

Lecture 3

In this lecture we looked at ideas for how to implement recursion and dynamic programming to align two DNA sequences, and calculate the optimal score. Here is a quick writeup of some of the ideas we considered.

For the entire lecture we used the simple example which consisted of the following sequences:

- Sequence 1 = ACG
- Sequence 2 = AGT

The scoring system we used was:

- +2 for a match
- -1 for a mismatch
- -2 for a gap

First we look at computing the number of alignments we need to check in our recursive code. We decided that we could either compute this number in advance, or we could count the alignments as we score them later. If we wanted to compute the number of them upfront we needed to use the following function:

$$al(i, j) = al(i - 1, j - 1) + al(i - 1, j) + al(i, j - 1)$$

In our example we wanted to calculate $al(3, 3) = al(2, 2) + al(2, 3) + al(3, 2)$. We need to recursively call the same function to calculate the values of $al(2, 2)$, $al(2, 3)$ and $al(3, 2)$. Eventually we will reach one of our base cases of our recursive function, that is something of the form $al(i, 0)$ or $al(0, j)$. There is only one way to align a non-empty sequence with an empty sequence, so $al(i, 0) = al(0, j) = 1$. You can try calculating some examples by hand if you wish. The values of $al(0, 0)$, $al(1, 1)$, $al(2, 2)$, ..., $al(10, 10)$ can be found in the table on slide 11 of lecture2.pdf.

Next we took a look at how we might implement our recursive algorithm for working out the alignments and their scores. First we must fix the last position of our alignment - either by using a letter from the end of each sequence, or matching one of them against a gap.

1. Call our function f , giving it our two sequences as input, and an empty alignment.
i.e. $f(< , >, < ACG, AGT >)$
2. Fix $\frac{G}{T}$, recursively call our function - $f(< G, T >, < AC, AG >)$
3. Fix $\frac{G}{_}$, recursively call our function - $f(< G, _ >, < AC, AGT >)$
4. Fix $\frac{_}{T}$, recursively call our function - $f(< _, T >, < ACG, AG >)$

Each of these recursive function calls would again create three function calls, with one (or both) of our sequences reducing in length at each step, and the size of the alignment growing. Eventually we reach a function call where one of the sequences is now empty. This is a base case for our algorithm - once one of the sequences is empty we know the rest of the alignment will just match gaps against characters.

We decided there were multiple ways in which to score our alignments. We could score each element of the alignment at each recursive function call, and add them up to get the score of that alignment. Alternatively we could score an alignment once it is complete i.e. once we've reached a base case. Finally we could decide to store our alignments in some array, and score them all once we've created them all.

Secondly we looked at how to implement Dynamic Programming. We need some formulae from lecture 2.

Remember that $s(i, 0) = -2i$ and $s(0, j) = -2j$ as we are just aligning characters against gaps.
All other entries in the score matrix are computed using the formula:

$$s(i, j) = \max\{c(i, j) + s(i - 1, j - 1), s(i - 1, j) - 2, s(i, j - 1) - 2\}$$

where $c(i, j) = +2$ if the characters match, or $c(i, j) = -1$ if they mismatch, as per our scoring system.

	-	A	C	G
-	0	-2	-4	-6
A	-2	2	0	-2
G	-4	0	1	2
T	-6	-2	-1	0

Compute the backtrack matrix as we go along. Initialise entries $(i, 0) = U$ and $(0, j) = L$

- Enter D if the max score came from the term $c(i, j) + s(i - 1, j - 1)$.
- Enter U if the max score came from the term $s(i - 1, j) - 2$.
- Enter L if the max score came from the term $s(i, j - 1) - 2$

	-	A	C	G
-	END	L	L	L
A	U	D	L	L
G	U	U	D	D
T	U	U	D/U	D/U

The score of the best alignment is 0 (just the score in the bottom right of the score matrix). Use the backtracking matrix to find the alignments that have that score.

- L means use last available character in Sequence 1 and match against a gap in Sequence 2.
- U means use last available character in Sequence 2 and match against a gap in Sequence 1.
- D means match the last available character in both Sequence 1 and Sequence 2 against each other.

Working from the bottom right position there are 2 solutions.

1. DDD which gives the alignment:

A	C	G
A	G	T
2. UDLD which gives the alignment:

A	C	G	-
A	-	G	T

We can easily check both of these alignments have score 0.

Finally we looked at scoring the same two sequences, but this time using Ends-free scoring.

Small change to initial conditions for ends-free alignment. Now we set $s(i, 0) = 0$ and $s(0, j) = 0$ as gaps are free at the beginning of either sequence.

Still compute other entries in the score matrix using the formula:

$$s(i, j) = \max\{c(i, j) + s(i - 1, j - 1), s(i - 1, j) - 2, s(i, j - 1) - 2\}$$

except if we are in the last row or column.

If we are in the last column (so $j=n=\text{length of Seq 1}$) we use:

$$s(i, n) = \max\{c(i, n) + s(i - 1, n - 1), s(i - 1, n), s(i, n - 1) - 2\}$$

If we are in the last row (so $i=m=\text{length of Seq 2}$) we use:

$$s(m, j) = \max\{c(m, j) + s(m - 1, j - 1), s(m - 1, j) - 2, s(m, j - 1)\}$$

The backtracking matrix is constructed exactly as before.

	-	A	C	G	
-	0	0	0	0	
A	0	2	0	0	
G	0	0	1	2	
T	0	0	0	2	

	-	A	C	G	
-	END	L	L	L	
A	U	D	L	U	
G	U	U	D	D	
T	U	L	L	U	

Working from the bottom right position there is just 1 solution.

UDLD which gives the alignment: $\begin{array}{c|c|c|c|c} A & C & G & - \\ \hline A & - & G & T \end{array}$

We can easily check that this alignment has score 2.