

1 Lecture 10, April 22, 2002

I think that in this lecture I fully discussed how to efficiently compute the optimal local alignment of two strings. The notes on this are under lecture 8 where we discussed local alignment without fully discussing how to compute it.

2 Lecture 11, Efficiently computing end-gap free alignments.

An end-gap in an alignment is a gap (a run of spaces) in a string S , so that either all the characters of S or to the right of the gap or to the left of the gap. In an alignment, there can be 0, 1 or 2 end-gaps. In end-gap free alignments, we do not wish to count the spaces that appear in either of the two end-gaps (if there are any).

So the objective function is

Max (#matches - #mismatches - #(spaces not in an end-gap)).

In class we discussed a few alignment applications where the end-gap free objective function was required. Then we showed how to efficiently (using about $n*m$ operations when the two strings are of lengths n and m) compute the optimal end-gap free alignment.

The main points are that we set $V(i, 0) = V(0, j) = 0$; that we use the general recurrences from global alignment; that we find the optimal value in row n or column m of the DP table: the largest value in row n or column m is the value of the optimal end-gap free alignment. To actually construct the alignment, do a backtrace from that largest value, until you reach either row 0 or column 0, forming the alignment in the usual way.

I mentioned in class that I had a good midterm question that requires understanding the logic of how to compute end-gap free alignment.