iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** IV **Month of publication:** April 2018

DOI: http://doi.org/10.22214/ijraset.2018.4746

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue IV, April 2018- Available at www.ijraset.com

A Survey on Longest Common Subsequence

Deena Nath[1], Jitendra Kurmi[2], Vipin Rawat[3]
[1, 2, 3]*Department of Computer Science, Babasaheb Bhimrao Ambedkar University Lucknow*

Abstract: The main aim of this paper is to present a general observation of popular longest common subsequence algorithms and study their functions in multiple applications. There are algorithms which calculate only the length but not the actual LCS and others are which do determine the actual LCS, we need a clear distinction among them in order to determine the complexity since space and time complexity rely on it.
Sequence matching is an approach to find a common subsequence among two or more sequences. The subsequence with the largest length is the LCS. This paper is a survey which briefly describes the successful attempts made to get the longest common subsequence of two strings. The comparisons of different algorithms are made and their performances are also analysed.
Keywords: Sequence matching, DNA sequence, Dynamic Programming, Longest Common Subsequence, Pattern Matching.

I. INTRODUCTION

Sequence comparison is a fundamental operation used in various application of computer science and information processing. A spelling error correction program tries to find the dictionary entry which resembles most a given word, in molecular biology as it can be used to compare two or more given sequences. Sequence comparison is especially beneficial for investigating genes.
In molecular biology sequence is represented as symbols of nucleotides. DNA and RNA are sequences of 4 characters Σ=(A,C,G,T) and Σ=(A,C,G,U) respectively.
During the study of expansion of long molecules which are generally proteins or nucleic acids. It is generally observed that researches have tried to do the construction of big set of correspondences or matches between such kind of two molecules.
The series of the sequence can be achieved by eliminating some elements of the strings by not disturbing the placement of the strings. A possible longest common subsequence is supposed to be searched from the sequences present in common between two of the sequences. To test the homology between two organisms we will have to compare the DNAs or protein sequences. A verification is required as a new sequence or homology when we discover a new biological sequence. It's a good way to know the exact sequence matching of the DNA in recapture of specific erudition like genetics, genetic disorder, parent-ship, medication, etc.

A. What is Longest Common Subsequence?

The LCS is an attempt of seeking the longest subsequence which is common to all sequence in a set of strings. Original sequences may not have continuous positions in subsequence to acquire. In information processing comparisons of data programs like molecular biology are involved in getting the idea of longest common subsequence problem.
In our subject, the maximum length subsequence LCS(X,Y) is the part of a common subsequence of string X and Y. The motivational factor of this biological LCS problem is that proteins and nucleic acids like DNA and RNA can be portrayed as sequences from a set of finite alphabet. It is obvious to find the closest common ancestors to compare two DNA sequences. The process of insertion of new symbols into representing a string, where ancestral substrings are the part of representing molecule. Thus, the length of the longest common subsequence of two strings is a rational standard of how nearby both strings are.

B. Length of LCS

The length of LCS of the two strings X and Y is denoted by r (X, Y), or when the input strings are known, by r.
Length of LCS (r)= c[m-1,n-1]

C. Dynamic Programming

On comparing greedy algorithm with Dynamic programming algorithms we found that greedy algorithm treated the solution as some sequence of steps and picks the locally optimal choice at each step but dynamic programming algorithm showed that it is the technique for solving problems with overlapping subproblems. One more thing was observed that greedy algorithm did not guarantee an optimal solution. Therefore, we reach to a conclusion that dynamic programming algorithm will examine the previously solved subproblems and provide the best and accurate solution by optimization.

In the 1950s "Richard Bellman" an American mathematician gave the concept of Dynamic programming which is a technique to solve problems by overlapping smaller subproblems in order to evade recompilation. Lcs(X,Y) is typically solved with the dynamic programming technique and filling an m x n table.

II. LITERATURE REVIEW

The research on literature of LCS algorithms renders a comprehensive range of opportunities to attune the system to enhance the overall performance.

1) The Levenshtein distance is a method to find the edit distance between two sequences by counting the number of insertions, deletions, and substitutions needed to makes valuable changes in the string. Wagner and Fischer in the year 1974 introduced an algorithm using the concept of a matrix in order to get the solution for this problem with dynamic programming. This algorithm just gives the length of LCS as a result but not the LCS.

2) Using divide-and-conquer strategy and dynamic approach altogether, Hirschberg in 1975 introduced a concept to find LCS. Now, the problem is divided recursively into smaller and easier subproblems and solved individually and then combining their solution to solve the whole problem. Firstly, the matrix is traversed in the forward direction and then it is traversed in the reverse direction. The time and space complexity of this algorithm is O(mn) and O(m) respectively[2].

3) Dominant matches was another approach given by Hirschberg in 1977. The complexity of this algorithm is O (rn+nlogn) here r is the total number of ordered doublets at which the matching of two strings is performed.

4) J.W. Hunt and T.G. Szymanski's in the year 1977, stated that computing LCS from two strings is equivalent to find the longest monotonically increasing path in the graph, where xi = yj. Hence introduced an algorithm on the same concept used in determining the longest increasing path.

5) In this case, we select consecutive symbols from X and traverse Y in reverse order in order to search for matches until all the elements of Y has been traversed. The solutions to these subprobelms are then merged in order to get the Longest Common Subsequence as a result. The process gets terminated as we get the optimal solution. The time complexity of this algorithm is O(n (m - r)). This method is used for fast processing to fetch the LCS when the given strings are of a large length and are suitable for similar texts.

6) Another algorithm was published by Apostolico, A. & Guerra in 1987, an alternative to support the framing of the LCS. The time complexity of this algorithm is O(mlogn+dlog (2mn/d)) where d is the smallest distance of the next nearest common elements. The closest occurrence of elements in Y is a representation of each symbol.

7) In the year 1990 Wu ET puts an effort to minimize the edit distance problem to reduced edit distance and apply it to find LCS. This reduced edit distance in spite of calculating the actual edit distance, it is directed to reduce the number of deletion. Hence, this algorithm is suitable for small strings. The time complexity of this algorithm is O(n (m-r)).

8) Rick in 1955 introduced an algorithm based on advancing from contour to contour which was compared with other existing algorithms and proved to be better among them. Contour is a region where the matching values are found & it was bounded by broken lines. The concept of dominant matches is used in this algorithm achieved by a class of dualization. The time complexity of this algorithm is O(min {(rm), r (n-r)}).

A. Some Related Works

1) The string editing problem is as attempt of transforming edit operation to minimise the cost. There is a specific cases of string editing that involves the longest common subsequence problem [1].

2) This is an attempt to use structures like trees and arrays in order to minimise the problem of storing, transforming and fetching information. The main aim of pattern matching is to fetch all the information of that unit[7].

3) In this paper an attempt is made to analyse all the aspects involved in LCS and compare it with all other algorithms meant for finding the longest common subsequence[4].

III. THE EXISTING APPROACH

The The common and popular algorithm to find LCS between two strings is the improved dynamic programming algorithm. A DNA is a linear sequence as a basic structure of x1, x2, x3... xm of nucleotide. Each x_i is recognized by the set of the four alphabets which are: {A, T, G, C}. Applications in the field of bioinformatics require comparing the DNA of various organisms. A sample of DNA comprises a sequence of molecules known as bases, which are possibly Adenine, Cytosine, Guanine, and Thymine i.e. {A, C, G, T}.

1) *Step 1:* Identifying a longest common subsequence.

Theorem: The optimal substructure property of LCS problem.

Let, X= (x_1, x_2, x_3... x_m) and Y= (y_1, y_2, y_3…, y_n) be sequences. Let C= (c_1, c_2, c_3…, c_i) be any LCS of X & Y. The theorem states three cases given below:

a) If $x_m = y_n$, then $c_i = x_m = y_n$ and LCS of X_{m-1} and Y_{n-1} is C_{i-1}.

b) If $x_m \neq y_n$, then $c_i \neq x_m$ and LCS of X_{m-1} and Y is C.

c) If $x_m \neq y_n$, then $c_i \neq y_n$ and LCS of X and Y_{n-1} is C.

2) *Step 2:* Generate a recursive loop for LCS solution

$$C[i, j] = \begin{cases} 0 & if\ i = 0\ or\ j = 0 \\ C[i-1, j-1]+1 & if\ i, j > 0\ and\ a_i = b_j \\ \max(C[i, j-1], C[i-1, j]) & if\quad i, j > 0\ and\ a_i \neq b_j \end{cases}$$

3) *Step 3:* Calculating the length of LCS

The length of LCS of strings X and Y is denoted by r (X, Y), or when the input strings are known, by r.

Length of LCS (r)= C[m-1,n-1].

Algorithm

a) Get two string X and Y from the user of length m and n

b) Add prefix to both the strings

c) Take two double dimensional array C[0..m, 0..n] for storing entries of matching subsequense and B[1..m, 1..n] for back tracking

d) Insert 0(zero) for all the entries of first row

e) Insert 0(zero) for all the entries of first column

f) In a recursive nested loop if $X_i=Y_j$ then C[i,j]=C[i-1,j-1]+1 and B[i,j]="\"

g) elseif C[i-1,j]>= C[i,j-1] then C[i,j]=C[i-1,j] and B[i,j]="↑

h) otherwise C[i,j]=C[i-1,j-1]+1 and B[i,j]="←"

i) return both the matrices C and B

Figure. 1 illustrate the length of LCS on the sequences X = (C, B, A, B, D, C, B) and Y =(B, D, A, C, B, C).

4) *Step 4:* Backtracking to construct an LCS

In the two given strings X and Y, let's say C be a common subsequence between the two strings if the subsequence C exists in both the strings[5]. In the given figure X= (C,B,A,B,D,C,B) and Y= (B,D,A,C,B,C), the sequence (B,A,C) is common but not the longest common subsequence of X and Y. however, since it has length 3 whereas the sequence (B,A,B,C), is also exactly same in both X and Y, with length 4. Hence the sequence (B,A,B,C) is an LCS of X and Y, same as the sequence (B,D,C,B), since X and Y have no common subsequence of length more than 4.

We are given two sequences X and Y and we wish to have a maximum-length common subsequence which can be solved using dynamic programming.

		j	0	1	2	3	4	5	6
i			y_j	B	D	A	C	B	C
0	x_i		0	0	0	0	0	0	0
1	C		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	A		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	C		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Fig. 1 LCS matrix of X and Y

With this procedure, we meet the elements of this LCS in reverse order by traversing the longest common subsequence using backtracking. In order to get an appropriate result, the following recursive method is used to print the LCS of X and Y.

STEP 1- if i==0 or j==0 then terminate the loop
STEP 2- if b[i,j]=="\" then i=i-1 and j=j-1
STEP 3- PRINT x_i
STEP 4- elseif b[i,j]=="↑" then i=i-1
STEP 5- otherwise j=j-1
STEP 6- Goto STEP 1

In Figure. 1, this procedure gives BABC as a result. The running time of this procedure is O(m+n), since it has nested loop of i and j which decrease by 1 in each iteration.

IV. ISSUES WITH PREVIOUS ALGORITHMS

1) Sub Problem One: To eliminate the prefixes (an empty subsequence) added in both the sequences or strings X and Y.

Previously we used to add prefix to both the sequences.

For example, if X= {C, B, A, B, D, C, B} and Y = {B, D, A, C, B, C}.

We are supposed to add Prefix to both the Strings

i.e. X= {Φ, C, B, A, B, D, C, B} and Y = {Φ, B, D, A, C, B, C}.

But, in the revised Algorithm the prefixes of the sequences can be eliminated.

2) Sub Problem Two: To create an LCS matrix (m, n) instead of (m+1, n+1) by eliminating the first row and first column (initialized 0), in order to reduce the iterations.

We add prefixes to both the sequences in order to get the initial values as zero(0) to start tracing both the sequence and find the common elements in both. Hence we have to take the matrix of (m+1, n+1) size.

But, in the revised Algorithm we don't need to initialize the first row and column by 0 separately and hence we can remove the first row and column and reduce the size of the matrix.

0	ϕ	A	B	C	A	B
ϕ	0	0	0	0	0	0
A	0	1	1	1	1	1
A	0	1	1	1	2	2
B	0	1	2	2	2	3
A	0	1	2	2	3	3
C	0	1	2	3	3	3
A	0	1	2	3	4	4

LCS ALGORITHM

0	ϕ	A	B	C	A	B
ϕ	0	0	0	0	0	0
A	0	1	1	1	1	1
A	0	1	1	1	2	2
B	0	1	2	2	2	3
A	0	1	2	2	3	3
C	0	1	2	3	3	3
A	0	1	2	3	4	4

REVISED LCS ALGORITHM

Fig. 2 Comparison between two matrices

3) Sub Problem Three: To reduce Time Complexity.

After reducing the iteration, most probably we can reduce the time complexity.

In the LCS algorithm, for example, we can eliminate the b table altogether. Each c[i,j]entry depends on only three other c table entries: c[i-1,j -1], c[i-1,j], and c[i,j-1]. Given the value of c[i,j], we can determine in O(1)time which of these three values was used to compute c[i,j], without inspecting table b. Thus, we can reconstruct an LCS in O(m+n) time using a procedure similar to PRINT-LCS.

Although we save Θ(mn) space by this method, the auxiliary space requirement for computing an LCS does not asymptotically decrease, since we need Φ(mn) space for the c table anyway.

4) Sub Problem Four: To reduce Space Complexity by improving the Code.

We can, however, reduce the asymptotic space requirements for LCS-LENGTH, since it needs only two rows of table c at a time: the row being computed and the previous row.

It is stated in the text book that "this improvement works if we need only the length of an LCS; if we need to reconstruct the elements of an LCS, the smaller table does not keep enough information to retrace our steps in O(m+n) time"[6]. After modifying the algorithm, I would say that it is possible to retrace the table even after eliminating the first row and first column of the table, and making it smaller in O(m+n) time.

V. PROPOSED APPROACH

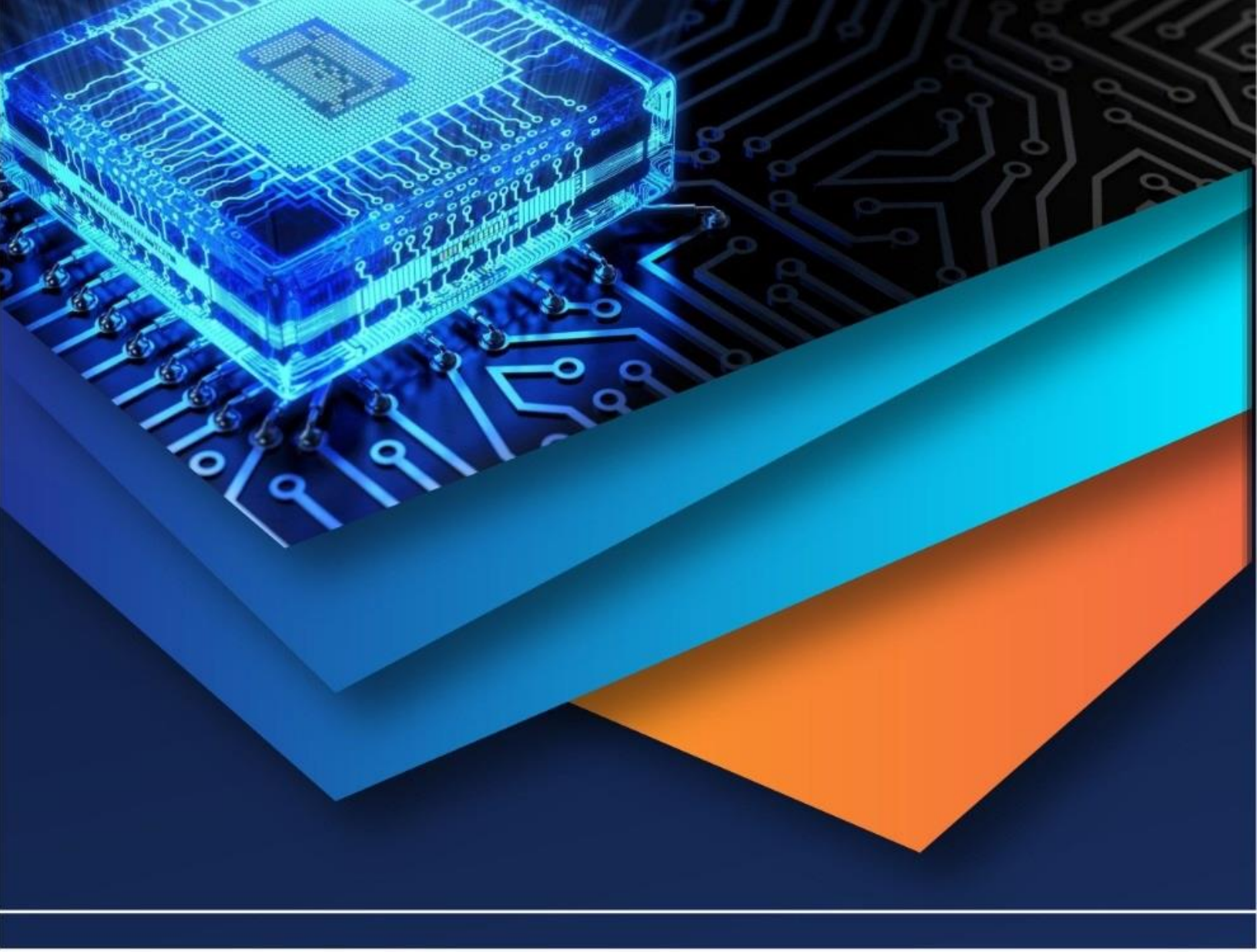
1. To eliminate the prefixes (an empty subsequence) added in both the sequences or strings X and Y.
2. To create an LCS matrix (m, n) instead of (m+1, n+1) by eliminating the first row and first column (initialized 0), in order to reduce the iterations.
3. After reducing the iteration, most probably we can reduce the time complexity.
4. To reduce Space Complexity by improving the Code.

VI. CONCLUSION & FUTURE SCOPE

In this paper we have proposed a comprehensive observation of the LCS algorithms from sequences of DNA or protein to analyse the problems and get the key idea which is helpful to enhance its performance. I feel that there is still some scope to improve the time complexity and performance of this approach through different data structure. Once the algorithm is proposed, you observe a scope of an amendment in the algorithm to enhance the performance.

REFERENCES

[1] A. Apostolico, String editing and longest common subsequences, in: G. Rozenberg, A. Salomaa (Eds.), Linear Modeling: Background and Application, in: Handbook of Formal Languages, Vol. 2, Springer-Verlag, Berlin, 1997, pp. 361–398.
[2] D.S. Hirschberg, Serial computations of Levenshtein distances, in: A. Apostolico, Z. Galil (Eds.), Pattern Matching Algorithms, Oxford University Press, Oxford, 1997, pp. 123–141.
[3] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1974) 168–173.
[4] L. Bergroth, H. Hakonen, T. Raita, A Survey of Longest Common Subsequence Algorithms (IEEE Computer Society Washington, DC, USA ©2000).
[5] SAM Rizvi, P. Agarwal, A New Index-Based Parallel Algorithm for finding Longest Common Subsequence in Multiple DNA Sequences. International Conference in Cognitive Systems, 2005.
[6] T. H. Corman, R L. Rivest, Charles E.Leiserson, C.Stein, "Dynamic Programming", Introduction to Algorithm**,** Third Edition, Cambridge, MA: MIT Press, 2010, Ch. 15, sec. 15.4, pp.390-397.
[7] By Mikhail J. Atallah, General pattern matchings (Handbook of Algorithms and Theory of Computation, CRC, Boca Raton, FL, 1998)



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY