17 Multiple sequence alignments (MSA)

In pairwise alignment, we were given two sequences, x and y and wanted to align them by judiciously inserting gaps so that homologous residues were lined up with one another.

Multiple alignment is similar except now we have $k \geq 3$ sequences to align, so we want to form columns of homologous residues by adding gaps to each sequence.

Experts can construct multiple alignments by hand by considering a number of factors like secondary and tertiary protein structure, highly conserved regions, patterns of gaps, evolutionary processes etc. This is, however, subjective, difficult and tedious.

We want to come up with a method that is probabilistic, automatic and produces alignments that experts are happy with.

A good entry point to understanding the problem of global alignment is the `Phylo` puzzle game at `http://phylo.cs.mcgill.ca/` where you work on short multiple alignments by hand.

17.1 Dynamic programming

It is tempting to try to extend the dynamic programming methods that we used for pairwise alignments to the MSA problem. It is relatively easy to write down the naive extension of pairwise alignment algorithms to more than 2 sequences. But the naive implementation quickly becomes impossible as the number of sequences and the length of the sequences increases.

For example, for 2 sequences of length L, the Needleman-Wunsch algorithm requires a 2-dimensional array with L^2 cells in total to be stored in memory. The MSA analogue on n sequences requires storing an n-dimensional array containing L^n cells in total. Even for short sequences of $L = 100$, we would require memory for $100^5 = 10^{10}$ cells (at 4 bytes per cell, that's about 37 GB).

Some clever work from Lipman, Altschul and Kececioglu (the first two co-wrote BLAST) managed to reduce the size of the space that needs to be considered. That is, instead of calculating all L^n cells, they calculate upper and lower bounds on the score of the best MSA and then need only calculate the cells in the n-dimensional array that will produce scores lying between these two bounds. This work allows a few sequences (5-10) of moderate length (300 residues) and not too far diverged to be optimally aligned but even this requires a large computational resource.

17.2 Progressive alignment

Finding the optimal MSA is computationally prohibitive, as discussed in the previous Section. Typically, we resort to finding a good-enough alignment using heuristic techniques. The most widely used heuristic is progressive alignment.

Progressive alignment involves a series of successive pairwise alignments. At it's most basic, an initial pair of sequences are chosen and aligned, a third is chosen and aligned to

the first two and so on until all sequences are included in the MSA. Other methods also allow the aligning of two alignments to each other. For example, if there are 4 sequences, two pairs may be aligned first then the two alignments aligned to complete the MSA.

These methods require that we can: decide on an order in which to align the sequences, align two sequences together, align a sequences to a MSA and align two MSAs together.

The typical way to decide on the order in which to align the sequences is to build a guide tree, using a clustering methods such as UPGMA, and align sequences in the order that nodes occur from the leaves to the root of the tree.

17.3 Building trees with distances and UPGMA

UPGMA is a method of building trees based on distances: we are given a set of objects, and for each pair of objects we have some measure of the distance between them.

UPGMA stands for unweighted pair group method using arithmetic averages and is a simple method with an ugly name. The idea can be thought of as a clustering algorithm where we start with all individual sequences and start clustering them together, building the tree up from the leaves to the root. The height of the internal nodes (or, equivalently, the edge lengths) is determined by the distances between the two clusters being joined.

For two sequences, x and y, we assume we have a method of defining the distance d_{xy}. We define the distance between two clusters of sequences, C_i and C_j as the average distance between all pairs between clusters:

$$d_{ij} = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d_{xy}$$

where $|C|$ is the number of sequences in cluster C.

We define the algorithm as follows:

Initialise Assign each sequence i to it's own cluster C_i. Assign a leaf node to each cluster and give it height 0.

Repeat until there only one cluster remains Find clusters C_i and C_j such that d_{ij} is minimal (choose randomly between equidistant candidates).

Join i and j to make the new cluster $C_k = C_i \cup C_j$.

Define a node k in the tree placed at height $d_{ij}/2$ with child nodes i and j.

Update the distance matrix.

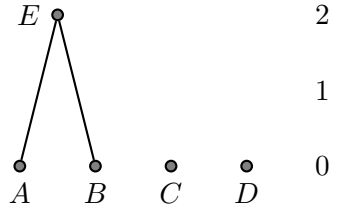
This procedure results in a well-defined tree (we need to check that all node heights are above the heights of the their children). The algorithm is quadratic ($O(n^2)$) in the number of sequences.

Example: Given 4 sequences, A, B, C and D, which have the pairwise distances given by the distance matrix d below, construct the UPGMA tree.

$$d = \begin{array}{c|cccc} & A & B & C & D \\ A & - & 4 & 8 & 8 \\ B & & - & 8 & 8 \\ C & & & - & 6 \\ D & & & & - \end{array}$$

Solution: Start by assigning leaf nodes to each of the sequences with height 0:

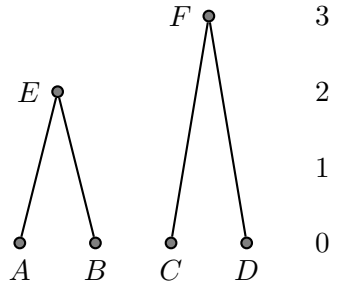
Now choose the pair of clusters that are closest to each other according to the distance matrix d. This is the pair (A, B) with distance $d(A, B) = 4$. Join the cluster $E = A \cup B = \{A, B\}$ which has height $d(A, B)/2 = 2$.



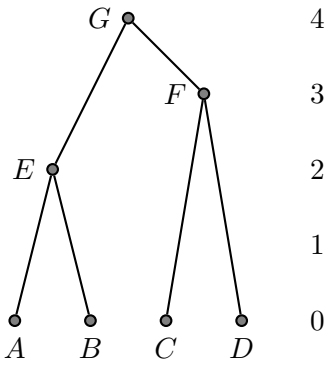
The distance matrix is by calculating $d(E, C)$ and $d(E, D)$. $d(E, C) = \frac{1}{2 \cdot 1}(d(A, C) + d(B, C)) = \frac{1}{2}(8 + 8) = 8$ and similarly for $d(E, D)$.

$$\begin{array}{c|ccc} & E & C & D \\ E & - & 8 & 8 \\ C & & - & 6 \\ D & & & - \end{array}$$

Now form the cluster $F = \{C, D\}$ and place the node at $d(C, D)/2 = 3$.



The distance matrix is now the single distance between the remaining clusters: $d(E, F) = \frac{1}{2 \cdot 2}(d(A, C) + d(A, D) + d(B, C) + d(B, D)) = \frac{1}{4}(8 + 8 + 8 + 8) = 8$. So make the last node $G = \{E, F\}$ and place it at height $8/2 = 4$. The UPGMA tree is thus



□

17.4 Feng-Doolittle progressive alignment

The Feng-Doolittle algorithm (1987) takes the approach described above. The steps for aligning n sequences are as follows.

1. Calculate the $n(n-1)/2$ distances between all sequences pairs. The distances are found by aligning each pair and recording a normalized score. The score used is

$$D = -\log S_{eff} = -\log \frac{S_{obs} - S_{rand}}{S_{max} - S_{rand}}$$

   where S_{obs} is the score from the pairwise alignment, S_{max} is the average of scores obtained by aligning each sequences of the pair to itself and S_{rand} is the expected score for an alignment of the pair when the residues are randomly shuffled. The effective score S_{eff} can thus be viewed as a normalised percentage similarity which roughly decays exponentially towards zero with increasing evolutionary distance. Thus, we take $-\log$ to make the score decay approximately linearly with evolutionary distance.

2. Build a guide tree based on the recorded scores (we use UPGMA).

3. Build the alignment in the order that nodes were added to the tree.

A pair of sequences is aligned in the normal way. A sequence is aligned to an MSA by aligning it to each sequence in the MSA and choosing the highest scoring alignment. Two alignments are aligned to each other by aligning all pairs of sequences between the two groups and choosing the best alignment.

After a sequence or group of sequences is added to an alignment, the introduced gap characters are replaced with a neutral X character which can be aligned to any other character (gap or residue) with no cost. Crucially, there is no penalty for aligning a gap to an X which tends to make gaps align with each other, giving us the characteristic pattern we see in multiple alignments of gaps clustered in columns.

Once the initial MSA has been found, we can use further heuristics to improve it and lessen any effect of the order in which sequences were added. For example, we can choose

a sequence uniformly at random, remove it from the MSA and then realign it to the the MSA. This process can be iterated until the MSA becomes stable, that is, no or very few changes occur when a sequence is removed and re-aligned