Design and Analysis of Algorithms
Lecture 12
Prof. Aaron Clauset

CSCI 5454
2 March 2011

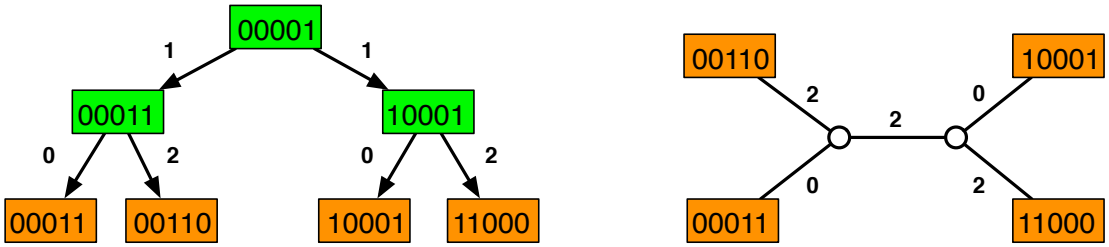
1 More Phylogenetic Trees

Recall that the usual phylogenetic tree inference setup is that we are provided n entities, each of which has a character vector (e.g., a DNA sequence or a vector of morphometric measurements) of length ℓ. Our goal is to infer a phylogenetic tree that accurately captures a plausible evolutionary history of the relatedness of these entities. If we assume only vertical transmission of characters (that is, traits only pass from a parent to its offspring) and that new species are derived from older species, then the evolutionary history has the structure of a tree. For n leaves, there are $(2n-5)!!$ rooted binary trees. The different approaches to choosing these trees are largely differences in the way the quality of candidate trees are scored.

1.1 Unrooted phylogenies

Last time, we met the maximum parsimony principle, which searches this space of trees for the one or set in which the number of character differences between parents and offspring is minimized. We also met distance-based measures, which are fairly similar, but typically operate on continuous-valued characters. Both approaches estimate the ancestral (hidden) character vectors in the phylogeny, and explicitly focus on the evolutionary paths from one state to another.

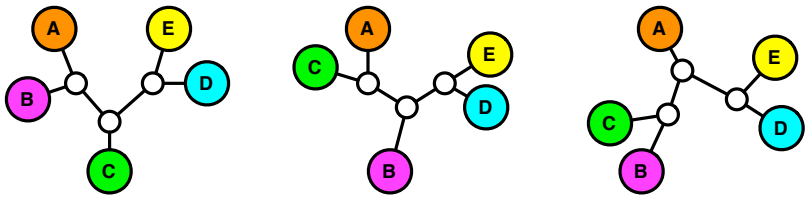
Suppose that we don't particularly care about these ancestral states or the particular evolutionary paths between them. If so, then we only need to consider *unrooted* binary trees. A *rooted* tree has a single tree node with degree 2, which is the root, while all other tree nodes have degree 3, because they have 2 offspring and 1 parent; in an unrooted tree, even tree node as degree 3 and the parent-offspring directionality is omitted. For instance, consider our running example from Lecture 11; the left-hand figure shows the maximum parsimony tree, with ancestral states and the evolutionary distance, while the right-hand figure shows an equivalent unrooted phylogeny:



As before, the weight of the tree is the sum of the edge distances, and we can use various score functions to choose which unrooted tree is a better representation of the evolutionary relationships.

1.2 Consensus trees

What happens when we have multiple plausible (low weight) candidate trees for a set of n species? For instance, here are three trees on 5 vertices:



But which tree is the *right* tree? Or, more generally, which set of evolutionary relationships are common among these three trees? This problem is called the *consensus tree* problem, in which we are given a set of unrooted phylogenetic trees $\{T_i\}$ and our task is to derive from them a single "consensus tree" T_c that includes the common set of evolutionary relationships present in the set. Although the input trees are usually binary trees, the output tree is not necessarily so, i.e., the degree of internal nodes is $k \geq 3$ although leaf nodes are still required to have degree $k = 1$.

In order to build a consensus tree, we need some new formalisms that will help us see how to extract specific evolutionary statements from each tree T_i and then build the consensus tree T_c out of them.

Definition: A *split* on a tree T, denoted $A|B$ is a bipartition of the leaves of $T = (V, E)$ into exactly two non-empty subsets. Further, each edge $(u, v) \in E$ defines such a split: removing (u, v) bipartitions the leaves of T into a set reachable from u through v, and a set reachable from v through u.

Definition: Two splits $A_1|B_1$ and $A_2|B_2$ are *compatible* if at least one of the sets $\{A_1 \cap A_2\}, \{A_1 \cap B_2\}, \{B_1 \cap A_2\}, \{B_1 \cap B_2\}$ is empty.

For the first of the three trees above, its split set is $\{AB, CDE\}, \{ABC, DE\}$ plus the set of trivial partitions $\{X, \{ABCDE\} - X\}$, which separate a single leaf from the rest of the tree. It's straightforward to verify that the set of splits induced by a tree T are pairwise compatible (do you see how?). The converse is also true:

Splits-Equivalence Theorem: Let Ω be a collection of splits. Then, $\Omega \rightarrow T$ for some tree T if and only if the splits in Ω are pairwise compatible. Such a tree is unique up to isomorphism.

1.2.1 Tree re-construction

We'll skip the proof of the splits-equivalence theorem and instead simply discuss the constructive algorithm that makes it work, i.e., the algorithm by which we can take a collection $\Omega = \{\omega_1, ..., \omega_k\}$ of pairwise compatible splits and returns a tree T such that the splits of T are Ω.[1]

The idea is the following. Without loss of generality, we can represent a split as a binary vector s_i of length n where $s_i(j) = 1$ if $v_j \in A$ and $s(j) = 0$ otherwise, and, again w.l.o.g. A always denotes the smaller of the two sets, i.e., $|A| < |B|$. (Do you see why this convention is general?) We then initialize a forest F composed of n singletons, one for each leaf vertex (taxon). We'll then iterate over the set of splits s_i and add a single internal tree node to F that serves as the nearest common ancestor in F of all vertices $s_i(j) = 1$. If there are k non-trivial splits in the set, T must contain $k+1$ internal nodes (do you see why?); thus, after adding one node for each of the k splits, we must add one final internal node that connects the remaining components. What remains is to choose an ordering of the splits that guarantees that we build the tree correctly.

This can be done by noting that if we always add an internal node that connects the two smallest components in F, then we are guaranteed to build the tree from the bottom up. This is equivalent to ordering the splits in increasing size of their smaller set: on the ith pass of the algorithm, we add a node u to F and make it the common ancestor of all the subtrees that are full contained in A_i.

We can make this step easier by using a tricky similar to that of Kruskal's algorithm in Lecture 8, which assigned each component a "leader." In our case, the leader of a component is the nearest common ancestor. For each subtree that will connect to the new node u, we simply look up its leader and connect it to u.[2]

Here's pseudo-code:

```
Build-Consensus-Tree(Q) {
// Q is the set of splits, e.g., k binary vectors of length n
// {A|B} is a split; let A be the smaller set

   sort Q in order of smallest subset A
```

[1] Although we'll only consider unrooted trees here, the split-equivalence theorem and the corresponding algorithms can easily be generalized to rooted trees. The key insight is to recognize that in a rooted tree, each internal node tri-partitions the leaves, and one of those partitions is always "above" that node in the tree; this provides the necessary directionality to choose a root of the tree.

[2] Another split-to-tree building algorithm, called "tree popping," accomplishes the same goal but using a different approach.

```
   for each vertex v in V {
      v[i] = i                  // initialize nearest common ancestor list
   }
   for i = 1 to n+k {           // initialize tree structure
      F[i] = -1
   }
   for i = 1 to k {             // for each split
      {A|B} = split with ith smallest subset A
      // add node n+i to F
      for each v in A {
         F[nca[v]] = n+i       // point  NCA at node n+i
         nca[v]    = n+i       // update NCA of v
      }
   }
   for each v {
      F[nca[v]] = n+k          // point  NCA at node n+k
      nca[v]    = n+i          // update NCA of v
   }
}
```

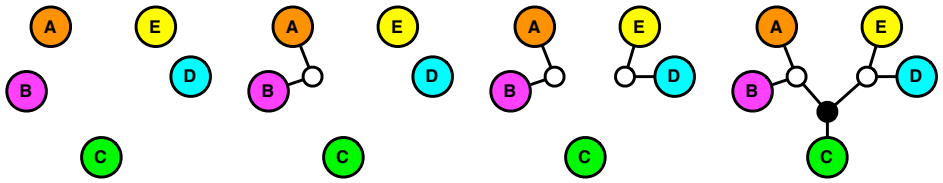
The running time of `Build-Consensus-Tree()` depends partly on the time to sort the splits; since we're representing them as n-bit binary vectors, this can be done in $O(nk \log k)$ (the additional factor of n is the comparison cost for a n-bit binary number). The time to sort the splits dominates the time to initialize the data structures, which takes $O(n + k)$ time. Finally, the main loop takes $O(nk)$ because we make k passes and each pass potentially requires reading the entire length of an n-bit vector. Thus, the running time is dominated by the time to sort the splits. (This algorithm could be more efficient in the way it updates the tree structure, but asymptotically, the duplicated work doesn't matter.)

Let's apply this algorithm to the splits of the first tree in the figure above; recall that the non-trivial splits were $\{AB, CDE\}, \{ABC, DE\}$, which we can rewrite as $\{11000\}, \{00011\}$ following our convention. Here's the reconstruction process, the black internal node is the node added in the last step of the algorithm:

1.2.2 Building the consensus tree

Thus, we can convert any tree T into a set of splits (we haven't described an algorithm for doing so; do you see how to do it? how long does it take?) and any set of splits Ω that satisfies the splits-equivalence theorem can be converted back into a tree.[3] (Note that our function

[3]The splits-equivalence theorem is sometimes also called the "four gametes condition."



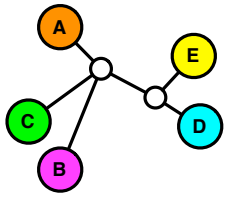
$s_i = \{11000\}$ $s_i = \{00011\}$

`Build-Consensus-Tree()` does not verify that its input satisfies the requirements—what will happen if we pass the function a bad input?) This suggests an algorithm for deriving a consensus tree: convert each tree T_i into its set of splits and combine these into a common pool; then, extract the subset of splits with high frequency and construct a new tree T_c composed only of those splits. What remains however to decide which splits to extract and to guarantee that the set we extract satisfy the splits-equivalence theorem. There are, of course, several approaches, depending on what kind of consensus you want.

The *strict consensus* approach extracts only those splits that occur in every tree T_i while the *majority consensus* approach extracts all splits whose frequency $f_{A|B} \geq 1/2$.[4] Both of these approaches fundamentally construct a histogram on the set of splits, and thus takes time $O(nL \log(nL))$ for L trees on n leaves. If we consider our three trees above, we have a normalized split histogram that looks like this (again, following our labeling convention):

$$\frac{1}{3} \times \{11000\},\ \frac{1}{3} \times \{10100\},\ \frac{1}{3} \times \{01100\},\ \frac{3}{3} \times \{00011\}$$

Thus, the majority consensus tree and the strict consensus tree in this case are the same: only one of the splits appears in at least half of the input trees and it also appears in all of them. Thus, the consensus tree omits any structure on the nodes A, B, C because the input trees disagree on their proper evolutionary relationship.



[4]Naturally, there are additional approaches, e.g., the "semi-strict" approach, which retains all splits from the strict consensus along with any splits that are not incompatible with the strict consensus and each other.

2 For Next Time

1. Optimization algorithms