Mining Longest Common Subsequence and other Related Patterns using DNA Operations

A. Murugan
Department of Computer Science
Dr. Ambedkar Government College
Chennai. INDIA.

B. Lavanya
Department of Computer Science
University of Madras
Chennai. INDIA.

ABSTRACT
Longest Common Subsequence (LCS) and Shortest Common Subsequence (SCS) problems are to find subsequences in given sequences in which the subsequence is as long as possible and as short as possible subsequence respectively. These subsequences are not necessarily contiguous or unique. In this paper we have proposed two new approaches to find LCS and SCS, of N sequences parallely, using DNA operations. These approaches can be used to find LCS and SCS, of any window size, from any number of sequences, and from any type of input data. The proposed work can be applied to finding diverging patterns, constraint LCS, redescription mining, sequence alignment, speech recognition, find motifs in genetic data bases, pattern recognition, mine emerging patterns, contrast patterns in both scientific and 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
Data Mining, Pattern Recognition, Molecular Computing.

Keywords
DNA operations, Motifs, LCS, SCS, CLCS, Pattern recognition, Diverging pattern, Exceptional mining.

1. INTRODUCTION

One important area of algorithm design is the study of algorithms for different character strings. Among the most important is, efficiently searching for substrings or generally different patterns in large databases. In many instances we do not want to find a subsequence exactly, but rather something that is ``similar''. The process of discovery of patterns in the genetic data proves to be essential in many biological researches and commercial interpretations. Genetic codes are stored in DNA molecules. The DNA strands can be broken down into long sequences each of which is one of four basic types : A, T, C, G. But the exact matches rarely occur in biology because of small changes in DNA mutation. Exact substring search will only find exact motifs, like [3,4]. For this reason, it is of interest to compute similarities between subsequences that do not match exactly. The method of sequence similarity should be insensitive of random insertions, deletions and type of characters from some originating sequence. They are finding the edit distance, Generalized Center String [1], LCRS, CPM, gapped subsequences [3,4], Longest Common Subsequences [LCS] [23] etc. 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 longest common pattern from large number of sequences. The concern is also on the quality of identified patterns and time taken to discover them plays a vital role in huge researches. These prime issues motivates the proposed work.

The task of discovering frequent subsequences as patterns in a sequence database is done in [21,22,28,31,40]. 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 [5,9,13,16,26,35]. A detailed survey of several multiple-string alignment algorithms can be found in [11]. They encountered many notable problems like, the task of optimally aligning a set of strings is computationally very expensive[19] and they could only align the global similarities[30]. 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)[4] 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 [10,17,18,33,36]. A number of pattern discovery algorithms have been steadily appearing in the literature [1,4,7,12,13,14,15,19,25,27].

We note that solving the Longest Common Rigid Subsequence problem (LCRS), Generalized Centre String (GCS) and the Closest Substring Problem (CSP), are generalizations of the Longest Common Subsequence (LCS) problem and were found to be NP hard [8,24,29].

For huge databases, storing and retrieving of data is computationally expensive and time consuming, but with DNA strands and DNA operations[2], the storage and retrieval are done parallely, thus reducing the time complexity. Extracting such sequences and subsequences from a database of sequences [32], is an important data mining task with plenty of different application domains.

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 [18]. 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 [20].

2. LITERATURE REVIEW

There are studies on mining only representative patterns, such as closed sequential patterns by Yan et al [40]. 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 Simulation of all the DNA operations are done in [2], the proposed work uses the DNA operations cut and pcr operations. Mining GCS,

Using DNA operations and modified PWM, given a sequential database is performed in [1]. In DNA sequence mining, Zhang et al [28] 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

And different other patterns are captured, and the support is the total number of such occurrences are found in [3, 4]. This paper deals with finding longest common subsequences of any window size ,with given constraint, diverging pattern and contrast pattern [6,23,37,38].

2.1 Definitions

Definition1.(Subsequence and Landmark): Sequence $S = e_1, e_2, ...e_m$ is a subsequence of another sequence $S' = e'_1, e'_2, ...e'_n$ ($m \leq n$), denoted by $S \subseteq S'$ (or S' is a super sequence of S) $1 \leq l_1 \leq l_2 \leq ... \leq l_m \leq n$ such that $S[i] = S'[l_i]$ (i.e., $e_i = e'l_i$) for i = 1, 2, ..., m. Such a sequence of integers $\langle l_1, l_2, ...l_m \rangle$ is called a landmark of S in S'.

A pattern $P = e_1, e_2, ...e_m$ is also a sequence. For two patterns P and P', if P is a subsequence of P', then P is said to be a sub-pattern of P', and P' a super-pattern of P .

Definition 2 . Instances of Pattern: For a pattern P in a sequence database $SeqDB = S_1, S_2, ..., S_n$, if $\langle l_1, e_2, ...l_m \rangle$ is a landmark of pattern $P = e_1, e_2, ...e_m$ in $S_i \in SeqDB$, pair $(i, \langle l_1, e_2, ...l_m \rangle)$ is said to be an instance of P in SeqDB, and in particular, an instance of P in sequence S_i.

Definition 3. Repetitive Support and Support Set: The repetitive support of a pattern P in SeqDB is defined to be sup(P) = max (I) where I Ɛ SeqDB(P) is non-redundant. The non-redundant instance set I with I = sup(P) is called a support set of P in SeqDB.

Definition 4. Position Weight Matrix: Given a finite alphabet Σ and a positive integer m, a PWM M is a matrix with ||Σ|| rows and m columns. The coefficient M, (p, x) gives the score at position p for the letter x in Σ. The PWM defines a function from σ_m to ʀ, that associates a score to each word $u = u_1,u_2,...,u_p$ of σ_m :

$$\text{Score } M(u) = \Sigma mp-1\ M(p, up),$$

Let α be a score threshold. We say that M has an occurrence in a text T at position k if Scor eM ($T_k ...T_{k+m-1}$) ≥ α.

The most recurrent task is to predict binding sites in a large DNA sequence, that is to look for occurrences of a PWM, given a text.

Definition 5. Longest common subsequence: Given two sequences $X = < x_1 , x_2, \ldots, x_m >$ and $Z =< z_1, z_2, \ldots, z_k >$, we say that Z is a subsequence of X, if there is a strictly increasing sequence of k indices $< i_1, i_2, ..., i_k >$ ($1 \leq i_1 \leq i_2 \leq ... \leq i_k \leq n$) such that $Z = < x_1, x_2, \ldots x_{ik} >$.

For example, let X = < ABRACADABRA > and let Z =< AADAA >, then Z is a subsequence of X. Given two strings X and Y for example, let X be as before and let Y = <YABBADABBADOO>. Then the LCS is Z = < ABADABA >, refer Figure 1.

S1	**H U M A N**
S2	**C H I M P A N Z E E**
LCS	**HMAN**

Figure 1. Example of Longest Common Subsequence

There are many solutions for finding LCS like dynamic programming solution [37], Hunt-Szymanski algorithm [23], etc. This article proposes new approaches to find LCS using DNA operations and modified position weight matrices.

3. DNA BASED LCS AND DIFFERENT RELATED PATTERNS DISCOVERY

In this paper, we propose, two new approaches to study the Longest Common Subsequences mining problem and other different related patterns. Algorithms 1 and 2 searches for all common sequences of different window sizes and different other patterns, in input sequences, using support vector and modified Position Weight Matrices (PWM).

Our approaches 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 algorithms does not use any complex statistical models but rather uses DNA operations and DNA strands to search for the presence or absence of patterns. 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 [1,4], thus solving the above noted problem.

3.1 Finding LCS using Support Vector (LCSSV)

Algorithm LCSSV discovers LCS and different related patterns, with its support vector using DNA operations.

Let $S = (s_1,s_2...s_N)$, be the N input sequences, encoded in 0's and 1's, refer Figure 2, and the level_number be the maximum length of LCS sequence required, that is, the window size of LCS. LCS strand along with its supp, that is, number of times each subsequence is present in the given sequences be the output strands (support). Step 2 generate DNA_0, the possible combinations of 0's and DNA_1, the possible combinations of 1's [18], depending on the level_number. Let the number_of_nodes be the variable, which stores the total number of elements present in DNA_0 or DNA_1. Steps 4 and 5 performs the pcr operation on DNA_0 and DNA_1 strands, for N sequences and stores in $DNA_{\_01}...DNA_{0N}$ and

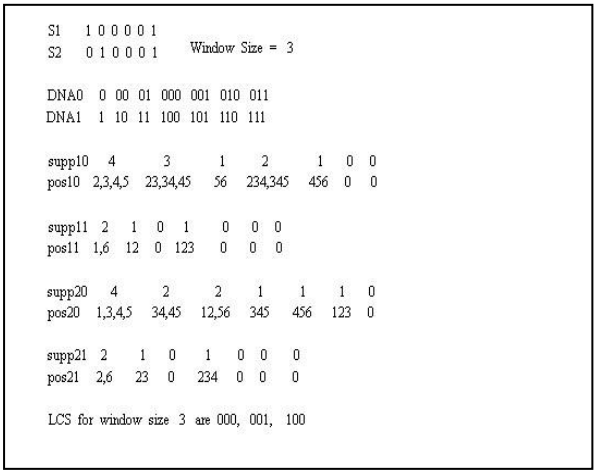

Figure 2. Finding LCS using Support Vector

DNA_{11}...DNA_{1N} respectively. In steps 6 to 10, for each element of DNA_{01}...DNA_{0N} and DNA_{11}...DNA_{1N}, threads are created parallely and cut operation is applied to find the support count and position of its occurrences and stored in $supp_{01}$...$supp_{0N}$, pos_{01}...pos_{0N}, $supp_{11}$...$supp_{1N}$ and pos_{11}...pos_{1N} respectively. In steps 12 to 21, the supp and pos strands are searched vertically for occurrences of all common subsequences for all window sizes and finally the LCS is found for the given level_number refer **Figure 3** shown below.

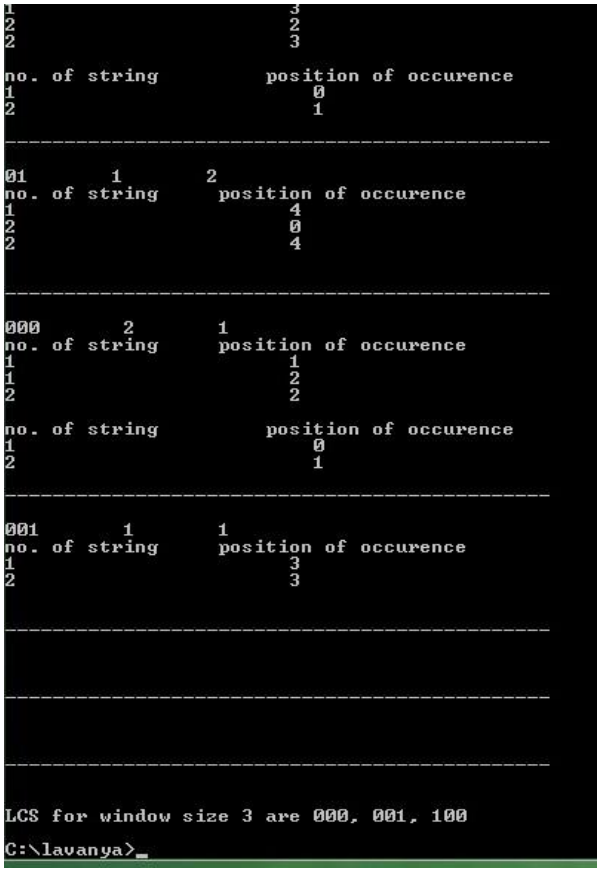

Figure 3. Illustration of LCSSV.

Algorithm 1: DNA-based-LCS discovery using Support Vector (LCSSV).

Input: S, level_number

Output: LCS strand, supp strand

1. **begin**
2. **Generate DNA_0 and DNA_1;**
3. **number_of_ nodes ← size(DNA_0) ;**
4. **DNA_{01}...DNA_{0N} ← pcr(DNA_0) ;**
5. **DNA_{11}...DNA_{1N} ← pcr(DNA_1) ;**
6. **foreach element of DNA_{01}...DNA_{0N} and DNA_{11}...DNA_{1N} do**
7. **Create threads parallely ;**
8. **let $supp_{01}$...$supp_{0N}$[],pos_{01}…pos_{0N}[][]← cut(S,DNA_{01}...DNA_{0N}[element]) ;**
9. **let $supp_{11}$...$supp_{1N}$[],pos_{11}…pos_{1N}[][]← cut(S,DNA_{11}...DNA_{1N}[element]) ;**
10. **end**
11. **[parallely for lcs0 and lcs1] ;**
12. **foreach j from 1 to number_ of_ nodes do**
13. **if ($supp_{01}$[j]…$supp_{0N}$[j]) > 0 then**
14. **lcs0[] ← DNA_{01}[j];**
15. **supp1[] ← min($supp_{01}$...$supp_{0N}$);**
16. **end**
17. **if ($supp_{11}$[j]…$supp_{1N}$[j]) > 0 then**
18. **lcs1[] ← DNA_{11}[j] ;**
19. **supp2[]← min($supp_{11}$...$supp_{1N}$);**
20. **end**
21. **end**

3.1.1 Time Complexity

TC(LCSSV) = max(O(max(PCR,CUT)),O(LCS)).

If levelnumber ≠ 0, then

TC(LCS) = O(levelnumber).

Therefore at its best case

TC(LCSSV) is between O((n/L) + n) and (O(levelnumber)).

At its average and worst case

TC(LCSSV) is between (O(n/M)+O((n/L)+n)) and O(levelnumber) .

3.1.2 Special Case: Shortest Common Subsequence

Algorithm LCSSV can be used to find Shortest Common Subsequence in S. Since all possible common sequences of all window sizes from 1 to level_number is generated, the Algorithm LCSSV can be used to find common sequence of any small length, thus SCS. From Figure 3 SCS is found by varying the window size, that is 1.

3.2. Finding LCS using modified PWM (LCSPWM)

DNA based LCS discovery using modified PWM, discovers LCS in the given N sequences using DNA operations and modified PWM.

Algorithm 2: DNA-based-LCS discovery using modified PWM (LCSPWM).

Input: S
Output: LCS strand, PWM strand

```
1.  begin
2.    L ←min(S);
3.    T1, T2, ..., TN ← pcr(S);
4.    PWM1[1…L]← cut(T1 , sL[element]);
5.    PWM2[1…L]← cut(T2 , sL[element]);
6.    ... PWMN[1..L] ← cut(TN , sL[element]);
7.    j ←1;
8.    foreach i ranging from 1 to L do
9.      if (PWM1[i][j] > 0) then
10.         test ← PWM2[i][0];
11.         lcs[][] ← i;
12.         lcs[][] ← PWM1[i][j];
13.     end
14.     foreach (i ranging from i + 1 to L) AND
                    (PWM1..N [i][j] ≠ ø)  do
15.       if (test < PWM2[i][j]) then
16.         test←  PWM2[i][j];
17.         lcs[][0] ← i;
18.         lcs[][1]  ←PWM2[i][j] ;
19.       end
20.       else
21.         j++ ;
22.       end
23.     end
24.     Extended for N number of PWM
      strands
25.   end
26.  end
```

Let S = (s_1, s_2,…s_N), be the N input sequences, and LCS and PWM are the output strands. Step 2 finds the length of the smallest sequence in S and stores in L. Step 3 performs the pcr operation on each of the sequence in S and stored in T_1, T_2,…,T_N. Steps 4-6 does the cut operation on T_1, T_2,…,T_N, for each of s_L[element] and their position weight matrices are generated as PWM_1[1…L], PWM_2[1…L], ... PWM_N[1…L] respectively. Steps 8 to 25 performs a vertical check operation on all PWM, checking for the occurrences of elements of s_L, in order of their presence and stored in LCS strand. Algorithm LCSPWM illustrates for PWM1 and PWM2 strands, as shown in Figure 4, thus can be extended for N number of PWM strands.

Algorithms LCSSV and LCSPWM can be used to generate LCS for different support counts, for any window size, and all LCS with position of its occurrences, discover Constrained Longest Common Subsequence (CLCS) and find diverging and emerging patterns.

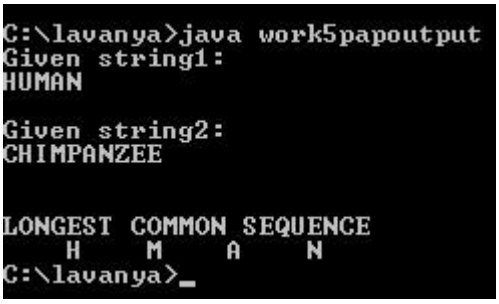

Figure 4. Discovery of LCS – using modified PWM

S1 T A G T C A C G

S2 A G A C T G T C

C = A T

Possible LCs of window size 4 is

A G A C & A G T C

CLCS = A G T C

Figure 5. Discovery of CLCS

3.2.1 Time Complexity

TC(LCSPWM) = O(O(max(PCR,CUT)),O(LCS)).

If PWM $\notin$ ø , then

TC(LCS) = O(|min(S)|),

Therefore at its best case

The TC(LCSPWM) is between O((n/L)+n) and O(|min(S)|),

At its average and worst case

The TC(LCSPWM) is between (O(n/M) + O((n/L) + n)) and O(|min(S)|)

If PWM $\in$ ø , iff $s_L \notin$ T [from Lemma 1]

TC(LCSPWM) = O(PCR + CUT)) implies O(n/M) at its average case.

4. DIFFERENT PATTERNS

4.1 Special Case 1: Find CLCS and Sequence Divergence

Algorithms LCSSV and LCSPWM can be extended to find Constraint Longest Common Subsequence (CLCS) for given S. Since all possible LCS are found, the constraint can be can be applied and the final CLCS can be found as shown in Figure 5 and thereby find sequence divergence also.

4.2 Special Case 2: Find Diverging and Emerging Patterns

Algorithms LCSSV and LCSPWM can also be used to find diverging and emerging patterns for given S. Steps 13 to 20 in Algorithm LCSSV and steps 14 to 23 in Algorithm LCSPWM, can be modified to find diverging and emerging patterns for given S.

4.3 Special Case 3: Re description Mining

The goal of re description mining is to use the given descriptors as a vocabulary and find subsets of data. Algorithms LCSSV and LCSPWM can be extended to find subsets from a given set of data.

S1 C O M P U T E R

S2 C A L C U L A T I O N

PWMS2 1,4 10 0 0 5 8 0 0

LCS 1<5<8 so C U T

or

1 < 10 so C O

Figure 6: Illustration of more number of LCS

Lemma: 1 Let T be set of alphabets. Let S = { $s_1, s_2, s_3,\ldots, s_n$}, where S ∈ T. If $s_i \notin T$ then LCS is ø where $1 \leq i \leq n$.

Proof: LCS is the longest common subsequence in S, thus if any of { $s_1, s_2, s_3,\ldots, s_n$}, as elements ∉ T, the set of alphabets, then there exists no common subsequence in S, thus LCS is ø .

Lemma: 2 Let T be set of alphabets. Let S = { $s_1, s_2, s_3,\ldots, s_n$}, where S ∈ T. Then $\max(|LCS|) \leq \min(|S||)$.

Proof: Let s_i be the shortest sequence in S, then length of LCS of n sequences S = { $s_1, s_2, s_3,\ldots, s_n$}, is at most equal to the number of characters of the shortest sequence in S, that is $\max(|LCS|) \leq \min(|s_i|)$.

Lemma: 3 Let T be set of alphabets. Let S = { $s_1, s_2, s_3,\ldots, s_n$}, where S ∈ T. Let CS = {cs_1 , cs_2, …, cs_k}, where $1 \leq k \leq n$, be the set of common sequences of all window size in S. Then LCS ⊆ CS.

Proof: The LCS of n sequences S, is a longest common sequence, that is, the element of common sequences set, of all window size. Let cs_1 be common sequence of window size 1, let cs_2 be common sequence of window size 2, and so on, then $cs_1 \subseteq cs2$, $cs_2 \subseteq cs_3, \ldots cs_{k-1} \subseteq cs_k$. Thus LCS ⊆ CS.

Lemma: 4 Let T be set of alphabets. Let S { $s_1, s_2, s_3,\ldots, s_n$}, where S ∈ T. Let CS = {cs_1 , cs_2, …, cs_k}, where $1 \leq k \leq n$, be the set of common sequences of all window size in S and F be set of constraint characters, F ∈ T. Then CLCS ⊆ CS.

Proof : CS is the set of all common sequences in S of all window size. So, the CLCS with the elements of F, is also present in CS. Thus CLCS ⊆ CS.

Algorithm LCSSV and LCSPWM can be used to generate LCS for different inputs, different support counts, for any window size of the sequence, and find all LCS and CLCS with position of its occurrences.

5. PERFORMANCE

Algorithms LCSSV and LCSPWM have been implemented and tested with simulated and real databases. The random DNA sequences of size varying from 100 to 25000, are generated from http://old.dnalc.org/bioinformatics/dnalc-nulceotide-analyzer.htm#randomizer 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 LCSSV and LCSPWM proved to be efficient and accurate in solving LCS and CLCS in the given sequences. Tested with randomly generated and real motifs, our work could discover all motifs present, with its positions of existence. All implementations are performed on a dual core computer and 5 GB main memory using Java. The operating system is Windows XP. The resulted data of these experiments are consistent. The limitation of these algorithm is that the maximum number of threads generated, is dependent on the efficiency of the system architecture.

6. 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 new algorithms use 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 finding, the existence of subsequences given a large database of commercial or genetic information are unlimited, the searching for LCS and CLCS 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 other commercial database analysis.

7. CONCLUSION

In this paper, we have designed and performed the implementations to find LCS, SCS, and different related patterns, in a highly parallel way, 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.

8. REFERENCES

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

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

[3] Lavanya. B. and Murugan. A. Discovering sequence motifs of different patterns parallelly using DNA operations. International Journal of Computer Applications, 3(1):18-24, Nov 2011.

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

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

[6] James Bailey, Thomas M Anoukian, and Kotagiri Ramamohanroa. Fast algorithms for mining emerging patterns. LCNS Springer, pages 39-50, 2002.

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

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

[9] Wu. T. D. and Brutlag. D. L. 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, 1995.

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

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

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

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

[14] Smith. T. F. and Waterman. M. S. Identification of common molecular subsequences. Journal of Molecular Biology, 147:195-197, 1981.

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

[16] Neville-Manning. C. G., Sethi. K .S., Wu. D.,and Brutlag. A. D. Enumerating and ranking discrete motifs. Proceedings of Intelligent Systems for Molecular Biology, pages 202-209,1997.

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

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

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

[20] Nan Li and Tompa. M. Analysis of computational tools for motif discovery. Algorithms of molecular biology, pages 1-8, 2006.

[21] Lo. D.and Khoo. S. D. Liu. Efficient mining of iterative patterns for software specification discovery. Int. Conf. on Knowledge Discovery and Data Mining, pages 460-469, 2007.

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

[23] Landau. G. M., Levy. A., and Newman. I. LCS approximation via embedding into locally nonrepetitive strings. Information and Computation, 209:705-716, 2011.

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

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

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

[27] Suyama. M., Nishioka. T., and Junichi. O. Searching for common sequence patterns among distantly related proteins. Protein Engineering, 8:1075-1080, 1995.

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

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

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

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

[32] Agarwal. R. and Srikant. R. Mining sequential patterns: Generalizations and performance improvements. Extending DataBase Technology, pages 3-17, 1996.

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

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

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

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

[37] Yin-Te Tsai. The constrained longest common subsequence problem. Information processing letter, 88:173-176, 2003.

[38] Qian Wan and Aijun An. Diverging patterns: Discovering signi_cant dissimilarities in large databases. Technical Report CSE-2008-10, Dec 2008.

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

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