DISCOVERY of LONGEST INCREASING SUBSEQUENCES and its VARIANTS using DNA OPERATIONS

B.LAVANYA [#1], A. MURUGAN [*2]

[#] Department of Computer Science, University of Madras
Chennai, India.

[1] lavanmu@gmail.com

[*] Department of Computer Science, Dr. Ambedkar College
Chennai, India

[2] amurugan1972@gmail.com

Abstract - The Longest Increasing Subsequence (LIS) and Common Longest Increasing Subsequence (CLIS) have their importance in many data mining applications. We propose algorithms to discover LIS and CLIS from varied databases. This work finds all increasing subsequences from the given database, find increasing subsequences in *n* sliding window, longest increasing sequences in one and more sequences, decreasing subsequences and common increasing sequences of varied window sizes. The proposed work can be applied to finding diverging patterns, constraint LIS, sequence alignment, find motifs in genetic data bases, pattern recognition, mine emerging patterns, and contrast patterns in both, scientific and commercial databases. The algorithms are implemented and tested for accuracy in both real and simulated databases. Finally, the validity of the algorithms are checked and their time complexity are analyzed.

Keywords: DNA operations, Motifs, LIS, LDS, CLIS, Pattern recognition, Exceptional mining, Molecular computing.

I. Introduction

We consider the problem of extracting a longest increasing subsequence (LIS) from a sequence of integers. The sequence S is assumed to be a permutation of the set {1, 2, . . . , n}, but having multiple occurrences of integers between 1 and n, in the sequence of length n, does not change the result. Longest Increasing Subsequence, searching from a large database is been widely studied. Efficiently searching for substrings or generally different patterns in large databases is needed today. 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. The method of subsequence searching 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 [18,19], Longest Common Subsequences [LCS] [21] etc. The nature of identifying patterns varies with applications. The concern is also on the quality of identified patterns. The time taken to discover them plays a vital role in huge researches. These prime issues motivates the proposed work.

A detailed survey of several multiple-string alignment algorithms can be found in [10]. They encountered many notable problems like, the task of optimally aligning a set of strings is computationally very expensive [17] and they could only align the global similarities [27]. To overcome the difficulty of alignment problem, modified Position Weight Matrices (PWM) [19] 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 [7,29,26].

For huge databases, storing and retrieving of data is computationally expensive and time consuming. Using DNA strands and DNA operations [22], the storage and retrieval of data can be done parallely, this reduces the time and space complexity. Extracting such sequences and subsequences from a database of sequences [24], is an important data mining task with plenty of 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 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].

II Literature Review

The LIS problem is closely related to the longest common subsequence problem, which has a quadratic time dynamic programming solution [6,13]. Algorithms for finding the LIS date back to Robinson [12] and Schensted [25] with a generalization by Knuth [16]. Fredman [11] showed how to compute an LIS of a length **n** sequence in optimal O(n log n) time. When the input sequence is a permutation of { 1, 2, . . . , n} , Hunt and Szymanski [14] designed an O(n log log n) time solution, which was later simplified by Bespamyatnikh and Segal [4]. A survey was done in [2] and by Odlyzko and Rains [23] who discuss on many related issues, and references therein.

The support of a pattern is the number of sequences containing the given pattern and its commonality between various other sequences. The longest increasing subsequence problem refers either to identifying the longest increasing subsequence(s) or, alternatively, to determining the length k of the LIS. In either of these forms, this problem has been the subject of intense study by mathematicians and computer scientists alike. This problem has interesting properties both from a purely combinatorial perspective, as well as actual applications in fields such as DNA sequence matching [8]. This problem should not be confused with the Longest Common Subsequence (LCS) problem which considers two sequences and locates a series of entries that appear in the same order in both sequences. However, LIS is a sub case of LCS.

Simulation of all the DNA operations are done in [22], the proposed work uses *cut* and *pcr* DNA operations. Mining GCS, using DNA operations and modified PWM, given a sequential database is performed in [1]. 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, with their support count [21,18,19]. This paper deals with finding longest increasing subsequences of any window size, with given constraint, from one or more sequences.

A. 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 $Si \in SeqDB$, 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 Si.

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 σm to ʁ, that associates a score to each word u = {u1,u2,...,up} of σm :

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

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

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 Increasing Subsequence: Given a sequence $S = \{ s_1, s_2, ..., s_n\}$ and a window size $w \leq n$, a window of width w is a subsequence $\{ s_{i+1}, s_{i+2}, ..., s_{i+w} \}$ for some $0 \leq i \leq n-w$. We also consider the truncated windows $(s_1, ... , s_j)$ for $j \leq w$ and $(s_j, ..., s_n)$ for $j \geq n-w$ as windows of size w. The general problem that we consider is that of determining a LIS in each of the windows wi and also termed as Longest Increasing Subsequence in Sliding Window (LISSW).If the size of w is fixed it is termed as Longest Increasing Subsequence in Fixed Window (LISFW).

Input Sequence 6 9 8 2 3 5 1 4 7

LIS 1 2 3 4 7

LIS 2 2 3 5 7

Figure 1: Example of Longest Increasing Subsequence

Longest Increasing Subsequence is the increasing subsequence of S, that has the maximal possible length. There can be many possible LIS of S since only the length is unique. For example, S = (3, 1, 4, 5, 9, 2, 6, 8, 7) has four longest increasing subsequences, including (1, 4, 5, 6, 7) and (3, 4, 5, 6, 8). There are many solutions for finding LIS like [3,5,28,9,15]. This article proposes new approaches to find LIS and its variants using DNA operations and modified position weight matrices, from one or more sequences.

Definition 6. Longest Increasing Subsequence: Let A and B be two sequences $A = (a_1,a_2,...,a_m)$ and $B = (b_1,b_2,...,b_n)$, where $m \geq n$ and each pair of elements in the sequences is comparables. A common increasing subsequence of A and B is a subsequence $(a_{i1} = b_{j1}, a_{i2} = b_{j2},...,a_{i3} = b_{j3})$, where $i_1 < i_2 <...< i_{il}$ and $j_1 < j_2 <...< j_{il}$, such that for all $1 \leq k < l$, we have $a_{ik} < a_{ik+1}$. The longest common increasing subsequence of A and B, is a common increasing subsequence of maximum length.

Definition 7. Common Decreasing Subsequence (CDS): Let A and B be two sequences $A = (a_1,a_2,...,a_m)$ and $B = (b_1,b_2,...,b_n)$, where $m \geq n$ and each pair of elements in the sequences is comparables. A common decreasing subsequence of A and B is a subsequence $(a_{i1} = b_{j1}, a_{i2} = b_{j2},...,a_{i3} = b_{j3})$, where $i_1 > i_2 >...> i_{il}$ and $j_1 > j_2 >...> j_{il}$, such that for all $1 \leq k < l$, we have $a_{ik} > a_{ik+1}$. The longest common decreasing subsequence of A and B, is a common decreasing subsequence of maximum length.

Definition 8. Common Heaviest Increasing Subsequence (CHIS): Let A and B be two sequences $A = (a_1,a_2,...,a_m)$ and $B = (b_1,b_2,...,b_n)$, where $m \geq n$, each element is accompanied by a weight and each pair of elements in the sequences is comparables. The goal is to find the increasing subsequence of maximal sum of weights. The Common Heaviest Increasing Subsequence of A and B, is a common increasing subsequence of maximum sum of weights.

III. DNA based LIS and its variant patterns discovery

In this paper, we propose, new methods to study the Longest Increasing Subsequences mining problem and other different related patterns. Algorithms 1 and 2 searches for all increasing sequences of different window sizes and different other variant patterns, in one and more input sequences, using modified Position Weight Matrix (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 given type 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 [1,19], thus solving the above noted problem.

Input sequence 6 9 8 2 3 5 1 4 7

Window size (W) { 2, 3, 4, 5}

W=2 6 9 / 6 8 / 6 7 / 2 3 / 2 5 / 2 4 / 2 7 /
3 5 / 3 7 / 3 4 / 5 7 / 1 4 / 1 7 / 4 7

W=3 2 3 5 / 2 3 4 / 2 3 7 / 2 5 7 /
2 4 7 / 3 5 7 / 3 4 7 / 1 4 7

W=4 2 3 5 7 / 2 3 4 7

W= 5 ----------

Figure 2: LIS for all window sizes possible

A. Finding LIS in single sequence for all Window sizes (LISW)

Algorithm LISW discovers all increasing subsequences, LIS and different related patterns, using DNA operations and modified PWM, as shown in Figure 2.

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

Input: S, no of elements (noe), window size
Output: LISW strands

```
1  begin
2    let n ← max(noe)
3    let m ← max(window size)
4    let t1 … tn ← pcr(S)
5    PWM1 ← cut(t1 , noe[1]) ;
6    PWMn ← cut(tn , noe[n]) ;
7   [parallely for each window size LISW2, LISW3, …, LISWm]
8    foreach windowsize from 2 to m do
9          [Create threads parallely] ;
10       foreach i from 1 to |S| do
11          if PWM1[i] > 0 then
12            test ← PWM1[i] ;
13         foreach j from i + 1 to |S| do
14           if (PWM1[j] > test) then
15             LISW2[k][0] ← i ;
16             LISW2[k][1] ← PWM1[j];
17           end
18         end
19       end
20     end
21   end
22 Extended for higher window sizes ;
23  end
```

Let no_of_elements (noe), be the set of elements, such that S = (s_1, s_2, s_3, … ,s_m) ε (noe), and window size be the set of window sizes starting from 2 to max(S). The ouput of Algorithm 1 is LISW strands for window size 2 to max(window size). In step 2, n is assigned the value of max(noe). In step 3, m is assigned the value of max(window size). Step 4 performs the pcr operation on S and stores them in t_1, t_2, t_3, … , t_n strands. Steps 5 and 6

performs cut operation on t_1 with noe[1], t_2 with noe[2], ... , t_n with noe[n] and stores in PWM_1, PWM_2, ... , PWM_n strands. Steps 8 to 21 performs the process of parallely finding longest increasing subsequences for different window sizes given. It vertically checks for the existence of increasing subsequences using position weight matrices of step 5 to 6. Algorithm 1 depicts steps for finding IS, for window size 2, which could be extended for any window sizes each done parallely. The contents of LISWm are the required LIS of the given S. This algorithm can be used for finding Shortest Increasing Subsequences (SIS). The minimum the window size, the shorter is the discovered subsequence length, thus algorithm 1, finds SIS also.

Time Complexity

The time complexity of Algorithm LISW consists of two sections. The section 1 comprises of steps from 2 to 6 and section 2 contains steps from 10 to 21. Therefore,

TC(LISW) = O(max(O(section1),O(section2)))

TC(section1) = O(max(PCR,CUT))

If PWM $\notin$ ϕ then

TC(section2) = O((n - 1)(n - 1)!)

Therefore from [22] at its best case

The TC(LISW) is between O((n/L) + n) and O((n - 1)(n - 1)!)

At its average and worst case

The TC(LISW) is between (O(n/M)+O((n/L)+n)) and O((n-1)(n-1)!)

If PWM ε ϕ,

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

B. ***Finding LIS for each of the Given Element (LISGE)***

Algorithm LISGE discovers all increasing sub sequences of, each of the given elements and its variant patterns, in a given sequence using DNA operations, as shown in Figure 3.

No-of-Elements { 1, 2, 3, 4, 5, 6, 7, 8, 9}

Input Sequence 6 9 8 2 3 5 1 4 7

ALL IS [1 4 7 / 2 3 4 7 / 2 3 5 7 / 3 4 7 / 3 5 7 / 4 7 / 5 7 / 6 9 / 6 8 / 6 7]

LIS 2 3 4 7 / 2 3 5 7

Figure 3: LIS for each of the given element

Algorithm 2: DNA-based-MLCS discovery using modified PWM (MLCSPWM)

Input: S, no of elements(noe)
Output: allLIS strand
1 begin
2 let n ← max(noe)
3 let t_1, ... , t_n ← pcr(S)
4 let f, s, j, z1 ← 0
5 PWM_1← cut(t_1 , noe[1]) ;
...
6 PWM_n ← cut(t_n , noe[n]) ;
7 [Parallely for each of the element in noe] ;
8 foreach i ranges from 1 to noe[1] ... noe[n] PWM_i[j] > 0 do
9 foreach j ranges from i + 1 to noe[1] ... noe[n] do
10 if (PWM_i[j] < PWM_{z1}[j])then
11 $allLIS_2$[f][s] = PWM_{z1}[j]
12 [Increment f and s] ;
13 end
14 end
15 end

16 end

Let no_of_elements (noe), be the set of elements, such that S = (s_1,s_2, s_3, ... , s_m) **ε** (noe). The output of Algorithm 2 is allLIS strands for all noe[1], noe[2], ... , noe[n]. In step 2, n is assigned the value of max(noe). Step 3 performs the pcr operation on S and stores them in t_1, t_2, t_3, ... , t_n strands. In step 3, f, s, z1 and j is assigned the value 0. Steps 5 to 6 performs cut operation on t1 with noe[1], t_2 with noe[2], ..., t_n with noe[n] and stores in PWM_1, PWM_2, ..., PWM_n strands respectively. Steps 8 to 14 finds all increasing sub sequences for each of the element in no_of_ elements. Algorithm 2 depicts finding all increasing sub sequences for _rst element of noe, by vertically checking the contents of PWM_1, with all other PWM_2, ... , PWM_n. Thus finding all increasing sub sequences of each of the element of noe, thus algorithm 2, also finds LIS for each of element of noe. Similarly, this algorithm can be used for finding Shortest Increasing Subsequences (SIS) for all elements of noe.

Input sequence 6 9 8 2 3 5 1 4 7
Fixed window size = 3

Sequences are checked as 6 9 8 – 9 8 2 – 8 2 3 – 2 3 5 – 3 5 1 – 5 1 4 – 1 4 7

Increasing sub sequences 6 9 / 2 3 5 / 3 5 / 1 4 7

Longest Increasing sub sequences 2 3 5 / 1 4 7

Figure 4 : Example of LISSW and LISFW

Time Complexity

Like Algorithm 1, the time complexity of algorithm LISGE also consists of two sections. The section 1 comprises of steps from 3 to 6 and section 2 contains steps from 9 to 15. Therefore,

TC(LISGE) = O(max(O(section1),O(section2)))

TC(section1) = O(max(PCR,CUT))

If PWM ¢ ø , then

TC(section2) = O(n!),

Therefore from [22] at its best case

The TC(LISGE) is between O((n/L) + n) and O(n!)

At its average and worst case

The TC(LISGE) is between (O(n/M) + O((n/L) + n)) and O(n!)

If PWM ε ϕ,

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

IV. Variants of LIS

There are many variants of LIS, depending on its application. Algorithm 1 and Algorithm 2 can be used to _nd some of the variants listed below.

Special Case 1: Finding LISSW and LISFW: Longest Increasing Subsequence in Sliding Window (LISSW), or Longest Increasing Subsequence in Fixed Window (LISFW) can be discovered with Algorithm 1, see Figure 4 . Since algorithm 1 finds increasing sequences for all given window sizes, it can be modified to find LIS with sliding window also. The step 13 of algorithm 1 can be modified as j ranging from i + 1 to window size.

Input Sequence 1 6 9 8 2 3 5 6 4 7
Input sequence 2 2 4 8 3 6 5 1 6 7

CIS of window size 3 and above 2 3 5 / 2 4 7 / 5 6 7 /
2 3 5 6 / 3 5 6 7 / 2 3 5 6 7

LCIS 2 3 5 6 7

Figure 5: Example of CIS and LCIS for two sequences

Special Case 2: Finding CIS and LCIS: Common Increasing Subsequence (CIS), and Longest Common Increasing Subsequence (LCIS) can be discovered by using Algorithm 1, 2. Algorithm 1 and Algorithm 2, _nds IS and LIS of single sequence, the procedure can be extended for any number of sequences, creating that many number of strands parallely. Finally the IS of the sequences can be compared to find CIS and therefore LCIS can also be found. Figure 5 depicts the finding of CIS and LCIS.

Input Sequence S1 6 9 8 2 3 5 6 4 7
Input Sequence S2 2 4 8 3 6 5 6 4 7

D S of S1 6 2 / 6 3 / 6 5 / 6 5 4 / 9 8 2 / 5 4 / 9 8 3 /
9 8 5 4 / 9 8 6 / 9 8 7 / 6 4

D S of S2 4 3 / 8 3 / 8 6 5 4 / 8 7 / 6 5 4 / 5 4 / 6 4

L D S of S1 9 8 5 4
L D S of S2 8 6 5 4

C D S of S1 and S2 8 5 4 / 6 5 4 / 5 4 / 6 4

Figure 6: Example of DS and CDS for two sequences

Special Case 3: Finding DS and CDS : Algorithm 1, 2 can be used for finding Decreasing Subsequence (DS) and Common Decreasing Subsequence (CDS) in a single or more number of sequences. In step 14 of Algorithm 1, and step 10 of algorithm 2, the relational operator used to discover increasing subsequences have to be changed to find DS, thereby finding CDS as shown in Figure 6.

Input Sequence 6 9 8 2 3 5 1 4 7
Weights 2 3 4 2 6 8 4 5 3
Window size = 4

LIS 2 3 5 7 / 2 3 4 7
Weights 2 6 8 3 / 2 6 5 3

HLIS 2 3 5 7

Figure 7: Example of Heaviest Increasing Subsequence

Special Case 4 : Finding HIS and HCIS: With a minimum modification, Heaviest Increasing Subsequence (HIS) and Heaviest Common Increasing Subsequence (HCIS) can be found using Algorithm 1 and 2. This needs

an additional DNA strand to store each element's respective weight. The weight DNA strand along with PWM strand have to be checked for discovery of HIS, refer Figure 7 and if applied to more number of sequences finds HCIS also. These algorithms can further be used to find Heaviest Decreasing Subsequence (HDS) and Heaviest Common Decreasing Subsequence (HCDS).

V. Performance

Algorithms 1, 2 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 1, 2 proved to be efficient and accurate in solving LIS and CLIS 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 algorithms is that the maximum number of threads generated, is dependent on the efficiency of the system architecture.

VI. 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. The applications for finding, the existence of subsequences given a large database of commercial or genetic information are numerous. The searching for LIS and CLIS and all its variants, 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, analyse, 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, pattern avoidance, pattern matching, pattern mutation and other similar commercial database analysis.

VII. Conclusion

In this paper, we have designed and performed the implementations to find LIS, CLIS, and different variants of it, 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.

REFERENCES

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

[2] D. Aldous and P. Diaconis. Longest increasing subsequences: From patience sorting to the baik-deift-johansson theorem. bull. AMS, 36(4):413-432, 1999.

[3] J. Mikhail Atallah, K. Gleen Manacher, and J. Urrutia. Finding shortest maximal increasing subsequences and domination in permutation graphs.

[4] S. Bespamyatnikh and M. Segal. Enumerating longest increasing subsequences and patience sorting. Information Processing Letters, 76(1-2):7-11, 2000.

[5] Greth Stolting Brodal, Kanela Kaligosi, Irit Katriel, and Martin Kutz. Faster algorithms for computing longest common increasing subse-quences. LNCS, 4009:330-341, 2006.

[6] Maxime Crochemore and ely Porat. Fast computation of longest increasing subsequences and application. Information and computation, 208:1054-1059, 2010.

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

[8] Delcher, Kasif, Feischmann, Peterson, White, and Salzberg. Alignment of whole genomes. Nucleic Acids Research, 27:2369-2376, 1999.

[9] Sebastian Deorowicz. On some variants of the longest increasing subsequence problem. Theoretical and Applied Informatics, 21(3-4):135-148, 2009.

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

[11] M. L. Fredman. On computing the length of longest increasing sub sequences. Discrete Mathematics, 11(1):29-35, 1975.

[12] De B Robinson. G. On representation of symmetric group. Am.J.Math, 60:745-760, 1938.

[13] J. Hunt and T. Szymanski. A fast algorithm for longest common subsequences. Communications of ACM, 20(5):350-353, 1977.

[14] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. Communications of ACM, 20(5):350-353, 1977.

[15] Guy Jacobson and Kiem-Phong Vo. Heaviest increasing / common subsequence problems. LNCS, 644:52-66, 1992.

[16] D. E. Knuth. Permutations,matricesand generalized young tableaux. Paci_c.J.Math, 34:709-727, 1970.

[17] Wang. L. and Jiang. T. On the complexity of multiple sequence alignment. Journal of Computational Biology, 1:337-348, 1994.
[18] B. Lavanya and A. Murugan. Discovering sequence motifs of different patterns parallelly using DNA operations. International Journal of Computer Applications, 3(1):18-24, Nov 2011.
[19] B. Lavanya and A. Murugan. A DNA based approach to find closed repetitive gapped subsequence from a sequence database. International Journal of Computer Applications, 29(5):45-49, Sep 2011.
[20] Nan Li and M. Tompa. Analysis of computational tools for motif discovery. Algorithms of molecular biology, pages 1-8, 2006.
[21] A. Murugan and B. Lavanya. Mining longest common subsequences and other related patterns using DNA operations. International Journal of computer Applications, 49(18):38-44, july 2012.
[22] A. Murugan, B. Lavanya, and K. Shyamala. A novel programming approach for DNA computing. International Journal of Computational Intelligence Research, 7(2):199-209, 2011.
[23] A. M. Odlyzko and E. M. Rains. On longest increasing subsequences in random permutations. AMS, 1999.
[24] Agarwal. R. and Srikant. R. Mining sequential patterns: Generalizations and performance improvements. Extending DataBase Technology, pages 3-17, 1996.
[25] C. Schensted. Longest increasing and decreasing subsequences, Can.J.Math, 13:179-191, 1961.
[26] Saurabh Sinha. On counting position weight matrix matches in a sequence, with application to discriminative motif finding. Bioinformatics, 22(14):454-463, 2006.
[27] H. O. Smith, T. M. Annau, and S. Chandrasegaran. Finding sequence motifs in groups of functionally related proteins. Proceedings of National Academy (USA), 87:826-830, 1990.
[28] Richard Stanley. Increasing and decreasing subsequences and their variants. Proceedings of International Congress of Mathematical Society, pages 545-579, 2006.
[29] G. Stormo. DNA binding sites: representation and discovery. Bioinformatics, 16:16-23, 2000.