SCIE2100 Practical 4, Week 7

Aim

In the previous practical you implemented dynamic programming algorithms for the local and global alignment of two sequences. In this practical you will use the popular multiple sequence alignment tool ClustalW to construct multiple sequence alignments. You will construct a motif from a multiple sequence alignment and play with two different representations of sequence motifs.

Files

For this practical you will need this guide, an updated *sequence.py*, and four data files: *dna.matrix*, *multiAlign.fasta, motifSequences.fasta* and *motifSearch.fasta*. The *sequence.py* updates are explained later in the practical, and the FASTA files contain sequences for multiple alignment and motifs.

Dynamic Programming Multiple Sequence Alignment

It is possible to extend the algorithm we used to align pairs of sequences in the last practical to align an arbitrary number of sequences. The method is the same, except that to align *n* sequences, we need an *n*-dimensional score matrix. For two sequences there were three options for each alignment position: gap in one, gap in the other or a match. For three sequences, this becomes seven: gap in one sequence (three options), gap in two sequences (three options) or full alignment.

The *sequence.py* file for this practical has a new function, `tripletAlignGlobal`, which is similar to the alignment function you worked with last week except that it takes and aligns three sequences using the method discussed above. Have a look at the code to see how this has been implemented.

1. Take the three sequences below, each 30 base pairs long, and align them using the triplet-wise alignment function. If you are having trouble using the function, have a look at the code you used to perform pair-wise alignment in the previous practical (you will need a substitution matrix, e.g. *dna.matrix*).

   Submit (Q1): What is the difference in the time it takes the function to run, compared to pair-wise alignment? (Pair-wise align two of these sequences as a comparison.)

   ```
   AGCGCGATTATATAAGACGGACGGCTAAAG
   AGCGGATATTTATATCGCACGACGACTACG
   GGATCGATTATATAGCCTGGACGAGACATG
   ```

Tip: Here is how you can time a section of your code:

```
import time
start_time = time.time()
# ... do stuff
end_time = time.time()
print("Elapsed time was %g seconds" % (end_time - start_time))
```

Note: These sequences, as well as the ones below, are given at the bottom of *sequence.py*, so you don't have to copy and paste them from this guide.

2. Now try aligning the longer three sequences below, each twice as long as the above sequences. Would you expect this to take twice as long, and if so, does it seem to? How long do you think sequences 120 base pairs long (double the length of these sequences) will take to align? (There are sequences 120 base pairs long given in *sequence.py*, if you want to see for yourself.)

```
AGCGCGATTATATAAGACGGACGAGACGACAGCGCGATAGACGAGACGGACGAGACGACT
GGATCGATTATATAGCCTGGACGAGACATGGGATCGAAGACGACCTGGACGAGACATGAC
AGCGGATATTTATATCGCACGACGACTACGAGCGGAAGACGAGTTTCGCACGACACTACG
```

Submit (Q2): What is the difference in time it takes the alignment to complete, doubling the size of the sequences? Try to express the time taken mathematically in terms of the sequence lengths. (Let *N* be the sequence length of each sequence.) **Tip:** think about the number of cells that needs filling out when the program completes the "scoring matrix".
According to your mathematical formula, how long would it take to align the three sequences if you double the lengths again?

Formally analysing how long algorithms take to run and how much memory they use is called the study of computational complexity, and it plays an important role in programming. In a field like bioinformatics, where the amount of data being analysed is huge and constantly growing, developing algorithms and methods that are practical in terms of time and memory usage is vital to the progress of the field.

ClustalW and Multiple Sequence Alignments

ClustalW is a popular program for aligning many sequences. We are going to use it to build an alignment of a number of similar sequences that come from different species but are all part of a conserved gene.

ClustalW implements a method known as 'progressive multiple alignment', the idea of which is to build up a multiple alignment by progressively aligning similar sequences. The algorithm starts by pair-wise aligning all sequences to obtain similarity scores, from which a guide tree is then built. This tree is used to determine which sequences are aligned to each other, starting with the most-similar sequences; aligned sequences form a 'cluster', which can then be used for further alignment.

Because this approach relies on a series of pair-wise alignments, it avoids some of the computational issues discussed above. However, it does so at the cost of accuracy; there is no guarantee that ClustalW will produce optimal alignments.

Using ClustalW from Python

ClustalW is another online biological data tool that can be accessed via Python, as we did with sequence search and retrieval in the previous practicals. It is possible to use a web form (at http://www.ebi.ac.uk/Tools/msa/clustalw2/, if you want to see) to use the program, but if you wanted to align a large number of very large sequences, or perform a large number of alignments, it would be quite cumbersome to use the web page interface.

The *sequence.py* included for this week contains a function, `runClustal`, which takes a list of sequence objects, a string indicating the type of the sequences (either `'protein'` or `'dna'`) and optionally either the `'fast'` or `'slow'` parameter, and returns an Alignment object, just as your sequence alignment function did in the previous practical.

We are not using many sequences, and they are relatively short, so the slow option is fine for us. For more numerous and larger sequences, the fast option will perform the alignment quicker at the cost of a possibly less optimal alignment.

The supplied *multiAlign.fasta* file contains a number of sequences, which are all nucleotide sequences from different species, but are all from the highly conserved FOXP2 gene. We will use ClustalW to construct a multiple alignment of these sequences, and then evaluate the alignment.

3. Use the `readFastaFile` function to load the sequences from *multiAlign.fasta*. You should know how to do this by now (if not, have a look at a previous practical where code was given).

   Now use the `runClustal` function to submit these sequences to ClustalW and get an alignment as a result. Save the alignment to a file using `writeClustal`.

   Hint:
   ```
   aln = runClustal(seqs)
   aln.writeClustal('multiAlign.aln')
   ```

Important: Please be patient while waiting for the alignment to be completed as it may take several minutes. If you stop Python, you will still not be able to submit another job until the previous one has been completed.

4. You can open the alignment file you just saved in a text editor such as TextWrangler or Notepad to have a look at the alignment, but it is difficult to evaluate the alignment when it is split into multiple blocks.

   Instead, we will use the `writeHTML` function you used in the previous practical, to write the alignment to a HTML file that will (via a browser) show the alignment all in one block, and will colour code it for easier evaluation.

   Have a look at the alignment in a web browser, and see what you think of it.
   Submit (Q4): Answer the following questions:
   a. Does this seem like a reasonable alignment? (**Hint:** You will need to consider and define what a 'reasonable alignment' is in this context. Include this definition in your answer.)
   b. Do the sequences look to be highly conserved?
   c. Does it seem like any sequences don't belong? Which? Justify.

Motifs

We are now going to introduce motifs, and find a particular one in an alignment of a given collection of sequences. You will investigate two different methods of representing motifs, and then use them to search other sequences.

The supplied *motifSequences.fasta* file contains a number of short DNA sequences, which all contain a similar binding site. There is experimental evidence to suggest that they are bound by the human transcription factor MEF2. Aligning these sequences should help us to identify similarities between them, and therefore hopefully find a motif for the binding site.

5. Firstly we will align the sequences in *motifSequences.fasta*. As above, load in the sequences from this file and then align them using the ClustalW web service. Save the alignment to a file so you don't have to align the sequences more than once, to *myAlign.aln*.

   Open the alignment file in a text editor such as TextWrangler or Notepad, and try to identify a motif in the sequences. That is, can you find a highly conserved region of the aligned sequences? This is the binding site. We will now explore ways of describing it.

A **sequence logo** is one way of visualising a multiple sequence alignment that helps identify motifs by showing conservation in sequence positions. Each position in the alignment is given a space on the horizontal axis of the graph, and a number of letters are shown in that column, their size proportional to how likely they are to appear in that position. A highly conserved region or motif will clearly show up as a series of large letters.

6. Go to http://weblogo.threeplusone.com/create.cgi, and paste your alignment in Clustal format into the box at the top of the page. There are many settings to play with to customise the generated logo, but for now just leave them and click the Create Logo button on the right. Look at the logo, and relate it to the multiple sequence alignment.

   Play around with the settings and try to create a sequence logo that shows the motif as clearly as possible. You can save your best sequence logo by right clicking the image and selecting 'Save Image.'

   Submit (Q6): Provide your logo. Identify which positions are more "informative", i.e. are helpful to check if a new sequence is tested for similarity.

Previously, you aligned a number of sequences that all contained a similar binding site, and then saw that the binding site could be represented as a sequence motif.

motifSearch.fasta contains a number of sequences which contain promoters which are believed to be bound to by the MEF2 transcription factor. To find the binding site within them, we will search them using the binding site motif from the DNA sequences above.

We will now use two different methods to build a representation of this motif and then search the sequences mentioned above. You can then compare how well the two methods work to identify the binding sites.

The *sequence.py* for this practical contains two new classes related to motifs, Consensus and PWM. These represent consensus sequences and position weight matrices, respectively, which are two ways of representing motifs.

Consensus sequences and regular expressions

Looking at the alignment file from the previous practical in TextWrangler (i.e. *myAlign.~~fasta~~*), consider each column of the multiple alignment, and pick the most likely option based on the symbols in the column: this is your consensus sequence. If there always or almost always is a certain symbol in this column, a consensus sequence is good enough. If there always is one of two or three symbols in the spot, you need a more “flexible” representation. “Regular expressions” allow you to list the different symbols (in square brackets). If all symbols are likely, a regular expression would use a “wild card” (you put a period ‘.’ in that spot).

As an example, consider the following alignment:

```
CTAG
GTAA
TTAG
```

A suitable regular expression for this alignment would be `'.TA[AG]'`. The first space can have any nucleotide (use a period), the second and third T and A always (use those symbols), and the last column either an A or G (use square brackets).

7. Construct a regular expression from the multiple sequence alignment you just obtained. You should do this just for the highly conserved region of the sequences (so your motif should only be about 5 to 10 places long), instead of the entire sequence. Look at your logo as well to inform yourself of what positions that a genuinely informative.

   Submit (Q7): Give both the consensus sequence and your regular expression. Justify the length and composition of the latter.

8. Now search the sequences given in *motifSearch.fasta* using your regular expression. You can search by creating a new Regexp object with the regular expression as its argument, and then using its `search` function, supplying the sequence (as a Sequence object or just a string) you want to search:

   ```
   re = Regexp('.TA[AG]')
   results = re.search('CTAGAGCGCTAAGCCGA')
   print results
   ```

   Hint: First load the sequences in from the file just as you have done before. Then you might like to loop through the sequences using a `for` loop, searching them with your regular expression one by one.

   Because a regular expression (or a consensus sequence) doesn’t have partial matches (a section of sequence either matches it or it doesn’t), you may have to give a ‘looser’ definition than you might initially think (e.g., make the consensus sequence shorter, and include more square bracket options).

The `search` function returns its matches as a list of lists, each one containing three items: the index in the sequence where the match occurred, the matched section itself, and the match score. For regular expressions, the match score is always 1 because there are no partial matches.

Position Weight Matrices (PWM)

In a previous practical you constructed a substitution matrix based on a multiple sequence alignment. The substitution matrix used all the information in the alignment to calculate a score for aligning any two pairs of amino acids, without regards to their location. Position weight matrices are constructed by essentially the same means (mathematically speaking), except that a score for each position is calculated for each column in a multiple alignment. This represents a motif in a more flexible way than a consensus sequence.

The PWM class works the same as the Regexp class you just used, except that it takes an Alignment object as an argument instead of a sequence, and generates scores for all symbols and columns in the given alignment. You can use also supply start and end indices, if you don't want the motif to extend across the entire alignment. Look at the alignment and choose the most conserved 8 to 10 columns as start and end positions.

9. Use the same multiple sequence alignment to create a PWM, using only the most conserved region of the alignment. Search the same sequences in *motifSearch.fasta* as you did using regular expressions in the previous section.

   Hint:
   ```
   aln = readClustalFile('myAlign.aln', DNA_Alphabet)
   pwm = PWM(aln, start=10, end=20)
   results = pwm.search(mySequence)
   print results
   ```

   Submit (Q9): Present your PWM. State whether you can roughly identify by eye where in the alignment the binding site is. Identify this location (either by highlighting your alignment, or providing start/end indices for the relevant substring of the alignment).

   Hint: The `__str__` function of the PWM class returns the matrix itself in a printable format, showing all the scores for each symbol in each column of the alignment. Use it by '`print pwm`'. Note that the scores are log-odds scores, and so are negative for unlikely symbol positions and positive for likely ones.

Comparison

You may have had a hard time evaluating your results from the above searches, because you don't know where the actual binding sites are in the sequences being searched. If you open *motifSearch.fasta* in the text editor TextWrangler, however, you will see that each sequence has a section in uppercase. This is the binding site.

10. Now that you know what you're looking for, compare the two methods of searching (regular expression and PWM), seeing how well they identify the

binding sites. You may need to go back and adjust your consensus sequence now that you know what you're looking for.

You had to construct the regular expression yourself by inspection of the multiple sequence alignment, whereas the PWM is generated mathematically from the data.

Submit (Q10): State with justification which motif representation you think works best, and which is likely more accurate. Discuss the potential uses of the two representations.

Assessment

Respond to the problems marked with **Submit** above tasks in a **PDF or Word document** and submit it to Blackboard as an **attached file** to the assessment by the due date. You must clearly define any symbols or variables used in formulae and calculations in your answer to each question.