ISSN 2349-6193
Impact Factor: 2.805

IJESMR

International Journal OF Engineering Sciences &Management Research

AN EFFICIENT ALGORITHM FOR SOLVING LONGEST COMMON SUBSEQUENCE PROBLEM

U. Udayakumar [*1] & Dr. A. Murugan[2]
[*1]Research Scholar PG and Research Department of Computer Science, Dr.Ambedkar Government Arts College, Chennai - 600 039, India.
[2]Associate Professor PG and Research Department of Computer Science, Dr.Ambedkar Government Arts College, Chennai - 600 039, India

Keywords: *No of data, Longest Common Subsequence, Dynamic Programming.*

ABSTRACT

Sequence alignment is an important problem in computational biology and finding the longest common subsequences(LCS) of multiple string sequences is an essential and effective technique in sequence alignment problem. In this paper solving LCS problem using dynamic programming method have been proposed with reduced time and space complexity. We developed an efficient algorithm to solve the longest common subsequence problem, using a new technique that improved the LCS algorithm with time complexity of O(m*n). The algorithm takes the advantage over the existing algorithm and improving the performance and tested on randomly generated sequences of different length.

INTRODUCTION

The simplest form of a sequence similarity analysis is the Longest Common Subsequence (LCS) problem, where we eliminate the operation of substitution and allows only insertion and deletion. A subsequence of a string v is simply a sequence of characters from v. The longest common subsequence (LCS) problem is one of the classical and well-studied problem in computer science which has extensive applications in diverse areas ranging from spelling error corrections to molecular biology: a spelling error correction problem tries to find the dictionary entry which resembles most a given word; in a file archive we want to store several versions of a source program compactly by storing only the original version and construct all other versions from the differences to the previous one. In Molecular Biology, we want to compare DNA or protein sequences to learn how much similarity between them. All these cases can be seen as an investigation for the "closeness" among strings. And an obvious measure for the closeness of strings is to find the maximum number of identical characters in them preserving the order of the characters. This, by definition, the longest common subsequence of the strings.

Problem 1 ("LCS"). Given two strings X and Y, we want to find out the Longest Common Subsequence of X and Y. In this paper we are proposed in several variants of LCS Problem. In the rest of this section we formally define the new variants and given some examples. We assume that the two given strings are of equal length. But the results can be easily extended to handle two strings of different length.

Definition-1 ("Correspondence Sequence"). Given a string X[1..n] and a subsequence S[1...r] of X, we define the correspondence sequence of X and S, C(X, S) = C[1] C[2] . . .C[r] to be the strictly increasing sequence of integers.

Definition-2 ("Fixed Gapped Correspondence Sequence"). A correspondence sequence of a string X of length n and one of its subsequences S of length r is said to be a Fixed Gapped Correspondence Sequence with respect to a given integer K if and only if we have $C[i] - C[i - 1] \leq K + 1$ f or all $2 \leq i \leq r$. We sometimes use CFG(K) to denote a Fixed Gapped Correspondence Sequence with respect to K.

Definition-3 ("Elastic Gapped Correspondence Sequence"). A correspondence sequence of a string X of length n and one of its subsequences S of length r is said to be an Elastic Gapped Correspondence Sequence with respect to a given integer K1 and K2, K2 > K1 if and only if we have $K1 < C[i] - C[i - 1] \leq K2 + 1$ for all
$2 \leq i \leq r$. We sometimes use CEG(K1,K2) to denote an Elastic Gapped Correspondence Sequence with respect to K1 and K2.

IJESMR
International Journal OF Engineering Sciences &Management Research

Definition-4 ("Fixed Gapped Common Subsequence and Elastic Gapped Common Subsequence"). Suppose that we are given two strings X[1...n] and Y [1...n] and an integer K. A common subsequence S[1...r] of X and Y is a Fixed Gapped Common Subsequence, if there exist Fixed Gapped Correspondence Sequences CFG(K)(X, S) and CFG(K)(Y, S). Elastic Gapped Common Subsequences can be defined analogously

LITERATURE REVIEW

The longest common subsequence problem for k strings (k > 2) was first shown to be NP-hard [25] and later proved to be hard to be approximated [19]. The restricted but probably the more studied problem that deals with two strings has been studied extensively [26,18,17,16]. The classic dynamic programming solution to LCS problem, invented by Wagner and Fischer [35], has O(n2) worst case running time. Masek and Paterson [26] improved this algorithm using the "Four-Russians" technique to reduce the worst case running time to (n2/logn)2, Since not much improvement in terms of n can be found in the literature. However, several algorithms exist with complexities depending on other parameters. For example Myers in [25] and Nakatsu et al. in [24] presented an O(n) algorithm, where the parameter D is the simple Liechtenstein distance between the two given strings [22]. Another interesting and perhaps more relevant parameter for this problem is R. Hunt and Szymanski [18] presented an algorithm running in O((R + n) log n). They have also cited applications where R ~ n and thereby claimed that for these applications the algorithm would run in O(nlogn) time. For a comprehensive comparison of the well-known algorithms for LCS problem and study of their behaviour in various application environments are found in [9]. In [8], the authors find the LCS using Java threads, they claim O(m*n) for best case.

LCS ALGORITHM

In LCS problem, where we eliminate the operation of substitution and allow only insertions and deletions. A Subsequence of a string v is simply an ordered sequence of characters from v. For example, if v = ATTGCTA, then AGCA and ATTA are subsequences of v whereas TGTT and TCG are not. Where ATGC denotes Adenine, Thymine, Cytosine. Formally, we define the common subsequence of strings.

$1 \leq i1 \leq i2 < . . . ik \leq n$ (1)

And a sequence of positions in w,

$1 \leq j1 \leq j2 < . . . jk \leq m$ (2)

such that the symbols at the corresponding positions in v and w coincide:

$vit = wjt \text{ for } 1 \leq t \leq$ (3)

for example, TCTA is common to both ATCTGAT and TGCATA

Alignment

A **T**–**C**–**T** G **A** T
–**T** G **C** A **T**–**A**–

Let $s_{i,j}$ to be length of an LCS between $v_1 ... v_i$, the i-prefix of v and $w_1 ... w_j$, the j-prefix of w. Clearly, $s_{i,0} = s_{0,j} = 0$ for all $1 \leq i \leq n$ and $1 \leq i \leq n$ one can see that $s_{i,j}$ satisfies the following recurrence:

$$s_{i,j} = \max\begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1}+1, \quad \text{if } v_i=w_j \end{cases} \qquad (4)$$

Note that one can "rewrite" these recurrencesby adding some zeros here and there as

$$s_{i,j}=\max\begin{cases} s_{i-1,j}+0 \\ s_{i,j-1}+0 \\ s_{i-1,j-1}+1, \quad \text{if } v_i=w_j \end{cases} \qquad (5)$$

This recurrence for the LCS computation is like the recurrence given at the end, if we were to build a particularly gnarly version of Manhattan and gave horizontal and vertical edges weights of 0, and set the weights of diagonal edges equal to + 1.

DNA OPERATIONS BASED ALGORITHMS

In this paper, we proposed new approaches to study Longest Common Subsequences algorithms searches for all common sequence string in different input pattern sequences. We propose an efficient algorithm added to s distance from the last matched for computing similarity between two strings. Let s(v,w) be the length of the longest common subsequence of v and w, then the edit distance between v and w under the assumption that only insertions and deletions are allowed is d(v,w) = n +m - 2s(v,w), and corresponds to the minimum number of insertions and deletions needed to transform v into w. (bottom) presents an LCS of length 4 for the strings v = ATCTGAT and w = TGCATA and a shortest sequence of two insertions and three deletions transforming from v into w. Every common subsequence corresponds to an alignment with no mismatches. This can be obtained simply by removing all diagonal edges from the edit graph whose characters do not match, thus transforming it into a graph.

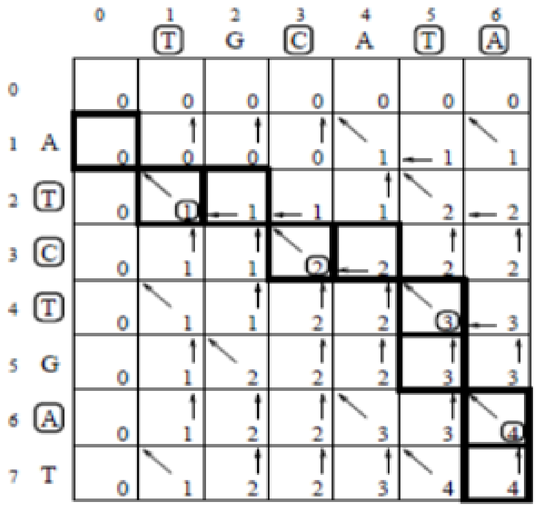


Fig 1 :Computing similarity s(V,W) = 4 V and W have a subsequence TCTA in common

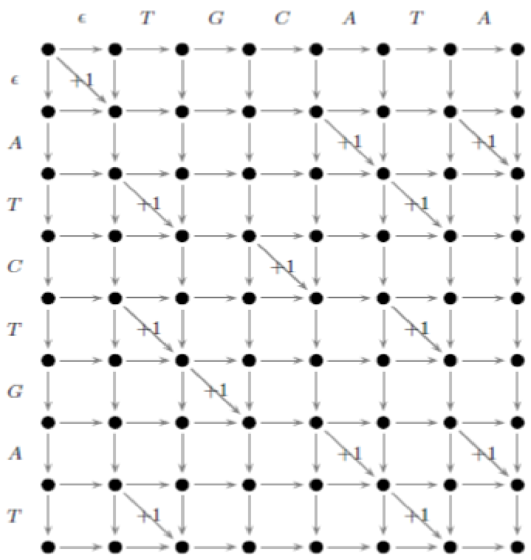


Fig 2 :LCS Edit Graph

PROPOSED ALGORITHM

We propose an efficient algorithm for computing similarity between two strings. Let s to represent our dynamic programming table, the data structure that we use to fill in the dynamic programming recurrence. The length of an LCS between v and w can be read from the element(n,m) of the dynamic programming table, but to reconstruct the LCS from the dynamic programming table, one must keep some additional information about which of the three quantities, si1,j , si,j1, or si1,j1 + 1, corresponds to the maximum in the recurrence for si,j.The computation of the similarity score s(v,w) between v and w, while the table on the right presents the computation of the edit distance between v and w under the assumption that insertions and deletions are the only allowed operations.

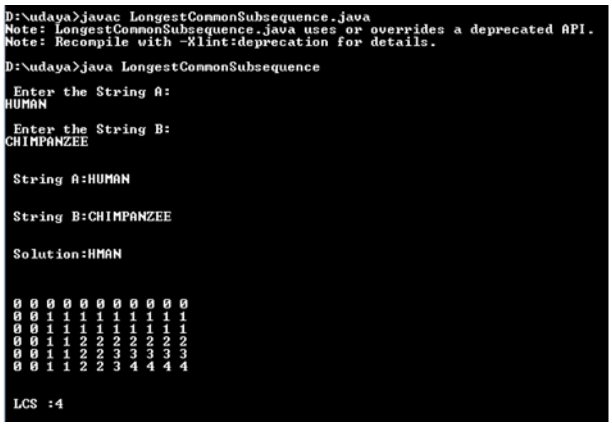
Algorithm-1 depicts the process of two string sequences in efficient multiple longest common subsequences

```
functionLCS(A,B)          Where A – array, B- array
    LCS[][] := [A.length +1] [A.length +1]
    soln[][] := [A.length +1] [A.length +1]
    res := null;
    for i := 0 to B.lengthdo
        $LCS_{0,i}$:= 0
        $soln_{0,i}$:= “0”
    end for
    for i := 0 to A.lengthdo
        $LCS_{i,0}$:= 0
        $soln_{i,0}$ := “0”
    end for

    for i := 1 to A.lengthdo
        for j:= 1 to B.lengthdo

            if A[i-1] == B[j-1] then
                $LCS_{i,j}$ := LCS[i-1][j-1] + 1
                $soln_{i,j}$:= “Diagonal”
            else
                $LCS_{i,j}$ := max(LCS[i-1][j]
            endif
            if$LCS_{i,j}$== $LCS_{i-1,j}$ then
                $soln_{i,j}$:= “Top”
            else
                $soln_{i,j}$:= “Left”
            endif
        endfor
    endfor
end function
```

Algorithm-2 Backtracking Pointers and stored three values up, right and diagonal in an Multidimensional array

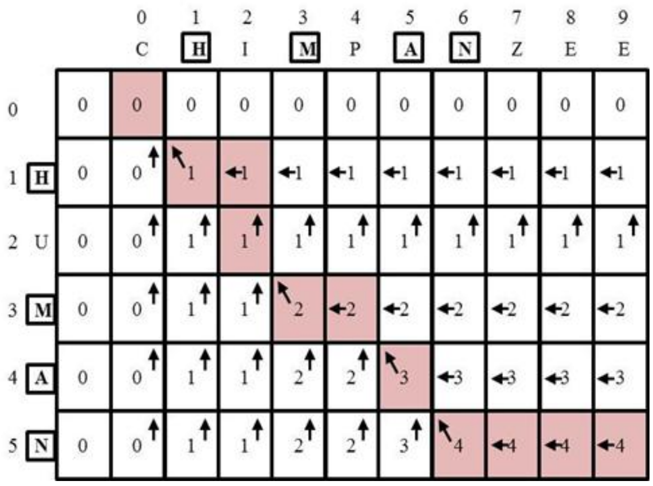
```
while x! := “0” then
        if$soln_{a,b}$== “Diagonal” then
            a := a - 1
            b := b - 1
        else
            b := b - 1
            a := a - 1
        endif
        return LCS[A.length][B.length]
end while
```

String A : HUMAN
String B : CHIMPANZEE



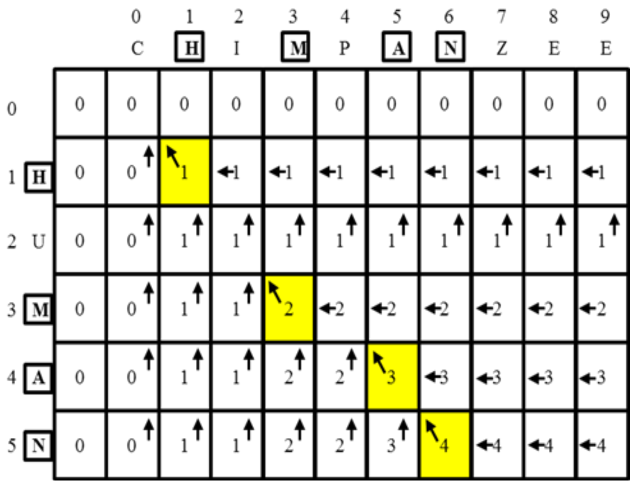
```
D:\udaya>javac LongestCommonSubsequence.java
Note: LongestCommonSubsequence.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

D:\udaya>java LongestCommonSubsequence

 Enter the String A:
HUMAN

 Enter the String B:
CHIMPANZEE


 String A:HUMAN


 String B:CHIMPANZEE


 Solution:HMAN



 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1
 0 0 1 1 2 2 2 2 2 2 2
 0 0 1 1 2 2 3 3 3 3 3
 0 0 1 1 2 2 3 4 4 4 4


 LCS :4
```

Fig 3 :Output for Efficient LCS Algorithm



		0 C	1 H	2 I	3 M	4 P	5 A	6 N	7 Z	8 E	9 E
0	0	0	0	0	0	0	0	0	0	0	0
1 H	0	0↑	↖1	←1	←1	←1	←1	←1	←1	←1	←1
2 U	0	0↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑
3 M	0	0↑	1↑	1↑	↖2	←2	←2	←2	←2	←2	←2
4 A	0	0↑	1↑	1↑	2↑	2↑	↖3	←3	←3	←3	←3
5 N	0	0↑	1↑	1↑	2↑	2↑	3↑	↖4	←4	←4	←4

Fig 4 :Multidimensional Array representation for efficient LCS Algorithm



		0 C	1 H	2 I	3 M	4 P	5 A	6 N	7 Z	8 E	9 E
0	0	0	0	0	0	0	0	0	0	0	0
1 H	0	0↑	↖1	←1	←1	←1	←1	←1	←1	←1	←1
2 U	0	0↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑
3 M	0	0↑	1↑	1↑	↖2	←2	←2	←2	←2	←2	←2
4 A	0	0↑	1↑	1↑	2↑	2↑	↖3	←3	←3	←3	←3
5 N	0	0↑	1↑	1↑	2↑	2↑	3↑	↖4	←4	←4	←4

Fig 5 :Multidimensional Array representation for diagonal edges

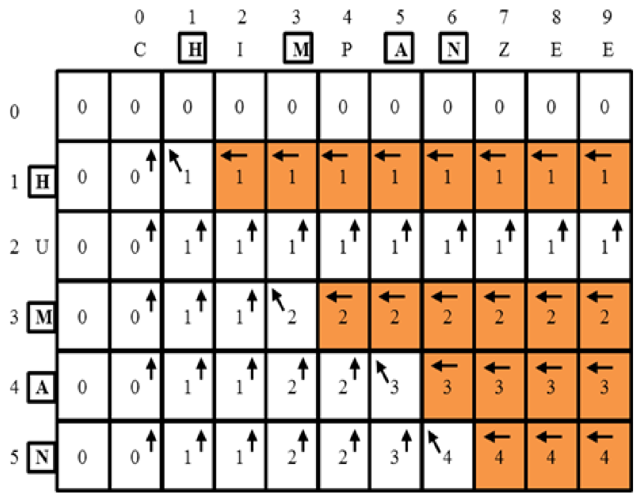


Fig 6 :Multidimensional Array representation for Left edges

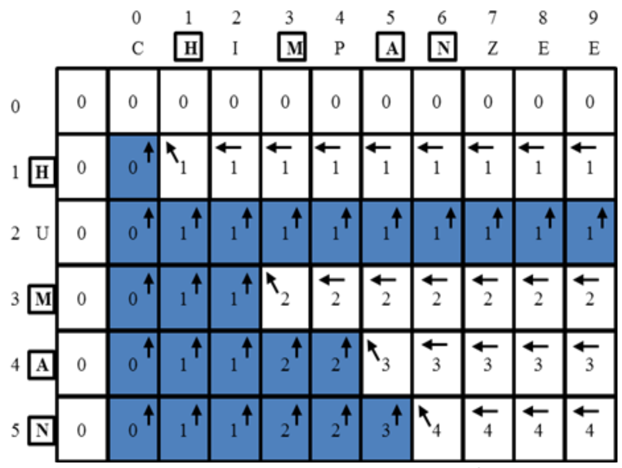


Fig 7 :Multidimensional Array representation for Top edges

Time Complexity

The time complexity of Algorithm1 can be analyzed in two phases is and it is O(m*n)
Therefore at its best case
T(LCS) = max((O((n/L) and O(levelnumber)), O(|LCS|)).
At its average and worst case
T(LCS) = max((O((n/M) + O((n/L)+n)) and O(levelnumber)), O(|LCS|).

CONCLUSION AND FUTURE ENHANCEMENT

In this paper, we proposed an algorithm for finding similarity between two strings. It calculates the longest common subsequence (LCS) by avoiding unnecessary comparisons that reduces its performance. The running time is better than dynamic programming based algorithm and it is due to time control parameter. The proposed algorithm is tested on 50 samples with two input strings and randomly selected in pentium processor machines as well as is shows good results for the algorithm. In future the algorithms is implemented in Multiple Longest Common sequences, which will also have many applications

REFERENCES

1. *Neil C.Jones and PavelA.Pevzner, "An Introduction to Bioinformatics Algorithms" (2004)*
2. *Nikhil Bansal, Moshe Lewenstein, Bin Ma and Kaishong Zhang, "On the longest common rigid subsequence problem, Algorithmica", (56), 2010, 270-280.*
3. *Jiaoyun Yang, Yun Xu, Yi Shang, "An Efficient Parallel Algorithm for Longest Common Subsequence Problem on GPUs", World Congress Engineering, 2010.*
4. *Maier. D, "The Complexity of some problems on subsequences and super sequences", ACM, (25), 1978, 332-336.*
5. *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, 1995, 402-410.*
6. *Isabelle da Piedade, Man-Hung Eric Tang, and Oliver Elemento, "DISPARE: discriminative pattern refinement for position weight matrices". BMC Bioinformatics, 10(388): 2009, 1471-2105.*
7. *B.Lavanya and A.Murugan, "Discovery of Longest Increasing Subsequences and its Variants Using DNA Operations", International Journal of Engineering and Technology (IJET),5(2), 2013, 1169-1177.*
8. *B.Lavanya and A.Murugan, "Mining Longest Common Subsequence and other related patterns using DNA operations", International Journal of Computer Application, (18), 2012, 38-44.*
9. *B.Lavanya and A.Murugan, "Discovering sequence motifs of different patterns parallel using DNA operations", International Journal of Computer Applications, 2011.*
10. *A.Murugan and B.Lavanya, "A DNA Algorithmic approach to solve GCS Problem". Journal of Computational Intelligence and Bioinformatics. 3(2), 2010, 239-247.*
11. *Nikhil Bansal, Moshe Lewenstein, Bin Ma, and Kaishong Zhang, "On the longest common rigid subsequence problem". Algorithmica, 2010, 270-280.*
12. *Hirosawa et al, "Comprehensive study on iterative algorithms of multiple sequence alignment". Computational Applications in Biosciences, 1995, 13-18.*
13. *Neuwald.A.F and Green.P, "Detecting patterns in protein sequences". Journal of Molecular Biology, 1994, 698-712.*
14. *Mahdi Esmaieli and Mansour Tarafdar, "Sequential Pattern Mining from multidimensional sequence data in parallel", International journal of Computer theory and engineering, 2010, 730-733.*
15. *Smith. T. F. and Waterman. M. S, "Identification of common molecular subsequences". Journal of Molecular Biology, 1981 147:195- 197.*
16. *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, 1994, 83-98.*
17. *Neville-Manning. C. G., Sethi. K .S., Wu. D.,andBrutlag. A. D, "Enumerating and ranking discrete motifs". Proceedings of Intelligent Systems for Molecular Biology, 1997, 202-209.*
18. *Stormo. G, "DNA binding sites: representation and discovery". Bioinformatics, 2000, 16:16-23*
19. *Kyle Jensen. L., Mark Styczynski. P., IsidoreRigoutsos, and Gregory Stephanopoulos. V, "A generic motif discovery algorithm for sequential data". Bioinformatics, 22(1) 2006, 21-28.*
20. *Wang. L. and Jiang. T, "On the complexity of multiple sequence alignment". Journal of Computational Biology, 1994, 337-348.*
21. *Nan Li and Tompa. M, "Analysis of computational tools for motif discovery". Algorithms of molecular biology, 2006, 1-8.*
22. *Lo. D.andKhoo. S. D. Liu, "Efficient mining of iterative patterns for software specification discovery". International Conference.on Knowledge Discovery and Data Mining, 2007, 460-469.*
23. *Annila. H. M., Toivonen. H., and Verkamo. A.I, "Discovery of frequent episodes in event sequences". Data Mining and Knowledge Discovery, 1(3), 1997, 259-289.*
24. *Martinez. M, "An efficient method to find repeats in molecular sequences". Nucleic Acid Research, 1983, 4629-4634.*
25. *Suyama. M., Nishioka. T., and Junichi. O, "Searching for common sequence patterns among distantly related proteins". Protein Engineering, 1995,1075-1080.*
26. *Smith. H. O., Annau. T. M., and Chandrasegaran.S, "Finding sequence motifs in groups of functionally related proteins". Proceedings of National Academy (USA), 1990, 826-830.*
27. *Bin Ma, "A polynomial time approximation scheme for the closest substring problem". LCNS Springer, 2000, 99-107*