-28.

[3] Boyer R. S., and J. S. Moore, A Fast String Searching Algorithm‘Communications of the ACM, 20, 762- 772, 1977.

[4] D.M. Sunday, “A Very Fast Substring Search Algorithm”, Comm. ACM 33 (8) (1990) 132–142.

[5] Devaki-Paul, “Novel Devaki-Paul Algorithm for Multiple Pattern Matching”, International Journal of Computer Applications (0975 – 8887) 13– No.3, January 2011.

[6] Horspool, R.N., 1980. “Practical Fast Searching in Strings”, Software Practice Experience, 10:501-506

[7] Knuth D., Morris. J Pratt. V “Fast Pattern Matching in Strings”, SIAM Journal on Computing, Vol 6(1), 323-350, 1977.

[8] Kurtz. S, “Approximate String Searching Under Weighted Edit Distance”, In proceedings of the 3rd South American Workshop on String Processing, Carleton Univ Press, pp. 156-170, 1996

[9] Needleman, S.B Wunsch, C.D(1970). “A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of two Proteins.” J.Mol.Biol.48,443-453.

[10] Raita, T,“Tuning the Boyer-Moore-Horspool string-Searching Algorithm, Software Practice Experience 1992, 22(10), 879-884.

[11] Rami H. Mansi, and Jehad Q. Odeh, “On Improving the Naive String Matching Algorithm,” Asian Journal of Information Technology, 8, No. I, ISS N 1682-3915,2009, pp. 14-23.

[12] Raju Bhukya, DVLN Somayajulu,“An Index Based Forward Backward Multiple Pattern Matching Algorithm”, World Academy of Science and Technology. June 2010, pp347-355

[13] Raju Bhukya, DVLN Somayajulu, "An Index Based K-Partition Multiple Pattern Matching Algorithm", Proc. of International Conference on Advances in Computer Science 2010, pp 83-87.

[14] Smith,T.F and waterman, M (1981). Identification of Common Molecular Subsequences T.mol.Biol.147,195-197.

[15] Ukkonen,E., Finding Approximate Patterns in Strings, J.Algor. 6, 1985, 132-137.

[16] Wu S., and U. Manber, ''Agrep — A Fast Approximate Pattern-Matching Tool,'' Unix Winter 1992 Technical Conference, San Francisco (January 1992), pp. 153 162.

[17] Ziad A.A Alqadi, Musbah Aqel & Ibrahiem "M.M.EI Emary, Multiple Skip Multiple Pattern Matching Algorithms", I AENG International Vol 34(2),2007.