Lecture 2 - Sequence alignment

Gidon Rosalki

2025-10-21

1 Sequence alignment

Let us suppose that we have three sequences:

1. CTAACTG...
2. GAGTG....
3. GACTG...

As we can see, sequences 2 and 3 appear to be very similar, they appear directly above each other, and only have one letter of difference. What is less obvious, is the similarity between sequences 1 and 2. If we begin the second sequence from the first A of the first sequence, then they are remarkably similar, with only the difference of 2 letters. Additionally, we could just add 2 spaces between the first and second letters of the second sequence, and once again have a great degree of similarity between them.

Let there be $s^*, t^* \in \{A, C, G, T, -\}^*$, such that $|s^*| = |t^*|$ (as in, two sequences of letters of the same length). Let us define a function *remove* such that

$$remove\left(s^*\right) = s$$

Effectively, the star indicates we are in the extended alphabet, that contains spaces. We may *align* sequences through adding spaces. These alignments need not be minimal, for example:

$$\begin{bmatrix} A & A & C & T & - & - & - \\ - & - & - & - & A & G & T \end{bmatrix}$$

is an acceptable alignment. Not useful, perhaps, but still acceptable. Our only limitation regarding alignments is that we may not match (defined below) 2 spaces.

Let us consider the following matching:

$$\begin{bmatrix} C & - & G & A & G & T & G \\ C & T & A & A & C & T & G \end{bmatrix}$$

In the first position, we have a "match". Second position, where there is T to $-$ we have an "indel". In the third position, A and G do not match, and so we have a "mismatch".

In order to define how good a match is, we need to give scores to each of the options match, indel, and mismatch. These are effectively learned parameters. Our question here would be "Given s, t, σ, find the alignment with the maximum score".

1.1 Dynamic programming

Since we learnt this in *Algorithms*, we will go over this very quickly. The idea is that we want to split a large problem recursively down into smaller problems, that we can then solve quickly, and use their results to solve larger problems. How would we solve this with dynamic programming? Let us suppose the sequences s, t, such that

$$s = s_p T$$
$$t = t_p T$$

There are 3 options we may take when making s^*, t^*:

$$s_p^* T$$
$$t_p^* T$$
$$====$$
$$s_p^* -$$
$$t_p^* T$$
$$====$$
$$s_p^* T$$
$$t_p^* -$$

So, we may now solve

$$V(s,t) = \max \begin{cases} V(s_p, t_p) + \sigma(T,T), & \text{s.t. } |s^*| = |t^*| \\ V(s_p, t) + \sigma(T,-), & \mathit{remove}(s^*) = s \\ V(s, t_p) + \sigma(-,T), & \mathit{remove}(t^*) = t \end{cases}$$

From here, we may build the table, such that

$$V[0,0] = 0$$
$$V[1,j] = \mathit{maxscore}\{s[1:i], t[1:j]\}$$
$$V[i,j] = \max \begin{cases} V[i-1,j-1] + \sigma(s_i, t_i) \\ V[i-1,j] + \sigma(s_i, -) \\ V[i,j-1] + \sigma(-, t_i) \end{cases}$$

The best score will appear in the bottom right hand corner of the table. However, this is just the score, and not the chosen matching. In order to find the chosen matching, we must work backwards through the table. We may work backwards to any of the following three cells: above, left upper diagonal, and left. We remember the paths that lead us to each cell, and may then chose whichever path that leads to the final cell that we like.

Given $|s| = n, |t| = m$, this will use $O(n \cdot m)$ space, and $O(n \cdot m)$ time. This is polynomial, so in the eyes of the algorithms course, wonderful! However, it is nonlinear, so in the eyes of DAST, not so good. Additionally, we know that m and n are usually very large, so this is not great...
Since space requires hardware, whereas time just requires us to wait, we will worry about space first, and then time, since we can (probably) just wait longer.

1.1.1 Linear space alignment

The idea is that if we can know where the optimal alignment crosses the halfway points of the sequences, then we have split the problem into two smaller problems. So instead of

$$V[1,j] = \mathit{maxscore}\{s[1:i], t[1:j]\}$$

Let us define

$$U[1,j] = \mathit{maxscore}\{s[i+1:n], t[j+1:m]\}$$

So, the value of

$$V\left[\frac{n}{2}, j\right] + U\left[\frac{n}{2}, j\right]$$

From this, we will get the point that the optimal path crosses $\frac{n}{2}$. Now we have split the problem into the two sub problems, we can now repeat recursively.
Now, how is this done in linear space? Well, we only need to save one cell in every column, so this will take $O(n)$ space. However, since we still do this calculation for every cell in the matrix, the time will still be $O(n \cdot m)$.

1.2 Different alignments

There are two main types of alignments. Global alignment, and local alignment. Global alignment is interested in finding the alignment between the entire two sequences and is what we have discussed until now. Local alignment is looking if there are local sections within the sequences that align. For local alignment, let us define V as follows:

$$V_l\left[i,j\right] = \max_{l\leq i, k\leq j} score\left\{s\left[l:i\right], t\left[k:j\right]\right\}$$

$$V_l\left[i,j\right] = max\begin{cases} V_l\left[i-1,j-1\right] + \sigma\left(s_i, t_j\right) \\ V_l\left[i-1,j\right] + \sigma\left(s_i, -\right) \\ V_l\left[i,j-1\right] + \sigma\left(-, t_j\right) \\ 0 \end{cases}$$

Our matrix is now always non-negative.