Bioinformatics Coursework: Part 2

1 My Cost Function

A cost function for comparing strings of DNA should account for mutations and slicing. Mutations can reasonably be accounted for by a simple one letter substitution score. Some mutations may be more likely than others, this can be encoded into the costs (an unlikely mutation would have a high cost). Slicing, however, is less simple. Intuitively the sequences:

$$S_1 = ATAGCCATAAAGCC \tag{1}$$
$$S_2 = ATAGCCGAGAAAATAAAGCC \tag{2}$$

Could very likely be closely related, ie. If our cost function was trying to represent how closely related two animals' DNA is (if you wanted a cost function based on how similar the things the two sequences coded for you would probably need something very different). Notice S1 could have the slice GAGAA inserted into the 7th position and be identical to S2. Surely this should not have a high cost as 5 indel operations. For this reason, define a cost function as before, but now check if the previously aligned characters were indels '_'. Consider computing the score for two sequences with indels in one of them.

$$\begin{array}{rl} & 3\ \ 2\ \ 1\ \ 0\ \ 0\ \ 0 \\ S_1 = & A\ T\ A\ G\ C\ C\ \_\ \_\ \_\ \_\ \_\ \_\ A\ T\ A\ A\ A\ G\ C\ C \\ S_2 = & A\ T\ A\ G\ C\ C\ G\ A\ G\ A\ A\ A\ A\ T\ A\ A\ A\ G\ C\ C \end{array}$$

1. If the previous character is not an indel add a cost of 3 (this corresponds to the first indel).
2. If the previous character is an indel, but one before is not, add a cost of 2 (corresponds to the second indel).
3. If the previous character is an indel, but one before is not, add a cost of 1 (corresponds to the third indel).
4. Any following, consecutive indels has cost equal to 0.

This has the effect of assuming long missing sections are from one slicing action, given a penalty of 6 (=3+2+1) as a pose to a much larger cost from a single letter substitution score from many indels.

2 Implementing My Cost Function

I now show how to implement this cost function with dynamic programming (with space and time complexity $\mathcal{O}(nm)$, m and n are the lengths of the two sequences). I assume the sequences are from an alphabet of "ABC", and I will use the following single letter substitution score matrix:

$$\begin{array}{c} \\ A \\ B \\ C \\ \_ \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & \_ \end{array} \\ \begin{pmatrix} 2 & -2 & -2 & -3 \\ -2 & 2 & -2 & -3 \\ -2 & -2 & 2 & -3 \\ -3 & -3 & -3 & 0 \end{pmatrix} \end{array}$$

A better substitution matrix could be designed for use comparing DNA sequences that penalises the less likely pairs more.

2.1 Algorithm

The algorithm works similarly to the standard quadratic space and time DP algorithm. The change is that, at each step, it considers the indel options (up and left) and checks if they continued beforehand, ie. if it was checking the up indel, we follow the pointers as long as they point in the up direction, and report how many there were. This is the job of the traceback function. This allows us to give different scores depending on how many indels there are from the max alignment back from that point.

Algorithm 1 Dynamic programming implementation of my cost function

Data: Two sequences S_1, S_2 only containing letters "ABC"
Result: Score, Path_1, Path_2
pointers ← [n × m matrix of 2's], sc_matrix ← [n × m matrix of 0's]
for *j = 1:n* **do**
  for *i = 1:m* **do**
    upcount = traceback(i-1, j, 'up')
    case_0 = *sc_matrix*[i-1, j] + (if upcount == {0: delete(S_1[i-1]), 1: -2, 2: -1, else: 0})
    leftcount = traceback(i, j-1, 'left')
    case_1 = *sc_matrix*[i, j-1] + (if leftcount == {0: insert(S_2[j-1]), 1: -2, 2: -1, else: 0})

    local_options = [case_0, case_1, 0, *sc_matrix*[i-1, j-1] + match(S_1[i-1], S_2[j-1])]
    pointers[i, j] = argmax(local_options)
    matrix[i, j] = local_options[pointers[i, j]]
  end
  mx ←max(sc_matrix[:, *j*])
  if *local_max* < *mx* **then**
    local_max ←mx
    max_pos ← (argmax(sc_matrix[:, *j*]), *j*)
    end
  end
  return fulltraceback(max_position)

I have defined the pointers to have codes for each direction: 0 = up, 1 = left, 2 points to itself (an alignment starts here), 3 = diagonal. This is so that the argmax for local_options will select any traces of previous indels with equal score to either of the other two options (starting afresh and matching (diagonally)). The functions: delete(a), insert(a) and match(a, b) correspond to the substitutions scores for score(a, '_'), score('_', a) and score(a, b).