International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 7 57 – 60

Sequence Similarity between Genetic Codes using Improved Longest Common Subsequence Algorithm

A. Murugan
Associate Professor, PG & Research Department
of Computer Science
DrAmbedkar Government Arts College,
Vyasarpadi, Chennai – 600 039, Tamilnadu.
e-mail: amurugan1972@gmail.com

U. Udayakumar
M.Phil Research Scholar, PG & Research Department
of Computer Science
DrAmbedkar Government Arts College,
Vyasarpadi, Chennai – 600 039, Tamilnadu.
e-mail: udaya.vijay13@gmail.com

Abstract— Finding the sequence similarity between two genetic codes is an important problem in computational biology. In this paper, we developed an efficient algorithm to find sequence similarity between genetic codes using longest common subsequence algorithm. The algorithm takes the advantages over the edit distance algorithm and improves the performance. The proposed algorithm is tested on randomly generated DNA sequence and finding the exact DNA sequence comparison. The DNA genetic code sequence comparison can be used to discover information such as evolutionary divergence and ways to apply genetic codes from one DNA sequence to another sequence.

Keywords-*Dynamic Programming, DNA sequence comparison, DNA Similarity algorithms, Longest Common Subsequence.*

*****

I. INTRODUCTION

DNA is the hereditary material found in almost all organisms [1]. DNA resides in every cell in the body of organism. The DNA is a double helix like structure made up of two twisted strands. A single strand carries a nucleotide bases, these bases are adenine (A), guanine (G), cytosine (C), thymine (T). These chemical letters are in the form of triplets which defines a meaningful code. The triplets give arise to specific amino acid which later helps in the formation of protein[2]. An important property of DNA is that it can replicate, or make copies of itself. Each strand of DNA in the double helix can serve as a pattern for duplicating the sequence of bases as shown in Fig.1. This is critical when cells divide because each new cell needs to have an exact copy of the DNA present in the old cell.

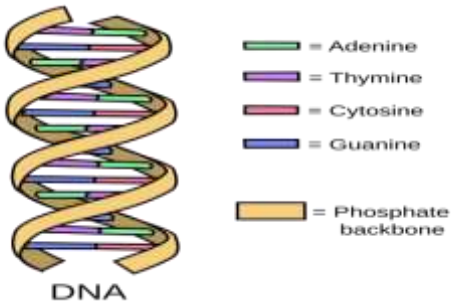

Figure 1. Structure of DNA

The knowledge of a DNA sequence and gene analysis can be used in several biological, medicine and agriculture research fields such as: possible disease or abnormality diagnoses, forensic science, pattern matching, biotechnology, etc[3][4]. The analysis and comparison studies for DNA sequences connected information technology tools and methods to accelerate findings and knowledge in biological related sciences.

II. LITERATURE SURVEY

There are many traditional pattern matching methodologies available in literature, they are Naive Brute force, Boyer Moore, Knuth Morris Pratt and Dynamic algorithms[2]. Pattern matching is used in various processes, like codon optimization which is carried out to enhance the efficiency of the DNA expression vectors used in DNA vaccination by increasing protein expression[5].

A. Naive Brute force

It is one of the simplest algorithm having complexity O(m*n). In this, first character of pattern P(with length m) is aligned with first character of text T (with length n). Then scanning is done from left to right. As shifting is done at each step it gives less efficiency[6].

B. Boyer-Moore Algorithm

It performs larger shift-increment whenever mismatch is detected. It differs from Naïve in the way of scanning. It scans the string from right to left; unlike Naive i.e. P is aligned with T such that last character of P will be matched to first character of T. If character is matched then pointer is shifted to left to very rest of the characters of the pattern[7][8]. If a mismatch is detected at character c, in T which is not in P, then P is shifted right to m positions and P is aligned to the next character after c. If c is part of P, then P is shifted right so that c is aligned with the right most occurrence of c in P. The worst time complexity is still O(m+n).

C. Knuth-Morris-Pratt

This algorithm is based on automaton theory. First a finite state automata model M is being created for the given pattern P. The input string T with Σ= {A, C, T, G} is processed through the model. If the pattern is present in text, the text is accepted otherwise rejected.

III. EFFICIENT LCS ALGORITHM

The longest common subsequence problem is to find a substring that is common to the two given strings. Given two sequences, find the length of the longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous [16].

In LCS problem, 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, Guanine, Cytosine. Formally, we define the common subsequence of strings[9].

$$1 \le i_1 \le i_2 < \ldots i_k \le n \qquad (1)$$

and a sequence of positions in w,

$$1 \le j_1 \le j_2 < \ldots j_k \le m \qquad (2)$$

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

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1}+1, \quad \text{if } v_i = w_j \end{cases} \qquad (3)$$

Note that one can "rewrite" these recurrences by adding some zeros

$$s_{i,j} = \max \begin{cases} s_{i-1,j}+0 \\ s_{i,j-1}+0 \\ s_{i-1,j-1}+1, \quad \text{if } v_i = w_j \end{cases} \qquad (4)$$

The LCS computation is like the recurrence given at the equations (3) and (4), if we were to build particularly gnarly version of Manhattan and gave horizontal and vertical edges weights of 0, and set the weights of diagonal edges equal to + 1[10].

The following is the efficient algorithm for solving longest common subsequence problem [15] that improves the time complexity and performance.

Algorithm Backtracking Pointers and stored three values up, right and diagonal in 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
        end if
    return LCS[A.length][B.length]
end while
```

The following Fig. 2 shows Multidimensional array representation of the LCS algorithm

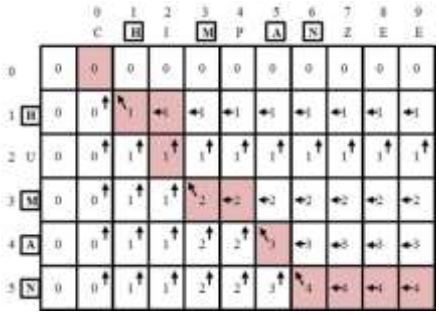

Figure 2. Multidimensional Array representation for efficient LCS Algorithm

The following Fig. 3 shows LCS Time Complexity and Performance

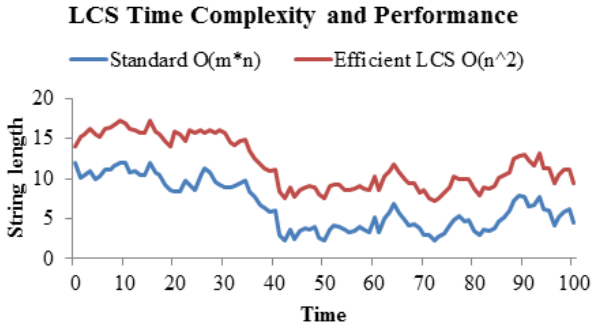

Figure 3. LCS Time Complexity and Performance

IV. DNA SEQUENCE SIMILARITY

We proposed an algorithm to find the similarity between DNA genetic code sequences. In DNA sequence similarity, find the matching percentage between DNA genetic codes. In this paper, we proposed a new method for DNA sequence similarity.

A modified Longest Common Subsequences algorithm is proposed in [15] and which is applied in the edit distance algorithm, which is an existing algorithm.

The following is the modified edit distance algorithm used to find the similarity between two genetic codes.

Algorithm 1: Edit Distance for DNA genetic code

```
Algorithm editDistance(s1,s2)
cost[] : = s2.length + 1
for i:= 0 to s1.length do
lastvalue := i
        for j:= 0 to s2.length do
                if(i == j)
                cost[j] : = j
                elseif(j > 0)
                        newValue : = costs[j - 1]
                        costs[j - 1] : = lastValue
                        lastValue : = newValue
                end if
end for
end for
end algorithm
```

 ISSN: 2321-8169
57 – 60

The following Fig. 4 explain the entire process of finding DNA sequence similarity

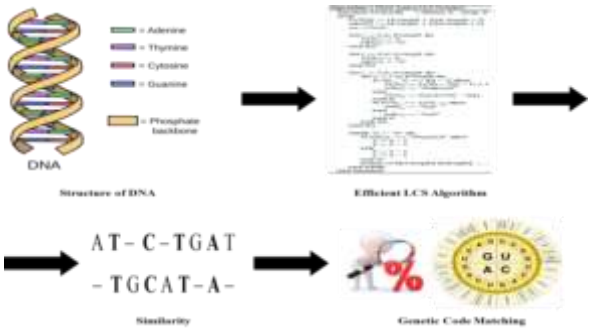

Figure 4. Implementing LCS Algorithm in DNA Genetic Code Sequence Similarity

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[14].

The following TABLE Ishows DNA genetic code sequence similarity

TABLE I. GENETIC CODE SIMILARITY BETWEEN SEQUENCES

Human DNA	Animals DNA	% of matching
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACGATTTGGGGGATGC TTCTGGCTC------A	100.00%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACGATTTGGGGGA TGCTTCTGGCTC------A-	100.00%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACGATTTGGGAGA TGCTTCTGGCTC-----G-	91.67%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCAGAATTTGGGGGA TGCTTCTGGCTC-----T-	88.89%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCAGAATTTGGGGGA TGCTTCTGGCTC-----T-	88.89%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCAGAATTTGGGGGA TGCTTCTGGCTC-----T-	88.89%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	ATCACAGTTGGGGGAT GCCACTGGCCT-----C-	75.00%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	ATCACAA-TTGGGGG-TGCCACGGTCCT-----C-	69.44%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	ATCACAATTTGGGGAA CACCACTGGCAT-----C-	69.44%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACAATTTGGAGGA TGTTACTGGCAT-----C-	77.78%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACAGTTTGGAGGA TGTTACTGACAT-----C-	72.22%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCATAGTTT---- GATTATATGGGCTT----- C-	58.33%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACAATTTGGGGGA TACTACTGGCAT-----C-	80.56%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACAGTTTAGGGGG TACTACTGGCAT-----C-	72.22%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GTCACAATTTAGGAAG TGCCACTGGCCT-----C-	72.22%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	GCCTCTCTTT---------- CTGCCCTGCAGGC-	33.33%
GTCACGATTTGGGGGATGC TTCTGGCTC------A-	---------- TGCTACTAATAT-----T-	36.11%

Algorithm 2: genetic code similarity

*Algorithm*similarity(s1, s2)

longer := s1, shorter := s2

if (s1.length() < s2.length())

longer := s2

shorter : = s1

longerLength := longer.length();

if (longerLength == 0)

return (longerLength - editDistance(longer, shorter)) / (double) longerLength;

end if

end algorithm

The following Fig. 6 shows the Output for Similarity Sequences between genetic codes

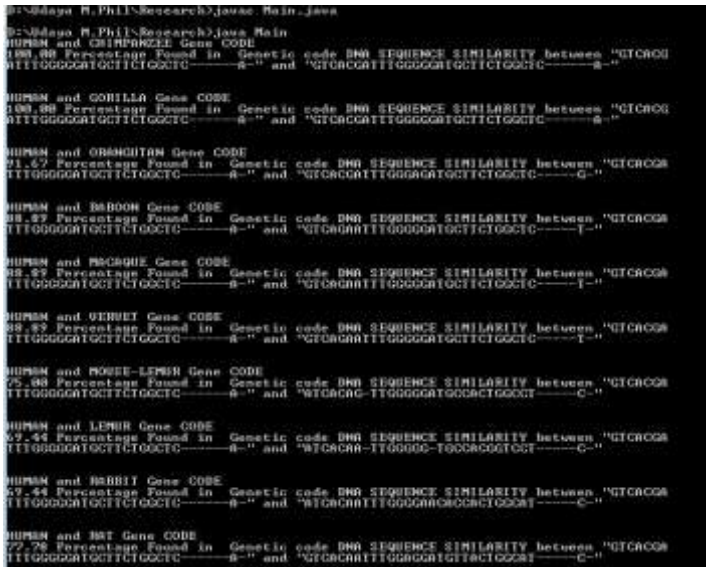

Figure 5. Output for DNA Genetic Code Similarity Sequences

A. *Time Complexity*

The Time Complexity of Sequence Similarity Algorithms

TC(DSSA) = max{TC(LCS), TC(EDA), TC(SS)}

TC(DSSA) = max{$O(n^2)$, $O(n^2)$, $O(n^2)$}

So the time complexity of the DNA Sequence Similarity Algorithm is $O(n^2)$

The best case and average case time complexity
T(n) = max((O(n^2) and O(levelnumber)), O(|n|).

The Worst case time complexity
T(n) = max((O(n^3) and O(levelnumber)), O(|n|).

The following Figure 7 shows Graphical Representation of DNA Genetic Code between Human and Animals

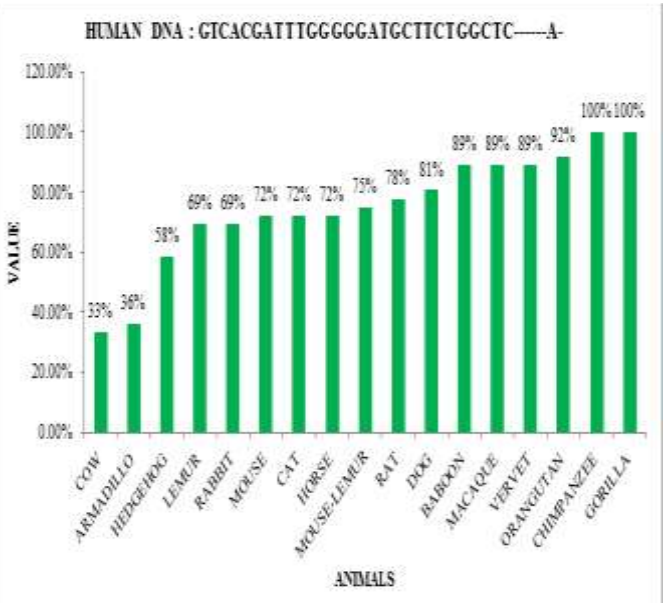

Figure 6. Graphical Representation of DNA Genetic Code Similarity Sequences

V. DNA SEQUENCE SIMILARITY

In this paper, we proposed an algorithm for finding similarity between two strings. It calculate the matching percentage between two strings by using longest common subsequence (LCS) and edit distance approach by avoiding unnecessary comparisons that reduces its time complexity. The running time is better than dynamic programming based algorithms and it is due to time control parameter. The DNA sequence similarity algorithm is tested on 50 samples with two input DNA genetic code sequence strings and selected randomly in Pentium processor machine is used and it shows good results. In future the algorithm is implemented in Multiple Longest Common Subsequences which will also have many applications.

REFERENCES

[1] Neil C.Jones and Pavel A.Pevzner, "An Introduction to Bioinformatics Algorithms" (2004)

[2] Jiaoyun Yang, Yun Xu, Yi Shang, “An Efficient Parallel Algorithm for Longest Common Subsequence Problem on GPUs”, World Congress Engineering, 2010.

[3] Maier. D, “The Complexity of some problems on subsequences and super sequences”, ACM, (25), 1978, 332-336.

[4] 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.

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

[6] Mahdi Esmaieli and Mansour Tarafdar, "Sequential Pattern Mining from multidimensional sequence data in parallel", International journal of Computer theory and engineering, 2010, 730-733.

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

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

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

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

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

[12] 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.

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

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

[15] U. Udayakumar and A. Murugan "An Efficient Algorithm For Solving Longest Common Subsequence Problem", IJESMR, 2017,4(5).

[16] www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence