

1 UPGMA

UPGMA is simply hierarchical clustering with average link. We also define the height of a node to be half the distance between two clusters. If the molecular clock assumption is true, i.e. the distance between the root and any leaf is equal, then our distance metric is ultrametric.

A distance measure is ultrametric if and only if for any three points two of the three distances are the same and larger than the third.

2 Neighbor Joining

If we can create an unrooted tree where the distance between two leaves is equal to the path length between them then the lengths are said to be additive. You can readily verify that all ultrametric distance measures are also additive; but the converse is not true. Neighbor joining allows us to recreate any tree that has additive distances and can often do well even if the distances are not additive.

Define $D_{ij} = d_{ij} - (r_i + r_j)$ where $r_i = \frac{1}{|L|-2} \sum_{k \in L} d_{ik}$. Initially, we have all of our leaves unclustered and d gives the pairwise distance between each of them. At each iteration we choose $i, j = \arg \min_{i,j} D_{ij}$. We create a new node k and compute $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$.

We add links from the new node to i and j with weights $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$, $d_{jk} = d_{ij} - d_{ik}$.

Finally, we remove i, j from the nodes we are considering and add k .

When we have only two nodes i, j left we link them with weight d_{ij} .

A distance measure is additive if for every set of four leaves i, j, k, l two of $d_{ij} + d_{kl}$, $d_{ij} + d_{jl}$, $d_{il} + d_{jk}$ must be equal and larger than the third.

Unlike UPGMA neighbor joining gives an unrooted tree. Often it may be useful to obtain a rooted tree. Often this is done by finding an *outgroup* or sample that is distantly related to the others. Where this out group is connected to the rest of the tree is where the root can be placed.

3 Parsimony

Tree building by parsimony is used to discover the tree that can explain the sequences using the fewest number of substitutions. We assume that our sequences have already been aligned and we just want to explain the substitutions.

We will consider the problem of finding the fewest number of substitutions given a full binary tree (each node has either zero or two children). First, we notice that because we do not allow gaps, the number of substitutions for one position can be computed independently of another. So, we are left with single characters at each of the leaves. Let $s(a, b)$ be the substitution cost for characters a and b . We will use the following recurrence to do this:

```

1 # S(k, a) gives the fewest number of substitutions for node k given the
2 # inferred ancestral sequence at k is a
3 S(k, a):
4   if k is a leaf:
5     if x_k = a:
6       return 0
7     else
8       return ∞
9   else
10    Let i and j be the children of k
11    return min_b(S(i, b) + s(a, b)) + min_b(S(j, b) + s(a, b))

```

The optimal assignment has cost $\min_b S(T, b)$. The basic idea is that we want to pick the optimal character at a node so that we reduce the cost of converting the two children to ourself. We need to keep track of the b 's we pick at each node to construct the actual sequences at each location.

Notice, that even though this is written as a recursive algorithm, it should be solved using dynamic programming in a bottom up manner to avoid repeated computations. The location of a root does not matter but it leads the recurrence. Thus, the specific location of the root does not change the actual tree.