Department of Computing and Information Systems
COMP 90016
Assignment 1
Release date: 8th of March 2016
Due date: 12th of April 2016 (11:59pm)

Introduction:

Alignment is the task of matching two sequences to each other. In this assignment you are going to write an alignment program with the specific task of matching sequencing reads to a reference genome.
The program is to be written in python. If you are inexperienced with python, there is plenty of online material to guide you. Specifically, the standard library is described here: https://docs.python.org/2/library/
The weekly labs are going to provide help and guidance as well.
For the task of reading files with specific formats (such as FASTA), the BioPython library may be helpful to you:
http://biopython.org/DIST/docs/tutorial/Tutorial.html
You will need to check that you have a working account on the Melbourne School of Engineering UNIX servers. See the instructions on the LMS for accessing these servers. In particular, you will need to use the Melbourne School of Engineering UNIX machine *digitalis.eng.unimelb.edu.au*. If you have accounts on some machines, but not on digitalis, you should lodge a ticket with ithelp.eng.unimelb.edu.au/servicedesk. The MSE Linux machines can be accessed from the Engineering student labs, using *MobaXterm*, which is found on the lab machines under Start → MobaXterm Personal Edition → MobaXterm Personal Edition. To access these machines from home, you will have to use the university's Virtual Private Network. For instructions and support see the documents on the LMS in the section *LabDocuments*. The instructions for using *MobaXterm*, both from the laboratories and from your own machine, are found in the first pages of the Unix document.
The assignment directory on the LMS and digitalis contain a program skeleton that you may or may not elect to use to get your own program started. It shows the use of the BioPython library to read FASTA files as well as how to create classes in python for an object oriented approach to the code design.

Tasks:

You are going to analyse simulated sequencing data from a short reference genome. The data are accessible on the LMS and in the comp90016 folder on the MSE unix servers (nutmeg, digitalis, dimefox – in /home/subjects/comp90016/assignments/assignment1/). You are free to work with the data in our lab environment or download the sequencing data to your own computer. The data contains the following files: 1) *reference.fa*, 2) *reads.fa*, 3) *reads_R1.fa*, 4) *reads_R2.fa*, and 5) *reads_err.fa*. All the files are in FASTA format. The set *reads_R1.fa*/*reads_R2.fa* is to be interpreted as paired-end reads. Each read in

reads_R1.fa is paired with the same line number (and read identifier) in *reads_R2.fa*. You can assume that the reference contains only one sequence. Only align to the first sequence in the reference file (not to be confused with the first line of the first sequence!)

Task 1. Write a python program (*aligner1.py*) that takes following inputs from the command line:

a. A reference filename (expecting a FASTA file).
b. A read file name (expecting a FASTA file).

The program should perform the following actions:

c. Align each read to the reference sequence. The alignment should only consider perfect matches to either the forward or reverse strand of the reference. The position of an alignment is the coordinate closest to the start of the reference. For a read on the reverse strand the position is therefore its last base. Count the number of possible alignments for each read and their positions within the genome.
d. Write the alignments to an output file in the same order as the reads present in the input file. The output file should be called *alignment1.txt*. There should be one line per read containing the following fields:
READ_NAME, REF_NAME, POS, STRAND, NUMBER_OF_ALIGNMENTS
The fields should be tab delimited. The *POS* field should contain the "left-most" position out of all possible alignments, that is, the position that is closest to the 5' end of the reference sequence, regardless of strand. The strand should indicate "+" if the specific alignment is to the forward strand ("-" otherwise).
Finally, the last field should specify how many possible alignments there are for the read.
If no alignments are found for a read, the output should read:
"*READ_NAME* * 0 * 0" (with the according read name).
e. Write alignment statistics to the command line. Specifically, how many reads were aligned to 0 positions, exactly 1 position, and more than one position (in absolute numbers and percent). Discuss the results. In particular, what could be the reasons for reads not aligning to the reference?

The program should perform sensible error checking regarding the existence and readability of the provided filenames.
For task 1 you should submit the program (*aligner1.py*), the alignment output (*aligment1.txt*) when run on data set *reads.fa* with the reference *reference.fa*, and the alignment statistics along with a short discussion of the outcome (*performance1.txt*).
Task 1 is worth 10 marks.

Task 2. The lectures introduced the concept of paired-end sequencing to disambiguate repetitive sequence content. Paired-end reads should align to the part of the genome that they originally came from in an approximately

known distance, and in the following orientation: the first read (the one that maps closest to the 5' end of the reference to the forward strand, the second to the reverse strand. If a read pair aligns this way, it is referred to as a *concordant alignment*. Files *reads_R1.fa*/*reads_R2.fa* are simulated paired-end data from the same reference as in task 1. Your task is to write a second aligner (*aligner2.py*) that is capable of utilizing such data.

a. Modify your program from task 1 to now accept three inputs: The reference, read1, read2 (all in FASTA format).
b. For all alignments of read 1 and read 2 (the two reads in a read pair) your aligner should identify if there is a concordant alignment. That is, an alignment where the two reads point at each other, as described above. For this task we are not investigating the insert size between reads in a pair.
c. The output of your program is to be written to the file alignment2.txt, and modified from task 1 as follows:
   Each line should have the tab-delimited fields *READ_NAME, REF_NAME1*, *POS1*, *STRAND1*, *NUMBER_OF_ALIGNMENTS1, REF_NAME2*, *POS2*, *STRAND2*, *NUMBER_OF_ALIGNMENTS2* to report the alignments for each read in the pair.
d. If there exists at least one concordant alignment for a read pair, the output of your program should report only these (again reporting the left-most alignment). In this case, *NUMBER_OF_ALIGNMENTS1* is necessarily equal to *NUMBER_OF_ALIGNMENTS2.* If no alignments are found for a read, the fields are to be replaced by "* 0 * 0" as before. Note that each possible position for one read could form a concordant alignment with each position of the other read. The reported alignment in case of multiple concordant alignments should be that with the overall smallest coordinates (favouring the smallest coordinate on the + strand over the – strand).
e. Write alignment statistics to the command line as before. Extend the output to include the number of concordant and discordant alignments. Compare the results to those of task 1 in a discussion. Make sure to include observations on the following questions: Have the results improved? What is the reason for this improvement/lack of improvement? How could the program be refined further to improve alignment specificity?

For task 2 you should submit the program (*aligner2.py*), the alignment output (*aligment2.txt*) when run on data set *reads_R1.fa*/*reads_R2.fa* and reference file *reference.fa*, and the alignment statistics along with a short discussion of the outcome (*performance2.txt*).
Task 2 is worth 5 marks.

Task 3. The Lecture "Alignment" demonstrates how inexact matching can be performed with the Needleman-Wunsch and Smith-Waterman algorithms. Consider a function that takes two strings as input. The function is then to perform a modified Needleman-Wunsch alignment of the two strings with

the gap and mismatch costs given below. The function assumes that string 2 is shorter than string 1 (return an error if not). The alignment is to be a full-length alignment of string 2 (end to end) against a local area of string 1. That is, every base of string 2 has to be matched against any number of bases in string 1. The costs for the alignments are to be implemented as 1 for a mismatch, and 1 for an insertion or deletion gap.

Write a python program *aligner3.py* that uses your implementation of above Needleman-Wunsch variant and extends *aligner1.py*.

a. The inputs to the program are as in task 1: The reference, a read file (both in FASTA format).
b. For each read the aligner is to identify the equal best mapping coordinates within the reference (tied alignment costs) according to your dynamic programming solution.
c. The output is to be written to *alignment3.txt*, and modified from task1 as follows:
   Each line should contain an additional field *MAPPING_COST* in the output. That is, each line should report *READ_NAME, REF_NAME, POS, STRAND, NUMBER_OF_ALIGNMENTS, MAPPING_COST.*
d. As before, the coordinates of the left-most best alignment are to be reported.
e. Write alignment statistics as before and include additional discussion of the mapping costs for reads: what is the highest observed cost, the average over all reads, the median (make sure to weigh your observations by the number of alignments)? Further, include a discussion of the changes in alignments in comparison to task 1. Are the changes as expected? Also address the following question: Are all of the reported alignments sensible? How could the output space be controlled?

For task 3 you should submit the program (*aligner3.py*), the alignment output (*aligment3.txt*) when run on data set *reads.fa* with the reference *reference.fa*, and the alignment statistics along with a discussion of the outcome (*performance3.txt*).

Task 3 is worth 5 marks.

Task 4 (Bonus questions)

- From what organism is the sequence in the file *reference.fa* taken from?

Notes:

- The alignments in tasks 1, 2 and 3 should be reported as 1-based coordinates. That is, a read that align to the first base of the reference should have position 1.
- All alignments are reported with respect to the forward strand of the reference. That is, if a read aligns to the reverse strand, the position should still reflect how far away the read is from the 5' end of the reference. To

illustrate the point. The output to the file *example1.fa* should look like this:

```
r1	ref	1	+	2
r2	ref	1	-	2
```

Observe that the example reads are indeed the reverse complement of each other, and should therefore map to the same position.

- Similarly, the tiebreaker in task 2 is to sort on forward alignments first, and reverse alignment second and identify the minimum such alignment. The result for *example2_1.fa* and *example2_2.fa* should read:

```
r1	ref	1	+	3	ref	50	-	3
```

- A read can be palindromic. A palindromic read has the same sequence as its reverse complement. For example, “ACGT”. To report an alignment for such a read, default to the forward strand of the reference.
- If you develop the program for tasks 1-3 one your own computer, make sure that it also runs on the MSE machines, as this will be the environment we test in.