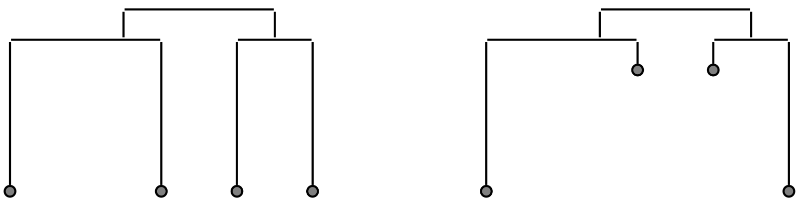
20.2 Ultrametric distances

UPGMA produces the correct tree (i.e., produces the tree along which the sequences actually evolved) if the sequences evolved according to a *molecular clock* in which sequences evolved at a constant rate over the whole tree. In that case, the number of mutations is proportional to the temporal distance of a node to the ancestor.

In these cases, the distances are said to be *ultrametric* and UPGMA will reconstruct the correct tree. The ultrametric condition is that d_{ij} is ultrametric when, for and points i, j, k, the distances d_{ij}, d_{jk}, d_{ik} are either all equal or two are equal and the remaining one is smaller.

More simply, in an ultrametric tree, the distance from the root to the leaves is the same for every leaf. So if all leaf nodes are sampled at the same time and the ultrametric property holds, the tree displaying the distances will have all the leaf nodes at the same level.



The tree on the left is ultrametric, the tree on the right is not.

In most cases, the ultrametric assumption is not a good one, as different regions of sequences vary at different rates and different lineages of the tree may have different rates of mutation. Thus, UPGMA will not reconstruct the correct tree in most cases.

20.3 Additive distances

A less stringent condition is that distances are *additive.* A tree is said to have additive edge lengths if the distance between two leaves is the sum of the edge lengths connecting them. You can show that ultrametric distances are additive but the reverse does not hold. In an additive tree, the *four point condition* is satisfied, in which any four leaves can be relabelled so that $d(x, y) + d(u, v) \leq d(x, u) + d(y, v) = d(x, v) + d(y, z)$.

A set of additive distance can be thought of as tree-like — there is a tree that correctly displays those distance as branch lengths.

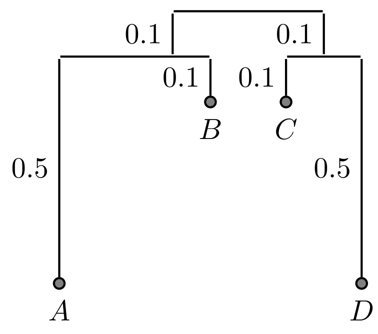
So the question is, given a set of additive distances, can we reconstruct the correct tree?

20.4 Neighbour joining

The answer turns out to be yes, and the algorithm that lets us achieve this is known as *neighbour joining* (NJ). NJ is similar to UPGMA but instead of simply using a pairwise evolutionary distance matrix, NJ takes that matrix as a starting point and then builds a rate-corrected distance matrix before proceeding to join nearest neighbours.

First, note that to find the nearest neighbour on a tree, it is not sufficient to simply calculate the smallest distance.

Example: Consider the tree



from which we derive the pairwise distance matrix

$$d = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 0.6 & 0.8 & 1.2 \\ & 0 & 0.4 & 0.8 \\ & & 0 & 0.6 \\ & & & 0 \end{pmatrix} \end{array}.$$

If we try to reconstruct the tree using UPGMA, the first step is to choose the pair with the smallest distance and join them. That has us choosing B and C first as sharing a common ancestor before anything else. This immediately leads to the the wrong tree topology. □

To find the nearest neighbour instead of just the node at the smallest distance, we need to subtract the average distance to all other leaves. Let $D_{ij} = d_{ij} - (r_i + r_j)$ where

$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}$$

and L is the number of leaves (sequences). Note that the denominator in calculating r is deliberately one less than the number of items summed. It can be shown (with a bit of work, omitted here) that the pair of leaves i, j for which D_{ij} is minimal are neighbouring leaves.

This leads us to the NJ algorithm, which progressive builds up a tree T by keeping a list of active nodes L and finding the closest amongst them.

Neighour joining algorithm

1. Let T be the set of all leaf nodes and set $L = T$.
2. Iterate until $|L| = 2$:
    (a) Calculate (or update) D from the distance matrix d.

(b) Pick i, j for which D_{ij} is minimal.

(c) Define k so that $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ for all $m \in L$.

(d) Add k to T with edges joining to i and j with lengths $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$ and $d_{jk} = d_{ij} - d_{ik}$.

(e) Set $L = L - \{i, j\} + k$.

3. $|L| = 2$, so add remaining edge connect i, j with length d_{ij}.

To see this works, consider the reverse process where we strip away leaves from an additive tree by removing neighbouring pairs. Find leaves i, j with parent k. Remove i, j and add k to the list of leaves, defining $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ where m is some other leaf node.

Example: Perform neighbour joining on the distance matrix from the previous example

$$d = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 0.6 & 0.8 & 1.2 \\ & 0 & 0.4 & 0.8 \\ & & 0 & 0.6 \\ & & & 0 \end{pmatrix} \end{array}.$$

Solution: We first need to calculate D for which we will need r. Here $L = 4$, so

$$r_A = \frac{1}{2}(d_{AB} + d_{AD} + d_{AD}) = \frac{1}{2}(0.6 + 0.8 + 1.2) = 2.6/2 = 1.3.$$

Get other elements of r similarly so $r = (1.3, 0.9, 0.9, 1.3)$.

From r and d we can thus calculate

$$D = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} - & -1.6 & -1.4 & -1.4 \\ - & - & -1.4 & -1.4 \\ - & - & - & -1.6 \\ - & - & - & - \end{pmatrix} \end{array}.$$

D is symmetric and the diagonal is irrelevant so only need calculate either the elements of the upper or lower traingle.

The minimum value of the rate-adjusted matrix is found at AB and CD. Choose AB to merge into new node E. The length of the edge from A to E is $d_{AE} = \frac{1}{2}(d_{AB} + r_A - r_B) = \frac{1}{2}(0.6 + 1.3 - 0.9) = 0.5$ while the distance from B to E is found by $d_{AE} = d_{AB} - d_{AE} = 0.6 - 0.5 = 0.1$. Check these branch lengths against the values in the original tree: they match.

A and B can now be removed from the leaf set and replaced with E and a new rate adjusted matrix D derived. Completing a further iteration and the final step reconstructs the original tree. □

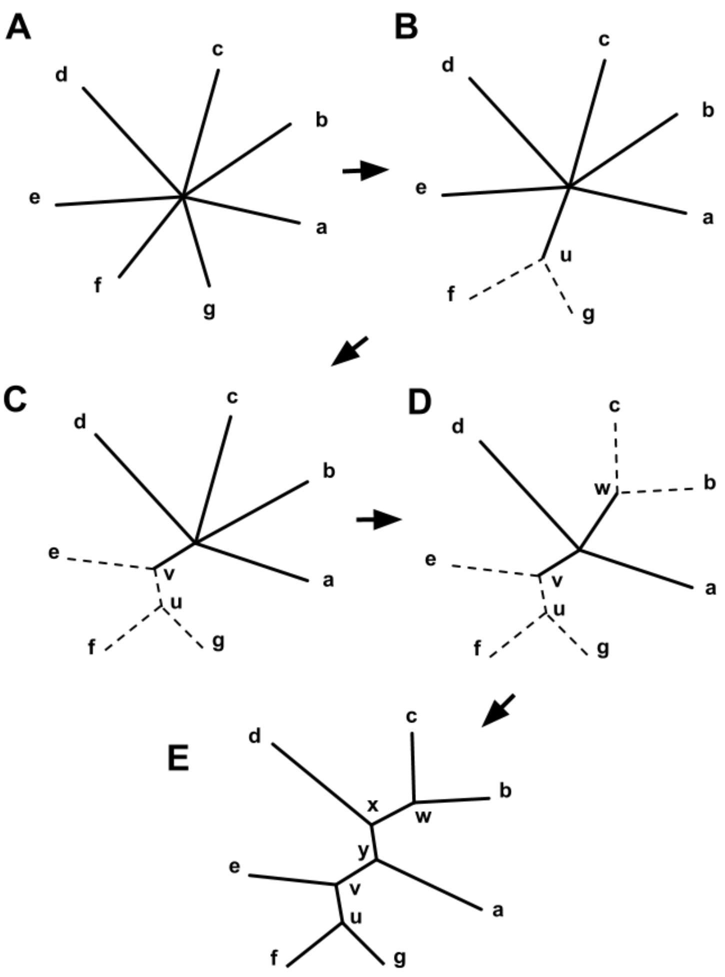


Figure 13: Example from wikipedia http://en.wikipedia.org/wiki/Neighbor_joining: Starting with a star tree in which all leaf nodes are active (A). The matrix D is calculated and used to choose a pair of nodes for joining, in this case f and g. These are joined to a newly created node, u, as shown in (B). The part of the tree shown as dotted lines is now fixed and will not be changed in subsequent joining steps. The distances from node u to the nodes a-e are computed from the formula given in the text. This process is then repeated, using a matrix of just the distances between the nodes, a,b,c,d,e, and u, and a new D matrix derived from it. In this case u and e are joined to the newly created v, as shown in (C). Two more iterations lead first to (D), and then to (E), at which point the algorithm is done, as the tree is fully resolved.

20.4.1 Unrooted vs rooted trees

Notice that the neighbour-joining algorithm produces a tree with no root. That is, we known branch lengths (in terms of distance between sequences — roughly, the number of changes along a branch) but we don't know the actual times of the nodes, so we don't know the position of the root. A tree with no root is an *unrooted tree* and a tree with a known root position is called a *rooted tree.*

In some cases we can determine the position of the root by including a known *out-group* in the analysis. For example, if we have samples from 20 hominids, we could include a chimp as an out-group since we know that the hominids all share a recent common ancestor before the most recent common ancestor of hominids and chimps. In this example, we would place the root on the branch separating the chimp from the hominids.

The number of trees, rooted or unrooted is huge. If we have n taxa, there are

$$\frac{(2n-5)!}{2^{n-3}(n-3)!}$$

unrooted trees and

$$\frac{(2n-3)!}{2^{n-2}(n-2)!}$$

rooted trees. So when $n = 5$, we have 15 unrooted and 105 rooted trees, but for $n = 10$ there are about 2 million unrooted and 3.5 million rooted trees.

20.4.2 Complexity of neighbour jointing and UPGMA

UPGMA has time and space complexity of $O(n^2)$ while neighbour-joining has the same space complexity but time complexity of $O(n^3)$.

However, these are worst case complexities, and there are various heuristics that result in average time performance for neighbour-joining appears somewhat better than $O(n^3)$.