Chapter 15

UNROOTED SUPERTREES
Limitations, traps, and phylogenetic patchworks

Sebastian Böcker

Abstract: Whereas biologists might think of rooted trees as the natural, or even the only, way to display phylogenetic relationships, this is not the case for a mathematician, to whom rooted and unrooted trees are graph-theoretical constructions that can be transformed easily into one another. An unrooted tree contains the same information as its rooted counterpart with the single exception that it does not tell you where the "evolutionary process" started. Rooting a tree is often more of an art than a science, and a pressing problem in systematic biology is precisely the exact placement of a root. In addition, many phylogenetic algorithms in fact output unrooted trees that are rooted (artificially) in a subsequent step.

From this, it is clear that finding an unrooted supertree or parent tree is of the same interest as it is for the rooted case. But, whereas a single unrooted tree can always be transformed into a rooted tree carrying the same information, this is no longer the case for collections of unrooted trees. Hence, the supertree problem for rooted trees is a special case of that for unrooted trees. As is often the case, this means that many things that can be done with rooted trees (the special case) are no longer valid for unrooted trees (the general case). In fact, the smallest possible example of a collection of unrooted trees that cannot be transformed into a collection of rooted trees is already sufficient to demonstrate that, unfortunately, many convenient features of the rooted supertree problem do not carry over to the unrooted supertree problem.

On the positive side, if the set of input trees fulfills some minimality criterion, then there exists a simple set of conditions to check whether there is exactly one parent tree for this collection. In addition, the unique parent tree, should one exist, can be constructed quickly because the set of input trees always shows a certain "patchwork" structure.

Keywords: parent trees; patchworks; quartet trees; unrooted supertrees;

Bininda-Emonds, O. R. P. (ed.) Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life, pp. 331–351. Computational Biology, volume 3 (Dress, A., series ed.).
© 2004 Kluwer Academic Publishers. Printed in the Netherlands.

1. Introduction

In the following, I will assume that all trees are unrooted unless stated otherwise. The supertree problem — that is, combining a collection of leaf-labeled trees with overlapping sets of labels (taxa) into a single "best" leaf-labeled tree — has been studied for some time (Gordon, 1986; Purvis, 1995; Sanderson *et al.*, 1998). Supertree construction methods for unrooted trees include DISK-COVERING (Huson *et al.*, 1999a, b) and dyadic closure-based approaches (Bryant and Steel, 1995; Böcker *et al.*, 2000), to name just a few. In the following, I will limit my attention to the problem of finding a "parent tree" of the input collection: given a collection $\mathcal{F}$ of leaf-labeled trees with generally distinct, although not necessarily disjoint label sets, we want to *amalgamate* these trees into one leaf-labeled parent tree T so that all trees in $\mathcal{F}$ are "induced" subtrees of T (see Section 2 for the distinction between supertrees and parent trees). Hence, we want to know whether such a parent tree T exists at all and, if so, whether it is determined uniquely by $\mathcal{F}$. Many supertree methods will try to amalgamate input trees even if no such parent tree exists. But if a unique parent tree does exist for the input collection, all reasonable supertree methods should return this parent tree.

There are several (computational) problems when trying to construct an unrooted parent tree or when determining if such a parent tree is unique:

- There exist certain limitations that apply to *all* unrooted supertree methods, and certain desirable characteristics for such supertree methods cannot be comprised in a single supertree method (see Section 3).
- The problem of determining whether there exists *at least one* parent tree of $\mathcal{F}$ is provably difficult; that is, it is NP-complete (Steel, 1992).
- The complexity of determining whether some (known!) tree is the *unique* parent tree of a given collection is still unknown.
- The problem of determining whether some input set of trees $\mathcal{F}$ contains an excess-free subset (see Section 5) that has a unique parent tree is also NP-complete (Böcker *et al.*, 2000).
- Finally, there exist certain collections $\mathcal{F}$ of trees that have exponentially many parent trees in the number of trees in $\mathcal{F}$, as well as in the number of leaves of $\mathcal{F}$. Of course, this would be a trivial result if $\mathcal{F}$ had a highly unresolved parent tree because all possible refinements of this tree would also be parent trees of $\mathcal{F}$. What makes the construction presented in Section 4 more compelling is that our (exponentially large) collection of parent trees will consist solely of binary trees; that is, none of these trees will have any refinement. The latter poses a threat to all supertree heuristics trying to circumvent the

runtime constraint by using a (more-or-less) intriguing construction recipe. Such heuristics might return just *one* binary parent tree T to the user that supports some hypothesis, concealing the fact that there exist exponentially many such parent trees that possibly contradict the same hypothesis.

But for certain collections of input trees, we can do better. If the set of input trees fulfills some minimality criterion, then there exists a simple set of conditions to check whether there is *exactly one* parent tree for this collection. In addition, the unique parent tree can be constructed in quadratic runtime. Here, the set of input trees always shows a certain "patchwork" structure that allows us to merge only *two* trees at a time. Although the result itself appears to be rather simple, its proof is cumbersome and lengthy. I hope that by solving the case fulfilling the minimality criterion, we can find a more efficient algorithm for a more general class of tree collections.

2. Definitions

Following Böcker *et al.* (2000), I begin by introducing some terminology.

- Given a tree T, a *leaf* is a vertex of T of degree one. Both a vertex that is not a leaf, as well as an edge that is not incident with a leaf are called *interior*. In the following, I assume that all interior vertices of T are of degree at least three, and that there is at least one interior vertex. Such a tree T is also called a *phylogenetic* tree.[1] I will use the terms "leaves", "leaf labels", and "taxa" interchangeably, depending on the context. If all of the interior vertices have degree three, the tree is said to be a *binary* (phylogenetic) tree.
- For a tree $T = (V, E)$, let $\mathcal{L}(T) \subseteq V$ denote the set of leaf labels of T, and for a collection $\mathcal{F}$ of such trees, let $\mathcal{L}(\mathcal{F})$ denote the union $\cup_{T \in \mathcal{F}} \mathcal{L}(T)$. Recall that the number of interior edges of T never exceeds $|\mathcal{L}(T)| - 3$, and equality holds if and only if T is binary (see, for instance, Proposition 2.1.3 of Semple and Steel, 2003). Two trees $T = (V, E)$ and $T' = (V', E')$ are *isomorphic* if $\mathcal{L}(T) = \mathcal{L}(T')$, and there exists a bijection $\psi : V \to V'$ that, restricted to $\mathcal{L}(T)$, is the identity, and that induces a bijection of $E \to E'$.
- Given a tree T and a subset $L \subseteq \mathcal{L}(T)$, I denote by $T|_L$ the phylogenetic tree obtained from the smallest connected subgraph of T containing (the leaves labeled by) L by making this subgraph

[1] This definition differs from the one given in Semple and Steel (2003), but one can see easily that both definitions are in fact equivalent.

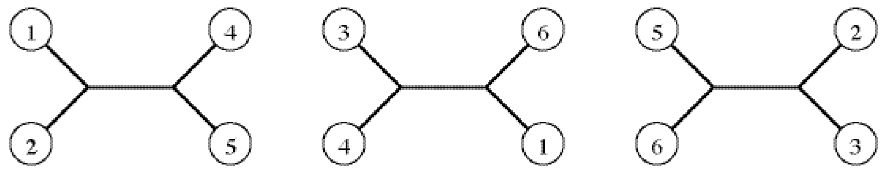

Figure 1. The three quartet trees of Example 1.

homeomorphically irreducible (i.e., by suppressing all degree-two vertices). I refer to $T|_L$ as an *induced subtree* of T and, more specifically, as the subtree of T induced by L. Note that $T|_L$ is binary whenever T is, and that $(T|_{L'})|_L$ is (isomorphic to) $T|_L$ for any tree T and sets $L \subseteq L' \subseteq \mathcal{L}(T)$.

- Given two trees T, T' with $\mathcal{L}(T) = \mathcal{L}(T')$, I write $T \leq T'$ if, up to a label-preserving isomorphism, T can be obtained from T' by contracting some interior edges of T'.
- Suppose that $\mathcal{F} := \{T_1, \ldots, T_k\}$ is a collection of trees. A tree T *displays* $\mathcal{F}$ if $T_i \leq T|_{\mathcal{L}(T_i)}$ holds for all $i = 1, \ldots, k$. I say also that T displays T' in case T displays $\{T'\}$.
- If T displays $\mathcal{F}$ and, in addition, $\mathcal{L}(\mathcal{F}) = \mathcal{L}(T)$ holds, then T is called a *parent tree* of $\mathcal{F}$.[2] In this case, we say also that $\mathcal{F}$ can be *amalgamated* into the parent tree T. If this parent tree T is unique (up to isomorphisms), then I say that $\mathcal{F}$ *defines* T. Note that if $\mathcal{F}$ defines T, then T is necessarily binary.
- A *quartet tree* is a binary tree T with $|\mathcal{L}(T)| = 4$. I write $xy|wz$ to denote the quartet tree that has leaves labeled x, y separated from leaves labeled w, z by its unique interior edge. For example, the quartet tree $12|45$ is depicted on the left in Figure 1.

3. Limitations of unrooted supertree methods

I start this section with a simple example of a collection of trees that does not have an equivalent in the setting of rooted trees. This example is smallest possible with this property: it contains only three quartet trees on six taxa. Clearly, all input collections consisting of two trees will either have at least one leaf common to all (two) trees of the input collection, or the two trees will not share a single leaf and construction of a supertree is not reasonable. I

[2] Such trees are called supertrees in mathematical literature, but here I want to differentiate between the output of a supertree method and the mathematically rigid concept of "parent trees."

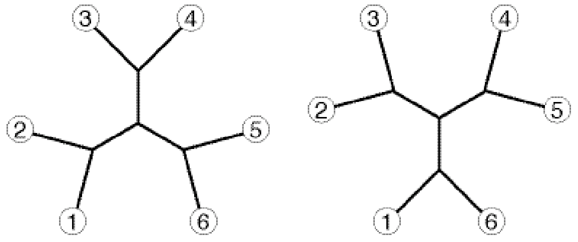

Figure 2. The parent trees T_+ (left) and T_- (right) of Example 1.

will make use of Example 1 throughout this chapter to illustrate several limitations for the unrooted supertree problem.

Example 1. Consider the collection of quartet trees

(1) $$\mathcal{Q} := \{12\,|\,45,\ 34\,|\,61,\ 56\,|\,23\}$$

with leaf set $\mathcal{L}(\mathcal{Q}) = \{1, \ldots, 6\}$ as displayed in Figure 1. Note that any two trees in this collection have exactly two leaves in common, and that no leaf is present in *all* trees of the collection. The key observation is that there exist (up to isomorphisms) exactly two parent trees for this collection of trees as depicted in Figure 2 (see Böcker *et al.*, 1999). I will denote the left parent tree in Figure 2 by T_+, the right parent tree by T_-.

Following the reasoning of Steel *et al.* (2000), I will now collect several properties that should be achieved (simultaneously) by all supertree methods for unrooted trees (see also Wilkinson *et al.*, 2004). Afterwards, I will show, using the above example, that no supertree method exists that can satisfy all the stated properties simultaneously.

First, it is clearly desirable that a supertree method should not rely on the ordering of the input trees. In fact, I have only talked about collections (sets) of input trees up to now that are unordered by definition. As long as we have equal confidence in all the input trees, changing the order in which we present these input trees to the supertree method should not change the output of the method.

Second, renaming the taxa should not change the output of the method. That is, if we replace a leaf label in all our input trees, then the output of the supertree method should be the old supertree, except that the new label also replaces the old label. In fact, this condition consists of two parts. First, no supertree method should force the output of a unique supertree by, say, lexicographically ordering the input taxa before applying the true supertree construction. Otherwise, renaming a taxon from "cat" to "Felidae" could change the output of the method. Second, and more compelling, exchanging the names of two taxa in our input collection of trees should return the old supertree, except that the two taxa are also exchanged. As stated above, the supertree method cannot pre-sort the input taxa to circumvent this requirement.

Third, if the input collection has at least one parent tree, then the supertree method should return a parent tree of the collection. So, if the input trees do fit together, then the supertree method should select one of the parent trees that achieves this.

I will now show that no supertree method for unrooted trees exists that satisfies these three conditions at the same time (Proposition 1 in Steel *et al.*, 2000). Suppose our supertree method satisfies the first and second conditions above. We will see that the third condition must fail necessarily for certain input collections. To this end, consider again the collection of input trees from Example 1. Recall that this collection has exactly two parent trees (Figure 2). Suppose that our supertree method outputs the first parent tree, T_+. If we exchange taxa 2 and 6, and also taxa 3 and 5 of our input collection, the collection becomes $\{16|43, 54|21, 32|65\}$, which is exactly the same input collection as before because the "changed order" of elements is of no relevance. By the first condition, the method should therefore output the same parent tree T_+. But, by the second condition, the method should output the tree T_+, where leaves 2 and 6, as well as 3 and 5 are exchanged, and this is in fact the other parent tree, T_-! The same holds true if the supertree method would output T_-. Thus, if the first *and* the second conditions hold, the output can be neither T_+ nor T_-. But this means that the third condition is necessarily violated because these are the only parent trees of the input collection.

We could abandon the third condition and ask the supertree method to return *all* parent trees in case more than one exists. This might not be desirable from a phylogenetic standpoint, but it would at least allow the supertree method to return a sensible output in case there is more than one parent tree. But, besides the possible biological concerns over such an output, I will show in the next section that there might be exponentially many parent trees for certain collections of input trees. For example, for an input collection of 40 quartet trees on a set of 43 taxa, there can be as many as 1024 binary parent trees. In general, we could apply a consensus method to the set of parent trees of our input collection, but it is not clear how to carry out such calculations in a reasonable time.

Note that as soon as all the input trees have a leaf in common, we can transform the whole collection into rooted trees and use a supertree method to finally obtain an unrooted supertree by reattaching the artificial root leaf. In case a unique parent tree of such an input collection exists, this parent tree can be found and its uniqueness can be tested quickly. But, if there is more than one choice for our pseudo-root, choosing different pseudo-roots might lead to different (rooted and unrooted) supertrees, violating the second condition we wanted our supertree method to fulfill.

4. Exponentially many parent trees

Clearly, as a result of the formal definition of "parent trees" I have introduced above, the existence of (super-)exponentially many parent trees is not always a relevant problem. This is because we require a parent tree to "include all the information" of the input set of trees, but allow it to include additional information that is not supported by the input trees. As an example, define the collection of quartet trees

$$\mathcal{F}_* := \{12 | jk \mid 3 \leq j < k \leq n\}$$

for some integer $n \geq 4$ that has leaf set $\mathcal{L}(\mathcal{F}_*) = \{1, \ldots, n\}$. All such input trees separate leaves 1 and 2 from any two other leaves. We can see easily that $(2n - 7)!! := 1 \cdot 3 \cdot 5 \cdots (2n - 7)$ many binary parent trees of F_* exist and, therefore, also super-exponentially many parent trees in total. The binary parent trees are exactly those binary trees with leaf set $\{1, \ldots, n\}$ having an edge that separates leaves labeled 1 and 2 from leaves labeled $3, \ldots, n$, and $n - 4$ arbitrary other interior edges. But, there is exactly *one* parent tree of $\mathcal{F}_*$ that is minimal with respect to $\leq$, and this tree (i.e., the tree with one interior edge separating leaves 1 and 2 from all other leaves) is the one every practitioner would probably be interested in. In fact, this tree is the strict consensus (see next section) of all possible parent trees of the input tree collection. Note that the production of so-called "novel clades" in supertrees has been found to be highly undesirable by several biologists (e.g., Pisani and Wilkinson, 2002; Gatesy and Springer, 2004; Wilkinson *et al*., 2004).

I will now use Example 1 to construct an input collection of trees such that not only do exponentially many parent trees for this input tree collection exist, but also such that *every* such parent tree is necessarily binary. In addition, we will see such that these parent trees share practically no information (i.e., the strict consensus of all these parent trees is almost the star graph). This means that although the input tree collection $\mathcal{F}$ *can* be amalgamated into a single parent tree, doing so would conceal the fact that there are an exponential number of parent trees that, in total, contradict the information yielded by the single parent tree completely. All heuristic supertree methods returning one, or even a few, parent trees would fall into

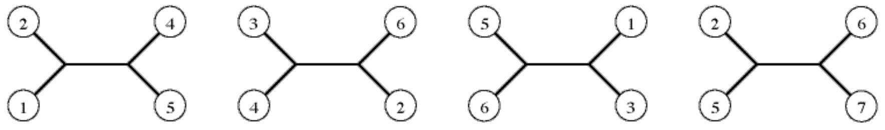

Figure 3. The four quartet trees of Example 2.

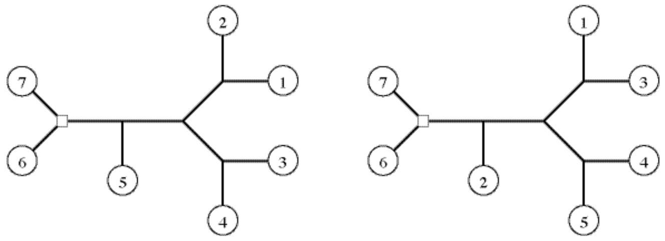

Figure 4. The parent trees T^+ (left) and T^- (right); the median of the leaves 5, 6, and 7 is indicated by a square.

this trap. Let us suppose that such a heuristic returns a (randomly chosen) parent tree T, then the fact that

- the constructed parent tree is binary,
- no less refined tree obtained by contracting edges in T that is a parent tree of the input collection exists, and
- any two parent trees of $\mathcal{F}$ differ strongly; that is, they cannot be transformed into each other by local optimization heuristics like a single branch swapping

might falsely lead the user to believe that the constructed parent tree is unique. By contrast, if a supertree method tries to construct all possible parent trees of the input collection, then it has to output a huge number of trees, and the size of the output itself makes this problem computationally hard. In other words, polynomial runtime in the input size cannot be achieved by such an approach.

Example 2. Consider the collection of quartet trees

$$\mathcal{Q}_* := \{21|45, 34|62, 56|13, 25|67\} \tag{2}$$

with leaf set $\mathcal{L}(\mathcal{Q}_*) = \{1, \ldots, 7\}$ as depicted in Figure 3. It is easy to check that this collection is displayed (up to isomorphism) exactly by the two trees depicted in Figure 4. In fact, I have extended Example 1 by adding a single

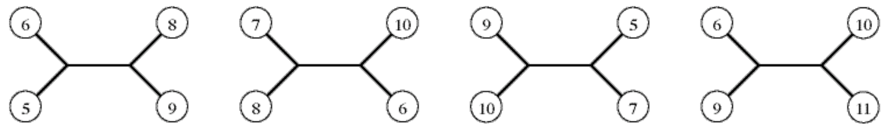

Figure 5. The four quartet trees $\phi_4(T)$ for $T \in \mathcal{Q}_*$.

quartet tree $25 \mid 67$ so that the additional leaf 7 has to be placed together with leaf 6 in both cases. Note also that I have exchanged leaves 1 and 2 to simplify the construction presented below. I will denote the left parent tree in Figure 4 by T^+ and the right parent tree by T^-.

Why did I add an additional leaf and a quartet tree to Example 1? To answer this question, look at the median of the leaves 5, 6, and 7 (i.e., the unique vertex separating these three leaves) in T^+. The median of 5, 6, and 7 is the vertex adjacent to leaves 6 and 7. But this is also true for T^-! This guarantees that if we amalgamate *any* binary tree T to T^+ (or T^-) satisfying the three conditions

1. $5, 6, 7 \in \mathcal{L}(T)$,
2. $1, 2, 3, 4 \notin \mathcal{L}(T)$, and
3. the median of 5, 6, and 7 in T is adjacent to leaf 5,

then there exists a unique parent tree of T and T^+ or of T and T^-, respectively (for a formal proof, see Böcker, 1999). And why did I swap leaves 1 and 2? Look at the median of leaves 1, 2, and 3 in T^+ and T^-. In both cases, this median is adjacent to leaf 1.

We now want to "transpose" trees by adding a natural number to their leaves that are also (labeled by) natural numbers. For example, the quartet tree $12 \mid 34$ can be transposed to the tree $67 \mid 89$ by adding five to its leaves / labels. We can assume without loss of generality that no interior vertex is (labeled by) a natural number. Formally, given a tree $T = (V, E)$, I assume $V \cap \mathbf{N} = \mathcal{L}(T)$, where $\mathbf{N}$ denotes the set of natural numbers. For an arbitrary tree T such that $\mathcal{L}(T) \subseteq \mathbf{N}$, I define the mapping ϕ_j for $j \in \mathbf{N}$ such that $\phi_j(T)$ is the same tree as T, except that every leaf l of T is replaced by the leaf $l + j$ in $\phi_j(T)$. Clearly, $\phi_0(T)$ is (isomorphic to) the input tree T. As an example, Figure 5 depicts the four trees $\phi_4(T)$ for $T \in \mathcal{Q}_*$. Note that both $\phi_4(T^+)$ and $\phi_4(T^-)$ satisfy the conditions stated in the previous paragraph: in both cases, the median of 5, 6, and 7 is adjacent to leaf 5.

Define the transposed collections of quartet trees

$$\mathcal{Q}_k := \{\phi_{4k}(T) \mid T \in \mathcal{Q}_*\}$$

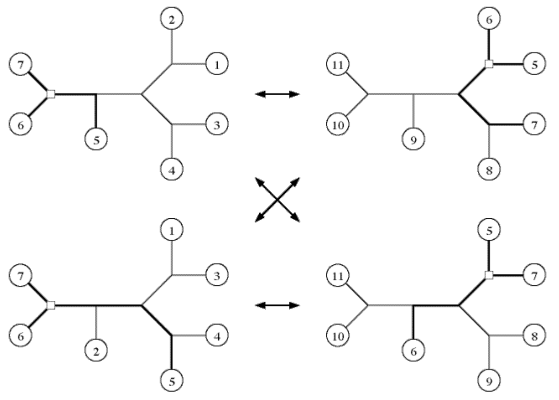

Figure 6. The four trees T_k^i for $k = 0, 1$ (left, right) and $i \in \{+, -\}$ (top, bottom); the median of the leaves 5, 6, and 7 is indicated by a square.

as well as the trees $T_k^+ := \phi_{4k}(T^+)$ and $T_k^- := \phi_{4k}(T^-)$. Clearly, T_k^+ and T_k^- are the unique parent trees of the collection $\mathcal{Q}_k$. As an example, the trees T_0^+, T_0^-, T_1^+, and T_1^- are depicted in Figure 6.

Now consider the collection of quartet trees $\mathcal{Q}_0 \cup \mathcal{Q}_1$ with leaf set $\mathcal{L}(\mathcal{Q}_0 \cup \mathcal{Q}_1) = \{1, \ldots, 11\}$. One can show that any parent tree of this collection is the amalgamation of a parent tree T_0^+ or T_0^- of $\mathcal{Q}_0$, and a parent tree T_1^+ or T_1^- of $\mathcal{Q}_1$. These four parent trees are depicted in Figure 7. This example indicates how the trees T_j^+ and T_j^- can be used as "binary switches" to construct exponentially many parent trees. Eight parent trees exist for the collection $\mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \mathcal{Q}_2$, and these parent trees can be constructed by amalgamating any parent tree of $\mathcal{Q}_0 \cup \mathcal{Q}_1$ in Figure 7 with either T_2^+ or T_2^-. By repeating this process, we see that the collection

$$\mathcal{Q}^*_k := \mathcal{Q}_0 \cup \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k \tag{3}$$

with $4k + 4$ elements and leaf set $\{1, \ldots, 4k + 7\}$ has exactly 2^{k+1} parent trees, of which every one is binary (for a formal proof, see Böcker, 2002).

Finally, we want to know what information is shared by all parent trees of the collection $\mathcal{Q}^*_k$. The *strict consensus* of a collection $\mathcal{F}$ of trees is a tree T^* such that $T^* \leq T$ holds for all $T \in \mathcal{F}$, and T^* is maximal with respect to this condition. Now, what is the strict consensus tree T^* of $\mathcal{Q}^*_k$? This is in fact a unique tree:

Lemma 1. *For $\mathcal{Q}^*_k$ as defined in Equation 3, let $\mathcal{F}_k$ denote the set of parent trees of $\mathcal{Q}^*_k$. The strict consensus of all parent trees in $\mathcal{F}_k$ is the phylogenetic*

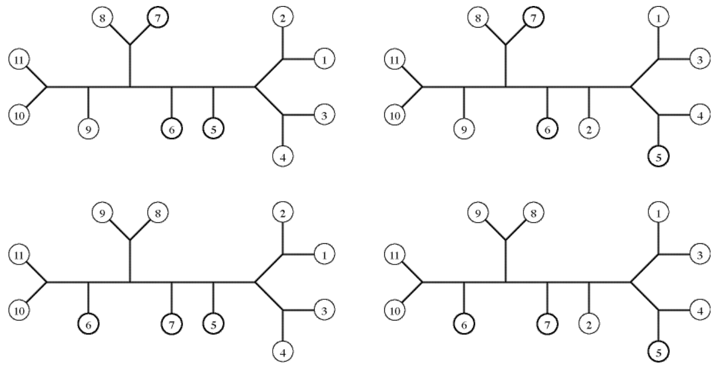

Figure 7. The four parent trees of $\mathcal{Q}_0 \cup \mathcal{Q}_1$.

tree T^*_k with leaf set $\mathcal{L}(T^*_k) = \{1,\ldots, 4k + 7\}$ having a single interior edge separating leaves $\{1,\ldots, 4k + 5\}$ from leaves $\{4k + 6, 4k + 7\}$.

I will omit the proof of this lemma, and just note that induction on k can be used to prove the claim. Furthermore, all consensus methods applied to the set of parent trees $\mathcal{F}_k$ face the following problem: every interior edge of the potential consensus tree separates the leaves into two sets, both of cardinality larger or equal two. Except for the edge separating $4k + 6$, $4k + 7$ from all other leaves, all such interior edges are supported either by none of the input parent trees and should never be included in the output consensus tree, or by exactly half of the input parent trees and contradicted by the other half! Thus, the only reasonable output for any consensus method seems to be the strict consensus tree described above — otherwise, the output would be completely arbitrary. For example, if we choose one of the input parent trees randomly as the output of the consensus tree method, every edge of the output tree would be supported by half of the input trees, and this is as good as it gets.

Note that the input collections $\mathcal{Q}^*_k = \mathcal{Q}_0 \cup \cdots \cup \mathcal{Q}_k$ have no equivalent in the setting of rooted trees. The leaf sets of the subcollections $\mathcal{Q}_j$ are "walking" in the sense that $\mathcal{L}(\mathcal{Q}_j) \cap \mathcal{L}(\mathcal{Q}_{j+1}) = \{4j + 5, 4j + 6, 4j + 7\}$ contains exactly three elements, whereas all other intersections of leaf sets $\mathcal{L}(\mathcal{Q}_j) \cap \mathcal{L}(\mathcal{Q}_k)$ are empty for $|j - k| > 1$. The presented construction cannot work when all leaf sets have at least one element in common.

It is worth mentioning that the constructed collections of quartet trees $\mathcal{Q}^*_k$ are excess-free (see the next section). Thus, $\mathcal{Q}^*_k$ is of minimal cardinality in the sense that all collections of quartet trees of smaller cardinality, but with same leaf set, must have at least one non-binary parent tree.

5. Solution to the parent tree problem for the minimum case

As I mentioned in Section 1, the general problem of finding a parent tree is NP-complete, and the complexity of deciding whether some given tree is the unique parent tree of a collection is unknown. In this section, I finally present a positive result: in those cases where some minimality criterion is satisfied, it is possible to construct a unique parent tree of an input collection of unrooted trees in polynomial time, even when there is no single leaf shared by all trees of the input collection. If such a unique parent tree exists, it is also the "most natural" output of any supertree method.

This section reviews work found in more detail in Böcker (1999), and all theorems and lemmata are drawn from that work unless stated otherwise.

Let $\mathcal{F} = \{T_1, T_2, \ldots, T_k\}$ denote a collection of binary trees. If T is the unique parent tree of the collection $\mathcal{F}$, then we can show that

$$|\mathcal{L}(T)| - 3 \leq \sum_{j=1}^{k} (|\mathcal{L}(T_j)| - 3) \tag{4}$$

must always hold. I refrain from giving a formal proof here (see, for example, Böcker *et al.*, 1999), but will explain instead the idea behind this formula. Every binary tree T has exactly $|\mathcal{L}(T)| - 3$ interior edges. Thus, Equation 4 compares the number of interior edges of the parent tree with the sum of interior edges of the input trees. Now, for every interior edge of the parent tree, at least one interior edge in at least one input tree should exist that "distinguishes" the interior edge of the parent tree. Otherwise, we could remove the interior edge from the parent tree, and the resulting tree would still be a parent tree, violating our assumption of uniqueness.

Equation 4 suggests that particular attention should be paid to the *minimum case*; that is, the case where equality holds. To this end, I define the excess of the collection $\mathcal{F}$ by

$$\text{exc}\,(\mathcal{F}) := |\mathcal{L}(\mathcal{F})| - \sum_{T \in F} (|\mathcal{L}(T)| - 3) - 3,$$

and I say that $\mathcal{F}$ is *excess-free* if $\text{exc}\,(\mathcal{F}) = 0$ holds. Clearly, $\text{exc}\,(\{T\}) = |\mathcal{L}(T)| - (|\mathcal{L}(T)| - 3) - 3 = 0$ holds for every binary tree T. The main result of this section now follows.

Theorem 1. *Suppose $\mathcal{F}$ is a non-empty and excess-free collection of binary trees. Then $\mathcal{F}$ uniquely defines a parent tree T if and only if $\#\mathcal{F} = 1$ or there* *exists a bipartition of $\mathcal{F}$ into two proper, disjoint subsets $\mathcal{F}_1$, $\mathcal{F}_2 \subset \mathcal{F}$ such that $\mathcal{F}_j$ is excess-free and uniquely defines a parent tree T_j for $j = 1, 2$, and T is the unique parent tree of T_1, T_2.*

The non-trivial part of this theorem is equivalent to the following lemma.

Lemma 2. *Given a collection of binary trees $\mathcal{F}$ that is excess-free, uniquely defines a parent tree, and has cardinality at least two, then there exist two distinct trees $T, T' \in \mathcal{F}$ such that the collection $\{T, T'\}$ is excess-free and uniquely defines a parent tree.*

Check carefully what Theorem 1 says — and even more importantly — what it *does not* say. First, the theorem guarantees the existence of a *unique* parent tree. But if the conditions of the theorem are violated, there might be either no parent tree at all or more than one parent tree. Using Theorem 1, we cannot distinguish between these two cases! But this does not come as a surprise because, as I stated above, the problem of deciding if there exists at least one parent tree of a collection of input trees is NP-complete, and below I present an algorithm with polynomial runtime to check whether the conditions of the theorem are fulfilled. Second, Theorem 1 deals with *binary* input trees only, and cannot be extended to non-binary input trees. In the following, I assume that our input collection consists solely of binary trees. Third, the theorem deals with (excess-free) tree collections of *minimum size*, and cannot be extended to minimal tree collections; that is, collections that define a unique parent tree, while any subcollection on the same leaf set has at least two parent trees (see Example 3 below). Finally, it should be obvious that biological data will almost always violate the strict conditions of the theorem, prohibiting its direct application. The idea is that, by using Theorem 1, we might be able to prove that certain supertree methods have certain desirable properties.

Example 3 (Steel, 1992). The set of quartet trees $\mathcal{Q} := \{12|35, 24|57, 13|47, 34|56, 15|67\}$ has a unique parent tree with leaf set $\{1, \ldots, 7\}$; namely, the caterpillar depicted at the top of Figure 8. For every subcollection of $\mathcal{Q}$ of cardinality two to four, at least two parent trees exist.

Actually, Steel (1992) showed only that no subcollections of cardinality four that have a unique parent tree exist, which is sufficient to show that $\mathcal{Q}$ is minimal as defined above. But, we can see easily that all *but one* subcollections $\mathcal{Q}' \subseteq \mathcal{Q}$ of cardinality $|\mathcal{Q}'| \in \{2, 3\}$ have positive excess and, hence, cannot have a unique parent tree; whereas $\mathcal{Q}' := \{12|35, 24|57, 13|47\}$, being the unique excess-free subcollection, also has two parent

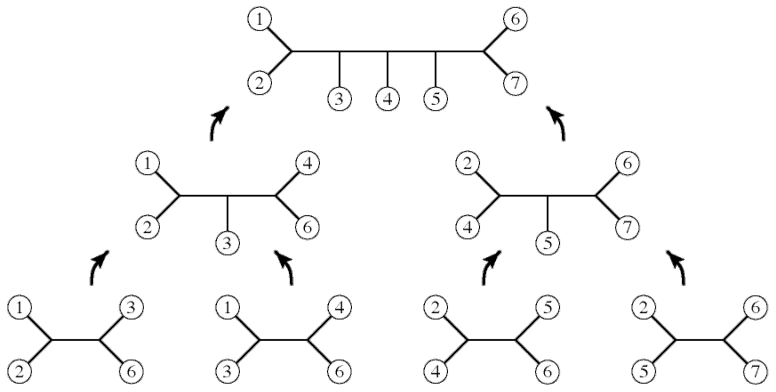

Figure 8. Hierarchy of amalgamation for the input collection $\mathcal{F} := \{12|36, 13|46, 24|56, 25|67\}$ from Example 4.

trees. In particular, this means that no two distinct trees $T, T' \in \mathcal{F}$ exist such that the collection $\{T, T'\}$ defines a parent tree (compare to Lemma 2).

Lemma 2 indicates how we can reconstruct a parent tree for the minimum case. Given our excess-free input collection $\mathcal{F}$, we search for two trees $T, T' \in \mathcal{F}$ such that the collection $\{T, T'\}$ is excess-free and uniquely defines a parent tree. Then, we amalgamate T and T' into a parent tree T'' (in fact, Lemma 4 below allows us to check whether a unique parent tree exists without actually constructing it), and replace T and T' in $\mathcal{F}$ by T''. In so doing, we have reduced the cardinality of the set $\mathcal{F}$ by one, and we repeat the process until only one parent tree is left. Surprisingly, this means that we construct a rooted tree (or, equivalently, a hierarchy) with leaves labeled by the trees of our input collection that tells us the order in which the input trees must be amalgamated to construct the parent tree of our collection (see Example 4). Böcker *et al.* (2000), present an algorithm with runtime $O(|\mathcal{L}(\mathcal{F})|^2)$ to reconstruct the unique parent tree of an excess-free input collection.

Example 4. Let $\mathcal{F} := \{12|36, 13|46, 24|56, 25|67\}$ denote a collection of quartet trees. One can show that the unique parent tree of this collection is the caterpillar tree depicted at the top of Figure 8. In addition, Figure 8 displays an amalgamation hierarchy that shows how the input trees can be amalgamated pairwise.

But why does this simple algorithm work? That is, why can we amalgamate *any* two trees of the input collection that form an excess-free subcollection? Is there no possibility that we will run into a dead end by amalgamating two "wrong" trees in the beginning such that we end up with a partially merged set of trees and can no longer find two trees to merge?

In fact, Lemma 2 is sufficient to prove that the algorithm will return the unique parent 0 in case it exists. To this end, note that the parent tree T^* of $\mathcal{F}$ necessarily displays T'', where T'' denotes the unique parent tree of $\{T, T'\}$. By contrast, all trees that display T'' must also display T and T'. This implies that the collection $\mathcal{F}' := \mathcal{F} - \{T, T'\} \cup \{T''\}$ also has the unique parent tree T^*. Finally, we can show that the collection $\mathcal{F}'$ is also excess-free, and the lemma guarantees that we can find two trees in $\mathcal{F}'$ that we can amalgamate, and so on.

But a more exhaustive answer to these questions lies in a certain structure that subcollections of our input collection exhibit. We need a new mathematical tool to capture the concept behind this structure. Let X denote an arbitrary set, and let $\mathcal{P}(X)$ denote the set of all subsets of X. I say that $C \subseteq \mathcal{P}(X)$ (that is, $C \subseteq X$ holds for all $C \in C$) is a *patchwork* if the following condition is satisfied:

$$\text{If } A, B \in C \text{ and } A \cap B \neq \emptyset, \text{ then } A \cap B \in C \text{ and } A \cup B \in C.$$

Example 5. Let $X := \mathbf{R}$ denote the set of real numbers. Then, the set of all intervals in X, $C := \{[a, b] \mid a, b \in \mathbf{R}\}$ forms a patchwork. Given two intervals $[a, b]$ and $[c, d]$, these intervals are either disjoint, or $[a, b] \cap [c, d]$ and $[a, b] \cup [c, d]$ both form intervals. The same holds true for open and half-open intervals, and if X is the set of rational or natural numbers, or the set of integers.

The following two lemmata show that such patchwork structures appear naturally in the context of constructing parent trees from minimum collections, and that the elements of this patchwork are of interest when trying to reconstruct the parent tree.

Lemma 3 (Lemma 3.10 of Böcker *et al.*, 1999). *Given an excess-free collection $\mathcal{F}$ of binary trees with a unique parent tree, the subcollections of $\mathcal{F}$ that are excess-free form a patchwork.*

Lemma 4. *Given an excess-free collection $\mathcal{F}$ of binary trees with a unique parent tree, a subcollection $\mathcal{F}' \subseteq \mathcal{F}$ is excess-free if and only if the collection $\mathcal{F}'$ has a unique parent tree.*

The latter lemma tells us that to check whether some subcollection of our input collection $\mathcal{F}$ (that has a unique parent tree) also defines some unique parent tree, it is sufficient to calculate the excess of the collection. For

Example 4, we know already that subcollections $\{12|36, 13|46\}$ and $\{24|56, 25|67\}$ are excess-free. In addition, the collection $\{12|36, 13|46, 24|56\}$ is excess-free and, hence, has a unique parent tree: the caterpillar on leaves 1, …, 6. This allows for an alternative hierarchy to reconstruct the unique parent tree of the collection. Also note that if we replace the input tree $24|56$ by $24|57$ in Example 4, then the "hierarchy of amalgamation" presented in Figure 8 displays the unique way of constructing the parent tree.

Patchworks were introduced in Böcker and Dress (2001), and several equivalent conditions were introduced for a patchwork to be ample. I call $\mathcal{C} \subseteq \mathcal{P}(X)$ *ample* if the following condition holds:

$$\text{If } A, C \in \mathcal{C} \text{ satisfies } A \subset C, \text{ and there exists no } B \in \mathcal{C} \text{ with } A \subset B \subset C, \text{ then } C \setminus A \in \mathcal{C}.$$

Recall that "$A \subset B$" denotes a proper subset $A \subseteq B$ with $A \neq B$. A set $\mathcal{C} \subseteq \mathcal{P}(X)$ is called a *hierarchy* if it satisfies:

$$\text{If } A, B \in \mathrm{C}, \text{ then } A \subseteq B, B \subseteq A, \text{ or } A \cap B = \emptyset \text{ holds.}$$

A hierarchy $\mathcal{C} \subseteq \mathcal{P}(X)$ is called *maximal* if there exists no hierarchy $\mathcal{C}' \subseteq \mathcal{P}(X)$ such that $\mathcal{C} \subset \mathcal{C}'$. Recall that there is a one-to-one correspondence between hierarchies $\mathcal{C} \subseteq \mathcal{P}(X)$ and rooted trees with leaf set X.

Theorem 2 (Theorem 1 of Böcker and Dress, 2001). *A patchwork $\mathcal{C} \subseteq \mathcal{P}(X)$ contains a maximal hierarchy if and only if $\mathcal{C}$ is ample; $\emptyset, X \in \mathcal{C}$; and $\{x\} \in \mathcal{C}$ for all $x \in X$ holds.*

In view of exc $(\{T\}) = 0$ and exc $(\mathcal{F}) = 0$ for our input collection $\mathcal{F}$, this implies that the excess-free subcollections of $\mathcal{F}$,

$$\mathcal{C}(\mathcal{F}) := \{\mathcal{F}' \subseteq \mathrm{F} \mid \text{exc}(\mathcal{F}') = 0\}$$

form an ample patchwork if and only if $\mathcal{C}(\mathcal{F}) \cup \{\emptyset\}$ contains a maximal hierarchy.

We can use the theory of patchworks to prove some non-trivial equivalences. Theorem 3 of Böcker and Dress (2001) states more equivalent conditions for a patchwork to be ample. I utilized these conditions in Böcker (1999) to show that Theorem 1 and Theorem 3 below are in fact equivalent, and that the non-trivial part of these theorems is equivalent to Lemma 2.

Proving the results presented up to this point is possible almost completely using combinatorics on leaf sets without referring explicitly to the parent tree T of the input collection. Unfortunately, this is not enough to prove Theorem 1. For its proof as presented in Böcker (1999), many more mathematical tools would have to be introduced. But even then, a lengthy "proof residual" remains, going beyond the scope of this chapter. Thus, I will present only some of the concepts and ideas used for proving Theorem 1 because they are of interest even without the proof itself.

In the following, we want to take the structure of the parent tree into account. To this end, we suppose that we know the parent tree in advance. From that, we can derive certain necessary conditions on our input collection, and, if an input collection violates any of these conditions, we know that our assumption of a unique parent tree must be violated as well.

Limiting ourselves to quartet trees only can reduce the complexity of the formalism used. However, because every binary tree T can be encoded uniquely using $|\mathcal{L}(T)| - 3$ quartet trees, this does not limit the results obtained. For example, the caterpillar from Example 4 can be encoded using the collection $\mathcal{F}$ of four elements provided in the example. To this end, let us suppose from now on that we are given a collection of *quartet* trees $\mathcal{Q}$.

The next complexity reduction comes from the following. I say that an (interior) edge e of a tree T *displays* a quartet tree $ab|cd$ with $a, b, c, d \in \mathcal{L}(T)$ if, by removing the edge e from T, the resulting graph contains a, b in one connected component and b, c in the other component. I say that the quartet tree $ab|cd$ *distinguishes* e if e is the unique edge of T that displays $ab|cd$. Now suppose that T is the unique parent tree of the collection $\mathcal{Q}$ of quartet trees. One can show easily that every interior edge e of T is distinguished by at least one quartet tree in $\mathcal{Q}$. In addition, every quartet tree distinguishes at most one interior edge of T.

If we now assume that our input collection is excess-free, then there are exactly as many interior edges in the binary parent tree T as there are quartet trees in $\mathcal{Q}$. Thus, every quartet tree in $\mathcal{Q}$ distinguishes *exactly one* interior edge of T. This means that we can assume in the following that, given an excess-free collection of quartet trees $\mathcal{Q}$ and parent tree T, we can construct a one-to-one mapping q from the interior edges of T (denoted E^*) onto the quartet trees $\mathcal{Q}$. In addition, we can assume that $q(e)$ distinguishes e for all $e \in E^*$; such mappings q are called *tight* in Böcker (1999) and Böcker *et al.* (1999). If no such mapping exists, the parent tree is not unique, and, as noted above, constructing all parent trees of $\mathcal{Q}$ might be computationally hard.

I now formulate Theorem 1 in a way that allows us to use the tools introduced above:

Theorem 3. *Given a quartet encoding q of a binary tree T with interior edges E^*, the tree T is the unique parent tree of the collection $\mathcal{Q} := q(E^*)$ if and only if* (a) *q is tight and* (b) *the patchwork C of subsets $F \subseteq E^*$ satisfying exc $(q(F)) = 0$ is ample.*

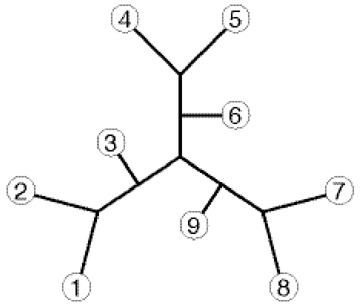

Figure 9. Parent tree from Example 6.

By looking at interior edges of T instead of quartet trees in $\mathcal{Q}$, we have gained the possibility to derive certain features of sets of interior edges. In fact, patchwork structures appear for a second time. Given a tree T with interior edges E^*, I say that $F \subseteq E^*$ is a *patch* if the subgraph induced by F (that is, removing vertices v if no edge $e \in F$ containing v exists) is connected and, therefore, a tree. Now, one can see easily that the set of all patches of a given tree T forms a patchwork too! One useful application of this concept is that excess-free subsets form patches in the tree domain:

Lemma 5 (Lemma 3.6 iii of Böcker *et al.*, 1999). *Suppose q is a tight quartet encoding of a tree T and that $F \subseteq E^*$ is a subset of interior edges. If the collection of quartet trees $q(F)$ is excess-free, then F is a patch.*

A simple application of this lemma is the following example.

Example 6. Let q denote a tight quartet encoding of the tree T depicted in Figure 9. Let F_1, F_2 denote a partitioning of E^*; that is, subsets $F_1, F_2 \subseteq E^*$ of interior edges of T with $F_1 \cap F_2 = \emptyset$ and $F_1 \cup F_2 = E^*$. Clearly, there exist no such sets F_1, F_2 with $|F_1| = |F_2| = 3$, and F_1 and F_2 are both patches. This implies that for all collections of quartet trees $\mathcal{Q}$ that have the unique parent tree T depicted in Figure 9, there is no partitioning of $\mathcal{Q}$ into two subsets $\mathcal{Q}_1, \mathcal{Q}_2$ of cardinality three that are both excess-free. In turn, this implies that no two trees T_1, T_2 with $|\mathrm{L}(T_1)| = |\mathrm{L}(T_2)| = 6$ exist such that T is the unique parent tree of the collection $\{T_1, T_2\}$.

As mentioned above, applying our results to biological data will not lead to satisfactory results because the strict conditions of Theorem 1 will be violated almost always. But the theorem can be used to prove that the *dyadic closure* (Colonius and Schulze, 1981; Dekker, 1986) can be guaranteed to return the "correct" answer in certain cases. Suppose $\mathcal{Q}$ is an arbitrary collection of quartet trees, and that all these quartet trees are induced subtrees of some (unknown) parent tree T. The dyadic closure applies two simple rules to any two trees in the collection $\mathcal{Q}$ to infer other quartet trees that are also induced subtrees of T, and adds these trees to $\mathcal{Q}$. The dyadic closure of any collection $\mathcal{Q}$ can be computed in $O(n^5)$ time for $n := |\mathcal{L}(\mathcal{Q})|$ (see Erdös *et al.*, 1999). In Böcker *et al.* (2000), the dyadic closure operation was combined with the Berry-Gascuel construction (Berry and Gascuel, 1997) to form the DYADIC TREE CONSTRUCTION algorithm.

The "performance guarantee" mentioned above is as follows. Suppose we are given a collection of quartet trees $\mathcal{Q}$. If $\mathcal{Q}$ contains an (unknown) subset $\mathcal{Q}' \subseteq \mathcal{Q}$ with leaf set $\mathcal{L}(\mathcal{Q}') = \mathcal{L}(\mathcal{Q})$ that is excess-free and has a unique parent tree T, then the DYADIC TREE CONSTRUCTION algorithm will (in polynomial runtime) either construct T or output that no parent tree of $\mathcal{Q}$ exists. But in case no such subset exists, the algorithm might be able to construct one or more parent trees, to decide that no such parent tree can exist, or get stuck without providing such information. It would be nice to decide upfront if a collection of quartet trees $\mathcal{Q}$ contains a subset $\mathcal{Q}' \subseteq \mathcal{Q}$ that is excess-free and has a unique parent tree T, but, unfortunately, this problem is also NP-complete (Böcker *et al.*, 2000).

6. Conclusions

The problem of constructing unrooted supertrees or parent trees comprises certain risks that have no equivalent in the rooted tree setting. In particular, we have seen that unrooted supertree methods cannot achieve certain desirable properties simultaneously, and that there can be a large number of supertrees containing contradicting information. These problems can be circumvented by choosing input collections such that all input trees share one or more leaves. Finally, we have seen how to derive performance guarantees for an intuitively stringent supertree method in Section 5. Although Theorem 1 cannot be applied to biological data, it might allow for other performance guarantees of this type, or for loosening the "suggestion" that all input trees should share a leaf.

Is the problem of exponentially many parent trees, as introduced in Section 4, likely to arise in practice? This depends strongly on the kind as well as the "amount" of input data provided to the given supertree method. If an unrooted supertree method tries to reconstruct a parent tree given a collection of trees of *minimal cardinality* then, as a result of the small size of Example 2, it is possible that one or more subcollections of trees analogous to $\mathcal{Q}^*$ (from Equation 3) can be found in the input collection. But if the input collection is not of minimal size, it is unclear if and how frequently these problems might arise. Yet, the existence of this phenomenon suggests the possibility of circumventing it in the first place, for example by choosing input trees with at least one leaf in common.

Acknowledgements

I want to thank Mike Steel who suggested the outline of this chapter and gave helpful feedback on the manuscript, and Olaf Bininda-Emonds for not losing hope when I was late with my submission. I also thank the latter and my two anonymous reviewers for many helpful suggestions that improved the readability of this chapter. Support was provided by the Deutsche Forschungsgemeinschaft (BO 1910/1–1) within the Computer Science Action Program.

References

BERRY, V. AND GASCUEL, O. 1997. Inferring evolutionary trees with strong combinatorial evidence. In T. Jiang and D. T. Lee (eds), *Computing and Combinatorics: Third Annual International Conference, COCOON '97, Shanghai, China, August 20–22, 1997: Proceedings*. Lecture Notes in Computer Science 1276:111–123. Springer, Berlin.

BÖCKER, S. 1999. *From Subtrees to Supertrees*. Ph.D. thesis, Universität Bielefeld, Germany. (Available from http://archiv.ub.uni-bielefeld.de/disshabi/2000/0001.ps)

BÖCKER, S. 2002. Exponentially many supertrees. *Applied Mathematical Letters* 15:861–865.

BÖCKER, S., BRYANT, D., DRESS, A. W., AND STEEL, M. A. 2000. Algorithmic aspects of tree amalgamation. *Journal of Algorithms* 37:522–537.

BÖCKER, S. AND DRESS, A. W. 2001. Patchworks. *Advances in Mathematics* 157:1–21.

BÖCKER, S., DRESS, A. W., AND STEEL, M. A. 1999. Patching up *X*-trees. *Annals of Combinatorics* 3:1–12.

BRYANT, D. AND STEEL, M. A. 1995. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics* 16:425–453.

COLONIUS, H. AND SCHULZE, H.-H. 1981. Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology* 34:167–180.

DEKKER, M. 1986. *Reconstruction Methods for Derivation Trees*. Master's thesis, Vrije Universiteit, Amsterdam, the Netherlands.

ERDÖS, P. L., STEEL, M. A., SZÉKELY, L. A., AND WARNOW, T. J. 1999. A few logs suffice to build (almost) all trees (Part 1). *Random Structures and Algorithms* 14:153–184.

GATESY, J. AND SPRINGER, M. S. 2004. A critique of matrix representation with parsimony supertrees. In O. R. P. Bininda-Emonds (ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pp. 369–388. Kluwer Academic, Dordrecht, the Netherlands.

GORDON, A. 1986. Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification* 3:335–348.

HUSON, D., NETTLES, S. AND WARNOW, T. 1999a. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology* 6:369–386.

HUSON, D., VAWTER, L., AND WARNOW, T. 1999b. Solving large scale phylogenetic problems using DCM2. In T. Lengauer, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer (eds), *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pp. 118–129. AAAI Press, Menlo Park, California.

PISANI, D. AND WILKINSON, M. 2002. Matrix representation with parsimony, taxonomic congruence, and total evidence. *Systematic Biology* 51:151–155.

PURVIS, A. 1995. A composite estimate of primate phylogeny. *Philosophical Transactions of the Royal Society of London, Series B* 348:405–421.

SANDERSON, M. J., PURVIS, A., AND HENZE, C. 1998. Phylogenetic supertrees: assembling the trees of life. *Trends in Ecology and Evolution* 13:105–109.

SEMPLE, C. AND STEEL, M. 2003. *Phylogenetics*. Oxford University Press, Oxford.

STEEL, M. A. 1992. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification* 9:91–116.

STEEL, M. A., DRESS, A. W., AND BÖCKER, S. 2000. Simple but fundamental limitations on supertree and consensus tree methods. *Systematic Biology* 49:363–368.

WILKINSON, M., THORLEY, J. L., PISANI, D., LAPOINTE, F.J., AND MCINERNEY, J. O. 2004. Some desiderata for liberal supertrees. In O. R. P. Bininda-Emonds (ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pp. 227–246. Kluwer Academic, Dordrecht, the Netherlands.