

Sequence Alignment, cont.

3. Scoring Algorithms. Dynamic Programming.

As mentioned earlier, there are too many possible alignments (an exponentially large number) of two sequences of lengths m and n , especially when one considers the sequences are considered as a query sequence and a reference sequence, where the reference sequence is actually a data base of sequences, such as GenBank at the NCBI. This database is, itself, growing exponentially. Fortunately, the linearity of the strings gives a way to make the computation necessary very efficiently (in time $O(mn)$).

4. Variants.

Last time we covered the **Needleman-Wunsch** dynamic programming algorithm for finding the optimal global sequence alignment. Today we

will discuss a number of variants which are useful.

A.) The first important variant is the **local alignment** algorithm, or **Smith-Waterman** algorithm. This is meant to find the subsequences $\mathbf{a}' = (a_{j_i}, \dots, a_{j_k})$, $\mathbf{b}' = (b_{\ell_1}, \dots, b_{\ell_s})$ which have the highest possible alignment score, taken over all possible subsequences, and all possible alignments of the subsequences. Again, we could try to solve this by redoping the global alignment algorithm for each subsequence \mathbf{a}' , etc., but this would again be exponentially difficult. Fortunately there is a simple trick which avoids this. (The Smith-Waterman paper is very short, but it saved people a lot of time over what they had been doing at the time.)

The set-up is the same as before. We make one additional assumption: at least one of the scoring matrix entries is going to be positive: some $s(a, b) > 0$. Then we are expecting to find a score for best subalignment which is

positive. Whereas the NW algorithm said, in the key inductive step,

$$F(i + 1, j + 1) = \max \begin{cases} F(i, j) + s(b_{i+1}, a_{j+1}) \\ F(i + 1, j) - d \\ F(i, j + 1) - d, \end{cases}$$

for SW, we will simply modify this to

$$F(i + 1, j + 1) = \max \begin{cases} F(i, j) + s(b_{i+1}, a_{j+1}) \\ F(i + 1, j) - d \\ F(i, j + 1) - d \\ 0. \end{cases}$$

Notice that the effect of this is to arrive at a square in the DP grid where one arrives at the optimal score up to that point being negative. The new alternative simply says “I can do better than this by starting over again and looking for positive terms”. The 0 entered in the grid square at this position becomes a new initialization value. Now, as before, when the grid is completed (you still have to compute out to the entire grid), you scan the grid for the maximum value in the grid. Starting from this

point, you use the arrows saved for the traceback as in the NW algorithm, and construct a traceback until the first time the traceback encounters a 0 in the grid. At this point you terminate the traceback procedure, and collect the string positions which have been aligned by this partial traceback, including the gaps. This is the optimal subalignment. The proof is the same as before. If it were not the optimal alignment, then there would be a substring with a higher alignment score. You cannot extend any SW traceback, however, because to extend through a 0 in the grid for SW would mean that you were adding a match with a negative value, lowering the score as opposed to the value computed with SW (you were doing better by taking 0 as the value rather than the negative NW value at that point).

Remark on negative versus positive scores.

If we have a scoring matrix $s(a, b)$ such that all entries are positive, then local alignment will

not work in the sense that it will likely produce nonsense in the above algorithm when applied to searching a sequence database, for example. What one usually does (it happens naturally) is to assume that the expected score at each of the independent string positions is negative:

$$\sum p_a p_b s(a, b) < 0.$$

(Notice we take the expectation with respect to the random model.) In the case when we use a probability model to set up the scoring matrix, we get

$$s(a, b) = \log \frac{p_{ab}}{p_a p_b},$$

and then

$$\sum p_a p_b s(a, b) = \sum p_a p_b \log \frac{p_{ab}}{p_a p_b} = -H(R||P) \leq 0,$$

where we get 0 in the last term iff the product distribution R is the same as the observed (related) distribution P . Recall that the H is the relative entropy, and the Gibbs inequality tells us the non-positivity noted. What you are saying here is that if the average score of a string

position is negative, then you cannot simply find illusory better matches simply by making the strings longer. In a database search this is a serious consideration.

B.) The **overlap variant** attempts to find sub-optimal alignments by finding “new” areas of alignment space. The idea here is that there may be many sequences which are close to the sequence we have found with SW say, and which score very close to the SW optimum. This may constitute a large collection of sequences, and to sift through these is usually a matter treated by hand (i.e., list the first N scores and tracebacks), or by stochastic sampling (later; i.e., one looks randomly through the highest scores if there are too many of them). We will look at this later, but the idea is to find successive local alignments which do not overlap in the grid (when you draw in the traceback curve).

C.) Affine gap penalties can be incorporated in the following way. At each grid, one now has to

compute three quantities: $M(i, j)$, $I_a(i, j)$, $I_b(i, j)$.
 These are defined inductively as follows:

$$M(i+1, j+1) = \max \begin{cases} M(i, j) + s(b_{i+1}, a_{j+1}) \\ I_a(i+1, j) + s(a_{i+1}, b_{j+1}) \\ I_b(i, j+1) + s(a_{i+1}, b_{j+1}), \end{cases}$$

$$I_a(i+1, j+1) = \max \begin{cases} M(i+1, j) - e \\ I_a(i+1, j) - d \\ I_b(i, j+1) - e, \end{cases}$$

$$I_b(i+1, j+1) = \max \begin{cases} M(i, j+1) - e \\ I_a(i, j+1) - e \\ I_b(i+1, j) - d, \end{cases}$$

Then we set

$$F(i+1, j+1) = \max\{M(i+1, j+1), \\ I_a(i+1, j+1), I_b(i+1, j+1)\}.$$

Note that here we have used d for the continuation of a gap penalty, and e is the gap initiation penalty. Recall that the second entry of the matrix represents the sequence a 's entries.