A Modified algorithm to find Longest Common Subsequence and optimized space solution

Abstract:
In the study of Longest Common Subsequence (LCS) is an important topic and an important component in the application of Computer Science and Mathematics. The LCS problem is to find a subsequence which is common to at least two or more given sequences. The largest length subsequence is called as LCS. In this paper, we revisit the much-studied LCS problem for two given sequences. Based on the previous research, this paper is to make a modified algorithm to find the longest common subsequences from two given sequences. The analysis of comparison between modified and existing approach that this research gives better approximation. An algorithm is developed , when a LCS found, it search what other sequences has similar to that with same length.

Keywords: Longest Common Subsequence, Dynamic Programming, String Matching.

1. Introduction

In the study of Longest common Subsequence (LCS) is an important topic and an important component in the application of Computer Science and Mathematics to apply DNA matching in Molecular Biology. Many Scientific tasks involve measuring the similarity between two data sets.

In a diverse array of applications ranging from automated spell-checking to alignment of DNA sequences, the data we wish to compare is naturally represented as a sequence of letters. In this context, two of the most foundational and popular measures sequence similarity are the Longest Common Subsequence.

A subsequence of a string is a sequence of letters appearing left to right in the original string. Hence, given two sequences A and B. Intuitively, two strings with a larger LCS value are more similar than two strings with a smaller LCS.

Consequently, much recent research on sequence similarity has involved finding faster algorithms for returning good approximations for LCS. There has been a long line of work obtaining better and better approximation algorithms running in near linear time.

2. Literature Review

A large number of research has been conducted in finding LCS between two sequences. The Needleman Wunsch [1] algorithm was the first application of dynamic programming which provide a global alignment between two sequences. This algorithm leads to the evolution of various efficient LCS algorithms. It is only suitable if the two sequences are similar of length. The Hirschberg [2] algorithm evolved from Needleman-Wunsch algorithm provides optimize version of Needleman- Wunsch. Another proposal for LCS proposed and optimized for [2] is proposed in [3]. Various parallel algorithms have been proposed in earlier to reduce the computation time and such algorithms are CREW PRAM model and Systolic arrays. In [4], they proposed fast LCS algorithm. Fast LCS efficiency which has been proposed in [4], further improved in [5]. A parallel LCS algorithm based on dynamic programming has also been proposed in [6].

3. Problem Description

LCS approach required a large amount of time for the deluge of genetic data which is represented billions of characters. Time for finding LCS can be reduced tremendously if we

can be able to solve the problem using non -alignment based distributed algorithm. When a new LCS is found, it is important to know what other sequences it is most like that and find those sequences. Sequence comparison has been used successfully to establish the limit between genes and gene evolved in normal growth and development. One way of detecting the similarity of two or more sequences is found their LCS.

A subsequence of a given sequence of letters is just the given sequence with some letters (possibly none) left out. Generally, gives a sequence of letter X= ($x_1, x_2, \ldots x_m$) the sequence Z=($z_1, z_2, \ldots z_k$) is a subsequence of X if there exists a strictly increasing sequence ($z_1, z_2, \ldots z_k$) of indices of X such that for all j=1,2…k, we have $x_{ij} = z_j$. For example Z= (B, C, D, B) is a subsequence of X= (A, B, C, B, D, A, B) with corresponding index sequence (2,3,5,7).

Given two sequences X and Y, we say that a sequence Z is a common Subsequence of X and Y if Z is a Subsequence of both X and Y. For example if X= (A, B, C, B, D, A, B) and Y=(B,D,C,A,B,A), the subsequence (B,C,A) is a Common Subsequence of both X and Y. The Sequence (B,C,A) is not a longest Common Subsequence (LCS) of X and Y, however, since it has length 3 and the sequence (B,C,B,A) is LCS of X and Y, as is the sequence (B,D,A,B), since there is no common Subsequence of length 5 or greater.

4. Existing Approach

In the LCS problem, we are given two sequences $X = (x_1, x_2 \ldots \ldots x_m)$ and $Y = (y_1, y_2 \ldots \ldots y_n)$ and wish to find minimum length common sequence of X and Y. As modified approach to solve LCS problem is to enumerate all subsequences of X and check each subsequence to see if it is also a subsequence of Y, keeping track of the LCS found. Each subsequence of X corresponds to a subset of indices (1, 2….., m) of X, so this approach requires exponential time, making it impractical for long sequence.

The common and popular algorithm of finding the LCS between two strings is the well-known dynamic programming approach. A DNA of any organism is a linear sequence as a basic structure $x_1, x_2 \ldots \ldots x_m$ 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,Cystosine,Guanine and Thymine.ie A,C, G, T[7]. There are some important steps were included and is given in detail.

Step1: Identifying a longest common subsequence.

Theorem 4.1.
The optimal substructure property of LCS problem. Let, $X = (x_1, x_2 \ldots \ldots x_m)$ and $Y = (y_1, y_2 \ldots \ldots y_n)$ be sequences. Let $C = (c_1, c_2, \ldots \ldots c_i)$ be any LCS of X and Y. The theorem states three cases given below:
1. If $X_m = Y_n$ then $c_i = X_m = Y_n$ and C_{i-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $X_m \neq Y_n$, then $c_i \neq X_m$ means that C is an LCS of X_{m-1} and Y.
3. If $X_m \neq Y_n$, then $c_i \neq Y_n$ means that C is an LCS of X and Y_{n-1}.

Step 2: Generate a recursive loop for LCS solution

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

Step3: 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 Length of LCS (r) = C [m − 1, n − 1].

Step4: Backtracking to construct an LCS.

In the two given sequences X and Y, let's say C be a common subsequence between two sequences if the subsequence C exists in both the strings [8]. 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.
It is noted that we are given two sequences X and Y we wish to have a maximum-length common subsequence which can be solved by dynamic programming.
With this procedure, we meet consequences of this LCS in reverse order by traversing the matrix by backtracking. In order to get an appropriate result, recursive methods is to be used to print the LCS of X and Y. This procedure prints BABC. The procedure takes time O(m+n), since it has nested loop of I and j which decrease by 1 in each iteration.

5. Modified Approach

LCS[i,j] is the length of LCS of S[1,2…..,i] with T[1,2,……,j]. How can we solve for LCS[i,j] interms of the LCS's problems.

Case I: If S[i] ≠T[j], then the desired subsequence has to ignore one of S[i] or T[j], so

LCS[I,j]=max(LCS[i-1,j-1], LCS[I,j-i]).

Case II: If S[i] = T[j] ,then the LCS of S[1,2…..,i] and T[1,2,……,j] might as well as match them up.

LCS[i,j]= 1+LCS[i-1,j-1].

In this paper we have proposed modified algorithm in order to find LCS for string matching problem. It is mainly used for protein and DNA. After examine this algorithm, we analyze that, this algorithm can efficiently be used for improving time and space complexity.

Some preliminary definitions are as follows: We represent the concatenation of strings X and Y by XIIY.

P_{1i}, represents the string $p_1, p_2, \ldots \ldots \ldots p_i$ (elements 1 through i of string P). Similarly, the

prefix of length j of string Q is represented by Q_{1j}.

We define L(i,j) to be the length of the LCS of prefixes of lengths i and j of strings P

and Q, i e. the length of the LCS of P_{1i} and Q_{1j}.

Theorem 5.1.

For $n \geq 1$, (i, j) is n-letters iff L(i, j) $\geq n$ and $p_i = q_j$. Thus, there is a common subsequence of length n of P_{1i} and Q_{1j}.

Proof:

By induction on n. (i, j) Is a 1-letter if and only if a, = be (by definition), in which case *L(i, j)* necessarily is at least 1. Thus the theorem is true for n=1.

Assume it is true for n-1, consider n if

(i,j) n-letter then there exists $i' < i$ and $j' < j$ such that (i', j') is n-1 letter sequence. By assumption there is a subsequence $D^i = d_1, d_2, \ldots\ldots\ldots d_{n-1}$ of $P_{1i'}$ and $Q_{1j'}$.

Since $p_i = q_j$ ((i,j) is a n-letter), $D = D'$ is a common sequence of P_{1i} and Q_{1j}.

Thus $L[i, j] \geq n$

Conversely $L[i, j] \geq n$ and $p_i = q_j$ then there exists $i' < i$ and $j' < j$ such that

$p_i' = q_j'$ and $L[i', j']$= $L[i, j]$ -1 $\geq n - 1$

(i', j') is a n-1 letter sequence (by inductive hypothesis)

And thus (i, j) is a n-letter sequence.

The length of LCS is p, the maximum value of n such that there exists a n-letter sequence. As we see, to recover an LCS, if suffices to maintain the sequence of a 0-letter, 1-letter,……(p-1) letter and a n-letter such that in this sequence i-letter generate the i+1 letter for $0 \leq i < p$.

Rule: Let x = (x_1, x_2) and y = (y_1, y_2) be two n-candidates. $x_1 \geq x_2$ and $y_1 \geq y_2$ then we say that y rules out x (x is a superfluous n-letter sequence) since any (k + 1)- letter that could be generated by x can also be generated by y. Thus, from the set of n-letters, we need consider only those that are minimal under the usual vector ordering. Note that if x and y are minimal elements then $x_1 < y_1$ if and only if $x_2 > y_2$.

Theorem 5.2.

For fix integers m and n with m>n≥2. Suppose that there is a optimized time algorithm (T) that achieves a 1/n-α approximation of the LCS of two sequences of length at most k from an alphabet size n. Then, there is also a O((n+T)$_m$C$_n$) time algorithm that achieves as 1/(m(1-α/n)) approximation of the LCS of two sequences of length atmost n from an alphabet of size m.

Proof:

We how to come across the LCS for two sequences of length at most k over an m-any alphabet, to the LCS for two sequences of length at most k over an l-length sequence for any n<m.

The reduction runs in O((k)$_m$C$_n$) time and produces $_m$C$_n$ instances of l length LCS.

Let P and Q be two sequences of length at most n over an alphabet of size m. Then define L to be the longest common subsequence of P and Q (It is not known the identity of L). In order that, sort the letter symbol according to their number of occurrences in L.

Let a be the collection of n most frequent letter symbols in L. Let L_a be the subsequence of L obtained by restricting L to the letters of alphabet of that contains the symbols of a. Since a has the n most frequent symbols in L, L_a contained least an n/m fraction of the characters in L.

Now let us illustrate our approach. Given P and Q, we consider all subsets of the alphabet consisting of precisely n symbols. Note that one of these subsets will be a.

For each such collection b, consider the subintervals of LCS problem formed by restricting the symbols of y. Let OPT be the optimal LCS for this interval.

From that

|Optimize sequence of a| ≥ L_a ≥ ${}_mC_n$ |C|.

So, we consider b=a, if we can efficiently obtain a 1/n- α approximation for optimized a, we will get a common subsequence of P and Q of length at least

$$\frac{|\text{Optimized (a)}|}{\text{n} - \alpha} \geq \frac{|L|}{m(1 - \alpha/\text{n})}$$

Which yields the desired approximation for the LCS of P and Q. The running time is multiplied by ${}_mC_n$, which is simply a constant provided m is bounded above by constant.

By setting

A=nβm/1+βm. We obtain a linear time reduction from obtaining a 1/m+β approximation for m-sequence of letters to a 1/n+(βm/n) approximation for n-sequence of letters.

Corollary 5.3.

Fix an integer m ≥ 3 and a constant β > 0. The problem of obtaining a 1/m+β approximation for the LCS of two sequences of letters from an alphabet of size m can be reduced in linear time to the problem of obtaining a 1/2+((βm)/2) approximation for the LCS of two sequences.

Proof:

This follows from theorem 5.2 by taking n=2. Note that the reduction has a multiple over head of $O(m^2)$, which is constant when m is constant.

6. Conclusion

In this paper we have proposed a modified algorithm in order to find all LCS for sequence of letters. After examine this algorithm, we analyze that this algorithm can efficiently be used for improving optimize the time and space complexity. It has been examined that that this algorithm has also been able to find LCS from multiple sequence of letters. We feel that there are still some scope to improve the time complexity and performance of our approach through different data structure such as space complexity and finding equal length with different letters.

References

[1]. Needleman, Saul B., and Christian D. Wunsch. “A general method applicable to search for similarities in the amino acid sequence of two proteins.” Journal of molecular biology 48.3 (1970): 443-453.

[2]. Hirschberg, Daniel S. “A linear space algorithm for computing maximal common subsequences”. Communications of the ACM 18.6 (1975): 341-343.

[3]. Hunt, James W., and Thomas G Szymanski. “A fast algorithm for computing longest common subsequences.” Communications of the ACM 20.5 (1977): 350-353.

[4]. Chen, Yixin, Andrew Wan and Wei Liu. “A fast parallel algorithm for finding the longest common subsequence of multiple bio sequences.” BMC bioinformatics 7.4(2006),1.

[5]. Eswaran S and RajaGopalan SP. “An Efficient fast Pruned parallel algorithm for finding LCS in Bio sequences.” Anale Seria Informatics. Vol. VIII fasc.1, 2010.

[6]. Dharaief, Amine, Raik Issaoui, and Abdelfettah Belghith. “Parallel computing the Longest Common Subsequence (LCS) on GPUs: efficiency and language stability.” The 1st international conference on Advanced Communications and Commputation(INFOCOMP). 2011.

[7]. T. H. Corman, C. E. Leiserson, R. L. Rivest, C. Stein, Cambridge, Introduction to Algorithm, Third edition,” Dynamic Programming”, ch. 15, sec. 15.4, pp. 390-397 MA: MIT Press, 2010. Trans. on Math. Software, 11: 289(1985).

[8]. Bergroth, L, Hakonen, H, and Raita,T.”A survey of Longest common subsequence algorithms” , IEEE computer Society Washington, DC,USA 2000.