Sequence alignment using Longest Common Subsequence algorithm [Edit]

11 min read

Introduction

In molecular biology, DNAs and proteins can be represented as a sequence of alphabets. DNA sequences consist of A, T, G, C representing nucleobases adenine, thymine, guanine and cytosine. Proteins consist of 20 different letters indicating 20 different amino acids.

Comparison of two sequences, known as **sequence comparison**, either from the same organism or from different organism is an important task in molecular biology. It is helpful in providing solutions to many biological questions, for example:

1. predicting structure and function of proteins
2. inferring evolutionary history and relatedness of species
3. locating common subsequences in genes / proteins to identify common *motifs*,
4. as a sub-problem in genome assembly for DNA sequencing

In order to perform sequence comparison, we first perform **sequence alignment**.

What is a Sequence Alignment?

A **sequence alignment** is a way of placing one sequence above the other in order to identify the correspondence between similar characters or substrings. It can be performed on Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA) or protein sequences.

Let's consider an example. Say we have the following two DNA sequences `GACGGATTAG` and `GATCGGAATAG`. One possible alignment between these two sequence is shown below

```
GA-CGGATTAG
|| |||| |||
GATCGGAATAG
```

Note the differences in the two sequences as per this alignment. An extra T in the second sequence, and a substitution from T to A. A 'space' is added to the first sequence at a suitable location to align the sequences.

Types of alignment

There are two types of alignment based on what we are looking for:

1. **Global alignment**: Aligns a query sequence with the target sequence along the entire length of the sequence. Global alignment is a global optimization process that spans the entire length of two query sequences.
2. **Local alignment**: Aligns a substring of the query sequence to a substring of the target sequence, i.e. finding *local* regions of high similarity between two sequences.

There are also two types of alignment based on the number of sequences being aligned

1. **Pairwise alignment**: Involves two sequences, the query and the target with which it is aligned.
2. **Multiple alignment:** Involves alignment with more than two sequences.

Software tools for sequence alignment

Locating regions of similarity of the query sequence with the database sequences is a challenging task in bioinformatics. Tools like BLAST and FASTA helps in detecting regions of similarity among organisms. It involves the use of local pairwise alignment, exhaustive heuristic algorithms and dynamic programming approaches like Smith-Waterman algorithm to detect regions of similarity of the query sequence with the database sequence.

Longest common subsequence (LCS) problem

Illustration

Let's consider an example. Suppose we have the following two DNA sequences: `TAGTCACG` and `AGACTGTC`. The LCS of the two sequences is `AGACG`, which can be obtained from the following alignment.

```
TAGTCAC-G--
 ||  || |
-AG--ACTGTC
```

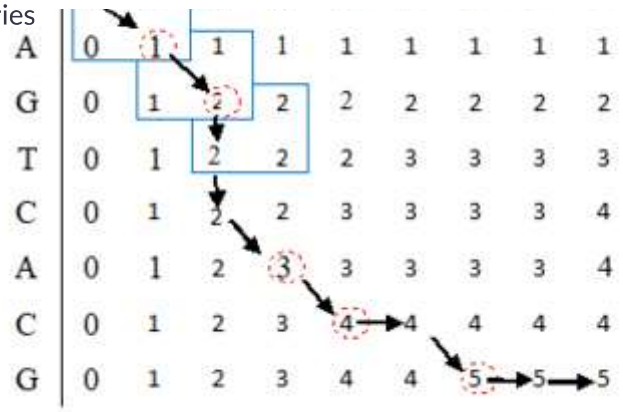
There are other possible common sequences of shorter length, such as `AGCG` or `AGTC`. Although multiple LCS are possible in general, there is only one LCS for this particular example, i.e. there is no other common subsequence of length 5 for these two sequences.

The LCS also helps in computing how similar the two sequences are: the longer the LCS, the higher the similarity.

LCS algorithm

Let us suppose we have two sequences S1 and S2 of lengths *m* and *n* respectively, where S1 = a_1 a_2 ... a_m and S2 = b_1 b_2 ... b_n.

We will construct a matrix A where $A_{i,j}$ denotes the length of the longest common subsequence of a_1 a_2 ... a_i and b_1 b_2 ... b_j.



		A	G	T	C	A	C	G
	0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1	1
G	0	1	2	2	2	2	2	2
T	0	1	2	2	2	3	3	3
C	0	1	2	2	3	3	3	4
A	0	1	2	3	3	3	3	4
C	0	1	2	3	4	4	4	4
G	0	1	2	3	4	4	5	5

> Example matrix for sequences S1 = TAGTCACG and S2 = AGACTGTC.

Sequence S1 is written vertically, and S2 horizontally. Each letter representing the rows is compared with each letter representing the columns. We'll proceed row by row, column by column.

1. If $a_i = b_j$, then we have found a match. We get a score of 1 for the current match, and $A_{i-1,j-1}$ from the rest of the LCS we already obtained from substrings $a_1 \dots a_{i-1}$ and $b_1 \dots b_{j-1}$.
2. If $a_i \neq b_j$, then we have a mismatch. In this case, we need to consider two possibilities. The LCS $a_1 \dots a_i$ and $b_1 \dots b_{j-1}$, and the LCS of $a_1 \dots a_{i-1}$ and $b_1 \dots b_j$.

Hence, we have

$$A_{i,j} = \begin{matrix} A_{i-1,j-1} + 1 & \text{if } a_i = b_j \\ max(A_{i-1,j}, A_{i,j-1}) & \text{if } a_i \quad b_j \end{matrix}$$

and $A_{0,0} = A_{0,j} = A_{i,0} = 0$ for $1 \leq i \leq m$, $1 \leq j \leq n$.

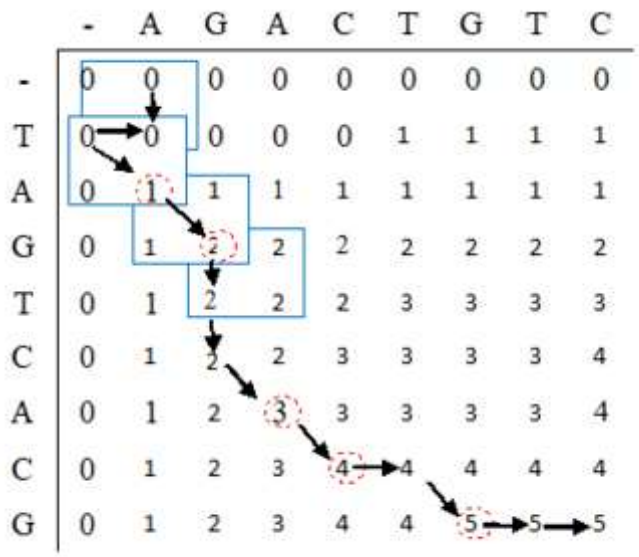
It takes O(*nm*) time to fill in the *m* by *n* matrix A. This approach is called dynamic programming.

Step by step example

The matrix is filled row by row, column by column.

1. The first row R1 and column C1 are filled with zero.
2. (**mismatch example**) The first letter in the row R2 is compared with the column C2. The cells in this case contains 'T' and 'A'. Since it is a mismatch, the score is taken from left or

4. (**match example**) When R3 is compared with C2, both the cell contain 'A'. Since it is a match, the score is given by 1 + the score at diagonally up and to the left, i.e. (R2, C1), which is '0' in this case. So the total score becomes 1.
5. We continue filling the rest of R3.
6. (**match example**) In a similar way, when R4 is compared with C3, it is again a match. This time, the score is given by 1 + score from (R3, C2) which is 1. Hence, total score becomes 2.
7. The entire matrix is filled in this way. The final score is given by $A_{m,n} = 5$.

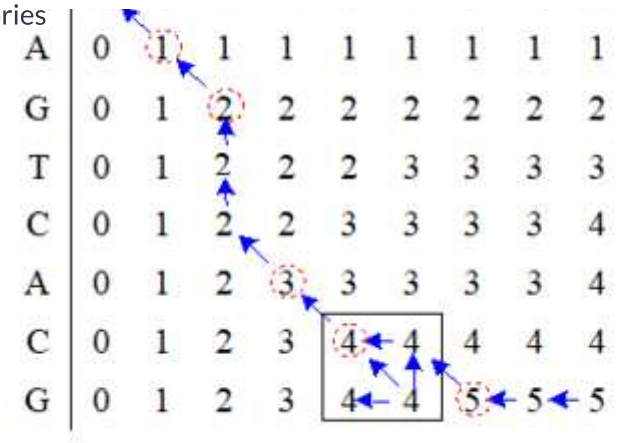


	-	A	G	A	C	T	G	T	C
-	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	1	1	1	1
A	0	1	1	1	1	1	1	1	1
G	0	1	2	2	2	2	2	2	2
T	0	1	2	2	2	3	3	3	3
C	0	1	2	2	3	3	3	3	4
A	0	1	2	3	3	3	3	3	4
C	0	1	2	3	4	4	4	4	4
G	0	1	2	3	4	4	5	5	5

> Example matrix for sequences S1 = TAGTCACG and S2 = AGACTGTC.

Traceback

The actual subsequence is deduced by performing a *traceback*, i.e. tracing backwards from the right bottom to the top left. When the length decreases is all directions, the sequences must have had a match. Several paths are possible when two arrows are possible in a cell (given in the box below). In such cases either path can be chosen.



A	0	1	1	1	1	1	1	1	1
G	0	1	2	2	2	2	2	2	2
T	0	1	2	2	2	3	3	3	3
C	0	1	2	2	3	3	3	3	4
A	0	1	2	3	3	3	3	3	4
C	0	1	2	3	4	4	4	4	4
G	0	1	2	3	4	4	5	5	5

The locations of the diagonal edges contains all the required information for constructing the LCS, as well as knowing the locations in S1 and S2 which the LCS corresponds to.

The LCS for the above example is **AGACG** of length **5.** S1 and S2 with locations of diagonal edges underlines are as follows: **TAGTCACG** and **AGACTGTC**.

Scoring matrices

In the above example, we said we get a 0 for a mismatch and a 1 for a match. This can be generalized to different scores for deletion, insertion or match of each of the alphabets, and different scores for each possible pair of mismatched alphabets.

	A	C	G	T
A	1	0	0	0
C	0	1	0	0
G	0	0	1	0
T	0	0	0	1

identity

	A	C	G	T
A	1	-3	-3	-3
C	-3	1	-3	-3
G	-3	-3	1	-3
T	-3	-3	-3	1

BLAST

	A	C	G	T
A	1	-5	-1	-5
C	-5	1	-5	-1
G	-1	-5	1	-5
T	-5	-1	-5	1

Transitions & transversions

The score for the matches and mismatches are based on evolutionary significance of bases (in case of DNA) and amino acid (in case of proteins). For example, the amino acids are given weightage based on their conserved nature of occurrence observed empirically.

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

> BLOSUM (**BLO**cks **SU**bstitution **M**atrix). Commonly used scoring matrix for sequence alignment of proteins. Each row, column is a amino acid, and a protein is a sequence of amino acids.

List of sequence alignment software Data science Longest common subsequence problem Sequence alignment Bioinformatics BLOSUM

Category: Machine Learning • Last updated: 11 months ago

ABOUT THE CONTRIBUTORS

Keshav Dhandhania
Machine Learning @ MIT (2014). DM me for 1-on-1 mentorship (paid).

Subha Yegnaswamy
Bioinformatics Expert with 15 yrs of work experience

Anant Jain
Co-Founder, Commonlounge

Add a comment

Back to top

POPULAR PATHS & COURSES

Machine Learning

Deep Learning

Natural Language Processing

Big Data

Data Science

Bioinformatics

Algorithms

Cryptography

NEW PATHS & COURSES

Web Development

UX & UI Design

Startups

Product Management

Cryptocurrencies

Finance

College Admissions

ABOUT US

Our Mission

How to Contribute

Upgrade to Pro

Help and FAQ

Commonlounge Meta

Team

GET IN TOUCH

hello@commonlounge.com

Facebook

Twitter

Church St, San Francisco, CA, USA

+1 844 318 7406