3

Constructing Trees from True Subtrees

3.1 Introduction

In many approaches to constructing phylogenetic trees, the input is a set $\mathscr{T}$ of trees, each leaf-labeled by a subset of a set S, and the objective is to construct a tree T on S from $\mathscr{T}$. In this chapter, we discuss methods for constructing trees when the set $\mathscr{T}$ is compatible – which in essence means that it is possible to construct a tree T that agrees with every tree in $\mathscr{T}$. In general, this will only happen when every tree in $\mathscr{T}$ is the true species tree on its leafset, and so we refer to this problem as "Constructing trees from true subtrees." This is a simplification of the more general problem where the input trees may have some estimation error, which is the more realistic case (most estimated species trees have some error); this version of the problem is addressed in Chapter 7. Another variant of this problem is where the input trees are estimated (and perhaps correct) gene trees. Since gene trees can differ from the species tree due to biological processes such as incomplete lineage sorting (ILS) and gene duplication and loss, the estimation of a species tree from a set of gene trees presents its own challenges, which are discussed in Chapter 10.

3.2 Tree Compatibility

One of the key concepts in this chapter is tree compatibility, both for rooted and unrooted trees. In Chapter 2, we defined the concept of compatibility in the context of clades and bipartitions; as we will see, these definitions extend naturally to rooted and unrooted trees, respectively.

3.2.1 Unrooted Tree Compatibility

We begin with unrooted trees. Recall that if we are given a tree T on leafset S and a proper subset X of S, we can compute the homeomorphic tree $T|X$ (see Section 2.9). Now suppose we have two trees, t and T, where t has leafset X and T has leafset S, with $X \subseteq S$. Then we will say that the larger tree T is **compatible** with the smaller tree t if $T|X$ is a refinement of t. Note that we do not require that $t = T|X$; what this means is that t may be unresolved, which can allow $C(T|X)$ to have additional bipartitions beyond those present in $C(t)$. Note

also that the term "compatibility" is not symmetric, even when the two trees have the same leafset. For example, the unrooted tree $(1,(2,(3,4)))$ is compatible with the unrooted star tree $(1,2,3,4)$, but not vice versa.

Definition 3.1 Let $\mathscr{T}$ be a set of unrooted trees and let S be the set of taxa that appear at the leaves of trees in $\mathscr{T}$. We will say that $\mathscr{T}$ is **compatible** if and only if there is a tree with leafset S that is compatible with every tree in $\mathscr{T}$. When $\mathscr{T}$ is compatible, then any minimally resolved supertree that is compatible with every tree in $\mathscr{T}$ is called a **compatibility supertree** for the set $\mathscr{X}$. Furthermore, any supertree that is compatible with every tree in $\mathscr{T}$ but that is not minimal is a refinement of some compatibility supertree, and so is called a **refined compatibility supertree**.

Recall that the Robinson–Foulds (RF) distance between two trees is the bipartition distance defined by the edges in the trees.

Theorem 3.2 *Let* $\mathscr{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{k}\}$ *be a set of compatible fully resolved trees with* $\mathrm{S_i}$ *the leafset of* $\mathrm{t_i}$. *Then* $\sum_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{RF}(\mathrm{T}|\mathrm{S_i}, \mathrm{t_i}) = 0$, *where* T *is a compatibility supertree for* $\mathscr{T}$.

Proof Note that two binary trees on the same leafset that are compatible must be identical. Hence, since every tree $t_i \in \mathscr{T}$ is fully resolved, $t_i = T|S_i$ for $i = 1, 2, \ldots, k$. The rest follows. □

3.2.2 Rooted Tree Compatibility

Just as we said that a set $\mathscr{T}$ of unrooted trees is compatible if there is a compatibility supertree (i.e., a tree on the full set of taxa that is compatible with every tree in $\mathscr{T}$), the same statement can be made for rooted trees. However, to make this precise we need to say what we mean for two rooted trees t and T to be compatible, when the leafset X of t is a subset of the leafset S of T. The only difference between rooted and unrooted trees is that to determine if t and T are compatible, we examine the homeomorphic *rooted* subtree induced in T by X.

3.3 The Algorithm of Aho, Sagiv, Szymanski, and Ullman: Constructing Rooted Trees from Rooted Triples

We now present an algorithm for constructing a rooted tree from its set of "rooted triples," where by "rooted triple" we mean a rooted three-leaf tree. We will also sometimes refer to rooted triples as "triplets," "triplet trees," or "rooted triplet trees." We indicate the rooted triple on a, b, c in which a and b are more closely related by $((a,b),c)$, by $ab|c$, or by any of the equivalent representations. We describe this algorithm for the more general case, where the input is a set of rooted triplets (maybe not one containing a tree on all possible sets of three leaves), and we wish to know if the set is compatible, and construct a tree agreeing with the input if it exists.

Algorithm for determining if a set of rooted triples is compatible. Suppose we are given a set X of rooted triples, and we wish to know if X is compatible, which means that there is a tree T on which all the rooted triples in X agree. Furthermore, when the set X is compatible, we wish to return some tree T on which all the rooted triples agree (i.e., a refined compatibility supertree).

The first algorithm for this problem was developed by Aho et al. (1978), and is widely known in phylogenetics. For the sake of brevity, we will often refer to this algorithm as the ASSU algorithm, after the four authors Aho, Sagiv, Szymanski, and Ullman.

The input to the ASSU algorithm is a pair (S, Trip), where S is a set of taxa and *Trip* is a set of rooted three-leaf trees on S, with at most one tree for any three leaves; furthermore, we assume that every tree in *Trip* is fully resolved (i.e., of the form $((a,b),c)$).

- Group the set S of taxa into disjoint sets, as follows. We make a graph where the vertices are the elements of the set S and we add an edge (a,b) for every $((a,b),c) \in \mathit{Trip}$.
- If the graph is connected, then return *Fail* and exit – no tree is possible. Otherwise, let $C_1, C_2, \ldots, C_k$ ($k \geq 2$) be the connected components of the graph. For each connected component C_i,
  - Let Trip_i be the set of triplets in *Trip* that have all their leaves in C_i.
  - Recurse on (C_i, Trip_i); if this returns *Fail*, then also return *Fail*. Otherwise, let t_i be the output of the recursion.

  Make the roots of the trees $t_1, t_2, \ldots, t_k$ all children of a common root, and return the constructed rooted tree.

This surprisingly simple algorithm is provably correct, and runs in polynomial time. A simple proof by induction on the number of leaves establishes correctness (Aho et al., 1978).

3.4 Constructing Unrooted Binary Trees from Quartet Subtrees

3.4.1 Notation

Definition 3.3 The binary quartet tree on a,b,c,d that splits a,b from c,d can be represented by $ab|cd$, $(ab|cd)$, (ab,cd), $(a,b|c,d)$, or $((a,b),(c,d))$. We represent the star tree on a,b,c,d by (a,b,c,d) (see Definition 2.25). Every set X of four leaves defines a quartet tree $T|X$. Therefore, given an unrooted tree T on n distinctly labeled leaves, we denote by $\mathbf{Q(T)}$ the set of quartet trees induced by T on its leafset, and by $\mathbf{Q}_r(\mathbf{T})$ the set of all fully resolved (i.e., binary) quartet trees in $Q(T)$. Hence, if T is a binary tree, then $Q_r(T) = Q(T)$.

3.4.2 The All Quartets Method

The Quartet Tree Compatibility problem is as follows:

- Input: Set Q of quartet trees.

- Output: Tree T such that $Q \subseteq Q(T)$ (if such a tree T exists) or *Fail*.

The Quartet Tree Compatibility problem is NP-complete, even when all the trees in Q are binary (Steel, 1992), but some special cases of the problem can be solved in polynomial time, as we will show. In particular, the case where all the trees in Q are binary and Q has at least one tree on every four leaves in a set S can be solved in polynomial time, using the All Quartets Method, which we now describe.

The *All Quartets Method* takes as input a set Q of quartet trees (each of them fully resolved), with one tree for every four leaves. The output is either a tree T such that $Q(T) = Q$ or else *Fail*.

We call this algorithm the *All Quartets Method*, because it assumes that the input is the set of all quartet trees for an unknown tree T; indeed, while the method can be applied to proper subsets of $Q(T)$, there are no guarantees that the method will correctly answer whether the input quartet trees are compatible under this more general condition. However, when the input contains a tree on every four leaves, then we can prove that the All Quartets Method is correct.

All Quartets Method:

- Step 1: If $|S| = 4$, then return the tree in Q. Else, find a pair of taxa s_i, s_j that are always grouped together in any quartet that includes both s_i and s_j. If no such pair exists, **return** "Incompatible input" and **exit**. Otherwise, remove s_i.
- Step 2: Recursively compute a tree T' on $S - \{s_i\}$.
- Step 3: Return the tree created by inserting s_i next to s_j in T'.

Example 3.4 Consider the unrooted tree $T = (1,(2,(3,(4,5))))$ and its set of quartet trees $Q(T) = \{12|34, 12|35, 12|45, 13|45, 23|45\}$. Note that taxa 1 and 2 are always grouped together in all the quartets that contain them both, but so also are 4 and 5. On the other hand, no other pair of taxa are always grouped together. If we remove taxon 1, we are left with the single quartet on $2, 3, 4$, and 5. The unrooted quartet tree on that set is $23|45$. We reintroduce the leaf for 1 as sibling to 2, and obtain the unrooted tree given by $(1,(2,(3,(4,5))))$.

3.4.3 Inferring Quartet Trees from Other Quartet Trees

Recall that the All Quartets Method will construct T given $Q(T)$, the set of homeomorphic quartet trees of T. This suggests a method for constructing a tree, in which unrooted trees are estimated on four leaves at a time, and then combined into a tree on the full dataset using the All Quartets Method. This approach will produce the true tree, but only if all of the estimated trees are correct – even one single error will make the entire approach fail.

Since some of the estimated quartet trees might be incorrect, we may wish to try to compute a tree from a proper subset of its quartet trees – ones that look like they may be correctly computed. Here we present one attempt at solving this problem, in which we use just a subset of the possible quartet trees and try to infer the remaining quartet trees. If we

succeed, then we can apply the All Quartets Method to the final set of quartet trees, and construct the tree T.

Suppose Q is a set of quartet trees. We will show two rules for inferring new quartet trees from pairs of quartet trees, so that whenever all the trees in Q are true, then the added trees *must also be true*. In other words, we assume that the input set Q satisfies $Q \subseteq Q(T)$ for some (unknown) tree T, and we ensure that any added quartet trees are also in $Q(T)$. In the rules below, we will consider $ab|cd$ to be the same quartet tree as $ba|cd$, $ba|dc$, and $ab|dc$.

- Rule 1: If $ab|cd$ and $ac|de$ are in $Q \subseteq Q(T)$, then $ab|ce, ab|de$, and $bc|de$ are also in $Q(T)$. Hence, if any of these three quartet trees are missing from Q, we can add them to Q.
- Rule 2: If $ab|cd$ and $ab|ce$ are in $Q \subseteq Q(T)$, then $ab|de$ must be in $Q(T)$. Hence, if $ab|de$ is missing from Q, we can add $ab|de$ to Q.

It is easy to see that these rules are valid, and so if the input set Q contains only correct quartet trees (meaning true quartet trees for some unknown tree T), then the quartet trees that are added are also correct quartet trees for that unknown tree T. These two rules are *dyadic* (also called "second order") rules, in that they are based on combining two quartet trees to infer additional quartet trees.

The dyadic closure of a set Q of quartet trees is computed by repeatedly applying Rules 1 and 2 to pairs of quartet trees until no additional quartet trees can be inferred. The final set of quartet trees is the **dyadic closure** of Q, and is denoted by $cl_2(\mathbf{Q})$. The discussion above clearly shows the following:

Theorem 3.5 *Suppose* $\mathrm{Q} \subseteq \mathrm{Q(T)}$ *for some tree* T. *Then* $\mathrm{cl_2(Q)} \subseteq \mathrm{Q(T)}$.

Now suppose we were lucky, and $cl_2(Q) = Q(T)$; then we can construct T using the All Quartets Method. However, if Q is too small a set, then $cl_2(Q)$ may not be equal to $Q(T)$. In the next section, we investigate how big Q has to be, in order for $cl_2(Q) = Q(T)$.

3.4.4 Constructing a Tree from a Subset of its Quartet Trees

We begin by defining the "short quartet trees" of an edge-weighted tree T. As we will see, if Q contains all the short quartet trees of a tree T and no incorrect quartet trees, then $cl_2(Q) = Q(T)$.

Definition 3.6 Let T be a binary tree and $w : E(T) \to R^+$ be the positive edge weighting of T. The deletion of an internal edge $e \in E(T)$ (and its endpoints) creates four subtrees, A, B, C, and D. Let a, b, c, d be four leaves nearest to e from these four subtrees, with $a \in A, b \in B, c \in C$ and $d \in D$. The definition of "nearest" is based on the path length, and takes the edge weights into account. Then a, b, c, d is a **short quartet** around e, and the quartet tree on a, b, c, d defined by T is called a **short quartet tree**. Since there can be more than one nearest leaf in a given subtree to the edge e, there can be more than one short quartet

around e. The set of all short quartets over all internal edges of T is called the set of **short quartets of** T and is denoted $Q_{short}(T)$, while the set of short quartet trees over all internal edges of T is called the set of **short quartet trees of** T and is denoted $Q^*_{short}(T)$.

Example 3.7 Consider the caterpillar tree $(1,(2,(3,\ldots,(99,100)\ldots)$. There are 97 internal edges of the tree, each of which contributes at least one short quartet. A careful inspection of this tree shows that the set $Q_{short}(T)$ has 99 quartets. For example, $Q_{short}(T)$ includes $\{1,2,3,4\},\{2,3,4,5\},\{1,3,4,5\}$, and $\{3,4,5,6\}$. Now consider the set $Q^*_{short}(T)$ of trees on the short quartets of T. A little examination will show that T is the only tree on the same leafset that can contain all the short quartet trees. For example, Rule 1, applied to $12|34$ and $23|45$, produces $13|45, 12|45$, and $12|35$. In other words, applying Rule 1 produced the five quartet trees on four-leaf subsets of $1,2,3,4,5$. Indeed, if we compute $cl_2(Q^*_{short}(T))$, we will obtain $Q(T)$.

In other words, the following theorem can be proven:

Theorem 3.8 *(From Erdös et al. (1997)) If* $\mathrm{Q}^*_{\mathrm{short}}(\mathrm{T}) \subseteq \mathrm{Q} \subseteq \mathrm{Q}(\mathrm{T})$ *for some binary unrooted tree* T*, then* $\mathrm{cl}_2(\mathrm{Q}) = \mathrm{Q}(\mathrm{T})$*. In other words, under the assumption that* T *is the true tree, then if* Q *has no incorrect quartet trees and also contains all the short quartet trees of* T*, then the dyadic closure of* Q *will include the true tree on every four leaves in* T*, and nothing beyond that.*

Corollary 3.9 *(From Erdös et al. (1997)) Let* T *be a binary unrooted tree and let* $\mathrm{Q}^*_{\mathrm{short}}(\mathrm{T})$ *be the set of short quartet trees for* T *for some edge-weighting of* T*. If* T' *is a tree on the same leafset as* T *and* $\mathrm{Q}^*_{\mathrm{short}}(\mathrm{T}) \subseteq \mathrm{Q}(\mathrm{T}')$*, then* $\mathrm{T}' = \mathrm{T}$.

Proof Assume that T and T' are on the same leafset, and that $Q^*_{short}(T) \subseteq Q(T')$. We begin by noting that Theorem 3.8 implies that $cl_2(Q^*_{short}(T)) = Q(T)$. Now, since $Q^*_{short}(T) \subseteq Q(T')$, by Theorem 3.5, $cl_2(Q^*_{short}(T)) \subseteq Q(T')$. Hence, $Q(T) \subseteq Q(T')$. Since T and T' are on the same leafset, $Q(T)$ and $Q(T')$ have the same cardinality. Hence, it must follow that $Q(T) = Q(T')$, and so $T = T'$. □

What these results establish is that if we were lucky enough to find such a set Q of quartet trees (one that has no incorrect quartet trees, and yet contains all the short quartet trees), we could use the dyadic closure to infer $Q(T)$, and then construct the tree T using the All Quartets Method. See Section 8.12 for the Dyadic Closure Method (Erdös et al., 1999a), which builds on the theory we have outlined here.

3.5 Testing Compatibility of a Set of Trees

Recall that a compatibility supertree for a set $\mathscr{T}$ of trees is a minimally resolved tree that is compatible with every tree in $\mathscr{T}$ (Definition 3.1).

Example 3.10 Consider the set X of unrooted trees $X = \{(ab|cde), (bc|def), (cd, eg)\}$. The caterpillar tree $(a, (b, (c, (d, (e, (f, g))))))$ (see Definition 2.23) is a refined compatibility supertree for X, and so X is compatible.

Theorem 3.11 *The* Unrooted Tree Compatibility problem – *determining if a set* X *of unrooted trees, each leaf-labeled by elements from* S*, is compatible – is NP-complete, even if all the trees are binary (fully resolved).*

Proof This result follows from the fact that quartet compatibility is NP-complete (Steel, 1992). □

As noted before, some special cases of Unrooted Tree Compatibility can be solved in polynomial time. For example, we already know that the All Quartets Method can construct a tree T from its set $Q(T)$ of quartet trees, and furthermore that the All Quartets Method can be used to determine if a set X of quartet trees is compatible when X contains a tree on every four taxa. Hence, if X is a set of unrooted trees and every four taxa are in at least one tree in X, then we can determine if X is compatible in a straightforward, if brute-force, way: We replace every tree t in X by its set $Q(t)$, and thus make X into a set of quartet trees that contains a tree on every four taxa. We can then apply the All Quartets Method to the set of quartet trees we have created to determine if the quartet trees are compatible.

Similarly, suppose that we are given a set X of rooted leaf-labeled trees and we want to know if there is a rooted tree T that induces each of the trees in X as homeomorphic subtrees. To answer this question, we can *encode* each of the rooted leaf-labeled trees in X by its set of rooted triplet trees, and then run the ASSU algorithm (see Section 3.3) on the resultant set of rooted triplet trees. If the output is a rooted tree that induces all the rooted triplet trees, then it follows that the set X is compatible. The other possible outcome is that the algorithm fails to return a tree (because during at least one of the recursive calls, the graph has a single connected component); in that case, the rooted triplet trees are not compatible, and hence the set X is incompatible. In other words, it is easy to see that testing a set of rooted binary trees for compatibility is a polynomial time problem. We summarize this as follows:

Theorem 3.12 *The* Rooted Binary Tree Compatibility Problem – *determining if a set* X *of rooted binary trees, each leaf-labeled by elements from* S*, is compatible – can be solved in polynomial time.*

The extension of this problem to rooted trees that can have polytomies is also solvable in polynomial time, but the proof of this is left to the reader.

3.6 Further Reading

In this chapter we described several quartet-based methods for tree estimation; in each of these cases, we assumed that the set of quartet trees is computed using some technique, and the objective is to construct a tree that is consistent with the quartet trees. Yet, since quartet trees are not always perfectly estimated, these tree estimation methods can fail to construct

any tree. In later chapters, we will return to quartet-based methods for tree estimation, and address this inference problem when the input trees are presumed to have some error.

3.7 Review Questions

1. What is a rooted triple?
2. For each problem below, state whether it is solvable in polynomial time, NP-hard, or of unknown computational complexity:
   - Determining if a set of rooted triples is compatible.
   - Determining if a set of unrooted quartet trees is compatible.
   - Determining if a set of rooted leaf-labeled trees is compatible.
   - Determining if a set of unrooted leaf-labeled trees is compatible.
3. If T is an unrooted leaf-labeled tree, what does $Q(T)$ refer to?
4. What does $ab|cd$ refer to?
5. What is the *All Quartets Method*? Does it run in polynomial time?
6. Suppose you are given a set Q of unrooted quartet trees that contains a tree for *some but not all* of the different sets of four species taken from a species set S. Can you use the All Quartets Method to test for compatibility of the set Q?
7. Suppose you are given a set R of rooted triplet trees that contains a rooted tree for *some but not all* of the different sets of three species taken from a species set S. Can you use the ASSU method to test for compatibility of the set R?
8. What is the difference between a refined compatibility supertree and a compatibility supertree?

3.8 Homework Problems

1. Make up a rooted tree on six leaves, and write down all its rooted triples. Then make up another rooted tree on the same six leaves, and write down all its rooted triples. How many rooted triples do your trees disagree on?
2. Make up two rooted trees on at least five leaves that differ in exactly one rooted triple.
3. Consider the rooted caterpillar tree given by $(1,(2,(3,(4,5))))$.
   a. Write down the set of rooted triples for the tree.
   b. Apply the ASSU algorithm to this set of rooted triples. What do you find?
4. By design, if the ASSU algorithm returns a tree on input set X of rooted triplets, then the tree it returns will be compatible with all the input triplet trees in X. But is it guaranteed to produce a compatibility supertree, or might the output be a refined compatibility supertree?
5. Is it possible to have a compatible set X of rooted triplets for which some pair of leaves i,j is not separated in any rooted triplet in which they both appear, but where i and j are not siblings in *any* tree that is compatible with the set of rooted triplets? If so, provide the example, and otherwise prove it is impossible.

6. Suppose we modify the ASSU algorithm as follows. We compute the equivalence relation, and if there is more than two equivalence classes, $C_1, C_2, \ldots, C_k$ (with $k > 2$) we make *two* subproblems, C_1 and $C_2 \cup C_3 \cup \ldots \cup C_k$. Otherwise, we don't change the algorithm. Does this also solve rooted triplet compatibility? (Prove or disprove.)
7. Prove that the ASSU algorithm correctly solves the problem of determining if a set *Trip* of rooted, fully resolved, three-leaf trees is compatible. (Hint: use induction.)
8. Consider input sets $\mathscr{T}$ of rooted trees, each on a subset of taxon set S, and suppose some of them have polytomies. We will consider the polytomies to be *soft*, meaning that we do not consider them to imply any constraint on the tree on the full set S. We would like to find a tree T on the full taxon set that is compatible with every tree in $\mathscr{T}$, meaning that it will either agree with the trees in $\mathscr{T}$ or refine the trees when restricted to the same leafset.

   a. Show how to use the ASSU algorithm so that it solves this problem.
   b. Prove your algorithm correct.

9. Consider input sets *Trip* of rooted triplet trees, each on a subset of taxon set S, and suppose some of them are polytomies (i.e., of the form (a,b,c)). Suppose we consider the triplet tree (a,b,c) to be a *hard polytomy*, meaning that it imposes a constraint on the tree T on S to induce the rooted tree (a,b,c) on $\{a,b,c\}$, rather than just be compatible with it. (A soft polytomy is a polytomy that does not imply such a constraint.) In other words, we would like to find a tree on the full taxon set that induces each triplet tree in *Trip*.

   a. Modify the ASSU algorithm so that it solves this problem.
   b. Prove your algorithm correct.

10. Suppose that we allow triplet trees to represent hard polytomies. For example, we would use (a,b,c) to indicate that the compatibility supertree (if it exists) induces the unresolved tree (a,b,c). Suppose that ASSU ignores these triplet trees. Give an example of an input set *Trip* of triplet trees that is allowed to have these hard polytomies, and show that the ASSU algorithm will not correctly solve the compatibility problem on *Trip*. Thus, either *Trip* should be compatible but the ASSU algorithm should say it is not, or vice versa.
11. In the text, we stated that the ASSU algorithm is polynomial time. Provide a running time analysis, where the input is a set of k rooted triplet trees drawing their leaves from set S of n taxa.
12. Make up two different unrooted trees on the same set of five leaves, but try to make them disagree on as few unrooted quartet trees as possible. How many do they disagree on?
13. Construct a tree on leafset $\{a,b,c,d,e,f\}$ that induces each of the following quartet trees:

    - $(ab|cd)$
    - $(ab|ce)$

- $(ac|de)$
- $(bc|de)$
- $(ab|de)$
- $(ab|cf)$
- $(ab|df)$
- $(ab|ef)$
- $(ac|df)$
- $(ac|ef)$
- $(ad|ef)$

14. Recall that the All Quartets Method is designed to solve the Quartet Compatibility problem when the input is a set of fully resolved (i.e., binary) trees, with exactly one tree on every set of four taxa. Prove that the All Quartets Method is correct for such inputs. (Hint: use induction.)
15. Consider the case where the unrooted tree T is not binary, and so can have a node of degree greater than three. Give an example of such a tree T, so that when the All Quartets Method is applied to $Q(T)$ it fails to recover the tree T.
16. Modify the All Quartets Method so that it will correctly handle inputs Q that contain quartet trees with hard polytomies.
17. Suppose we have a set X of unrooted binary trees, and we encode each tree $T \in X$ by its set $Q(T)$ of quartet trees. Prove or disprove: The set X is a compatible set of unrooted trees if and only if $\bigcup_{T \in X} Q(T)$ is a compatible set of quartet trees.
18. Suppose we have a set X of rooted binary trees, and we encode each tree $T \in X$ by its set $R(T)$ of rooted triplet trees. Prove or disprove: The set X is a compatible set of rooted trees if and only if $\bigcup_{T \in X} R(T)$ is a compatible set of rooted triplet trees.
19. Consider the Split Constrained Quartet Support problem. How would you define the input set X of allowed bipartitions so that the solution to the problem gave an optimal tree over all possible binary trees on the taxon set?
20. Suppose you have a collection $\mathscr{T}$ of unrooted trees, not necessarily binary, all with exactly the same leafset $\{1,2,3,\ldots,n\}$. Suppose that the set $\mathscr{T}$ is compatible. Express the maximum size of $\mathscr{T}$ as a function of n.
21. Suppose you have a collection $\mathscr{T}$ of unrooted binary trees, each of them different, all with exactly the same leafset $\{1,2,3,\ldots,n\}$. Suppose that the set $\mathscr{T}$ is compatible. Express the maximum size of $\mathscr{T}$ as a function of n.
22. Consider the following three unrooted trees:

    - $T_1 = (1,(3,(5,(6,7))))$
    - $T_2 = (1,(2,((4,8),(3,7))))$
    - $T_3 = (2,((4,(3,5)),1))$

    Answer the following questions:

    a. Are these unrooted trees compatible? Justify your answer.
    b. Root all three trees at leaf 1, and draw the rooted versions of these trees. Are these rooted trees compatible? Justify your answer.