7
Supertrees

7.1 Introduction

The basic objective of most supertree studies is the assembly of a large species tree from a set of species trees that have been estimated on potentially smaller sets of taxa. Indeed, it is generally believed that construction of the Tree of Life, which will encompass millions of species, will require supertree methods, because no software will be able to deal with the computational challenges involved in such a difficult task. More recently, however, new uses of supertree methods have been discovered, especially in the context of divide-and-conquer strategies. This chapter examines both applications of supertree methods.

Traditionally, supertree methods were used to combine trees computed by different researchers that had already been estimated for different taxon sets. In this case, the person constructing the supertree has no control over the inputs, neither how the different subset trees were constructed nor how the taxon sets of the different subset trees overlap. Furthermore, the person constructing the supertree may not have easy access to the data (e.g., sequence alignments) on which the subset trees were constructed.

A modern and more interesting use of supertree methods is in the context of a divide-and-conquer strategy to construct a very large tree, or to enable a statistically powerful but computationally intensive method to be applied to a larger dataset. In such a strategy, a large set of taxa is divided into overlapping subsets, trees are estimated (using the desired method) on the subsets, and the estimated subset trees are combined into a tree on the full set of taxa using a supertree method.

Divide-and-conquer techniques for constructing large trees have many desirable features: (1) the subsets can be made small enough that expensive methods can be used to construct trees on them; (2) different methods can be used on each subset, thus making it possible to better address heterogeneity within the full dataset; and (3) the subsets can be created so as to have desirable overlap patterns. The first two of these features tend to increase the accuracy of the estimated subset trees, while the third feature can make it easier to construct an accurate supertree from the subset trees (Wilkinson and Cotton, 2006). We will return to the topic of divide-and-conquer strategies and how to use them to construct large trees under a variety of scenarios in Chapter 11. For now, just be aware that supertree methods are more than just ways of assembling large trees from other trees; they are key ingredients in developing methods to enable powerful but expensive methods to run on ultra-large datasets.

Just as in the consensus setting, the input to the supertree method is a set of source trees, referred to as a profile; the distinction between consensus and supertree problems is that in the supertree setting we do not constrain the input trees to have identical leafsets. Hence, any supertree method can be used as a consensus tree method, but the converse is not true. Because of this difference, supertree methods have greater applicability than consensus methods. On the other hand, many optimization problems posed in the supertree setting are NP-hard, whereas the same optimization problems posed in the consensus setting are polynomial time. Seeing supertree method development as an extension to a consensus tree method was first applied by Gordon (1986) in the development of the Gordon's consensus, an extension of the strict consensus to the supertree setting, and then formalized in Cotton and Wilkinson (2007), who developed two supertree methods (both called Majority-Rule Supertrees) that are generalizations of the majority consensus tree.

In the idealized condition, all the source trees are compatible with each other, and the objective will be to construct a compatibility supertree. However, since source trees are estimated trees, they are likely to have some estimation error, and no compatibility supertree will exist. Therefore, rather than trying to find a compatibility supertree (see Section 3.5), the main objective is to find a supertree that is somehow as close as possible, with respect to some criterion, to the input source trees (e.g., minimizing the sum of the distances to the input source trees (Thorley and Wilkinson, 2003)). Examples of optimization problems that have been used to define supertrees include Matrix Representation with Parsimony (MRP) (Baum and Ragan, 2004), Matrix Representation with Likelihood (MRL) (Nguyen et al., 2012), Robinson–Foulds Supertree (RFS) (Wilkinson et al., 2005), Quartet Median Tree, and Maximum Quartet Support Supertree (MQS), all described below. Each of these optimization problems is formulated for inputs containing unrooted source trees, but most have equivalent formulations for inputs of rooted source trees. Supertree software implementing specific approaches (some of which are based on optimality criteria) include MinCut Supertree (Semple and Steel, 2000), Modified MinCut Supertree (Page, 2002), MinFlip Supertree (Chen et al., 2003, 2006), and the MRF Supertree (Burleigh et al., 2004).

Unfortunately, all the optimization problems mentioned above, and in fact nearly all optimization problems that have been posed for supertree construction, are NP-hard. Therefore, heuristics rather than exact algorithms are used to find good solutions to these optimization problems. Thus, supertree methods are, like most problems in phylogenetics, understood by the theoretical properties of the optimization problems on which they are based (such as whether the criteria are biased toward large source trees (Wilkinson et al., 2005)), by the details of their implementations, and by how well they perform on data.

For the rest of this chapter, we will assume that the profile given as input to the supertree problem is $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$. We will let S_i denote the leafset of tree t_i, and we let $S = \cup S_i$. Most of the supertree problems we will discuss involve profiles of unrooted trees; however, we will also (briefly) describe some methods for computing supertrees when the profile contains rooted trees.

7.2 Compatibility Supertrees

Recall that if T is a compatibility supertree, then T is a minimally resolved tree on leafset $S = \cup S_i$ that is compatible with every tree t_i (Definition 3.1). Equivalently, T is the minimally resolved tree such that one of the following conditions holds: $C(t_i) \subseteq C(T|S_i)$ for every i or $\sum_{i=1}^{k} |C(t_i) \setminus C(T|S_i)| = 0$. Recall also that a refined compatibility supertree is any tree that is compatible with all the trees in the profile, and need not be a minimally resolved such tree.

Now, consider the compatibility supertree problem:

- Input: $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$, where t_i has leafset S_i and is an unrooted tree.
- Output: A compatibility supertree, if it exists.

Note that even if a profile has a compatibility supertree, it can have more than just one.

Example 7.1 Let $t_1 = ((a,b),(c,d))$ and $t_2 = ((a,b),(c,e))$. Then $(a,(b,(c,(d,e))))$ and $(a,(b,(d,(c,e))))$ are both compatibility supertrees for the profile $\{t_1,t_2\}$.

Determining if a compatibility supertree exists is NP-complete, since if every source tree is a quartet tree, the problem reduces to quartet compatibility, which is NP-complete (Steel, 1992).

7.3 Asymmetric Median Supertrees

In Section 6.2.5, we described a consensus method called the asymmetric median tree that is related to determining if a profile of source trees was compatible. Since consensus methods require that the profile contain trees that have the same set of leaves, we extend that approach to allow for profiles of trees that are on different taxon sets, and we call this the **Asymmetric Median Supertree**:

- Input: Profile $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ of unrooted trees, with t_i having leafset S_i.
- Output: Tree T with leafset $S = \cup_{i=1}^{k} S_i$ that minimizes $\sum_{i=1}^{k} |C(t_i) \setminus C(T)|$.

Lemma 7.2 *Let $\mathscr{T}$ be a profile of unrooted source trees, and suppose that $\mathscr{T}$ is compatible so that a compatibility supertree exists. Then* T *is a refined compatibility supertree for $\mathscr{T}$ if and only if* T *is an asymmetric median supertree for $\mathscr{T}$. Furthermore, any refined compatibility supertree* T *satisfies $\sum_{i=1}^{k} |C(t_i) \setminus C(T)| = 0$.*

Computing the asymmetric median supertree is also NP-hard, since a compatibility supertree would be an optimal solution (if it exists), and determining if a compatibility supertree exists is NP-complete. In general, source trees are not required to be binary; however, if we add that constraint, then any compatibility supertree T will satisfy $T|S_i = t_i$ for each $i = 1,2,\ldots,k$, and so will also satisfy $RF(T|S_i,t_i) = 0$ for each $i = 1,2,\ldots,k$. This leads to the statement of the Robinson–Foulds Supertree problem, the subject of the next section.

7.4 Robinson–Foulds Supertrees

7.4.1 Problem Formulation

The Robinson–Foulds Supertree (RFS) is a supertree T that minimizes the total Robinson–Foulds (RF) distance between T and the source trees (Cotton and Wilkinson, 2007; Bansal et al., 2010; Chaudhary et al., 2012; Chaudhary, 2015; Kupczok, 2011). Since source trees can be small, we use the extension of the definition of the RF distance from Cotton and Wilkinson (2007) so that it can be used in supertree estimation:

Definition 7.3 Let T have leafset S and let t have leafset $S' \subseteq S$. Then $RF(T,t)$ denotes the RF distance between $T|S'$ and t.

Example 7.4 Let $T = (1,((6,7),(4,(3,(5,2)))))$ and $t = (5,((3,2),(6,7)))$. S', the leafset of t, is $\{2,3,5,6,7\}$. To compute the RF distance between T and t we compute $T|S'$, which is $((6,7),(3,(5,2)))$. The non-trivial bipartitions of $T|S_i$ are $(235|67)$ and $(25|367)$. The non-trivial bipartitions of t are $(235|67)$ and $(23|567)$. Therefore, $RF(T,t) = 2$.

The RF Supertree problem is:

- Input: Profile $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ of unrooted source trees.
- Output: Tree T_{RFS} that minimizes the total RF distance to the source trees, i.e.,

$$T_{RFS} = arg\,\min_{T} \sum_{i=1}^{k} RF(T,t_i).$$

The RFS and the asymmetric median supertree are clearly similar, as both are based on bipartitions. However, the asymmetric median supertree does not penalize the supertree for containing bipartitions that do not appear in the source trees, and the RFS does. Thus, an optimal RFS and an optimal asymmetric median supertree can be different. However, the two problems are related to the compatibility supertree problem, as we now show:

Theorem 7.5 *Let* $\mathscr{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{k}\}$ *be a set of trees where* $\mathrm{t_i}$ *has leafset* $\mathrm{S_i}$*, and let* $\mathrm{S} = \cup_\mathrm{i} \mathrm{S_i}$. *Then* T *is a compatibility supertree for* $\mathscr{T}$ *if and only if* T *has leafset* S *and* $|\mathrm{C}(\mathrm{t_i}) \setminus \mathrm{C}(\mathrm{T}|\mathrm{S_i})| = 0$ *for all* $\mathrm{i} = 1,2,\ldots,\mathrm{k}$. *Hence, when a compatibility supertree exists, then* T *is an asymmetric median supertree if and only if* T *is a refined compatibility supertree. Furthermore, if each* $\mathrm{t_i}$ *is a fully resolved tree, then any refined compatibility supertree* T *satisfies* $\mathrm{RF}(\mathrm{T},\mathrm{t_i}) = 0$ *for all* $\mathrm{i} = 1,2,\ldots,\mathrm{k}$. *Conversely, if* T *is a tree on leafset* $\mathrm{S} = \cup_\mathrm{i} \mathrm{S_i}$ *that satisfies* $|\mathrm{C}(\mathrm{t_i}) \setminus \mathrm{C}(\mathrm{T}|\mathrm{S_i})| = 0$ *for all* i, *then by definition* $\mathscr{T}$ *is a compatible set of trees, and* T *is a refined compatibility supertree for* $\mathscr{T}$.

We will use this analysis to prove that finding the optimal RFS is NP-hard.

Theorem 7.6 *The Robinson–Foulds Supertree problem is NP-hard.*

Proof Suppose that $\mathscr{A}$ is an algorithm that solves the RFS problem in polynomial time; thus, $\mathscr{A}(\mathscr{T})$ is an RFS for the profile $\mathscr{T}$. We will show that we can use $\mathscr{A}$ to solve the Unrooted Tree Compatibility problem in polynomial time, for the case where the input trees are all binary. Since Unrooted Tree Compatibility is NP-complete even when the input trees are all binary (Theorem 3.11) this will prove that the RFS problem is NP-hard.

Let $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ be a profile of unrooted binary trees, with S_i the leafset of t_i for $i = 1, 2, \ldots, k$. We run $\mathscr{A}$ on the profile $\mathscr{T}$ to obtain $T = \mathscr{A}(\mathscr{T})$; thus, T is an optimal solution to the RFS problem. We then compare $T|S_i$ to t_i for each $i = 1, 2, \ldots, k$. If $T|S_i$ refines t_i for every $i = 1, 2, \ldots, k$, then T is a refined compatibility supertree, and otherwise T is not a refined compatibility supertree. This is easily checked in polynomial time. Since we assume that each t_i is binary (i.e., fully resolved), then if $T|S_i$ refines t_i it follows that $T|S_i$ is identical to t_i. If T passes this test for every $i = 1, 2, \ldots, k$, then we will say that $\mathscr{T}$ is compatible, and otherwise we will say that $\mathscr{T}$ is not compatible.

To prove that we have solved the Unrooted Tree Compatibility problem (for the case where every tree in the input profile is fully resolved), we just need to show that the profile $\mathscr{T}$ is compatible if and only if $T = \mathscr{A}(\mathscr{T})$ passes this test for every i. So suppose T passes the test for every $i = 1, 2, \ldots, k$. Then T is a refined compatibility supertree for $\mathscr{T}$, and so the profile $\mathscr{T}$ is compatible. Now suppose that the profile $\mathscr{T}$ is compatible, but that T is not a refined compatibility supertree for $\mathscr{T}$. Since $\mathscr{T}$ is compatible, there is a compatibility supertree, T^*. Note that $T^*|S_i = t_i$ (i.e., $RF(T^*, t_i) = 0$) for $i = 1, 2, \ldots, k$, and so T^* is an optimal solution to the RFS problem. Since $\mathscr{A}$ is an exact algorithm for the RFS problem, T^* and T must both have the same score under the RFS criterion. Hence, $RF(T, t_i) = 0$ for each i as well, and so T is also a refined compatibility supertree for $\mathscr{T}$, contradicting our assumption. Hence, the profile $\mathscr{T}$ is compatible if and only if $T = \mathscr{A}(\mathscr{T})$ is a refined compatibility supertree.

Since $\mathscr{A}$ runs in polynomial time, we have shown that if the RFS problem can be solved in polynomial time then the Unrooted Tree Compatibility problem can also be solved in polynomial time when all the source trees are binary. Since Unrooted Tree Compatibility is NP-complete, even when all the source trees are binary (Steel, 1992), the RFS problem is NP-hard. $\square$

The key part of the proof is that when the input profile is compatible, the compatibility tree would be an optimal solution to the RFS problem. Therefore, the same kind of argument can be used to establish that any supertree problem is NP-hard if a compatibility supertree (if it exists) would be an optimal solution. We summarize this point now, since nearly every supertree problem we discuss henceforth will have this property:

Corollary 7.7 *Let π be a supertree optimization problem where the input is a profile of unrooted source trees. If an exact algorithm for π would return a compatibility supertree* *(when it exists), then π is NP-hard. Hence, in particular, the asymmetric median supertree problem is NP-hard.*

7.4.2 Methods for Robinson–Foulds Supertrees

The RFS problem is a popular approach to supertree estimation, and several methods have been developed to try to find good solutions to the problem. Examples of these methods include PluMiST (Kupczok, 2011), Robinson–Foulds Supertrees (Bansal et al., 2010), MulRF (Chaudhary, 2015), and FastRFS (Vachaspati and Warnow, 2016). Most methods for the RFS problem use heuristic searches to attempt to find good solutions to the optimization problem, but FastRFS (Vachaspati and Warnow, 2016) uses an alternative strategy. FastRFS uses the input source trees $\mathscr{T}$ to compute a set X of allowed bipartitions, and then uses dynamic programming to find an exact solution to the RFS problem within that constrained search space. In other words, the dynamic programming strategy guarantees that the tree that is returned has the best possible score among all trees that draw their bipartitions from X. Furthermore, the running time of FastRFS is $O(nk|X|^2)$, where n is the number of species and k is the number of source trees. Hence, when X is small enough, the algorithm is very fast.

The basic version of FastRFS just uses the bipartitions from the input trees; however, since the input trees may not have all the species, this set needs to be enlarged. The enhanced version of FastRFS uses bipartitions from supertrees computed by other fast supertree methods, and adds them to the bipartition set X. As shown in Vachaspati and Warnow (2016), FastRFS finds better solutions to the RFS than the alternative heuristics, and does so very efficiently.

7.5 Matrix Representation with Parsimony

The Matrix Representation with Parsimony (MRP) (Baum and Ragan, 2004) supertree problem is by far the most well known (and most popular) supertree optimization problem. Although several variants of MRP have been posed that differ depending on whether the input trees are rooted or unrooted, we will discuss the version where the input trees are unrooted.

The input to MRP is a profile $\{t_1, t_2, \ldots, t_k\}$ of unrooted trees where t_i has leafset S_i. From this set of trees, we compute a matrix, called the **MRP matrix**, defined by the concatenation of the matrices obtained for each of the trees in the profile. Thus, to define the MRP matrix for the profile, it suffices to show how we define the MRP matrix for a single tree t_i.

The matrix for source tree t_i on taxon set S_i has a row for every element of $S = \cup_i S_i$ and a column for every internal edge in t_i. To define the column associated to the internal edge e in t_i, we compute the bipartition on the leafset S_i defined by removing e from t_i, and we arbitrarily assign 0 to the leaves on one side of this bipartition and 1 to the other side.

We then assign ? to every $s \in S \setminus S_i$. Thus, for $|S| = n$, each edge in t_i is represented by an n-tuple with entries drawn from $\{0, 1, ?\}$. For each such edge we create a column defined by its n-tuple, and we concatenate all these columns together into a matrix that represents the tree t_i. After computing all the matrices for all the trees, we concatenate all the matrices together into one large matrix, which is called the "MRP matrix." Note that in this matrix, every element $s \in S$ is identified with its row. The number of columns is the sum of the number of internal edges among all the trees. Since each tree can have up to $n - 3$ internal edges, this means that the number of columns is $O(nk)$, where k is the number of source trees.

Under the MRP criterion, we seek the tree that optimizes the maximum parsimony criterion with respect to the input MRP matrix. Since the MRP matrix will in general have ?s, we need to explain how these are handled. Let M be the MRP matrix computed from the profile. Since M may have ?s, we consider the set $\mathscr{M}$ of all matrices that can be formed by replacing the ? entries in M by 0 or 1; hence, if M has p entries that are ?, then $|\mathscr{M}| = 2^p$. In other words, $\mathscr{M}$ is the set of binary matrices that agree with M.

Let $MP(T,M)$ denote the maximum parsimony score of tree T for the matrix M. We denote by $MRP(T, \mathscr{T})$ the MRP score of a tree T with respect to the profile $\mathscr{T}$. Then, $MRP(T, \mathscr{T}) = min\{MP(T,M') : M' \in \mathscr{M}\}$. In other words, we are seeking the *best* way of replacing all the question marks (?s) by zeros and ones so that the result gives us the best possible maximum parsimony score.

Thus, the MRP problem is really the maximum parsimony problem on the MRP matrix, with the understanding of how missing data (as represented by ?s) are handled by maximum parsimony. Because binary character maximum parsimony is NP-hard (Foulds and Graham, 1982), MRP is NP-hard. Here we present a proof of this statement.

Theorem 7.8 *Let $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ be an input of unrooted source trees. If $\mathscr{T}$ is compatible, then the MRP matrix defined on this input has a perfect phylogeny, and any optimal solution to MRP will be a refined compatibility supertree. Furthermore, the MRP problem is NP-hard.*

Proof Under the assumption that the source trees are compatible, a compatibility supertree T exists. Then, every column in the MRP matrix (which corresponds to a bipartition in some source tree) will be compatible with T in the sense that the ?s can be replaced by 0s and 1s to correspond to some edge in T. In other words, the columns in the MRP matrix are compatible partial binary characters (where "partial binary characters" are characters with 0s, 1s, and ?s, or equivalently bipartitions on subsets of the taxon set), and T is a perfect phylogeny for the MRP matrix (see Section 4.1). Now suppose T' is an optimal solution to MRP. Then T is also a perfect phylogeny for the input matrix, and so is a refined compatibility supertree. To establish that MRP is NP-hard, we reduce from partial binary character compatibility, which is NP-complete (Bodlaender et al., 1992; Steel, 1992). For each partial binary character (i.e., a bipartition $A|B$ on a subset $S_0 \subseteq S$), we define a tree on leafset S_0 and with one edge defining the same bipartition. The profile of trees defined in this way has a compatibility supertree if and only if the set of partial binary characters is compatible. Hence, MRP is NP-hard. □

7.6 Matrix Representation with Likelihood

Matrix Representation with Likelihood (MRL) was introduced in Nguyen et al. (2012). MRL is nearly identical to the MRP problem, but instead of seeking the maximum parsimony tree for the MRP matrix, the objective is a maximum likelihood tree for the MRP matrix, where maximum likelihood is computed under the CFN model. The CFN model is the symmetric binary model of sequence evolution where the state at the root is equiprobable to be 0 or 1; therefore, the choice of 0 or 1 is randomized to ensure that the state at the root is selected randomly. As with the MRP supertree methods, MRL treats ?s as missing data.

Nguyen et al. (2012) compared leading heuristics for MRP and MRL on simulated datasets, and found that MRL supertrees were generally at least as accurate as MRP supertrees. They also saw that MRL scores were more closely correlated with topological accuracy than MRP scores. Hence, MRL is competitive with MRP, and one of the promising approaches to supertree estimation.

7.7 Quartet-based Supertrees

Quartet amalgamation methods construct trees by combining quartet trees, and are useful in many contexts. Examples of quartet amalgamation methods include the Quartet Puzzling algorithm (Strimmer and von Haeseler, 1996), Weight Optimization (Ranwez and Gascuel, 2001), Quartet Joining (Xin et al., 2007), Short Quartet Puzzling (Snir et al., 2008), Quartets MaxCut (Snir and Rao, 2010), and QFM (Reaz et al., 2014). While some of these methods require a quartet tree on every set of four leaves, others (e.g., Quartets MaxCut) can be used on arbitrary inputs of quartet trees. Therefore, one approach to constructing supertrees is to encode each source tree as a set of quartet trees, and then use one of the appropriate quartet amalgamation methods to assemble a tree from the set of quartet trees.

7.7.1 Maximum Quartet Support Supertrees

A natural optimization problem is to find the supertree that agrees with the largest number of quartet trees defined by the source trees (Wilkinson et al., 2005). We define this problem, as follows. Recall that $Q(t)$ denotes the set of homeomorphic quartet trees induced by a tree t, and that this set can contain star trees (quartet trees that do not have any internal edges) if t is unresolved. Suppose t is a source tree and T is a supertree, and so the leafset of t is a subset of the leafset of T. We define the quartet support of t for T to be $|Q(t) \cap Q(T)|$. Then, the quartet support of a profile $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ for the tree T is $\sum_{i=1}^{k} |Q(t_i) \cap Q(T)|$. The Maximum Quartet Support Supertree (MQS) T_{MQS} is the tree

$$T_{MQS} = arg\,\underset{T}{max} \sum_{i=1}^{k} |Q(t_i) \cap Q(T)|.$$

This problem can be rephrased in terms of quartet distance, in the obvious way. Define the quartet distance of t to T to be the number of four-taxon subsets of $\mathscr{L}(t)$ on which T and t induce different quartet trees, and denote this by $d(T,t)$. Given a set $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$ of gene trees and a candidate species tree T, we define $d(T, \mathscr{T}) = \sum_{i=1}^{k} d(T, t_i)$. Then the tree on leafset $\cup_i \mathscr{L}(t_i)$ that has the smallest total quartet distance to the trees in $\mathscr{T}$ is the Quartet Median Tree, i.e.,

$$T_{median} = arg\,\underset{T}{min}\{d(T, \mathscr{T})\}.$$

It is easy to see that any MQS is also a Quartet Median Tree.

Finding the MQS is NP-hard, even if there is a quartet tree on every four leaves (Jiang et al., 2001). However, an easy proof when the set of quartet trees can be incomplete (and so not include a tree on every four leaves) is as follows. A compatibility supertree (if it exists) would be an optimal solution to the MQS problem, and so by Theorem 7.6, the MQS problem is NP-hard.

Since finding a maximum number of the possible quartet trees is NP-hard, approximation algorithms have also been developed; for example, Jiang et al. (2001) developed a polynomial time approximation scheme (PTAS) for the case where the set contains a tree on every quartet.

An interesting variant allows the input quartet trees to be equipped with arbitrary positive weights $w(q)$ and seeks the tree T with the maximum total quartet weight (i.e., maximizing $\sum_{q \in Q(T)} w(q)$). Since the unweighted version is NP-hard, the weighted version problem is also NP-hard. Heuristics for maximum weighted quartet support have been proposed, including Weighted Quartets MaxCut (Avni et al., 2015).

7.7.2 *Split-Constrained Quartet Support Supertrees*

One way of addressing NP-hard problems is to constrain the search space, and then solve the problem exactly within that constrained space. This kind of approach has been used very effectively with quartet-based optimization problems, and is the subject of this section. Specifically, we define the *Split-Constrained Quartet Support Supertree problem* as follows:

- Input: Function w that assigns non-negative weights to all binary quartet trees on a set S, and set X of bipartitions of S.
- Output: Unrooted binary tree T such that $Q(T)$ has maximum total weight among all unrooted binary trees T' that satisfy $C(T') \subseteq X$. In other words, $T = arg\,max_{T'}$

$\sum_{t \in Q(T')} w(t)$, where the max is taken among all unrooted binary trees T' that draw their bipartitions from X.

Note that when X is the set of all possible bipartitions on S, there is no constraint on the set of trees T that can be considered, and so the Split-Constrained Quartet Support Supertree problem is just the Maximum Quartet Compatibility problem. However, for other settings for X, the constraint on the set of possible trees can be very substantial.

The Split-Constrained Quartet Support problem can be solved in time that is polynomial in the number of species, source trees, and size of $|X|$, using dynamic programming (Bryant and Steel, 2001). To do this, we first define a nearly identical problem (the Clade-Constrained Quartet Support problem) where we seek a rooted tree instead of an unrooted tree, and we constrain the set of clades the rooted tree can have.

Because we are constraining the output tree to be binary, we do not need to consider unresolved (i.e., non-binary) quartet trees. Hence, we will say that quartet tree $ab|cd$ supports the unrooted tree T if $ab|cd \in Q(T)$. Then, since every rooted tree T defines an unrooted tree T_u (see Definition 2.12), we will say that a quartet tree $ab|cd$ supports the rooted tree T if $ab|cd$ supports T_u.

The input to the Clade-Constrained Quartet Support problem is a set C of subsets of S, and a non-negative function w on the set of all possible binary quartet trees of S. The objective is a rooted binary tree T such that $Clades(T) \subseteq C$ and T has maximum quartet support among all rooted binary trees that satisfy this constraint. In other words, letting $R_{C,S}$ denote the set of rooted binary trees on taxon set S that draw their clades from C, then the Clade-Constrained Quartet Support tree is

$$T_{CCQS} = arg \max_{T \in R_{C,S}} \sum_{t \in Q(T)} w(t).$$

We can construct a rooted binary tree T that has the best Clade-Constrained Quartet Support using dynamic programming. After we find a rooted binary tree with the best Clade-Constrained Quartet Support, we will unroot the rooted tree, thus producing an unrooted binary tree with the best quartet support.

To see how this works, let $\mathscr{T}$ be the set of source trees and X the set of allowed bipartitions given as input to the Split-Constrained Quartet Support Supertree problem. To construct C, we include every half of every bipartition in X, and then we add the singletons (i.e., the elements of S) and the full set S. Note that for every bipartition $A|B$ in X, the set C will contain both A and B.

If $R_{C,S} = \emptyset$, then there are no feasible solutions to the Clade-Constrained Quartet Support problem, and also no feasible solutions to the Split-Constrained Quartet Support problem, and we return *Fail*. Otherwise, suppose $T \in R_{C,S}$ and let $t \in Q(T_u)$. Let v be the (unique) *lowest* node v in T (i.e., the node that is furthest from the root of T) for which at least three of t's leaves are below v. We will say that the *quartet tree* t *is mapped to the node* v *with this property*.

Let v's children be v_1 and v_2. Since $T \in R_{C,S}$, the sets of leaves below v, v_1, and v_2 are all elements of C. Furthermore, since the set A of leaves below v is an element of C, then if v is not the root of T, then the set $S \setminus A$ is also in C. In other words, the node v defines a *tripartition* of the leafset S into three sets of allowed clades, $(A_1, A_2, S \setminus A)$, where A_i is the set of leaves below v_i for $i = 1, 2$. The root also has an associated tripartition, where the third component is the empty set ($\emptyset$). Just as we talked about mapping quartet trees to nodes, we can say that a quartet tree is mapped to the tripartition associated to the node. We will say that a quartet tree is *induced by* the tripartition it is associated to. Furthermore, the quartet tree $ab|cd$ maps to (U, V, W) if and only if the following properties hold:

1. $ab|cd$ is induced by the tripartition (U, V, W).
2. If the set $\{a, b, c, d\}$ does not split 2–2 among U and V, then two of its leaves are in U and one is in V, or vice versa.

Thus, we can determine if $ab|cd$ maps to a given tripartition just by looking at the tripartition, and we do not need to consider the tree as a whole. Therefore, given any tripartition (U, V, W) of S, we can compute the set of quartet trees that map to the tripartition, and hence the total weight of all quartet trees that map to the tripartition. We will denote this by $QS(U, V, W)$. Since every quartet tree that supports a binary rooted tree T is mapped to exactly one node in T, we can write

$$Support(T) = \sum_{v \in V(T)} QS(U_v, V_v, W_v),$$

where v defines tripartition (U_v, V_v, W_v). We generalize this by letting $score(T, v)$ be the total quartet support at all the nodes in T at or below v. Then

$$score(T, v) = score(T, v_1) + score(T, v_2) + QS(A_1, A_2, A_3),$$

where v_1 and v_2 are the children of v, A_i is the set of leaves below v_i for $i = 1, 2$, and $A_3 = S \setminus (A_1 \cup A_2)$.

We will use these concepts by computing, for every allowed clade A, the best possible quartet support score achievable on any rooted binary tree T on A, which will be the total support contributed by quartet trees that map to nodes in the tree T. Letting $Qscore(A)$ denote this best possible score, we obtain $QScore(A) = 0$ for any clade A where $|A| \leq 2$. Otherwise, we look over all ways of dividing A into two disjoint sets A_1 and A_2 where each A_i is an allowed clade, and we set

$$QScore(A) = QScore(A_1) + QScore(A_2) + QS(A_1, A_2, A_3),$$

where $A_3 = S \setminus A$. This calculation requires that we compute $QScore(A)$ for each clade A in order of increasing size, and that we precompute the $QS(A_1, A_2, A_3)$ values. In other words, we have formulated a dynamic programming solution to the Clade-Constrained Quartet Support problem, which we now present:

Dynamic Programming Algorithm for the Clade-Constrained Support Problem

- Given set X of allowed bipartitions, compute set C of allowed clades, and include the full set S and all the singleton sets.
- Order the set C by cardinality, from smallest to largest, and process them in this order.
- Compute $QS(U,V,W)$ for all tripartitions (U,V,W) where U,V, and W are each non-empty allowed clades, $U \cup V \cup W = S$, and the three sets are pairwise disjoint.
- For clades $A \in C$ where $|A| \leq 2$, set $QScore(A) = 0$. For all larger $A \in C$, compute $QScore(A)$ in order from smallest to largest, setting

$$QScore(A) =$$

$$max\{QScore(A_1) + QScore(A_2) + QS(A_1, A_2, S \setminus A) : A_i \in C, A = A_1 \cup A_2\}.$$

- Return $QScore(S)$, the maximum quartet support of any $T \in R_{C,S}$. To construct the optimal tree, use backtracking through the dynamic programming matrix.

Lemma 7.9 *For all sets* C *of allowed clades and all* $A \in C$, QScore(A) *is the maximum number of quartet trees* t *that can be satisfied by any rooted tree* T *on leafset* A *where at least three of the leaves of* t *are in* A.

Proof The proof follows by induction, after noting that every quartet tree that is induced in a rooted tree T is mapped to exactly one node in T. □

We summarize this discussion with the following theorem.

Theorem 7.10 *(From Bryant and Steel (2001)) The Clade-Constrained Quartet Optimization problem and the Split-Constrained Quartet Optimization problem can be solved exactly in* $O(np + n^4|X| + n^2|X|^2)$ *time, where the input has* p *quartet trees on* n *taxa and* X *is the set of constraints (clades or bipartitions).*

A variant of this constrained search problem has an input set $\mathscr{T}$ of k source trees (each leaf-labeled by a subset of S), and sets the weight of a quartet tree $ab|cd$ to be the number of these source trees that induce $ab|cd$. If the algorithm from Bryant and Steel (2001) were applied directly, the running time would be $O(n^5k + n^4|X| + |X|^2)$. However, an optimal solution to the Split-Constrained Quartet Support Supertree problem for this particular variant can be found in $O(nk|X|^2)$ time, where n is the number of species and k is the number of source trees (Mirarab and Warnow, 2015). This variant of the problem is discussed in greater detail in Chapter 10, due to its relevance to constructing species trees from gene trees in the presence of incomplete lineage sorting.

7.8 The Strict Consensus Merger

7.8.1 Overview of the Strict Consensus Merger

The Strict Consensus Merger (SCM) is a supertree method that is used in the supertree meta-method SuperFine (Swenson et al., 2012b), described in the next section, and

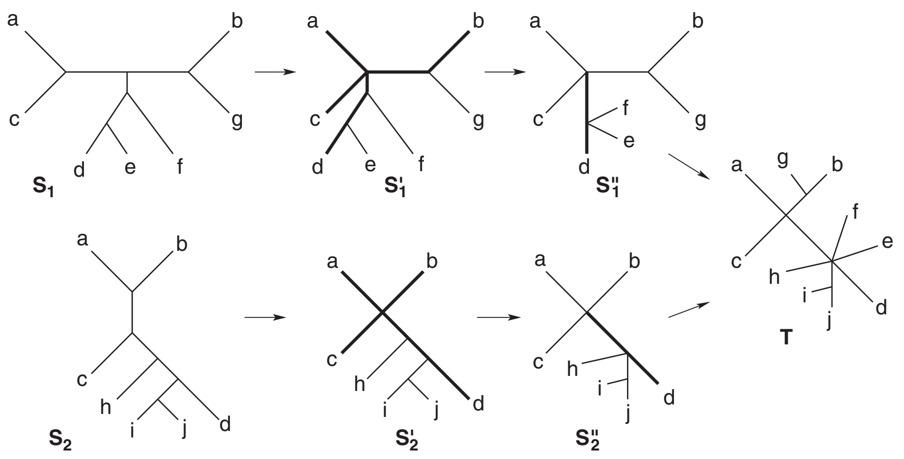


Figure 7.1 (From Swenson et al. (2012b)) Strict consensus Merger (SCM) of two trees S_1 and S_2. Step 1: The two trees both share leafset $B = \{a,b,c,d\}$, but have conflict on this subset. The edges in the two trees that create the conflict are collapsed, producing trees S_1' and S_2'; the backbone (i.e., the strict consensus tree of S_1 and S_2 restricted to B) is shown in bold. Step 2: Both S_1' and S_2' contribute additional taxa to the edge in the backbone leading to the leaf d. The internal edges in the paths corresponding to that edge in S_1' and S_2' are each collapsed, producing trees S_1'' and S_2''; these paths are shown in bold. Finally, the subtrees in S_1'' and S_2'' contributing additional taxa to the edge leading to d are added to the backbone, producing the SCM tree T. Note that the SCM tree is unresolved as the result of both conflict (during Step 1) and a collision (during Step 2).

DACTAL (Nelesen et al., 2012), an iterative divide-and-conquer technique for improving the scalability and accuracy of phylogeny estimation methods, described in Section 11.8. Because of its general utility, we describe the SCM here. For additional details, see Huson et al. (1999a) and Warnow et al. (2001).

The SCM (originally described in Huson et al. (1999a)) is a polynomial time supertree method. The SCM tree is generally highly unresolved, and can in some cases have large polytomies. Hence, unlike most supertree methods, the SCM tree is designed for use as a constraint tree that will need to be subsequently refined into a binary tree.

7.8.2 Computing the Strict Consensus Merger of Two Trees

We begin by describing how the SCM of two trees is performed; see Figure 7.1. The input is two source trees, S_1 and S_2.

- Step 1: We compute B, the common leafset of S_1 and S_2. We then compute the strict consensus tree of $S_1|B$ and $S_2|B$, and refer to this as the "backbone." We modify S_1 and S_2, if necessary, to make them induce the backbone on B; thus, any conflict between $S_1|B$ and $S_2|B$ results in collapsing edges in S_1 and S_2, and introduces at least one polytomy in the SCM tree. After this step completes, we refer to the pair of source trees as S_1' and S_2'.

- Step 2: The taxa that are not in the backbone are now added to it. Each edge in the backbone corresponds to an edge or a path of more than one edge in each of S'_1 and S'_2. We say that S'_i *contributes additional taxa to an edge e in the backbone* if and only if e corresponds to a path of more than one edge in S'_i. When only one of the two trees S'_1 and S'_2 contribute additional taxa to the edge, then those taxa are added directly, maintaining the topological information in the source tree. However, if both trees contribute taxa to the same edge, then this is a "collision," and the additional taxa are added in a more complicated fashion, as we now describe. In the presence of a collision involving edge e, the paths corresponding to edge e are collapsed to a path with two edges in both S'_1 and S'_2, producing trees S''_1 and S''_2. Each S''_i has a single rooted subtree to contribute to the backbone at the edge e. The edge e is subdivided by the addition of a new vertex v_e, and the two rooted subtrees (one from each of S''_1 and S''_2) are attached to vertex v_e, by identifying their roots with v_e. Thus, each collision creates an additional polytomy in the SCM tree.

7.8.3 Computing the Strict Consensus Merger of a Set of Trees

To compute the SCM of a set of trees, an ordering on the set of pairwise mergers must be defined. Different proposals have been made for how to define the ordering, with the objective being that the ordering should maintain as much resolution as possible; thus, collisions should be avoided. One approach that has been used in the SuperFine (Swenson et al., 2012b) supertree method (which uses the SCM in its protocol) is to pick the pair of trees to merge that maximizes the number of shared taxa; however, other approaches can be included.

Given an ordering of pairwise mergers, the SCM of a set of trees proceeds in the obvious way. After each pairwise merger is computed, the set of subset trees is reduced by one, and the process repeats. At the end, a tree is returned that has all the taxa; this is the SCM tree. Hence, there can be more than one SCM tree, depending on the ordering. It is easy to see that the SCM of two trees can be computed in polynomial time, and so the SCM of a set of trees is also polynomial time. We summarize this as follows:

Theorem 7.11 *Let $\mathscr{T}$ be a profile of* k *trees on set* S *of* n *taxa. An SCM of $\mathscr{T}$ can be computed in time that is polynomial in* k *and* n.

7.8.4 Theoretical Properties of the SCM Tree

If the set of trees is compatible, then there will never be a conflict in any pairwise merger; however, it is still possible for a pair of compatible trees to have a collision, which will result in the SCM tree having at least one polytomy. In other words, the SCM of a set of compatible binary trees may not be fully resolved. In particular, the SCM tree is not guaranteed to solve the Unrooted Tree Compatibility problem. This is not surprising, since the

Unrooted Tree Compatibility problem is NP-hard (Theorem 3.11) and hence a polynomial time algorithm cannot be expected to solve it.

On the other hand, if the source trees are compatible and satisfy some additional constraints, then there is at least one ordering (that can be computed from the data) of the source trees that will produce the unique compatibility supertree (see Theorem 11.8). Thus, the SCM method solves the Unrooted Tree Compatibility problem under some conditions.

By design the SCM is very conservative, and so will collapse edges in the presence of any conflict or collision. As a result, the SCM tends to be highly unresolved. Indeed, the topological error in an SCM tree is high – but the error is in the form of false negatives (i.e., missing branches) instead of false positives.

7.9 SuperFine: A Meta-Method to Improve Supertree Methods

7.9.1 Overview of SuperFine

SuperFine (Swenson et al., 2012b) is a general two-step technique that can be used with any supertree method; see Figure 7.2.

- Step 1: The SCM tree is computed on the profile of source trees.
- Step 2: The SCM tree is resolved into a binary tree using the selected supertree method; see Figure 7.3.

Thus, SuperFine is a meta-method that is designed to work with a selected supertree method, and we refer to SuperFine run with supertree method *M* by "SuperFine+M." SuperFine has been tested with MRP, MRL, and QMC (Quartets MaxCut) (Swenson et al., 2011, 2012b; Nguyen et al., 2012), and shown to improve the accuracy and scalability of these base supertree methods.

In Section 7.8, we described the SCM and how the SCM tree is computed. We also noted that the SCM tree is typically highly unresolved as a result of collisions (where two source trees can be combined in more than one way) and conflicts (where two source trees are incompatible). Here we show how the SCM tree is refined into a binary tree, one polytomy at a time, using SuperFine.

7.9.2 SuperFine: Refining Each Polytomy in the SCM Tree

The key technique in SuperFine is to recode the topological information in each source tree using a new but smaller taxon set, and then SuperFine runs its selected supertree method on these recoded source trees. The outcome of this process is a supertree on the new taxon set, which is then used to refine the polytomy. This refinement step is applied to each polytomy in turn.

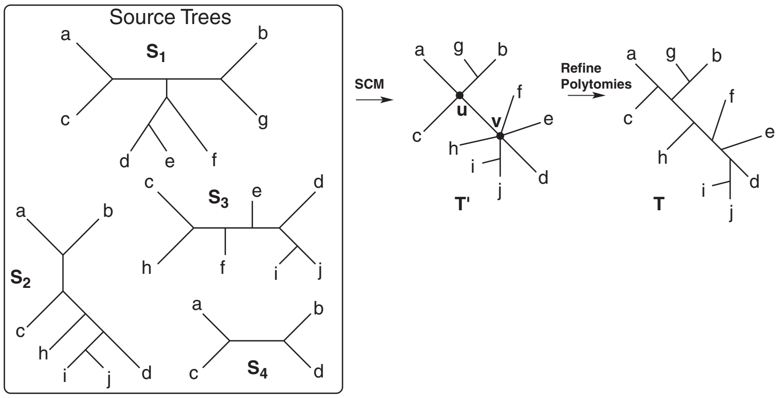
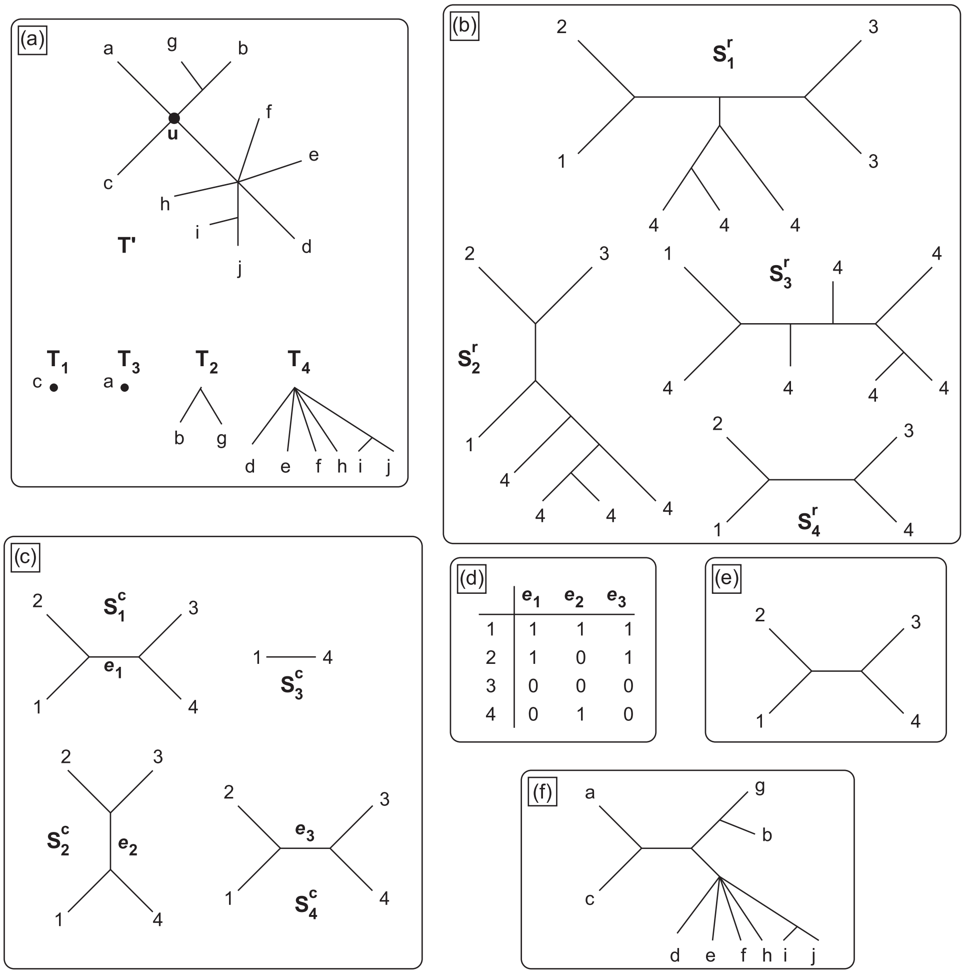


Figure 7.2 (From Swenson et al. (2012b)) Schematic representation of the algorithmic strategy of SuperFine+MRP. Source trees S_1–S_4 are combined pairwise to produce a Strict Consensus Merger (SCM) tree; this process collapses edges, and will only retain internal branches that are compatible with all of the source trees. However, note that it may fail to retain some branches that are compatible with the source trees, as the example shows (for example, $abcefgh|dij$ is not in the SCM tree but is compatible with all the source trees). Each polytomy in the SCM tree is then refined by running a preferred supertree method, such as MRP or MRL, on modified source trees.

Here we describe how this recoding is done. Let v be a polytomy in the SCM tree with degree $d > 3$. First, we relabel every leaf in the SCM tree using labels $\{1,2,\ldots,d\}$, according to which of the d subtrees off the vertex v it belongs to. This produces a relabeling of the leaves of every source tree in $\mathscr{T}$ with $\{1,2,\ldots,d\}$. Now, if two sibling leaves x,y in a source tree $t \in \mathscr{T}$ have the same label L, we label their common neighbor by L and remove both x and y from t. We repeat this process until no two sibling leaves have the same label. As proven in Swenson et al. (2012b), when this process terminates, the source tree t will have at most one leaf of each label. We refer to this process as "recoding the source trees for the polytomy v," and the modified source trees are referred to as "recoded source trees." Note that each recoded source tree has at most d leaves. Thus, when the polytomy degree is relatively small (compared to the original dataset), this produces recoded source trees that are much smaller than the original source trees.

To refine the SCM tree at the polytomy v, we apply the selected supertree method to the recoded source trees, and obtain a tree $T(v)$ on leafset $\{1,2,\ldots,d\}$. We then use $T(v)$ to refine the SCM tree at node v. Note that when $d = deg(v)$ is sufficiently small, this step can be quite fast. Furthermore, the refinements around the different polytomies can be performed in parallel (or sequentially, but in any order that is desired), since the different refinements do not impact each other. See Figure 7.3.



	e_1	e_2	e_3
1	1	1	1
2	1	0	1
3	0	0	0
4	0	1	0

Figure 7.3 (From Swenson et al. (2012b)) Schematic representation of the second step of the algorithmic strategy of SuperFine+MRP, in which we refine the SCM tree produced in the first step. The steps here refer to the SCM tree T', polytomy u, and source trees shown in Figure 7.2. (a) The deletion of the polytomy u from the tree T' partitions T' into four rooted trees, T_1, T_2, T_3, and T_4. (b) The leaves in each of the four source trees are relabeled by the index of the tree T_i containing that leaf, producing relabeled source trees S_1^r, S_2^r, S_3^r, and S_4^r. For example, the relabeled version of $S_4 = ac|bd$ is $S_4^r = 12|34$. (c) Each S_i^r is further processed by repeatedly replacing sibling nodes with the same label, until no two siblings have the same label; this results in trees S_1^c, S_2^c, S_3^c, and S_4^c. (d) The MRP matrix is shown for the four source trees, including only the parsimony informative sites; thus, S_3^c does not contribute a parsimony informative site and is excluded. (e) The result of the MRP analysis on the matrix given in (d). (f) The tree resulting from identifying the root of each $T_i, i = 1,2,3,4$, with the node i in the tree from (e).

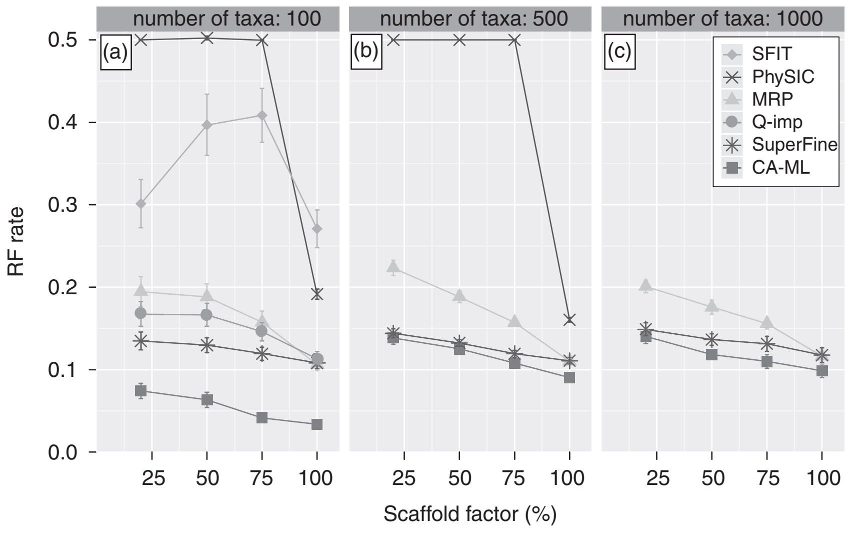


Figure 7.4 (From Swenson et al. (2012b)) Robinson–Foulds (RF) error rates (mean with standard error bars) for supertree methods SFIT, PhySIC, MRP, Q-imputation, SuperFine+MRP, and concatenation using maximum likelihood (CA-ML) for three different taxon sizes, as a function of the scaffold factor. (a) shows results for 100 taxa, (b) shows results for 500 taxa, and (c) shows results for 1000 taxa. Methods not shown in (b) and (c) were unable to complete on these datasets within the allowed timeframe.

7.9.3 Performance in Practice

As shown in Swenson et al. (2011, 2012b) and Nguyen et al. (2012), SuperFine is generally very fast even when run sequentially, and efficient parallel implementations have also been developed (Neves et al., 2012; Neves and Sobral, 2017). Figure 7.4 (from Swenson et al., 2012b) shows a comparison of several supertree methods on estimated source trees, including MRP (the most well-known supertree method), SuperFine using heuristics for MRP as the base supertree method, and concatenation using maximum likelihood (CA-ML). The *x*-axis shows the "scaffold factor," which is the percentage of the leafset that is in the randomly selected "backbone" source tree. Most methods improve in accuracy as the scaffold factor increases. While CA-ML has the best accuracy (i.e., lowest RF topological error rate), the supertree method with the best accuracy is SuperFine+MRP. Furthermore, although all methods could complete on the 100-taxon datasets, fewer could complete on the 500-taxon datasets, and only CA-ML, SuperFine+MRP, and MRP could complete on the 1000-taxon datasets. A comparison of the running times of these methods is shown in Figure 7.5. Note that the difference in running times is very large on the 500- and 1000-taxon datasets, where CA-ML takes more than 1440 minutes, MRP takes 180–240 minutes, and SuperFine takes 90–180 minutes. Studies evaluating SuperFine with heuristics for MRL (Nguyen et al., 2012) and with QMC (Snir and Rao, 2010) have also shown similar

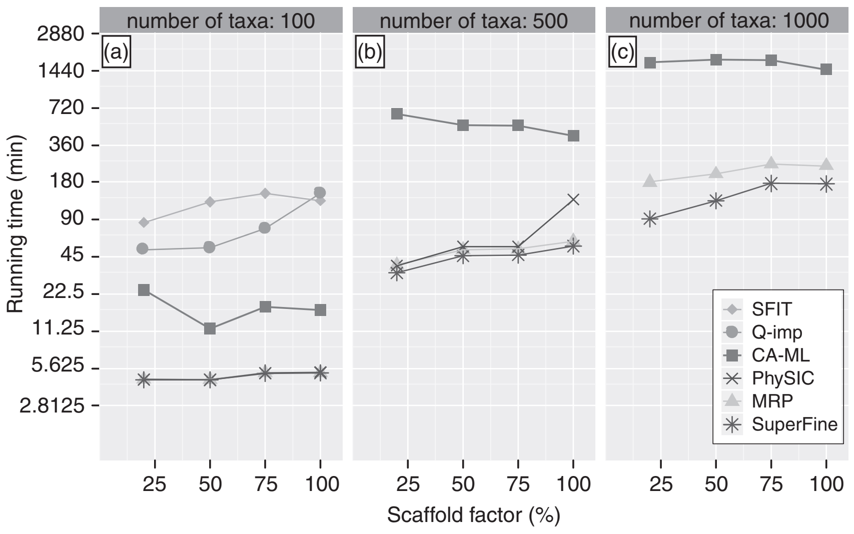


Figure 7.5 (From Swenson et al. (2012b)) Average running times (logarithmic scale), including the time needed to compute source trees, for supertree methods SFIT, Q-imputation, CA-ML, PhySIC, MRP, and SuperFine+MRP for three different taxon sizes, as a function of the scaffold factor. For the supertree methods, running times shown include the time required to generate maximum likelihood source trees using RAxML (Stamatakis, 2006). (a) shows results for 100 taxa, (b) shows results for 500 taxa, and (c) shows results for 1000 taxa. The curves for PhySIC, MRP, and SuperFine+MRP overlap for the 100-taxon datasets. Methods not shown in (b) and (c) were unable to complete on these datasets within the allowed timeframe.

results (Swenson et al., 2011; Nguyen et al., 2012). Thus, SuperFine is a meta-method that uses divide-and-conquer to improve the accuracy and speed of supertree methods. The best supertree analyses come close to the accuracy of a concatenation analysis, and can be much faster.

7.10 Further Reading

Quartet-based supertree methods: Since unrooted trees can be encoded as quartet trees, quartet-based tree construction methods can be used as supertree methods. For example, the QMC method (Snir and Rao, 2010) has been studied as a supertree method in Swenson et al. (2011), and shown to have good accuracy.

Supertrees for rooted source trees: Optimization problems for supertree construction when the input is a set of rooted trees have also been explored. One such approach encodes each rooted source tree as a set of rooted triplet trees (i.e., three-leaf trees), and then seeks a rooted tree that maximizes the number of triplet trees (from the rooted source trees) that agree with it. This is called the **Maximum Triplet Support** problem. Equivalently, this can be formulated as finding a median tree with respect to the triplet distance (Ranwez et al., 2010).

The Maximum Triplet Support problem is NP-hard (Bryant, 1997), and so heuristics have been developed to find good solutions to the problem. The most well known of these heuristics is MinCutSupertree (Semple and Steel, 2000), which the authors describe as "a recursively optimal modification of the algorithm described by Aho et al." In other words, MinCutSupertree is a modification of the ASSU algorithm described in Section 2.9 so that it can be run on incompatible source trees. A modification to the MinCutSupertree method was developed in Snir and Rao (2006), and found to be more accurate than MinCutSupertree. More recent heuristics for this problem include the SuperTriplets method (Ranwez et al., 2010) and Triplets MaxCut (Sevillya et al., 2016).

Distance-based supertree estimation: Another class of supertree methods takes advantage of the fact that the source trees for the supertree problem typically define distance matrices of some type, so that supertree estimation methods can use the distance matrices to infer the final supertree. For example, estimated phylogenetic trees are often based on maximum likelihood, and so come with branch lengths that reflect the expected number of changes on the branch; thus, a maximum likelihood tree defines an additive matrix for its taxon set. The same is true for trees computed using distance-based methods such as neighbor joining or FastME, when the distances are based on statistical sequence evolution models. Even trees computed using maximum parsimony come with branch lengths, although these cannot be interpreted in quite the same way. Thus, all these methods can be used to produce additive matrices, and in some cases these matrices are even ultrametric.

As Willson (2004) observed, the use of these "distance matrices" is a valuable source of information in supertree construction, since they naturally enable the estimation of branch lengths in the supertree, something that is not possible when just using the source trees alone.

Here we describe a classical distance-based supertree problem, **Matrix Representation with Distances** (MRD) from Lapointe and Cucumel (1997). The input is a set of k dissimilarity matrices, each on a subset of the taxon set S. We seek an additive matrix on the full species set that *minimizes the total distance to the input matrices*; in other words, we seek a median tree with respect to some way of measuring distances between matrices. Common ways of measuring distances use the L_∞, L_1, and L_2 norms, but other norms can also be used. Note also that we require that the output be an additive matrix, so that we can use it to define the supertree topology and branch lengths. If we wish to compute a rooted supertree with branch lengths reflecting elapsed time, then we will require that the output be an ultrametric matrix.

Distance-based supertree estimation is not new (e.g., Lapointe and Cucumel, 1997; Lapointe et al., 2003; Willson, 2004; Criscuolo et al., 2006), but none of the current methods has become widely used. However, Build-with-Distances (Willson, 2004) came close to MRP and in some cases was more accurate (Brinkmeyer et al., 2011), leading the authors to conclude that distance-based supertree estimation might have the potential to replace MRP.

As discussed in Chapter 5, there is a substantial literature on related problems, where the input is a single dissimilarity matrix M and the objective is an additive matrix or an ultrametric matrix that is optimally close to M, under some metric between distance matrices. Some of these problems are solvable in polynomial time, others are NP-hard but can be approximated, and some are hard to approximate; see Ailon and Charikar (2005); Fakcharoenphol et al. (2003); Agarwala et al. (1998) for an entry to this literature.

Statistical properties of supertree methods: The supertree methods presented so far are heuristics for NP-hard optimization problems. As such, understanding their statistical properties is quite challenging, since characterizing the conditions under which the heuristics are guaranteed to find globally optimal solutions to their criteria is difficult. However, suppose that each of the optimization problems could be solved exactly – i.e., suppose that globally optimal solutions could be found. Could we say anything about the probability of recovering the true supertree? To answer this question, we pose this as a statistical estimation problem in which the (unknown) true supertree is used to generate a sequence of source trees under some random process.

Suppose that the source trees are on subsets of the full taxon set, and are generated by a random process defined by a model species tree on the full set of taxa. For example, the model could assume that a source tree is the tree induced by the model species tree on a random subset of the taxon set. Under this model of source tree generation, all the source trees are compatible, and the true species tree is a compatibility supertree. Since the random process generates all subtrees with non-zero probability, the model species tree is identifiable (i.e., the model species tree has non-zero probability, and no other tree on the full taxon set has non-zero probability). Furthermore, any method that is guaranteed to return a compatibility supertree for the input set is statistically consistent under this model. Thus, exact solutions for many supertree optimization problems (e.g., MRP, RFS, and MQS) will be statistically consistent methods for species tree estimation under this model.

When the source trees are estimated species trees, then species tree estimation error is part of the generative model. Similarly, when the source trees are gene trees, then gene tree heterogeneity due to biological factors such as incomplete lineage sorting and gene duplication and loss is also part of the generative model. Finding a supertree that maximizes the probability of generating the observed source trees is the **Maximum Likelihood Supertree problem**, proposed in Steel and Rodrigo (2008). Several variants of the generative model have been proposed, and maximum likelihood and Bayesian methods have been developed under these models (Ronquist et al., 2004; Steel and Rodrigo, 2008; Bryant and Steel, 2009; Cotton and Wilkinson, 2009; De Oliveira Martins et al., 2016).

Under some conditions, the RFS problem will provide a good solution to the maximum likelihood supertree under one of the exponential models described in Steel and Rodrigo (2008) (see discussion in Bryant and Steel (2009)). Therefore, heuristics for the RFS discussed in Section 7.4.2 (e.g., PluMiST (Kupczok, 2011), RFS (Bansal et al., 2010), MulRF (Chaudhary, 2015), and FastRFS (Vachaspati and Warnow, 2016)) can be used as heuristics for the Maximum Likelihood Supertree problem.

7.11 Review Questions

1. Define the MRP problem (what is the input and what is the output?).
2. Explain how to write down the MRP matrix.
3. What is the difference between the MRP and MRL optimization problems?
4. Suppose you have a set of 100 binary trees, each tree has ten species, and the total number of species is 500. How many rows and columns are in the MRP matrix?
5. Define the Compatibility Supertree problem (what is the input and what is the output?).
6. Suppose that $\mathscr{T}$ is a set of source trees and it has a compatibility supertree T. Now suppose that T' is the solution to the Robinson–Foulds Supertree problem given input $\mathscr{T}$. Must T' also be a compatibility supertree for $\mathscr{T}$?
7. Define the Split-Constrained Quartet Support Supertree problem (what is the input and what is the output?).
8. Let $T = ((a,f),(b,(c,(d,e))))$ and $T' = (g,(a,(c,(b,(h,d)))))$. Let T^* be the SCM of T and T', and let T^{**} be the homeomorphic subtree of T^* restricted to the leafset of T. Report the RF distance between T^{**} and T.

7.12 Homework Problems

1. Suppose X is a set of compatible unrooted trees on different sets of leaves. What can you say about the solution space to MRP on input X?
2. Suppose you have 1000 trees, each with 100 leaves, and 5000 taxa overall. How big is the MRP matrix?
3. Let T_1 and T_2 be binary trees with leafset S, and let $\mathscr{T}$ be a set of source trees with leaves drawn from S, with $S = \cup_{t \in \mathscr{T}} C(t)$. Let $X = C(T_1) \cup C(T_2)$. Let T be an optimal solution to the Split-Constrained Quartet Support Supertree problem given the set $\mathscr{T}$ of source trees and constraint set X. Must T be one of T_1 or T_2? If so prove it, and otherwise give a counterexample.
4. Prove Lemma 7.2.
5. Under the assumption that the input trees are fully resolved (i.e., binary), is it always the case that every optimal solution to the Maximum Quartet Support Supertree problem is fully resolved?
6. Prove that the Quartet Median Tree and the Maximum Quartet Support Supertree are the same.
7. Recall that the Split-Constrained Quartet Support Supertree problem requires that the output tree be binary. Consider the relaxed version of this problem where the supertree is not required to be binary, and in fact can have unbounded degree. Would the problem still be solvable in polynomial time?