Algorithms in Bioinformatics
Project 2
Global alignment with different gap costs

Ditte Torlyn (201408508)
Rune Wind (201608439)
Astrid Christiansen (201404423)

February 2020

Introduction

In this project, we implement and experiment with pairwise sequence comparison methods to compute optimal global alignments of two sequences where the aim is to minimize a cost.
We have implemented the two mandatory exercises with linear gap cost and affine gap cost and our solutions work as expected.

Methods

Our solution is in two python files, `project2_linear.py` and `project2_affine.py`. We use the `BioPython` library to read FASTA files and we have implemented a helper function, `parse_phylip` to parse substitution matrices from a Phylip-like format, as described in the next subsection. We also use this helper function to determine the alphabet, so that we can check whether the given input sequences contain only allowed letters. We have implemented the algorithms very alike to the ones discussed in class, where our function `calculate_alignment_matrix` fills out an alignment matrix of costs row by row. So this function works as a structural template of our algorithm, while we have chosen to delegate the task of actually computing the cost of a single matrix cell to the function `calc_cost`. So if we simply want to know the optimal alignment score of two sequences, we call `calculate_alignment_matrix` and look at the last cell in the outputted matrix. For convenience, we print this value inside `calculate_alignment_matrix`. Now, if we also want to know an optimal alignment, we call our function `backtrack` which is simply an implementation of the algorithm we have seen in class. It writes the found optimal alignment in a file, `alignment.fasta`. If more than one optimal alignment is found it only writes one of them.

How to run our programs

Our program files are called `project2_linear.py` and `project2_affine.py`. Running these programs will output an optimal alignment score. To run our programs from the command line, type:

`python project2_linear.py submatrix.txt gapcost seq1.fasta seq2.fasta backtrackingOption`

and

`python project2_affine.py submatrix.txt gapcost_a gapcost_b seq1.fasta seq2.fasta backtrackingOption`

where `submatrix.txt` is a text file of a Phylip-like format, eg.:

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

and `gapcost` is a number (eg. 5) and `seq1.fasta` and `seq2.fasta` are FASTA files with the two sequences to align. Lastly, `backtrackingOption` is an optional variable which can be set to True if we want to find an optimal alignment and it is false by defult. The optimal alignment of the sequences is then saved in a FASTA-file named `alignment.fasta`. An example of how to run our files could be:

`python project2_affine.py submatrix.txt 5 2 seq1.fasta seq2.fasta True`

Where `submatrix.txt` is a text file of Phylip-like format, `seq1.fasta` and `seq2.fasta` are FASTA files; all located in the same folder as `project2_affine.py`. Here, we have specified 5 and 2 as gap costs (2 is the opening gap cost), and True indicates that we want to know an optimal alignment.

Test

We have firstly verified the correctness of our programs on the test cases in `project2_examples.txt`. We have also used some of the examples from classes to test our programs. For all tested cases, we get the expected results.

We consider the following substitution matrix, M, for DNA sequences:

	A	C	G	T
A	0	5	2	5
C	5	0	5	2
G	2	5	0	5
T	5	2	5	0

and the following 5 sequences

```
>seq1
tatggagagaataaaagaactgagagatctaatgtcgcagtcccgcactcgcgagatact
cactaagaccactgtggaccatatggccataatcaaaaag

>seq2
atggatgtcaatccgactctacttttcctaaaaattccagcgcaaaatgccataagcacc
acattcccttatactggagatcctccatacagccatggaa

>seq3
tccaaaatggaagactttgtgcgacaatgcttcaatccaatgatcgtcgagcttgcggaa
aaggcaatgaaagaatatggggaagatccgaaaatcgaaa

>seq4
aaaagcaacaaaaatgaaggcaatactagtagttctgctatatacatttgcaaccgcaaa
tgcagacacattatgtataggttatcatgcgaacaattca

>seq5
atgagtgacatcgaagccatggcgtctcaaggcaccaaacgatcatatgaacaaatggag
actggtggggagcgccaggatgccacagaaatcagagcat
```

Question 1

Compute the score of an optimal alignment and an optimal alignment of seq1 and seq2 above using the programs global-linear using the above score matrix M and gap cost $g(k) = 5 \cdot k$.

The optimal score is 226.

Question 2

Compute the score of an optimal alignment and an optimal alignment of seq1 and seq2 above using the program global-affine using the above score matrix M and gap cost $g(k) = 5 + 5 \cdot k$

The optimal score is 266.

Question 3

Compute the optimal score of an optimal alignment for each pair of the 5 sequences above using *global-linear* with the score matrix M and gap cost g(k)=5*k. The result is a 5x5 table where entry (i,j) the optimal score of an alignment of seqi and seqj.

	$seq1$	$seq2$	$seq3$	$seq4$
$seq1$				
$seq2$	226			
$seq3$	206	239		
$seq4$	202	223	219	
$seq5$	209	220	205	210

Question 4

Compute the optimal score of an optimal alignment for each pair of the 5 sequences above using *global-affine* with the score matrix M and gap cost g(k)=5+5*k. The result is a 5x5 table where entry (i,j) the optimal score of an alignment of seqi and seqj.

	$seq1$	$seq2$	$seq3$	$seq4$
$seq1$				
$seq2$	266			
$seq3$	242	283		
$seq4$	243	259	269	
$seq5$	256	254	243	247

Experiments

We timed the algorithms with the function time(), and from that we captured the following running times for the *global-linear* and *global-affine*:

(a) Linear gap cost (b) Affine gap cost

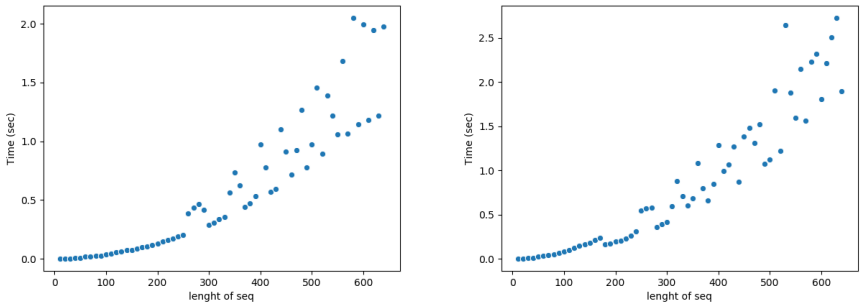

Figure 1: Running time

The running times looks as if they grow more rapidly than linearly in the (maximum) length of the sequences, n. Our hypothesis is that the running time is polynomial, $O(n^2)$. To check if this is correct, we plot the running times divided by n^2 as a function of n. Then the plots look like this:

(a) Linear gap cost (b) Affine gap cost

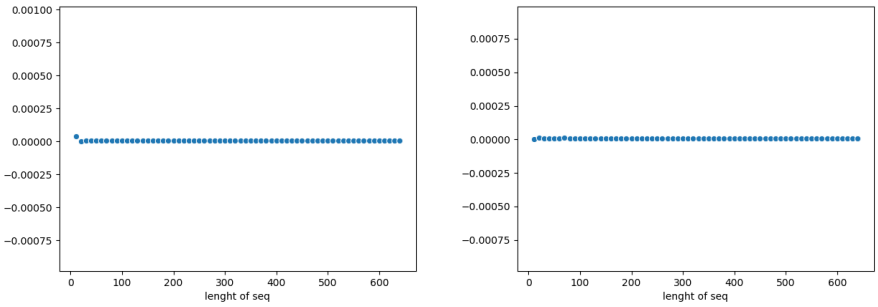

Figure 2: Running time over squared length

Since the running times divided by n^2 are approximately constant, we know that the running time indeed is $O(n^2)$ for both *global-linear* and *global-affine* gap cost. When performing the alignment with affine gap cost we fill out two additional tables to keep track of deletions and insertions. Because of the more complex algorithm, we expect the alignments with affine gap cost to have slightly longer running times, which is also what Figure 1 shows.