2
Trees

2.1 Introduction

Mathematically, a tree is a graph (i.e., a pair $G = (V, E)$, where V is the vertex set and E is the edge set) that is connected and acyclic; equivalently, a tree is a graph so that for every pair of vertices v, w in the graph there is a unique path between v and w. When the tree is rooted, we denote its root by $r(T)$. Also, we may denote the set of vertices of T by $V(T)$, the edges by $E(T)$, and the leaves of T by $\mathscr{L}(T)$.

Trees, and especially rooted trees, are used to represent evolutionary histories, and are called "phylogenies," "phylogenetic trees," or "evolutionary trees." Most statistical models of evolution assume that the model tree is a rooted binary tree, so that every node that is not a leaf has exactly two children. However, estimated trees will in general be unrooted and may not be binary. This chapter discusses both rooted and unrooted trees, but we will begin with rooted trees before moving to unrooted trees.

2.2 Rooted Trees

We begin with some basic definitions.

Definition 2.1 In a rooted tree T, we can orient the edges in the direction of the root $r = r(T)$, so that all vertices other than r have outdegree one. Thus, for all nodes $v \neq r$, there is a unique vertex w such that $v \rightarrow w$ is an arc (directed edge) in the tree; w is called the **parent** of v, and v is called the **child** of w. Two or more vertices sharing the same parent are **siblings**. A vertex without any children is called a **leaf** or a **tip**; all other nodes are called **internal nodes**. A vertex with more than two children is a **polytomy**. A rooted tree that has no polytomies is said to be **binary**, **bifurcating**, or **fully resolved**, and is also called a **binary tree**. If a tree is not binary, it is said to be **multifurcating**. The edges of the tree may be referred to as **branches** or even as **inter-nodes**.

If a and b are vertices and there is a directed path from b to a, then a is said to be an **ancestor** of b and conversely b is a **descendent** of a. The most recent common ancestor (**MRCA**) of a set X of leaves in a rooted tree T is the node v that is a common ancestor of all nodes in X, and that is further from the root of T than all other common ancestors.

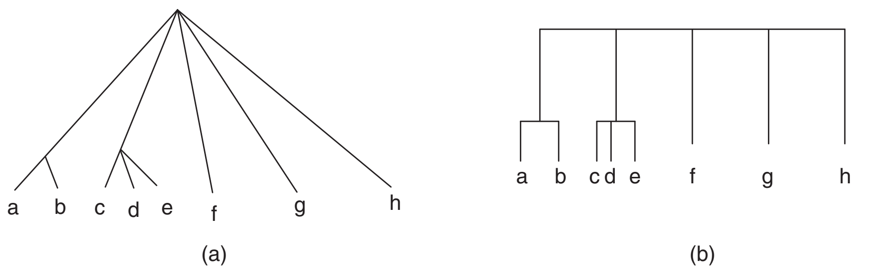


Figure 2.1 Two ways of drawing the same tree

In a phylogenetic tree, the leaves represent the taxa of interest (generally extant species, but sometimes different individuals from the same species) and the internal nodes represent the ancestors of the taxa at the leaves.

A rooted tree can be drawn with the root r on top, on the bottom, on the left, or on the right. However, we'll draw trees with the roots at the top. Graphical representations of trees sometimes include branch lengths, to help suggest relative rates of change and/or actual amounts of elapsed time. The **topology** of the tree is the graphical model without branch lengths.

There are multiple ways to draw the topology of a rooted phylogenetic tree. For example, Figure 2.1 shows two representations of the same evolutionary history. One of these (on the left) is standard in computer science, and the other (on the right) is often found within biological systematics. Note that the root has five children in the tree on the left. Hence, when interpreting a tree in the form given on the right, you should remember that the horizontal lines do not correspond to edges.

2.2.1 Newick Notation for Rooted Trees

Newick notation is the standard way that trees, both rooted and unrooted, are represented in phylogenetic software. The Newick notation for a rooted binary tree with subtrees A and B is given by (A', B'), where A' is the Newick notation for the subtree A and B' is the Newick notation for the subtree B. Since we don't care about the left-to-right ordering of subtrees, it follows that (A', B') is the same thing as (B', A'). Also, the Newick notation for a leaf is the leaf itself.

Some examples of Newick notation should make this process clear. The Newick notation $((a,b),(c,d))$ represents the rooted tree with four leaves, a,b,c,d, with a and b siblings on the left side of the root, and c and d siblings on the right side of the root. The same tree could have been written $((c,d),(a,b))$, or $((b,a),(d,c))$, etc. Thus, the graphical representation is somewhat flexible – swapping sibling nodes (whether leaves or internal vertices in the rooted tree) doesn't change the tree "topology." In fact, there are exactly eight different Newick representations for the rooted tree given in Figure 2.2:

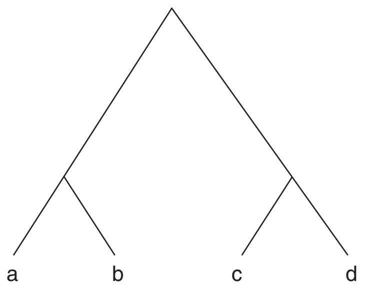


Figure 2.2 Tree $((a,b),(c,d))$

- $((a,b),(c,d))$
- $((b,a),(c,d))$
- $((a,b),(d,c))$
- $((b,a),(d,c))$
- $((c,d),(a,b))$
- $((c,d),(b,a))$
- $((d,c),(a,b))$
- $((d,c),(b,a))$

Similarly, the following Newick strings all refer to the rooted tree in Figure 2.3:

- $((d,e),(c,(a,b)))$
- $((e,d),((a,b),c))$
- $((e,d),(c,(a,b)))$

The second fundamental task is to be able to recognize when two rooted trees are the same when you don't consider branch lengths. For example, the three trees given in Figure 2.3 are different drawings of the same basic tree.

Sometimes the rooted tree you want to represent is not binary. To represent these trees using Newick notation is quite simple. For example, a rooted star tree (i.e., tree without any internal edges) on six leaves a,b,c,d,e,f, is represented by (a,b,c,d,e,f). Similarly, imagine a tree with three children u,v,w off the root, where u has children u_1 and u_2, v has children v_1 and v_2, and w has children w_1 and w_2. A Newick string for the tree is $((u_1,u_2),(v_1,v_2),(w_1,w_2))$. Thus, we can also use Newick strings to represent non-binary trees.

2.2.2 The Clade Representation of a Rooted Tree

Definition 2.2 Let T be a rooted tree in which the leaves are bijectively labeled by a set S; hence, every element of S appears as the label of exactly one leaf. Thus, $\mathscr{L}(T) = S$ and

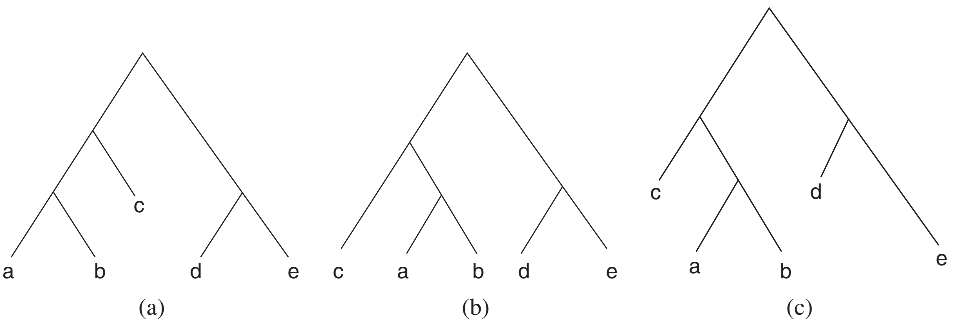


Figure 2.3 Three drawings of the same rooted tree, given by $(((a,b),c),(d,e))$. The trees are considered identical in terms of their topologies (rotations preserve the topology), and branch length does not matter.

is called the **leafset** of T. T_v denotes the subtree of T below node v. The **clades** of T are the subsets of $\mathscr{L}(T)$ that are equal to $\mathscr{L}(T_v)$ for some vertex v.

We now show how to use the clades of a tree to compare it to other trees.

Definition 2.3 Let T be a rooted tree on leafset S. We define the set **Clades(T)** = $\{\mathscr{L}(T_v) : v \in V(T)\}$. Thus, Clades(T) has all the singleton sets (each containing one leaf), a set containing all of S (defined by the root of T), and a clade associated to each vertex of T. The clades that always appear in every possible tree with leafset S are called the **trivial clades**, and all other clades are called the **non-trivial clades**. Thus, all the singleton clades and the set S are trivial clades.

Example 2.4 Consider the rooted tree $T = ((a,b),(c,(d,e))$. The trivial clades are $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, and $\{a,b,c,d,e\}$; these appear in every possible tree on the leafset of T. The non-trivial clades are $\{a,b\}$, $\{c,d,e\}$, and $\{d,e\}$. Hence,

$$Clades(T) = \{\{a\},\{b\},\{c\},\{d\},\{e\},\{a,b\},\{d,e\},\{c,d,e\},\{a,b,c,d,e\}\}.$$

Testing if two rooted trees are identical: Determining if two rooted leaf-labeled trees are the same (with all leaves labeled distinctly) can be difficult if they are drawn differently; but this is easy if you examine the clades! Thus, to determine if two trees T and T' are the same, you can write down the set of clades for the two trees, and see if the sets are identical. If $Clades(T) = Clades(T')$, then $T = T'$; otherwise, $T \neq T'$. For example, if you examine the rooted trees in Figure 2.3, you'll see that they all have the same clade sets. Thus, they are all identical.

2.2.3 Using Hasse Diagrams to Construct Rooted Trees from Sets of Clades

To compute a tree from its set of clades, we will draw a Hasse Diagram based on the binary relation R on the set $Clades(T)$, where $\langle A, B\rangle \in R$ if and only if $A \subseteq B$ (see Section B.1.3). It is not hard to see that R is a partial order, and so the set of clades of a tree, under this relation, is a partially ordered set.

To construct the Hasse Diagram for this partially ordered set, we make a graph with vertex set $Clades(T)$ and a directed edge from a node x to a node y if $x \subset y$. Since containment is transitive, if $x \subset y$ and $y \subset z$, then $x \subset z$. Hence, if we have directed edges from x to y, and from y to z, then we know that $x \subset z$, and so can remove the directed edge from x to z without loss of information. This is the basis of the Hasse Diagram: you take the graphical representation of a partially ordered set, and you remove directed edges that are implied by transitivity. Equivalently, for a given subset x, you find the smallest subsets y such that $x \subset y$, and you put a directed edge from x to y.

As we will see, the Hasse Diagram formed for a set $Clades(T)$ is the tree T itself. You can run the algorithm on an arbitrary set of subsets of a taxon set S, but the output may or may not be a tree.

Example 2.5 Consider

$$A = \{\{a\}, \{a,b,c,d\}, \{a,d,e,f\}, \{a,b,c,d,e,f\}\}.$$

On this input, there are four sets, and so the Hasse Diagram will have four vertices. Let v_1 denote the set $\{a\}$, v_2 denote the set $\{a,b,c,d\}$, v_3 denote the set $\{a,d,e,f\}$, and v_4 denote the set $\{a,b,c,d,e,f\}$. Then, in the Hasse Diagram, we will have the following directed edges: $v_1 \to v_2$, $v_1 \to v_3$, $v_2 \to v_4$, and $v_3 \to v_4$. This is not a tree, since it has a cycle (even though this is only a cycle when considering the graph as an undirected graph).

Theorem 2.6 *Let* T *be a rooted tree in which every internal node has at least two children. Then the Hasse Diagram constructed for* Clades(T) *is identical to* T.

Proof We prove this by strong induction on the number n of leaves in T. For $n = 1$, then T consists of a single node (since every node has at least two children). When we construct the Hasse Diagram for T, we obtain a single node, which is the same as T.

The inductive hypothesis is that the statement is true for all positive n up to $N-1$, for some arbitrary positive integer N. We now consider a tree T with N leaves for which every internal node has at least two children. Since the root of T has at least two children, we denote the subtrees of the root as $t_1, t_2, \ldots, t_k$ (with $k \geq 2$). Note that $Clades(T) = Clades(t_1) \cup Clades(t_2) \cup \ldots \cup Clades(t_k) \cup \mathscr{L}(T)$. The set of vertices for the Hasse Diagram on T contains one vertex for $\mathscr{L}(T)$ and then each of the vertices for the Hasse Diagrams on the trees t_i, $i = 1, 2, \ldots, k$. Also, every directed edge in the Hasse Diagram on T is either a directed edge in the Hasse Diagram on some t_i, or is the directed edge from $\mathscr{L}(t_i)$ to $\mathscr{L}(T)$. By the inductive hypothesis, the Hasse Diagram defined on $Clades(t_i)$ is isomorphic to t_i for $i = 1, 2$, and hence the Hasse Diagram defined on $Clades(T)$ is isomorphic to T. □

2.2.4 Compatible Sets of Clades

Up to now, we have assumed we were given the set $Clades(T)$ for a binary tree T, and we wanted to construct the tree T from that set. Here we consider a related question: Given a set $\mathscr{X}$ of subsets of a set S of taxa, is there a tree T so that $\mathscr{X} \subseteq Clades(T)$?

Definition 2.7 A set $\mathscr{X}$ of sets is said to be **compatible** if and only if there is a rooted tree T with each leaf in T given a different label, so that $\mathscr{X} \subseteq Clades(T)$.

To answer this, see what happens when you construct the Hasse Diagram for the set $\mathscr{X}$.

Example 2.8 We begin with a simple example, $\mathscr{X}_1 = \{\{a,b,c\},\{d,e,f\},\{a,b\}\}$. Note that $\mathscr{X}_1$ contains three subsets and the set S contains six elements. Thus, $\mathscr{X}_1$ does not contain the singleton sets, nor the full set of leaves, and so it is not possible for $\mathscr{X}_1$ to be equal to the set of clades of any tree; and as we observe, the Hasse Diagram we construct is not connected and so is not a tree. Therefore, we add all the trivial clades (the singletons and the full set of leaves) to $\mathscr{X}_1$ and obtain $\mathscr{X}_1'$. We then compute the Hasse Diagram on this set. The result is a tree T, with Newick string $(((a,b),c),(d,e,f))$. This is not a binary tree, but it *is* a tree, and $\mathscr{X}_1 \subset Clades(T)$. However, there are other trees, such as T' denoted by $(((a,b),c),(d,(e,f)))$, that also satisfy $\mathscr{X}_1 \subset Clades(T')$. Note that T can be derived from T' by contracting an edge in T' (i.e., removing an edge and merging the endpoints into a single vertex).

Example 2.9 Consider $\mathscr{X}_2 = \{\{a,b\},\{b,e\},\{c,d\}\}$. Note that the set S contains five elements, but these singleton sets do not appear in $\mathscr{X}_2$. Similarly, the full set S is not in $\mathscr{X}_2$. Hence, $\mathscr{X}_2$ is not the set of clades of any tree. We add all the trivial clades to $\mathscr{X}_2$ to obtain $\mathscr{X}_2'$, and construct the Hasse Diagram for $\mathscr{X}_2'$. Note that $\{b\} \subset \{a,b\}$ and $\{b\} \subset \{b,e\}$. Hence, the Hasse Diagram for $\mathscr{X}_2'$ has a node with outdegree two – which is inconsistent with X_2' being the subset of $Clades(T)$ for any tree T.

2.2.5 Hasse Diagram Algorithm

These two examples suggest an algorithm that you can use to determine if a set of clades is compatible. We call the algorithm the Hasse Diagram algorithm, since it operates by computing a Hasse Diagram, and then checking to see if the Hasse Diagram is a tree.

The input will be a set $\mathscr{X}$ of subsets of a taxon set. The output will either be a rooted tree T for which $\mathscr{X} \subseteq Clades(T)$, establishing that $\mathscr{X}$ is compatible, or *Fail*.

- Step 1: Compute $S = \cup_{X \in \mathscr{X}} X$ (i.e., all the elements that appear in any set in $\mathscr{X}$). Let $S = \{s_1, s_2, \ldots, s_n\}$. Define $\mathscr{X}' = \mathscr{X} \cup S \cup \{s_1\} \cup \{s_2\} \cup \ldots \cup \{s_n\}$; in other words, $\mathscr{X}'$ is the set of subsets of S obtained by adding the full set S and all the singletons to $\mathscr{X}$.
- Step 2: Construct the Hasse Diagram for $\mathscr{X}'$.
- Step 3: If the Hasse Diagram is a tree T, then return T; otherwise return *Fail*.

If the only interest is in determining whether the set of clades is compatible, the following well-known theorem, perhaps originally from Estabrook et al. (1975), is useful:

Lemma 2.10 *A set $\mathscr{X}$ of subsets is compatible if and only if for any two elements* X_1 *and* X_2 *in $\mathscr{X}$, either* X_1 *and* X_2 *are disjoint or one contains the other.*

Proof If a set $\mathscr{X}$ of subsets is compatible, then there is a rooted tree T on leafset S, in which every leaf has a different label, so that each element in $\mathscr{X}$ is the set of leaves below some vertex in T. Let X_1 and X_2 be two elements in $\mathscr{X}$, and let x_1 be the vertex of T associated to X_1 and x_2 be the vertex associated to X_2. If x_1 is an ancestor of x_2, then X_1 contains X_2, and similarly if x_2 is an ancestor of x_1 then X_2 contains X_1. Otherwise neither is an ancestor of the other, and the two sets are disjoint. For the reverse direction, note that when all pairs of elements in set $\mathscr{X}$ satisfy this property, then the Hasse Diagram will be a tree T so that $\mathscr{X} = Clades(T)$. □

The following corollary is very useful in algorithm design, and its proof is left to the reader.

Corollary 2.11 *A set $\mathscr{X}$ of subsets of* S *is compatible if and only if every pair of elements in $\mathscr{X}$ is compatible.*

2.3 Unrooted Trees

According to its mathematical definition, a tree is just a connected acyclic graph; hence, unrooted trees are actually the norm in mathematics. The use of rooted trees is dominant in computer science, however, and also makes sense in phylogenetics. However, the output of phylogenetic software is generally just an unrooted tree. One reason that the output is generally unrooted is that most methods for computing trees are based on time-reversible models of sequence evolution, which means that the root of a tree is not identifiable. However, in practice biologists often use outgroups (i.e., taxa that are clearly not as closely related to the rest of the input taxa as the remaining taxa are to each other) to root trees. Unfortunately, outgroup rooting is also challenging (for reasons that we will discuss later). Hence, typically phylogenetic trees are unrooted trees, and being able to switch between thinking about rooted trees and their unrooted versions is an important skill.

Definition 2.12 The **unrooted version** of a rooted tree T is denoted by T_u, and is formed by suppressing the root and then treating the tree as unrooted. Thus, if the root of T has two children, then these two children are made adjacent to each other, and if the root has more than two children then the graphical model is unchanged.

Consider, for example, the rooted tree T given in Figure 2.4. To turn this into its unrooted version T_u, we would ignore the location of the root, and we'd obtain the unrooted tree given in Figure 2.5. Thus, each rooted tree has a unique unrooted version. However, if we were given the unrooted tree in Figure 2.5, there would be multiple ways of producing rooted

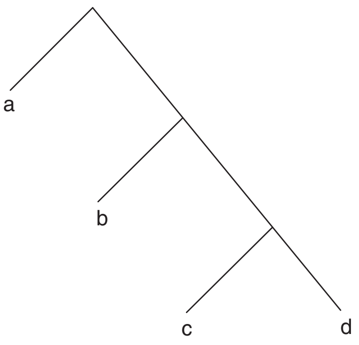


Figure 2.4 Rooted tree $(a,(b,(c,d)))$.

versions. For example, Figures 2.4 and 2.6 are both rooted versions of the same unrooted tree. In fact, you can generate rooted trees from an unrooted tree by picking up the tree at any edge, or at any node. You can even pick up the tree at one of its leaves, but then the tree is rooted at one of its own taxa – which we generally don't do (in that case, we'd root it at the edge leading to that leaf instead, thus keeping the leafset the same).

Definition 2.13 Every node in an unrooted tree is either a **leaf** (in which case it has degree one) or an **internal node**. Two or more leaves with a common neighbor are **siblings**.

The unrooted tree in Figure 2.5 has four leaves, a,b,c,d. We consider a and b to be siblings because they share a common neighbor, and similarly c and d are siblings.

2.3.1 Newick Notation for Unrooted Trees

Notice that the unrooted tree shown in Figure 2.5 was presented with its Newick notation. In fact, to obtain a Newick string for an unrooted tree T, just look at a rooted version of T and write down its Newick string. However, each rooted tree has multiple ways of representing it in Newick format, and every unrooted tree can be rooted in multiple ways. Thus, every unrooted tree will have several Newick representations, each of which is completely valid.

2.3.2 The Bipartitions of an Unrooted Tree

To determine if two unrooted trees are the same, we do something similar to what we did to determine if two rooted trees are the same. The bipartitions of an unrooted tree are formed by taking each edge in turn, and writing down the two sets of leaves that would be formed by deleting that edge. We use $\pi(e)$ to denote the bipartition defined by edge e, with $\pi(e) = A|B$, where A is one half of the bipartition and B is the other. Note that when the edge e is incident to a leaf, then $\pi(e)$ splits the set of leaves into one set with a

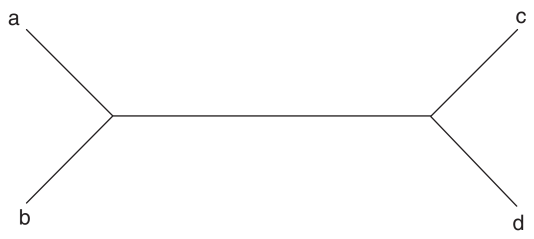


Figure 2.5 Unrooted tree $((a,b),\ (c,d))$

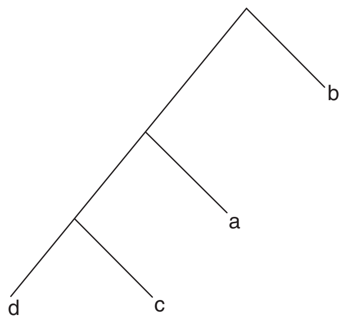


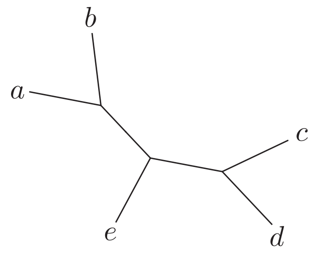
Figure 2.6 Rooted tree $(b,(a,(c,d)))$.

single leaf, and the other set with the remaining leaves. These bipartitions are present in all trees with any given leafset, and hence are called **trivial bipartitions**. Hence, we will focus the discussion just on the **non-trivial bipartitions**. We summarize this discussion with the following definition:

Definition 2.14 Given an unrooted tree T with no nodes of degree two, the **bipartition encoding** of T, denoted by $C(T) = \{\pi(e) : e \in E(T)\}$, is the set of bipartitions defined by each edge in T, where $\pi(e)$ is the bipartition on the leafset of T produced by removing the edge e (but not its endpoints) from T. We also refer to $C(T)$ as the character encoding or split encoding of T; see Figure 2.7 for an example of a tree T and its split encoding $C(T)$. If we restrict this set to the bipartitions formed by the internal edges of the tree T, we obtain $C_I(T)$.

2.3.3 Representing Unresolved Unrooted Trees

Sometimes the unrooted tree we wish to represent is not fully resolved, which means it has nodes of degree greater than three. How do we represent such a tree? For example, consider the tree that has one internal node and four leaves, a,b,c,d. We can represent this



(a) Unrooted tree T

$\{a\}|\{b,c,d,e\}$
$\{b\}|\{a,c,d,e\}$
$\{c\}|\{a,b,d,e\}$
$\{d\}|\{a,b,c,e\}$
$\{e\}|\{a,b,c,d\}$
$\{a,b\}|\{c,d,e\}$
$\{a,b,e\}|\{c,d\}$

(b) Split encoding of T

Figure 2.7 (Figure 5.2 in Huson et al., 2010) We show an unrooted tree T and its set of bipartitions, $C(T)$, referred to here as the split encoding of T. The bottom two splits are the non-trivial splits, and come from the internal edges in T; the top five splits are derived from the edges that are incident with leaves, and are the trivial splits.

simply by (a,b,c,d). Note also that representing it by $(a,(b,c,d))$ yields the same *unrooted* tree. Similarly, what about a tree that has six leaves, a,b,c,d,e,f, and one internal edge that separates a,b, from c,d,e,f? We can represent this unrooted tree by $(a,b,(c,d,e,f))$, or any of the alternatives that also yield one single bipartition separating a,b from the remaining leaves.

Sometimes, if the tree has only a single bipartition, we will simplify our representation by just giving the bipartition; i.e., we represent the tree with one edge separating a,b from c,d,e,f by $\{a,b\}|\{c,d,e,f\}$, or more simply by $ab|cdef$. In other words, the representations for trees that appear in the mathematical literature are quite flexible. (Of course, representations of trees in software must be done very precisely, using the requirements for the software ... but that is another matter.)

2.3.4 Comparing Unrooted Trees Using Their Bipartitions

It is easy to see that we can write down the set of bipartitions of any given unrooted tree, and that two unrooted trees are identical if they have the same set of bipartitions. An important additional concept is **refinement**, which we now describe.

Definition 2.15 Given two trees T and T' on the same set of leaves (and each leaf given a different label), tree T is said to **refine** T' if T' can be obtained from T by contracting a set of edges in T. We also express this by saying T **is a refinement of** T' and T' **is a contraction of** T. In fact, T refines T' if and only if $C(T') \subseteq C(T)$.

Note that each tree refines itself and is also a contraction of itself (since we can choose to contract an empty set of edges).

Definition 2.16 An unrooted tree T is **fully resolved** if there is no tree $T' \neq T$ that refines T. Equivalently, T is fully resolved if all the nodes in T have degree one or three.

An unrooted tree that is fully resolved is also called a **binary tree**. (Note, however, that we also referred to rooted binary trees, so that "binary tree" has a slightly different meaning for rooted and unrooted trees.)

2.3.5 Constructing Unrooted Trees from Their Bipartitions

Sometimes we are given a set A of bipartitions, and we are asked whether these bipartitions could co-exist within a tree (i.e., whether there exists a tree T so that $A \subseteq C(T)$). When this is true, the set of bipartitions is said to be *compatible*, and otherwise the set is said to be *incompatible*.

Definition 2.17 A set A of bipartitions on the set S is **compatible** if there exists an unrooted tree T in which every leaf has a distinct label from a set S, so that $A \subseteq C(T)$.

To construct a tree from a compatible set A of bipartitions, we will use A to construct a set C of clades that will be compatible if and only if the set A is compatible. We will then run the Hasse Diagram Algorithm from Section 2.2.5 on the set C. If C is a compatible set of clades, this will return a rooted tree T realizing the set C. Then, to construct the unrooted tree for A, we will return T_u, the unrooted version of T (see Definition 2.12). T_u is then the **canonical tree** for the set A.

To complete this description, we only need to say how we compute the set C of clades given A. First, we add all of the missing trivial bipartitions (the ones of the form $x|S \setminus \{x\}$) to A. Then, pick any leaf (call it "r") in the set to function as a root. This has the result of turning the unrooted tree into a rooted tree, and therefore turns the bipartitions into clades! In other words, for each bipartition $A|B$, we write down the subset that does not contain r, and denote it as a clade. We also include S (the full set of taxa) and $\{x\}$ for each $x \in S$ (the singleton sets). This is set C of clades we obtain from the set A of bipartitions.

Example 2.18 We will determine if the set A of bipartitions given by

$$A = \{(123|456789), (12345|6789), (12|3456789), (89|1234567)\}$$

is compatible, and if so we will construct its canonical tree. First, we decide to root the tree at leaf 1. We look at each bipartition, and select the half of the bipartition that does not contain 1. Thus, we obtain the following set of clades:

$$\{\{4,5,6,7,8,9\}, \{6,7,8,9\}, \{3,4,5,6,7,8,9\}, \{8,9\}\}$$

We then add the full set S and all the singleton sets, and construct a Hasse Diagram for this set of sets. Note that every non-root node in the Hasse Diagram for this set of sets has outdegree 1, and hence defines a rooted tree given by $(1,(2,(3,(4,5,((6,7),(8,9))))))$. Although we treat 1 as a root in order to form clades, this technique produces a tree in which 1 is a leaf and not the root. We then unroot this tree to obtain the tree given in Figure 2.8.

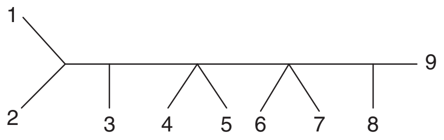


Figure 2.8 Unrooted tree on $\{1\ldots 9\}$, obtained by running the Hasse Diagram algorithm on the set $A = \{(123|456789), (12345|6789), (12|3456789), (89|1234567)\}$; see Example 2.18.

2.3.6 Testing Compatibility of a Set of Bipartitions

What we have described is an algorithm that will construct a tree from a set of compatible bipartitions. With a simple modification, we can use the algorithm to test if a set of bipartitions is compatible: When we construct the Hasse Diagram, we check that it creates a tree. If it does, then we return "Compatible" and otherwise we return "Not Compatible." It is easy to verify that this method returns the correct answer when the set is compatible. What about when the set is not compatible? We demonstrate this with an example.

Example 2.19 Suppose the set of bipartitions has two bipartitions $ab|cd$ and $ac|bd$. We root the bipartitions at leaf a, and obtain the non-trivial clades $\{c,d\}, \{b,d\}, \{b,c,d\}$. We add $\{a,b,c,d\}$ and the singleton sets. When we compute the Hasse Diagram, we note that the graph has a cycle (as an undirected graph) on the vertices for clades $\{d\}, \{c,d\}, \{b,d\}$, and $\{b,c,d\}$. Hence, the Hasse Diagram is not a tree, and the algorithm returns "Not Compatible."

Pairwise compatibility ensures setwise compatibility: Just as we saw with testing compatibility for clades, it turns out that bipartition compatibility has a simple characterization, and pairwise compatibility ensures setwise compatibility.

Theorem 2.20 *A set* A *of bipartitions on a set* S *is compatible if and only if every pair of bipartitions is compatible. Furthermore, a pair of bipartitions* $\mathrm{X}_1|\mathrm{X}_2$ *and* $\mathrm{Y}_1|\mathrm{Y}_2$ *of bipartitions is compatible if and only if at least one of the four pairwise intersections* $\mathrm{X_i} \cap \mathrm{Y_j}$ *is empty.*

Proof We begin by proving that a pair of bipartitions is compatible if and only if at least one of the four pairwise intersections is empty. It is easy to see that a pair of bipartitions is compatible if and only if the clades produced (for any way of selecting the root) are compatible. So let's assume that we set some arbitrary element s of S to be the root, and that $s \in X_1 \cap Y_1$. Therefore, X and Y are compatible as bipartitions if and only if X_2 and Y_2 are compatible as clades. Therefore, X and Y are compatible as bipartitions if and only if one of the following statements holds:

- $X_2 \subseteq Y_2$
- $Y_2 \subseteq X_2$
- $X_2 \cap Y_2 = \emptyset$

If the first condition holds, then $X_2 \cap Y_1 = \emptyset$, and at least one of the four pairwise intersections is empty. Similarly, if the second condition holds, then $Y_2 \cap X_1 = \emptyset$, and at least one of the four pairwise intersections is empty. If the third condition holds, then directly at least one of the four pairwise intersections is empty. Thus, if X and Y are compatible as bipartitions, then at least one of the four pairwise intersections is empty.

For the converse, suppose that X and Y are bipartitions on S, and at least one of the four pairwise intersections is empty; we will show that X and Y are compatible as bipartitions. First, not all four pairwise intersections can be empty. For example, suppose $X_1 \cap Y_1 \neq \emptyset$ and let $s \in X_1 \cap Y_1$. To show that X and Y are compatible as bipartitions it will suffice to show that X_2 and Y_2 are compatible as clades. Since $X_1 \cap Y_1 \neq \emptyset$, the pair that produced the empty intersection must be one of the other pairs: i.e., one of the following must be true: $X_1 \cap Y_2 = \emptyset, X_2 \cap Y_2 = \emptyset$, or $X_2 \cap Y_1 = \emptyset$. If $X_1 \cap Y_2 = \emptyset$, then $Y_2 \subseteq X_2$, and X_2 and Y_2 are compatible clades; thus, X and Y are compatible bipartitions. If $X_2 \cap Y_1 = \emptyset$, then a similar analysis shows that $X_2 \subseteq Y_2$, and so X and Y are compatible bipartitions. Finally, if $X_2 \cap Y_2 = \emptyset$, then directly X_2 and Y_2 are compatible clades, and so X and Y are compatible bipartitions.

Now that we have established that two bipartitions are compatible if and only if at least one of the four pairwise intersections is empty, we show that a set of bipartitions is compatible if and only if every pair of bipartitions is compatible. So let $s \in S$ be selected arbitrarily as the root, and consider all the clades (halves of bipartitions) that do not contain s. This set of subsets of S is compatible if and only if every pair of subsets is compatible, by Lemma 2.10. Hence, the theorem is proven. □

Hence, if all we want to do is determine if a set of bipartitions is compatible but we don't need to construct the tree demonstrating the compatibility, we can just check that every pair of bipartitions is compatible. If all pairs are compatible, then we return "Compatible," and otherwise we return "Not Compatible."

2.4 Constructing the Strict Consensus Tree

A common event in phylogeny estimation is that a set of trees is computed for a given dataset, and we will wish to compute a consensus of these trees.

Definition 2.21 The **strict consensus tree** of a set $\mathscr{T} = \{T_1, T_2, \ldots, T_k\}$ of trees is the most resolved common contraction of the trees in $\mathscr{T}$. Hence, T is the strict consensus of $\mathscr{T}$ if and only if every tree $T_i \in \mathscr{T}$ refines T, and every other tree satisfying this property is a refinement of T (see Definition 2.15). Furthermore, the strict consensus tree T satisfies $C(T) = \cap_i C(T_i)$.

2.5 Quantifying Error in Estimated Trees

Since the true tree is unknown, determining how close an estimated tree is to the true tree is typically difficult. However, for the purposes of this section, we will presume that the true tree is known (perhaps because we have performed a simulation), so that we can compare estimated trees to the true tree. Furthermore, since estimated trees are generally unrooted, we will compare the unrooted estimated tree to the unrooted version of the true tree.

Let us presume that the tree T_0 on leafset S is the true tree, and that another tree T is an estimated tree for the same leafset. There are several techniques that have been used to quantify errors in T with respect to T_0, of which the dominant ones are these:

False negatives (FN): The **false negatives** are those edges in T_0 inducing bipartitions that do not appear in $C(T)$; these are also called "missing branches." The **false negative rate** is the fraction of the total number of non-trivial bipartitions that are missing, or $\frac{|C(T_0)\setminus C(T)|}{|C(T_0)|}$.

False positives (FP): The **false positives** in a tree T with respect to the tree T_0 are those edges in T that induce bipartitions that do not appear in $C(T_0)$. The **false positive rate** is the fraction of the total number of non-trivial bipartitions in T that are false positives, or $\frac{|C(T)\setminus C(T_0)|}{|C(T)|}$.

Robinson–Foulds (RF): The most typically used error metric is the sum of the number of FPs and FNs, and is called the **Robinson–Foulds distance** or the **bipartition distance**. The RF rate is obtained by dividing the number of FNs and FPs by $2n-6$ where n is the number of leaves in each tree.

When both the estimated and true trees are binary, then FN and FP rates are equal, and these also equal the RF distance. The main advantage in splitting the error rate into two parts (FN and FP) is that many estimated trees are not binary. In this case, when the true tree is presumed to be binary, the FP error rate will be less than the FN error rate. Note also that the reverse can happen – the FN error rate could be smaller than the FP error rate – when the true tree is not binary. Also note that the RF error rate is not necessarily equal to the average of the FN and FP error rates. Finally, the RF rate of a **star tree** (i.e., a tree with no internal edges, see Definition 2.25) with respect to a binary true tree is 50 percent, which is the same as the RF rate for a completely resolved tree that has half of its edges correct. Using the RF rate has been criticized because of this phenomenon, since it tends to favor unresolved trees (Rannala et al., 1998). Therefore, when estimated trees may not be binary, it makes sense to report both FN and FP rates instead of just the RF rate.

Observation 2.22 Let T be the true tree, and T_1 and T_2 be two estimated trees for the same leafset. If T_1 is a refinement of T_2, then the number of FNs of T_1 will be less than or equal to that of T_2, and the number of FPs of T_1 will be at least that of T_2.

This observation will turn out to be important in understanding the relationship between the error rates of consensus trees (described in Chapter 6) and how they compare to the error rates of the trees on which they are based.

2.6 The Number of Binary Trees on n Leaves

Since we are interested in estimating phylogenetic trees, knowing the number of possible leaf-labeled binary trees (rooted or unrooted), where each leaf has a different label drawn from $\{1,2,\ldots,n\}$, is relevant to understanding the computational challenges in exploring "treespace."

We first consider the unrooted case. For $n = 1,2$, or 3, the answer is 1: there is only one unrooted binary tree when $n \leq 3$. However, when $n = 4$, there are three possible trees. Furthermore, it is easy to see this algorithmically: to construct a tree on $n = 4$ leaves, s_1, s_2, s_3, and s_4, take a tree T on $n = 3$ leaves, and then add the remaining leaf by subdividing an edge in the tree T, and making s_4 adjacent to this newly introduced node. Thus, the number of possible trees on $n = 4$ leaves is equal to the number of edges in T. Since T has three leaves, it has exactly three edges (you can see this by drawing it). Hence, there are three unrooted binary trees on four leaves.

Things become a bit more difficult for larger values of n, but the same algorithmic analysis applies. Take a tree T on $n-1$ leaves, pick an edge in T and subdivide it, and make s_n adjacent to the newly created node. The number of unrooted binary trees on n leaves is therefore equal to the product of the number t_{n-1} of unrooted binary trees on $n-1$ leaves and the number e_{n-1} of edges in an unrooted binary tree on $n-1$ leaves. It is not hard to see that $e_{n-1} = 2(n-1)-3 = 2n-5$. Hence, t_n satisfies $t_n = t_{n-1}(2n-5)$, and $t_3 = 1$. Thus, for $n \geq 3$, $t_n = (2n-5)!! = (2n-5)(2n-7)\ldots 3$.

Now, we examine the number of different rooted binary trees on n leaves. To compute this, note that every rooted binary tree T on n leaves defines an unrooted binary tree T_u (obtained by ignoring the root of T), and that every unrooted binary tree T_u corresponds to $2n-3$ rooted binary trees formed by rooting the tree T_u on one of its edges. Hence, the number of rooted binary trees on n leaves is $(2n-3)!! = (2n-3)(2n-5)\ldots 3$.

2.7 Rogue Taxa

Sometimes a phylogenetic analysis of a collection of taxa will produce many trees with nearly identical scores (for whatever optimization problem is used) that differ primarily in terms of where a particular taxon is placed. Such a taxon is called a "rogue taxon" in the biological literature (Sanderson and Shaffer, 2002). Because the inclusion of rogue taxa in a phylogenetic analysis can increase the error of the phylogenetic analysis, they are often removed from the dataset before the final tree is reported.

Causes for rogue taxa vary, but a common cause is having a distantly related outgroup taxon in the dataset (the subject of the next section). The sequences for such taxa can be extremely different from all other sequences in the dataset, so that there is close to no similarity beyond what two random sequences would have to each other. In the extreme case of using a random sequence, the taxon with the random sequence could fit equally well into any location of the tree, and hence its location cannot be inferred with any reliability. When a phylogenetic analysis explores multiple optimal or near-optimal trees for the dataset, this will mean that the profile (set of trees) computed for the dataset will include trees that differ substantially in the placement of the rogue taxon. The strict consensus tree of any such set of trees will be highly unresolved, and may even be a star tree (see Definition 2.25).

2.8 Difficulties in Rooting Trees

Although evolutionary trees are rooted, estimations of evolutionary trees are almost always unrooted, for a variety of reasons. In particular, unless the evolutionary process obeys the strict molecular clock, so that the expected number of changes is proportional to the time elapsed since a common ancestor, rooting trees requires additional information. The typical technique is to use an "outgroup" (a taxon which is not as closely related to the remaining taxa as they are to each other). The outgroup taxon is added to the set of taxa and an unrooted tree is estimated on the enlarged set. This unrooted tree is then rooted by "picking up" the unrooted tree at the outgroup. See Figure 2.9, where we added a fly to a group of mammals. If you root the tree on the edge incident with fly, you obtain the rooted tree $(\mathit{fly}, (\mathit{cow}, (\mathit{chimp}, \mathit{human})))$, showing that chimp and human have a more recent common ancestor than cow has to either human or chimp.

The problem with this technique is subtle: While it is generally easy to pick outgroups, the less closely related they are to the remaining taxa, the less accurately they are placed in the tree. That is, very distantly related taxa tend to fit equally well into many places in the tree, and thus produce incorrect rootings. See Figure 2.10, where the outgroup attaches to two different places within the tree on the remaining taxa.

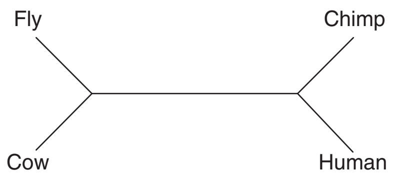


Figure 2.9 Tree on some mammals with fly as the outgroup.

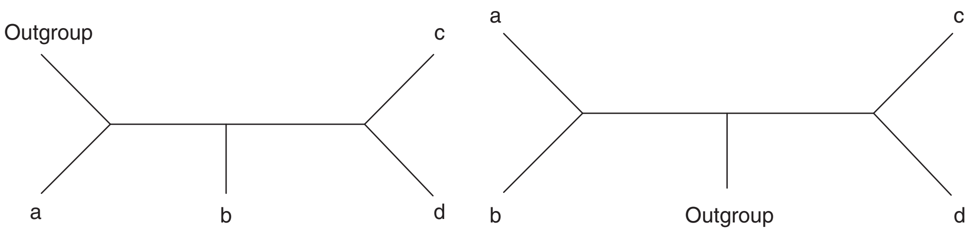


Figure 2.10 Two unrooted trees that differ only in the placement of the outgroup. If these trees were rooted at the outgroup, they would produce different rooted trees on the ingroup taxa a, b, c, d.

Furthermore, it is often difficult to distinguish between an outgroup taxon that is closely related to the ingroup taxa, and a taxon that is, in fact, a member of the same group that branched off early in the group's history. When this happens, even if the unrooted tree is correct, the rooted version of the unrooted tree will be wrong. For this reason, even the use of outgroups is somewhat difficult.

2.9 Homeomorphic Subtrees

Many of the approaches to constructing large trees operate by computing smaller trees and then combining these smaller trees. To understand these techniques, we need to understand the notion of "homeomorphic subtree," which we now define.

Suppose we are given a set X of taxa and an unrooted tree T with leafset S, and suppose $X \subset S$. Now, consider the tree that is obtained by removing from T every node and edge that is not on a path between two leaves in X; this produces a tree that will have at least one node (and possibly many nodes) of degree two. Now, we modify T further so that it has no nodes of degree two, as follows. If T has any nodes of degree two, then it has at least one path $P = v_1, v_2, \ldots, v_t$ such that all of its internal nodes (i.e., v_i for $i = 2, 3, \ldots, t-1$) have degree two, and its endpoints have degree greater than two. Find any such path P, and replace P by the single edge (v_1, v_t). Repeat this process until there are no degree-two nodes. We will denote this final tree by $T|X$, and refer to the final tree produced by this process as the **homeomorphic subtree of T induced by** X (or sometimes more simply just as "induced subtree").

When the tree T is rooted, the process for computing the homeomorphic subtree is nearly the same, except that the path P can never have the root as an internal node, and the root (which may well have degree two) is allowed to be an endpoint of P. Hence, the result of suppressing nodes of degree two maintains the location of the root.

Homeomorphic subtrees are often used when you have a tree T (rooted or unrooted) but you are only interested in what the tree says about a particular subset of its leafset. For example, T could have leafset $\{a,b,c,d,e,f,g,h,i\}$, but you are only interested in the evolutionary history of a, b, c, and d. To understand what T tells you about just their evolutionary history, you should construct the homeomorphic subtree induced by T on $\{a,b,c,d\}$. An example of this is shown in Figure 1.5.

Homeomorphic subtrees are also relevant when we try to construct large trees using divide-and-conquer, so that smaller trees are computed and then combined into larger trees. In that case, the hope is that the smaller trees that have been computed are all "correct" in the sense that they would all be homeomorphic subtrees of a common larger tree.

2.10 Some Special Trees

Some types of trees are used frequently as examples to illustrate different properties of algorithms.

Definition 2.23 The **caterpillar** tree on n leaves $s_1, s_2, \ldots, s_n$ is $(s_1,(s_2,(s_3,(s_4,\ldots))))$. The caterpillar tree is also referred to as the **comb** tree, and the tree is described as being **pectinate** or **ladder-like**. Note that the maximum distance (in terms of the number of edges) between any two leaves in a caterpillar tree on n leaves is $n-1$, and that the maximum pairwise distance in any other tree on n leaves is smaller.

Definition 2.24 The **completely balanced** tree on 2^n leaves is a rooted tree, where the root is the parent of the roots of two completely balanced trees on 2^{n-1} leaves. Note that in the completely balanced tree on 2^n leaves, the maximum distance between any two leaves is only $2n$, and that any other binary tree on 2^n leaves will have a larger maximum pairwise distance.

Definition 2.25 The **star tree** on n leaves has one internal node that is adjacent to all its leaves. Thus, the star tree has no internal edges.

2.11 Further Reading

In this chapter we presented three ways (false positives, false negatives, and Robinson–Foulds) to quantify error rates in phylogeny estimation methods. While these techniques (and especially the RF error metric) remain the most commonly used, they have some limitations, and other methods have been proposed in order to address these limitations.

One of the limitations of these criteria is that they are vulnerable to rogue taxa. For example, consider the following pair of caterpillar trees:

$$T = (s_1,(s_2,(s_3,(\ldots(s_{n-1},s_n)\ldots)$$
$$T' = ((s_n,s_1),(s_2,(s_3,(\ldots(s_{n-2},s_{n-1})\ldots)$$

Thus, T' is obtained by taking s_n and moving it to the other end of T. By design, T and T' share no common bipartitions, and hence have the largest possible FP, FN, and RF distances. Methods for quantifying distances between trees that are less affected by rogue taxa include the quartet distance (i.e., the number of quartets of taxa that the two trees resolve differently), the size of the maximum agreement subtree (i.e., the largest number of taxa on which the two trees have the same homeomorphic subtree see Chapter 6), and a metric proposed by Lin et al. (2012a).

Another weakness of these three metrics is that they do not take branch length into account; this was rectified by Kuhner and Felsenstein (1994), who presented the "branch score," which is the sum of the squares of the differences in branch lengths, where an edge has branch length 0 if it does not appear in the tree. The branch score metric can also be used to take branch support (see Section 1.7) into account, by treating a missing branch as having no support and hence zero branch length, and a branch that is present as having the length equal to its support (however that support is calculated). When one of the trees is the model tree, then support values would be 1 for the edges that are present and otherwise 0.

2.12 Review Questions

1. Is the Newick representation of a rooted tree unique, or can there be multiple Newick representations of any given tree? What about unrooted trees?
2. Let T be a rooted binary tree on ten leaves. How many clades does T have, including the singleton clades and the full set of taxa?
3. Let T_u be an unrooted binary tree on ten leaves; how many bipartitions does T_u have?
4. What is the running time to compute a rooted binary tree from its set of clades?
5. What is the largest possible Robinson–Foulds distance between two unrooted binary trees on the same set of n leaves?
6. What is the number of rooted binary trees on ten leaves?
7. What is the number of unrooted binary trees on ten leaves?
8. What is the definition of the strict consensus tree of a set of trees?

2.13 Homework Problems

1. Draw the rooted tree that is given by $(f,((a,b),(c,(d,e))))$.
2. Draw a rooted tree and give its Newick format representation.
3. Draw the rooted tree given by $(1,(2,(3,(4,(5,6)))))$, and write down the set of clades of that tree.
4. Draw the same rooted tree using the different styles as described in the text.
5. For the rooted tree T given by $(a,((b,c),(d,(e,f))))$,
   - write down at least three other Newick representations;
   - write down the set of clades, and indicate which of the clades is non-trivial.
6. Compute the Hasse Diagram on the partially ordered sets given by the following sets of clades, and then draw the rooted tree for each set.
   - $\{\{a,b\},\{a,b,c\},\{a,b,c,d\},\{e,f\},\{e,f,g\}\}$
   - $\{\{a,b,c\},\{a,b,c,d\},\{e,f\},\{e,f,g\}\}$

   Which one of these trees is *not binary*?
7. Draw all rooted binary trees on leafset $\{a,b,c,d\}$. (Note that trees that can be obtained by swapping siblings are the same.)
8. Draw all rooted trees (not necessarily binary) on leafset $\{a,b,c,d\}$.
9. Give a polynomial time algorithm to determine if two Newick strings represent the same rooted tree. For example,
   - your algorithm should return “YES” on $(a,(b,c))$ and $((c,b),a)$; and
   - should return “NO” on $(a,(b,c))$ and $(b,(a,c))$.
10. Draw the rooted and unrooted versions of the unrooted tree given by the following Newick string: $((a,b),(c,(d,e))$.
11. Draw all the rooted versions of the unrooted tree $(x,(y,(z,w)))$, and give their Newick formats.

12. Draw the unrooted version of the trees given below, and write down the set $C(T)$ of each tree T below. Are the two trees the same as unrooted trees?
    1. $(a,(b,(c,((d,e),(f,g)))))$
    2. $(((a,b),c),((d,e),(f,g)))$
13. Consider the two unrooted trees given below by their bipartition encodings. Draw them. Do you see how one tree can be derived from the other by contracting a single edge? Which one refines the other?
    - T_1 is given by $C(T_1) = \{(ab|cdef),(abcd|ef)\}$.
    - T_2 is given by $C(T_2) = \{(ab|cdef)\}$.
14. Draw two unrooted trees, so that neither can be derived from the other by contracting a set of edges.
15. Draw three different unrooted trees, T_1, T_2, and T_3, on no more than eight leaves, so that T_1 is a contraction of T_2, and T_2 is a contraction of T_3 (identically, T_3 is a refinement of T_2, and T_2 is a refinement of T_1). Write down the bipartition encodings of each tree.
16. Apply the technique for computing unrooted trees from compatible bipartitions to the input given below, using leaf 3 as the root. After you are done, do it again but use a different leaf as the root. Compare the rooted trees you obtained using the different leaves as roots: are they different? Unroot the trees, and compare the two unrooted trees. Are they the same?
    Input: $\{(123|456789),(12345|6789),(12|3456789),(89|1234567)\}$
17. Compute the unrooted trees compatible with the following sets of bipartitions (use the algorithm that operates on clades, using the specified roots):
    - $\{(ab|cdef),(abc|def),(abcd|ef)\}$, with root "b." Then do this again using root c. Are the unrooted trees you get different or the same?
    - $\{(ab|cdef),(abc|def),(abcd|ef)\}$, with root "d."
    - $\{(abcdef|ghij),(abc|defghij),(abcdefg|hij)\}$, using any root you wish.
18. Give a polynomial time algorithm to determine if the unrooted trees defined by two Newick strings are the same.
    - Your algorithm should return "YES" for $(a,(b,(c,d)))$ and $(c,(d,(b,a)))$; and
    - should return "NO" for $(a,(b,(c,d)))$ and $((b,d),(a,c))$.
19. Consider the unrooted tree given by $(1,((2,3),(4,(8,9)),(5,(6,7))))$. Root this tree at leaf 5, draw this rooted tree, and write the Newick string for the rooted tree you obtain.
20. Draw two binary unrooted trees on leafset $\{a,b,c,d,e,f\}$ that induce the same homeomorphic subtree on $\{a,b,c,d,e\}$ but have no non-trivial bipartitions in common.
21. Suppose T_0 is the true tree and T is the estimated tree. Which of the following statements are not possible, under the assumption that both T_0 and T are unrooted trees on ten leaves, and that T_0 is binary and T may not be binary? If you think the statement is impossible, explain why. Else, give an example where it is true.
    - There are five false negatives and three false positives.

- There are three false negatives and five false positives.
- There are three false negatives and three false positives.
- There are eight false negatives and two false positives.
- There are eight false negatives and eight false positives.
- There are seven false negatives and one false positive.
- There are one false negative and seven false positives.

22. Answer the same questions as for the previous problem, but do not assume now that the true tree T_0 is binary, but do require that T is binary.
23. Let T_0 be the unrooted true tree (and hence, binary). Let T_1 and T_2 be estimated unrooted trees on the same leafset as T_0, where T_1 is a star tree (see Definition 2.25) and T_2 is fully resolved (binary).
    1. What is the Robinson–Foulds (RF) rate of T_1 with respect to the true tree?
    2. For what trees T_2 will T_1 have a better RF rate than T_2?
    3. What do you think of using the RF rate as a way of comparing trees? What alternatives would you give?
24. Let T_0 be the unrooted tree given by splits $\{123|456, 12|3456, 1234|56\}$, and let T_1 be an estimated tree. Suppose T_1 is missing split $123|456$, but has a single false positive $124|356$. Draw T_1.
25. Give an algorithm for the following problem:

    Input: unrooted tree T_0 and two sets of bipartitions, C_1 and C_2, where $C_1 \subseteq C(T_0)$ and $C_2 \cap C(T_0) = \emptyset$.

    Output: tree T_1 (if it exists) such that T_1 has false negative set C_1 and false positive set C_2, when T_0 is treated as the true tree. (Equivalently, $C(T_1) = [C(T_0) - C_1] \cup C_2$.)
26. Let T be a caterpillar tree on n leaves (i.e., $T = (s_1, (s_2, (s_3, \ldots, (s_{n-1}, s_n)) \ldots))))$ (see Definition 2.23). Now let $\mathscr{T}$ be the set of trees on $n+1$ leaves formed by adding a new taxon, s_{n+1}, into T in all the possible ways. What is the expected RF distance between two trees picked at random from $\mathscr{T}$?
27. Prove using induction that the number of edges in an unrooted binary tree on $n \geq 2$ distinctly labeled leaves is $2n-3$.
28. Consider the set $\mathscr{T}_n$ of unrooted binary trees on leafset $S = \{s_1, s_2, \ldots, s_n\}$. If you pick a tree uniformly at random from $\mathscr{T}_n$, what is the probability that s_1 and s_2 are siblings in $\mathscr{T}$?
29. Consider a caterpillar tree T on a set S of n taxa. Suppose there is a very rogue taxon, x, which can be placed into T in any possible position. Consider the set $\mathscr{T}$ that contains all the trees formed by adding x into T.
    1. What is $|\mathscr{T}|$?
    2. What is the strict consensus of all the trees in $\mathscr{T}$? (Give its bipartition set.)
30. For each of the given unrooted trees, draw the homeomorphic subtree induced on $\{a, b, c, d\}$.
    - T_1 has Newick format $(b, (a, (f, (c, (g, (d, e))))))$.

- T_2 has Newick format $(f,(a,(c,(g,(d,(b,e))))))$.

31. a. Give two unrooted trees on $\{a,b,c,d,e,f,g\}$ that induce the same homeomorphic subtree on $\{a,b,c,d\}$ but which are different trees.
    b. Give two unrooted trees on $\{a,b,c,d,e,f,g\}$ that are identical on $\{a,b,c,d\}$ and different on $\{d,e,f,g\}$.
    c. Give two rooted trees on $\{a,b,c,d,e\}$ which are identical on $\{a,b,c\}$ but different on $\{d,e,f\}$.
    d. Let T and T' be two different binary trees on the same leafset S. Suppose T^* is a binary tree on leafset S with $C(T^*) \subset C(T) \cup C(T')$. Must T^* be one of T_1 or T_2? If so, prove it, and otherwise provide a counterexample.
    e. Let T and T' be two different trees (not necessarily binary) on the same leafset S. Suppose T^* is a binary tree on leafset S with $C(T^*) \subset C(T) \cup C(T')$. Must T^* be a refinement of T_1 and T_2? If so, prove it, and otherwise provide a counterexample.
32. Suppose that the reference tree T_0 is not fully resolved, and we want to compute the tree error for an estimated tree T on the same leafset that *is* fully resolved. What can you say about the number of false negatives and false positives? Can they be the same? If not, which must be larger? Why?