Sequence alignment

1. Introduction

Bioinformatics developed in the 1970s, when molecular biologists started determining biological sequences – of nucleotides (DNA, ESTs) or of amino acids (proteins). Researchers soon realized that the resulting data should be stored in an easily accessible way, to allow sharing and replication across the world, which led to the formulation of various database and file format standards. As the number and length of sequences became too large to analyze by hand, it also spurred the development of algorithms to automatically compare sequences. Such comparisons are at the core of bioinformatics, underlying comparative genomics, evolutionary studies (phylogeny), functional classification of proteins etc. In itself, sequence comparison is a straightforward problem. In a computer, two strings of characters can easily be compared by exhaustive search: numerically compare the (ASCII) code assigned to each letter, an $O(n)$ algorithm for two sequences of length n.

A complicating factor is that evolution "obscures" sequence comparison by introducing mutations, leading to changes in characters, to the insertion or deletion of a few characters in certain places or to even full translocations, where different sequences are concatenated into a single one. A method for calculating sequence similarity should therefore be able to deal with:

- *substitutions* of nucleotides or amino acids by others, taking into account the likelihood of such substitutions;
- *gaps*, i.e. short insertions or deletions in one sequence w.r.t. the other;
- local (or partial) *alignment*, where we look for a (longest) matching sub-sequence.

This makes the problem quite a bit harder. First, we need to define a criterion that assigns a score to how well two sequences align, including penalties for substitutions and gaps. Here, the work of (among others) Margaret Dayhoff resulted in substitution matrices (PAM, later BLOSUM), derived from sets of conserved sequences, that assign a score to each possible substitution of an amino acid by another given a certain assumed evolutionary distance (for nucleotides, usually a very simple scoring function is used).

Second, we need a smarter way of searching: exhaustive search is no longer feasible, as there are simply too many different options to consider. A major breakthrough was the realization by Needleman and Wunsch in 1970 that an algorithmic technique developed in the 1950s, dynamic programming, could be applied to this problem.

In this assignment, you will implement this basic approach to global sequence alignment. You will be given a skeleton implementation, containing the BLOSUM matrix, a function to calculate the substitution score between two amino acids and some example sequences. The goal is to write a number of specific functions that, taken together, allow for global alignment of these sequences.

Reading material

Chapters 4 and 5 of "Understanding bioinformatics" contain an overview of the problem of sequence alignment resp. in-depth discussion of a number of algorithms. Start by reading section 4.1-4.4 and 5.2, where 5.2 contains most of the details needed to work out the assignment below. Optionally, you can also read section 5.1, to learn how the substitution matrices are constructed.

2. Assignment

Use the code skeleton in `assignment1_skeleton.py` to:

- Implement a function for a linear gap penalty (EQ5.13).
- Implement a function that returns an initial matrix (Fig 5.8; see Hints).
- Implement the Needleman-Wunsch algorithm for global sequence alignment, including the following steps:
  1. filling the matrix cells, using EQ5.17. You can use the function `score` to look up a value in the BLOSUM62 matrix.
  2. a traceback strategy to obtain the aligned sequences (e.g. Fig 5.9)
- Add a parameter to your matrix function to allow a separate end_gap penalty, different from the regular gap penalty (like fig. 5.12, but also considering gaps at the end of the alignment (last row and column)).
- Implement a function to calculate the percentage of identity between two aligned sequences (defined as: `num_identical_residues/aln_len * 100`).

Questions

Write a short report containing the answers to the following questions:

1. Using `seq1` and `seq2`, can you reproduce the alignments from figure 5.9, 5.11, 5.12? Put the output in your report.

2. Describe your traceback strategy in words. In case of equal scores from multiple cells, which direction is preferred?

3. When you align `seq1` and `seq2` using different linear gap penalties, ranging from 1 to 20 (and no separate end gap penalty), how many different alignments do you get? List the different alignments. Explain the differences, given the settings.

4. Align two related proteins, `seq3` and `seq4`, and report the alignment score (right bottom in the matrix) using 2 different settings:
   linear gap penalty = 5 & end gap = 1
   linear gap penalty = 5 & end gap = 10

5. What is the percentage of identity between the aligned sequences (`seq3` and `seq4`) for both settings above?

6. Can you reproduce your results using the EMBOSS needle program?
   `http://www.ebi.ac.uk/Tools/psa/emboss_needle/`
   Note the settings under “more options”. Print the needle output.

7. What is the criterion that the algorithm optimizes?

8. Is the algorithm exact or approximate?

9. What are the time and space complexity of the algorithm you implemented?

10. Describe how you could decrease running time by limiting the number of gaps allowed in the alignment.

Hints

In Python, you may create a matrix as a list of lists. For example:

```
matrix = [[0 for j in range(10)] for i in range(5)]
```

which can then be addressed using `matrix[i][j]`.

Please, email your Python script and a PDF file containing the answers to the questions to algorithms@bioinformatics.nl no later than 9.00 on the day of the lecture.