June 12, 2000

Algorithmic aspects of tree amalgamation

Sebastian Böcker[*]
GK Strukturbildungsprozesse, FSP Mathematisierung, Universität Bielefeld, PF 100 131, 33501 Bielefeld, Germany
boecker@mathematik.uni-bielefeld.de

David Bryant[†]
LIRMM, 161 rue Ada, 34392 Montpellier, Cedex 5, France
bryant@lirmm.fr

Andreas W.M. Dress[†]
GK Strukturbildungsprozesse, FSP Mathematisierung, Universität Bielefeld, PF 100 131, 33501 Bielefeld, Germany
Current Address: The City College, The City University of New York, Dept. of Chem. Engineering, Convent Avenue at 140th Street, New York, NY 10031, USA
dress@mathematik.uni-bielefeld.de

Mike A. Steel[‡]
Biomathematics Research Centre, University of Canterbury, Private Bag 4800, Christchurch, New Zealand
m.steel@math.canterbury.ac.nz

[*]Sebastian Böcker was supported by "DFG – Graduiertenkolleg Strukturbildungsprozesse", Forschungsschwerpunkt Mathematisierung, University of Bielefeld, Germany.

[†]This work was carried out while D. Bryant held a Bioinformatics Postdoctoral Fellowship from the Canadian Institute for Advanced Research, Evolutionary Biology Program (research supported in part by the Natural Sciences and Engineering Research Council of Canada and the Canadian Genome Analysis and Technology grants to D. Sankoff) and while A. Dress was working at City College, CUNY, Dept. of Chem. Engineering as a Distinguished Visiting Professor (research supported in part by "Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie", FKZ 0311642, Germany).

[‡]Mike Steel thanks the New Zealand Marsden Fund (UOC-MIS-003) for supporting this research

Proposed running head: Tree amalgamation

Contact:
Mike Steel
Biomathematics Research Centre
University of Canterbury
Private Bag 4800
Christchurch
New Zealand

`m.steel@math.canterbury.ac.nz`

Phone: +64-3-3667 001 Fax: +64-3-364 2587

Abstract. The amalgamation of leaf-labelled trees into a single (super)tree that “displays” each of the input trees is an important problem in classification. We discuss various approaches to this problem and show that a simple and well known polynomial-time algorithm can be used to solve this problem whenever the input set of trees contains a minimum size subset that uniquely determines the supertree. Our results exploit a recently established combinatorial property concerning the structure of such collections of trees.

1991 Mathematics Subject Classification. 05C05, 92D15

Key words and phrases. Trees, algorithms, dyadic closure, tree amalgamation

1. Introduction

In evolutionary biology and other fields involving tree-like classification, one is often faced with the following tree-amalgamation problem: How can one combine — or "amalgamate" — trees that classify different but overlapping sets of species into one big supertree [15]? More precisely, given a collection of trees, each of which has its leaves (vertices of degree one) labelled bijectively by some species from a given "large" collection of species, we wish to amalgamate these input trees into a single supertree (parent tree) in such a way that each input tree is "displayed" by that supertree.

Clearly, it may be impossible to amalgamate the input trees in this way, and just determining whether this is the case is known to be an NP-hard problem [16]. Furthermore, even when the trees can be amalgamated, there may be exponentially many supertrees. For example, there may be a supertree that has internal vertices of high degree, in which case any "refinement" of this tree also gives a supertree. Yet, even if every supertree is binary, an exponentially large number of supertrees can occur, see [5, 6].

In this paper, we consider the question of determining whether the collection of input trees uniquely determines a possible supertree. We begin by introducing some terminology. We will view (leaf-labelled) trees as graphs, rather than representing them via systems of splits.

Definitions 1.

- We consider trees whose leaves (degree-one vertices) are labelled,[1] and whose remaining vertices (of which we assume that there exists at least one) are unlabelled and of degree at least three. Such a tree is also called a *phylogenetic* tree or, even more specifically, a phylogenetic X-tree where X denotes the set of its labels. If all of the non-leaf vertices have degree three, the tree is said to be a *binary* tree. An edge incident with a leaf is said to be a *pendant* edge, every other edge is said to be *interior*.

- For a tree T, let $\mathcal{L}(T)$ 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 $\bigcup_{T\in\mathcal{F}} \mathcal{L}(T)$. Recall that the number $i(T)$ of interior edges of T never exceeds $|\mathcal{L}(T)| - 3$, and equality holds if and only if T is binary (see for instance [6]).
- The *excess* of a collection $\mathcal{F}$ of trees, denoted $\mathrm{exc}(\mathcal{F})$, is defined by

$$\mathrm{exc}(\mathcal{F}) := |\mathcal{L}(\mathcal{F})| - 3 - \sum_{T\in\mathcal{F}} i(T).$$

  We shall see that if $\mathcal{F}$ has positive excess then it contains too many leaves to define a (unique) tree (Lemma 1). In this paper, we will be paying particular attention to collections of trees $\mathcal{F}$ for which $\mathrm{exc}(\mathcal{F}) = 0$ holds. We will call such collections *excess-free.*
- Given a tree T and a subset $L \subseteq \mathcal{L}(T)$, we denote by $T|_L$ the phylogenetic tree obtained from the smallest connected subgraph of T containing (the leaves labelled by) L, by making this subgraph "homeomorphically irreducible" (i.e. by suppressing all degree two vertices). We 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.
- Given two trees T, T' with $\mathcal{L}(T) = \mathcal{L}(T')$, we 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 $\mathcal{F} := \{T_1, \dots, T_r\}$ is a collection of trees. We say that a tree T *displays* $\mathcal{F}$ if $T_i \leq T|_{\mathcal{L}(T_i)}$ holds for all $i = 1, \dots, r$. The collection $\mathcal{F}$ is said to be *compatible* if it is displayed by at least one tree T, in which case $\mathcal{F}$ is said to *define* T if T is the only tree with leaf set $\mathcal{L}(\mathcal{F})$ that displays $\mathcal{F}$. Note that a tree T that is defined by some collection $\mathcal{F}$ of trees is necessarily binary. We say that $\mathcal{F}$ is *definitive* if $\mathcal{F}$ is compatible and defines a tree T.

- A *quartet tree* is a binary tree T with $|\mathcal{L}(T)| = 4$.

In general, it appears to be a difficult problem to determine whether or not a given collection $\mathcal{F}$ of trees is definitive. However, we show that it has a polynomial time solution whenever the input trees comprise or, at least, contain a definitive and excess-free subset $\mathcal{F}'$ of binary trees. Our results lean heavily upon, and provide a nice application of, a recent combinatorial result concerning the reconstruction of leaf-labelled trees from "tight" sets of subtrees (cf. [7] for a general account and [6] for a rigorous proof). To explain our results in more detail, note that (i) given a collection $\mathcal{F}$ of input trees, a tree T is said to be *implied* by $\mathcal{F}$ if there exists a compatible subset $\mathcal{F}' \subseteq \mathcal{F}$ such that T is displayed by every tree T' that displays $\mathcal{F}'$, and that (ii) there exist arbitrarily large collections $\mathcal{F}$ of quartet trees such that every tree implied by a proper subset $\mathcal{F}' \subsetneq \mathcal{F}$ is already contained in $\mathcal{F}'$ (cf. [9]). In contrast, we will show here that "dyadic" closure operations (using just two trees at a time) suffice to reconstruct the unique supertree T defined by an excess-free, definitive collection $\mathcal{F}$ of trees in $O\big(|\mathcal{L}(\mathcal{F})|^2\big)$ time.

This results suggests to search for an efficient algorithm that would, given an arbitrary collection $\mathcal{F}$ of trees, return an excess-free and definitive subset $\mathcal{F}' \subseteq \mathcal{F}$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ in case such a subset exists and, otherwise, the information that such a subset does not exist. Such an algorithm, however, cannot be expected to exist because — as we will show in the last section — this task belongs in fact to the class of NP-complete problems.

Yet, remarkably, we can still find the unique supertree T for a compatible collection $\mathcal{F}$ that just contains (but does not necessarily coincide with) an excess-free, definitive collection $\mathcal{F}' \subseteq \mathcal{F}$ of quartet trees with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ by using another, already existing algorithm (also based on dyadic closure operations) in $O\big(|\mathcal{L}(\mathcal{F})|^5\big)$ time. As pointed out already in [7], this generalizes in particular results obtained in [12] where dyadic closure operations were shown to suffice for supertree construction if the set of input trees contained all the "short quartets" of the supertree.

So, to summarize clearly what we can and what we cannot do in polynomial time (unless $\mathrm{P} = \mathrm{NP}$ holds), we distinguish four cases regarding a given collection $\mathcal{F}$ of input trees:

Case $(++)$: $\mathcal{F}$ is compatible and contains an excess-free, definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$;

Case $(-+)$: $\mathcal{F}$ is incompatible and contains such a subset $\mathcal{F}'$;

Case $(+-)$: $\mathcal{F}$ is compatible and does not contain an excess-free, definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$;

Case $(--)$: $\mathcal{F}$ is incompatible and does not contain such a subset $\mathcal{F}'$.

In case $(++)$, there exists a unique supertree for $\mathcal{F}$, and the algorithm referred to above will find it in polynomial time. In case $(-+)$, the same algorithm will output in polynomial time that no supertree can exist. In case $(+-)$, the algorithm might provide enough information to find one or several supertrees, it might also establish that $\mathcal{F}$ is definitive — yet, it might also get stuck without providing this information. And in case $(--)$, the algorithm might output that no supertree exists, or it might get stuck without doing so.

So, whenever this algorithm finds several supertrees or gets stuck before being able to decide whether $\mathcal{F}$ is compatible or not, we learn (in polynomial time) that no excess-free, definitive set $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ is contained in $\mathcal{F}$; while if it finds a unique supertree or establishes that $\mathcal{F}$ is incompatible, we have solved the supertree problem for $\mathcal{F}$, yet we do not learn from this solution whether or not an excess-free, definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ is contained in $\mathcal{F}$. So, this special question remains unanswered only in case the problem that we really want to solve (i.e. the problem of deciding whether $\mathcal{F}$ is definitive or incompatible) can be solved in polynomial time.

Our approach complements some earlier results that also deal with special cases where one can easily determine whether or not $\mathcal{F}$ is compatible, and, if so, definitive. For example, if

$$\bigcap_{T\in\mathcal{F}} \mathcal{L}(T) \neq \emptyset$$

then one can determine in polynomial time (in $|\mathcal{L}(\mathcal{F})|$) whether or not $\mathcal{F}$ is compatible [1] and if so whether $\mathcal{F}$ is definitive [16]. Alternatively, if the number of trees in $\mathcal{F}$ is bounded, then there is also an algorithm that runs in polynomial time in $|\mathcal{L}(\mathcal{F})|$ for answering these last two questions, see [16]. Some heuristic and approximation-based approaches to tree amalgamation have also been proposed, particularly for (possibly incompatible) collections of quartet trees. Two such heuristic methods include *Quartet puzzling*, introduced by Strimmer and von-Haeseler [17], and a novel approach based on semi-definite programming by Ben-Dor *et al.* [3]. A polynomial time approximation scheme for the problem of finding the largest compatible subset of a set $\mathcal{F}$ of quartet trees has recently been described by Jiang *et al* [14] (under the strong assumpution that for each subset L of $\mathcal{L}(\mathcal{F})$ of size four there is a quartet tree T in $\mathcal{F}$ with $\mathcal{L}(T) = L$).

This paper is organized as follows: We first list some further definitions that are required for the remainder of the paper. In the next section, we consider the tree reconstruction problem when $\mathcal{F}$ consists of just two trees. In Section 3, we provide a simple algorithm that solves the tree reconstruction problem on sets of trees that are excess-free. And in Section 4, we describe an algorithm that works in a slightly more natural as well as more general setting, and finally show the NP-completeness of the problem whether or not $\mathcal{F}$ contains some definitive and excess-free subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$.

We end this section with some further definitions that will be required below.

Definitions 2.

- We write $xy|wz$ to denote the quartet tree that has leaves labelled x, y separated from leaves labelled w, z by its unique interior edge. More generally, we let $x_1 \dots x_r|y_1 \dots y_s$ denote the tree with exactly one interior edge $e = \{u, v\}$, with leaves labelled $x_1, \dots, x_r$ adjacent to u, and leaves labelled $y_1, \dots, y_s$ adjacent to v.
- For a tree T, let
$$\mathcal{Q}[T] := \{T|_L : L \subseteq \mathcal{L}(T),\ |L| = 4,\ T|_L \text{ is a binary tree}\}$$
denote the collection of quartet trees induced by all 4-subsets L of $\mathcal{L}(T)$, and for a collection $\mathcal{F}$ of trees, put
$$\mathcal{Q}[\mathcal{F}] := \bigcup_{T \in \mathcal{F}} \mathcal{Q}[T].$$
- Given a quartet tree $xy|wz$ that is displayed by a tree T, we say that $xy|wz$ *distinguishes* an edge e of T if e is the only edge of T that separates the leaves labelled x, y from the leaves labelled w, z.

2. Amalgamating pairs of trees

Our discussion on tree amalgamation begins with the simplest case: Amalgamating pairs of trees.

Theorem 1. *Suppose $\mathcal{F} = \{T_1, T_2\}$ consists of two trees and consider $\mathcal{I} := \mathcal{L}(T_1) \cap \mathcal{L}(T_2)$.*

1. *$\mathcal{F}$ is compatible if and only if $\{T_1|_{\mathcal{I}}, T_2|_{\mathcal{I}}\}$ is compatible.*
2. *Suppose that T displays $\mathcal{F}$. Then the following three assertions are equivalent:*
   (a) *$\mathcal{F}$ defines T;*
   (b) *T is binary and no "contraction" of T (that is, some tree T' with $T' < T$) displays $\mathcal{F}$;*
   (c) *T is binary and, for every interior edge e of T, there is an induced quartet tree of T_1 or T_2 that distinguishes e.*

Proof. The proof of (1) is straightforward.

2a $\Rightarrow$ 2b follows from the fact that if a non-binary tree T' displays $\mathcal{F}$, then any refinement of T' also displays $\mathcal{F}$.

2b $\Rightarrow$ 2c follows from the observation that if e is not distinguished by a quartet tree induced by T_1 or T_2, then contracting e in T gives a non-binary tree that displays $\mathcal{F}$.

2c $\Rightarrow$ 2a: Counting the interior edges of T, T_1, and T_2 and noting that Condition 2c implies $i(T) \leq i(T_1) + i(T_2)$, we get

$$\begin{aligned}\big|\mathcal{L}(T_1) \cup \mathcal{L}(T_2)\big| &= |\mathcal{L}(T)| = i(T) + 3\\ &\leq i(T_1) + i(T_2) + 3\\ &\leq |\mathcal{L}(T_1)| + |\mathcal{L}(T_2)| - 3\\ &= \big|\mathcal{L}(T_1) \cup \mathcal{L}(T_2)\big| + |\mathcal{I}| - 3\end{aligned}$$

and, therefore, $|\mathcal{I}| \geq 3$. Choose $x \in \mathcal{I}$ and consider the set Q_x of induced quartet subtrees of T_1 or T_2 that contain x. For every edge e there is an induced quartet subtree of T_1 or T_2 that distinguishes e, so there must also be an induced quartet subtree in Q_x that distinguishes e. The result then follows from Theorem 3 of [16]. □

Remark 1. If T_1 and T_2 are binary, then $\mathcal{F}$ is compatible if and only if $T_1|_{\mathcal{I}} \cong T_2|_{\mathcal{I}}$, since two binary trees on the same leaf set are compatible if and only if they are isomorphic.

Given two trees T_1 and T_2, we can use the linear time compatibility algorithm of [18] to see whether the collection $\{T_1, T_2\}$ is definitive: First, we choose a leaf in $\mathcal{L}(T_1) \cap \mathcal{L}(T_2)$ to root these two trees. We can then determine whether $T_1|_{\mathcal{I}}$ and $T_2|_{\mathcal{I}}$ are compatible using the compatibility algorithm of [18]. If they are compatible but the output tree (denoted $T_{\mathcal{I}}$ here) is not binary then $\{T_1, T_2\}$ is clearly not definitive. Suppose in the following that $T_{\mathcal{I}}$ is binary.

Using a depth first traversal on T_1 and then T_2, we can append the leaves appearing in only one tree to the tree $T_{\mathcal{I}}$ to obtain a binary tree T that displays both T_1 and T_2. A third depth first search, this time on T, can then be used to determine which subtrees of T contain leaves in $\mathcal{L}(T_1)$ and which contain leaves in $\mathcal{L}(T_2)$. We can then test for which edges Condition 2c of Theorem 1 holds. In this way, we can determine in $O(|\mathcal{L}(T_1)| + |\mathcal{L}(T_2)|)$ time whether two trees T_1 and T_2 form a definitive collection.

3. Amalgamating excess-free collections of trees

Lemma 1. *Suppose $\mathcal{F}$ defines T_0. Then, the following holds:*

1. $\mathrm{exc}(\mathcal{F}) \leq 0$
2. *If $\mathcal{F}' \subseteq \mathcal{F}$ defines a tree T', then $\mathcal{F}_* := \{T'\} \cup (\mathcal{F} - \mathcal{F}')$ defines T_0 and $\mathrm{exc}(\mathcal{F}) = \mathrm{exc}(\mathcal{F}') + \mathrm{exc}(\mathcal{F}_*)$; in particular, $\mathrm{exc}(\mathcal{F}) = 0$ implies $\mathrm{exc}(\mathcal{F}') = \mathrm{exc}(\mathcal{F}_*) = 0$.*
3. *If $\mathcal{F}' \subseteq \mathcal{F}$ and $\mathrm{exc}(\mathcal{F}') = \mathrm{exc}(\mathcal{F}) = 0$ then $\mathcal{F}'$ is definitive.*
4. *There is a collection $\mathcal{F}_*$ of* binary *trees that defines T_0, with each tree $T' \in \mathcal{F}_*$ being an induced subtree of some tree in $\mathcal{F}$.*

Proof. Part 1: This follows immediately from

$$\begin{aligned} |\mathcal{L}(\mathcal{F})| = |\mathcal{L}(T_0)| &= i(T_0) + 3 \\ &\leq \sum_{T \in \mathcal{F}} i(T) + 3 \end{aligned}$$

(see also [9], Proposition 3).

Part 2: Clearly, $\mathcal{F}_*$ defines T_0. Furthermore,

$$\begin{aligned}
\mathrm{exc}(\mathcal{F}') + \mathrm{exc}(\mathcal{F}_*) &= \left|\mathcal{L}(\mathcal{F}')\right| - 3 - \sum_{T \in \mathcal{F}'} i(T) + |\mathcal{L}(\mathcal{F}_*)| - 3 \\
&\qquad - i(T') - \sum_{T \in \mathcal{F} - \mathcal{F}'} i(T) \\
&= |\mathcal{L}(\mathcal{F})| + \big(\left|\mathcal{L}(\mathcal{F}')\right| - 3 - i(T')\big) - 3 - \sum_{T \in \mathcal{F}} i(T) \\
&= \mathrm{exc}(\mathcal{F}).
\end{aligned}$$

Part 3: This follows from Lemma 6.10 of [6].

Part 4: Given any two trees T and T', we have $T \leq T'$ precisely if T' displays $\mathcal{Q}[T]$ (the set of induced binary quartet subtrees of T). Thus, we may set $\mathcal{F}_* = \bigcup_{i=1}^{k} \mathcal{Q}[T_i]$ to satisfy part (4). □

Remark 2. Lemma 1(4) cannot be strengthened by insisting that $|\mathcal{F}_*| = |\mathcal{F}|$ or $\mathrm{exc}(\mathcal{F}_*) = \mathrm{exc}(\mathcal{F})$ should hold even if $\mathcal{F}$ is excess-free and definitive. An example is provided by the set

$$\mathcal{F} := \{123|47, 45|16, 67|25, 345|12\}$$

which defines the binary tree on the leaf set $\{1, 2, \ldots, 7\}$ shown in Fig. 1, yet no four induced quartet trees define this tree: This follows immediately from the fact that none of the three collections

$$\begin{aligned}
\mathcal{F}_1 &:= \mathcal{F} - \{345|12\} \cup \{34|12\}, \\
\mathcal{F}_2 &:= \mathcal{F} - \{345|12\} \cup \{35|12\}, \quad \text{and} \\
\mathcal{F}_3 &:= \mathcal{F} - \{345|12\} \cup \{45|12\}
\end{aligned}$$

is definitive, see Fig. 2 and note that the tree depicted in Fig. 1 still displays $\mathcal{F}_3$ when contracting the interior edge separating leaves labeled 1, 2 from $3, \ldots, 7$.

[Figure 1 should appear approximately here.]

[Figure 2 should appear approximately here.]

The following theorem is a simple consequence of (and essentially equivalent to) the main theorem in [7].

Theorem 2. *Any excess-free, definitive collection $\mathcal{F}$ of binary trees contains two trees that together form an excess-free definitive set.*

Proof. Write $\mathcal{F} = \{T_1, \dots, T_k\}$. For each tree T_i in $\mathcal{F}$, we can choose an excess-free collection $\mathcal{Q}_i$ of induced binary quartet trees that define T_i. For instance (cf. [16], Proposition 6) we may choose some leaf x_i in $\mathcal{L}(T_i)$, and for every interior edge e of T_i, we may choose leaves a_e, b_e, c_e in $\mathcal{L}(T_i)$ such that $x_i a_e | b_e c_e$ distinguishes e. Then, we put

$$\mathcal{Q}_i := \{x_i a_e | b_e c_e : e \text{ is an interior edge of } T_i\}.$$

It is easily seen that the union of these sets is also excess free and definitive, so by [7], Theorem 3.11 and [8], Theorem 3 the union two of them — say $\mathcal{Q}_i$, $\mathcal{Q}_j$ — is excess-free and definitive, too. But this implies that $\{T_i, T_j\}$ is excess-free and definitive. □

Remark 3. Theorem 2 fails if we drop the restriction that the trees in the collection be binary, see $\mathcal{F}$ given in Remark 2. Theorem 2 also requires the condition that $\mathcal{F}$ is excess-free: The set

$$\mathcal{F} := \{12|35, 24|57, 13|47, 34|56, 15|67\} \tag{1}$$

defines the tree depicted in Fig. 3 (because starting with the hypothetical quartet tree 13|45 and then consecutively adding the quartet trees 12|35, 24|57, and 15|67 leaves us no other choice; while starting with 14|35 or 15|34 leads to a contradiction in both cases). Yet, for any subset $\mathcal{F}' \subset \mathcal{F}$ of size two there are at least two trees that display $\mathcal{F}'$.

[FIGURE 3 SHOULD APPEAR APPROXIMATELY HERE.]

Theorem 2 leads directly to a polynomial time algorithm for determining whether an excess-free set $\mathcal{F}$ of binary trees defines a tree. A straightforward approach would be to search for two trees T_1, T_2 in $\mathcal{F}$ such that $\{T_1, T_2\}$ is excess-free and definitive, and to replace these two trees by the tree T they define. Repeating these replacements, we end up with an algorithm that obviously has polynomial runtime. Yet, using an approach established in [8], we can construct an even faster algorithm:

To this end, we recall why the above algorithm actually works: Let $\mathcal{F}$ denote an excess-free and definitive set of binary trees. In [7], we established that a non-empty subset $\mathcal{F}'$ of $\mathcal{F}$ is definitive if and only if it is excess-free, and that, given excess-free subsets $\mathcal{F}_1$, $\mathcal{F}_2$ of $\mathcal{F}$ with non-empty intersection, the sets $\mathcal{F}_1 \cap \mathcal{F}_2$ and $\mathcal{F}_1 \cup \mathcal{F}_2$ are excess-free, too. So, the excess-free subsets of $\mathcal{F}$ form a *patchwork* as defined in [8], and this is the reason why the simple algorithm described above cannot run into a "dead end". We can use a (slightly modified) algorithm with square runtime as described in Fig. 4 of [8] to check whether $\mathcal{F}$ defines a tree:

[ALGORITHM AEFT SHOULD APPEAR APPROXIMATELY HERE.]

The algorithm (which we abbreviate as AEFT) can be implemented to run in $O(n^2)$ time for $n := |\mathcal{L}(\mathcal{F})|$. In this algorithm, *Ins* is an array of sets that contain at most three elements (and that is indexed by two-element sets $\{T_1, T_2\}$), so the union of two such sets can be constructed in constant time. Recalling that we can compute a tree T that displays both T_1 and T_2 in linear time (see again [18]), the total runtime is in fact $O(n^2)$. The following lemma can be deduced from the above observations (see also [6]):

Lemma 2. *Given an excess-free collection of binary trees $\mathcal{F}$, the algorithm AEFT either returns some binary tree T (in case $\mathcal{F}$ defines T) or the statement* non-definitive *(otherwise).*

Note that, in case AEFT returns the statement *non-definitive*, there might be either no or several trees displaying $\mathcal{F}$, and deciding which of these two alternatives holds is an NP-hard problem in general.

4. An algorithm for arbitrary collections of binary trees

We now have a polynomial time algorithm for the case where an excess-free collection $\mathcal{F}$ of binary trees defines some binary tree T. However, it would be considerably more useful in "real world" applications to provide a polynomial time algorithm that applies to arbitrary collections of binary trees. We would like the algorithm to have the property that when $\mathcal{F}$ contains an (unknown) excess-free definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ — in which case $\mathcal{F}$ must either be incompatible, or define a tree (namely, the tree T defined by $\mathcal{F}'$) — then the algorithm will determine which of these two alternatives holds, and in the latter case it should actually reconstruct the tree T.

In this section we describe such a polynomial time algorithm that applies to any collection of binary trees. The algorithm returns one of the following:

- the statement that $\mathcal{F}$ is incompatible;
- a binary phylogenetic tree T that is defined by $\mathcal{F}$;
- a non-binary tree that comprises some of the information given by $\mathcal{F}$ (as described in detail below) in case $\mathcal{F}$ does *not* contain an excess-free, definitive set $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$.

In this general setting it is no longer always possible to "blindly search" for two trees that define a third and then amalgamate these (as was possible in the previous algorithm). For example, consider the collections

$$\mathcal{F}' = \{12|34, 23|47, 17|45, 25|67\} \quad \text{and} \quad \mathcal{F} = \mathcal{F}' \cup \{13|46\}.$$

Then $\mathcal{F}$ is compatible, $\mathcal{F}'$ is an excess-free subset of $\mathcal{F}$ that defines the tree shown in Fig. 1, and $\mathcal{L}(\mathcal{F}) = \mathcal{L}(\mathcal{F}')$. Suppose that we initially decided to consider the pair of quartet trees $12|34, 13|46$ which forms an excess-free definitive subset of $\mathcal{F}$. If we now replace these two quartet trees in $\mathcal{F}$ by the tree they define, then we find that no two trees in the resulting set of trees form a definitive set. Consequently, this set does not reduce further by the rules of that algorithm. Of course, had we chosen our two trees differently, we could have avoided this problem. But without knowing $\mathcal{F}'$, there seems to be no obvious way to determine in advance which pair to consider while, if we try all possible pairs, the required time might grow at an exponential rate.

Instead, we base the algorithm on the the notion of "dyadic closure" introduced by Dekker [10] and further developed in [11, 12].

Definition 3. The *dyadic closure* of a collection $\mathcal{Q}$ of quartet trees, denoted $cl_2(\mathcal{Q})$, is the minimal set of quartet trees that contains $\mathcal{Q}$ and satisfies the following two rules:

(dc1) $\qquad ab|cd,\ ab|ce \in cl_2(\mathcal{Q}) \implies ab|de \in cl_2(\mathcal{Q})$

(dc2) $\qquad ab|cd,\ ac|de \in cl_2(\mathcal{Q}) \implies ab|ce,\ ab|de,\ bc|de \in cl_2(\mathcal{Q})$

We define the *semidyadic closure* of $\mathcal{Q}$, denoted $scl_2(\mathcal{Q})$, in the same way as $cl_2(\mathcal{Q})$, except that we only require (dc2) to hold.

Dekker [10] observed that $\mathcal{Q}$ is compatible if and only if $cl_2(\mathcal{Q})$ is compatible. We can also use the dyadic closure to extend Theorem 1:

Lemma 3. *Suppose that $\mathcal{F} = \{T_1, T_2\}$ and T is a binary tree that displays both T_1 and T_2. Then $\mathcal{F}$ defines T if and only if*

$$scl_2(\mathcal{Q}[\mathcal{F}]) = \mathcal{Q}[T].$$

Proof. If $scl_2(\mathcal{Q}[\mathcal{F}]) = \mathcal{Q}[T]$ and T' displays $\mathcal{F}$, then T' must also display T, which implies $T' = T$ since T is binary. Hence $\mathcal{F}$ defines T.

Conversely, suppose that $\mathcal{F}$ defines T. We prove the result by induction, noting that it is trivially true for $|\mathcal{L}(T)| = 4$. Suppose that the hypothesis holds in case $|\mathcal{L}(T)| \leq n$ and that $|\mathcal{L}(T)| = n + 1 \geq 5$. Let y, y' denote two leaves of T which form twins of T, that is, which are adjacent to the same vertex of T; and let z, z' denote a second such pair of twins — it is easy to see that at least two (leaf-disjoint) pairs of twins exist in every tree with at least four distinct leaves.

Let T' be T with the leaf y' removed, and let $\mathcal{F}'$ be $\mathcal{F}$ with the leaf y' removed from trees containing it. If there were some tree $T'' \ncong T'$ also displaying $\mathcal{F}'$, then we could construct a tree different from T that would display $\mathcal{F}$ by appending y' to the edge adjacent to y in T''. Hence $\mathcal{F}'$ defines T'. By the induction hypothesis, $scl_2(\mathcal{Q}[\mathcal{F}']) = \mathcal{Q}[T']$ and $\mathcal{Q}[T'] \subseteq scl_2(\mathcal{Q}[\mathcal{F}])$.

The same holds true replacing y by y', z, z', respectively. This shows that $scl_2(\mathcal{Q}[\mathcal{F}])$ contains all of $\mathcal{Q}[T]$ except the quartet tree $yy'|zz'$ which can be derived by choosing any other leaf x in $\mathcal{L}(T)$ and applying the rule

$$yy'|xz,\, yx|zz' \in scl_2(\mathcal{Q}) \implies yy'|zz' \in scl_2(\mathcal{Q}).$$

□

The following theorem implies that the algorithm DCT below does in fact behave as promised above.

Theorem 3. *If $\mathcal{F}$ is a compatible collection of binary trees containing an excess-free collection $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$ which defines a binary tree T, then*

$$scl_2(\mathcal{Q}[\mathcal{F}]) = \mathcal{Q}[T].$$

Proof. Clearly, $scl_2(\mathcal{Q}[\mathcal{F}]) \subseteq \mathcal{Q}(T)$ must hold. To establish the converse recall that, by Theorem 2, we can combine pairs of trees from (or derived from) $\mathcal{F}'$ consecutively to obtain T. Each time we combine two trees T_1 and T_2 to form a third tree T_3, we have $scl_2(\mathcal{Q}(\{T_1, T_2\})) = \mathcal{Q}[T_3]$ by Lemma 3 and, hence, $\mathcal{Q}[T_3] \subseteq scl_2(\mathcal{Q}[\mathcal{F}])$. This, however, implies $\mathcal{Q}[T] \subseteq scl_2(\mathcal{Q}[\mathcal{F}])$, as claimed. □

Remark 4. The converse of Theorem 3 does not hold for arbitrary collections of trees (see, for example, the collection $\mathcal{F}$ described in Remark 2). We do not know whether it holds for collections $\mathcal{F}$ containing only quartet trees.

Theorem 3 leads directly to the tree amalgamation algorithm (DCT). In this algorithm, the Berry-Gascuel tree (see [4]) is a tree that can be constructed from any (compatible or incompatible) set $\mathcal{Q}$ of quartet trees provided that $\mathcal{Q}$ contains at most one tree on the same leaf set. We will denote the resulting Berry-Gascuel tree for $\mathcal{Q}$ by $T_{\mathcal{Q}}$ (rather than by $\mathcal{Q}^*$ as in [4]). Then, the Berry-Gascuel construction satisfies the following properties:

- $\mathcal{L}(T_{\mathcal{Q}}) = \mathcal{L}(Q)$;
- $T = T_{\mathcal{Q}}$ satisfies $\mathcal{Q}[T] \subseteq \mathcal{Q}$ and, moreover, $T_{\mathcal{Q}}$ displays all trees T that satisfy this condition;
- in particular, if $\mathcal{Q} = \mathcal{Q}[T]$ for some tree T, then $T_{\mathcal{Q}} \cong T$.

Informally, $T_{\mathcal{Q}}$ is a conservatively resolved tree whose edges induce exactly those "splits" (i.e. bipartitions) of $\mathcal{L}(\mathcal{Q})$ $(= \mathcal{L}(T))$ that are unanimously supported by the quartet trees in $\mathcal{Q}$.

[Algorithm DCT should appear approximately here.]

The algorithm DCT can be implemented to run in $O(n^5)$ time, using an approach similar to the one described in [12] for computing $cl_2(\mathcal{Q}[\mathcal{F}])$, and the $O(n^4)$ time algorithm for the Berry-Gascuel construction of [4]. The following result justifies the correctness of the algorithm and follows from Theorem 3 (together with the third listed property of the Berry-Gascuel construction).

Lemma 4. *If a collection of trees $\mathcal{F}$ contains a definitive excess-free subset of binary trees $\mathcal{F}' \subseteq \mathcal{F}$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$, then DCT either returns the binary tree defined by $\mathcal{F}$ (in which case $\mathcal{F}$ is compatible) or, otherwise, the statement* incompatible*.*

In fact, the algorithm is stronger than Lemma 4 implies. Consider the collection

$$\mathcal{Q}^* := \{12|34, 12|45, 26|15, 45|36\}.$$

The algorithm DCT will return the tree T defined by $\mathcal{Q}^*$ even though $scl_2(\mathcal{Q}^*) \neq \mathcal{Q}[T]$ holds, so (by Theorem 3) $\mathcal{Q}^*$ does not contain an excess-free definitive subset. There are also cases when a collection $\mathcal{Q}$ defines a tree T even though $cl_2(\mathcal{Q}) \neq \mathcal{Q}[T]$, one example being the collection described in Remark 3. Indeed, the computational complexity of determining whether a given set of quartet trees $\mathcal{Q}$ defines a (known!) binary tree T is unknown.

Even when we cannot determine a tree defined by $\mathcal{F}$, we can use the partially resolved tree T returned by DCT to infer some properties of the trees that display $\mathcal{F}$.

It would be useful to determine in advance whether an arbitrary collection $\mathcal{F}$ of trees contains an excess-free, definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$. Unfortunately, we may assume that this is difficult, due to the following theorem.

Theorem 4. *Let $\mathcal{Q}$ denote a set of quartet trees. Then, the problem of determining whether or not there exists an excess-free, definitive subset $\mathcal{Q}'$ of $\mathcal{Q}$ with $\mathcal{L}(\mathcal{Q}') = \mathcal{L}(\mathcal{Q})$ is NP-complete.*

Proof. The excess of a given subset $\mathcal{Q}'$ can be computed in polynomial time. Furthermore, given an excess-free subset $\mathcal{Q}'$, we have shown how to verify in polynomial time whether or not $\mathcal{Q}'$ defines some binary tree. Consequently, the problem is in NP.

We use a simple reduction from the NP-complete problem, DIRECTED HAMILTONIAN PATH [13].

By a *caterpillar tree* we mean a binary tree in which each non-leaf vertex is adjacent to at least one leaf, that is, a binary tree with exactly two pairs of twins. Given a digraph $G = (V, A)$, choose two “new” vertices $x, y \notin V$ and construct the set of quartet trees

$$\mathcal{Q} = \{xa|by : (a, b) \in A\}.$$

If $\mathcal{Q}$ contains a subset of size $|\mathcal{L}(\mathcal{Q})| - 3 = |V| - 1$ that defines a binary tree, then this tree T is necessarily a caterpillar tree with x and y at opposite "ends" of the tree (see [16]) and this holds if and only if the remaining leaves of the caterpillar tree trace a Hamiltonian path for G. $\square$

References

1. A. V. Aho, S. Yehoshua, T. G. Szymanski, and J. D. Ullman, Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions, SIAM J. Comput. **10(3)**: 405–421, 1981.
2. H.-J. Bandelt and A.W.M. Dress, Reconstructing the shape of a tree from observed dissimilarity data, Adv. Appl. Math. **7**: 309–343, (1986).
3. A. Ben-Dor, B. Chor, D. Graur, R. Ophir and D. Pelleg, Constructing phylogenies from quartets: elucidation of eutherian superordinal relationships, J. Comput. Biol. **5**(3): 377–390, 1998.
4. V. Berry and O. Gascuel, Inferring evolutionary trees with strong combinatorial evidence, Proc. COCOON, 1997.
5. S. Böcker, Exponentially many supertrees, in preparation.
6. S. Böcker, From subtrees to supertrees, Ph.D. thesis, Fakultät für Mathematik der Universität Bielefeld, Germany, 1999.
7. S. Böcker, A.W.M. Dress, and M.A. Steel, Patching up *X*-trees, Ann. Combin. **3**: 1–12, 1999.
8. S. Böcker and A.W.M. Dress, Patchworks, Adv. Math. (in press).
9. D. Bryant and M. Steel, Extension operations on sets of leaf-labelled trees, Adv. Appl. Math. **16**: 425–453, 1995.
10. M.C.H. Dekker, Reconstruction methods for derivation trees, Master's Thesis, Vrije Universiteit, Amsterdam, Netherlands, 1986.
11. P.L. Erdős, M. Steel, L.A. Székely and T. Warnow, Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule, Computers and Artificial Intelligence **16**(2): 217–227, 1997.
12. P.L. Erdős, L.A. Székely, M. Steel and T. Warnow, A few logs suffice to build (almost) all trees (I), Random Structures and Algorithms **14**: 153–184, 1999.
13. M.R. Garey and D.S. Johnson, Computers and intractability, W.H. Freeman and Co. San Francisco, 1979.

14. T. Jiang, P.E. Kearney and M. Li, Orchestrating quartets: approximation and data correction, Proc. 39th IEEE Symposium on the Foundations of Computer Science, 416–425, 1998.

15. M.J. Sanderson, A. Purvis, and C. Henze, Phylogenetic supertrees: Assembling the trees of life, Trends In Ecology & Evolution **13**(3): 105–109, 1998.

16. M. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, J. Classification **9**: 91–116 (1992).

17. K. Strimmer and A. von Haeseler, Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies, Mol. Biol. Evol. **13**(7): 964–969, 1996.

18. T.J. Warnow, Tree compatibility and inferring evolutionary history, J. Algorithms **16**(3): 388–407, 1994.

Footnotes

[1]That is, there exists a bijective mapping from the set of labels onto the leaves of T. In the following, we will usually suppose (without loss of generality) that this mapping is the identity.

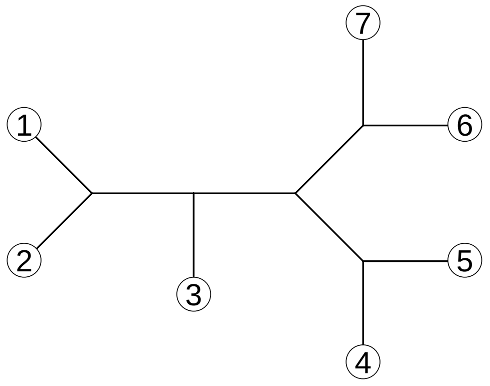

FIGURE 1. The binary tree that is defined by the four trees 123|47, 45|16, 67|25, 345|12.

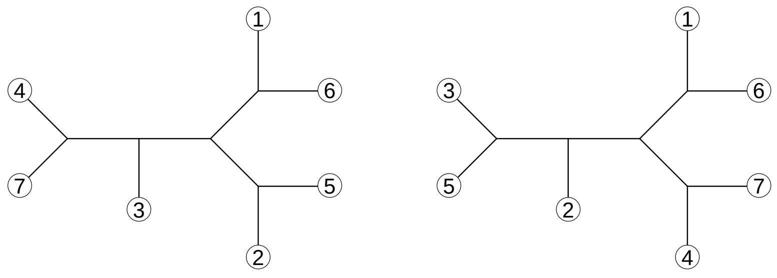

FIGURE 2. Two further binary trees that display the collections $\mathcal{F}_1 = \{123|47, 45|16, 67|25, 34|12\}$ and $\mathcal{F}_2 = \{123|47, 45|16, 67|25, 35|12\}$, respectively.

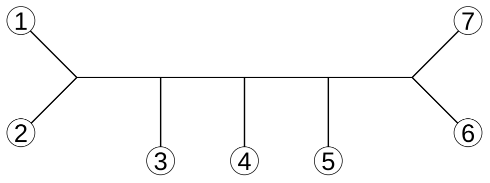

FIGURE 3. The binary tree that is defined by the collection $\{12|35, 24|57, 13|47, 34|56, 15|67\}$.

Amalgamating an excess-free collection of trees (AEFT)

```
Input:  An excess-free set $\mathcal{F}$ of binary phylogenetic trees
Output:
  --Either a binary phylogenetic tree that is defined by $\mathcal{F}$
  --or a statement that $\mathcal{F}$ is not definitive.
For each $\{T_1, T_2\} \in \binom{\mathcal{F}}{2}$ do
  $\mathit{Ins}[T_1, T_2] \leftarrow \mathcal{L}(T_1) \cap \mathcal{L}(T_2)$
  If $\big|\mathit{Ins}[T_1, T_2]\big| \geq 4$ then output non-definitive
end for
$\mathcal{A} \leftarrow \mathcal{F}$; $\mathcal{B} \leftarrow \emptyset$
While $\mathcal{A} \neq \emptyset$ and $(|\mathcal{A}| > 1$ or $\mathcal{B} \neq \emptyset)$ do
  Choose $T_1 \in \mathcal{A}$
  If there exists $T_2 \in \mathcal{B}$ with $\big|\mathit{Ins}[T_1, T_2]\big| = 3$ then
    Choose such $T_2 \in \mathcal{B}$
    Compute some tree $T$ that displays both $T_1$ and $T_2$
    If $\{T_1, T_2\}$ does not define $T$ then output non-definitive
    For each $T_3 \in \mathcal{A} \cup \mathcal{B} - \{T_1, T_2\}$ do
      $\mathit{Ins}[T, T_3] \leftarrow \mathit{Ins}[T_1, T_3] \cup \mathit{Ins}[T_2, T_3]$
      If $\big|\mathit{Ins}[T, T_3]\big| \geq 4$ then output non-definitive
    end for
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{T\} - \{T_1\}$; $\mathcal{B} \leftarrow \mathcal{B} - \{T_2\}$
  else \\ no such $T_2$
    $\mathcal{A} \leftarrow \mathcal{A} - \{T_1\}$; $\mathcal{B} \leftarrow \mathcal{B} \cup \{T_1\}$
end while
If $|\mathcal{A}| = 1$ and $\mathcal{B} = \emptyset$ then
  Choose $T$ with $\mathcal{A} = \{T\}$ and output $T$
else \\ $\mathcal{A} = \emptyset$
  output non-definitive
end.
```

Dyadic tree construction algorithm (DCT)

```
Input:  A set $\mathcal{F}$ of binary trees
Output:
  --Either a statement that $\mathcal{F}$ is incompatible
  --or a binary tree that is defined by $\mathcal{F}$
  --or a statement that $\mathcal{F}$ does not contain an excess-
    free, definitive subset $\mathcal{F}'$ with $\mathcal{L}(\mathcal{F}') = \mathcal{L}(\mathcal{F})$
    (and an additional output of a non-binary tree).
Construct $cl_2(\mathcal{Q}[\mathcal{F}])$
If $cl_2(\mathcal{Q}[\mathcal{F}])$ contains two distinct trees with the same
  leaf set then output incompatible
else
  output the Berry-Gascuel tree for $cl_2(\mathcal{Q}[\mathcal{F}])$.
  If this tree is not binary then also output the statement
    no excess-free definitive subset.
end.
```