Python Programming - Assessed Exercise No.2

Issued: Wednesday 13 November 2013
Due: Monday 25 November 2013

Objective（目标）

You are required to write a python script that implements the Smith Waterman algorithm to produce a sequence alignment. The program will reads two sequences from file and print the alignment to the screen.
请写一个smith waterman算法的python script用于基因序列的比对，这个程序需能读取两条序列并最终将对比结果显示在屏幕上

Attached to this assignment is a basic tutorial on sequence alignment, including both global and local alignments. The requirement of the assignment is an implementation of the Smith Waterman local alignment. The implementation is simplified by the fact that a basic scoring method will be used.
在文章后面附带了一个基础的序列比对的原理指南，包括了global alignment和local alignment（此次程序仅需使用后者）两种比对方式。

Task（任务）

Write pseudocode and a Python script that performs the following functions:
写一个具有如下功能的pseudocode和python script：

- The code should read 2 fasta sequences from files, which must be called "Sequence1.fasta" and "Sequence2.fasta" and align them using the Smith Waterman algorithm. The filenames should be hard coded in the script and must not be changed.
- 此程序应能从文件中读取2个fastq序列，而此两个文件必须命名为"sequence1.fastq"和"sequence2.fastq".同时使用smith waterman算法（local alignment）源文件名必须hard coded同时不能更改。

- The alignment should be printed to screen only and the format should be:
- 最终结果仅是将比对结果在屏幕上以如下格式输出

```
Sequence1 ATCGGAGCTGGACCTGATGATTCGCGCCGAT
Sequence2 ATCTGAGCT-GACCTCATGATTGGCGCCGAT
```

- Suitable sample files are provided at http://web.bioinformatics.ic.ac.uk/python/assign2 that can be used to test your script and should produce the alignment shown above.
- 以上两个序列比对结果的源基因序列能在如上网站里找到，并可以作为一个范例验证程序的正确性

- The algorithm is simplified in that it only needs to use a basic scoring method, regardless of the nature of the sequences being aligned. The scores are:
- 这个简化了的算法只需要程序采用基础的计分方式，而这些得分标准如下：

Match +1（字母匹配）
Mismatch –1（字母不匹配）
Gap –1（出现间隔）

- Ensure that your code is adequately commented.

IMPORTANT:

Your script should not require or use any command line arguments.
It should open all files within the same directory as the script is run. Do not use relative paths to any files.
在此程序里，不能使用command line argument。所有的源文件，程序本身以及输出文件都必须存储在同一个directory下
Do not change the name of the input files.
不能修改源文件的文件名
You are not permitted to use BioPython, PyCogent or any third party implementation of the Smith Waterman algorithm.
所有常用的第三方辅助扩展包及程序都不能使用，如biopython, pycogent, numpy(换而言之就是不能在程序里出现import指令)

The Python script you submit by email must be in a format that can be run for testing and not as a Word document or PDF.
以能够在python里面直接运行的电子格式存储

Basic Sequence Alignment

Sequence Alignment – theory

There are two different ways to align two sequences – globally and locally. Global alignment seeks the best alignment over the whole length of the two sequences; local alignment seeks the best region or regions of alignment. Sequences are aligned computationally using a method called dynamic programming. Whether you use a global alignment or a local alignment depends on what you are looking for. If you want to find the most similar sentences between two books, you would use a local alignment algorithm. If you wanted to compare those two sentences end to end, you would use a global alignment algorithm. A biological example would be that you would use local alignment to align a small primer to a large template, and you would use global alignment to align two closely-related orthologous sequences.

In both types of alignment, positive scores are given to matching amino acids or nucleotides, negative scores are given to mismatches, and penalties are imposed for generating gaps in the alignment.

Global Alignment – Needleman & Wunsch

Needleman & Wunsch is the name of the standard algorithm[1] used for global alignment. An implementation[2] of it is available on Codon in the guise of an Emboss program called needle. We will go through an example in detail and then do one on your own.

We will first align two words – pelican and coelacanth.

Firstly, we set up the following comparison matrix:

		C	O	E	L	A	C	A	N	T	H
P											
E											
L											
I											
C											
A											
N											

[1] An algorithm is a set of rules which must be followed in order to achieve a desired result

[2] An algorithm is implemented in a program, which is set up to follow the rules laid down by the algorithm.

Each cell will be given a score and a pointer, which will either point up (↑), left (←), or diagonally up and left (↖). First of all, we initialize the matrix so that the edges all point back to the origin and are given a decreasing negative score, as if the alignment contained only gaps. This enables us to calculate the other scores in the matrix:

		C	O	E	L	A	C	A	N	T	H
	0	← -1	← -2	← -3	← -4	← -5	← -6	← -7	← -8	← -9	← -10
P	↑ -1										
E	↑ -2										
L	↑ -3										
I	↑ -4										
C	↑ -5										
A	↑ -6										
N	↑ -7										

For all the other cells, three scores are calculated:

1. Match score = sum of preceeding diagonal cell and the score of alignment of the two letters (+1 for a match, -1 for a mismatch).
2. Horizontal gap score = sum of score in the cell immediately to the left and the gap score (-1)
3. Vertical gap score = sum of score in the cell immediately above and the gap score (-1)

The largest of the three scores is assigned to the cell in question, and it is assigned a pointer according to whichever score is chosen (Match score = ↖ Horizontal gap score = ←, Vertical gap score = ↑).

Starting with the top left hand cell (trying to align C with P), we calculate that the match score is going to be 0 + -1 = -1 (i.e. the score in the preceding diagonal cell (0), plus the alignment score (-1 as P and C do not match). The horizontal gap score is going to be -1 + -1 = -2. The vertical gap score is going to be -1 + -1 = -2. Therefore, the largest score is the match score. We assign that value to the cell, and point the arrow accordingly.

		C	O	E	L	A	C	A	N	T	H
	0	← -1	← -2	← -3	← -4	← -5	← -6	← -7	← -8	← -9	← -10
P	↑ -1	-1									
E	↑ -2										
L	↑ -3										
I	↑ -4										
C	↑ -5										
A	↑ -6										
N	↑ -7										

Moving along to the right (trying to align O with P), we calculate that the match score is going to be the same as the horizontal gap score (-2), which is greater than the vertical gap score (-3). Here we have to decide whether to favour matches over gaps, and stick to this throughout the alignment. We will favour matches, so we set the cell to -2 and point the arrow diagonally (shown below).

We can now fill in the rest of the top row (shown below)

We can fill in the second row in the same way, note what happens when we reach the E-E match:

		C	O	E	L	A	C	A	N	T	H
	0	← -1	← -2	← -3	← -4	← -5	← -6	← -7	← -8	← -9	← -10
P	↑ -1	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
E	↑ -2	-2	-2	-1	← -2	← -3	← -4	← -5	← -6	← -7	← -8
L	↑ -3										
I	↑ -4										
C	↑ -5										

A	↑ -6										
N	↑ -7										

Now we can fill in the rest of the matrix:

		C	O	E	L	A	C	A	N	T	H
	0	← -1	← -2	← -3	← -4	← -5	← -6	← -7	← -8	← -9	← -10
P	↑ -1	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
E	↑ -2	-2	-2	-1	← -2	← -3	← -4	← -5	← -6	← -7	← -8
L	↑ -3	-3	-3	↑ -2	0	← -1	← -2	← -3	← -4	← -5	← -6
I	↑ -4	-4	-4	↑ -3	↑ -1	← -1	← -2	← -3	← -4	← -5	← -6
C	↑ -5	-3	← -4	↑ -4	↑ -2	↑ -2	0	← -1	← -2	← -3	← -4
A	↑ -6	↑ -4	-4	-5	↑ -3	-1	↑ -1	1	← 0	← -1	← -2
N	↑ -7	↑ -5	-5	-5	↑ -4	↑ -2	-2	↑ 0	2	← 1	← 0

To get the alignment, we start from the bottom right cell, and trace the highest scores back up through the matrix, following the arrows (shown below in red). The alignment comes out backwards. If there is a horizontal or vertical arrow, we put a gap between the sequences. If there is a diagonal arrow, we put a match.

		C	O	E	L	A	C	A	N	T	H
	0	← **-1**	← −2	← −3	← −4	← −5	← −6	← −7	← −8	← −9	← −10
P	↑ −1	−1	**-2**	−3	−4	−5	−6	−7	−8	−9	−10
E	↑ −2	−2	−2	**-1**	← −2	← −3	← −4	← −5	← −6	← −7	← −8
L	↑ −3	−3	−3	↑ −2	**0**	← −1	← −2	← −3	← −4	← −5	← −6
I	↑ −4	−4	−4	↑ −3	↑ −1	**-1**	← −2	← −3	← −4	← −5	← −6
C	↑ −5	−3	← −4	↑ −4	↑ −2	↑ −2	**0**	← −1	← −2	← −3	← −4
A	↑ −6	↑ −4	−4	−5	↑ −3	−1	↑ −1	**1**	← 0	← −1	← −2
N	↑ −7	↑ −5	−5	−5	↑ −4	↑ −2	−2	↑ 0	**2**	← **1**	← **0**

The alignment comes out as follows:

```
HTNACALEOC
--NACILEP-
```

Which we then reverse to get the proper alignment:

```
COELACANTH
-PELICAN--
```

This is the basic method for global sequence alignment. Try doing the same to align the two words MORCHEEBA and AMOEBA:

		A	M	O	E	B	A
M							
O							
R							
C							
H							
E							
E							
B							
A							

Write your alignment in this box:

In a “real” implementation of the Needleman & Wunsch algorithm, alignments are scored using substitution matrices, which give more favourable scores to similar matches (e.g. Asp-Glu retains a similar chemical moiety), and less favourable scores to unconservative matches (e.g. Phe-Glu, which changes the chemical composition completely). In alignments of nucleotide sequences, usually a simple match/mismatch scoring system is used, although there are some nucleic-acid substitution matrices available.

Additionally, affine gap scores are used, which gives a negative score for opening a gap, but a smaller negative score for extending a gap. This is a better biological representation as large gaps are often seen in alignments where insertions or deletions to sequences have occurred.

Local Alignment – Smith Waterman

The Smith-Waterman algorithm is a modified version of the Needleman & Wunsch algorithm. The edges of the matrix are set to zero, instead of increasing the gap penalty, and the maximum score for a cell can never be less than zero. If the score for the cell is equal to or less than zero, no pointer is recorded. Instead of tracing back from the bottom right cell, the traceback starts at the highest score, and finishes if the score reaches zero.

Suppose we wanted to align the previous example words PELICAN and COELACANTH using local alignment. Initially, we set up the comparison matrix as before, but initialize the edges to zero:

		C	O	E	L	A	C	A	N	T	H
	0	0	0	0	0	0	0	0	0	0	0
P	0										
E	0										
L	0										
I	0										
C	0										
A	0										
N	0										

No pointers are needed here – recall that a pointer is only recorded if the score for the cell is greater than zero. For each other cell, scores are calculated in the same way as for Needleman & Wunsch – we calculate the match score, the horizontal gap score and the vertical gap score, again using a +1/-1 scoring system. The highest score is taken. If the highest score is negative, the cell score is set to zero, and no pointer is recorded.

		C	O	E	L	A	C	A	N	T	H
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	**1**	0	0	0	0	0	0	0
L	0	0	0	0	**2**	← 1	0	0	0	0	0
I	0	0	0	0	↑ 1	**1**	0	0	0	0	0
C	0	1	0	0	0	0	**2**	← 1	0	0	0
A	0	0	0	0	0	0	↑ 1	**3**	← 2	← 1	0

N	0	0	0	0	0	0	0	↑ 2	**4**	← 3	← 2

The best local alignment is highlighted in bold above, and is:

```
ELICAN
ELACAN
```

The version of Smith Waterman we have worked through only gives you the single highest scoring local alignment. Some implementations of Smith Waterman give you the N-best scoring alignments – this can be more useful if you are looking for similarities in more than one region, such as if you are finding exons in a cDNA sequence.

Try using the Smith Waterman method to align AMOEBA and MORCHEEBA:

		A	M	O	E	B	A
M							
O							
R							
C							
H							
E							
E							
B							
A							

Write your alignment in this box: