Discovering sequence motifs of different patterns parallel using DNA operations

B.Lavanya
Department of Computer Science
University of Madras
Chennai, Tamil Nadu, India.

A. Murugan
Department of Computer Science,
Dr.Ambedkar Government College,
Chennai, Tamil Nadu, India.

ABSTRACT
Discovery of motifs in biological sequences and various types of subsequences in commercial databases have varied applications and interpretations. This paper proposes a new approach to solve the Combinatorial Pattern Matching (CPM), search for continuous and gapped rigid subsequences and discover Longest Common Rigid Subsequences (LCRS) from the given sequences using DNA operations and modified Position Weight Matrix (PWM). The algorithm and its variations have been tested with both real and simulated databases. The proposed work can be applied to genetic, scientific as well as commercial databases. Implementation results shown the correctness of the algorithms. Finally, the validity of the algorithms are checked and their time complexity is analyzed.

General Terms
Pattern recognition, Sequence mining, Data mining.

Keywords
DNA operations, Motifs, LCRS, CPM, PWM, Molecular Computing.

1. INTRODUCTION

One of the problem arising in the analysis of biological sequences is the discovery of sequence similarity of various kinds, in the primary structure of related proteins or genes. The process of discovery of patterns in the genetic data proves to be essential in many biological researches and interpretations. Firstly, the nature of identifying patterns varies with applications, it can be the subsequences from a large sequence or more number of sequences, patterns with misplaced gaps, patterns with rigid continuous sequences or rigid gapped sequences, and identifying the common rigid pattern from large number of sequences. Secondly, the concern is about the quality of identified patterns and time taken to discover them plays a vital role in huge researches. These two prime issues motivates the proposed work. The task of discovering frequent subsequences as patterns in a sequence database is done in [1, 7, 18, 22, 36]. Such frequent subsequences usually corresponds to residues conserved during evolution due to important structural or functional behavior. The problem addressed by the previous studies was to sought a minimum-cost consensus sequence that highlights the regions of similarity among the input sequences. Several methods have been proposed for dealing with this problem like [14, 19, 21, 27, 34, 35]. A detailed survey of several multiple-string alignment algorithms can be found in [8]. They encountered many notable problems like,the task of optimally aligning a set of strings is computationally very expensive [33] and they could only align the global similarities [26]. If the sequences under comparison are distantly related or if the relative order of their similar regions varies among sequences, it is quite possible that no substantial alignment can be produced. To overcome the difficulty of alignment problem a modified Position Weight Matrices (PWM) [5] can be used to focus on the positions of the patterns in the sequences. Various ways of building a PWM have been carried out, some of them are found in [6, 10, 25, 29, 30]. Building a modified PWM is given in [5]. A number of pattern discovery algorithms have been steadily appearing in the literature [3, 4, 5, 9, 13, 16, 20, 24, 27, 28, 31, 33]. We note that solving the Longest Common Subsequence problem (LCS), Generalized Centre String (GCS) and the Closest Substring Problem (CSP), are generalizations of the trivial longest common substring problem and were found to be NP hard [11, 12, 15], the variation of which is discovery of Longest Common Rigid Subsequences (LCRS), and it is proved to be MAX SNP-hard [3].

For huge databases, storing and retrieving of data is computationally expensive and time consuming, but with DNA strands and DNA operations [17], the storage and retrieval are done parallely, thus reducing the time complexity. Extracting such sequences and subsequences from a database of sequences [23], is an important data mining task with plenty of different application domains, such as bioinformatics, web usage mining, mobility data analysis, motif discovery, commercial database analysis, program execution traces, search for sequences of words in a text, DNA and protein sequence extraction. Motif discovery in sequences, typically involves the discovery of binding sites, conserved domains or otherwise discriminatory subsequences. In bioinformatics, the two predominant applications of motif discovery are sequence analysis and micro array data analysis. The majority of the tools can be found at the extreme ends of the spectrum with tools that exhaustively enumerate regular expressions at one end and probabilistic tools, based on Position Weight Matrices (PWMs), at the other. This partitioning of tools is due to a computational trade-off: more descriptive motif representations such as PWMs frequently make exhaustive searches computationally infeasible[10]. The definition of the search problem, especially the formulation of objective functions, leaves space for substantial improvement in the performance of the motif discovery tool [32].

2. LITERATURE REVIEW

Our work is a variant of sequential pattern mining, first introduced by Agarwal and Srikant [22] and further studied by many, with different methods proposed, like SPAM by ayres et

al [2]. There are studies on mining only representative patterns, such as closed sequential patterns by Yan et al [36]. However, different from ours, sequential pattern mining ignores the (possible frequent) repetitions of patterns within a sequence. The support of a pattern is the number of sequences containing the given pattern and its commonality between various other sequences.

Simulation of all the DNA operations are done in [17], the proposed work uses the DNA operations *cut* and *pcr* operations [17]. Mining GCS, using DNA operations and modified PWM, given a sequential database is performed in [16]. In DNA sequence mining, Zhang et al [18] introduce gap requirement, in mining periodic patterns from sequences. In particular, all the occurrences (both overlapping ones and non overlapping ones) of a pattern in a sequence satisfying the gap requirement are captured, and the support is the total number of such occurrences [5] . Compared to GCS and LCS, the LCRS is more restricted in the sense that the common subsequence now must appear in every sequence with the same shape. If compared with CSP, on one hand LCRS is more generalized because a shape needs to be computed, on the other hand LCRS is more restricted because it does not allow errors.

3. DNA – BASED PATTERN DISCOVERY

In this paper, we propose a new approach to study the continuous and gapped subsequences mining problem. The Algorithms 1,2 and 3 searches for all different patterns of any length, with the positions of all their continuous, gapped rigid instances, Combinatorial Pattern Matching (CPM) and LCRS in input sequences, using Position Weight Matrices (PWM).

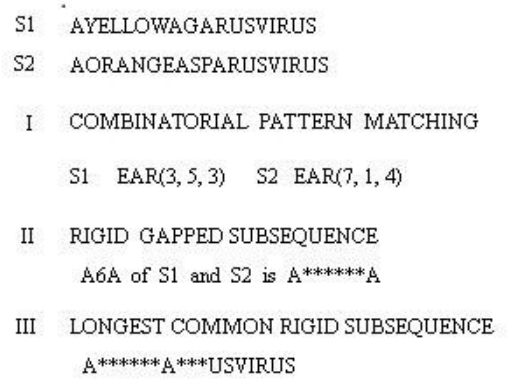

Figure 1: Example for CPM, Rigid gapped subsequence and LCRS

CPM means that the subsequence is checked for its existence in the given pattern in the database. Rigid gapped subsequences means a subsequence, which appears in all the sequences in the database, possibly with gaps between two successive events. Continuous subsequence means that the subsequence exists in the same pattern in all the sequences in the database without any gap and LCRS is the common rigid pattern existing in the same shape, in all the sequences in the database as shown in Figure 1.
Our approach makes minimal assumptions about the background sequence model and the mechanism by which elements affect gene expression. This provides a versatile motif discovery method, across all data types and genomes, with exceptional sensitivity and near-zero false-positive rates. Our algorithm does not use any complex statistical models but rather uses DNA operations and DNA strands to search for the presence or absence of a given motif in a regulatory region and the expression of the corresponding gene. The exponential nature of some PWM problems, is a limiting factor for using matrices of medium or large length. Here, we use DNA strands to store large data and DNA operations to access them parallely [5, 16] thus solving the above noted problem.

Lemma 1: Let T be set of alphabets. Let S = $\{s_1, s_2, s_3, \ldots s_n\}$, where $s_i \in T$ and $s_i \geq 2$ and any representative pattern R, $R \notin T$. Then $R \notin S$.

Proof: Let K be any set of alphabets, where $K \neq T$ and $R \in K$ then $R \notin T$. Thus $R \notin S$.

Lemma 1 shows the negative existence of the given pattern R in any of the sequences in S. For example for DNA sequences let T = {A,T,C,G}, s_1 ={ATCGATA}, s_2 = {CGATCCG}, s_3 = {AATTCGCGA} and R = {ATB}. Here the pattern $R \notin T$, thus proves the negative existence of R in S.

3.1 Combinatorial Pattern Matching

The Algorithm 1 can be used to solve Combinatorial Pattern Matching problem, that is, to find motifs in the given pattern from the given sequence, in the database and also checks for negative existence of the given pattern.

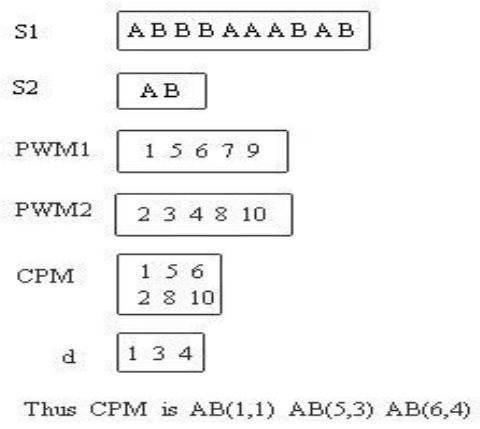

Figure 2: Combinatorial Pattern Matching

Inputs for Algorithm 1 are S1 (the DNA strand which contains the encoded sequence, for example A can be encoded as AT and B can be encoded as CG) and S2 (the subsequence for which the positions are to be searched for in S1). The algorithm outputs the Combinatorial Pattern Matching (CPM) strand containing the pattern to be searched and the occurrences, that is, the positions at which the elements of S2 are found in S1.

Algorithm 1: Combinatorial Pattern Matching
Input: S1, S2
Output: CPM strand

1 *begin*
2 let L ← length(S2) ;

```
3     foreach element of S2 do
4         Create threads for each element of S2 parallely ;
5       foreach thread do
6           let S3 ← pcr(S1) ;
7          PWM1[]…PWML[] ← cut(S3, S2[element1…L]);
8        end
9     end
10  Check for occurrence of S2 such that S2[1] <
                    S2[2] < ... S2[L] in PWM;
11  foreach PWM1[i1] PWM2[i2] PWM3[i3] … PWML[iL] do
12  foreach (i1, i2, i3,… iL) = 1 to |PWMi1, i2, i3,… iL |;
13          if PWM1[i1] < PWM2[i2] ... < PWML[iL] then
14                  j + +;
15      d[1, 2,…,L] ← PWM1[i1]- PWM2[i2] ... - PWML[iL];
16                  CPM[0…L][j] ← S2[i1, i2,…, iL] [d] ;
17                  i2 + +,…, iL + + ;
18          end
19      i1 + +; j ← -1 ;
20    end
21 end
```

Step 2 gets the length, L, of S2, that is, the number of elements in the subsequence. Steps 3-9 performs the task of *pcr* and *cut* operations, for each of the element in S2. A thread is generated for every element of S2 in parallel manner. S1 is multiplied and stored in S3, the cut operation is performed on S3 for each of the given element of S2 and the positions are stored in the respective PWM strands, that is, PWM_1, PWM_2, PWM_3, … PWM_L, one for each element, thus L number of PWM strands. In steps 11-19 the entries in the PWM strands are checked for the order of presence of, elements of S2, with respect to the positions in which they appear in S1. The entries in PWM are checked for the occurrences, such that the position of S2[1] < position of S2[2] etc till S2[L], and CPM strand is generated with the elements of S2 and the positions of their occurrences, as shown in Figure 2. The variable d gives the distance of the occurrence of S2[2] from S2[1] and so on, for every occurrence of the subsequence. The Algorithm 1 can also be used to find the non existence of S2 in S1. From step 12 of Algorithm 1, if any of |PWM| = 0, the Algorithm 1 identifies the non existence of S2 in S1.

Lemma 2: Let S1 ∈ T be a sequence, and subsequence S2 ∉ T; then S2 ∉ S1.

Proof: From lemma 1, if S2 ∉ T , then its PWM ∈ ∅ and so S2 ∉ S1.

Lemma 3: Let n = |S1|, where n > 0 and m = |S2| where 0≤m≤ n. Then |CPM| ≤ n/m.

Proof: If S2 occurs in equal probability in S1, where n = |S1| and m =|S2| then $|PWM_1| == |PWM_2| == |PWM_m|$ therefore |CPM| = n/m, otherwise |CPM| < n/m .

From Figure 2, n = 10, m = 2, if all elements of S2, occur with equal probability in S1, then |CPM| is 5 that is ≤ n/m.

3.1.1 Time Complexity

Let n be |S1|. Time complexity for pcr and cut operations, is O(1) at its best case, and O(n/M) at its average case [17].

TC(Algorithm1) = max(O(max(PCR,CUT)), O(CPM))

If PWM ∉ ∅, then

TC(CPM) = (n/L) iff |CPM| = n/m

TC(CPM) = max(|PWM_i|) where 1≤ i≤ L iff |CPM| < (n/m)

Therefore at best case

TC(Algorithm1) = O(n/L) to O(max |PWM_i|))

At average case

TC(Algorithm1) = (O(n/M) + O(n/L)) to

(O(n/M) + O(max |PWM_i|))

If PWM ∈ ∅, i.e, for the negative existence of the given subsequence, then

TC(Algorithm1) = O(PCR + CUT) i.e O(n/M) at the average case.

3.2 Exploring Rigid Subsequences

The Algorithm 2 can be used to find continuous and gapped rigid motifs from the given sequences and also checks for non existence of given rigid subsequence. Rigid gapped subsequences means a subsequence, which appears in all the sequences in the database, possibly with gaps between two successive events and continuous subsequence means that the subsequence exists in the same pattern in all the sequences in the database without any gap.

Inputs for Algorithm 2 are n, the number of sequences, S, the set of sequences (s_1, s_2,..., s_n), sub (the subsequence for which the positions are to be searched for in S) and dist is the rigid pattern. The algorithm outputs the RIGID strand. Step 3 gets the length, L, of sub, that is the number of elements in the subsequence. Steps 4-13 performs the task of pcr and cut operations, for each of the element in sub. A thread is generated for every element of sub in parallel manner. s_1, s_2,..., s_n are multiplied and stored in T_1, T_2, ..., T_n respectively. The cut operation is performed on T_1, T_2, ..., T_n for all the given sub[element] and the positions are stored in the respective PWM strands, that is, PWM_{11}, PWM_{12}, $PWM_{13,...}$ PWM_{1L}, one for each element, thus L number of PWM strands, for each sequence. Steps 16-27 generates the sequential strands parallely, one for each element of sub. The entries in the PWM strands are checked for the order of presence of elements of sub, with respect to the positions in which they appear in S. The entries in PWM are checked for the occurrences, such that the position of PWM_{11}[i] == position of PWM_{21}[j] etc till PWM_{L1}[k], where i, j, k ranges from 1 to max(PWM_{11}, PWM_{21} … PWM_{L1}) or until any of the PWM strand is empty and seq1[] … seqL[] strands are generated, that is L number of strands, one for each element of sub. Steps 28-33 are used to generate the RIGID strand, by checking the existence of elements of sub with the required rigid pattern given in dist, as shown in Figure 3.

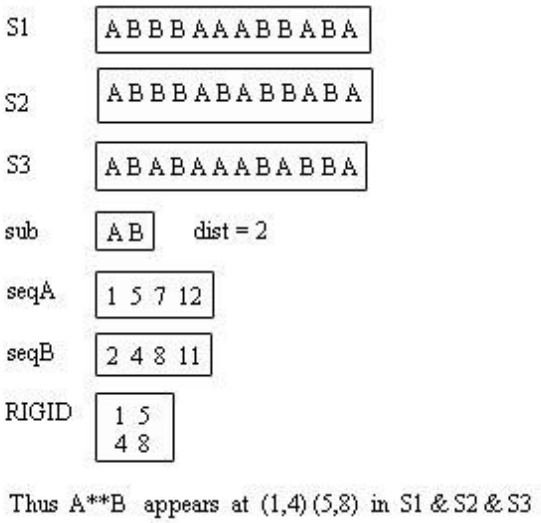

Figure 3: Rigid Gapped Subsequence

3.2.1 Special Case: Finding Rigid continuous and CPM from multiple sequences

For dist > 0, the Algorithm 2 works to find rigid gapped sequences, as explained above. If dist = 0, the Algorithm 2 works to find rigid continuous subsequences, and can be extended to find CPM, from more than one sequences with their repetitions and positions of occurrences, as shown in Figure 4. It can also be used to detect, the non existence of sub in S. From step 18 if any of the PWM strand is empty, the Algorithm 2 concludes the non existence of sub in S.

Lemma 4 : Let m = |min(S)| and m > 0 and n = |sub| where 0 < n < m. Then |seq| ≤ m/n.

Proof: If sub occurs in equal probability in S, where m = |min(S)|, then $|PWM_1| == |PWM_2| == |PWM_n|$ therefore |seq| = m/n, otherwise |seq| < m/n.

From Figure 3, m = 12, n = 2, if all elements of sub, occur with equal probability in the given pattern in S, then the size of seq strand is 6 that is ≤ m/n.

Lemma 5: Let S be a set of sequences where S Ɛ T and S = $\{s_1, s_2, s_3, \ldots, s_n\}$, x = |min(S) | and any representative pattern R, where R ∈ T, and y = |R|, where 0≤ y ≤ x. Then TC(R⊆ S)≤ x/y

Proof: Let rigid be the output strand. If R ∈ T, and if the elements of R occur at equal probability in S, then TC(R⊆ S)≤ x/y.

For example, from Figure 4, x = 12 and y = 2. If R occurs in equal probability then |rigid| = 6, else ≤ 6, that is |rigid| ≤ x/y.

Algorithm 2: Rigid Subsequence
Input: S, n, sub, dist
Output: RIGID strand

```
1   begin
2    let S ← s1, s2,…, sn;
3    let L ←  length(sub);
4   foreach element of sub do
5    Create threads for sub[1...L] parallely ;
6    foreach thread do
7       let T1 ←  pcr(s1) ;
8       let T2  ← pcr(s2) ;
9       let Tn ←  pcr(sn) ;
10      PWM11[]…PWM1L[]←  cut(T1, sub[1…L]) ;
11      PWM21[]…PWM2L[] ← cut(T2, sub[1…L]) ;
12      PWMn1[]…PWMnL[] ← cut(Tn, sub[1…L]) ;
13    end
14   end
15  Generate seq strands for each element of sub parallely ;
16  Generate threads for each element of sub parallely;
17  for PWM11[i1]  PWM21[i2]  PWMn1[in] …
                    PWM1L[i1]  PWM2L[i2]  PWMnL[in] ;
18     foreach thread do
19        for(i1, i2, i3,…, iL) = 1 to |PWM11| ;
20        for(j1, j2, j3,…, jL) = 1 to |PWM21| ;
21           if PWM11[i1] == PWM21[j1] then
22              for(k1, k2, k3,…, kL) = 1 to |PWM31|;
23                 if PWM11[i1] == PWM31[k1] then
24                    seq1[]…seqL[]← PWM11[i1] ;
25                 end
26              end
27           end
28     foreach seq1[]…seqL[]∉ ∅ do
29       if (seq1[] - seq2[]-…- seqL[] == dist) then
30          RIGID[1…L][] ← seq1…L[sub[element]] ;
31       end
32     end
33 end
```

3.2.2 Time Complexity

Let TC(Algorithm2) = max(O(max(PCR,CUT)), O(RIGID)).

If PWM ∉ ∅

TC(RIGID) = 2(n/L) iff |RIGID| = x/y

TC(RIGID) = (max ($|PWM_i|$) + max ($|seq_i|$))

where 1≤ i≤ L iff |RIGID| < (x/y).

Therefore at its best case

TC(Algorithm2) = O(2(n/L)) to (O(max $|PWM_i|$) + max($|seq_i|$))

At its average case

TC(Algorithm2) = ((O(n/M) + O(2(n/L))) to (O(n/M) + O(max$|PWM_i|$) + max($|seq_i|$))

If PWM ∈ ∅, i.e, for the negative existence of the given subsequence,

TC(Algorithm2) is O(PCR+CUT) that is O(n/M), at its average.

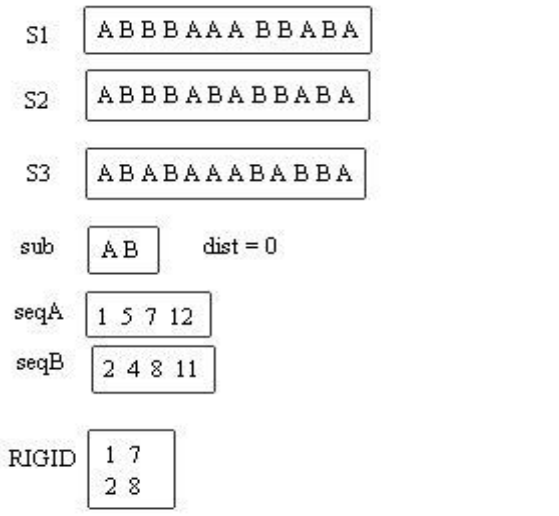

Figure 4: Rigid Continuous Subsequence

3.3 Longest Common Rigid Subsequences

The Algorithm 3 can be used to discover LCRS. LCRS is the common rigid pattern existing in the same shape, in all the sequences in the database as shown in Figure 5.

Inputs for Algorithm 3 are n, the number of sequences, S, the set of sequences (s_1, s_2,..., s_n), and ele (the number of common elements in database, for example AT and CG in DNA sequence). The algorithm outputs the LCRS strand. Steps 3-15 performs the tasks similar to Algorithm 2. Steps 17-28 generates the sequential seq strands parallely, one for each element of ele. The entries in the PWM strands are checked for the order of presence of elements of ele, with respect to the positions in which they appear in S. The entries in PWM are vertically checked for the occurrences, such that the position of $PWM_{11}[i]$ == position of $PWM_{21}[j]$ etc till $PWM_{L1}[k]$, and seq1[] … seqL[] strands are generated for the elements of ele, thus generating L number of strands. In steps 29-40, the LCRS strand is generated, by checking for each of the position of the sequences until the size, for its commonality of ele[elements]. The uncommon positions are marked with " * ", as shown in Figure 5.

Lemma 6: Let T be set of alphabets. Let S be set of sequences where $S = \{s_1, s_2, s_3, \ldots, s_n\}$, where $s_i \in T$ and threshold k, where $0 < k < n$. Let lcrs1 = LCRS(S), where lcrs1 $\subset$ S and lcrs2 = LCRS(k), where lcrs2 $\subset$ any k sequences $\in$ S. Then lcrs1 $\cap$ lcrs2 $\in \emptyset$.

Proof: If $s_1 \neq s_2 \ldots \neq s_n$ then $|PWM_{11}| \neq |PWM_{21}| \ldots \neq |PWM_{n1}|$. Similarly for other s_i, PWM strands also. So |seq| differ for every s_i. If lcrs1 = LCRS(S), where lcrs1 $\subset$ S and lcrs2 = LCRS(k), where lcrs2 $\subset$ any k sequences $\in$ S. Then lcrs1 $\cap$ lcrs2 $\in \emptyset$.

Algorithm 3: LCRS

```
Input: S, n, ele
Output: LCRS strand
1  begin
2    let S← s1,s2,…, sn;
3    let L← length(ele);
4    let size ← max(length(S));
5    foreach element of ele do
6     Create threads for each element of ele parallely ;
7     foreach thread do
8        let T1 ← pcr(s1) ;
9        let T2← pcr(s2) ;
10       let Tn← pcr(sn) ;
11       PWM11[]…PWM1L[]← cut(T1, sub[1…L]) ;
12      PWM21[]…PWM2L[]← cut(T2, sub[1…L]) ;
13      PWMn1[]…PWMnL[]← cut(Tn, sub[1…L]) ;
14     end
15   end
16   Generate seq strands for each element of ele parallely ;
17   Generate threads for each element of ele parallely;
18   for PWM11[i1] PWM21[i2] PWMn1[in] …
                    PWM1L[i1] PWM2L[i2] PWMnL[in] ;
19     foreach thread do
20      for(i1, i2, i3,…,iL) = 1 to |PWM11|
21      for(j1, j2, j3,…, jL) = 1 to |PWM21| ;
22        if PWM11[i1] == PWM21[j1] then
23          for(k1, k2, k3,…, kL) = 1 to |PWM31|;
24            if PWM11[i1] == PWM31[k1] then
25              seq1[]…seqL[] ← PWM11[i1];
26            end
27          end
28        end
29     foreach element of ele[1…L] do
30        for (i = 1 to size and seq1…L[]∉∅ );
31          if (seq1[j] == i) then
32             LCRS[i] ← ele[1] break;
33          end
34          if (seq2[j] == i) then
35            LCRS[i] ← ele[2] break;
36          end
37          Extended for L seq strands ;
38          LCRS[i] ← *;
39        end
40 end
```

Algorithm 3 finds the LCRS present in all the sequences in the database. This algorithm can also be extended to find LCRS, only for the specified number of sequences (k) in the database. For example, in Figure 5, if k = 2, the LCRS for S1 and S2 is ABBBA*ABBABA, for S1 and S3 is AB*BAAAB**BA and for S2 and S3 is AB*BA*AB**BA. Thus LCRS of k sequences are independent of LCRS of n sequences.

3.3.1 Special Case: Finding Diverging Pattern

Algorithm 3 can be extended to find diverging pattern with its position of divergence in the given sequences. From Figure 5 the divergent pattern of S is 3,6,9 and 10.

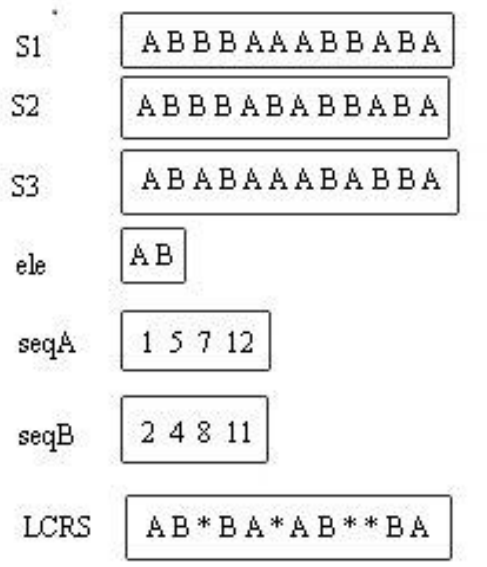

Figure 5: Longest Common Rigid Subsequence

3.3.2 Time Complexity

TC(Algorithm3) = max(O(max(PCR,CUT)),O(LCRS)).

If PWM $\notin \emptyset$, then
TC(LCRS) = (n/L) + n, iff |LCRS| = x/y
TC(LCRS) = (max($|PWM_i|$)+n) where $1 \leq i \leq L$
iff |LCRS| < (x/y).
Therefore at its best case
TC(Algorithm3) = O((n/L) + n) to (O(max$|PWM_i|$) + n)
At its average case
TC(Algorithm3) = ((O(n/M)+O((n/L)+n)) to
(O(n/M) + O(max$|PWM_i|$) + n)

If PWM $\in \emptyset$, i.e, for the negative existence of the given subsequence
TC(Algorithm3) = O(PCR + CUT) i.e O(n/M) at its average case.

4. PERFORMANCE

Algorithms 1,2,3 have been implemented and tested with both simulated and real databases. The random DNA sequences of size varying from 100 to 25000, are generated from http://old:dnalc:org/bioinformatics/dnalcnulceotideanalyzer:htmr andmizer and http://old:dnalc:org/bioinformatics:org/sms/rand dna:html. The real data is collected from EMBL database in FASTA format. The genome sequences of 3021 viruses are collected and tested for the existence of all required patterns. The database is got from http://www:ebi:ac:uk/genomes/ virus:html. Algorithms 1,2,3 proved to be efficient and accurate in solving CPM, continuous and rigid patterns and discovering LCRS in the given sequences. Tested with both randomly generated and real motifs our work could discover all motifs present, with its positions of existence, in the required patterns. All implementations are performed on a dual core computer and 5 GB main memory using Java language. The operating system is Windows XP. The resulted data of these experiments are consistent. The limitation of this algorithm is that the maximum number of threads generated is dependent on the efficiency of the system architecture.

5. APPLICATIONS

The assumption behind the discovery of patterns is that a pattern that appears often enough in a set of biological sequences is expected to play a role in defining the respective sequences functional behavior and evolutionary relationships. Since the proposed approach uses DNA strands for its DNA operations and other processing, the storage and retrieval processes can be implemented easily and parallely, whatever may be the size of the database. Since the applications for searching, the existence of subsequences from a large database of commercial or genetic information are unlimited, the searching for LCRS has its importance in many industrial, research and scientific applications. Especially in medical and genetic field, the finding of all patterns of motifs with its diverging pattern, can be used to predict, analysis, interpret and conclude the existence or future liability of any disease or abnormality present in the patient data or defaulters in any commercial databases. This work can also be applied to analysis of rule based systems, expert systems, rule mining, pattern mining, program execution traces, algorithm behavioral patterns and credit card data analysis.

6. CONCLUSION

In this paper, we have designed and performed the implementation to solve the problem in a highly parallel way, for finding all patterns, and can be extended to many other data mining applications also. In future, it is possible to solve more real time problems in molecular biology.

7. REFERENCES

[1] H.M. Annila, H.Toivonen, and A.I.Verkamo.1997,Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1(3):259-289.

[2] J. Ayres, J.Flannick, J.Gehrke, and T.Yiu.,2002, Sequential pattern mining using a bitmap representation. Int. Conf. on Knowledge Discovery and Data Mining, pages 429-435.

[3] Nikhil Bansal, Moshe Lewenstein, Bin MA, and Kaishong Zhang,2010, On the longest common rigid subsequence problem. Algorithmica, 56:270-280.

[4] G. Benson and M.S. Waterman,1994, A method for fast database search for all k-nucleotide repeats. 2nd International conference on Intelligent Systems for Molecular Biology, pages 83-98.

[5] B.Lavanya and A.Murugan,2011, A DNA based approach to find closed repetitive gapped subsequence from a sequence database. International Journal of Computer Applications,29(5),sep, pages 45-49.

[6] Isabelle da Piedade, Man-Hung Eric Tang, and Olivier Elemento,2009, DISPARE: discriminative pattern refinement for position weight matrices. BMC Bioinformatics, 10(388):1471-2105.

[7] D.Lo, S.C.Khoo, and C.Liu, 2007, Efficient mining of iterative patterns for software specification discovery. Int.

Conf. on Knowledge Discovery and Data Mining, pages 460-469.

[8] Hirosawa et al.,1995, Comprehensive study on iterative algorithms of multiple sequence alignment. Computational Applications in Biosciences, 11:13-18.

[9] X. Guan and E.C. Uberbacher,1996, A fast look-up algorithm for detecting repetitive DNA sequences. Proceedings of the paci_c symposium on Biocomputing, pages 718-719.

[10] L.Kyle Jensen, P. Mark Styczynski, Isidore Rigoutsos, and N. Gregory Stephanopoulos, 2006, A generic motif discovery algorithm for sequential data. Bioinformatics, 22(1):21-28.

[11] Bin Ma.,2000, A polynomial time approximation scheme for the closest substring problem. LCNS Springer, 1848:99-107.

[12] D. Maier,1978, . The complexity of some problems on subsequences and super sequences. ACM, 25:322-336.

[13] M. Martinez, 1983, An efficient method to find repeats in molecular sequences. Nucleic Acid Research, 11:4629-4634.

[14] M. Martine, 1988,. A flexible multiple sequence alignment program. Nucleic Acid Research, 16:1683-1691.

[15] M.Li, B.Ma, and L.Wang, 2002, On the closest string and substring problems. J. ACM, 49(2):151-171.

[16] A. Murugan and B.Lavanya,2010, DNA algorithmic approach to solve GCS problem. Journal of Computational Intelligence in Bioinformatics, 3(2):239-247.

[17] A. Murugan, B.Lavanya, and K. Shyamala, 2011, A novel programming approach for DNA computing. International Journal of Computational Intelligence Research, 7(2):199-209.

[18] M.Zhang, B.Kao, D.Cheung, and K.Yip, 2005, Mining periodic patterns with gap requirement from sequences. SIGMOD Int. Conf. on Management of Data, pages 623-633.

[19] S.B. Needleman and C.D. Wunsc,1970, A general method applicable to the search of similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48:443-453.

[20] A.F. Neuwald and P. Green, 1994, Detecting patterns in protein sequences. Journal of Molecular Biology, 239:698-712.

[21] C.G. Neville-Manning, K.S. Sethi, D. Wu, and D.L. Brutlag, 1977, Enumerating and ranking discrete motifs. Proceedings of Intelligent Systems for Molecular Biology, pages 202-209.

[22] R.Agarwal and R.Srikant.,1995, Mining sequential patterns. Int.Conf. on Data Engineering.

[23] R.Agarwal and R.Srikant, 1976. Mining sequential patterns: Generalizations and performance improvements. Extending DataBase Technology, pages 3-17.

[24] Isisdore Rigoutsos and Aris Floratos.1998, Combinatorial pattern discovery in biological sequences: the teiresias algorithm. Bioinformatics, 14(1):55-67.

[25] Saurabh Sinha,2006, On counting position weight matrix matches in a sequence, with application to discriminative motif finding. Bioinformatics, 22(14):454-463.

[26] H.O. Smith, T.M. Annau, and S. Chandrasegaran, 1990, Finding sequence motifs in groups of functionally related proteins. Proceedings of National Academy (USA), 87:826-830.

[27] R.F. Smith and T.F. Smith, 1990, Automatic generation of primary sequence patterns from sets of related protein sequences. Nucleic Acid Research, 18:118-122.

[28] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. Journal of Molecular Biology, 147:195-197.

[29] R. Staden,1984, Computer methods to locate signals in nucleic acid sequences. Nucleic Acids Res, 12:505-519.

[30] G. Stormo, 2000, DNA binding sites: representation and discovery. Bioinformatics, 16:16-23.

[31] M. Suyama, T. Nishioka, and O. Junichi,199,. Searching for common sequence patterns among distantly related proteins. Protein Engineering, 8:1075-1080..

[32] M. Tompa,1999, An exact method for finding short motifs in sequences with application to ribosome binding site problem. Proc. Seventh Int'l Conf Intelligent Systems for Molecular Biology, pages 262-271.

[33] L. Wang and T. Jiang, 1994, On the complexity of multiple sequence alignment. Journal of Computational Biology, 1:337-348.

[34] M.S. Waterman, D.J. Galas, and R. Arratia, 1984, Pattern recognition in several sequences: consensus and alignment. Bulletin of Mathematical Biology, 46:515-527.

[35] T.D.Wu and D.L. Brutlag,1995, Identification of protein motifs using conserved amino acid properties and partitioning techniques. Proceedings of the 3rd International conference on Intelligent Systems for Molecular Biology, pages 402-410.

[36] X.Yan, J.Han, and R.Afhar, 2003, Colspan: Mining closed sequential patterns in large datasets. SIAM Int. Conf. Data Mining, pages 166-177.