I519 Homework 3, Fall 2015

Due Oct 12th (Monday), 2015 11:59pm via canvas (total 100 pt)

September 27, 2015

HW3 is going to help you get a deeper understanding of the alignment problems and warm up for genome sequencing. You are going to implement the Smith-Waterman (SW) algorithm for local alignment! The SW algorithm can guarantee to give the optimal alignment score and path(s) for a pair of sequences given a scoring function, but it is the application of proper substitution matrix and gap penalty that makes the "optimal" alignment biologically meaningful. You will also write a program for simulating genome sequencing data. Provided files are available on burrow under /u/yye/I519/HW3/.

0 Review of sequence alignment

0.1 Affine gap penalty

Affine gap penalty is commonly used in alignment of biological sequences. Affine gap penalty for a gap of length x is $(\rho + \sigma x)$, where ρ is the penalty for introducing a gap, and σ is the gap extension penalty.

0.2 Smith-Waterman algorithm for local alignment with affine gap penalty

Smith and Waterman solved the local alignment problem by introducing the magic 0 into the recursive definition of alignment score. Below is the recursive equation for local alignment using affine gap penalty.

$$D(i,j) = max \begin{cases} D(i-1,j) + \sigma \\ S(i-1,j) + (\sigma + \rho) \end{cases}$$

$$I(i,j) = max \begin{cases} I(i,j-1) + \sigma \\ S(i,j-1) + (\sigma + \rho) \end{cases}$$

$$S(i,j) = max \begin{cases} 0 \\ S(i-1,j-1) + \delta(x_i, y_j) \\ I(i,j) \\ D(i,j) \end{cases}$$

where $I(i,j)$ and $D(i,j)$ are the best alignment scores for subsequences $x_1 \dots x_i$ and $y_1 \dots y_j$ ending with insertion, and deletion, respectively. These are the extra computation introduced by affine gap penalty.

1 Your own local alignment software (60 points)

You are going to implement the Smith-Waterman algorithm, using BLOSUM62 matrix (in a file named BLOSUM62), and affine gap penalty (new gap penalty = -11, and gap extension = -1). Your program will get the sequences from a file with two protein sequences in FASTA format. Your program needs to report the best alignment score, and outputs the alignment in a user-chosen format (FASTA, or PLAIN for plain text with detailed alignment information as shown below). A sample usage of the code is:

```
SWalign.py -i twoseq.fasta -f FASTA -o twoseq-aligned.fasta
```

where twoseq.fasta is an input file with two protein sequences to be aligned (available on burrow), -f FASTA tells the program to output the alignment in the FASTA format, and -o option tells the program to output the alignment in a file named twoseq-aligned.fasta.

Here is an example of the alignment output in FASTA format,

```
>seq1
SSVWILHD-AGWS
>seq2
STVW-LHDNAGWT
```

The same alignment in the plain text format:

```
Input seqs: seq1 (len=12); seq2 (len=12)
Alignment length=13; identity=69%
SSVWILHD-AGWS
|.|| ||| |||.
STVW-LHDNAGWT
```

In this format, lines are used to indicate identical amino acids, while the dots represent alignments between similar amino acids (two amino acids are considered similar if their substitution score is greater than 0).

Test your program using the given sample sequences, pairs of very similar sequences you make up (so the "correct" answers will be obvious), and pairs of more diverse (and longer) sequences.

You will receive 50 points if your program outputs correct alignment score for given input sequences; and full credit if your program backtracks correctly and outputs the "correct" alignments.

2 Running time estimation (20 points)

How long will it take your SW program to search a query protein against 1M sequences on burrow? Give an estimation of running time, using your version of the SW algorithm, for comparing a query protein of about 200 aa long against a collection of 1M sequences each of about the same length. Explain how you get this estimation.

3 A genome sequencing simulator (20 points)

You will write a program to simulate genome sequencing process. Your program takes in a genome sequence, read length, and a parameter for sequencing coverage. You need to understand that $coverage = read\_length * number\_of\_reads/genome\_len$. For example, given a genome of 4M bases, 40,000 reads of length 100 bases gives a coverage of 1, meaning that on average each nucleotide in the genome is covered by one read. But some positions may be covered by more than one read, and some positions may not covered at all, as genome sequencing is a random process. To simulate the sequencing process, your program picks up positions randomly along the genome, and takes slices (of given length) from the genome sequence. You code outputs the simulated reads in a file, and outputs the total number of positions in the genome (and the proportion) that are not covered. You may test your code with genome NC_010698.fna, using read length = 100, and coverage = 1. For simplicity, you do not worry about sequencing errors for this assignment.