Trees, agreement forests and treewidth

Nela Lekić

Trees, agreement forests and treewidth: combinatorial algorithms for constructing phylogenetic networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de Rector Magnificus,
Prof. dr. L.L.G. Soete,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op dinsdag 1 december 2015

door
Nela Lekić

Promotoren:
Prof. Dr. L. Stougie (Vrije Universiteit Amsterdam)
Prof. Dr. Ir. R. Peeters

Co-Promotor:
Dr. S.M. Kelk

Beoordelingscommissie:
Prof. Dr. C.P.M. van Hoesel
Prof. Dr. V. Moulton (University of East Anglia)
Dr. M.J.R. Bordewich (University of Durham)
Dr. L.J.J. van Iersel (Technical University Delft)
Dr. T. Vredeveld

This research has been funded by NWO.

Acknowledgement

Nederlandse samenvatting

Contents

1 Introduction **7**
- 1.1 Definitions . . . 10
- 1.2 Related work . . . 14
- 1.3 Structure of the thesis and contribution . . . 15

2 Cyclekiller... **17**
- 2.1 An improved bound on the size of reduced instances of MH . . . 19
- 2.2 An approximation-preserving reduction from MH to DFVS . . . 21
- 2.3 An approximation-preserving reduction from DFVS to MH . . . 26

3 ...is a practical approximation algorithm **32**
- 3.1 The algorithm for binary trees . . . 33
- 3.2 Experiments and discussion . . . 35

4 ...that can be generalized to nonbinary situations **38**
- 4.1 Nonbinary MAF . . . 40
- 4.2 Approximating nonbinary MAAF . . . 43
- 4.3 Experiments . . . 50

5 ...but not to instances with three trees **53**
- 5.1 Guessing the AAF . . . 55
- 5.2 Canonical networks . . . 57
- 5.3 Reconstructing a canonical network . . . 62
- 5.4 An Example of constructing a network from its description . . . 67

6 ...and certainly not to the Compatibility problem on unrooted trees. **74**
- 6.1 Preliminaries . . . 75
- 6.2 Main results . . . 76
- 6.3 Beyond treewidth 2 . . . 83

7 Conclusion **87**

Chapter 1

Introduction

Suppose one day you meet a biologist you want to impress and she tells you all about her four favorite species of orchids that she has been studying for years. In order to figure out the ancestral relationships among them and draw a phylogenetic tree (a diagram biologists use to show these evolutionary relationships), she has been chasing them around the mountains, took many DNA samples, later sat behind her computer, aligned them, and did many other things biologists do to arrive to their phylogenetic trees. But no matter how much she tried to eliminate any errors on her side, she kept getting two different phylogenetic trees, both equally likely. Finally she started suspecting a hybridization event took place during the evolution of these four species. Now she simply doesn't know how to represent that in a single tree.

This is a valid scenario and it might happen to you any time. In this thesis we will try to help you impress a biologist of your choice.

So let's look at her trees (T_1 and T_2 in figure 1.1). She marked her orchid species by black circles and labeled them a, b, c, d. The hypothetical ancestors she marked by black squares. One tree suggests that long time ago these orchids split into two lineages; from one later species a and b differentiated, and from the other, species c and d. The second tree paints a very different picture of what happened. It suggests that species a was the first one to split off from the common ancestor of all four of them. After that event, species d split off. Finally b and c differentiated as well.

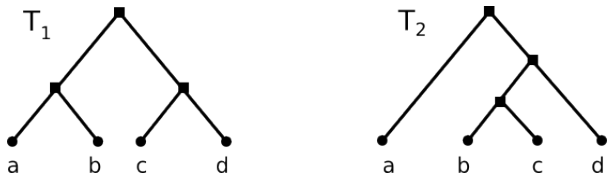

Figure 1.1: Two conflicting evolutionary histories for four species a, b, c and d

As a computer scientist or a mathematician you understand that no tree can represent all these events at the same time. You will need something more general than a tree, a directed acyclic graph, which we will here often call a network. So you draw two networks (N_1 and N_2 in figure 1.2) on four species (which you keep calling leaves instead of flowers much to her dismay). You explain why N_1 contains both trees T_1 and T_2 (in figure 1.3) and that the same argument holds for N_2. You interpret N_1 as follows: first species a differentiated from the common ancestor of all four orchids. Then d split off, after that c. Finally, ancestors of a and c exchanged their

genetic material and formed a hybrid species b. This is why species b seems genetically closest to a in T_1 and to c in T_2. A single tree simply cannot represent all that happened. Multiple trees or a network that summarizes them tell the full story.

You start interpreting N_2 when she interrupts you and says that it is unlikely N_2 happened according to Occam's razor, since it assumes more hybridization events than N_1 while explaining the same situation.

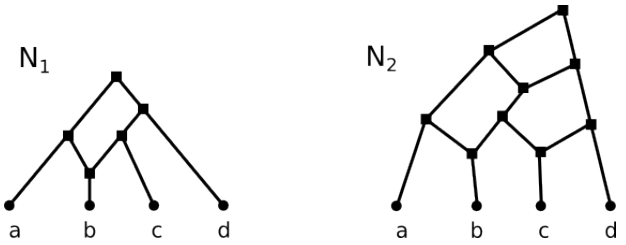

Figure 1.2: Two nontree-like (hypothetical) evolutionary histories, each of which contains both tree-like histories T_1 and T_2 at the same time.

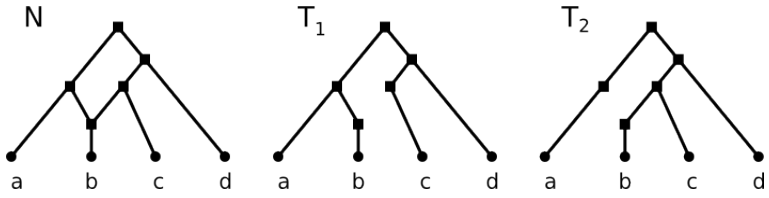

Figure 1.3: How removing appropriate edges of N_1 leads to either T_1 or T_2.

You know enough. You understand this is a combinatorial problem and you have a good idea of which parameter you need to minimize. You are interested in finding a network that contains all input trees and has a minimum number of hybridization events, or nodes with more than one incoming edge. Solving this problem requires searching for an optimal network across a vast space of possible networks. Since there are only finitely many such networks, one might think it is plausible (for a computer) to consider every single network and pick the best one, according to some measure of best. This approach is called exhaustive search and the problem with it is in its dependency on the number of the species we are considering. The bigger the number of species in a tree, the bigger the space of possible networks (and thus bigger the amount of time we need to solve the problem). In fact only a small increase in the size of the input trees leads to an exponentially big increase in the search space. The number of leaf-labeled rooted trees has been known since 1870 from classical work of Scröder [69], while the number of leaf-labeled rooted networks was counted recently in [61].

Ideally, we are interested in algorithms that don't depend this much on the varying size of the input, in the same way that adding two very big numbers is not much harder than adding two small ones. The question is, is it even possible to find such an algorithm for combining two trees into a network?

Complexity theory is a branch of mathematics that gives us a framework for classifying algorithmic problems according to their computational hardness. It distinguishes between two big classes of problems that are believed (but not proven) to be different, P and NP.

Officially these classes are defined with respect to decision problems, i.e. problems that are defined as a question about existence of a solution. A solution to such a problem is a correct

"yes" or "no" statement. The class of "easy" or polynomially solvable problems, P, is a class of decision problems for which an efficient algorithm exists, i.e. one that runs in time that is bounded by some polynomial function of the size of an instance. We say that a decision problem is in class NP, class of nondeterministically polynomially solvable problems, if for a "yes"-answer a certificate can be given that can be verified within a running time bounded by a polynomial function of the input size. The class of hardest problems, called NP-hard, contains decision problems that have an additional property: a solution of just one of these problems in polynomial time would imply solution of any other problem in NP in polynomial time. When a problem is both NP-hard and in NP we say that it is NP-complete. So problems in P are the easiest problems in NP, while the hardest problems in NP are NP-complete.

The toy problem we have just encountered was not formulated as a decision problem but an optimization problem [1], i.e. it asks for an optimum solution (fewest hybridization events) out of many possible ones. But any optimization problem has a decision version. In our case, instead of asking for a rooted network that contains two input trees and has a minimum number of "non-tree-like" nodes (vertices of in-degree higher than 1), we could chose a constant k and ask "does there exist a network that contains two input trees and has at most k non-tree-like nodes. These two formulations of the problem are equivalent because if we could solve the decision version in polynomial time, then we could (using binary search) solve the optimization version in polynomial time (and obviously if we could solve the optimization version in polynomial time we could solve the decision version in polynomial time as well).

Like many computational problems in life sciences or operations research, the problem of combining two trees into a network the way we described above, is NP-hard. The next question is of course what can be done to solve an NP-hard problem in practice. There are numerous standard approaches, but here we will encounter two of them: approximation and parameterized complexity.

Almost as soon as the concept of NP-hardness was introduced, people asked themselves if instead of finding an optimal solution to a minimization problem we can find, in polynomial time, a solution that is within a factor $(1+\epsilon)$ of an optimum for some $\epsilon > 0$ (or within a factor $(1-\epsilon)$ if we are dealing with a maximization problem). Furthermore, we would like $(1+\epsilon)$, also called approximation ratio, to be some constant rather than a function that depends on the input size of the problem. In that case we say a problem is in APX, the class of NP-hard problems for which polynomial time approximation algorithm exists that can achieve a constant worst-case approximation ratio.

Similarly to class NP we can define a class of APX-hard and APX-complete problems, which are seen as "difficult" approximation problems. If there is a polynomial-time algorithm to solve a problem to within *every* multiplicative factor of the optimum other than 1, then the problem is said to have a polynomial-time approximation scheme (PTAS). A problem is said to be APX-hard if there is a PTAS reduction (a reduction that preserves the property that a problem has a PTAS) from every problem in APX to that problem, and to be APX-complete if the problem is APX-hard and also in APX.

Sometimes however, it is not desirable to have an approximation algorithm. As we already saw, if a biologist is looking for a network to explain what exactly happened in the evolution of a small number of species, then giving as a solution a network from figure 1.2 on the right is unsatisfying, as it depicts a wrong picture of the reality if we stick to the principle of Occam's razor. In that case we want efficient exact algorithms and the key word here is parameterized

[1] To be more precise, the toy problem as we initially defined it is a construction one because we asked for a *network* with fewest reticulations, not just the minimum *number* of reticulations. But for now lets concentrate on its optimization variant, because we are not discussing complexity of that particular problem, we simply want to introduce some definitions.

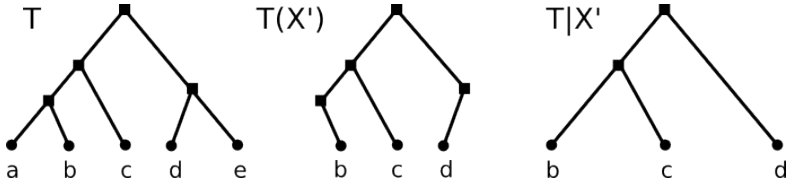

Figure 1.4: Let X be a set of taxa $\{a, b, c, d, e\}$ and X' its subset $\{b, c, d\}$. This figure shows an X-tree T, a minimal subtree of T on X' and a restriction of T to X'.

complexity.

The idea is to design an algorithm for a hard problem that runs in polynomial time with respect to the size of the instance, but exponential in the value of some chosen parameter. For example, such an algorithm can have time complexity $c^k poly(n)$, where c is some constant, n is the input size and k is the value of a parameter of the given input. For a small k this algorithm can be considered to be efficient. When such an algorithm exists, we say the problem is in FPT. See [23, 65] for an introduction to fixed parameter tractability.

For a book on complexity theory we refer a reader to [3]. Throughout the thesis we will assume the reader to be familiar with (very) basic graph theory; for a reference to basic graph theory see [20].

1.1 Definitions

Phylogenetic trees

As we have already seen in that simple example of interaction between a biologist and a mathematician in the beginning of this chapter, the evolutionary history of a set of contemporary species is often modeled as a *rooted phylogenetic tree*. Essentially, this is a rooted tree in which the leaves are bijectively labeled by the set of species and edges are directed away from the root, reflecting the direction of evolution [72]. Furthermore, the root represents the common ancestor of the set of species the tree is modeling and nodes of outdegree two or higher model the points in history at which a common ancestor of a subset of the species differentiated into two or more sublineages. The central problem in phylogenetics is to recover the topology of the "true" phylogenetic tree, given only information about the species, often DNA data. This is a challenging computational problem and has been the topic of intensive research during the last 40 years [30, 31, 74].

Without further ado, denote the set of species (or more generally, *taxa*) by X. A *rooted phylogenetic X-tree* T is a rooted tree with no nodes with indegree 1 and outdegree 1, a root with indegree 0 and outdegree 2, and leaves bijectively labeled by the elements of X. Such a tree is called *binary* if all inner nodes except the root have indegree 1 and outdegree 2. When some nodes have outdegree higher than 2, we speak of *nonbinary* trees. We will encounter nonbinary trees in Chapter 4 and we postpone definitions of nonbinary concepts until then. For now we only talk of binary setting and we will refer to a rooted binary phylogenetic X-tree as a *tree* for short. The edges of any rooted tree should be seen as being directed away from the root even though we will commonly draw them instead of arcs.

The set of leaves of T is denoted as $\mathcal{L}(T)$. We identify each leaf with its label. In the course of this thesis, different types of subtrees play an important role. Let T be a rooted phylogenetic X-tree and X' a subset of $\mathcal{L}(T)$. The minimal rooted subtree of T that connects all leaves in X' is denoted by $T(X')$. Furthermore, the tree obtained from $T(X')$ by suppressing all vertices with

indegree and outdegree both equal to 1 is the *restriction of* T *to* X' and is denoted by $T|X'$. See Figure 1.4.

Let T be a rooted phylogenetic X-tree and S a rooted phylogenetic X'-tree for some $X' \subseteq X$. We say that S is a *pendant subtree* of T if it is a subtree that can be detached from T by deleting a single edge. For a set $\mathcal{T}$ of phylogenetic X-trees and $X' \subseteq X$, a *common pendant subtree* of $\mathcal{T}$ is a rooted phylogenetic X'-tree that is a pendant subtree of each tree in $\mathcal{T}$.

For $n \geq 2$, an *n-chain* of T is an n-tuple $(a_1, a_2, \ldots, a_n)$ of elements of $\mathcal{L}(T)$ such that the parent of a_1 is either the same as the parent of a_2 (see Figure 1.5 left) or the parent of a_1 is a child of the parent of a_2 (see Figure 1.5 right) and, for each $i \in \{2, 3, \ldots, n-1\}$, the parent of a_i is a child of the parent of a_{i+1}. The subgraph induced by $a_1, a_2, \ldots, a_n$ and their parents is called a *caterpillar*. A *common chain* of a set $\mathcal{T}$ of trees is a maximal tuple $(x_1, x_2, ..., x_q)$ that is a chain of each tree in $\mathcal{T}$.

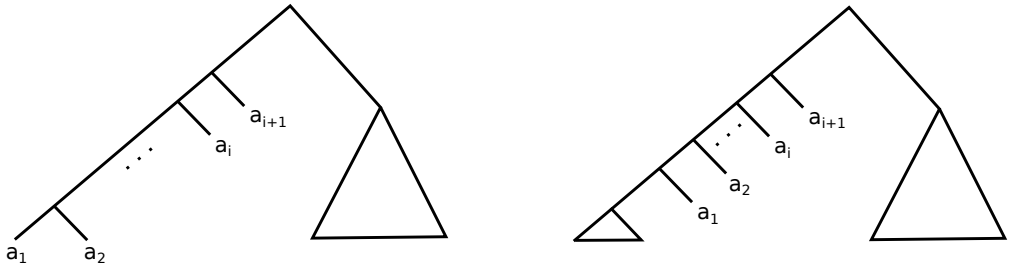

Figure 1.5: Example of n-chains

However, our understanding of evolutionary mechanisms has deepened and there is growing awareness that evolution is not always treelike [4]. In particular, due to *reticulate phenomena* such as hybridizations, recombinations and horizontal gene transfer the evolution of a set of species is sometimes better modeled as a network rather than a tree [63]. Trees remain still equally relevant and they are often used as building blocks for networks.

Hybridization networks

A *rooted phylogenetic network* is essentially a generalisation of phylogenetic trees to directed acyclic graphs (DAGs). In such graphs nodes with indegree two or higher, known as *reticulations*, represent the points at which two or more lineages merge, rather than diversify.

Formally, a *rooted phylogenetic network* N on a set X is a rooted acyclic directed graph, which has a single root of outdegree 2, has no vertices with indegree and outdegree both 1, and in which the vertices of outdegree 0 are bijectively labelled by the elements of X. A phylogenetic network is *binary* if all vertices have indegree and outdegree at most 2 and every vertex with indegree 2 has outdegree 1. Intuitively we say that a phylogenetic network N *displays* a phylogenetic tree T when all of the ancestral relationships visualized by T are visualized by N. A precise definition will be given shortly.

Following the publication of several seminal articles in 2004-5 (e.g. [5, 6]) there has been considerable research interest in the biologically-inspired question from the beginning of this chapter. Namely, given two rooted phylogenetic trees T_1 and T_2 on the same set of taxa X, what is the minimum number of reticulations required by a phylogenetic network N on X which displays both T_1 and T_2? This value is often called the *hybridization number* in the literature, and when addressing this specific problem the term *hybridization network* is often used instead of

the more general term phylogenetic network. For the purpose of consistency we will henceforth use the term hybridization network. We now list some necessary definitions.

For each vertex v of N, we denote by $d^-(v)$ and $d^+(v)$ its indegree and outdegree respectively. If (u, v) is an arc of N, we say that u is a *parent* of v and that v is a *child* of u. Furthermore, if there is a directed path from a vertex u to a vertex v, we say that u is an *ancestor* of v and that v is a *descendant* of u.

A vertex of indegree greater than one represents an evolutionary event in which lineages combined, such as a hybridization, recombination or horizontal gene transfer. We call these vertices *hybridization vertices*. To quantify the number of hybridization events, the hybridization number of a (nonbinary) hybridization network N is given by

$$h(N) = \sum_v (d^-(v) - 1)$$

where v goes over all vertices of N except the root. In a binary network, $h(N)$ is simply the number of vertices with indegree 2. Observe that $h(N) = 0$ if and only if N is a tree.

Let N be a hybridization network on X and T a rooted binary phylogenetic X'-tree with $X' \subseteq X$. We say that T is *displayed* by N if T can be obtained from N by deleting vertices and edges and suppressing vertices with $d^+(v) = d^-(v) = 1$ (or, in graph-theoretic words, if a subdivision of T is a subgraph of N).

The problem MinimumHybridization, abbreviated MH, is to compute the hybridization number of two rooted binary phylogenetic X-trees T_1 and T_2, which is defined as

$$h(T_1, T_2) = \min\{h(N) : N \text{ is a hybridization network that displays } T_1 \text{ and } T_2\},$$

i.e., the minimum number of hybridization events necessary to display two rooted binary phylogenetic trees.

This problem can be formulated as an optimization problem in the obvious way.

Problem: MinimumHybridization
Instance: Two rooted binary phylogenetic X-trees T_1 and T_2.
Solution: A hybridization network N that displays T_1 and T_2.
Objective: Minimize $h(N)$.

If N is a hybridization network that displays T_1 and T_2, then there also exists a binary hybridization network N' that displays T_1 and T_2 such that $h(N) = h(N')$ [43, Lemma 3]. Hence, we restrict our analysis to binary hybridization networks (even when we speak of nonbinary trees) and will not emphasize this again.

There is an interesting characterization of the hybridization number for two trees by another phylogenetic problem called Maximum Acyclic Agreement Forest, or MAAF for short. In this problem one is required to cut two input trees into common components, such that the number of components is minimized and there are no cyclical dependencies between them. In the next section we turn our attention to agreement forests and formally introduce this problem.

A reader interested in the topic of phylogenetic networks, concepts and models that emerge in this area and their applications, is referred to [38, 40, 64, 71].

Agreement Forests

Let T_1 and T_2 be two rooted binary phylogenetic X-trees. Consider now a set of labels $X \cup \{\rho\}$, where ρ is an additional leaf that we attach to the root of both trees T_1 and T_2. Or more precisely,

since a root of any binary phylogenetic tree has outdegree two and indegree zero, we attach an edge to the root and label its end by ρ. See figure 1.6.

A partition $\mathcal{F} = \{\mathcal{L}_\rho, \mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k\}$ of $X \cup \{\rho\}$ is an *agreement forest* for T_1 and T_2 if $\rho \in \mathcal{L}_\rho$ and the following conditions are satisfied:

(1) for all $i \in \{\rho, 1, 2, \ldots, k\}$, we have $T_1|\mathcal{L}_i \cong T_2|\mathcal{L}_i$, and

(2) the trees in $\{T_1(\mathcal{L}_i) : i \in \{\rho, 1, 2, \ldots, k\}\}$ and $\{T_2(\mathcal{L}_i) : i \in \{\rho, 1, 2, \ldots, k\}\}$ are vertex-disjoint subtrees of T_1 and T_2, respectively.

In the definition above, the notation $\cong$ is used to denote a graph isomorphism that preserves leaf-labels.

Note that, even though an agreement forest is formally defined as a partition of the leaves $\mathcal{F} = \{\mathcal{L}_\rho, \mathcal{L}_1, \ldots, \mathcal{L}_k\}$, it is often useful to see it as a collection of trees $\mathcal{F} = \{F_\rho, F_1, \ldots, F_k\}$ where $F_i = T_1|\mathcal{L}_i$ (or equivalently $F_i = T_2|\mathcal{L}_i$) for $i \in \{\rho, 1, 2, \ldots, k\}$. So, intuitively, an agreement forest for T_1 and T_2 can be seen as a collection of trees that can be obtained from either of T_1 and T_2 by deleting a set of edges and subsequently "cleaning up" by deleting unlabeled vertices and suppressing indegree-1 outdegree-1 vertices (see figure 1.6). Therefore, we often refer to the elements of an agreement forest as *components*. The *size* of an agreement forest $\mathcal{F}$ is defined as its number of elements (components) and is denoted by $|\mathcal{F}|$. A natural optimization problem arises:

Problem: Maximum Agreement Forest (MAF)
Instance: Two rooted binary phylogenetic X-trees T_1 and T_2.
Solution: An agreement forest $\mathcal{F}$ for T_1 and T_2.
Objective: Minimize $|\mathcal{F}| - 1$

The characterization of the hybridization number $h(T_1, T_2)$ in terms of agreement forests that we mentioned in the previous section requires an additional condition. Let $\mathcal{F} = \{\mathcal{L}_\rho, \mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k\}$ be an agreement forest for T_1 and T_2. Let $G_\mathcal{F}$ be the directed graph that has vertex set $\mathcal{F}$ and an edge $(\mathcal{L}_i, \mathcal{L}_j)$ if and only if $i \neq j$ and at least one of the two following conditions holds

(1) the root of $T_1(\mathcal{L}_i)$ is an ancestor of the root of $T_1(\mathcal{L}_j)$ in T_1;

(2) the root of $T_2(\mathcal{L}_i)$ is an ancestor of the root of $T_2(\mathcal{L}_j)$ in T_2.

The graph $G_\mathcal{F}$ is called the *inheritance graph* associated with $\mathcal{F}$. We call $\mathcal{F}$ an *acyclic agreement forest* for T_1 and T_2 if $G_\mathcal{F}$ has no directed cycles. If $\mathcal{F}$ contains the smallest number of elements (components) over all acyclic agreement forests for T_1 and T_2, we say that $\mathcal{F}$ is a *maximum acyclic agreement forest* for T_1 and T_2. Note that such a forest is called a *maximum* acyclic agreement forest, even though one *minimizes* the number of elements, because in some sense the "agreement" is maximized. (Also note that acyclic agreement forests were called *good* agreement forests in [5].)

We define $m_a(T_1, T_2)$ to be the number of elements of a maximum acyclic agreement forest for T_1 and T_2 minus one. Also the problem of computing $m_a(T_1, T_2)$ has an optimization counterpart:

Problem: Maximum Acyclic Agreement Forest (MAAF)
Instance: Two rooted binary phylogenetic X-trees T_1 and T_2.
Solution: An acyclic agreement forest $\mathcal{F}$ for T_1 and T_2.
Objective: Minimize $|\mathcal{F}| - 1$.

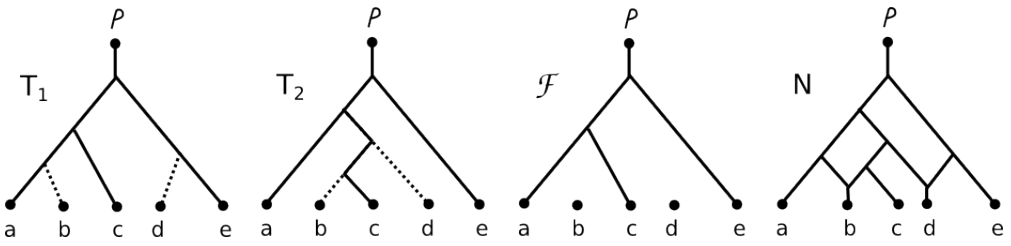

Figure 1.6: Two phylogenetic trees, T_1 and T_2, an acyclic agreement forest $\mathcal{F}$ for T_1 and T_2 with three components and its inheritance graph (which will later be drawn instead of that network). Forest $\mathcal{F}$ can be obtained from either of T_1 and T_2 by deleting the dashed edges.

We minimize $|\mathcal{F}|-1$, rather than $|\mathcal{F}|$, following [11], because $|\mathcal{F}|-1$ corresponds to the number of edges one needs to remove from either of the input trees to obtain $\mathcal{F}$ (after "cleaning up") and because of the relation we describe below between this problem and MinimumHybridization. Nevertheless, it can be shown that, from an approximation perspective, it does not matter whether one minimizes $|\mathcal{F}|$ or $|\mathcal{F}|-1$ (which is not obvious).

Theorem 1.1. *[5, Theorem 2] Let T_1 and T_2 be two rooted binary phylogenetic X-trees. Then*

$$h(T_1, T_2) = m_a(T_1, T_2).$$

1.2 Related work

There are many types of problems that arise in the area of phylogenetic networks, but they can all be summarized as having a goal of reconstructing a network from the available data. The variety of problems come (in part) from variety of different kinds of data. In this thesis we mainly look at the problem of constructing a phylogenetic network from trees, but there are also techniques which construct networks from clusters [57,58], triplets [41,42], quartets [28,88], networks [35–37,47], sequences [26,48–50,67] and distances between those sequences [27,86].

A standard reference book on this topic is [74], in which the authors set the mathematical foundations for phylogenetics. A more recent text [24] focuses on different ways of encoding phylogenetic networks and how they are related to each other. For a more practical flavour there is [38]. For algorithms on ancestral recombination graphs see [33].

Most of the problems on constructing a network on available type of data are NP-complete. In this thesis we mainly focus on MH. The ultimate goal for this problem is to develop algorithms that can cope with many input trees and nonbinary input trees [63] and to take different causes of incongruence into account, see e.g. [89]. However, until recently most algorithmic research has focused on the simplest possible case: two input trees, both binary. Unfortunately even this version of the problem is NP-hard and APX-hard [11].

Fortunately the binary two-tree problem is fixed parameter tractable. In [10], Bordewich and Semple proved that MH for two rooted binary trees is FPT and gave an algorithm with running time $O((28k)^k + n^3)$. This result was established via kernelization - a common technique in parameterized complexity where one wants to prove that a kernel, bounded in size as a function of some parameter, can be found in polynomial time. The theoretical state of the art is an algorithm based on bounded-search with running time $O(3.18^k n)$ [82] with $n = |X|$ and k the reticulation number.

A standard approach in the network literature is to look for algorithms parameterized by the number of reticulations of an optimal network. On this front, a variety of increasingly sophisticated algorithms have been developed [9, 10, 16, 18, 70, 83, 85]. These show that for many practical instances of MH the problem can be efficiently solved.

In the nonbinary setting, MH is of course NP-hard and APX-hard. While there do exist efficient FPT algorithms for the binary variant of the problem, the nonbinary variant of the problem has received comparatively little attention, although that too is FPT [60, 68]. In both of these results, the practical applicability of the FPT algorithms is limited to instances of small or moderate size, for larger instances approximation algorithms are required. An example of an approach via kernalization is [44] where the authors give two algorithms for MH on multiple nonbinary trees.

In the last chapter of this thesis we look at quite a different problem. There we are given unrooted trees, not always on the same set of taxa, and the questions is weather there exists a supertree that displays all the partial trees. When such a tree exists we say that the instance is compatible. Even though this is an NP hard problem, it is fixed parameter tractable [12]. That result is purely theoretic and no practical FPT algorithm is known. There exists however a number of useful characterizations.

There is a well known characterization of compatibility of undirected multi-state characters [13], and many other authors have since found interesting characterizations in terms of chordal graphs [34, 62, 73]. As a counterpart to Buneman's triangulation based characterization of undirected multi-state characters [13], Vakati and Fernández-Baca [79] gave a triangulation based characterization of the compatibility of unrooted phylogenetic trees in terms of the existence of a specific kind of triangulation in a structure known as the display graph. In [80] the same authors characterize compatibility of unrooted trees as a chordal graph sandwich problem in a edge label intersection graph, while in [32] authors define a quartet graph and give a characterization in terms of edge coloring.

Similar work on unrooted trees, but then with a goal of constructing an unrooted network that contains the input trees rather than solving the compatibility problem, was done by Gambette, Berry and Paul. In [28] for example they show how to adapt some of the well known results on constructing rooted network from triplets on constructing unrooted networks from quartets. In [53] the authors give a polynomial time algorithm but then with a restriction on the output: they construct level-1 networks from quartets.

Finally, building on the idea from [12], the authors in [55] show the power of monadic second order logic applied to phylogenetics. They observe that many (otherwise NP-hard to compute) measures of (dis)similarity of phylogenetic trees can be bounded and translated (via agreement forests) to a bounded display graph, which then leads to fixed parameter tractability.

1.3 Structure of the thesis and contribution

We start with an approximability result in Chapter 2. There we show that the problem MH has a constant factor polynomial-time approximation if and only if the problem of computing a minimum-size feedback vertex set in a directed graph (DFVS) has a constant factor polynomial-time approximation. Despite considerable attention from the combinatorial optimization community, it remains to this day unknown whether such an algorithm exists for DFVS. The proof of this result leads to a new insight into where the hardness of the problem lies, and based on this we develop a practically efficient algorithm in Chapter 3 and give its generalization to biologically more relevant situations in Chapter 4.

Chapter 2 is based on the paper *Cycle killer...qu'est-ce que c'est? On the comparative ap-*

proximability of hybridization number and directed feedback vertex set. It was written together with Steven Kelk, Leo van Iersel, Simone Linz, Celine Scornavacca and Leen Stougie in 2011. It is published in SIAM Journal on Discrete Mathematics. The follow-up paper written with Leo van Iersel, Steven Kelk and Celine Scornavacca, *A practical approximation algorithm for solving massive instances of hybridization number for binary and nonbinary trees* was first presented at WABI conference in 2012 and later published in BMC Bioinformatics. This is given in Chapter 3.

Chapter 4 is based on the third paper in the Cycle Killer series, *Approximation algorithms for nonbinary agreement forests*, coauthored with Leo van Iersel, Steven Kelk and Leen Stougie, published in SIAM Journal on Discrete Mathematics. This paper (and the corresponding chapter) are a little bit more than just a generalization of the previous work to nonbinary trees. We also consider the nonbinary version of problem MAF and give two algorithms for it: an approximation and a parameterized exact algorithm.

In Chapter 5 we consider the same problem, MH, but now with three binary input trees. Considering three instead of two trees is challenging because the similarity with MAAF weakens and all algorithms for two trees relied on it. Furhermore, all previous algorithms for the multiple tree hybridization problem relied on a brute force search through all possible underlying network topologies, leading to (theoretically unpleasant) running times with towers of exponents (i.e. not $O(c^k poly(n))$ for any c). We improve that by giving a $O(c^k poly(n))$ running time. This chapter is based on paper currently under review, *Hybridization Number on Three Trees*, written with Leo van Iersel, Steven Kelk, Chris Whidden and Norbert Zeh.

Chapter 6 has quite a different flavor than the rest of the thesis. Though still phylogenetically motivated, we turn our attention to unrooted trees. Here the goal is to puzzle the partial unrooted trees together into a single tree that contains all the topologies of the partial trees. We build on work of Bryant and Lagergren [12] and give a condition for when the partial trees do belong to a single larger tree. This work was done together with Alexandar Grigoriev and Steven Kelk and published in Journal of Graph Algorithms and Applications. It was also presented at AlCoB conference in 2014.

Chapter 2

Cyclekiller...

In this chapter we will show that the problem of computing the hybridization number of two rooted binary phylogenetic trees on the same set of taxa (MH) has a constant factor polynomial-time approximation if and only if the problem of computing a minimum-size feedback vertex set in a directed graph (DFVS) has a constant factor polynomial-time approximation.

As we saw in the introduction chapter, computing the hybridization number of two trees T_1 and T_2 is essentially identical to the problem of cutting T_1 and T_2 into as few vertex-disjoint subtrees as possible such that the subtrees of T_1 are isomorphic to the subtrees of T_2 and - critically - a specific "reachability" relation on these subtrees is acyclic. The latter condition seems to be the core of the issue, because without this condition the problem would be no different to the problem of computing the rSPR distance, which as previously mentioned seems to be comparatively tractable. The various FPT algorithms for computing hybridization number deal with the unwanted cycles in the reachability relation in a variety of ways but all resort to some kind of brute force analysis to optimally avoid (e.g. [70]) or break (e.g. [16, 83]) them.

In this chapter we demonstrate why it is so difficult to deal with the cycles. Problem DFVS belongs to Karp's famous 1972 list of 21 NP-complete problems [52] and is also known to be APX-hard [51]. However, despite almost forty years of attention it is still unknown whether DFVS permits a constant approximation ratio i.e. whether it is in APX.

By coupling the approximability of MH to DFVS we show that MH is just as hard as a problem that has so far eluded the entire combinatorial optimization community. Specifically, we show that for every constant $c > 1$ and every $\epsilon > 0$ the existence of a polynomial-time c-approximation for MH would imply a polynomial-time $(c+\epsilon)$-approximation for DFVS. In the other direction we show that, for every $c > 1$, the existence of a polynomial-time c-approximation for DFVS would imply a polynomial-time $6c$-approximation for MH. In other words: DFVS is in APX if and only if MH is in APX. Hence a constant factor approximation algorithm for either problem would be a major breakthrough in theoretical computer science.

The structure of this chapter is as follows. We start with some definitions and describe two types of reductions that were used to show that MH is fixed parameter tractable. We will need these in Section 2.1, where we show an improved bound on the sizes of reduced instances of MH. Subsequently, we use these results to show an approximation-preserving reduction from MH to DFVS in Section 2.2 and an approximation-preserving reduction from DFVS to MH in Section 2.3. We conclude with a summary and some open problems.

After establishing the NP-hardness of MH, the same authors showed that this problem is also fixed parameter tractable [10]. They show how to reduce a pair of rooted binary phylogenetic X-trees T_1 and T_2, such that the number of leaves of the reduced trees is bounded by $14h(T_1, T_2)$,

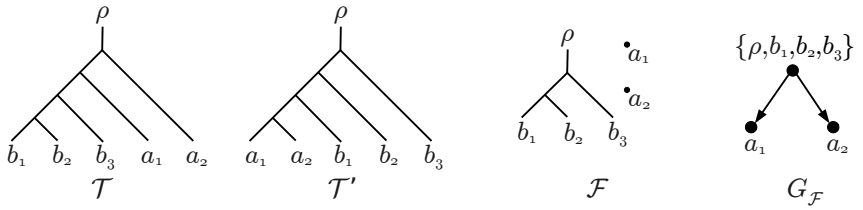

Figure 2.1: Two input trees T_1 and T_2, an agreement forest $\mathcal{F}$ for T_1 and T_2 and the inheritance graph $G_{\mathcal{F}}$. The trees have two common chains: (a_1, a_2) and (b_1, b_2, b_3). In the agreement forest $\mathcal{F}$, chain (a_1, a_2) is atomized while chain (b_1, b_2, b_3) survives. The agreement forest $\mathcal{F}$ is acyclic because $G_{\mathcal{F}}$ is acyclic.

where $h(T_1, T_2)$ is the hybridization number of T_1 and T_2. Since a brute-force algorithm can be used to solve the reduced instance, this leads to a fixed parameter tractable algorithm.

Let $A = (a_1, a_2, \ldots, a_n)$ be an n-chain that is common to two rooted binary phylogenetic X-trees T_1 and T_2 with $n \geq 2$, and let $\mathcal{F}$ be an acyclic agreement forest for T_1 and T_2. We say that A *survives* in $\mathcal{F}$ if there exists an element in $\mathcal{F}$ that is a superset of $\{a_1, a_2, \ldots, a_n\}$, while we say that A is *atomized* in $\mathcal{F}$ if each element in $\{a_1, a_2, \ldots, a_n\}$ is a singleton in $\mathcal{F}$ (see Figure 2.1). Similarly, for a common pendant subtree T of T_1 and T_2, we say that T survives in $\mathcal{F}$ if there is an element of $\mathcal{F}$ that is a superset of the label set of T.

The following lemma basically shows that we can reduce subtrees and chains. It differs slightly from the corresponding lemma in [10] because we consider approximations while Bordewich and Semple considered only optimal solutions in that paper.

Lemma 2.1. *[10, Lemma 3.1] Let $\mathcal{F}$ be an acyclic agreement forest for two trees T_1 and T_2. Then there exists an acyclic agreement forest $\mathcal{F}'$ for T_1 and T_2 with $|\mathcal{F}'| \leq |\mathcal{F}|$ such that*

1. *every common pendant subtree of T_1 and T_2 survives in $\mathcal{F}'$ and*
2. *every common n-chain of T_1 and T_2, with $n \geq 3$, either survives or is atomized in $\mathcal{F}'$.*

Moreover, $\mathcal{F}'$ can be obtained from $\mathcal{F}$ in polynomial time.

Proof. Follows from the proof of [10, Lemma 3.1]. There are two differences with [10, Lemma 3.1]. Firstly, our result is slightly simpler because we consider two unweighted trees T_1 and T_2, while the authors of [10] allow the unreduced trees T_1 and T_2 to already have weights on 2-chains. Secondly, [10, Lemma 3.1] only shows the result for optimal agreement forests. However, a careful analysis of the proof of [10, Lemma 3.1] shows that it can also be used to prove this lemma. □

We are now ready to formally describe the aforementioned tree reductions. Let T_1 and T_2 be two rooted binary phylogenetic X-trees, P a set that is initially empty and $w : P \to \mathbb{Z}^+$ a weight function on the elements in P.

Subtree Reduction. Replace any maximal pendant subtree with at least two leaves that is common to T_1 and T_2 by a single leaf with a new label.

Chain Reduction. Replace any maximal n-chain $(a_1, a_2, \ldots, a_n)$, with $n \geq 3$, that is common to T_1 and T_2 by a 2-chain with new labels a and b. Moreover, add a new element (a, b) with weight $w(a, b) = n - 2$ to P.

Let S_1 and S_2 be two rooted binary phylogenetic X'-trees that have been obtained from T_1 and T_2 by first applying subtree reductions as often as possible and then applying chain reductions as often as possible. We call S_1 and S_2 the *reduced tree pair* with respect to T_1 and T_2. Note

that a reduced tree pair always has an associated set P that contains one element for each chain reduction applied. Note that S_1 and S_2 are unambiguously defined (up to the choice of the new labels) because maximal common pendant subtrees do not overlap and maximal common chains do not overlap. Moreover, applications of the chain reduction can not create any new common pendant subtrees with at least two leaves. Hence, it is not necessary to apply subtree reductions again after the chain reductions.

Recall that every common n-chain, with $n \geq 3$, either survives or is atomized (Lemma 2.1). In S_1 and S_2, such chains have been replaced by weighted 2-chains. Therefore, we are only interested in acyclic agreement forests for S_1 and S_2 in which these weighted 2-chains either survive or are atomized. We therefore introduce a third notion of an agreement forest. Recall that P is the set of reduced (i.e. weighted) 2-chains. We say that an agreement forest $\mathcal{F}$ for S_1 and S_2 is *legitimate* if it is acyclic and every chain $(a, b) \in P$ either survives or is atomized in $\mathcal{F}$. Let $\mathcal{F}$ be an agreement forest for S_1 and S_2. The *weight* of $\mathcal{F}$, denoted by $w(\mathcal{F})$, is defined to be

$$w(\mathcal{F}) = |\mathcal{F}| - 1 + \sum_{(a,b)\in P:\ (a,b) \text{ is atomized in } \mathcal{F}} w(a, b).$$

Lastly, we define $f(S_1, S_2)$ to be the minimum weight of a legitimate agreement forest for S_1 and S_2.

Then, the following lemma says that computing the hybridization number of T_1 and T_2 is equivalent to computing the minimum weight of a legitimate agreement forest for S_1 and S_2. The second part of the lemma is necessary to show that an approximation to a reduced instance S_1 and S_2 can be used to obtain an approximation to the original instance T_1 and T_2.

Lemma 2.2. *[10, Proposition 3.2] Let T_1 and T_2 be a pair of rooted binary phylogenetic X-trees and let S_1 and S_2 be the reduced tree pair with respect to T_1 and T_2. Then*

(i) $h(S_1, S_2) \leq f(S_1, S_2) = h(T_1, T_2)$ and

(ii) given a legitimate agreement forest $\mathcal{F}_S$ for S_1 and S_2, we can find, in polynomial time, an acyclic agreement forest $\mathcal{F}$ for T_1 and T_2 such that $|\mathcal{F}| - 1 = w(\mathcal{F}_S)$.

The fixed parameter tractability of MH now follows from the next lemma, which bounds the number of leaves in a reduced tree pair.

Lemma 2.3. *[10, Lemma 3.3] Let T_1 and T_2 be two rooted binary phylogenetic X-trees, S_1 and S_2 the reduced tree pair with respect to T_1 and T_2, and X' the label set of S_1 and S_2. If $h(T_1, T_2) > 0$, then $|X'| < 14h(T_1, T_2)$.*

We show in Section 2.1 that the reduced trees have at most $9h(T_1, T_2)$ leaves. This improved bound will be important in the approximation-preserving reductions we give later in the chapter.

2.1 An improved bound on the size of reduced instances of MH

We start with some definitions and an intermediate result. The bound on the size of the reduced instance will be proven in Theorem 2.5.

An *r-reticulation generator* (for short, *r-generator*) is defined to be a directed acyclic multigraph with a single vertex of indegree 0 and outdegree 1 (which we can think of as being labelled by ρ), precisely r reticulation vertices (indegree 2 and outdegree at most 1), and apart from that only vertices of indegree 1 and outdegree 2 [57]. The *sides* of an r-generator are its edges (the

edge sides) and its vertices of indegree-2 and outdegree-0 (the *node sides*). Adding a set of labels L to an edge side (u, v) of an r-generator involves subdividing (u, v) to a path of $|L|$ internal vertices and, for each such internal vertex w, adding a new leaf w', an edge (w, w'), and labeling w' with some taxon from L (such that L bijectively labels the new leaves). On the other hand, adding a label l to a node side v consists of adding a new leaf y, an edge (v, y) and labeling y with l.

Lemma 2.4. *Let T_1 and T_2 be two rooted binary phylogenetic X-trees with no common pendant subtrees with at least 2 leaves and let H be a hybridization network that displays T_1 and T_2 with a minimum number of hybridization vertices. Then the network H' obtained from H by deleting all $|X|$ leaves and suppressing each resulting vertex v with $d^+(v) = d^-(v) = 1$ is an $h(H)$-generator.*

Proof. By construction, H' contains the same number of hybridization vertices as H. Additionally, by the definition of a binary hybridization network, no vertex has indegree 2 and outdegree greater than 1, indegree greater than 2, or indegree and outdegree both 1. Now, we claim that H' does not have any vertex with indegree 1 and outdegree 0. To see that this holds, suppose that there exists a vertex v in H' such that $d^-(v) = 1$ and $d^+(v) = 0$. Then v has two children in H. Since $d^+(v) = 0$ in H', no hybridization vertex can be reached by a directed path from v in H. This means that the subnetwork of H rooted at v is actually a rooted tree, contradicting the fact that T_1 and T_2 do not have any common pendant subtree with two or more leaves. We may thus conclude that H' conforms to the definition of an $h(H)$-generator. □

Conversely, by inverting the operations of suppression and deletion, H can be obtained from the $h(H)$-generator H' associated with H by adding leaves to its sides (in the sense described at the start of this section). This relies on the intuitive fact that modulo leaves and suppression the $h(H)$-generator obtained in Lemma 2.4 has essentially the same topology as H. A similar technique was described in [57] in a somewhat different context.

Theorem 2.5. *Let T_1 and T_2 be two rooted binary phylogenetic X-trees and S_1 and S_2 the reduced tree pair on X' with respect to T_1 and T_2. If $h(T_1, T_2) > 0$, then $|X'| < 9h(T_1, T_2)$.*

Proof. Let H' be the $h(H)$-generator that is associated with a hybridization network H for S_1 and S_2 whose number of hybridization vertices is minimized, i.e., $h(H) = h(S_1, S_2)$. By definition, H' has the following vertices:

- $r = h(H)$ reticulations; in particular r_0 reticulations with indegree 2 and outdegree 0 and r_1 reticulations with indegree 2 and outdegree 1,
- s vertices with indegree 1 and outdegree 2, and
- one vertex labelled ρ with indegree 0 and outdegree 1.

The total indegree of H' is $2r_0 + 2r_1 + s$. The total outdegree of H' is $r_1 + 2s + 1$. Hence, $2r_0 + 2r_1 + s = r_1 + 2s + 1$ implying $s = 2r_0 + r_1 - 1$. Moreover, the total number of edges of H', $|E(H')|$, equals the total indegree and, therefore,

$$|E(H')| = 2r_0 + 2r_1 + s = 2r_0 + 2r_1 + 2r_0 + r_1 - 1 = 4r_0 + 3r_1 - 1. \tag{2.1}$$

Note that for each of the r_0 node sides v in H' the child of v in H is a single leaf. Moreover, each edge side in H' cannot correspond to a directed path in H that consists of more than three edges since, otherwise, S_1 and S_2 would have a common n-chain, with $n \geq 3$. Thus, H can have

at most two leaves per edge side of H' and one leaf per node side of H'. Thus, the total number of leaves $|X'|$ of H is bounded by

$$\begin{aligned}|X'| &\leq 2|E(H')| + r_0 \\ &= 2(4r_0 + 3r_1 - 1) + r_0 \\ &= 9r_0 + 6r_1 - 2 \\ &\leq 9r - 2 \\ &< 9h(S_1, S_2) \\ &\leq 9h(T_1, T_2),\end{aligned}$$

where the last inequality follows from Lemma 2.2. □

2.2 An approximation-preserving reduction from MH to DFVS

We start by proving the following theorem, which refers to wDFVS, the *weighted* variant of DFVS where every vertex is attributed a weight and the weight of a feedback vertex set is simply the sum of the weights of its constituent vertices. Later in the section we will prove a corresponding result for DFVS.

Theorem 2.6. *If, for some $c \geq 1$, there exists a polynomial-time c-approximation for* wDFVS*, then there exists a polynomial-time $6c$-approximation for* MH*.*

Throughout this section, let T_1 and T_2 be two rooted binary phylogenetic X-trees, and let S_1 and S_2 be the reduced tree pair on X' with respect to T_1 and T_2. Using Lemma 2.1, we assume throughout this section without loss of generality that T_1 and T_2 do not contain any common pendant subtrees with at least two leaves. Thus, the reduced tree pair S_1 and S_2 can be obtained from T_1 and T_2 by applying the chain reduction only.

Before starting the proof, we need some additional definitions and lemmas. We say that a common chain (a, b) of S_1 and S_2 is a *reduced chain* if it is not a common chain of T_1 an T_2. Otherwise, (a, b) is an *unreduced chain*, see figure 2.1. Furthermore, a taxon $\ell \in X' \cup \{\rho\}$, is a *non-chain taxon* if it does not label a leaf of a reduced or unreduced chain of S_1 and S_2. Now, let $\mathcal{B}_S$ be the forest that exactly contains the following elements:

1. for each non-chain taxon ℓ of S_1 and S_2, a *non-chain element* $\{\ell\}$, and
2. for each reduced and unreduced chain (a, b) of S_1 and S_2, an element $\{a, b\}$.

Clearly, $\mathcal{B}_S$ is an agreement forest for S_1 and S_2, and we refer to it as a *chain forest* for S_1 and S_2. Now, obtain $\mathcal{B}_T$ from $\mathcal{B}_S$ by replacing each element in $\mathcal{B}_S$ that contains two labels of a reduced chain, say (a, b), of S_1 and S_2 with the label set that precisely contains all labels of the common n-chain that has been reduced to (a, b) in the course of obtaining S_1 and S_2 from T_1 and T_2, respectively. The set $\mathcal{B}_T$ is an agreement forest for T_1 and T_2, and we refer to it as a *chain forest* for T_1 and T_2. Since the chain reduction can be performed in polynomial time [10], the chain forests $\mathcal{B}_S$ and $\mathcal{B}_T$ can also be calculated in polynomial time from T_1 and T_2. Lastly, each element in $\mathcal{B}_T$ whose members label the leaves of a common n-chain in T_1 and T_2 with $n \geq 2$ is referred to as a *chain element*.

The next lemma bounds the number of elements in a chain forest.

Lemma 2.7. *Let T_1 and T_2 be two rooted binary phylogenetic X-trees. Let S_1 and S_2 be the reduced tree pair with respect to T_1 and T_2. Furthermore, let $\mathcal{B}_S$ and $\mathcal{B}_T$ be the chain forests for S_1 and S_2, and T_1 and T_2, respectively. Then $|\mathcal{B}_T| = |\mathcal{B}_S| < 5h(T_1, T_2)$.*

Proof. By construction of $\mathcal{B}_T$ from $\mathcal{B}_S$, it immediately follows that $|\mathcal{B}_T| = |\mathcal{B}_S|$. To show that $|\mathcal{B}_S| < 5h(T_1, T_2)$ let H be a hybridization network that displays S_1 and S_2 such that its number of hybridization vertices is minimized over all such networks. Furthermore, let H' be the $h(H)$-generator associated with H. As in the proof of Theorem 2.5, let r_0 be the number of node sides, i.e. reticulations with indegree 2 and outdegree 0, in H' and let r_1 be the number of reticulations in H' with indegree 2 and outdegree 1. Again, $r_0 + r_1 = h(H') = h(S_1, S_2)$. Recall that, to obtain H from H', we add one leaf to each node side of H', corresponding to a singleton in $\mathcal{B}_S$, and at most two leaves to each edge side of H'. Each edge side of H' to which we add two taxa corresponds to a 2-chain of S_1 and S_2 and, therefore, to a single element in $\mathcal{B}_S$. Hence, using (2.1) and Lemma 2.2, we have

$$|\mathcal{B}_T| = |\mathcal{B}_S| \leq |E(H')| + r_0 = 5r_0 + 3r_1 - 1 < 5(r_0 + r_1) = 5h(S_1, S_2) \leq 5h(T_1, T_2).$$

□

Consider again the chain forest $\mathcal{B}_T$ for T_1 and T_2. We define a *$\mathcal{B}_T$-splitting* as an acyclic agreement forest for T_1 and T_2 that can be obtained from $\mathcal{B}_T$ by repeated replacements of a chain element $\{a_1, a_2, \ldots, a_n\}$ with the elements $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$.

Lemma 2.8. *Let $\mathcal{B}_T$ be the chain forest for two rooted binary phylogenetic X-trees T_1 and T_2. Let $\{a_1, a_2, \ldots, a_n\}$ be a chain element in $\mathcal{B}_T$, and let $\mathcal{L}_j$ be a non-chain element in $\mathcal{B}_T$. Furthermore, let $\mathcal{B}_{T'} = (\mathcal{B}_T - \{\{a_1, a_2, \ldots, a_n\}\}) \cup \{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$. Then, for the inheritance graph $G_{B_{T'}}$ we have*

1. *no directed cycle of $G_{\mathcal{B}_{T'}}$ passes through an element of $\{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$ and*
2. *no directed cycle of $G_{\mathcal{B}_T}$ passes through $\mathcal{L}_j$.*

Proof. By the definition of $\mathcal{B}_T$, note that $|\mathcal{L}_j| = 1$. If $\mathcal{L}_j = \{\rho\}$, then the indegree of $\mathcal{L}_j$ is 0 in $G_{\mathcal{B}_T}$. Otherwise, if $\mathcal{L}_j \neq \{\rho\}$, then its element labels a leaf of T_1 and T_2 and, thus the outdegree of $\mathcal{L}_j$ is 0 in $G_{\mathcal{B}_T}$. Furthermore, since each element in $\{\{a_1\}, \{a_2\}, \ldots, \{a_n\}\}$ also labels a leaf of T_1 and T_2, the outdegree of the vertices $a_1, a_2, \ldots, a_n$ in $G_{\mathcal{B}_{T'}}$ is 0. This establishes the lemma. □

Let OPT($\mathcal{B}_T$-splitting) denote the size of a $\mathcal{B}_T$-splitting of smallest size.

Lemma 2.9. *Let T_1 and T_2 be two rooted binary phylogenetic X-trees, and let $\mathcal{B}_T$ be the chain forest for T_1 and T_2. Then,* OPT($\mathcal{B}_T$-splitting) $< 6h(T_1, T_2)$.

Proof. Let $\mathcal{F}_{T_1}$ be a maximum acyclic agreement forest for T_1 and T_2. In this proof, we see an agreement forest as a collection of trees. Thus, $\mathcal{F}_{T_1}$ can be obtained from T_1 (or equivalently from T_2) by deleting an $(|\mathcal{F}_{T_1}| - 1)$-sized subset, say $E_{\mathcal{F}_{T_1}}$, of the edges of T_1 and cleaning up. Similarly, $\mathcal{B}_T$ can be obtained from T_1 (or equivalently from T_2) by deleting a $(|\mathcal{B}_T| - 1)$-sized subset, say $E_{\mathcal{B}_T}$, and cleaning up. Now consider the forest $\mathcal{B}_{T'}$ obtained from T_1 by removing the edge set $E_{\mathcal{F}_{T_1}} \cup E_{\mathcal{B}_T}$ and cleaning up.

We claim that $\mathcal{B}_{T'}$ is a $\mathcal{B}_T$-splitting. To see this, first observe that $\mathcal{B}_{T'}$ is an acyclic agreement forest for T_1 and T_2 because it can be obtained by removing edge set $E_{\mathcal{B}_T}$ from $\mathcal{F}_{T_1}$ and cleaning up. Hence, to show that $\mathcal{B}_{T'}$ is a $\mathcal{B}_T$-splitting, it is left to show that it can be obtained from $\mathcal{B}_T$ by repeated replacements of a caterpillar on $\{a_1, a_2, \ldots, a_n\}$ by isolated vertices $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$.

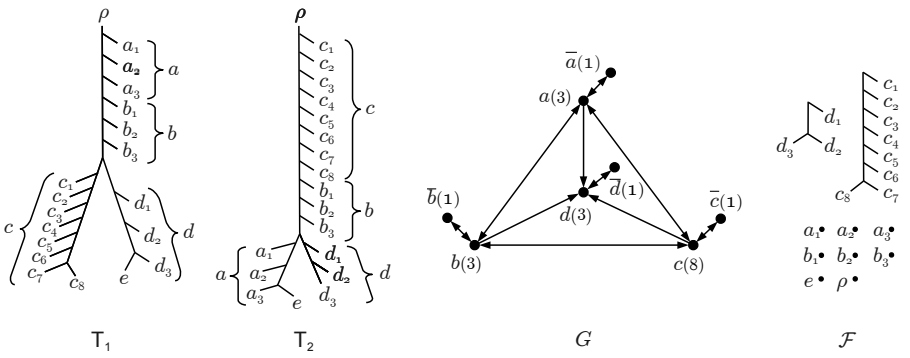

Figure 2.2: Two input trees T_1 and T_2, their graph G (with weights between parentheses) and an acyclic agreement forest $\mathcal{F}$ of T_1 and T_2. Note that $\mathcal{F}$ is a $\mathcal{B}_T$-splitting because it can be obtained from the chain forest $\mathcal{B}_T$ by atomizing chains $a = (a_1, \ldots, a_3)$ and $b = (b_1, \ldots, b_3)$. Also note that $\mathcal{F}$ has 10 components, which is equal to the weight of a minimum feedback vertex set $\{a, b, \bar{c}, \bar{d}\}$ of G, 8, plus 2, for two non-chain taxa (ρ and e).

By its definition, $\mathcal{B}_{T'}$ can be obtained from $\mathcal{B}_T$ by removing edges and cleaning up. Thus, what is left to prove is that each chain either survives or is atomized. For n-chains with $n \geq 3$, this follows from Lemma 2.1, and for $n = 2$ it is clear because $\mathcal{B}_{T'}$ can be obtained by removing edges from $\mathcal{B}_T$ in which each 2-chain is a component on its own.

As the size of $\mathcal{B}_{T'}$ is equal to the number of edges removed to obtain it from T_1 plus one, we have:

$$|\mathcal{B}_{T'}| \leq |E_{\mathcal{F}_{T_1}}| + |E_{\mathcal{B}_T}| + 1 = |\mathcal{F}_{T_1}| - 1 + |\mathcal{B}_T| < h(T_1, T_2) + 5h(T_1, T_2) = 6h(T_1, T_2),$$

where Lemma 2.7 is used to bound $|\mathcal{B}_T|$. This establishes the lemma. □

We are now in a position to prove the main result of this section.

Proof of Theorem 2.6. Throughout this proof, let $n \geq 2$. Furthermore, let $\mathcal{B}_T$ be the chain forest for T_1 and T_2, and let G be the graph obtained from the inheritance graph $G_{\mathcal{B}_T}$ by subsequently

1. weighting each vertex that corresponds to a common n-chain $(a_1, a_2, \ldots, a_n)$ of T_1 and T_2 with weight n;
2. deleting each vertex that corresponds to a non-chain taxon in $\mathcal{B}_T$; and
3. for each remaining vertex v, creating a new vertex $\bar{v}$ with weight 1 and two new edges $(v, \bar{v})$ and $(\bar{v}, v)$.

Furthermore, let w be the weight function on the vertices of G. See Figure 2.2 for an example of the construction of G. We call the added vertices $\bar{v}$ the *barred* vertices of G. Note that each common n-chain of T_1 and T_2 is represented by a vertex and its barred vertex in G. As $\mathcal{B}_T$ can be calculated in polynomial time, the construction of G also takes polynomial time, and the size of G is clearly polynomial in the cardinality of $\mathcal{B}_T$.

Now, regarding G as an instance of WDFVS, we claim the following.

Claim. There exists a $\mathcal{B}_T$-splitting of size $k + s$, where s is the number of non-chain elements in $\mathcal{B}_T$, if and only if G has a FVS of weight k.

Suppose that $\mathcal{B}_{T'}$ is a $\mathcal{B}_T$-splitting of size $k+s$. Hence, k is equal to the number of chain elements in $\mathcal{B}_T$ that are also elements in $\mathcal{B}_{T'}$ plus the total number of leaves in common n-chains that are atomized in $\mathcal{B}_{T'}$. Let T_2 be the forest that has been obtained from $\mathcal{B}_{T'}$ by deleting all singletons, and let $G_{\bar{\mathcal{B}}_{T_2}}$ be its inheritance graph. Since $G_{\mathcal{B}_{T'}}$ is acyclic, $G_{\bar{\mathcal{B}}_{T_2}}$ is also acyclic. Now, let G' be the directed graph that has been obtained from G in the following way. For each non-barred vertex v in G, delete v if v corresponds to an n-chain of T_1 and T_2 that is atomized in $\mathcal{B}_{T'}$, and delete $\bar{v}$ if v corresponds to an n-chain of T_1 and T_2 that is not atomized in $\mathcal{B}_{T'}$. Note that for each 2-cycle $(v, \bar{v}, v)$ of G either v or $\bar{v}$ is not a vertex of G' because each n-chain that is common to T_1 and T_2 is either atomized or not in $\mathcal{B}_{T'}$. This in turn implies that G' is acyclic because $G_{\bar{\mathcal{B}}_{T_2}}$ is isomorphic to $G' \backslash \bar{V}$, where $\bar{V}$ precisely contains all barred vertices of G'. Hence, an FVS of G, say V, contains each vertex of G that is not a vertex of G'. Furthermore, by the weighting of G, it follows that the weight of V is exactly k.

Conversely, suppose that there exists an FVS of G, say V, with weight k. This implies that we can remove a set V_1 of barred vertices and a set $V_2 = V \backslash V_1$ of non-barred vertices such that $\sum_{v_i \in V_2} w(v_i) + |V_1| = k$ and the graph $G' = G \backslash V$ is acyclic. For each vertex $v_i \in V_2$, let $A_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,n_i})$ be its associated common chain of T_1 and T_2, and let $w(v_i)$ be the number of elements in A_i. Furthermore, let V_1' be the subset of V_1 that contains precisely each vertex $\bar{v}$ of V_1 for which $v \notin V_2$. If $|V_1'| < |V_1|$, then it is easily checked that $V_1' \cup V_2$ is an FVS of G whose weight is strictly less than k. Therefore, we may assume for the remainder of this proof that $|V_1'| = |V_1|$. Now, let $\mathcal{B}_{T'}$ be the forest that has been obtained from $\mathcal{B}_T$ in the following way. For each vertex v_i in V_2, replace A_i in $\mathcal{B}_T$ with the elements $\{a_{i,1}\}, \{a_{i,2}\}, \ldots, \{a_{i,n_i}\}$. Thus, A_i is atomized in $\mathcal{B}_{T'}$. We next construct the inheritance graph $G_{\mathcal{B}_{T'}}$ from $G_{\mathcal{B}_T}$. For each vertex v of $G_{\mathcal{B}_T}$ that corresponds to a common n-chain $(a_1, a_2, \ldots, a_n)$ of T_1 and T_2 that is atomized in $\mathcal{B}_{T'}$, replace v with the vertices $a_1, a_2, \ldots, a_n$, delete each edge (v, w) of $G_{\mathcal{B}_T}$, and replace each edge (u, v) of $G_{\mathcal{B}_T}$ with the edges $(u, a_1), (u, a_2), \ldots, (u, a_n)$. By the proof of Lemma 2.8, the vertices $a_1, a_2, \ldots, a_n$ have outdegree 0 in $G_{\mathcal{B}_T'}$. Noting that there is a natural bijection between the cycles in $G_{\mathcal{B}_T}$ and the cycles in G that do not pass through any barred vertex, it follows that, as G' is acyclic, $G_{\mathcal{B}_T'}$ is also acyclic. Hence, $\mathcal{B}_{T'}$ is a $\mathcal{B}_T$-splitting for T_1 and T_2. The claim now follows from

$$|\mathcal{B}_{T'}| = s + \sum_{v_i \in V_2} w(v_i) + |V_1| = s + k.$$

It remains to show that the reduction is approximation preserving. Suppose that there exists a polynomial-time c-approximation for wDFVS. Let k be the weight of a solution returned by this algorithm, and let k^* be the weight of an optimal solution. By the above claim, we can then construct a solution to MAAF of size $k+s$, from which we can obtain a solution to MH with value $k+s-1$ by Theorem 1.1. We have,

$$k + s - 1 < ck^* + s \leq ck^* + cs = c(k^* + s) = c \cdot \mathrm{OPT}(\mathcal{B}_\mathrm{T}\text{-splitting})$$

and, thus, a constant factor c-approximation for finding an optimal $\mathcal{B}_T$-splitting. Now, by Lemma 2.9,

$$k + s - 1 \leq c \cdot \mathrm{OPT}(\mathcal{B}_\mathrm{T}\text{-splitting}) \leq 6c \cdot h(T_1, T_2),$$

thereby establishing that, if there exists a polynomial-time c-approximation for wDFVS, then there exists a polynomial-time $6c$-approximation for MH. This concludes the proof of the theorem. □

It is not too difficult to extend Theorem 2.6 to DFVS i.e. the unweighted variant of directed feedback vertex set.

Theorem 2.10. *If, for some $c \geq 1$, there exists a polynomial-time c-approximation for* DFVS*, then there exists a polynomial-time $6c$-approximation for* MH*.*

Proof. In the proof of Theorem 2.6 we create an instance G of wDFVS. Let w be the weight function on the vertices of G. Note that the function is non-negative and integral and for every vertex $v \in G$, $w(v) \leq |X|$ i.e. the weight function is polynomially bounded in the input size. We create an instance G' of DFVS as follows. For each vertex v in G we create $w(v)$ vertices in G' $v_1, \ldots, v_{w(v)}$. For each edge (u, v) in G we introduce edges $\{(u_i, v_j) | 1 \leq i \leq w(u), 1 \leq j \leq w(v)\}$ in G'. Solutions to wDFVS(G) and DFVS(G') are very closely related, which allows us to construct in polynomial-time a c-approximation algorithm for wDFVS from a c-approximation from DFVS. Formally, what we will demonstrate is an L-reduction [66] from wDFVS to DFVS which works for instances with polynomially-bounded weights. Specifically, consider any feedback vertex set F' of G' of size k. We create a feedback vertex set F of G as follows. For each vertex $v \in G$, we include v in F if and only if *all* the vertices $v_1, \ldots, v_{w(v)}$ are in F'. Note that the weight of F is less than or equal to k. To see that F is a feedback vertex set, suppose some cycle $C = u, v, w, \ldots, u$ survives in G. But then, for each vertex $u \in C$, some vertex u_i survives in G', which means a cycle also survived in G', contradicting the assumption that F' is a feedback vertex set. In the other direction, observe that any weight k feedback vertex set F of G can be transformed into a feedback vertex set F' of G' with size k as follows: for each $v \in F$, place all $v_1, \ldots, v_{w(v)}$ in F'. □

Notice that Theorem 2.6 does not hold only for constant c, an observation used in the next corollary.

Corollary 2.11. *There exists a polynomial-time $O(\log r \log\log r)$-approximation for* MH*, where $r = h(T_1, T_2)$*

Proof. In [25], which extended [75], a polynomial-time approximation algorithm for wDFVS is presented whose approximation ratio is $\mathrm{O}(\min(\log |V| \log\log |V|, \log \tau^* \log\log \tau^*))$, where $|V|$ is the number of vertices in the wDFVS instance and τ^* is the optimal *fractional* solution value of the problem. We show that in the wDFVS instance G that we create in the proof of Theorem 2.6, both the number of vertices in G and the weight of the optimal fractional solution value of wDFVS(G) are $\mathrm{O}(r)$. To see that G has at most $\mathrm{O}(r)$ vertices, observe that G contains two vertices for every chain element in the chain forest $\mathcal{B}_T$, and that (by Lemma 2.7) $|\mathcal{B}_T| < 5r$. Secondly, recall from Lemma 2.9 that $\mathrm{OPT}(\mathcal{B}_\mathrm{T}\text{-splitting}) < 6r$. By construction, $\mathrm{OPT}(\mathcal{B}_\mathrm{T}\text{-splitting})$ is an upper bound on the optimum solution value of wDFVS(G), hence on τ^*. Thus, given G as input, the algorithm in [25] constructs a feedback vertex set that is at most a factor $\mathrm{O}(\log r \log\log r)$ larger than the true optimal solution of wDFVS(G). As shown in the proof of Theorem 2.6 this can be used to obtain an approximation ratio at most 6 times larger for MAAF, which is clearly also $\mathrm{O}(\log r \log\log r)$. □

Finally, note that for a given instance the actual approximation ratio obtained by Corollary 2.11 will sometimes be determined by $|V|$, and sometimes by τ^*, and can potentially be significantly smaller than $\mathrm{O}(\log r \log\log r)$. For example, if there are very few chains in the chain forest, but they are all extremely long, then it can happen that $|V| << \tau^*$. Conversely, if the chain forest contains many short chains, and only a small number of them need to be atomized to attain acyclicity, then it can happen that $\tau^* << |V|$.

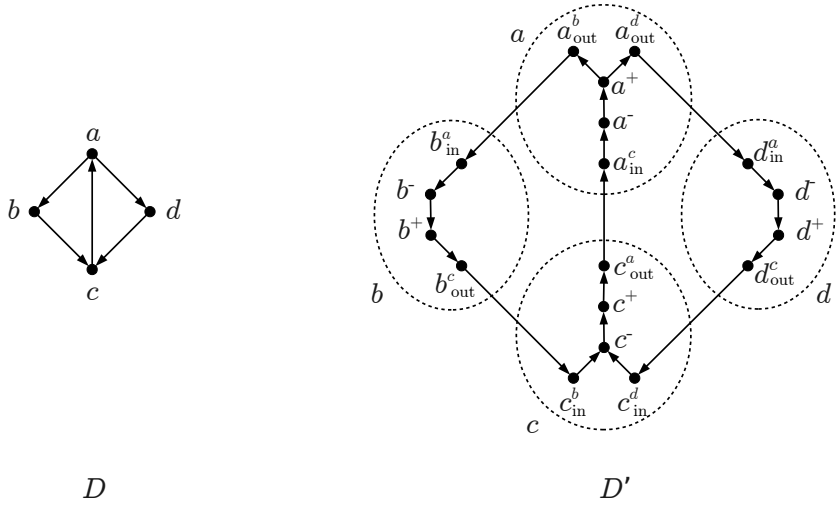

Figure 2.3: An instance D of DFVS and the modified graph D'.

2.3 An approximation-preserving reduction from DFVS to MH

In this section we prove the following theorem.

Theorem 2.12. *If, for some constant $c \geq 1$, there exists a polynomial-time c-approximation algorithm for* MH*, then there exists a polynomial-time $(c+\epsilon)$-approximation algorithm for* DFVS *for all $\epsilon > 0$.*

Proof. We show an approximation preserving reduction from DFVS to MAAF. The theorem then follows because of the equivalence of MAAF and MH described in Theorem 1.1.

Let $D = (V, A)$ be an instance of DFVS. First we transform D into an auxiliary graph $D' = (V', A')$. For a vertex v of D, we denote the parents of v as $u_1, u_2, \ldots, u_{d^-(v)}$ and the children of v as $w_1, w_2, \ldots, w_{d^+(v)}$ (To facilitate the exposition, we assume a total order on the parents of each vertex and on the children of each vertex.). We construct the graph D' as follows. For every vertex $v \in V$, D' has vertices $v_{\text{in}}^{u_1}, v_{\text{in}}^{u_2}, \ldots, v_{\text{in}}^{u_{d^-(v)}}$, vertices v^- and v^+ as well as vertices $v_{\text{out}}^{w_1}, v_{\text{out}}^{w_2}, \ldots, v_{\text{out}}^{w_{d^+(v)}}$. The edges of D' are as follows. For each vertex $v \in V$, D' has edges from each of $v_{\text{in}}^{u_1}, v_{\text{in}}^{u_2}, \ldots, v_{\text{in}}^{u_{d^-(v)}}$ to v^-, an edge from v^- to v^+ and edges from v^+ to each of $v_{\text{out}}^{w_1}, v_{\text{out}}^{w_2}, \ldots, v_{\text{out}}^{w_{d^+(v)}}$. In addition, for each edge (u, v) of D, there is an edge $(u_{\text{out}}^v, v_{\text{in}}^u)$ in D'. This concludes the construction of D'. An example is given in Figure 2.3.

We now first show that D has a FVS of size at most f if and only if D' has a FVS of size at most f. Observe that each directed cycle of D corresponds to a directed cycle of D' and vice versa. Also notice that any cycle in D' that contains a vertex v_{in}^u or a vertex v_{out}^w also contains the vertices v^- and v^+. Hence, for any FVS F of D' we can create a FVS F' of D' of at most the same size and containing only vertices of the type v^-. We assume from now on that any FVS of D' is of this form. It is now obvious that $F = \{v \in V \mid v^- \in F'\}$ is an FVS of D if and only if $F' = \{v^- \in V' \mid v \in F\}$ is an FVS of D'.

Intuitively, the idea of our reduction is as follows. We will construct two rooted binary trees T_1 and T_2 consisting of long chains. We build them in such a way that the graph D' is basically the

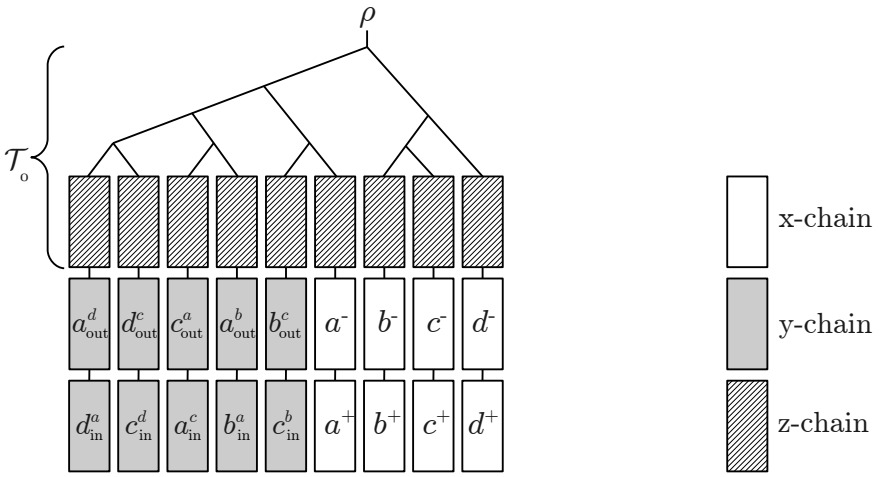

Figure 2.4: T_1: the first tree of the constructed MAAF instance.

inheritance graph of the chain forest for T_1 and T_2. This graph can be made acyclic by atomizing some of the chains. Thus, solving DFVS on D' is basically equivalent to deciding which chains to atomize. We make all the chains that can be atomized of the same length. Hence, since each chain that is atomized adds the same number of components to the agreement forest, solving DFVS on D' is essentially equivalent to finding a maximum acyclic agreement forest for T_1 and T_2.

Before we proceed, we need some more definitions. Recall that an *n-chain* of a tree is an n-tuple $(a_1, a_2, \ldots, a_n)$ of leaves such that the parent of a_1 is either the same as the parent of a_2 or the parent of a_1 is a child of the parent of a_2 and, for each $i \in \{2, 3, \ldots, n-1\}$, the parent of a_i is a child of the parent of a_{i+1}. A tree T whose leaf set $\mathcal{L}(T)$ is a chain of T is called a *caterpillar* on $\mathcal{L}(T)$[1] It is easy to see that, for every chain C, there exists a unique caterpillar on C. By *hanging* a chain C below a leaf x, we mean the following: subdivide the edge entering x by a new vertex v and add an edge from v to the root of the caterpillar on C. When we hang a chain C_1 below a chain C_2, we hang the caterpillar on C_1 below the lowest leaf (or a lowest leaf) x_1 of C_2. By *replacing* a leaf x by a chain C we mean: delete x and add an edge from its former parent to the root of the caterpillar on C.

We are now ready to construct an instance of MAAF. The trees, T_1 and T_2, will be built of chains of three types: x-type, y-type and z-type. The x-type chains have length ℓ while the y-type and z-type chains have length L (with $L >> \ell$). Each of these chains will be common to both trees. Recall that, by Lemma 2.1, we may assume that every chain either survives or is atomized. The idea is that y-type chains and z-type chains are so long that they will all survive. The x-type chains are shorter and might be atomized. In fact, the x-type chains that are atomized will correspond to a FVS of D'.

We build the trees T_1 and T_2 as follows. For each vertex of D' of the type v^- or v^+ we create an x-type chain. For each other vertex of D' we create a y-type chain. Finally, for each vertex and edge of the original graph D we create a z-type chain. All leaves of all chains have different labels. Now we combine the chains into two trees as follows.

First T_1. Start with an arbitrary rooted binary tree on $|V|+|A|$ leaves labeled by the vertices and edges of D. We replace each leaf by its z-type chain, creating the tree which we call T_0.

[1] In this context a caterpillar does *not* have an artificial vertex labelled ρ.

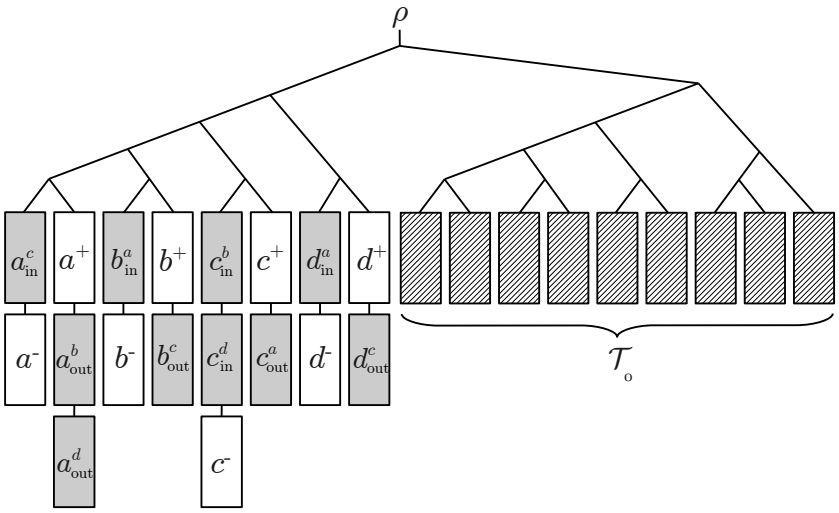

Figure 2.5: T_2: the second tree of the constructed MAAF instance.

Below each z-chain replacing a leaf labeled by an edge $(u, v) \in A$ we hang the y-type chain for u^v_{out} and below that the y-type chain for v^u_{in}. Below each z-chain replacing a leaf labeled by a vertex $v \in V$ we hang a x-type chain for v^- and below that a x-type chain for v^+.

Now T_2. Start with an arbitrary rooted binary tree on $2|V|$ leaves having two leaves for each vertex $v \in V$. Replace one of them by a concatenation of (from top to bottom) the y-type chains for $v^{u_1}_{\text{in}}, v^{u_2}_{\text{in}}, \ldots, v^{u_{d^-(v)}}_{\text{in}}$ and the x-type chain for v^-. Replace the other leaf for v by a concatenation of (from top to bottom) the x-type chain for v^+ and the y-type chains for $v^{w_1}_{\text{out}}, v^{w_2}_{\text{out}}, \ldots, v^{w_{d^+(v)}}_{\text{out}}$. Finally, hang a copy of T_0 below the root. This concludes the construction of the MAAF instance. For an example, see Figures 2.4 and 2.5.

We claim that D' (and thus D) has a FVS of size at most f if and only if there exists an acyclic agreement forest of T_1 and T_2 of size at most $1 + 2(|A| + |V|) + (\ell - 1)f$.

To show this, consider the agreement forest A_D for T_1 and T_2 in which T_0 is one component, each x-type chain is one component, and each y-type chain is one component. Represent the components of A_D by their corresponding vertices and edges. The inheritance graph G_{A_D} is obtained from D' by adding the edges $(v^{u_i}_{\text{in}}, v^{u_j}_{\text{in}})$ for all $1 \leq i < j \leq d^-(v)$, the edges $(v^{w_i}_{\text{out}}, v^{w_j}_{\text{out}})$ for all $1 \leq i < j \leq d^+(v)$, and creating a vertex v_{T_0} corresponding to T_0 with only outgoing edges to all other vertices. Hence v_{T_0} is not in any cycle of G_{A_D}. It is easy to see that any FVS of D', which by assumption has only vertices of the type v^-, is also a FVS of G_{A_D} and vice versa, since any directed cycle of G_{A_D} passing through an edge $(v^{u_i}_{\text{in}}, v^{u_{i+1}}_{\text{in}})$ or $(v^{w_i}_{\text{out}}, v^{w_{i+1}}_{\text{out}})$ must contain v^-. In addition, since v^--type vertices correspond to x-type chains, it is possible to make G_{A_D} acyclic by atomizing only x-type chains.

Let F be a FVS of D and let F' be the corresponding FVS of D'. Then we can construct an agreement forest $\mathcal{R}$ of T_1 and T_2 as follows. One component consists of the tree T_0. Each of the y-type chains is also one component, as well as each x-type chain that does not correspond to a vertex in F'. Finally, each x-type chain corresponding to a vertex in F' is atomized. Thus, the number of components is $1 + 2|A| + (2|V| - |F'|) + \ell|F'| = 1 + 2(|A| + |V|) + (\ell - 1)|F|$. We have to show that the inheritance graph $G_{\mathcal{R}}$ is acyclic. We can construct $G_{\mathcal{R}}$ from G_{A_D} as follows. Delete every vertex $v^- \in F'$ and instead add a vertex for each leaf of the corresponding x-type chain with incoming edges from T_0 and from $v^{u_1}_{\text{in}}, v^{u_2}_{\text{in}}, \ldots, v^{u_{d^-(v)}}_{\text{in}}$. Since we only introduced

leaves with incoming edges, this modification does not create any directed cycles. Thus, since F' contains a vertex of each directed cycle of G_{A_D}, and all vertices from F' have been removed, $G_{\mathcal{R}}$ is acyclic. It follows that $\mathcal{R}$ is an acyclic agreement forest for T_1 and T_2.

To show the other direction, let $\mathcal{A}$ be an acyclic agreement forest of T_1 and T_2. First, we may assume that all y-type chains and z-type chains survive in $\mathcal{A}$, since we will choose L sufficiently large (as will be specified later). To see this, recall that we may assume by Lemma 2.1 that each chain either survives or is atomized. Hence, if a y-type chain or z-type chain does not survive, it is atomized and adds L components to the agreement forest. Secondly, observe that we may assume that all z-type chains are together in a single component (if they are not, we can put them together and reduce the number of components).

Now we argue that any pair of chains, at least one of which is not a z-type chain, cannot be together in a single component of $\mathcal{A}$. Firstly, if the two chains are below each other in T_1, then they are next to each other in T_2. Secondly, if the two chains are next to each other in T_1, then they are separated by a z-type chain in T_1 but not in T_2. Hence, by (2) in the definition of an agreement forest, the two chains can not be together in a single component of $\mathcal{A}$. Thus, the components of $\mathcal{A}$ are as follows. Tree T_0 is the component containing the root and all z-type chains. Furthermore, each y-type chain, each surviving x-type chain, and each leaf of a non-surviving x-type chain is a separate component. Let $\tilde{F}$ be the set of vertices of G_{A_D} corresponding to the non-surviving x-type chains. Thus, each vertex in $\tilde{F}$ is of the type v^- or v^+. We will show that $\tilde{F}$ is a FVS of G_{A_D} and hence of D'. We can construct $G_{\mathcal{A}}$ from G_{A_D} as follows. Remove each vertex in $\tilde{F}$ from G_{A_D} and add each leaf of the corresponding x-type chain as a separate vertex. Then add edges to these newly added vertices (these edges are not important since they do not create any directed cycles). Since $\mathcal{A}$ is an acyclic agreement forest, $G_{\mathcal{A}}$ is acyclic and hence $\tilde{F}$ is a FVS. The size $|\tilde{F}|$ of the FVS is equal to the number of non-surviving x-type chains. Thus, $|\mathcal{A}| = 1 + 2|A| + (2|V| - |\tilde{F}|) + \ell|\tilde{F}| = 1 + 2(|A| + |V|) + (\ell - 1)|\tilde{F}|$.

The reduction is clearly polynomial time. It remains to show that it is approximation preserving. Suppose that there exists a c-approximation algorithm for MAAF. Say that m is the size of the MAAF returned by this algorithm and m^* the size of an optimal solution. Recall that MAAF minimizes the size of an agreement forest minus one, so $m - 1 \leq c \cdot (m^* - 1)$. We have shown that D has a FVS of size at most f if and only if T_1 and T_2 have an acylic agreement forest of size at most $1 + 2(|A| + |V|) + (\ell - 1)f$. Thus, $m^* = 1 + 2(|A| + |V|) + (\ell - 1)f^*$. Moreover, an approximate solution f of DFVS can be computed from an approximate solution m of MAAF by taking $f = (m - 1 - 2(|A| + |V|))/(\ell - 1)$. Then we have

$$\begin{aligned} f &= \frac{m - 1 - 2(|A| + |V|)}{\ell - 1} \\ &\leq \frac{c \cdot (m^* - 1) - 2(|A| + |V|)}{\ell - 1} \\ &= \frac{c(2(|A| + |V|) + (\ell - 1)f^*) - 2(|A| + |V|)}{\ell - 1} \\ &= c \cdot f^* + \frac{2(c - 1)(|A| + |V|)}{\ell - 1} \\ &= c \cdot f^* + 1 \end{aligned}$$

if we take $\ell = \lceil 2(c-1)(|A|+|V|)+1 \rceil$. We still need to specify the value of L, which needs to be sufficiently large so that all y-type chains and z-type chains survive. Since any graph trivially has a FVS of size $|V|$, any constructed MAAF instance has $m^* \leq 1 + 2(|A| + |V|) + (\ell - 1)|V|$.

Thus, a c-approximation algorithm will return an acyclic agreement forest of size m with $m-1 \leq c(m^*-1) \leq c(2(|A|+|V|)+(\ell-1)|V|)$. And hence with $m \leq c(2(|A|+|V|)+(\ell-1)|V|)+1$. So it suffices to take $L = \lceil c(2(|A|+|V|)+(\ell-1)|V|)+2 \rceil$.

Now take $\epsilon > 0$. If $f^* < 1/\epsilon$, we can compute an optimal solution for DFVS by brute force in polynomial time. Otherwise, $1 \leq \epsilon \cdot f^*$ and we have

$$f \leq c \cdot f^* + \epsilon \cdot f^* = (c+\epsilon) f^*.$$

Thus, if there exists a c-approximation for MAAF, then there exists a $(c+\epsilon)$-approximation for DFVS for every fixed $\epsilon > 0$.

□

In contrast to the result in Section 2.2, the reduction above can only be used for constant c. It does *not* show that e.g. an $O(\log |X|)$-approximation for MH would imply an $O(\log |V|)$-approximation for DFVS. Hence, it is indeed possible that MH admits an $O(\log |X|)$-approximation while DFVS does not admit an $O(\log |V|)$-approximation. For neither of the problems such an approximation is known to exist.

Finally, we note that Theorem 2.12 also allows us to improve upon the best-known inapproximability result for MH.

Corollary 2.13. *There does not exist a polynomial-time c-approximation for* MH*, where $c < 10\sqrt{5}-21 \approx 1.3606$, unless P=NP. If the Unique Games Conjecture holds, then there does not exist a polynomial-time c-approximation for* MH *where $c < 2$.*

Proof. In [52] a simple reduction is shown from the problem Vertex Cover to the problem DFVS. Specifically, given an undirected graph G as input to Vertex Cover we create a directed graph G' by transforming each edge $\{u, v\}$ in G into two directed edges $(u, v), (v, u)$ in G'. It is easy to show that G' has a feedback vertex set of size k if and only if G has a vertex cover of size k. Consequently, any polynomial-time c-approximation algorithm for DFVS can be used to construct a polynomial-time c-approximation for Vertex Cover. The latter problem does not permit a polynomial-time c-approximation, for any $c < 10\sqrt{5}-21 \approx 1.3606$, unless P=NP [21, 22]. Also, it has been shown that if the Unique Games Conjecture is true then no approximation better than 2 is possible [59]. Now, the proof of Theorem 2.12 shows that, if there exists a c-approximation for MH, then there exists a $(c+\epsilon)$-approximation for DFVS for *every* fixed $\epsilon > 0$. Hence the existence of a c-approximation for MH where $c < 10\sqrt{5}-21$ (respectively, $c < 2$) would mean the existence of a c'-approximation for DFVS (and thus also for Vertex Cover) where $c' < 10\sqrt{5}-21$ (respectively, $c' < 2$). □

We have seen several interesting spin-off consequences of the result presented in this chapter, both negative and positive. On the negative side, it is known that there is a very simple parsimonious reduction from the classical problem Vertex Cover to DFVS [52]. Consequently, a c-approximation for DFVS entails a c-approximation for Vertex Cover, for every $c \geq 1$. For $c < 10\sqrt{5}-21 \approx 1.3606$ there cannot exist a polynomial-time c-approximation of Vertex Cover, assuming P $\neq$ NP [21, 22]. Also, if the Unique Games Conjecture is true then for $c < 2$ there cannot exist a polynomial-time c-approximation of Vertex Cover [59]. (Whether Vertex Cover permits a constant factor approximation ratio strictly smaller than 2 is a long-standing open problem). The main result in this chapter hence not only showed that MH is in APX if and only if DFVS is in APX, but also that MH cannot be approximated within a factor of 1.3606, unless P=NP (and not within a factor smaller than 2 if the Unique Games Conjecture is true). This improves significantly on the current APX-hardness threshold of $\frac{2113}{2112}$.

On the positive side, we observe that already-existing approximation algorithms for DFVS can be utilized to give asymptotically comparable approximation ratios for MH. To date the best polynomial-time approximation algorithms for DFVS achieve an approximation ratio of $O(\min\{\log n \log\log n, \log \tau^* \log\log \tau^*\})$, where n is the number of vertices in the graph and τ^* is the optimal fractional solution of the problem (taking the weights of the vertices into account) [25, 75]. We showed that this algorithm can be used to give an $O(\log r \log\log r)$-approximation algorithm for MH, where r is the hybridization number of the two input trees. To the best of our knowledge, this is the first non-trivial polynomial-time approximation algorithm for MH.

On the practical side, we realized that our insight into cycle breaking leads to a cute, simple and increadibly good approximation algorithm. In the next two chapters we improve on the theoretical work done here, present two algorithms and run experiments to convince the reader of just how good this algorithm is in practice.

Chapter 3

...is a practical approximation algorithm

In this and the next chapter we extend the theoretical work established so far and give it a practical twist to yield two fast approximation algorithms Cycle Killer and Nonbinary Cycle Killer.

Both of these algorithms have two desirable qualities: they terminate quickly even for massive instances of hybridization number and give a non-trivial guarantee of proximity to optimality. These are the first algorithms with such properties. Both algorithms are based on a non-trivial marriage of maf and dfvs solvers (both exact and approximate), meaning that further advances in solving maf and dfvs will directly lead to improvements in Cycle Killer and Nonbinary Cycle Killer.

This chapter also improves the theoretical work given in the previous one, which also proposed using dfvs but beginning from a trivial Agreement Forest (AF) known as a *chain forest*. Here we use a smarter starting point: an (approximate) maf, and it is this insight which makes a 2-approximation (rather than the 6-approximation) possible when using an exact dfvs solver. Other authors have also had the idea of cycle-breaking in AFs: the advanced FPT algorithm of Whidden et al [83] and the algorithms in the HybridNet family. However, both algorithms start the cycle-breaking from many starting points. In contrast, our algorithm requires only a *single* starting point, i.e. a single (approximate) solution to maf.

The worst-case running time of both of these approximation algorithms is exponential. However, as we demonstrate with experiments, the running time of our algorithms is in practice extremely fast. For large and/or massively discordant binary trees, Cycle Killer is typically orders of magnitude faster than the HybridNet algorithms and the algorithm in Dendroscope. The performance gap between Nonbinary Cycle Killer and its exact counterparts is less pronounced, but still significant, especially in its fastest mode of operation.

Of course, exact algorithms attempt to compute optimum solutions, whereas our algorithms only give approximate solutions. Nevertheless, our experiments show that when Cycle Killer and Nonbinary Cycle Killer are run in their most accurate mode of operation, an approximation ratio very close to 1 is not unusual, suggesting that the algorithms often produce solutions close to optimality and well within the worst-case approximation guarantee.

The idea behind the binary and nonbinary algorithm is similar. Specifically, we describe an algorithm with approximation ratio $d(c+1)$ for the hybridization number problem on two binary trees and an algorithm with approximation ratio $d(c+3)$ for the hybridization number problem on two nonbinary trees by combining a c-approximation for the problem maf with a

d-approximation for the problem DFVS. Both these problems are NP-hard so polynomial-time algorithms attaining $c = 1$ or $d = 1$ are not realistic. Nevertheless, there exist extremely fast FPT algorithms for solving MAF on binary trees exactly (i.e. $c = 1$), the fastest is RSPR by Whidden, Beiko and Zeh [81, 84] although the MAF algorithm inside [17] is also competitive. Moreover, we observe that the type of DFVS instances that arise in hybridization practice can easily be solved using Integer Linear Programming (ILP) (and freely-available ILP solver technology such as GLPK), so $d = 1$ is also often possible.

Combining these two exact approaches gives us, in the binary case, an exponential-time approximation algorithm with worst-case approximation ratio 2 that for large instances still runs extremely quickly; this is the `2-approx` option of CYCLE KILLER. In practice, we have observed that the upper bound of 2 is often pessimistic, with much better approximation ratios observed in experiments (1.003 on average for the simulations presented in this chapter). We find that this algorithm already allows us to cope with much bigger trees than the HYBRIDNET algorithms or the algorithm in DENDROSCOPE.

Nevertheless, for truly massive trees it is often not feasible to have $c = 1$. Fortunately there exist linear-time algorithms which achieve $c = 3$ [83]. This, coupled with the fact that (even for such trees) it remains feasible to use an exact ($d = 1$) solver for DFVS, means that in practice we achieve a 4-approximation for gigantic binary trees; this is the `4-approx` option of CYCLE KILLER. Again, the ratio of 4 is a worst-case bound and we suspect that in practice we are doing better than 4. However, this cannot be experimentally verified due to the lack of good lower bounds for such massive instances. In any case, the main advantage of this option is that it can, without too much effort, cope with trees with hundreds or thousands of taxa and hybridization number of a similar order of magnitude. An implementation of CYCLE KILLER and accompanying documentation can be downloaded from `http://skelk.sdf-eu.org/cyclekiller`. Networks created by the algorithm can be viewed in DENDROSCOPE.

In this chapter we only present the experiments and the theory behind the binary algorithm. The nonbinary case is more involved and we will revisit it in Chapter 4 once the basic idea behind the binary case is clear.

3.1 The algorithm for binary trees

We show how MAAF can be approximated by combining algorithms for MAF and DFVS. In particular, we will prove the following theorem.

Theorem 3.1. *If there exists a c-approximation for* MAF *and a d-approximation for* DFVS*, then there exists a $d(c+1)$-approximation for* MAAF *(and thus for* MH*).*

To prove the theorem, suppose there exists a c-approximation for MAF. Let T_1 and T_2 be two trees and let M be an agreement forest returned by the algorithm. We use notation $\text{MAF}(T_1, T_2)$ and $\text{MAAF}(T_1, T_2)$ for the optimal objective value of a maximum agreement forest and maximum acyclic agreement forest respectively, for input trees T_1 and T_2. Then,

$$|M| - 1 \leq c \cdot \text{MAF}(T_1, T_2) \leq c \cdot \text{MAAF}(T_1, T_2). \tag{3.1}$$

An *M-splitting* is an acyclic agreement forest that can be obtained from M by removing edges and cleaning up.

Lemma 3.2. *Let T_1 and T_2 be two trees and M an agreement forest of T_1 and T_2. Then, there exists an M-splitting of size at most $\text{MAAF}(T_1, T_2) + |M|$.*

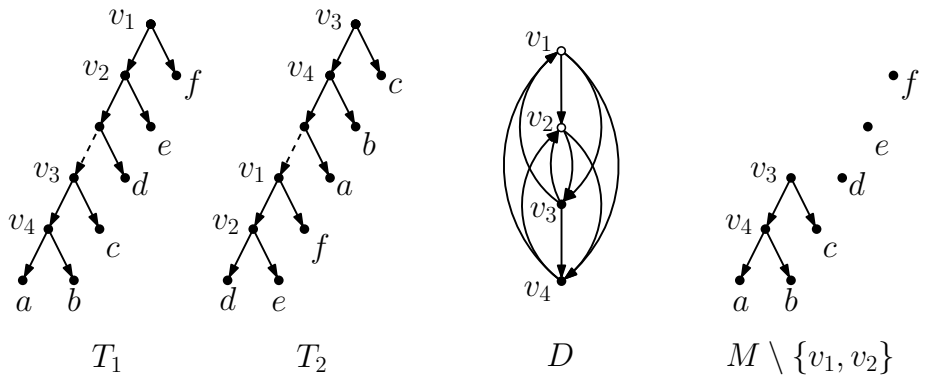

Figure 3.1: Two binary trees T_1 and T_2 and the auxiliary graph D. A maximum agreement forest M of T_1 and T_2 is obtained by deleting the dashed edges. Graph D can be made acyclic by deleting either both filled or both unfilled vertices. Hence, removing either v_1 and v_2 or v_3 and v_4 from M makes it an acyclic agreement forest for T_1 and T_2, see Lemma 3.3. The acyclic agreement forest $M \setminus \{v_1, v_2\}$ obtained by removing v_1 and v_2 from M is depicted on the right.

Proof. Consider a maximum acyclic agreement forest F of T_1 and T_2. For $i \in \{1, 2\}$, F can be obtained from T_i by removing a set of edges, say E_F^i, and cleaning up. Moreover, also M can be obtained from T_i by removing a set of edges, say E_M^i, and cleaning up.

Now consider the forest S obtained from T_1 by removing $E_M^1 \cup E_F^1$ and cleaning up. Then,

- S is an agreement forest of T_1 and T_2 because it can be obtained from T_2 by removing edges $E_M^2 \cup E_F^2$ and cleaning up;
- S is acyclic because it can be obtained by removing edges from F, which is acyclic, and cleaning up;
- S can be obtained from M by removing edges and cleaning up.

Hence, S is an M-splitting. Furthermore, $|S| \leq |E_F^1| + |E_M^1| + 1$. The lemma follows since $|E_F^1| = \text{MAAF}(T_1, T_2)$ and $|M| = |E_M^1| + 1$. □

Let $\text{OptSplitting}_{T_1,T_2}(M)$ denote the size of a minimum-size M-splitting. Combining Lemma 3.2 and equation (3.1), we obtain

$$\text{OptSplitting}_{T_1,T_2}(M) - 1 \leq (c+1)\text{MAAF}(T_1, T_2) \tag{3.2}$$

We will now show how to find an approximation for the problem of finding an optimal M-splitting. We do so by reducing the problem to DFVS. We construct an input graph D for DFVS (called the *extended inheritance* graph) as follows. For every vertex of M that has outdegree 2 (in M), we create a vertex in D. There is an edge in D from a vertex u to a vertex v precisely if in either T_1 or T_2 (or in both) there is a directed path from u to v. An example is in Figure 3.1. We claim the following.

Lemma 3.3. *A subset V' of the vertices of D is a feedback vertex set of D if and only if removing V' from M makes it an acyclic agreement forest.*

Proof. We show that $D \setminus V'$ has a directed cycle if and only if the inheritance graph of $M \setminus V'$ has a directed cycle.

To prove this, first suppose that there is a cycle $v_1, v_2, \ldots, v_k = v_1$ in the inheritance graph of $M \setminus V'$. The vertices in the inheritance graph of $M \setminus V'$ correspond to the roots of the components of $M \setminus V'$. Since these roots have outdegree 2 in $M \setminus V'$, they had outdegree 2 in M, and are thus vertices of D. So the vertices $v_1, v_2, \ldots, v_k$ that form the cycle are vertices of D. Since these vertices are in the inheritance graph of $M \setminus V'$, they can not be in V' and so they are vertices of $D \setminus V'$. The reachability relation between these vertices in $D \setminus V'$ is the same as in the inheritance graph of $M \setminus V'$. So, the vertices $v_1, v_2, \ldots, v_k$ form a cycle in $D \setminus V'$.

Now suppose that there is a cycle $w_1, w_2, \ldots, w_k = w_1$ in $D \setminus V'$. Each of the vertices $w_1, w_2, \ldots, w_k$ is a vertex with outdegree-2 in M. Some of them might be roots of components, while others are not. However, observe that if there is a directed path from a vertex u to a vertex v in T_1 (or in T_2) then there is also a directed path from the root of the component of $M \setminus V'$ that contains u to the root of the component of $M \setminus V'$ that contains v. Hence, there is a directed cycle in the inheritance graph of $M \setminus V'$, formed by the roots of the components of $M \setminus V'$ that contain $w_1, w_2, \ldots, w_k$. □

Proof of Theorem 3.1. Suppose that there exists a d-approximation for DFVS. Let FVS be a feedback vertex set returned by this algorithm and let MFVS be a minimum feedback vertex set. Then, removing the vertices of MFVS from M gives an optimal M-splitting. Furthermore, $\text{OptSplitting}_{T_1,T_2}(M) = |M| + |\text{MFVS}|$. This is because for every vertex in a cycle C, its parent in M must participate in some cycle that contains elements of C. So if we start by removing the root of the component we are splitting and subsequently remove only those vertices whose parents have already been removed we see that we add at most one component per vertex. In fact, because vertices of D all have out-degree 2 in M, we add exactly one component per vertex.

By removing the vertices of FVS from M, we obtain an acyclic agreement forest $\mathcal{F}$ such that

$$\begin{aligned} |\mathcal{F}| - 1 &= |M| + |\text{FVS}| - 1 \\ &\leq |M| + d \cdot |\text{MFVS}| - 1 \\ &\leq d(|M| + |\text{MFVS}| - 1) \\ &= d(\text{OptSplitting}_{T_1,T_2}(M) - 1) \\ &\leq d(c+1)\text{MAAF}(T_1, T_2), \end{aligned}$$

where the last inequality follows from equation (3.2). Thus, $\mathcal{F}$ is a $d(c+1)$-approximation to MAAF, which concludes the proof of Theorem 3.1. □

Theorem 3.1 implies that a solution to the MAAF problem for a given instance can be constructed by (i) finding a solution $\mathcal{F}$ to the MAF problem for the same instance (ii) constructing the extended inheritance graph D for $\mathcal{F}$ (iii) finding a solution V for the DFVS problem on the graph D and (iv) modifying $\mathcal{F}$ accordantly to V.

3.2 Experiments and discussion

To assess the performance of CycleKiller, a simulation study was undertaken. We generated 3 synthetic datasets, an *easy*, a *medium* and a *hard* one, containing respectively 800, 640 and 640 pairs of rooted binary phylogenetic trees.

The easy data set was created by varying two parameters, namely the number of taxa n and the number of rSPR-moves k used to obtain the second tree from the first (note that this number is an upper bound on the actual rSPR distance). The 800 pairs of rooted binary phylogenetic trees were created by varying n in $\{20, 50, 100, 200\}$ and k in $\{5, 10, ..., 25\}$, and then creating 40 different instances per each combination of parameters. Each pair (T_1, T_2) of rooted binary

phylogenetic trees for a given set of parameters n and k is created as follows: The first tree T_1 on $\mathcal{X} = \{x_1, \ldots, x_n\}$ is generated by first creating a set of n leaf vertices bijectively labeled by the set $\mathcal{X}$. Then, two vertices u and v, both with indegree 0, are randomly picked and a new vertex w, along with two new edges (w, u) and (w, v), is created. This is done until only one vertex with no ancestor, the root, is present. The second tree T_2 is obtained from T_1 by applying k rSPR-moves. The medium and the hard data sets were generated in the same way as the easy one, but for different choices of the parameters: n in $\{50, 100, 200, 300\}$ and k in $\{15, 25, 40, 55\}$ for the medium one and n in $\{100, 200, 400, 500\}$ and k in $\{40, 60, 80, 100\}$ for the hard one.

The exact hybridization number has been computed by HybridNet [16], available from `http://www.cs.cityu.edu.hk/~lwang/software/Hn/treeComp.html` or with Dendroscope [39], available from `http://www.dendroscope.org`. We will refer to these algorithms as the *exact algorithms*. Each instance has been run on a single core of an Intel Xeon E5506 processor.

Each run that took more than one hour was aborted. For each instance, we ran our program with the option `2-approx`, and, in case the latter did not finish within one hour, we ran it again, this time using the option `4-approx`, always with a one-hour limit. We used the program rSPR v1.03 [81, 84] to solve or approximate MAF and GLPK v4.47 (`http://www.gnu.org/software/glpk/`) to solve the ILP formulation of DFVS.

For all instances of the easy data set, CycleKiller finished with the `2-approx` option within the one hour limit, while for 33 instances the exact algorithms were unable to compute the hybridization number. Note that, even for "easy" instances, computing the exact hybridization number can take a very long time. To give the reader an idea, for 9 runs of the easy data, Dendroscope and HybridNet did not complete within 10 days. Table 1 shows a summary of the results. It can be seen that CycleKiller was much faster than the exact algorithms. Moreover, for 96.6% of the instances for which an exact algorithm could find a solution, CycleKiller also found an optimal solution. While the theoretical worst-case approximation ratio of the `2-approx` option of CycleKiller is 2, in our experiments it performed very close to a 1-approximation.

For the medium data set, CycleKiller finished with the `2-approx` option for 613 instances, and for the remaining ones with the `4-approx` option. The exact algorithms could compute the hybridization number for only 199 instances (out of 640). For 97.5% of these instances, CycleKiller also found an optimal solution, but with a much better running time. Regarding the hard data set, 444 runs were completed with the `2-approx` option and for the remaining ones we were able to use the `4-approx` option within the given time constraint. Unfortunately, the exact algorithms were unable to compute the hybridization number for any tree-pair of this data set and hence we could not compute the average approximation ratios. Over all our experiments, the maximum hybridization number that the exact algorithms could handle was 25.[1] In contrast, the `2-approx` option of CycleKiller could be used for instances for which the size of a MAF was up to 97, and thus for instances for which the hybridization number was at least 97.

To find the limits of the `4-approx` option of Cycle Killer, we also tested it on randomly generated trees. On a normal laptop, it could construct networks with up to 10,000 leaves and up to 10,000 reticulations within 10 minutes. Since the number of reticulations found is at most four times the optimal hybridization number, this implies that the `4-approx` option of Cycle Killer can handle hybridization numbers up to at least 2,500.

[1] In [2], it has been shown that this number can go up to 40 when running Dendroscope on a similar processor but allocating all cores for one instance, i.e. exploiting the possibilities of parallel computation of this implementation.

		Exact algorithms		CycleKiller					
Dataset	Total runs	Completed	Running time (RT)	`2-approx` Compl.	`2-approx` RT	`4-approx` Compl.	`4-approx` RT	Average appr. ratio	Opt. found
Easy	800	767	$13m18s$	800	$3s$	-	-	1.003	96.6%
Medium	640	199	$42m52s$	613	$3m32s$	27	<1 s	1.002	97.5%
Hard	640	0	$1h$	440	$21m11s$	200	1.5 s	-	-

Table 3.1: Experimental results. The third column indicates for how many instances at least one exact algorithm finished within one hour. The fifth column indicates for how many instances the `2-approx` option of CycleKiller finished within one hour. For the remaining instances, the `4-approx` option finished within one hour, as can be seen from the seventh column. The average running time for the `2-approx` and the `4-approx` are reported respectively in the sixth and eighth column. The average approximation ratio (ninth column) is taken over all instances for which at least one exact method finished. The last column indicates the percentage of those instances for which CycleKiller found an optimal solution.

Chapter 4

...that can be generalized to nonbinary situations

In applied phylogenetics it rarely happens that trees are binary. Often there is insufficient information available to be able to determine the exact order in which several branching events occurred, and it is standard practice to model this uncertainty using vertices with outdegree higher than two: a *soft polytomy* [63]. Hence it is important to develop algorithms for the *nonbinary* case i.e., when the trees are not necessarily binary and high-degree vertices capture a set of equally likely branching scenarios.

As was the case in the binary setting, the problem MAAF is NP-hard and APX-hard [11]. In the previous chapter we showed how, for the binary variant of MAAF, large instances *can* however be very well approximated using a specific marriage of MAF and DFVS solvers. This approach is exponential-time in the worst case but in practice is very fast and yields highly competitive approximation factors.

In this chapter we give three algorithms for nonbinary agreement forests. We start with a polynomial-time 4-approximation for MAF on two nonbinary trees. Proving its correctness almost immediately led to an $O(4^k \text{poly}(n))$ time exact algorithm for the same problem. This is presented in Section 4.1. As promised in the previous chapter, we then move on to show that a c-approximation algorithm for nonbinary MAF and a d-approximation algorithm for DFVS can be combined to yield a $d(c+3)$-approximation for nonbinary MAAF in Section 4.2. Combining this with our polynomial-time 4-approximation for MAF, we obtain a $7d$-approximation for MAAF. As we discuss in the conclusion, it is likely that in practice $d = 1$ can be obtained using ILP to solve the generated DFVS instances. Hence, using the 4-approximation for nonbinary MAF we get an exponential-time 7-approximation for nonbinary MAAF and using the FPT algorithm for nonbinary MAF we get an exponential-time 4-approximation for nonbinary MAAF. If we wish a purely polynomial-time approximation, we can use the best-known polynomial-time approximation algorithm for DFVS, yielding overall a polynomial-time $O(\log(k)\log(\log(k)))$-approximation for nonbinary MAAF, with k the size of a maximum acyclic agreement forest minus one.

We start by giving definitions for the nonbinary versions of the concepts we have already encountered in the introduction and previous chapters. Recall that *forest* is defined as a set of trees. To avoid confusion, we call each element of a forest a *component*, rather than a tree. Let T be a tree and $\mathcal{F}$ a forest. We say that $\mathcal{F}$ is a *forest for* T if:

- each component $F \in \mathcal{F}$ is a refinement of $T|L(F)$;
- the subtrees $\{T(L(F)) \mid F \in \mathcal{F}\}$ are edge-disjoint; and

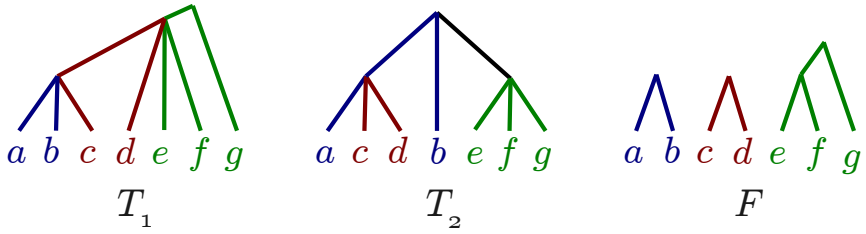

Figure 4.1: Two nonbinary rooted phylogenetic trees T_1 and T_2 and a maximum agreement forest F of T_1 and T_2.

- the union of $L(F)$ over all $F \in \mathcal{F}$ is equal to $L(T)$.

By this definition, if $\mathcal{F}$ is a forest for some tree T, then $\{L(F) \mid F \in \mathcal{F}\}$ is a partition of the leaf set of T. It will indeed sometimes be useful to see an agreement forest as a partition of the leaves, and sometimes to see it as a collection of trees.

If T_1 and T_2 are two trees, then a forest $\mathcal{F}$ is an *agreement forest* of T_1 and T_2 if it is a forest for T_1 and a forest for T_2. The *size* of a forest $\mathcal{F}$, denoted by $|\mathcal{F}|$, is defined as the number of its components. We consider the following computational problem.

Problem: Nonbinary Maximum Agreement Forest (Nonbinary maf)
Instance: Two rooted phylogenetic trees T_1 and T_2.
Solution: An agreement forest $\mathcal{F}$ of T_1 and T_2.
Objective: Minimize $|\mathcal{F}| - 1$.

We define the *inheritance graph* $IG(T_1, T_2, \mathcal{F})$ of an agreement forest, as the directed graph whose vertices are the components of $\mathcal{F}$ and which has an edge (F, F') precisely if either

- there exists a path in T_1 from the root of $T_1(L(F))$ to the root of $T_1(L(F'))$ containing an edge of $T_1(L(F))$ or;
- there exists a path in T_2 from the root of $T_2(L(F))$ to the root of $T_2(L(F'))$ containing an edge of $T_2(L(F))$.

Note that if there exists a path in T_i (for $i \in \{1, 2\}$) from the root of $T_i(L(F))$ to the root of $T_i(L(F'))$ containing an edge of $T_i(L(F))$, then this directly implies that this path also contains such an edge that has the root of $T_i(L(F))$ as tail.

An agreement forest $\mathcal{F}$ of T_1 and T_2 is called an *acyclic agreement forest* if the graph $IG(T_1, T_2, \mathcal{F})$ is acyclic.

We call a forest an *$\mathcal{F}$-splitting* if it is an acyclic agreement forest that can be obtained from a refinement of $\mathcal{F}$ by removing edges and suppressing vertices with in- and outdegree 1.

A *maximum acyclic agreement forest* (maaf) of T_1 and T_2 is an acyclic agreement forest of T_1 and T_2 with a minimum number of components.

Problem: Nonbinary Maximum Acyclic Agreement Forest (Nonbinary maaf)
Instance: Two rooted phylogenetic trees T_1 and T_2.
Solution: An acyclic agreement forest $\mathcal{F}$ of T_1 and T_2.
Objective: Minimize $|\mathcal{F}| - 1$.

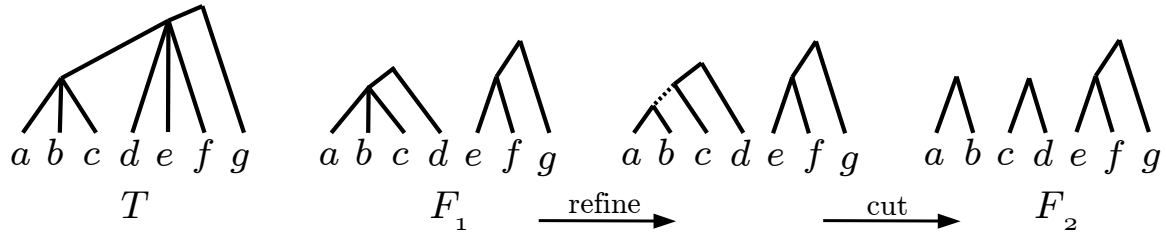

Figure 4.2: A tree T and two forests F_1 and F_2 for T. Forest F_2 can be obtained from F_1 by refining the parent of a, b and c and subsequently removing an edge and suppressing an indegree-1 outdegree-1 vertex.

We use the notation MAF (T_1, T_2) and MAAF (T_1, T_2) for the optimal objective value of, respectively, a maximum agreement forest and a maximum acyclic agreement forest for input trees T_1 and T_2. Hence, if $\mathcal{A}$ is a maximum agreement forest and $\mathcal{M}$ is a maximum acyclic agreement forest, then MAF $(T_1, T_2) = |\mathcal{A}| - 1$ and MAAF $(T_1, T_2) = |\mathcal{M}| - 1$.

4.1 Nonbinary MAF

In the first part of this section we present a 4-approximation algorithm for MAF and prove the following theorem:

Theorem 4.1. *There is a polynomial-time 4-approximation for (nonbinary)* MAF.

The performance analysis leads almost straightforwardly to a fixed parameter tractability result, which we present in the second part as this theorem.

Theorem 4.2. *Nonbinary* MAF *can be solved exactly in $O(4^k poly(n))$ time, with n the number of leaves and k the number of components of a maximum agreement forest minus one.*

Approximation of nonbinary MAF

Let T_1 and T_2 be the input trees to MAF. They do not have to be binary. We will construct a forest by iteratively "cutting" T_2 and collapsing leaves of T_1 and T_2. A "cut" operation of T_2 consists of removing an edge (and suppressing indegree-1 outdegree-1 vertices) or of first refining a vertex (with outdegree greater than 2) and then removing an edge (and suppressing indegree-1 outdegree-1 vertices), see Figure 4.2.

Let F_2 be the forest obtained by cutting T_2 in a particular iteration. In each iteration, we further cut F_2 until at some point F_2 becomes a forest also of T_1. At that point, we have successfully obtained an agreement forest for T_1 and T_2 and we terminate the algorithm.

We describe an algorithm to determine which edges to cut by defining an iteration of the algorithm. Suppose that at the start of the iteration we have an (intermediate) forest F_2. There are two main cases. In each case, the algorithm will make at most 4 cuts, and we will show that at least one of these cuts is unavoidable, thus showing that in each case a 4-approximation is attained.

Take an arbitrary internal vertex u of a reduced T_1 with the property that all its children are leaves. Such a vertex clearly exists in any tree. Let C be the set of children of u in T_1 and $\bar{C}$ the set of all leaves that are not in C.

First the algorithm checks the following three simple cases in the given order.

Case 0a. There exist $c_1, c_2 \in C$ that have a common parent in F_2.

In this case, we collapse the subtree on c_1 and c_2 to a single leaf in both T_1 and F_2. To be precise, we do the following in both T_1 and F_2. If c_1 and c_2 have no siblings, we delete c_1 and c_2 and label their former parent by $\{c_1, c_2\}$. If c_1 and c_2 do have siblings, we delete c_1 and replace label c_2 by label $\{c_1, c_2\}$.

Case 0b. Some leaf $c \in C$ is an isolated vertex in F_2.

In this case, we remove c from both T_1 and F_2 and suppress any resulting outdegree-1 vertices. At the end, after recursively having computed an agreement forest, we add an isolated vertex for c.

Case 0c. The leaves in C are all in different components of F_2.

In this case, we remove all $c \in C$ from both T_1 and F_2 and suppress any resulting outdegree-1 vertices. At the end, after recursively having computed an agreement forest, we add isolated vertices for all $c \in C$.

Correctness of the procedure followed in the first two cases is obvious. To prove correctness in Case 0c, let F be an agreement forest of T_1 and T_2 that can be obtained by cutting F_2. Since the leaves in C are all in different components of F_2, they are all in different components of F. Observe that any component of F that contains an element of C and an element of $\bar{C}$ has to use the edge entering u in T_1. It follows that at most one component of F contains an element of C and an element of $\bar{C}$. Since the elements of C are all in different components of F, it follows that at most one element from C is *not* a singleton in F. Hence, at most one of the cuts made in this step is avoidable. At least 2 cuts were made because $|C| \geq 2$ and no $c \in C$ was already an isolated vertex in F_2 by Case 0b. It follows that at least half of the cuts made were unavoidable.

If none of the above cases applies, the algorithm picks c_1, c_2 from C in such a way that c_1 and c_2 are in the same component A_2 of F_2 and such that their lowest common ancestor in A_2 is at maximum distance from the root of the component. Moreover, if there exists such a pair for which neither of c_1 and c_2 is a child of their lowest common ancestor, we pick such a pair first.

Case 1. Neither of c_1 and c_2 is a child of their lowest common ancestor v in F_2.

Let p_1 be the parent of c_1 and p_2 the parent of c_2 in F_2. (We have $p_1 \neq p_2$ because we are not in Case 0a, but also because otherwise the lowest common ancestor of v would be $p_1 = p_2$.) Let S_1 be the set of all leaves that are descendants of p_1 except for c_1. Similarly, let S_2 be the set of all leaves that are descendants of p_2 except for c_2. Finally, let R be the set of other leaves of component A_2, i.e. leaves that are not in $S_1 \cup S_2 \cup \{c_1, c_2\}$. We cut A_2 by creating separate components for c_1, c_2, S_1, S_2 and R, thus making four cuts, see Figure 4.3.

We claim that at least one of these four cuts was unavoidable. Let F be an agreement forest of T_1 and T_2 with a minimum number of components. If any of c_1, c_2 is a singleton in F, then the cut that removed the edge entering that vertex was unavoidable. Hence, we assume that c_1 and c_2 are both non-singletons in F. Suppose c_1 and c_2 are in the same component of F. Because of the choice of c_1 and c_2 having their lowest common ancestor furthest from the root of the component, $S_1, S_2 \subset \bar{C}$; they contain no element of C. Hence cutting off both S_1 and S_2 is unavoidable.

Hence, we assume that c_1 and c_2 are in different, non-singleton components of F. As argued before, in justifying Case 0c, at most one of c_1 and c_2 can be in a component with elements from $\bar{C}$. Moreover, as also argued before, $S_1, S_2 \subset \bar{C}$. Therefore, at most one of c_1 and c_2 can be in a component with elements from $S_1 \cup S_2$. W.l.o.g. suppose c_1 is not in a component with elements from $S_1 \cup S_2$. Since c_1 is not a singleton component, it is contained in a component which uses the edge of A_2 entering p_1 and the edge from p_1 to c_1, but none of the other edges leaving p_1. Hence, the cut cutting off S_1 is unavoidable. Similarly, if c_2 is not in a component with elements from $S_1 \cup S_2$, then the cut cutting off S_2 is unavoidable. Thus, always at least one

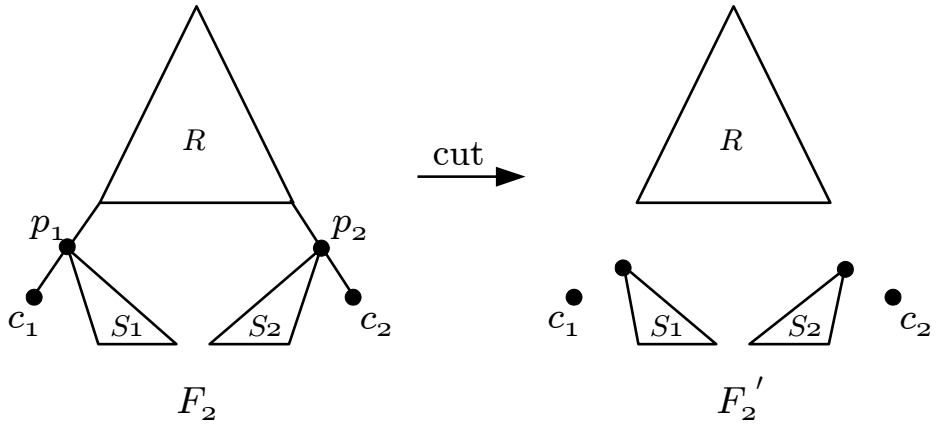

Figure 4.3: Case 1: none of c_1 and c_2 is a child of their lowest common ancestor. We cut F_2 into F_2' by creating separate components for c_1, c_2, S_1, S_2 and R.

of the four cuts was unavoidable.

Case 2. Either c_1 or c_2 is a child of their lowest common ancestor v in F_2.

Let c_1, c_2 be any such pair. Let again p_1 be the parent of c_1 and p_2 the parent of c_2 in F_2. Assume without loss of generality that p_1 is the lowest common ancestor of c_1 and c_2. Let S_2 contain all leaves that are descendants of p_2 except for c_2. Let S_1 contain all leaves that are descendants of p_1 except for c_1, c_2 and the leaves in S_2. Let R contain all remaining leaves. We cut F_2 by creating separate components for c_1, c_2, S_1, S_2 and R, thus making four cuts, see Figure 4.4.

We claim that also in this case at least one of the four cuts was unavoidable. Let F again be an agreement forest of T_1 and T_2 with a minimum number of components. We can argue as before that we can restrict attention to the situation in which c_1 and c_2 are non-singletons and belong to different components. It is also again true that S_1 and S_2 cannot contain elements from C. To see this, first note that no element c_3 of $C \setminus \{c_1\}$ can be a child of p_1 because then $c_1, c_3 \in C$ would have a common parent in F_2, which is a Case 0a situation. Moreover, no $c_3 \in C \setminus \{c_1, c_2\}$ can be reached from p_1 by a directed path with at least one internal vertex that does not belong to the path from p_1 to c_2, because then c_2, c_3 would conform to Case 1. Finally, as before, no $c_3 \in C \setminus \{c_1, c_2\}$ can be reached from an internal vertex of the path from p_1 to c_2 because then c_3 and c_2 would have a lowest common ancestor further away from the root than p_1. We conclude that S_1 and S_2 contain no elements from C.

As in Case 1, it follows that at most one of c_1 and c_2 is in a component with elements from $S_1 \cup S_2$. Also, similar to the arguments in Case 1, if c_1 is not in a component with elements from $S_1 \cup S_2$ then that component uses the edge of A_2 entering p_1 and the edge from p_1 to c_1, but no other edges leaving p_1. Hence, the cutting off $S_1 \cup S_2 \cup \{c_2\}$ is unavoidable. If c_2 is not in a component with elements from $S_1 \cup S_2$, then cutting off S_2 is unavoidable.

Hence, we conclude that in each case at least one of the four cuts is unavoidable, and the algorithm thus yields a 4-approximation.

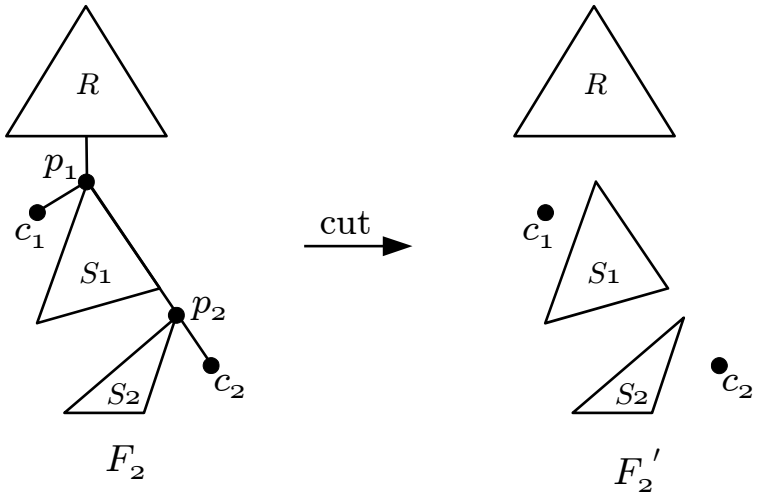

Figure 4.4: Case 2: c_1 is a child of the lowest common ancestor of c_1 and c_2. We cut F_2 into F_2' by creating separate components for c_1, c_2, S_1, S_2 and R.

An FPT algorithm for nonbinary MAF

In this section, we show that there exists an $O(4^k \text{poly}(n))$ time algorithm for nonbinary MAF, i.e., we prove Theorem 4.2.

The algorithm follows the same ideas as the approximation algorithm in the proof of Theorem 4.1. Cases 0a and 0b are executed in exactly the same way. In Case 0c, instead of removing all $c \in C$, we pick $c_1, c_2 \in C$ arbitrarily and branch into two subproblems. In one subproblem c_1 is removed and in the other subproblem c_2 is removed. After recursively computing an agreement forest, the removed leaf is added as an isolated vertex. This step is correct since, by the proof of Theorem 4.1, in any maximum agreement forest at least one of c_1 and c_2 is an isolated vertex. For Cases 1 and 2, instead of making four cuts, we branch into four subproblems, one for each possible cut. By the proof of Theorem 4.1, at least one of the four cuts is unavoidable, and hence at least one subproblem has the same optimum as the original problem.

It remains to analyze the running time. In each step we branch into at most four subproblems. For each subproblem we make one cut, and hence we reduce the parameter k by one. Therefore, at most 4^k subproblems are created. For each subproblem, we need only time polynomial in n. This concludes the proof.

4.2 Approximating nonbinary MAAF

In this section we relate approximability of Nonbinary MAAF to that of Nonbinary MAF and DFVS and prove the following theorem:

Theorem 4.3. *If there exists a c-approximation for Nonbinary* MAF *and a d-approximation for* DFVS*, then there exists a $d(c+3)$-approximation for Nonbinary* MAAF *and hence for Nonbinary* MH.

Combining this Theorem with a Theorem 4.1 from the previous section, we obtain the following corollary.

Corollary 4.4. *If there exists a d-approximation for* DFVS *then there exists a $7d$-approximation for Nonbinary* MAAF *and hence for Nonbinary* MH*.*

Moreover, using the $O(\log(\tau)\log(\log(\tau)))$-approximation for weighted DFVS from [25], with τ the weight of a minimum feedback vertex set, we also obtain the following.

Corollary 4.5. *There exists a polynomial-time $O(\log(k)\log(\log(k)))$-approximation for nonbinary* MAAF*, with k the number of components of a maximum acyclic agreement forest minus one.*

An agreement forest $\mathcal{A}$ is said to be *maximal* if there is no agreement forest that can be obtained from $\mathcal{A}$ by merging components. It is clear that, given any agreement forest, a maximal agreement forest with at most as many components can be obtained in polynomial time.

Let T_1 and T_2 be two nonbinary trees. Consider a maximal agreement forest $\mathcal{A}$ and let $\mathcal{M}$ be some maximum acyclic agreement forest for these two trees. We will first prove that there exists an $\mathcal{A}$-splitting of size at most $|\mathcal{A}| + 3|\mathcal{M}|$. After that, we will show how the problem of finding an optimal $\mathcal{A}$-splitting can be reduced to DFVS.

The idea of the first part of the proof is to split components of $\mathcal{A}$ according to $\mathcal{M}$. We show that to make $\mathcal{A}$ acyclic we will increase the number of components of $\mathcal{A}$ by at most three times the size of $\mathcal{M}$.

We start with some definitions. In the first part of the proof, we see an agreement forest $\mathcal{A}$ for T_1 and T_2 as a partition of the leaf set X for which holds that:

1. $T_1|A$ and $T_2|A$ have a common refinement, for all $A \in \mathcal{A}$; and
2. the subtrees $\{T_i(A) \mid A \in \mathcal{A}\}$ are edge-disjoint, for $i \in \{1, 2\}$.

Using this definition of agreement forests, an $\mathcal{A}$-splitting is an acyclic agreement forest that can be obtained by splitting components of $\mathcal{A}$. The following observation is easily verifiable.

Observation 4.6. *If $\mathcal{M}$ is an acyclic agreement forest and $\mathcal{M}'$ is an agreement forest that can be obtained from $\mathcal{M}$ by splitting components, then $\mathcal{M}'$ is an acyclic agreement forest.*

For a component A of an agreement forest for two trees T_1 and T_2, we write $r_i(A)$ to denote the root of $T_i(A)$, for $i \in \{1, 2\}$. For two components M and M' of $\mathcal{M}$, we say that M is *lower* than M' if in the inheritance graph of $\mathcal{M}$ there is a directed path (and hence an edge) from M' to M. Since the inheritance graph of $\mathcal{M}$ is acyclic, $\mathcal{M}$ contains some lowest element. Moreover, for any subset of components of $\mathcal{M}$, there exists a component that is lowest over that subset. For a component A of $\mathcal{A}$ and a component M of $\mathcal{M}$, we say that M *properly intersects* A (and that A is *properly intersected by* M) if $M \cap A \neq \emptyset$ and $A \setminus M \neq \emptyset$.

We are now ready to describe the procedure for splitting $\mathcal{A}$. Initially, all components of $\mathcal{A}$ and $\mathcal{M}$ are unmarked. We iteratively choose a component M^* of $\mathcal{M}$ that is lowest over all unmarked components of $\mathcal{M}$. For each component A of $\mathcal{A}$ that is properly intersected by M^*, we split A into two components $A \cap M^*$ and $A \setminus M^*$ and we mark $A \cap M^*$. Then we mark M^* and any unmarked components A' of $\mathcal{A}$ with $A' \subseteq M^*$ and proceed to the next iteration. We continue this procedure until all components of $\mathcal{M}$ are marked.

It is clear that, if $\mathcal{A}^*$ is the agreement forest obtained at the end of some iteration, then no marked component of $\mathcal{M}$ properly intersects any component of $\mathcal{A}^*$. Moreover, no marked component of $\mathcal{A}^*$ is properly intersected by any component of $\mathcal{M}$.

Lemma 4.7. *Let $\mathcal{A}$ be an agreement forest for T_1 and T_2, let M^* be a component of $\mathcal{M}$ that is lowest over all unmarked components of $\mathcal{M}$ and let A be a component of $\mathcal{A}$ that is properly intersected by M^*. Then, $T_i(A \cap M^*)$ and $T_i(A \setminus M^*)$ are edge-disjoint subtrees of T_i, for $i \in \{1, 2\}$.*

Proof. Suppose to the contrary that in at least one tree, say T_1, there exists an edge e such that $e \in T_1(A \cap M^*)$ and $e \in T_1(A \setminus M^*)$. Consider the set of leaves S that can be reached from e. Clearly, some leaf of $A \setminus M^*$ has to be in S. Let $x \in S \cap (A \setminus M^*)$. Clearly, x must be in some component of $\mathcal{M}$ other than M^*. Call that component M'. Since components of $\mathcal{M}$ are edge disjoint, $r_1(M')$ has to be below e. Because $e \in T_1(A \cap M^*)$, it follows that $e \in T_1(M^*)$ and hence that M' is lower than M^*. Since M^* is lowest over all unmarked components of $\mathcal{M}$, it follows that M' is marked and hence that M' does not properly intersect any component of $\mathcal{A}$. This is a contradiction because M' properly intersects A. □

Lemma 4.7 shows that when we split a component, the resulting two subtrees are edge-disjoint. This property holds for all components properly intersected by M^*. Observe that when we split a component, the two newly-created subtrees cannot possibly share edges with other components, because of the assumption that at the start of the iteration all the subtrees were edge-disjoint. Hence, at the end of the iteration, all of the subtrees are mutually edge-disjoint. To show that we still have an agreement forest, it is necessary to show that the components still obey the refinement criterion. This follows from the following (unsurprising) observation.

Observation 4.8. *Let T_1 and T_2 be two trees on taxon set X such that T_1 and T_2 have a common refinement. Then, for any $X' \subseteq X$, $T_1|X'$ and $T_2|X'$ have a common refinement.*

We have now shown that the result of each iteration is an agreement forest for T_1 and T_2 such that no marked component of $\mathcal{M}$ properly intersects any component of this agreement forest. Let $\mathcal{A}'$ be the agreement forest obtained at the end of the last iteration. Since at the end of the procedure all components of $\mathcal{M}$ are marked, no component of $\mathcal{M}$ properly intersects any component of $\mathcal{A}'$. It follows that $\mathcal{A}'$ can be obtained from $\mathcal{M}$ by splitting components. Since $\mathcal{M}$ is acyclic, it follows by Observation 4.6 that $\mathcal{A}'$ is acyclic. Hence, $\mathcal{A}'$ is an $\mathcal{A}$-splitting. It remains to bound the size of this $\mathcal{A}$-splitting. To do so, we will need the following observation and lemma.

Observation 4.9. *Let A and B be components of some agreement forest for T_1 and T_2. If A and B have the same root in both T_1 and T_2, then the result of merging A and B into a single component $A \cup B$ is still an agreement forest of T_1 and T_2.*

Lemma 4.10. *If $\mathcal{A}$ is the agreement forest at the beginning of some iteration, then there are no four unmarked components of $\mathcal{A}$ such that in both trees the four components have a same vertex.*

Proof. First let $\mathcal{A}$ be the agreement forest at the beginning of the first iteration. Suppose that A_1, A_2, A_3, A_4 are unmarked components of $\mathcal{A}$ and that v_i is common to $T_i(A_1), \ldots, T_i(A_4)$ for $i \in \{1, 2\}$. For each $i \in \{1, 2\}$, there is at most one $j \in \{1, 2, 3, 4\}$ for which $T_i(A_j)$ contains the edge entering v_i. Hence, there are at least two components, say A_1 and A_2, that do not use this edge in either tree. It follows that $r_i(A_1) = r_i(A_2) = v_i$ for $i \in \{1, 2\}$. However, then A_1 and A_2 can be merged into a single component by Observation 4.9, which is a contradiction because $\mathcal{A}$ is maximal.

We have shown that the lemma is true at the beginning of the first iteration. Now assume that it is true at the beginning of some iteration. Each component that is split during the iteration is split into one marked and one unmarked component. Hence, for each vertex, the number of unmarked components using that vertex does not increase. It follows that the lemma is still true at the end of the iteration. This completes the proof. □

Lemma 4.11. *Let $\mathcal{A}$ be the agreement forest at the beginning of some iteration. If M^* is a component of $\mathcal{M}$ that is lowest over all unmarked components of $\mathcal{M}$, then M^* properly intersects at most three components of $\mathcal{A}$.*

Proof. Let A be a component of $\mathcal{A}$ that is properly intersected by M^*. We claim that there is a directed path from $r_i(A)$ to $r_i(M^*)$ for $i \in \{1,2\}$ (possibly, $r_i(A) = r_i(M^*)$). To see this, first note that, since $A \cap M^* \neq \emptyset$, there must be either a directed path from $r_i(A)$ to $r_i(M^*)$, or from $r_i(M^*)$ to $r_i(A)$, for $i \in \{1,2\}$. Suppose that this path goes from $r_i(M^*)$ to $r_i(A)$ and contains at least one edge. Since $A \cap M^* \neq \emptyset$, this path has to contain at least one edge of $T_i(M^*)$. Now observe that, since M^* properly intersects A, there exists some $a \in A \setminus M^*$, which is in some component of $\mathcal{M}$, say in M'. Note that M' is unmarked since it properly intersects A. However, then we obtain a contradiction because M' is lower than M^*, while M^* is lowest over all unmarked components of $\mathcal{M}$. Hence, there is a directed path from $r_i(A)$ to $r_i(M^*)$ for $i \in \{1,2\}$. This path is contained in $T_i(A)$ because $A \cap M^* \neq \emptyset$.

Now assume that the lemma is not true, i.e. that there exist four components A_1, A_2, A_3, A_4 of $\mathcal{A}$ that are properly intersected by M^*. We have seen that there is a directed path from $r_i(A_j)$ to $r_i(M^*)$, which is contained in $T_i(A_j)$, for $i \in \{1,2\}$ and $j \in \{1,2,3,4\}$. Hence, $r_i(M^*)$ is common to all four components A_1, A_2, A_3, A_4. Moreover, A_1, A_2, A_3, A_4 are all unmarked since they are properly intersected by M^*. This is a contradiction to Lemma 4.10. □

Lemma 4.11 shows that each iteration splits at most three components. Each split adds one component. Thus, for every component M of $\mathcal{M}$, the number of components of the $\mathcal{A}$-splitting is increased by at most three, which concludes the proof of the following theorem.

Theorem 4.12. *Let T_1 and T_2 be two (nonbinary) trees. If $\mathcal{A}$ is an agreement forest and $\mathcal{M}$ a maximum acyclic agreement forest of T_1 and T_2, then there exists an $\mathcal{A}$-splitting of size at most $|\mathcal{A}| + 3|\mathcal{M}|$.*

Now, suppose we have a c-approximation $\mathcal{A}$ for MAF; i.e.,

$$|\mathcal{A}| - 1 \leq c \cdot \mathrm{MAF}(T_1, T_2) \leq c \cdot \mathrm{MAAF}(T_1, T_2).$$

Let *OptSplit*$(\mathcal{A})$ denote the size of an $\mathcal{A}$-splitting with smallest number of components. The last inequality together with Theorem 4.12 imply that

$$\mathit{OptSplit}(\mathcal{A}) - 1 \leq |\mathcal{A}| - 1 + 3 \cdot \mathrm{MAAF}(T_1, T_2) \leq (c+3) \cdot \mathrm{MAAF}(T_1, T_2).$$

Reducing optimal $\mathcal{A}$-splitting to DFVS

The remaining part of the proof will be accomplished by reducing the problem of finding an optimal $\mathcal{A}$-splitting to DFVS in such a way that a d-approximation algorithm for DFVS gives a d-approximation for an optimal $\mathcal{A}$-splitting. Combining this with the above inequality, Theorem 4.3 will follow.

From now on, we see an agreement forest as a collection of trees, as specified in the preliminaries section. We can and will assume that the components of $\mathcal{A}$ are never more refined than necessary (i.e. there is no agreement forest of T_1 and T_2 that can be obtained from $\mathcal{A}$ by contracting edges).

To prepare for the construction of an input graph to DFVS, we label the vertices and edges of T_1 and T_2 by the vertices and edges of $\mathcal{A}$ that they correspond to. Note that each vertex of $\mathcal{A}$ is used exactly once as a label but, due to refinements, some vertices of the trees can have multiple labels. Moreover, some edges of $\mathcal{A}$ can be used as a label multiple times (when the edge corresponds to a path in a tree) but each edge is used as a label at least once in at least one tree (by the assumption that $\mathcal{A}$ is not more refined than necessary). For example, for the trees and agreement forest in Figure 4.5, the labelling is illustrated in Figure 4.6.

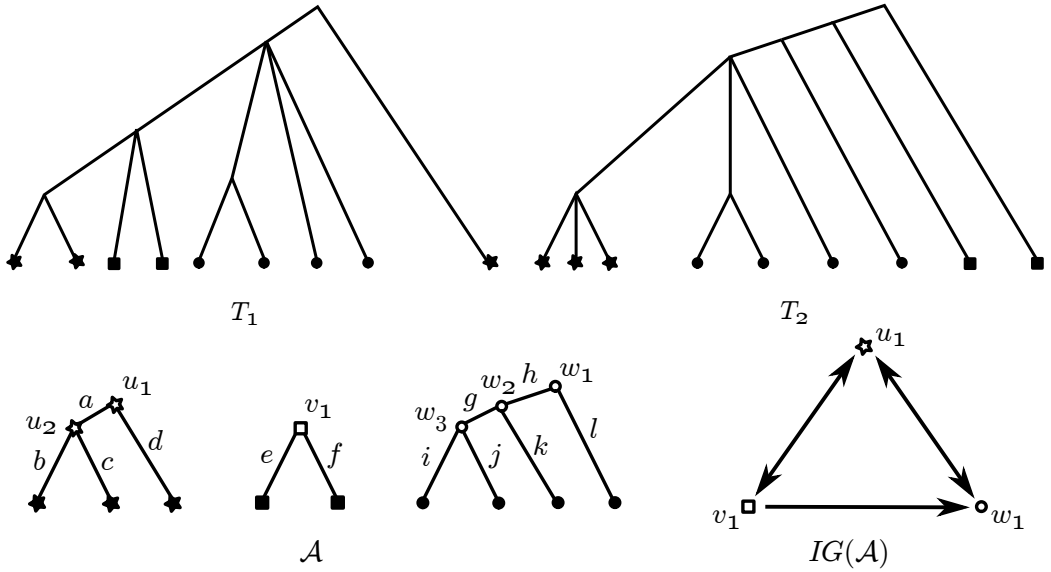

Figure 4.5: Two trees T_1 and T_2, an agreement forest $\mathcal{A}$ for T_1 and T_2 and its inheritance graph $IG(\mathcal{A})$. Leaf labels have been omitted.

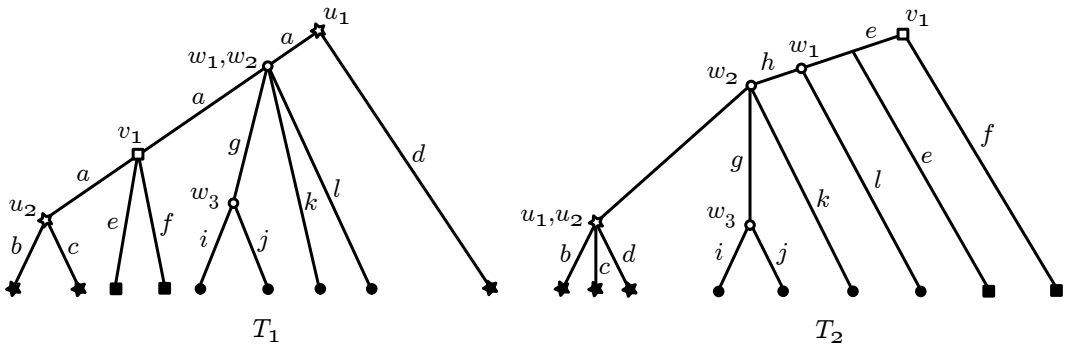

Figure 4.6: The trees T_1 and T_2 from Figure 4.5 labelled with the internal vertices and all edges of $\mathcal{A}$.

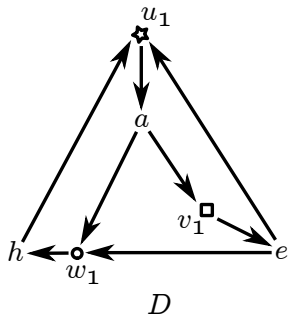

Figure 4.7: Part of the input graph D for DFVS for the trees T_1, T_2 and agreement forest $\mathcal{A}$ of Figures 4.5 and 4.6. Vertices that do not appear in any directed cycle of D have been omitted. Minimum feedback vertex sets of D are $\{u_1\}$ and $\{a\}$.

We construct an input graph D for DFVS as follows. For every internal vertex of $\mathcal{A}$, we create a vertex for D. Denote the set of these vertices by $V_V(D)$. In addition, for every edge of $\mathcal{A}$ we create a vertex for D. This set of vertices is denoted by $V_E(D)$. We write $V(D) = V_V(D) \cup V_E(D)$. We create edges of D as follows. For every $v \in V_V(D)$ and every $e \in V_E(D)$ create an edge (v, e) if $tail(e) = v$ in $\mathcal{A}$ and we create an edge (e, v) if $\hat{v}$ can be reached from $head(\hat{e})$ in at least one tree, for some edge $\hat{e}$ labelled by e and the vertex $\hat{v}$ labelled by v. See Figure 4.7 for an example.

We will show that any feedback vertex set of D corresponds to an $\mathcal{A}$-splitting and vice-versa. Moreover, after appropriately weighting the vertices of D, the number of "splits" of the $\mathcal{A}$-splitting (i.e. the size of the $\mathcal{A}$-splitting minus the size of $\mathcal{A}$) will be equal to the weight of the corresponding weighted feedback vertex set.

Let $F \subset V(D)$. In what follows, we use F both for sets of vertices in D and for the set of vertices and edges they represent in $\mathcal{A}$. Let $\mathcal{A} \setminus F$ be the forest obtained from $\mathcal{A}$ by removing the vertices and edges of F and repeatedly removing indegree-0 outdegree-1 vertices and suppressing indegree-1 outdegree-1 vertices.

Lemma 4.13. *A subset F of $V(D)$ is a feedback vertex set of D if and only if $\mathcal{A} \setminus F$ is an $\mathcal{A}$-splitting.*

Proof. Since F can contain only inner vertices of $\mathcal{A}$ it is clear that $\mathcal{A} \setminus F$ is an agreement forest of T_1 and T_2. Hence, it is enough to show that $D \setminus F$ has a directed cycle if and only if $IG(\mathcal{A} \setminus F)$ has a directed cycle.

First, let $C_1, C_2, \ldots, C_k = C_1$ be a cycle in $IG(\mathcal{A} \setminus F)$. We will show that this implies that there exists a cycle in $D \setminus F$. Let $u_1, u_2, \ldots, u_k$ be the roots of the components $C_1, C_2, \ldots, C_k$, respectively. Notice that $u_1, u_2, \ldots, u_k$ are internal vertices of $\mathcal{A}$ and hence represented in D. Moreover, since these vertices are in $\mathcal{A} \setminus F$, they are also in $D \setminus F$.

We prove this side of the lemma by showing that the presence of an edge (C_i, C_{i+1}) of $IG(\mathcal{A} \setminus F)$ implies the existence of a directed path from u_i to u_{i+1} in $D \setminus F$, for $i = 1, \ldots, k-1$. By the definition of the inheritance graph, an edge (C_i, C_{i+1}) of $IG(\mathcal{A} \setminus F)$ implies that there exists a directed path from u_i to u_{i+1} that uses an edge of C_i in at least one of the trees. It is easy to see that this directed path uses an edge, a say, of C_i that has u_i as its tail. Moreover, since C_i and C_{i+1} are components of $\mathcal{A} \setminus F$, u_1, a and u_{i+1} are vertices of $D \setminus F$ and (u_i, a) and (a, u_{i+1}) are edges of $D \setminus F$, forming a directed path from u_i to u_{i+1} in $D \setminus F$.

Now, assume that $D \setminus F$ contains a cycle. Let $u_1, \ldots, u_k = u_1$ be a longest cycle. We will show that this implies the existence of a cycle in $IG(\mathcal{A} \setminus F)$. Clearly, $u_1, \ldots, u_k$ cannot all correspond to vertices and edges from the same component of $\mathcal{A} \setminus F$. From the assumption that

$u_1, \ldots, u_k = u_1$ is a longest cycle, it can be argued that there are at least two vertices among $u_1, \ldots, u_k$ that are roots of components of $\mathcal{A} \setminus F$. Let $r_1, \ldots, r_s = r_1$ denote those vertices of the cycle that correspond to roots of components. Let $C_1, \ldots, C_s$ be the corresponding components of $\mathcal{A} \setminus F$, respectively. We will show that $C_1, \ldots, C_s$ form a cycle in $IG(\mathcal{A} \setminus F)$.

For $i = 1, \ldots, s-1$, we show that there exists an edge (C_i, C_{i+1}) in $IG(\mathcal{A} \setminus F)$. First observe that there exists a directed path from r_i to r_{i+1} in $D \setminus F$ and that, since D is bipartite, this path is of the form

$$(r_i = v_0, e_1, v_1, e_2, \ldots, e_t, v_t = r_{i+1})$$

where $e_1, \ldots, e_t$ are edges of $\mathcal{A} \setminus F$ and $v_1, \ldots, v_{t-1}$ are internal vertices of C_i.

First assume $t = 1$. In this case, the path is just (r_i, e_1, r_{i+1}). By the definition of D, $tail(e_1) = r_i$ in C_i and there is a directed path from $head(\hat{e}_1)$ to r_{i+1}, with $\hat{e}_1$ some edge labelled by e_1, in at least one of the two trees (such an edge $\hat{e}_1$ definitely exists because of the assumption that $\mathcal{A}$ is not more refined that necessary). This tree then contains a path from r_i to r_{i+1} that contains edge $\hat{e}_1$ of C_i. Hence there is an edge (C_i, C_{i+1}) in $IG(\mathcal{A} \setminus F)$.

Now assume $t > 1$. For $j = 1, \ldots, t$, by the definition of D, $tail(e_j) = v_{j-1}$ in C_i and there is a directed path from $head(\hat{e}_j)$ to v_j, with $\hat{e}_j$ some edge labelled by e_j, in at least one of the two trees. For $j < t$, e_j and v_j are both in the same component C_i, implying that there is a directed path from $head(e_j)$ to v_j in C_i. Hence, there is a directed path from r_i to v_{t-1} in both trees. Thus, the tree that contains a directed path from v_{t-1} to r_{i+1} also contains a directed path from r_i to r_{i+1} containing edge $\hat{e}_t$ of C_i. Hence, (C_i, C_{i+1}) is an edge of $IG(\mathcal{A} \setminus F)$. □

The above lemma showed that we can turn $\mathcal{A}$ into an $\mathcal{A}$-splitting by removing vertices and edges corresponding to a feedback vertex set of D. We will now add weights to the vertices of D in order to enforce that an optimal feedback vertex set gives an optimal $\mathcal{A}$-splitting and, moreover, that an approximate feedback vertex set gives an approximate $\mathcal{A}$-splitting.

We define a weight function w on vertices of D as follows, using $\delta^+_{\mathcal{A}}(v)$ to denote the outdegree of a vertex v in $\mathcal{A}$.

$$w(v) = \begin{cases} \delta^+_{\mathcal{A}}(v) - 1 & \text{if } v \in V_V(D) \\ 1 & \text{if } v \in V_E(D). \end{cases}$$

The weight of a feedback vertex set F is defined as $w(F) = \sum_{v \in F} w(v)$.

Intuitively, the weight of each vertex v equals the number of components its removal would add to the $\mathcal{A}$-splitting.

To make this precise, we need the following definition. We call a feedback vertex set F *proper* if it is minimal (i.e. no proper subset of F is a feedback vertex set) and if for every vertex $v \in V_V(D)$ at most $\delta^+_D(v) - 2$ children of v in D are contained in F, with $\delta^+_D(v)$ denoting the outdegree of v in D. (Recall that the children of v in D are elements of $V_E(D)$). For example, proper feedback vertex sets of the graph D in Figure 4.7 are $\{u_1\}$ and $\{v_1, w_1\}$. The idea behind this definition is that when all, or all but one, of the children of v are in F, then we could just as well add v instead, which does not increase the total weight of the feedback vertex set. Hence, given any feedback vertex set F, a proper feedback vertex set F' with $w(F') \leq w(F)$ can be found in polynomial time.

Lemma 4.14. *If F is a proper feedback vertex set of D, then $\mathcal{A} \setminus F$ is an $\mathcal{A}$-splitting of size $|\mathcal{A}| + w(F)$.*

Proof. For an edge e of $\mathcal{A}$, we use $tail(e)$ and $head(e)$ to refer to the tail and head of e in $\mathcal{A}$, respectively.

Consider a vertex of F corresponding to an edge e of $\mathcal{A}$. Then, every cycle in D that contains e also contains $tail(e)$. Hence, since F is minimal, $tail(e)$ is not contained in F. Moreover, suppose

that $tail(e)$ is not the root of a component of $\mathcal{A}$ and let e' be the edge with $head(e') = tail(e)$. Then, for every cycle in D containing e but not e' (and hence containing $tail(e)$ but not $tail(e')$), replacing e and $tail(e)$ by e' and $tail(e')$ gives again a directed cycle in D. Hence, since F is minimal, it follows that either e' or $tail(e')$ is contained in F, but not both.

Now consider a vertex of F corresponding to a vertex v of a component C of $\mathcal{A}$. Then, by the previous paragraph, no edge e with $tail(e) = v$ is contained in F. Moreover, if v is not the root of C and e' is the edge with $head(e') = v$, then, as before, either e' or $tail(e')$ is contained in F, but not both.

To summarize the previous two paragraphs, if F contains a vertex corresponding to a vertex v of $\mathcal{A}$ that is *not* the root of its component, then F also contains either the edge entering v or the parent of v. Similarly, if F contains a vertex corresponding to an edge e of $\mathcal{A}$ that is *not* leaving the root of its component, then F also contains either the edge entering $tail(e)$ or the parent of $tail(e)$. Informally speaking, this means that if you remove something from a component, you also remove everything above it.

More formally, it follows that $\mathcal{A} \setminus F$ can be obtained from $\mathcal{A}$ by, repeatedly, either removing the root of a component or removing a subset of the edges leaving the root of a component. Moreover, if we remove a subset of the edges leaving the root of a component, at least two of these edges are not removed, because F is proper.

Removing edges and vertices from $\mathcal{A}$ in this way, it is easy to see that the number of components is increased by $w(F)$. □

To complete the proof of Theorem 4.3, let $\mathcal{A}$ be a c-approximation to MAF, F a minimal feedback vertex set that is a d-approximation to weighted DFVS on D and F^* an optimal solution to weighted DFVS on D, then we have:

$$\begin{aligned}
|\mathcal{A} \setminus F| - 1 &= |\mathcal{A}| + w(F) - 1 \\
&\leq |\mathcal{A}| + d \cdot w(F^*) - 1 \\
&\leq d(|\mathcal{A}| + w(F^*) - 1) \\
&= d(\mathit{OptSplit}(\mathcal{A}) - 1) \\
&\leq d(c+3)\,\mathrm{MAAF}(T_1, T_2).
\end{aligned}$$

We have thus shown how to construct an agreement forest that is a $d(c+3)$-approximation to MAAF and with that we conclude the proof of Theorem 4.3.

4.3 Experiments

To run the simulations with NonbinaryCycleKiller, we used a subset of the trees from the easy set of binary experiments. We then applied random edge contractions in order to obtain nonbinary trees. Hence, we have the same two parameters as before, namely the number of taxa $n \in \{20, 50, 100\}$ and the number of rSPR-moves $k \in \{5, 10, 15, 20\}$, and an additional parameter $\rho \in \{25, 50, 75\}$ which measures the percentage of the edges of an original binary tree that were contracted in order to obtain a nonbinary tree. We could only use smaller values of n and k from the easy set of experiments because exact solvers for nonbinary MAF (upon which NonbinaryCycleKiller is built) and exact solvers for nonbinary MinimumHybridization (which is important to measure the accuracy of NonbinaryCycleKiller in practice) are slower than their binary counterparts.

We performed two runs of experiments. One run with instances consisting of one binary and one nonbinary tree, and one run with instances consisting of two nonbinary trees.

For the experiments with one binary and one nonbinary tree, we were still able to use the rSPR algorithm [81, 83], which has a better running time and approximation ratio compared to the available algorithm for two nonbinary trees. When rSPR is used in exact mode, NonbinaryCycleKiller yields a theoretical worst-case approximation ratio of 4. When rSPR is used in its 3-approximation mode, NonbinaryCycleKiller yields a theoretical worst-case approximation ratio of 6. The results of this run are summarized in Table 4.8.

For the experiments with two nonbinary trees, the rSPR software can no longer be used, and instead we used the exact and 4-approximate MAF algorithm described in [45]. This makes NonbinaryCycleKiller behave as a 4-approximation and 7-approximation respectively. Note that the exact algorithm [45] is considerably slower than rSPR, meaning that in practice NonbinaryCycleKiller struggles with two nonbinary trees more than when at most one of the trees is nonbinary. The results for this run are summarized in Table 4.9.

The exact hybridization number in both runs was computed by TerminusEst [54].

Each instance that took longer than 10 minutes to compute was aborted and the running time was set to 600 seconds. The averages of the running-times are taken over all instances, with running-time taken to be 600 if the program timed out for that instance. (We used a shorter time-out than in the binary experiments because of the observation that, in the nonbinary case, exact algorithms running longer than 10 minutes almost always took longer than 60 minutes too.)

Note that we did not compare the performance of NonbinaryCycleKiller to Dendroscope because TerminusEst has better running times than the exact nonbinary MinimumHybridization solver inside Dendroscope (data not shown).

To enable a clearer analysis we divided the trees into representative "simple" and "tricky" ones based on two parameters, n and k. Parameter values for the simple set were $n \in \{20, 50\}$, $k \in \{5, 10, 15\}$ and for the tricky set $n \in \{50, 100\}$, $k = 20$. In addition we varied the percentage of contracted edges (in a single tree in the first run and in both trees in the second run).

In Table 4.8 we show running times and solution quality of our algorithm when one of the input trees is binary. For the simple set of instances (regardless of the percentage of edge-contractions) we see that the more accurate version of our algorithm, the 4-approximation, had a better running time than the exact algorithm, and at the same time had an average approximation ratio very close to 1. Far more interesting is to see what happens with tricky instances. As predicted, the running time of the exact algorithm is much higher for tricky instances due to the higher hybridization numbers. On the other hand, the running time of the 4-approximation does not rise significantly at all, whilst still attaining an approximation ratio again very close to 1. Another thing to note is that the percentage of contraction only seems to affect the running time of the exact algorithm. The practical worst-case approximation ratio observed in these experiments was 1.75 for the 4-approximation and 3 for the 6-approximation.

Table 4.9 shows our results on instances with two nonbinary trees. The exact algorithm for MAF is in this case much slower and this affects the running times even for the simple set. While the 4-approximation version has an average approximation ratio very close to 1 again, the running time is in this case worse than that of TerminusEst. For the tricky set the situation is even more significant; the exact MAF algorithm cannot deal with reticulation numbers above 15, while TerminusEst can get slightly further. On the other hand, the 7-approximation still runs much faster than TerminusEst, both for simple and tricky instances, while having an average approximation ratio of less than 2.6. The practical worst-case approximation ratio observed in these experiments was 1.5 for the 4-approximation and 4 for the 7-approximation.

It is worth noting that, for the 4-approximation, the running time for the 75%-contraction trees is considerably lower than the one for the 50%-contraction trees. This is due to the fact that a high contraction in both trees causes the hybridization number of the instance to drop,

and a lower hybridization number leads to a better running time. Also note that the exact solver TerminusEst seems more able to cope with the tricky 25%-contraction instances than the tricky 50%-contraction instances. This is probably because, although low contraction rates yield a higher hybridization number, the trees remain "relatively binary" and this can induce more efficient branching in the underlying FPT algorithm [68]. It is plausible that with 50%-contraction the instances suffer from the disadvantage of relatively high hybridization number without the branching advantages associated with (relatively) binary trees.

T_1 binary T_2 nonbinary		TerminusEst		NBCK 4-appx			NBCK 7-appx		
		Opt	Time	Opt	Time	Ratio	Opt	Time	Ratio
25.00%	Easy	7.504	8.004	7.567	0.967	1.007	11.421	0.996	1.532
	Hard	17.000	203.650	17.288	3.675	1.003	27.238	3.638	1.600
50.00%	Easy	6.736	1.450	5.531	0.919	1.003	9.231	0.919	1.690
	Hard	14.976	374.263	16.288	3.388	1.006	26.413	3.438	1.640
75.00%	Easy	5.139	12.304	5.263	0.867	1.011	8.692	0.963	1.659
	Hard	10.500	391.575	13.475	3.263	1.006	23.200	3.275	1.633

Figure 4.8: Summary of results for instances with one binary and one nonbinary tree. We list the average hybridization number found (Opt), the average running time in seconds (Time) and where applicable the average approximation ratio (Ratio) for the three algorithms.

T_1 nonbinary T_2 nonbinary		TerminusEst		NBCK 4-appx			NBCK 7-appx		
		Opt	Time	Opt	Time	Ratio	Opt	Time	Ratio
25.00%	Easy	7.168	12.971	7.240	43.967	1.032	16.338	2.463	2.343
	Hard	16.148	279.100	/	/	/	35.638	7.000	2.193
50.00%	Easy	5.933	11.150	5.900	41.325	1.030	13.721	2.004	2.405
	Hard	13.216	379.238	/	/	/	32.363	7.200	2.331
75.00%	Easy	3.654	1.121	3.729	4.208	1.015	9.075	1.483	2.590
	Hard	8.672	183.150	/	/	/	21.950	5.800	2.294

Figure 4.9: Summary of results for instances with two nonbinary trees. The layout of the table is the same as that of table 4.8.

Chapter 5

...but not to instances with three trees

In the previous three chapters we have focused on one of the most well-studied phylogenetic network problems to date, the MH problem. We were given two rooted phylogenetic trees, T_1 and T_2, on the same set of taxa X, and the goal was to construct a phylogenetic network that contains an image of each of the input trees, while minimizing the hybridization number k of the network.

The holy grail for this problem is to develop algorithms that can cope with many input trees and nonbinary input trees [63] (and to take different causes of incongruence into account, see e.g. [89]). However, thus far most algorithmic research has focused on the simplest possible case: $|\mathcal{T}| = 2$ and both input trees are binary. As we have seen, the comparative tractability of the problem, both in theory and practice, stems from the essentially one-to-one relationship between solutions to the two-tree problem and the MAAF problem.

For $|\mathcal{T}| > 2$ the situation becomes more complex, however, even when restricted to binary trees and $|\mathcal{T}| = 3$. The MAAF abstraction weakens significantly and cannot (obviously) be used to generate optimal solutions to the hybridization number problem. Without the MAAF abstraction it seems that we have to explicitly confront the challenge of actually constructing the hybridization network itself. This is a theoretically daunting challenge, since the space of DAGs is huge. The good news is that for $|\mathcal{T}| > 2$ the problem nevertheless remains FPT in k [46, 56]. The bad news is that none of these results satisfactorily address the problem of actually constructing the network. The FPT result in [46] gives a quadratic kernel, but does not describe a (good) algorithm for solving the kernel. The bounded-search FPT result in [56], based on [57], does actually construct the network, but has an astronomical running time. The running time is so large because it brute forces over the space of all possible *generators* i.e. possible "backbone topologies" of the network [57, 58], a space which is not known to be $O(c^k)$, and continues with a tower of guesses, which is not $O(c^k)$, for each such generator. At present, therefore, the only FPT algorithms for the case of three binary trees are either kernelizations, or bounded-search algorithms with an exponential dependency on k with a non-linear exponent. Several exponential-time algorithms do exist, such as [87] and the algorithm discussed in [46], but using them to solve a kernelized hybridization number instance unfortunately does not help for two reasons. Firstly, the size of the best-known kernel (i.e. the number n of leaves of a kernelized instance) is quadratic, and not linear in k. Secondly, no previously-known exponential-time algorithm has an $O(c^n)$ running time. Therefore, the challenge is to determine whether an algorithm with running time $O(c^k \text{poly}(n))$ exists for the case of three binary trees.

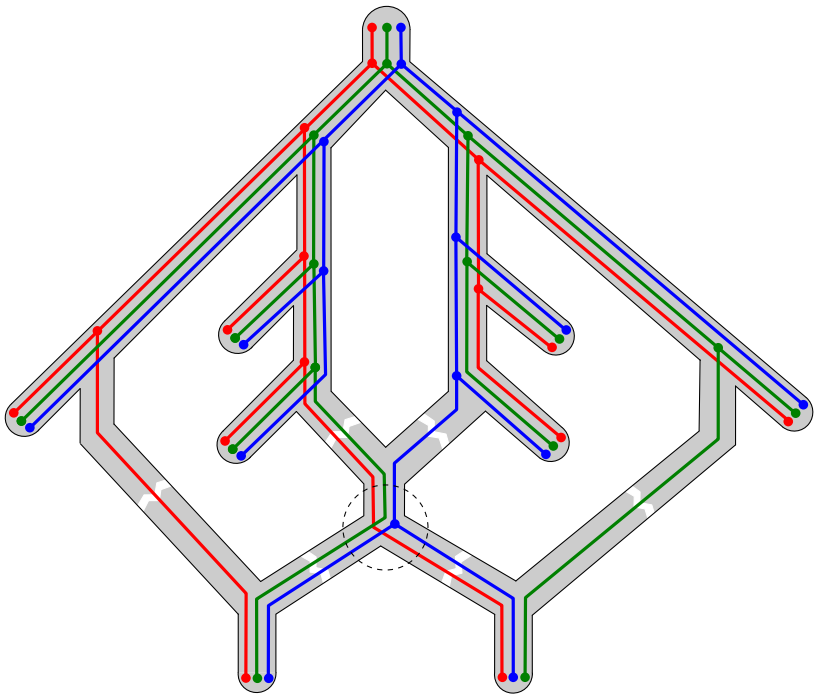

Figure 5.1: A hybridization network for three trees which contains an invisible component (inside the dashed circle), separated from all leaves by hybridization edges (marked with white arrows). It can be shown that any hybridization network for these trees contains an invisible component. However, the single node inside this component can be identified because it corresponds to a node of the blue input tree.

In this chapter we answer this challenge positively. Although the constant c that we find is astronomical - 1609891840 - it represents a significant development in our understanding of the underlying combinatorial structure of the hybridization number problem. We show that, although it is not clear how a MAAF can be pieced together into an optimal solution to the hybridization number problem, it is still possible to identify in $O(c^k \text{poly}(n))$ time a (not necessarily maximum) acyclic agreement forest (AAF) that does have this property. Having found the appropriate acyclic agreement forest, we use deep insights into the structure of optimal hybridization networks to piece the components of the forest together into a network.

The difficulty of this step comes from the fact that, unlike in the two-tree case, it is no longer possible to avoid having nodes in the network that are separated from all leaves by hybridization edges (which are simply the incoming edges to the reticulation vertices). These nodes are therefore not represented in the agreement forest and this was our motivation for calling them "invisible", see Figure 5.1. The main insight helping to overcome this problem is that, in the case $|\mathcal{T}| = 3$, there always exists an optimal hybridization network such that each of its outdegree-2 nodes corresponds to one or more outdegree-2 nodes of the input trees. This enables us to keep the combinatorial explosion in the number of possible network topologies under control.

Note that our algorithm can be viewed as a structural generalisation of existing algorithms for two trees, which also separate the identification of the underlying acyclic agreement forest and the construction of the network into two phases. In the case of two trees the second phase is

polynomial and it is comparatively easy to obtain $O(c^k \text{poly}(n))$ running times for the first phase. In fact, although our overall result at present only holds for the case $|\mathcal{T}| = 3$, the results for the first phase go through without modification for the case $|\mathcal{T}| > 3$. As we demonstrate, the only barrier to extending our result is the fact that, for $|\mathcal{T}| > 3$, the combinatorial insight mentioned in the previous paragraph no longer holds. Indeed, there are two new challenges stemming from this result. Firstly, to adapt and generalize the combinatorial insight so that the wider result can be extended to four or more trees. Secondly, to significantly optimize the constant c in our running time. How close can we get to the competitive $O(3.18^k \text{poly}(n))$ running time achieved in the two-tree case?

The structure of the chapter is as follows. We start by "guessing" the underlying acyclic agreement forest of an optimal hybridization network in $O(c^k \text{poly}(n))$ time in section 5.1. We then define a notion of canonical networks in Section 5.2 and show that we may restrict to them as long as there are at most three trees in the input. Subsequently, Section 5.3 shows how such a canonical network can be reconstructed from an acyclic agreement forest and the input trees in $O(c^k \text{poly}(n))$ time. Finally, we give an example of the algorithm in Section 5.4 and present our conclusions in Section **??**.

5.1 Guessing the AAF

Let $\mathcal{T}$ be a collection of input trees. Without loss of generality we will assume that $\mathcal{T}$ contains no nontrivial common pendant subtrees. In this section, we show how we can guess an AAF from which we can build an optimal hybridization network for $\mathcal{T}$. To make this precise, we define the *deletion forest* of a network N as the forest obtained by deleting all the reticulation edges of N, deleting all resulting connected components that do not contain any taxa, and then taking the partition of the taxa induced by the remaining connected components. By this definition a forest is not a set of trees but a partition of taxa set. However, as we did before, we use these two definitions interchangeably as sometimes it is more useful to see a forest as a collection of trees and sometimes as a collection of taxa sets. Note that for a given network the deletion forest is uniquely defined. We start by proving the following lemma.

Lemma 5.1. *Given a hybridization network N with hybridization number k for a set $\mathcal{T}$ of input trees, the deletion forest of N is an AAF of $\mathcal{T}$ with at most $k+1$ components.*

Proof. We first show that the deletion forest contains at most $k+1$ components. To see this, note that N contains exactly k reticulation nodes. For a reticulation node r, let $X(r)$ be the set of taxa that can be reached from r by directed paths that start at r and which do not intersect with any reticulation apart from r. (Possibly, $X(r) = \emptyset$.) By construction, none of the edges on these directed paths are deleted when the deletion forest is created. Hence, all the taxa in $X(r)$ will be in the same connected component. Similarly, if $X(\rho)$ denotes the set of taxa reachable by directed paths that start at the root and which do not intersect with any reticulations, then the taxa in $X(\rho)$ will also be together in the same connected component. Note that the deletion forest $\mathcal{F}$ of N (seeing it as a partition of the taxa) is the collection containing $X(\rho)$ if $X(\rho) \neq \emptyset$ and $X(r)$ for each reticulation r for which $X(r) \neq \emptyset$. Hence, the deletion forest containst at most $k+1$ components. Moreover, for each $F \in \mathcal{F}$, each input tree must yield the same subtree when restricted to the subset of taxa of F because N displays all the input trees. In addition, for each input tree $T \in \mathcal{T}$, the subtrees $\{T(L(F)) \mid F \in \mathcal{F}\}$ are node-disjoint, again because N displays T. It follows that the deletion forest $\mathcal{F}$ of N is indeed an AAF of the input trees, with at most $k+1$ components. □

As a consequence of Lemma 5.1, we will from now on refer to the deletion forest of a network

as its *deletion AAF*. We can now show how one can guess the deletion AAF of some optimal hybridization network for the input trees.

Lemma 5.2. *Let k be the hybridization number of the set $\mathcal{T}$ of input trees. Then, in time $O(c^k poly(n))$, we can guess an AAF that is the deletion AAF of some hybridization network for $\mathcal{T}$ with hybridization number k.*

Proof. Consider an arbitrary input tree $T \in \mathcal{T}$. Observe that an AAF of $\mathcal{T}$ with $k' + 1$ components can be obtained from T by deleting exactly k' edges and taking the partition of the taxa induced by the resulting connected components. By Lemma 5.1, the deletion AAF of an optimal hybridization network for $\mathcal{T}$ has at most $k + 1$ components. The goal therefore is to locate the at most k edges that need to be deleted from T in order to obtain the deletion AAF of some optimal hybridization network for $\mathcal{T}$.

Let N be a hybridization network for $\mathcal{T}$ with hybridization number k and consider its underlying generator, which is a k-reticulation generator and hence has at most k node sides and at most $4k - 1$ edge sides [46]. (For a reminder of these definitions see page 19.) It follows that there are at most $5k - 1$ common chains of $\mathcal{T}$, because any two taxa on the same edge side of N are in the same common chain, as argued in [46]. The set of common chains is unambiguously defined by the set of input trees, can be computed in polynomial time, and no two chains can share a taxon. Moreover, in [46] it is proven that if two or more taxa of a common chain are on a single edge side of the underlying generator, the entire chain can safely be moved onto that edge side. That is, the new network still displays the input trees and has hybridization number no higher than the old network.

This means that we can assume the existence of an optimal network N' such that for each common chain there are exactly two possibilities: (1) the chain is on a single edge side of the underlying generator, or (2) each taxon of the chain is on a different side of the underlying generator. For each chain we can guess which of the two cases holds, using at most 2^{5k-1} guesses for the entire set of chains. Since, as mentioned before, any two taxa that are on the same side are together in a common chain, it follows that each side of (the underlying generator of) N' contains either a complete case (1) chain or a single taxon (which is either in a case (2) chain or a singleton-chain) or no taxa at all.

Now, assume that we have identified the correct set of guesses describing the behaviour of the common chains in N'. It remains to show that we can identify the correct set of edges to delete in T to obtain the deletion AAF corresponding to N'. Observe that for each case (1) chain it is not necessary to delete any of the internal edges of the chain in T. This is because we have correctly identified that the entire chain sits on a single edge side of the generator and thus that it sits inside a single component of the deletion AAF. For all other edges in T we can simply guess whether to delete it or not. Fortunately, there are not too many of these edges. Specifically, recall that each side contains either a case (1) chain or a single taxon, and that the number of sides is at most $5k - 1$. Hence, if we collapse each case (1) chain C into a new taxon x_C, which is permitted because we will never cut its internal edges, there are in total at most $5k - 1$ taxa left. A binary tree with $5k - 1$ taxa has $10k - 4$ edges. By brute forcing on these edges (i.e. guessing whether to delete it or not) we observe that, in total, the deletion AAF of N' can be located in time at most $O(2^{5k} \cdot 2^{10k} \cdot \text{poly}(n))$. □

Notice that the proof of Lemma 5.2 tells us how to find a correct AAF, but it doesn't tell us how to construct an optimal hybridization network from this AAF.

5.2 Canonical networks

In this section we give the only lemma that is three-tree specific. We describe a transformation from a hybridization network to a structure, called a canonical network with embedded trees, that has some desirable properties. We prove that the transformation preserves the hybridization number in the three-tree case, and hence in the three-tree case we are allowed to concentrate on canonical networks only. For ease of notation, we will sometimes identify a directed graph with its edge set.

A *Canonical Network with Embedded Trees (CNET)* for a set $\mathcal{T}$ of phylogenetic trees over a label set X is a pair $\mathcal{H} = (H, \mathcal{E})$ with the following properties:

1. H is a DAG. We call its sources, i.e. indegree-0 nodes, *roots* and its sinks, i.e. outdegree-0 nodes, *leaves*.
2. Every root of H has one child.
3. The leaves of H are labelled bijectively with the leaf labels in X.
4. $\mathcal{E} = \{H(T) \subseteq H \mid T \in \mathcal{T}\}$ such that for all $T \in \mathcal{T}$, $H(T)$ is an *image* of T, that is, T can be obtained from $H(T)$ by suppressing degree-2 nodes.
5. Every tree image $H(T) \in \mathcal{E}$ contains an edge incident to a root of H.
6. $H = \bigcup_{T \in \mathcal{T}} H(T)$, that is, every edge of H belongs to at least one tree image in $\mathcal{E}$.
7. Every non-leaf non-root node of H has exactly two children.
8. For every non-leaf non-root node, there exists a tree image $H(T) \in \mathcal{E}$ that contains both its child edges.

We represent the tree images in $\mathcal{E}$ by associating a unique colour with each tree $T \in \mathcal{T}$ and colouring every edge in $H(T)$ with this colour. We call the colour associated with tree T *colour* T. We use $C(e)$ to denote the colour set of an edge e of H, that is, the set of trees $T \in \mathcal{T}$ whose images $H(T) \in \mathcal{E}$ include e. A CNET for the input trees in Figure 5.8 on page 67 is shown in Figure 5.11 on page 70. A corresponding hybridization network is shown in Figure 5.12 on page 71.

The hybridization network *induced* by a CNET $(H', \mathcal{E})$ is the hybridization network obtained by applying the following transformations to H':

- We replace every node x that is both a reticulation node and a split node with two nodes x_t and x_b, change the bottom endpoints of x's parent edges to x_t, change the top endpoints of x's child edges to x_b, and add an edge from x_t to x_b.
- We replace every reticulation node with more than two parents with a chain of binary reticulation nodes. See Figure 5.2 for an illustration of these first two steps.
- As long as there are at least two roots, we choose two such roots r_1 and r_2, change the top endpoint of r_1's child edge to r_2, and add an edge from r_1 to r_2. This reduces the number of roots by one, so we eventually obtain a network with a single root. See Figure 5.3.
- For every leaf x with more than one parent, we create a new node x', change the bottom endpoint of every parent edge of x to x', and add an edge from x' to x.

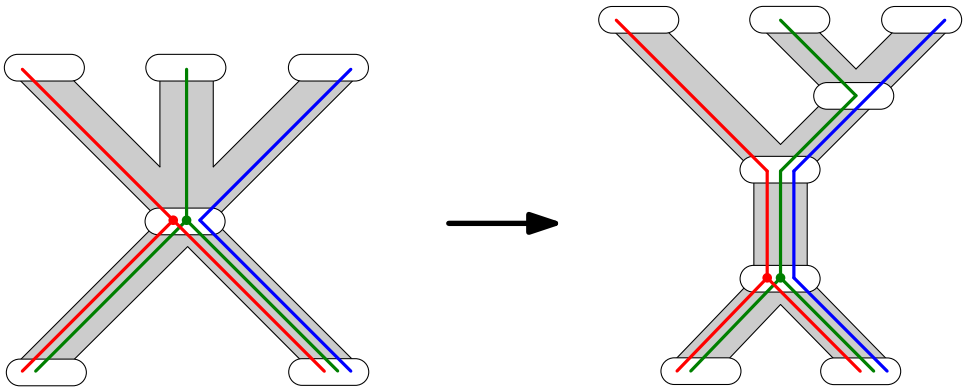

Figure 5.2: The first two steps of transforming a CNET into a hybridization network: expanding nodes that are both reticulation and split nodes and refining reticulations.

The *deletion AAF* of a CNET $(H, \mathcal{E})$ is defined to be the deletion AAF of the hybridization network induced by $(H, \mathcal{E})$. The *hybridization number* of a CNET $(H, \mathcal{E})$ is (as for networks) defined to be the sum $\sum(d^-(v) - 1)$ over all nodes v of H with indegree $d^-(v)$ at least 1 i.e. over all non-roots.

Lemma 5.3. *If $|\mathcal{T}| = 3$, then there exists a hybridization network H with hybridization number k for $\mathcal{T}$ if and only if there exists a CNET $\mathcal{H} = (H', \mathcal{E})$ with hybridization number k for $\mathcal{T}$. Moreover, if such H and $\mathcal{H}$ exist then they have the same deletion AAF.*

Proof. First suppose that there exists a CNET $\mathcal{H} = (H', \mathcal{E})$ with hybridization number k for $\mathcal{T}$. Then, the hybridization network H induced by $\mathcal{H}$ has the same hybridization number as H', and it is easy to see that H displays the trees in $\mathcal{T}$, given that H' displays these trees. It follows directly from the definition of the deletion AAF of a CNET that H and $\mathcal{H}$ have the same deletion AAF.

Now assume we are given a hybridization network H with hybridization number k for $\mathcal{T}$. Since H displays all trees in $\mathcal{T}$, we can choose a tree image $H(T)$, for every tree $T \in \mathcal{T}$, that includes the root of H. Then we set $\mathcal{H} = (H, \mathcal{E})$. $\mathcal{H}$ satisfies conditions 1–5 of a CNET but may violate the remaining three conditions. Next we describe transformations that we apply to $\mathcal{H}$ to ensure it satisfies these remaining conditions without introducing any violations of the conditions $\mathcal{H}$ already satisfies and without increasing the hybridization number of $\mathcal{H}$. Thus, after applying these transformations, we obtain a CNET with hybridization number at most k for $\mathcal{T}$.

Condition 6. Deleting all edges of H that are not contained in $\bigcup_{T \in \mathcal{T}} H(T)$ does not violate conditions 1–5 and establishes condition 6. Since it also does not increase the hybridization number of $\mathcal{H}$, we obtain a network with hybridization number at most k that satisfies conditions 1–6.

Condition 7. As long as there is a *non-splitting* non-root node x, that is, a non-root node with only one child, we contract the edge e between x and its child in H and in every tree image $H(T) \in \mathcal{E}$ that includes e, and merge any parallel edges this may create. Each such contraction reduces the number of non-splitting non-root nodes by one, does not introduce any violations of conditions 1–6, and does not increase the hybridization number of $\mathcal{H}$. Thus, we eventually obtain a network with hybridization number at most k that satisfies conditions 1–7.

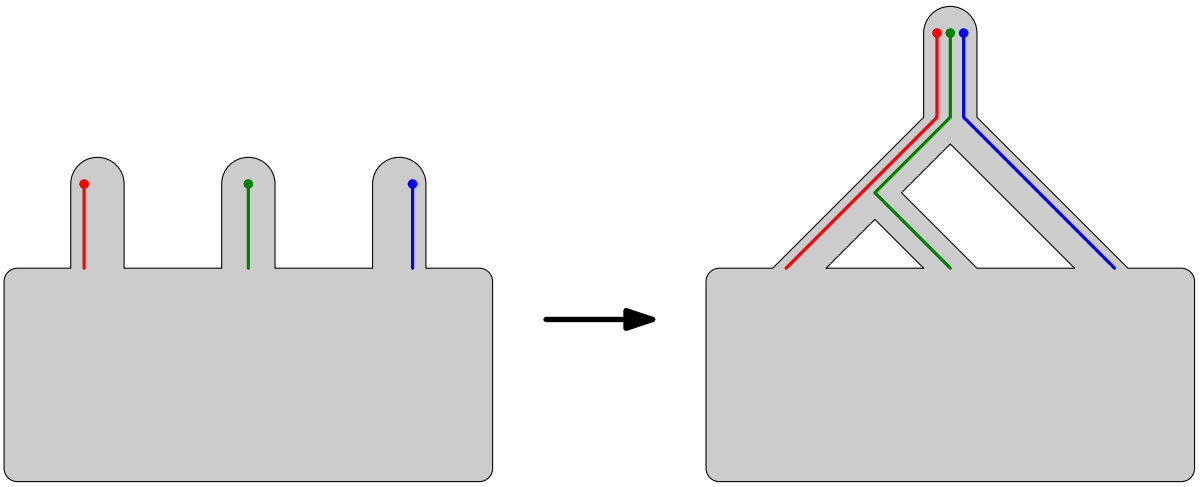

Figure 5.3: The third step of transforming a CNET into a hybridization network: combining multiple roots.

Condition 8. By condition 7, every non-root node of H is a split node. We call it a *true split node* if it also satisfies condition 8, and a *fake split node* otherwise. We also call a true split node a *T-split node* if the tree image $H(T)$ contains both its child edges. The *weight* of a node x is the number of trees $T \in \mathcal{T}$ such that x is a T-split node. The *weight* of a path is the sum of the weights of the nodes on the path. Now we define the *potential* ϕ_x of a fake split node to be one plus the maximum weight of a path from a root to x. All other nodes have potential 0. The potential of the network is $\Phi := \sum_x \phi_x$, where the sum is taken over all nodes of the network. Since every fake split node has a positive potential, a network has potential $\Phi = 0$ if and only if it contains no fake split node, that is, if and only if it satisfies condition 8. Next we describe a transformation that decreases the potential of the network without increasing its hybridization number or introducing any violations of conditions 1–7. Thus, after repeating this transformation as often as possible, we obtain a network with hybridization number at most k which satisfies conditions 1–8, so it is a CNET with hybridization number at most k for $\mathcal{T}$.

While the network contains fake split nodes, there exists such a node x all of whose ancestors are true split nodes or roots. (Simply remove all roots, true split nodes, and leaves from H and choose x to be an in-degree-0 node of the resulting subgraph of H.) Since the colour sets of x's child edges are disjoint and x has two child edges, one of these edges, e, must have exactly one colour: $C(e) = \{T\}$, for some $T \in \mathcal{T}$.[1] Let f be x's parent edge that has colour T. By the choice of x, the top endpoint y of f is a root or a true split node.

If y is a root (see Figure 5.4), we remove T from the colour set of f, create a new root node r, change e's top endpoint to r, remove f and its top endpoint y if the colour set of f is now empty and restore Condition 7. It is easily verified that this does not introduce any violations of conditions 1–6 and does not increase the hybridization number of the network. It also does not increase the potential of any node, and reduces the number of fake split nodes by one. The potential of the network therefore decreases.

If y is a true split node, we distinguish two cases. The first case is that y is a T'-split node, for some $T' \neq T$. This case is illustrated in Figure 5.5. Figure 5.5(a) depicts the subcase that y

[1] This is the only place where we use that $|\mathcal{T}| = 3$. All other arguments are easily seen to generalize to more than 3 trees. Alas, this argument is crucial because our algorithm does not work without condition 8.

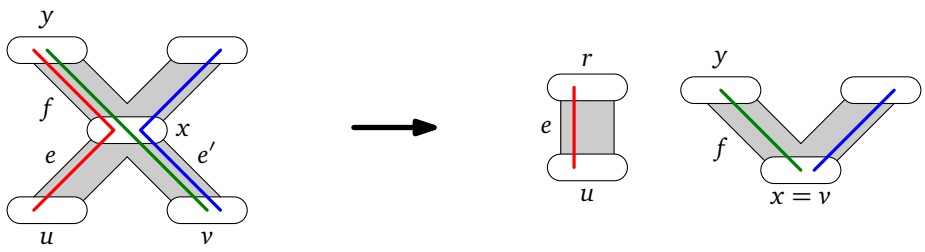

Figure 5.4: Eliminating a fake split node x below a root y.

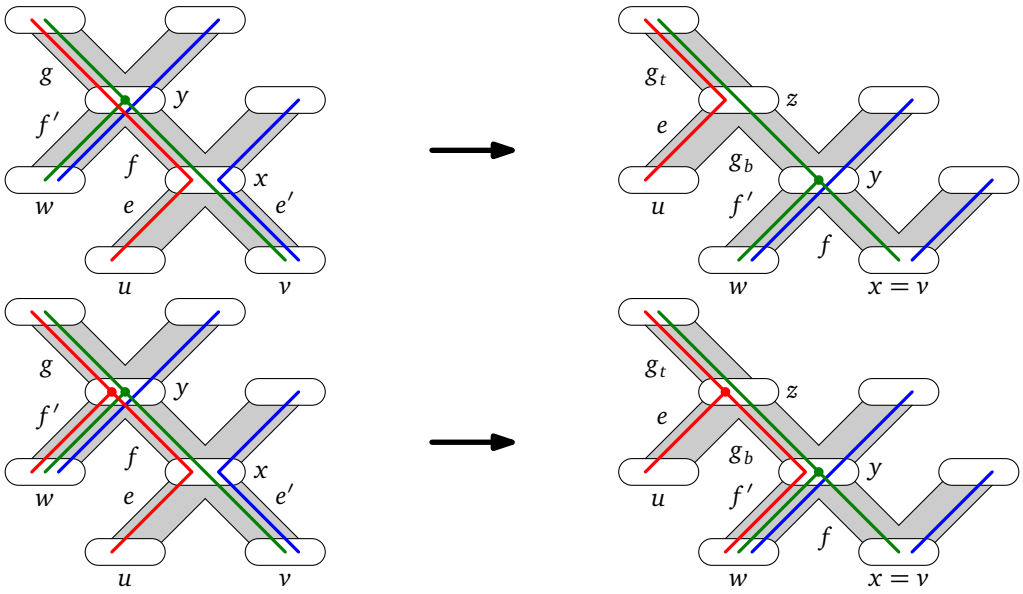

Figure 5.5: Eliminating a fake split node x below a split node y that is a T'-split node with T' not equal to the colour T (red) of the monochromatic edge below x. In the upper half of this figure y is not a T-split node. In the lower half y is a T-split node.

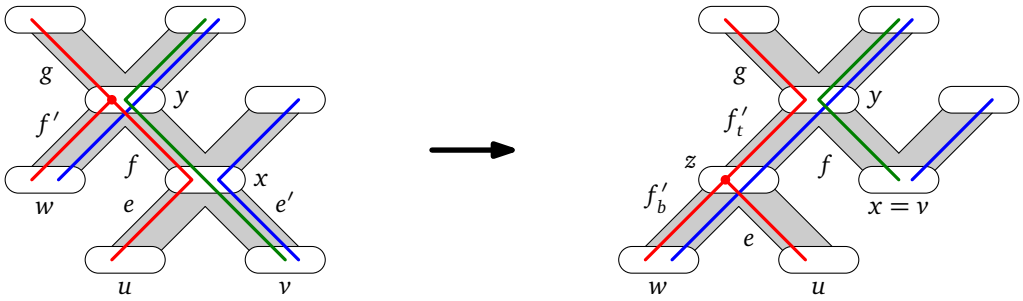

Figure 5.6: Eliminating a fake split node x below a split node y that is not a T'-split node for any T' that is not equal to the colour T (red) of the monochromatic edge below x.

is not also a T-split node while Figure 5.5(b) depicts the subcase that y is also a T-split node. Both cases can be handled in a similar way.

Let g be the parent edge of y whose colour set includes T, and let f' be f's sibling edge. We subdivide g into two edges g_t and g_b, with g_t above g_b, and denote their common endpoint by z. We remove T from the colour set of f, set $C(g_t) := C(g)$ and $C(g_b) := C(g) \setminus \{T\}$ if $T \notin C(f')$ and $C(g_b) := C(g)$ if $T \in C(f')$, change the top endpoint of e to z, remove all edges whose colour sets are now empty (this can only be f and g_b), and finally restore Condition 7. Again, it is easily verified that this does not introduce any violations of conditions 1–6 and does not increase the hybridization number of the network. It also does not increase the potential of any node and eliminates x from the network (because x has only one child edge e' after changing e's top endpoint and hence e' is being contracted). The only new node is z. If $T \in C(f')$, then z is a true split node and its contribution to the network's potential is 0. Thus, since x is eliminated from the network, the network's potential decreases. If $T \notin C(f')$, then z is a fake split node. However, its potential ϕ_z is less than ϕ_x because for each path from the root to z in the modified network there is a corresponding path from the root to x in the original network that has bigger weight because it contains true split node y. Thus, once again, the potential of the network decreases.

Finally, if y is not a T'-split node for any $T' \neq T$ (see Figure 5.6), it must be a T-split node. Let, as before, g be the parent edge of y whose colour set includes T, and let f' be f's sibling edge. We subdivide f' into f'_t and f'_b where f'_t is above f'_b and let z be the newly created node. Change the top endpoint of e to z, remove T from the colour set of f, remove all edges whose colour sets are now empty and restore Condition 7. This transformation maintains conditions 1– 6 and does not increase the hybridization number of the network. It eliminates fake split node x from the network (because it has only one child edge e' after changing the top endpoint of e) and makes y a fake split node. However, the potential of y in the modified network is less than the potential of x in the original network. To see this, let P be a path of maximum weight from a root to y. Then the potential of y in the modified network is one plus the weight of P. In the original network, the path P extended by the edge (y, x) is then a path from a root to x, and its weight is one higher than the weight of P because it contains the true split node y. Hence, the potential of x in the original network is at least one higher than the potential of y in the modified network. Since the potential of all other nodes remains the same or decreases, the potential of the network decreases.

Let $\mathcal{H} = (H', \mathcal{E}')$ be the CNET eventually obtained by the above transformations and let H be the original hybridization network. It remains to show that $\mathcal{H}$ and H have the same deletion AAF. Observe that any node and edge of any component of the deletion AAF of H is contained

in any embedding of each of the input trees. Using this observation, it can easily be checked that none of the modifications above alters the deletion AAF. In particular, the only case in the modifications for satisfying Condition 8 that involves an edge that is used by all three embeddings is the case depicted in Figure 5.5(b). Also the modification for this case does not alter the deletion AAF because the AAF component containing node w in the figure is not altered by the modification. □

By Lemma 5.3, it is sufficient if our algorithm can construct a CNET of the three input trees. In the next section, we show how to do this.

5.3 Reconstructing a canonical network

Let $\mathcal{H} = (H, \mathcal{E})$ be a CNET for $\mathcal{T}$, let F be its deletion AAF, and let k be its hybridization number. For each tree $T \in \mathcal{T}$, let $I(T)$ be the set of nodes in T that do not belong to any path between two leaves x and y in the same AAF component (considering the root as a leaf). In a sense, these nodes are "invisible" in F. The *extended AAF* F^* of $\mathcal{T}$ is defined as $F \cup I$, where $I := \bigcup_{T \in \mathcal{T}} I(T)$. We will refer to the elements of F^* as *components*. Let C be a component of F^*. Hence, C is either an AAF component or a vertex in I. If C is an AAF component, then r_C denotes the root of C. If C is a vertex of I, then r_C is equal to vertex C. In either case, we will refer to r_C as the *root* of component C.

By the following lemma, the size of I is at most $3(k-1)$, when $\mathcal{T}$ contains at most three trees.

Lemma 5.4. *For any $T \in \mathcal{T}$, $|I(T)| \leq k - 1$.*

Proof. Let $T \in \mathcal{T}$. Since the root of T is a leaf after omitting directions, we can see T as an unrooted tree. Since F is an AAF of $\mathcal{T}$ and $T \in \mathcal{T}$, we know that F can be obtained by deleting a set E^* of k edges from T and then taking the partition of the leaves induced by the resulting connected components. Let $\mathcal{C}$ be the partition of the vertices of $I(T)$ induced by the connected components of the subgraph of T induced by $I(T)$. For each $C \in \mathcal{C}$, let $\delta(C)$ denote the set of edges with exactly one endpoint in C. Then, by the definition of $I(T)$, at most one of the edges in $\delta(C)$ is not contained in E^*. Hence, from the $|C| + 2$ edges in $\delta(C)$, at least $|C| + 1$ edges are in E^*. Moreover, since T is a tree, at most $|\mathcal{C}| - 1$ edges can be in $\delta(C) \cap \delta(C')$ for two different $C, C' \in \mathcal{C}$. Hence,

$$|E^*| \geq \sum_{C \in \mathcal{C}} (|C| + 1) - (|\mathcal{C}| - 1) = |I(T)| + 1.$$

Since $|E^*| = k$, the lemma follows. □

We will construct $\mathcal{H}$ from F^* and $\mathcal{T}$ with the help of a *guess* of the structure of $\mathcal{H}$. In particular we construct $\mathcal{H}$ by "gluing together" the components of F^*. Our guess concerns how this gluing is to be done. The root r_C of every component $C \in F^*$ has a unique image $H(r_C)$ in H: If $C \in F$, this is true because F is the deletion AAF of $\mathcal{H}$. If $r_C = C \in I(T)$, for some $T \in \mathcal{T}$, r_C is a split node of T and, thus, has a unique image $H(r_C)$ in $H(T)$ which is also a node of H. Our guess for r_C defines the "wiring" of the in-edges of $H(r_C)$; we call it the *wiring guess* for r_C. Since the colour sets of the parent edges of every node in H are disjoint, $H(r_C)$ has between one and three parent edges. The first part of the wiring guess for r_C is the number of parent edges of $H(r_C)$. The second part of the wiring guess for r_C is which of these in-edges is included in which tree image. Finally, observe that the top endpoint x of each parent edge of $H(r_C)$ must once again be a T'-split node, for at least one $T' \in \mathcal{T}$. The third part of the wiring guess for

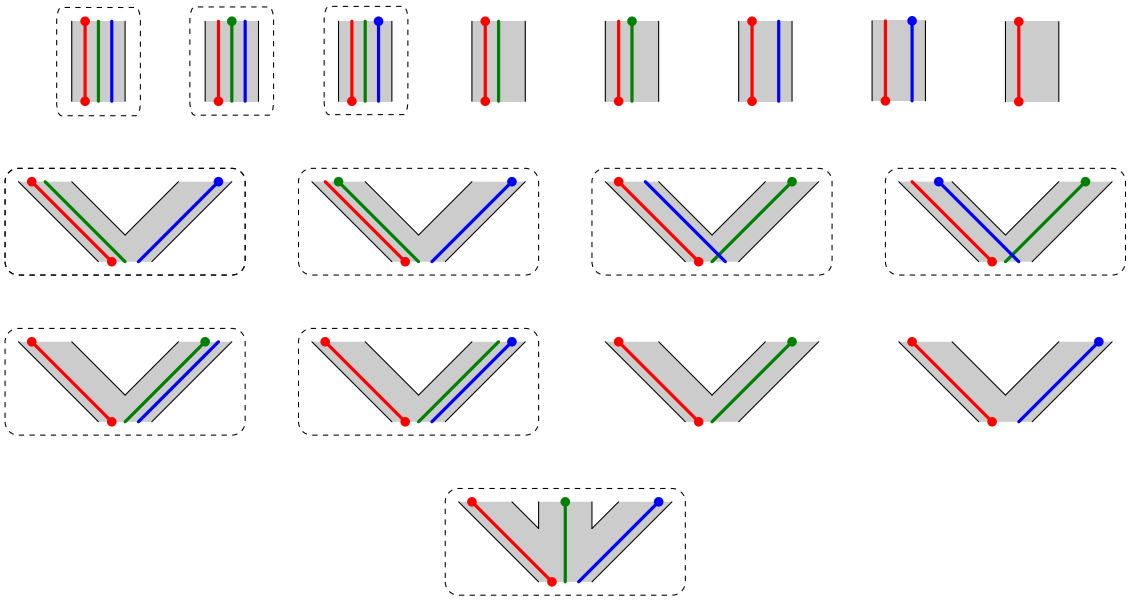

Figure 5.7: The 17 possible wiring guesses for the root r_C of a component C of the extended AAF F^* that is a node in $I(T)$, with T the red tree. The guess of the split node at the top of each edge is indicated by a dot. Valid guesses for a root r_C of an AAF component are indicated by dashed boxes around them.

r_C determines such a tree T' for each parent edge of $H(r_C)$. We will assume without loss of generality that any two nodes $r_C, r_{C'}$ for which $H(r_C)$ and $H(r_{C'})$ have a common parent x both guess the same tree T' as the tree for which x is a T'-split node.

First consider a component root r_C that is a vertex in $I(T)$ for some $T \in \mathcal{T}$. Note that (i) at least one parent edge of $H(r_C)$ must have colour T because $H(r_C)$ is a non-root node of $H(T)$, and (ii) the top endpoint of a parent edge e of $H(r_C)$ can be a T'-split node only for trees T' such that $T' \in C(e)$. This gives the 17 possible wiring guesses for $r(C)$ shown in Figure 5.7.

If r_C is an AAF root, the set of possible wiring guesses is more restricted. Since $H(r_C)$ is contained in every tree image, the only valid wiring guesses are the ones where the union of the colour sets of the parent edges of $H(r_C)$ contains all three colours. This reduces the number of possible wiring guesses for AAF roots to 10 (see again Figure 5.7). Finally note that, when r_C is the root ρ of the trees, there is only a single wiring guess.

Our *guess* $\mathcal{G}$ of the structure of $\mathcal{H}$ consists of the wiring guesses for all roots r_C, $C \in F^*$.

Since we have 17 wiring guesses to choose from for each component in I, and 10 wiring guesses to choose from for each component in F that does not contain the tree root ρ, there are $10^{|F|-1} \cdot 17^{|I|}$ possible guesses $\mathcal{G}$. Since $|F| \leq k+1$ and $|I| \leq 3(k-1)$ by Lemma 5.4, the number of possible guesses $\mathcal{G}$ is bounded by $10^k \cdot 17^{3(k-1)} = 49130^k/4913$.

Our algorithm considers each guess in $\mathcal{G}$ in turn and attempts to construct a CNET $\mathcal{H}$ from $(\mathcal{G}, F^*, \mathcal{T})$ in polynomial time. We call $(\mathcal{G}, F^*, \mathcal{T})$ the CNET's *description*. To establish our algorithm's correctness, we prove that, given the description $(\mathcal{G}, F^*, \mathcal{T})$ of a CNET $\mathcal{H}$ for $\mathcal{T}$, our algorithm succeeds in constructing a CNET $\mathcal{H}'$ for $\mathcal{T}$ with description $(\mathcal{G}, F^*, \mathcal{T})$ that differs from $\mathcal{H}$ only in insignificant details and in particular has the same hybridization number as $\mathcal{H}$. Our proof is divided into two lemmas. The first one shows that two CNETs with the same description are essentially the same. We make this precise below. The second one shows that, given the

description of a CNET, we can construct a CNET with this description in polynomial time. Note that not every description $(\mathcal{G}, F^*, \mathcal{T})$ is necessarily a valid description of any CNET. If there is no CNET with the given description, then our algorithm will determine this polynomial time. The description of this algorithm is given in the proofs of Lemmas 5.5 and 5.6 below. Appendix 5.4 provides an example of the operation of the algorithm.

Given a CNET $\mathcal{H} = (H, \mathcal{E})$ for $\mathcal{T}$, we obtain a DAG $\tilde{H}$ by contracting the image of every component of the deletion AAF F of H into a single node, keeping parallel edges this creates. We call $\tilde{H}$ the *signature* of H. For an example, see Figure 5.10 in the appendix. It is easy to see that the node set of $\tilde{H}$ is $\{H(r_C) \mid C \in F^*\}$ and that $\tilde{H}$ has the same hybridization number as H. Note that two nodes $C \in I(T)$, $C' \in I(T')$ might map to the same node $H(r_C) = H(r'_C)$ of $\tilde{H}$. We label every node x of $\tilde{H}$ with the set of roots r_C, $C \in F^*$, such that $x = H(r_C)$. We call roots that label the same node of $\tilde{H}$ *buddies* (of each other). Note that roots of components of F have no buddies. We call two CNETs $\mathcal{H} = (H, \mathcal{E})$ and $\mathcal{H}' = (H', \mathcal{E}')$ *equivalent* if $\tilde{H} = \tilde{H}'$. In particular, two equivalent CNETs have the same hybridization number.

Before proving that CNETs with the same description are always equivalent, we introduce two useful notions which we will use in the two upcoming proofs.

We will use the following notion of the "attached subtrees" of an AAF component C in a tree T. For each edge (u, v) of T for which u is contained in $T(C)$ but v is not contained in $T(C)$, we say that the subtree of T induced by u, v and all nodes reachable from v is a subtree of T *attached* to C.

In addition, we will use the following notion of the "pendant subtrees" represented by an edge $e = (r_C, r_{C'})$ of the signature $\tilde{H}$. For each $T \in C(e)$, the subtree of T induced by $r_{C'}$, the parent of $r_{C'}$ and all vertices reachable from $r_{C'}$ is said to be a *pendant subtree* of T represented by the edge e of $\tilde{H}$.

Lemma 5.5. *If two CNETs have the same description, then they are equivalent.*[2]

Proof. Given the description $(\mathcal{G}, F^*, \mathcal{T})$ of a CNET $\mathcal{H} = (H, \mathcal{E})$, we show how to uniquely reconstruct $\tilde{H}$ from this description. We define a DAG D whose nodes are the roots of components of F^*. There is an edge from a root r_C to a root $r_{C'}$ if and only if there exists a tree $T \in \mathcal{T}$ such that $r_{C'}$ is an ancestor of r_C in T and there exists no component root $r_{C''}$ that is an ancestor of r_C and a descendant of $r_{C'}$ in T. If this is the case, we call r_C a *direct descendant* of $r_{C'}$ in T. We incrementally construct a topological ordering of D, define U_i to be the first i nodes in this topological ordering, D_i to be the subgraph of D induced by U_i and $V_i := \{H(r_C) \mid r_C \in U_i\}$. $\bar{D}_i$ is the subgraph of D induced by all nodes of D not in U_i. Let the partial signature $\tilde{H}_i$ be the graph obtained from the subgraph of $\tilde{H}$ induced by V_i by, for each parent edge e in $\tilde{H}$ of a node $H(r_C) \in V_i$ whose top endpoint does not belong to V_i, adding a new root with an edge $\tilde{e}$ from this new root to $H(r_C)$ and giving $\tilde{e}$ the same colour set as e (which is determined by the set of guesses). For an example, see Figure 5.10 in the appendix. We will call an edge whose top endpoint is a root a *root edge*.

We assign these new roots colours, where the colour of the top endpoint of edge $\tilde{e}$ is T if the bottom endpoint of $\tilde{e}$ guesses that the top endpoint of e is a T-split node.

We will show that, from $D_i, \tilde{H}_i$ and $\mathcal{G}$, we can determine a unique nonempty set of nodes U' of $\bar{D}_i$, the set U^+ of all buddies of nodes in U' and an extended partial signature $\tilde{H}_j$ corresponding to the graph D_j induced by $U_j := U_i \cup U' \cup U^+$.

Once $D_i = D$, we thus obtain $\tilde{H}_i = \tilde{H}$, that is, we are able to reconstruct the signature $\tilde{H}$ only given the description of $\mathcal{H}$.

We initialize the reconstruction by defining D_0 and $\tilde{H}_0$ to be empty digraphs. Now consider an iteration with input digraph D_i and partial signature $\tilde{H}_i$. For an in-degree-0 node r_C of

[2]The reverse statement does not hold.

$\bar{D}_i$ that belongs to $I(T)$, for some $T \in \mathcal{T}$, observe that both child edges of $H(r_C)$ in $\tilde{H}$ have corresponding root edges in $\tilde{H}_i$ (by the construction of $\tilde{H}_i$). For such a node r_C, we call r_C *free* if the top endpoints of both of these root edges are coloured T. For an in-degree-0 node r_C of $\bar{D}_i$ that is a root of a component of F, we say that r_C is *free* if, for every in-edge e of r_C in D, the corresponding root edge $\tilde{e}$ of $\tilde{H}_i$ has the property that, for every $T \in C(\tilde{e})$, the pendant subtree of T represented by $\tilde{e}$ is attached to the AAF component C in T. All other nodes of $\bar{D}_i$ are non-free. We claim that at least one of the nodes in $\bar{D}_i$ is free and that every free node can determine its buddies in $\tilde{H}$ and can augment the partial signature $\tilde{H}_i$ to $\tilde{H}_j$.

Consider a node x of the signature $\tilde{H}$ that does not belong to $\tilde{H}_i$ and all of whose children do belong to $\tilde{H}_i$. Since $\tilde{H}$ is a DAG, such a node exists. Node x is a T-split node, for some $T \in \mathcal{T}$, and is thus the image $H(r_C)$ of the root r_C of a component in $F \cup I(T)$.

First consider the case that $C \in F$. Observe that, because $\tilde{H}$ is the signature of a CNET $\mathcal{H}$ for $\mathcal{T}$, the subtrees attached to C in all input trees are exactly the pendant subtrees represented by the child edges of $H(r_C)$ in $\tilde{H}$. Since these child edges are root edges of $\tilde{H}_i$, r_C is free. Node r_C can trivially identify its set of buddies because it has no buddies besides itself. We obtain D_j by adding r_C to D_i and obtain $\tilde{H}_j$ from $\tilde{H}_i$ by locating the root edges of $\tilde{H}_i$ that correspond to child edges of $H(r_C)$ in $\tilde{H}$, merging the top endpoints of these root edges to a single node $H(r_C)$, and adding new root edges entering $H(r_C)$ according to r_C's wiring guess. This gives a new graph $D_j \supset D_i$ and the corresponding graph $\tilde{H}_j$.

Now consider the case that $r_C \in I(T)$. In this case, both child edges of r_C in $\tilde{H}$ correspond to root edges of $\tilde{H}_i$. Since these two edges have a common top endpoint in $\tilde{H}$, their top endpoints in $\tilde{H}_i$ must have the same colour T'. This implies that $H(r_C)$ is a T'-split node and is thus the image $H(r_{C'})$ of a node $r_{C'} \in I(T')$. Since the child edges of $H(r_{C'})$ in $\tilde{H}$ belong to $\tilde{H}_i$, both in-neighbours of $r_{C'}$ in D belong to D_i. Thus, $r_{C'}$ is free. Since $H(r_C) = H(r_{C'})$, we may assume that $r_{C'} = r_C$ (because in at least one set of guesses this is the case). We obtain $\tilde{H}_j$ from $\tilde{H}_i$ by locating the two root edges of $\tilde{H}_i$ that correspond to child edges of $H(r_C)$ in $\tilde{H}$, merging the top endpoints of these edges to a single node $H(r_C)$, and adding new root edges entering $H(r_C)$ according to r_C's wiring guess. Now consider the child edges of $H(r_C)$ in $\tilde{H}_j$. For each tree $T' \in \mathcal{T}$ such that these edges are both coloured T', $H(r_C)$ is the image $H(r_{C'})$ of a node in $I(T')$. The nodes of $\bar{D}_i$ that satisfy this condition are the buddies of r_C, and we add them to D_i along with r_C to obtain D_j.

We have shown that every node $H(r_C)$ of $\tilde{H}$ whose children belong to $\tilde{H}_i$ corresponds to a free node r_C in $\bar{D}_i$ and that r_C can determine its set of buddies and can construct the extended partial signature $\tilde{H}_j$ corresponding to the digraph $D_j \supset D_i$ obtained by adding these buddies to D_i. If r_C is an AAF root, it has no buddies besides itself. If $r_C \in I(T)$, for some $T \in \mathcal{T}$, observe that none of its buddies is free because it belongs to a tree $T' \neq T$ but the top endpoints of the child edges of $H(r_C)$ have colour T in $\tilde{H}_i$. Thus, to prove that *every* free node can determine its set of buddies and can construct $\tilde{H}_j$ from $\tilde{H}_i$, it suffices to show that there is no free node r_C such that $H(r_C)$ has an in-neighbour not in $\tilde{H}_i$.

Assume there exists such a node r_C. Then r_C has in-degree 0 in $\bar{D}_i$ because otherwise it is not free. First assume r_C is the root of a component in F. Let $H(r_{C'})$ be an in-neighbour of $H(r_C)$ in $\tilde{H}$ that does not belong to $\tilde{H}_i$, and let T be an arbitrary colour T in the colour set of the edge e between $H(r_C)$ and $H(r_{C'})$ in $\tilde{H}$. Since r_C has in-degree 0 in $\bar{D}_i$, the pendant subtree of T represented by e is also represented by some root edge f of $\tilde{H}_i$. Thus, $\tilde{H}$ contains a unique path from e to f and every node on this path is a T'-split node, for some $T' \neq T$. This implies in particular that the top endpoint of edge f is a T'-split node, $T, T' \in C(f)$, and the pendant subtree of T' represented by f is not a subtree attached to C in T'. Thus, since f represents the same in-edge of r_C in D as e, r_C is not free, a contradiction.

If $r_C \in I(T)$, for some tree $T \in \mathcal{T}$, we choose an in-neighbour $H(r_{C'})$ and a root edge f of $\tilde{H}_i$ as in the case when $r_c \in F$. Since the top endpoint of edge f is not a T-split node, its colour in $\tilde{H}_i$ must be $T' \neq T$. Thus, since f is the root edge of $\tilde{H}_i$ representing one of the in-edges of r_C in D, r_C is not free, again a contradiction. □

By Lemma 5.5, it suffices to provide a polynomial-time algorithm that decides whether there exists a CNET with description $(\mathcal{G}, F^*, \mathcal{T})$ and, if so, construct any such CNET. Our next lemma states that such an algorithm exists.

Lemma 5.6. *Given a description $(\mathcal{G}, F^*, \mathcal{T})$, it takes polynomial time to decide whether there exists a CNET with this description and, if so, to construct such a CNET.*

Proof. The proof of Lemma 5.5 already specifies an algorithm for constructing the signature $\tilde{H}$ of a CNET $\mathcal{H} = (H, \mathcal{E})$ with description $(\mathcal{G}, F^*, \mathcal{T})$, if such a CNET exists, and this algorithm is clearly implementable in polynomial time. This algorithm provides a first test that can be used to reject invalid descriptions: If the algorithm reaches an iteration where $\bar{D}_i$ is non-empty but none of its nodes is free, then the description is invalid because the proof of Lemma 5.5 shows that, if $(\mathcal{G}, F^*, \mathcal{T})$ is the description of a CNET, then there exists a free node in each iteration. Thus, the algorithm aborts and rejects the description. If the algorithm does not reject the description, its output is a signature $\tilde{H}$ that respects all wiring guesses in $\mathcal{G}$. However, this signature may not correspond to a network H that displays all input trees. Next we present a polynomial-time algorithm for constructing $\mathcal{H} = (H, \mathcal{E})$ from $\tilde{H}$, F^*, and $\mathcal{T}$ or determining that no CNET $\mathcal{H} = (H, \mathcal{E})$ with description $(\mathcal{G}, F^*, \mathcal{T})$ exists.

The nodes of $\tilde{H}$ are of two types: images of AAF roots and images of nodes in I. To obtain a CNET $\mathcal{H} = (H, \mathcal{E})$ with description $(\mathcal{G}, F^*, \mathcal{T})$ from $\tilde{H}$, we let H initally be equal to $\tilde{H}$ and will replace each AAF root image with the AAF component it represents. We process these AAF root images bottom-up, that is, in reverse topological order.[3] Consider such an image $H(r_C)$ of the root r_C of an AAF component C, and let E be the set of child edges of $H(r_C)$ in $\tilde{H}$. We remove the edges in E from $\tilde{H}$ and attach C below r_C, setting $C(e) = \mathcal{T}$ for every edge e of C. Our goal now is to reattach the edges in E to edges of C so that, for all $T \in \mathcal{T}$, the descendant edges of $H(r_C)$ with colour T form an image of the pendant subtree of T with root r_C. For each tree T, observe that the edges in E coloured T represent the subtrees attached to C in T. We need to attach these edges to C in H so that each edge branches off the same edge of C as in T and edges that branch off the same edge of C in T do so in the same order as in T.

First we test every edge $e \in E$ whether e branches off the same edge of C in every tree $T \in C(e)$. If this is the case for all edges $e \in E$, then we can partition E into subsets E_f, one per edge f of C, such that all edges in E_f branch off edge f. If there is an edge e that branches off some edge f in a tree $T \in C(e)$ and off a different edge f' in a tree $T' \in C(e)$, it is impossible to attach this edge to C in a way that satisfies both constraints. Since the edges of T and T' represented by e are determined by $\tilde{H}$, which in turn is uniquely defined by $(\mathcal{G}, F^*, \mathcal{T})$, there is thus no network $\mathcal{H}$ with description $(\mathcal{G}, F^*, \mathcal{T})$, so the algorithm aborts and reports that there is no such network.

Given the partition of E into subsets E_f such that the edges in E_f branch off edge f, we need to attach the edges in each such set E_f in an ordering consistent with the input trees. Let $E_{f,T}$ be the subset of edges in E_f that have colour T. Tree T determines the ordering in which these edges need to be attached to f. We define a DAG D_f whose nodes represent the edges in E_f, and which has an edge (g, g') precisely if there is a tree T such that $g, g' \in E_{f,T}$ and g branches off f in T above g'. There exists an ordering in which to attach the edges of E_f to f so that the ordering constraints imposed by all trees in $\mathcal{T}$ are satisfied if and only if D_f is acyclic.

[3] This is not really essential, but it simplifies the description of the algorithm.

Moreover, if D_f is indeed acyclic, then a topological ordering of D_f provides a valid ordering in which the edges can be attached. Thus, we can test whether such an ordering exists and, if so, compute such an ordering in time $O(|E_f|)$ per edge f. If no such ordering exists, the algorithm once again aborts and reports that there is no CNET with description $(\mathcal{G}, F^*, \mathcal{T})$. If we can find a correct ordering of the edges attached to each edge f of C, then the replacement of r_C with C in this manner results in a network where all descendant edges of $H(r_C)$ with colour T form an image of the pendant subtree of T with root r_C, for all $T \in \mathcal{T}$. Thus, after replacing each AAF root r_C with its corresponding component C in this fashion, we obtain a CNET $\mathcal{H} = (H, \mathcal{E})$ with description $(\mathcal{G}, F^*, \mathcal{T})$. □

5.4 An Example of constructing a network from its description

This appendix provides an example of the construction described in the proofs of Lemmas 5.5 and 5.6. Consider the description $(\mathcal{G}, F^*, \mathcal{T})$ in Figure 5.8. The CNET in Figure 5.11 has this description. We first show how to construct its signature using the construction in the proof of Lemma 5.5. This signature is shown in Figure 5.10. Then we discuss how to construct the CNET in Figure 5.11 from this signature. The hybridization network induced by this CNET is shown in Figure 5.12.

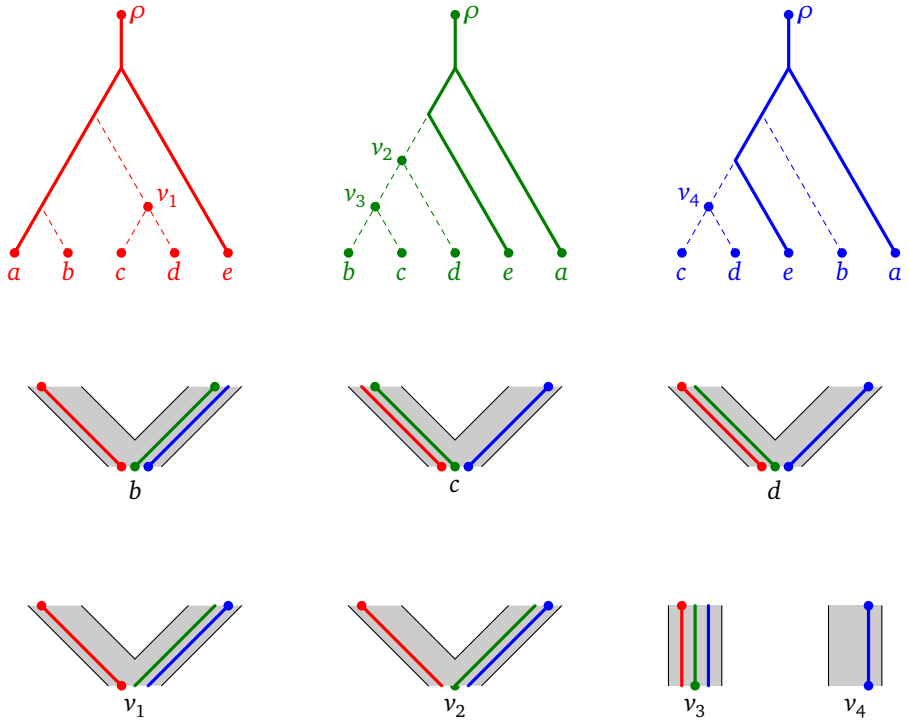

Figure 5.8: The input to the network reconstruction, including the AAF F (shown in bold in the three input trees), the set of "invisible nodes" $I = \{v_1, \ldots, v_4\}$, and the wiring guesses for the resulting set of components of F^*. There is no guess for the component $\{a, e, \rho\}$ because it includes the root ρ of the three trees.

Lemma 5.5: Constructing the signature. The DAG D used in the construction of the signature $\tilde{H}$ of any CNET with the description in Figure 5.8 is shown in Figure 5.9. This DAG represents the adjacency of components of F^* in the three input trees in $\mathcal{T}$.

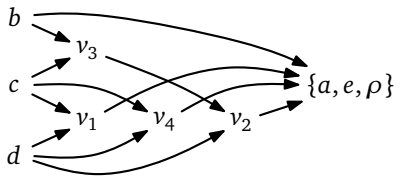

Figure 5.9: The DAG D used in the reconstruction of the signature $\tilde{H}$ in Figure 5.10 from the description in Figure 5.8.

Before the first iteration, we have $\bar{D}_0 = D$. The only nodes with in-degree 0 in $\bar{D}_0$ are b, c, and d. They are all free because they have no in-edges at all. Assume we pick b as the first node to add to $\tilde{H}$. We create the node $H(b)$ and add parent edges according to b's wiring guess to obtain the partial network $\tilde{H}_1$ in Figure 5.10. Since b is the root of an AAF component, it has no buddies, so D_1 has the node set $\{b\}$.

$\bar{D}_1$ has two nodes of in-degree 0, namely c and d. Both are again free. Assume we choose c as the next node to add to D_1 to obtain D_2. Since c is again the root of an AAF component, it has no buddies, so the node set of D_2 is $\{b, c\}$. We construct the graph $\tilde{H}_2$ in Figure 5.11 from $\tilde{H}_1$ by creating a node $H(c)$ and adding parent edges of this node according to c's wiring guess.

$\bar{D}_2$ has two nodes of in-degree 0, namely d and v_3. Both nodes are free: d is free because it is the root of an AAF component; v_3 is free because $\tilde{H}_2$ has two root edges above $H(b)$ and $H(c)$, which are children of v_3 in the green tree, and the top endpoints of both edges are coloured green by the wiring guesses for b and c. Let us assume we choose v_3 as the next node to add to D_2 to obtain D_3. We create the node $H(v_3)$ by identifying the top endpoints of the two green parent edges of $H(b)$ and $H(c)$ and add a parent edge above $H(v_3)$ according to v_3's wiring guess. This gives the graph $\tilde{H}_3$ shown in Figure 5.10. Since the only tree common to the colour sets of the parent edges of $H(b)$ and $H(c)$ is the green one, v_3 has no buddies. Thus, D_3 has node set $\{b, c, v_3\}$.

$\bar{D}_3$ has d as its only in-degree-0 node, and d is free. We obtain D_4 by adding d to D_3. Node d has no buddies because it is the root of an AAF component. To obtain $\tilde{H}_4$ from $\tilde{H}_3$, we create the node $H(d)$ and add parent edges according to d's wiring guess.

$\bar{D}_4$ has three nodes of in-degree 0, namely v_1, v_2, and v_4. The two child edges of v_1 are represented by red parent edges of $H(v_3)$ and $H(d)$ in $\tilde{H}_4$. According to the wiring guesses for v_3 and d, their top endpoints are red. Since v_1 belongs to the red tree, v_1 is free. The same two edges also represent the child edges of v_2. However, v_2 is green and, thus, is not free. Finally, the two child edges of v_4 are represented by the blue parent edges of $H(c)$ and $H(d)$, both of which have blue top endpoints according to the wiring guesses for c and d. Thus, v_4 is also free. Assume we choose v_4 as the next node to add to D_4 to obtain D_5. We create the node $H(v_4)$ in $\tilde{H}_5$ by identifying the top endpoints of the blue parent edges of $H(c)$ and $H(d)$ and then create a parent edge of $H(v_4)$ according to v_4's wiring guess. This produces the graph $\tilde{H}_5$ in Figure 5.10. Since the colour sets of the two child edges of $H(v_4)$ have only the blue tree in common, v_4 has no buddies and D_5 has node set $\{b, c, d, v_3, v_4\}$.

Nodes v_1 and v_2 are the only nodes of in-degree 0 in $\bar{D}_5$. Just as in $\bar{D}_4$, v_1 is free and v_2 is not. Thus, we choose v_1 as the node to add to D_5 to obtain D_6. The two child edges of v_1 are represented by the red parent edges of $H(v_3)$ and $H(d)$ in $\tilde{H}_5$. We identify their top endpoints to create the node $H(v_1)$ and add parent edges according to the wiring guess for v_1.

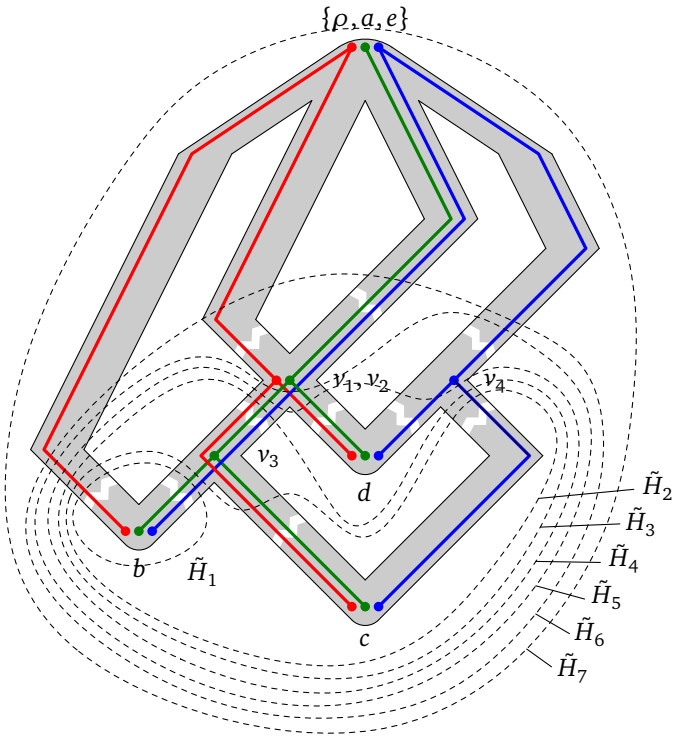

Figure 5.10: The signature $\tilde{H}$ of any CNET with the description $(\mathcal{G}, F^*, \mathcal{T})$ shown in Figure 5.8. The partial signatures $\tilde{H}_1, \tilde{H}_2, \ldots, \tilde{H}_7 = \tilde{H}$ constructed incrementally are indicated by dashed lines.

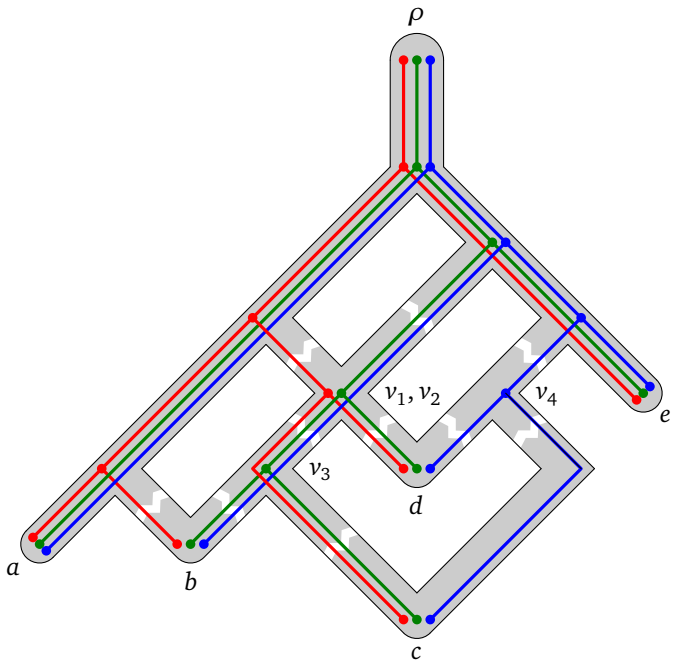

Figure 5.11: The CNET obtained from the signature in Figure 5.10 by expanding the components of F.

This produces the graph $\tilde{H}_6$ in Figure 5.10. Now observe that the two child edges of $H(v_1)$ in $\tilde{H}_6$ are also coloured green. Thus, we identify the node that is the parent of the two edges of the green tree represented by these child edges, which is node v_2. Node v_2 becomes a buddy of v_1 and is added to D_5 along with v_1 to obtain D_6. Thus, D_6 has node set $\{b, c, d, v_1, v_2, v_3, v_4\}$. Note that making v_1 and v_2 buddies does not create any conflicts because they both have the same wiring guess in $\mathcal{G}$.

Finally, the only node remaining in $\bar{D}_6$ is $\{a, e, \rho\}$. The root edges of $\tilde{H}_6$ represent exactly the set of pendant edges of the AAF component $\{a, e, \rho\}$ in the three input trees, so $\{a, e, \rho\}$ is free in $\bar{D}_6$. We create a node $H(\{a, e, \rho\})$ in $\tilde{H}_7$ by identifying the top endpoints of all root edges of $\tilde{H}_6$. Since there is no wiring guess for $\{a, e, \rho\}$ in $\mathcal{G}$, we do not add any parent edges to $H(\{a, e, \rho\})$, and $\tilde{H}_7 = \tilde{H}$ is the final signature.

It is easily verified that we would have obtained the exact same signature had we chosen different nodes to add to D_i in iterations where $\bar{D}_i$ contained more than one free node.

Lemma 5.6: Expanding AAF components. In our example, the only non-trivial AAF component to be expanded is the component $\{a, e, \rho\}$. This component has two non-root edges. Let f_a be the parent edge of a, and let f_e be the parent edge of e in this component. In $\tilde{H}$, the node $H(\{a, e, \rho\})$ has four child edges: a red parent edge e_1 of $H(b)$, a red parent edge e_2 of $H(v_1) = H(v_2)$, a green-blue parent edge e_3 of $H(v_1) = H(v_2)$, and a blue parent edge e_4 of $H(v_4)$. Edges e_1 and e_2 attach to f_a in the red tree and do not represent any edges in any other trees, so we add them to E_{f_a}. Edge e_3 attaches to f_e in the green and the blue tree, so there is no conflict and we add it to E_{f_e}. Edge e_4 attaches to f_e in the blue tree and does not represent any edge in any other tree, so we add it to E_{f_e}.

The DAG D_{f_a} has two nodes representing edges e_1 and e_2 with an edge from e_2 to e_1 because

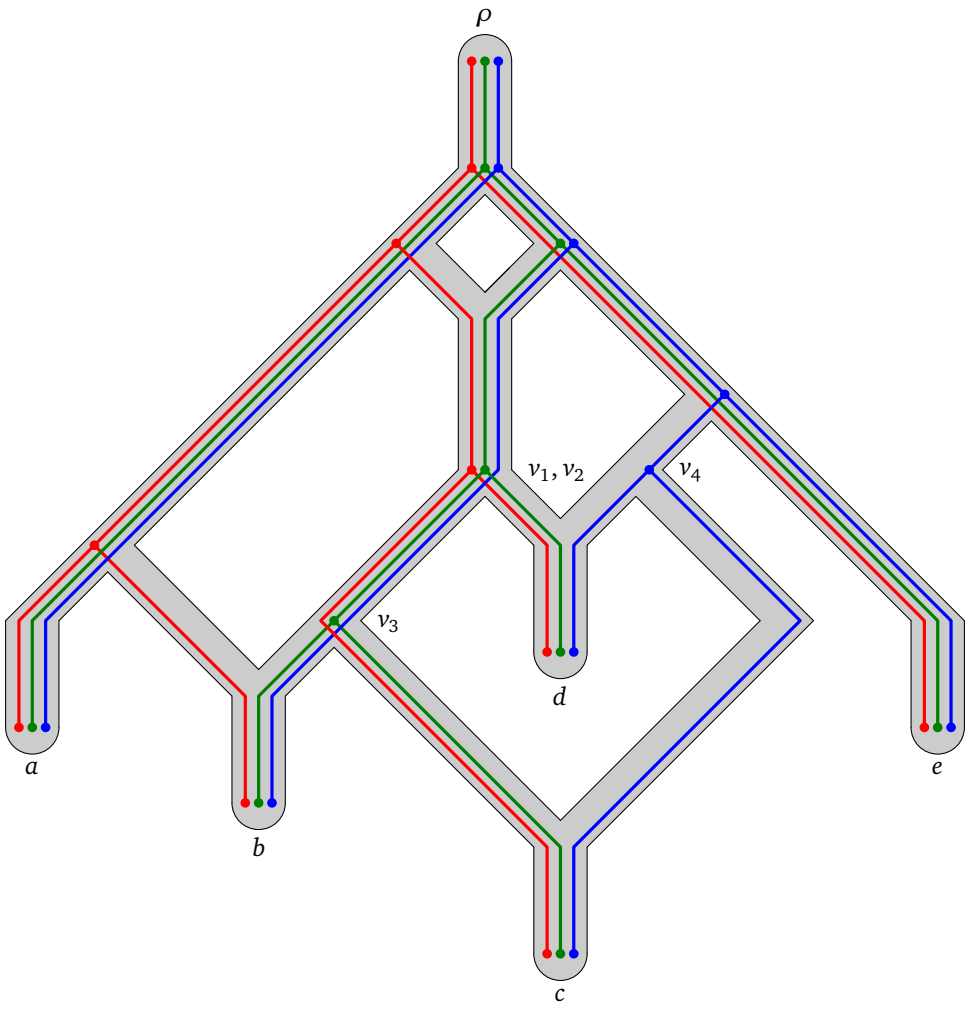

Figure 5.12: The hybridization network induced by the CNET in Figure 5.11, obtained by separating reticulations, split nodes, and leaves.

e_2 attaches to f_a above e_1 in the red tree. A topological ordering of D_{f_a} places these edges in the order $\langle e_2, e_1 \rangle$, and this is the order in which we attach e_2 and e_1 to f_a.

The DAG D_{f_e} has two nodes representing edges e_3 and e_4 with an edge from e_3 to e_4 because e_3 attaches to f_e above e_4 in the blue tree. The green tree does not impose any conflicting ordering constraints because only edge e_3 belongs to this tree. A topological ordering of D_{f_e} places e_3 and e_4 in the order $\langle e_3, e_4 \rangle$, and this is the order in which we attach e_3 and e_4 to f_e. The result is the CNET shown in Figure 5.11. Finally, the hybridization network induced by this CNET is shown in Figure 5.12.

It seemed natural enough to try to reconstruct a hybridization network from the corresponding AAF and to guess the wiring structure that determines how the AAF components need to be glued together to achieve this. In the case of two trees, no guessing is required: the AAF carries the necessary information. For more than two trees, there are only a constant number of choices for the wiring of the root of each component, so with $O(k)$ components, one obtains an $O(c^k \cdot \text{poly}(n))$ time algorithm. Unfortunately, guessing the wiring structure of AAF components turned out not to be enough, even for three trees, because there are examples of three input trees such that every optimal network displaying these trees contains an *invisible component*: a group of nodes that are isolated from all taxa once all hybridization edges are deleted, see Figure 5.1 in the introduction. We call these components invisible because they are not represented in any form in the AAF.

Guessing the number and structure of these invisible components seems extremely challenging. In this chapter, we showed that one can get away without having to guess these components in the case of three trees because, for three trees, these components are not invisible after all: They may not be represented in the AAF, but they are present as nodes in the three input trees, at least as long as we consider only canonical networks (and we have shown that we may do this without loss of generality). This is the key to our $O(c^k \cdot \text{poly}(n))$-time algorithm for three trees. Unfortunately, it appears that we simply scraped by. While the framework of our algorithm extends to more than three trees, it seems that already for four trees, there are input instances where the optimal network includes truly invisible nodes: nodes whose only purpose is to change the way in which edges of the tree images are braided together along network edges, see Figure 5.13. Thus, the main open problem is to discover structural properties that, while unlikely to eliminate the need to guess the existence of these braiding structures in the network altogether, at least limit the number of possible guesses to be explored.

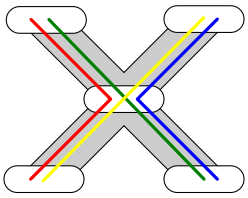

Figure 5.13: A "truly invisible" node of a network with four trees embedded in it. None of the four trees branches in this network node.

Another interesting question that arises from our work is whether guessing the wiring of the extended AAF components as part of the reconstruction is necessary at all, at least in the case of three trees. Since all these components are visible in the input trees, it could be possible that one can construct the entire network directly from the AAF, that is, as in the case of two trees, the hard core of the problem is finding the right AAF. However, we conjecture that this is not the case: i.e., that even given the deletion AAF of an optimal hybridization network for the three

input trees, it remains NP-hard to find the network.

Chapter 6

...and certainly not to the Compatibility problem on unrooted trees.

In this chapter we take a bit of a turn and consider a different phylogenetic problem. COMPATIBILITY OF UNROOTED TREES is a rather well studied problem. It asks to determine whether for a set of k input trees there exists a larger tree (called a supertree) that contains the topologies of all k input trees. When any such supertree exists we call the instance compatible and otherwise incompatible.

The computational complexity landscape of the compatibility problem is uneven. In the case that all the partial trees are rooted (i.e. in which the flow of evolution is assumed to be away from a designated root, towards the taxa, like the trees we have considered so far) the problem is polynomial-time solvable, using the algorithm of Aho [1]. However, in the case of unrooted trees the problem is NP-hard, even when all the partial trees have at most 4 taxa [78]. Nevertheless, due to the fact that many tree-building algorithms actually construct *un*rooted trees, and because of the risk of distorting the underlying phylogenetic signal through a poor choice of root location, it remains attractive to try and solve this NP-hard variant of the problem directly.

In this chapter we approach the unrooted compatibility problem from a graph-theoretical angle. There is a recent trend in this direction, which to a large extent can be traced back to a seminal paper of Bryant and Lagergren [12]. They observed that there is a relationship between the compatibility question and the *treewidth* of an auxiliary graph known as the *display graph*. The display graph is obtained by identifying the taxa of the input trees, and treewidth is an intensely well-studied parameter in the algorithmic graph theory literature (see e.g. [7]). Low (or bounded) treewidth often facilitates algorithmic tractability, and given that it is a measure of "distance from being a tree", it is tempting to try and exploit this tractability in questions pertaining to phylogenetic compatiblity and incongruence. Bryant and Largergren observed that for k unrooted trees to be compatible, it is necessary (but not sufficient) that the display graph has treewidth at most k. The upper bound on the treewidth that this condition generates, subsequently makes it possible to formulate and answer the compatibility question in a computationally efficient way. However, this efficiency is purely theoretical in nature, obtained via the indirect route of monadic second order logic [19], and it remains a challenge to succinctly characterize phylogenetic compatibility. Since Bryant and Largergren various other authors have

picked up this thread (e.g. [32]), with particular attention for triangulation-based approaches (see e.g. [34, 79, 80]) although the question remains: what *exactly* is the role of treewidth in compatibility?

Here we take a step forward in understanding the link between treewidth and compatibility. We prove that if the display graph of a set of unrooted binary trees has treewidth at most 2, then the input trees are compatible, and this holds for any number of input trees. In other words, it is not necessary to look deeper into the structure of the display graph, compatibility is immediately guaranteed. The proof of this, based on graph separators and graph minors, is surprisingly involved. Moreover, we describe a simple polynomial-time algorithm to construct a supertree for the input trees, when this condition holds. We also show that in some sense this result is "best possible": we show how to construct both compatible and incompatible instances that have display graphs of treewidth 3, for any number of trees. This confirms that the treewidth of the display graph cannot, on its own, fully capture phylogenetic compatibility, and that auxiliary information is indeed necessary if we are to obtain a complete characterization.

We start with some definitions of unrooted trees and networks because so far we have only dealt with rooted trees. Most of the concepts are analogous, but we repeat them here for the sake of completeness.

6.1 Preliminaries

Let X be a finite set . An *unrooted phylogenetic X-tree* is a tree whose leaves are bijectively labeled by the elements of set X. It is called *binary* when all its inner nodes (nonleaf nodes) are of degree 3. An unrooted binary phylogenetic tree on four leaves is called a *quartet*. In the remainder of the article we focus almost exclusively on unrooted binary trees, often writing simply trees or X-trees for short.

As before, we call elements of X *taxa* or *leaves*. For some X-tree T and some subset $X' \subseteq X$ we denote by $T(X')$ the subtree of T induced by X' and by $T|X'$ the tree obtained from $T(X')$ by suppressing vertices of degree 2. Furthermore, we say a tree S *displays* a tree T if T can be obtained from a subgraph of S by suppressing vertices of degree two.

Given a set X a *split* is defined as a bipartition of X. If we label the components of the partition by A and B, then we can denote the split by $A|B$. Note that each edge of an X-tree naturally induces a split. If $A|B$ is a split induced by an edge of a tree T, then we say that T *contains* split $A|B$. We use $ab|cd$ to denote the quartet in which taxa a and b are on one side of the internal edge and c and d are on the other. We write $ab|cd \in T$ if T displays $ab|cd$.

Given a set $\mathcal{T}$ of k trees $T_1, ..., T_k$ we wish to know if there exists a single tree S that displays T_i for all $i \in \{1, ...k\}$. A tree that displays all the input trees, if such a tree exists, is called a *supertree*. When a supertree does exist we call the instance *compatible*, otherwise *incompatible*. A supertree is not necessarily unique. To see when such a tree is unique and many more details on this topic we refer the reader to [24] or [74].

The *display graph* $D(\mathcal{T})$ of a set of trees $\mathcal{T}$ is the graph obtained from the disjoint union of trees in $\mathcal{T}$ by identifying vertices with the same taxon labels. Note that $D(\mathcal{T})$ can be disconnected if and only if the trees in $\mathcal{T}$ can be bipartitioned into two sets $\mathcal{T}_1, \mathcal{T}_2$ such that $X(\mathcal{T}_1) \cap X(\mathcal{T}_2) = \emptyset$, where $X(T)$ refers to the set of taxa of T. In such a case $\mathcal{T}$ permits a supertree if and only if both $\mathcal{T}_1$ and $\mathcal{T}_2$ do. Hence for the remainder of the article we focus on the case when $D(\mathcal{T})$ is connected.

Before we can start discussing our result we need a few graph theoretic definitions. Let $G = (V, E)$ be an undirected graph. For any two subsets of vertices $A, B \subseteq V$ and any $Z \subseteq V$ we say Z *separates* sets A and B in G if every path in G that starts at some vertex $u \in A$ and ends

at some vertex $v \in B$ contains a vertex from Z. Such a set Z is called an (A, B)-*separator*, or simply a *separator*. A graph M is a *minor* of a graph G if M can be obtained from a subgraph of G by contracting edges.

The treewidth of a graph G, denoted $tw(G)$, has a somewhat technical definition. We give it here for completeness although for the main result it is sufficient to note that trees have treewidth 1, and that graphs with treewidth at most 2 are exactly those graphs that do not have a K_4-minor (where K_4 is the complete graph on 4 vertices) [20]. We will also use the well-known fact that if M is a minor of G, $tw(M) \leq tw(G)$ [20].

Let G be a graph, T a tree and $(B_t)_{t \in T}$ a family of subsets of $V(G)$, also called *bags*, indexed by vertices of T. We say T is a *tree-decomposition* of G if the following conditions are satisfied:

(T_1) $V(G) = \cup_{t \in T} B_t$;

(T_2) for every edge $e \in G$ these exists a bag B_t in T such that both endpoints of e lie in B_t;

(T_3) $B_u \cap B_v \subseteq B_w$ whenever vertices u, v, w of T are such that w is on a path from u to v in T.

The width of a tree-decomposition is the size of its largest bag minus one. The *treewidth* of a graph G, also denoted $tw(G)$, is the minimum width over all possible tree-decompositions of G.

For remaining graph theory terminology we refer to standard texts such as [20].

6.2 Main results

We begin with some simple lemmas.

Lemma 6.1. [29, Corollary 1]. *Let T_1 and T_2 be two unrooted phylogenetic trees on the same set of taxa X. Then T_1 and T_2 are compatible if and only if there do not exist four taxa $a, b, c, d \subseteq X$ such that $ab|cd \in T_1$ and $ac|bd \in T_2$.*

Lemma 6.2. *Let D be the display graph of the two quartets $ab|cd$ and $ac|bd$. Then D has K_4 as a minor. Hence, $tw(D) \geq 3$.*

Proof. Both Q_1 and Q_2 have two inner nodes each. It is immediate to see that those four inner nodes form a K_4 minor in D. □

Theorem 6.3. *Let T_1 and T_2 be two unrooted phylogenetic trees. Let D be the display graph of T_1 and T_2. Then T_1 and T_2 are compatible if and only if $tw(D) \leq 2$.*

Proof. Let T_1 and T_2 be two trees on taxa sets X and X' respectively. Let $X^* = X \cap X'$. Then T_1 and T_2 are compatible if and only if $T_1|X^*$ and $T_2|X^*$ are compatible [74]. Thus we only have to consider two trees T_1 and T_2 on the same set of taxa X. Let $D(T_1, T_2)$ be their display graph. Suppose for the sake of contradiction that $tw(D(T_1, T_2)) \leq 2$ while T_1 and T_2 are incompatible. From Lemma 6.1, T_1 and T_2 contain incompatible quartets Q_1 and Q_2 (w.l.o.g. let T_i display Q_i) and since Q_i is displayed in T_i, $D(Q_1, Q_2)$ is a minor of $D(T_1, T_2)$. Since $D(Q_1, Q_2)$ is a minor of G, and using Lemma 6.2, $tw(G) \geq tw(D(Q_1, Q_2)) \geq 3$, contradicting the fact that $tw(D(T_1, T_2)) \leq 2$. This completes our proof in one direction; for the other see [12]. □

In the following main theorem we emphasize that the trees in $\mathcal{T}$ do not need to be on the same set of taxa, but that for this proof the input trees do need to be binary.

Theorem 6.4. *Let $\mathcal{T}$ be a set of k binary unrooted phylogenetic trees $T_1, ..., T_k$ and let D be their display graph. If $tw(D) \leq 2$, then $T_1, ..., T_k$ are compatible, in which case a supertree can be constructed in polynomial time.*

Proof. We give a constructive proof in which we will build a supertree S for $\mathcal{T}$. The idea is to find an appropriate separator of D and to reduce the problem into smaller instances of the same problem i.e. an induction proof. The induction will be on the cardinality of $X = \cup_{T_i \in \mathcal{T}} X(T_i)$. For the base case observe that an instance with $|X| \leq 3$ is trivially compatible.

Before we start the construction we apply a number of operations on D that are safe to do, in the sense that they preserve (in)compatibility of the instance and do not cause the treewidth of D to rise. We remove any taxon that has degree 1 in D and contract any inner vertex that has degree 2 in D. This clearly affects neither the compatibility nor the treewidth. Furthermore, for every tree T_i with $i \in \{1, ...k\}$ that has fewer than 4 leaves, we exclude it from the display graph. Such a tree carries no topological information and thus does not change the compatibility, while removing something from a graph cannot increase its treewidth. The *cleaning up* procedure means that we apply all these operations on D repeatedly until we cannot apply them anymore (see figure 6.1). In other words, we can assume D to have treewidth exactly 2, that all inner vertices of D have degree 3, that all taxa have degree at least 2 and that no tree has fewer than 4 taxa.

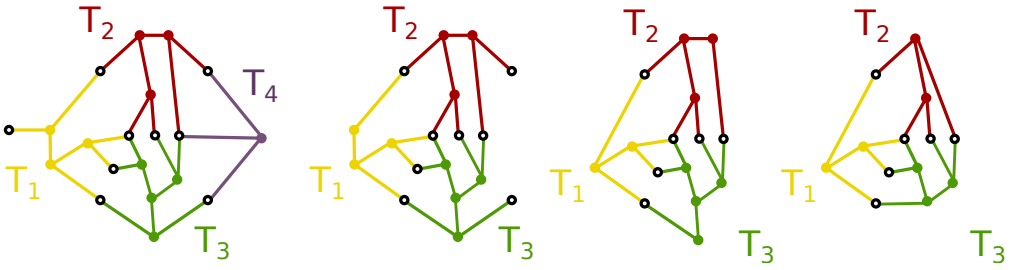

Figure 6.1: An example of cleaning up procedure. We start with a display graph of four trees and in the first step we remove the whole tree T_4 since it only has 3 leaves and we remove degree 1 taxon of T_1. In the second step we compress the resulting degree 2 inner vertex of T_1 and remove degree 1 taxa of T_2 and T_3. In the last step we compress the resulting degree 2 inner vertices of T_2 and T_3. We cannot apply any of the operations anymore.

Consider a planar embedding of the display graph $D(\mathcal{T})$. This exists and can be found in polynomial time because $D(\mathcal{T})$ has treewidth at most 2. The *boundary* of a face F of $D(\mathcal{T})$, denoted $B(F)$, is the set of edges and vertices that are incident to the interior of the face. We say that two distinct faces F_1, F_2 are *minimally adjacent* if the following three conditions hold: (1) F_1 and F_2 are adjacent; (2) $B(F_1) \cap B(F_2)$ is isomorphic to a path containing at least one edge; (3) the internal vertices of the path $B(F_1) \cap B(F_2)$ all have degree 2 in $D(\mathcal{T})$, and the two endpoints of the path each have degree 3 or higher in $D(\mathcal{T})$.

We now show that if the treewidth of D is 2 we can always find two such faces, neither equal to the outer face, in polynomial time.

Observation 6.5. *Let $\mathcal{T}$ be a non-empty set of unrooted binary trees on X such that $tw(D(\mathcal{T})) = 2$ and assume $D(\mathcal{T})$ is cleaned up. Consider any planar embedding of $D(\mathcal{T})$. Then there exist two distinct faces F_1, F_2 in $D(\mathcal{T})$ such that F_1 and F_2 are adjacent and neither is equal to the outer face.*

Proof. Without loss of generality we prove this for the case when $D(\mathcal{T})$ is connected. Recall that all vertices in $D(\mathcal{T})$ have degree at least 2, and at least one vertex has at least degree 3 (due to the existence of internal nodes). Hence, by the handshaking lemma, the number of edges in $D(\mathcal{T})$ is strictly larger than the number of vertices, and thus we can use Euler's formula to

conclude that $D(\mathcal{T})$ has at least 3 faces. One of these is the outer face, so $D(\mathcal{T})$ has at least two faces not equal to the outer face. Hence, $D(\mathcal{T})$ contains at least two simple cycles. If any two simple cycles have a common edge, then we are done, so let us assume that all simple cycles in $D(\mathcal{T})$ are edge disjoint (and chordless). However, this is not possible due to the fact that in every simple cycle at least two vertices have degree 3 or higher and the fact that all vertices in the graph have degree at least 2. (In particular, a simple cycle can never act as a "sink" to absorb excess degree, and there are also no leaves to fulfill this function.) □

Observation 6.6. *Let $\mathcal{T}$ be a cleaned up, non-empty set of unrooted binary trees on X such that $tw(D(\mathcal{T})) = 2$. Consider any planar embedding of $D(\mathcal{T})$. Let e be a cut-edge of $D(\mathcal{T})$, and let D_1, D_2 be the two components obtained by deleting e. Then both D_1 and D_2 have their own pair of adjacent faces, neither equal to the outer face.*

Proof. This is a simple adaptation of the previous proof. Deleting e reduces the degree of two vertices by exactly one, and all other degrees are unchanged. So D_1 and D_2 both contain at most one vertex of degree 1. From the previous "sink" observation we see that both D_1 and D_2 must contain two simple cycles with intersecting edges, and we are done. □

Lemma 6.7. *Let $\mathcal{T}$ be a cleaned up, non-empty set of unrooted binary trees on X such that $tw(D(\mathcal{T})) = 2$. Consider any planar embedding of $D(\mathcal{T})$. Then there exist two distinct faces F_1, F_2 in $D(\mathcal{T})$ such that F_1 and F_2 are* minimally *adjacent and neither is equal to the outer face. Also, these can be found in polynomial time.*

Proof. Fix any planar embedding of $D(\mathcal{T})$. Let G be the graph whose vertices are the faces of $D(\mathcal{T})$ (including the outer face) and whose edges are the adjacency relation on those faces. We label each face F of G with the length of a shortest path in G from F to the outer face. Clearly, the outer face has label 0. Let k be the maximum label ranging over all faces. We select a pair of distinct faces (F_1, F_2) such that (1) F_1 has label k; (2) F_2 is adjacent to F_1; (3) F_2 has the largest label ranging over all faces that are adjacent to a face with label k. By Observation 6.5, F_1 and F_2 both have label at least 1. Note also that the label of F_2 is either $k-1$ or k. Clearly, F_1 and F_2 both satisfy property (1) of minimal adjacency.

Consider now the sequence of vertices and edges $v_1, e_1, v_2, e_2, \ldots, v_n = v_1$ that define the boundary of face F_1. Observe that with the exception of $v_1 = v_n$ all vertices on the boundary are distinct. This is because, if the boundary of the face intersects with itself, it creates a new face F_3 "inside" F_1 whose shortest path to the outer face is strictly larger than k, contradicting the minimality of k. Hence, $B(F_1)$ is a simple cycle. From this it follows that $B(F_1) \cap B(F_2)$ is a subgraph of a simple cycle. In particular, it can be (a) a simple cycle or (b) a set of one or more paths (where some of the paths might have length 0). We show that (a) cannot happen. To see this, observe that (a) can only happen if $B(F_1) \subseteq B(F_2)$. From the degree constraints mentioned earlier the simple cycle defining F_1 contains at least 2 vertices of degree 3 or higher, in $D(\mathcal{T})$. These two vertices u_1, u_2 generate paths that cannot enter the interior of F_1, because they would then necessarily slice F_1 up into smaller faces. Moreover, there cannot exist a path from u_1 to u_2 that avoids $B(F_1)$, because this would imply the existence of a third face F_3 adjacent to F_1, such that $B(F_1) \cap B(F_3)$ contains an edge not in $B(F_2)$. In particular, this would contradict $B(F_1) \subseteq B(F_2)$. For a similar reason, the generated paths cannot re-intersect with $B(F_1)$. Careful analysis shows that the only remaining possibility is that F_2 is the outer face, contradicting the fact that the label of F_2 is at least 1. Hence we conclude that (a) is not possible, and that (b) must hold.

We now establish property (2) of minimal adjacency. In particular we show that $B(F_1) \cap B(F_2)$ has a single component. By the definition of adjacency, and the fact that (b) holds, at least one

component in $B(F_1) \cap B(F_2)$ is a path P on one or more edges. Clearly, the two endpoints of P must (in $D(\mathcal{T})$) have degree 3 or higher, otherwise P could be extended further. Without loss of generality consider the lower endpoint u. Let e be an edge incident to u (in $D(\mathcal{T})$) that is not in $B(F_1) \cap B(F_2)$ but which is incident to F_1 (such an edge must exist). The second face incident to e cannot be F_2, because otherwise e would be in P, so it must be some other face $F_3 \neq F_2$. Suppose there exists a path $P' \neq P$ in $B(F_1) \cap B(F_2)$. (Possibly, P' is a single vertex). In this case it is possible to draw a closed curve that passes through P and P' and such that the only face interiors that it intersects with, are those of F_1 and F_2 (see Figure 6.2). Informally this means that face F_3 is entirely "enclosed" by F_1 and F_2.

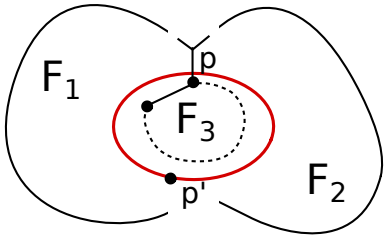

Figure 6.2: *If $B(F_1) \cap B(F_2)$ consists of two or more components P, P' then it is possible to draw a curve (shown in red) completely enclosing a third face F_3, yielding a contradiction on the choice of F_1 and/or F_2.*

More precisely, it means that any (shortest) path in G from F_3 to the outer face must pass through F_1 or F_2. If such a shortest path travels via F_1, then the label of F_3 is at least $k+1$, contradicting the maximality of k. If it travels via F_2, then it has label k or $k+1$. The latter is clearly a contradiction, but also the former because this contradicts our earlier choice of F_2 (i.e. we should have chosen F_3 instead of F_2). Hence, $B(F_1) \cap B(F_2)$ indeed consists of a single path P (containing at least one edge).

It remains to prove property (3). We have already established that the endpoints of P have degree 3 or more in $D(\mathcal{T})$. If P has no interior vertices, or all interior vertices of P have degree 2 in $D(\mathcal{T})$, we are done. So suppose P contains an interior vertex u of degree 3 or more in $D(\mathcal{T})$. Let e be an edge incident to u (in $D(\mathcal{T})$) that is not in $B(F_1) \cap B(F_2)$. Clearly, e starts a path that extends into the interior of F_1 or F_2. If e is not a cut-edge then the path it starts must re-intersect with the boundary of F_1 or F_2, but this causes a face to be partitioned into smaller pieces, which is not possible. Hence, e must be a cut-edge. From Observation 6.6 deleting e yields two or more adjacent faces that are entirely "enclosed" by F_1 or F_2. If they are enclosed by F_1 then they both have label $k+1$, which is a contradiction. If they are enclosed by F_2, and F_2 has label k, the same contradiction is obtained. If they are enclosed by F_2, and F_2 has label $k-1$, then they both have label k, contradicting the fact that we chose F_2 in the first place.

Polynomial time is assured since recognition of treewidth 2, planar embeddings and determination of the labels can all be computed in (low-order) polynomial time.

□

So let F_1 and F_2 be two minimally adjacent faces of D, neither equal to the outer face. Denote by $p(u, v)$ the path $B(F_1) \cap B(F_2)$ they share. (After locating F_1 and F_2 this path can easily be found in polynomial time). By definition u and v must have degree at least 3 in D. Also, by minimal adjacency of F_1 and F_2 and due to cleaning up, none of the interior nodes

of $p(u, v)$ can be internal tree nodes. Moreover, since we removed all trees on fewer than four taxa, at most one leaf can appear as an interior node of the path. Such a leaf can only exist if both u and v are inner nodes of some trees. Now, u and v can either be both leaves, both inner nodes or one of them a leaf another an inner node. These are the three cases we have to consider.

Case(i) is when both u and v are leaves. We claim this cannot happen. In this case, path $p(u, v)$ must be an edge. But if it is an edge it is connecting two leaves and will have already been removed during cleaning up.

Case(ii) is when u is a leaf and v is an inner node. Again we have that path $p(u, v)$ must be an edge (u, v) which both faces share. Let x, respectively y, be any vertex other than u or v on the boundary of F_1, respectively F_2. See Figure 6.3(a). We claim that any path between x and y must contain either u or v. In particular, suppose there exists a path $p(x, y)$ such that $u, v \notin p(x, y)$. Let x' and y' be vertices on $p(x, y)$ such that the subpath $p(x', y')$ is the shortest subpath of $p(x, y)$ with the property that both of its endpoints are on the boundaries of F_1 and F_2, respectively. See Figure 6.3(a). Then D contains a K_4 minor formed by vertices u, v, x' and y'. This is a contradiction on D having treewidth 2. So we have that any path between x and y passes through either u or v. Thus $\{u, v\}$ is a separator of D.

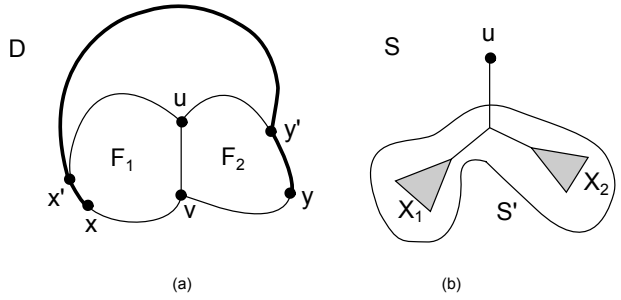

Figure 6.3: (a) Two minimally adjacent faces F_1 and F_2 in D. The vertices u, v, x', y' induce a K_4 minor. (b) A supertree as constructed in case (ii).

Removing u and v from the vertex set of D disconnects it and divides the set of taxa into two sets X_1 and X_2, such that $X = X_1 \cup X_2 \cup \{u\}$. We claim that supertree S as shown in Figure 6.3(b), where S' is a supertree of $T_1, ..., T_k$ restricted to taxa set $X \setminus \{u\}$, displays all k input trees $T_1, ..., T_k$. To prove this we have to show two things. One, that the supertree S' exists (and that it has an edge corresponding to split $X_1|X_2$) and two, that all quartets in $T_1, ..., T_k$ are also in S. The latter is sufficient because of a well known result in phylogenetics that a set of unrooted trees is compatible if and only if the set of quartets displayed by the trees is compatible [74].

To prove the first claim let $X' := X \setminus \{u\}$ and notice that by induction the instance $T_1, ..., T_k|X'$ is compatible and thus has a supertree. We now claim that there exists some supertree of $T_1, ..., T_k|X'$, call it S', which contains split $X_1|X_2$. First of all notice that (a restriction of) $X_1|X_2$ must be a split in every input tree restricted to X'. To see this we show that there does not exist a quartet $ab|cd$ with $a, c \in X_1$ and $b, d \in X_2$ in any of the input trees (prior to removal of u and v). Suppose such a quartet did exist in some tree. Then there would exist edge-disjoint paths $p(a, b)$ and $p(c, d)$ in D, where the interior nodes of these paths are internal tree nodes. Since removing u and v from D disconnects it (such that X_1 and X_2 are subsequently in separate components), it must be that those paths had to use either u or v. Since u is a taxon it cannot be used for this purpose. So both paths had to use inner vertex v. However, this

contradicts the edge-disjointness of the two paths. Hence quartet $ab|cd$ cannot be displayed by any tree.

We conclude from this that in each $T_i|X'$ there exists an edge e that induces a split $A|B$, such that $A \subseteq X_1$ and $B \subseteq X_2$. Furthermore both X_1 and X_2 must contain at least one taxon each. (This follows because edge (u, v) belongs to some input tree T, and walking from u to v along the boundary of F_1 whilst avoiding edge (u, v) necessitates entering and leaving T via its taxa, which in turn means that some taxon not equal to u must exist on the part of the boundary of F_1 not shared by F_2. The same argument holds for F_2.) As such, in each $T_i|X'$ it is possible to contract (the subtree induced by) X_1 and/or X_2 into a single "meta-taxon".

Let T^* (respectively, T^{**}) be the set of trees obtained by taking the trees on X' and contracting all the X_2 (respectively, X_1) taxa into a single meta-taxon W_2 (respectively, W_1). Note that contracting in this way cannot increase the treewidth of D and that $1 \leq |X_i| < |X|$ for $i \in \{1, 2\}$. Hence, by induction supertrees of T^* and T^{**} exist. Finally, construct supertree S' with split $X_1|X_2$ from two supertrees for T^* and T^{**} by adding an edge between W_1 and W_2 and afterwards suppressing W_1 and W_2. (The function of W_1 and W_2 was precisely to ensure that we would know how to glue the two separately constructed supertrees together).

To see the second claim note that since S' is a supertree of $T_1, ..., T_k$ restricted to $X \setminus \{u\}$ we only have to show that quartets of $T_1, ..., T_k$ that contain taxon u are displayed by S. So w.l.o.g. let $a \in X_1, b, c \in X_2$. Then if quartet $au|bc$ is displayed by some input tree T it is also clearly displayed by the supertree S. We claim quartets $ub|ac$ or $uc|ab$ cannot exist in any of the input trees. These two quartets are the same up to relabeling so let's consider quartet $ub|ac$ induced by some tree T sitting inside D. Then $p(u, b)$ and $p(a, c)$ are edge-disjoint and contain no taxa. As argued before $p(a, c)$ must pass through v. But since (u, v) is an edge it follows that it must belong to the same tree T, and therefore v also lies on the path $p(u, b)$. But then it is not possible that T displays $ub|ac$, contradiction.

Case(iii) is when both u and v are inner nodes. We could have that $p(u, v)$ is an edge, in which case u and v are inner nodes of the same tree, or we could have that $p(u, v)$ contains a single taxon t. Note that in the latter case u and v are inner nodes of two different trees and taxon t must have degree 2 in D due to the minimal adjacency of F_1 and F_2. The argument for $\{u, v\}$ being a separator of D goes through in this case as well regardless of $p(u, v)$ being an edge or a path containing a single taxon t. We again denote by X_1 and X_2 the two sets of taxa that emerge from splitting D by removing u and v (and t if it exists on (u, v)).

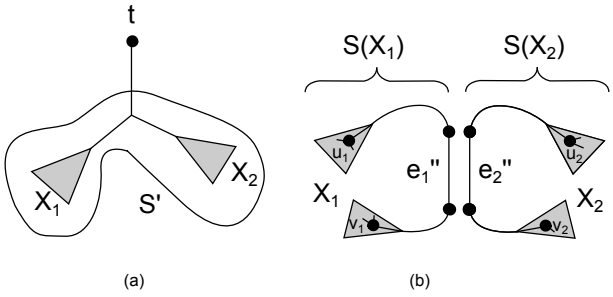

Figure 6.4: (a) A supertree constructed in case (iii) when there exists a taxon t on the common boundary of the two faces. (b) Construction of a supertree in case (iii) when the common boundary of the two faces is a single edge.

Subcase 1. Consider first the subcase when some taxon $t \in p(u, v)$. As before we have to show that there exists some S', a supertree of $T_1, ..., T_k$ restricted to $X' := X \setminus \{t\}$ with split

$X_1|X_2$, and that the supertree S as shown in Figure 6.4(a) displays all quartets induced by $T_1, ..., T_k$. The proof for this case is almost identical to that of case (ii). There we used the fact that u was a leaf. We now show that all the statements made in case (ii) also hold when u is an inner node and t is a taxon on path $p(u, v)$.

We saw that in case (ii) both X_1 and X_2 were nonempty and of strictly smaller cardinality than X. That they are strictly smaller than X is the case here as well since $t \in X$ but $t \notin X_1$ and $t \notin X_2$. That they are nonempty in this case also holds. Consider face F_1. Let u be a node of some tree T_1 and v a node of some other tree T_2. Then the path from u to v that follows the part of the boundary F_1 not shared by F_2, is a path between two vertices of different trees and so must contain some taxon $a \neq t$ (which is also in X_1). The same argument holds for F_2 and X_2. Another thing we have to show here is that all input trees $T_1, ..., T_k$ restricted to X' respect split $X_1|X_2$ (i.e. each tree contains an edge e inducing a split $A|B$ where $A \subseteq X_1$ and $B \subseteq X_2$). Suppose there exists a quartet $ab|cd$ with $a, c \in X_1$ and $b, d \in X_2$ in some input tree T. Since removing u and v from D disconnects it, it must be that two paths $p(a, b)$ and $p(c, d)$ had to pass through either u or v. W.l.o.g. let $u \in p(a, c)$ and $v \in p(b, d)$. Furthermore paths $p(a, b)$ and $p(c, d)$ are edge-disjoint in D and belong to the same tree T. But this is impossible since u and v belong to different trees in this case. Contradiction. So we conclude in this case too that every input tree restricted to X' respects split $X_1|X_2$, and hence the contraction of X_1 and X_2 into meta-taxa works exactly as described in case (ii). Hence, S' indeed exists, can be constructed and has split $X_1|X_2$.

Next, we claim that S as shown in Figure 6.4(a) displays all quartets induced by $T_1, ..., T_k$. As before, since S' is a supertree of $T_1, ..., T_k|X'$ we only need to check the quartets induced by $T_1, ..., T_k$ that contain taxon t. W.l.o.g. let $a \in X_1$ and $b, c \in X_2$. There are three possible topologies $at|bc, bt|ac, ct|ab$. As before, $at|bc$ is an easy case since if it appears in some T in D it clearly also appears in S, while topologies $bt|ac$ and $ct|ab$ are the same up to relabeling. So consider $bt|ac$ and suppose it is displayed by some tree T. Paths $p(b, t)$ and $p(a, c)$ are edge-disjoint and since the degree of t is 2 in D we have that $p(b, t)$ has to contain either u or v. W.l.o.g. let $u \in p(b, t)$. Now, since $\{u, v\}$ is a separator of D and $a \in X_1$ while $c \in X_2$, we have that either $u \in p(a, c)$ or $v \in p(a, c)$. If $u \in p(a, c)$ we have that paths $p(b, t)$ and $p(a, c)$ both contain node u, a contradiction on $bt|ac$ being displayed by T. If $v \in p(a, c)$ then edge (v, t) and (u, t) must both belong to the same tree T, which contradicts our earlier observation that u and v are necessarily in different trees.

Subcase 2. The last thing to consider is the subcase when (u, v) is an edge while both u and v are inner nodes (necessarily of the same tree T). Let X_1 and X_2 be two disjoint sets of taxa that result from splitting D after removing u and v. We claim that $|X_1| \geq 2$ and $|X_2| \geq 2$. This follows directly from $u, v \in T$: any cycle that links them together must leave the tree T via some taxon a and re-enter it via a (necessarily different) taxon b. Since u and v belong to both faces F_1 and F_2 it follows that the boundaries of these two faces must each contain (at least) two taxa. The two taxa on the boundary of (w.l.o.g) F_1 are still in the same connected component after deletion of $\{u, v\}$, but are not in the same connected component as the taxa from the boundary of F_2, so $|X_1| \geq 2$ and $|X_2| \geq 2$.

Now we claim that the tree shown in Figure 6.4(b) is a supertree of $T_1, ..., T_k$. Let's first explain what that image means. Note that apart from the tree T in which the internal edge $e = (u, v)$ can be found, all other trees have taxa sets either completely contained inside X_1 or completely contained inside X_2. This is the case because otherwise there would be a path from some element in X_1 to some element in X_2, contradicting the fact that $\{u, v\}$ is a separator. The idea is to cut T into two parts, one on X_1, one on X_2, recursively build supertrees of $T_1, ..., T_k|X_1$ and $T_1, ..., T_k|X_2$ and join them as indicated in the figure.

Now, consider the display graph D. Suppose we delete the edge $e = (u, v) \in T$, and replace

it with two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ (where u_i and v_i are u and v duplicated). Because $\{u, v\}$ is a separator, this creates two disjoint display graphs, one on X_1 and one on X_2. These are minors of the original display graph so have treewidth at most 2, and they are smaller instances of the problem. So by induction supertrees of these smaller instances exist. Let $S(X_1)$ be a supertree on X_1 and $S(X_2)$ be a supertree on X_2. All trees except T will be displayed by the disjoint union of $S(X_1)$ and $S(X_2)$, because only T has taxa from both X_1 and X_2. What is left to explain is how to glue $S(X_1)$ and $S(X_2)$ into a supertree S such that S displays T as well.

Note that $S(X_i)$ contains an image of edge e_i. The image need not be an edge in $S(X_i)$, it could also be a path, whose endpoint we denote by u_i and v_i in Figure 6.4(b). Take any edge on path $p(u_i, v_i)$, call it e'_i, and subdivide it twice to create two adjacent degree-2 vertices; let e''_i be the edge between them. Now, by identifying e''_1 and e''_2 we ensure that we get a supertree that displays (all the quartets in) T, as well as all the other trees.

This completes the case analysis. Polynomial time is achieved because all relevant operations (recognizing whether a graph has treewidth at most 2, finding a planar embedding, finding two minimally adjacent faces, finding the separator $\{u, v\}$, and all the various tree manipulation operations) can easily be performed in (low-order) polynomial time. □

We now give a summary of the algorithm that has already been implicitly described in the above proof. Given k input phylogenetic trees, construct their display graph D and clean it up. We start by verifying in polynomial time that the treewidth of D is at most 2. Let F_1 and F_2 be any two minimally adjacent faces of D (if two such faces do not exist, then the instance is trivially compatible). By definition of minimal adjacency we know that the intersection of borders of the two faces must be isomorphic to a path $p(u, v)$ containing at least one edge. Denote by X_1 and X_2 are two sets of taxa obtained from separating D by removing $\{u, v\}$.

We saw that we can w.l.o.g. assume v to be an inner node. If u is a leaf, then we construct a supertree S as shown in figure 6.3(b) and recursively solve two smaller instances with input trees $T_i|X_1$ and $T_i|X_2$ for $i \in \{1, ..., k\}$. (Note that in the actual algorithm we also add an extra "meta-taxon" into each of the two smaller instances which tells us where to graft the two solutions back together). Otherwise, u is an inner node. In this case, path $p(u, v)$ can either contain a taxon t or be an edge. When it contains a taxon t a supertree S is given in figure 6.4(a) and we recursively solve two smaller instances with input trees $T_i|X_1$ and $T_i|X_2$ for $i \in \{1, ..., k\}$ (note that in this case the two taxa sets X_1 and X_2 are obtained after removing $\{u, v, t\}$ from D). When u is an inner node and $p(u, v)$ is an edge, then we construct a supertree as in figure 6.4(b) and recursively solve two smaller instances $T_i|X_1$ and $T_i|X_2$ for $i \in \{1, ..., k\}$. We continue until the instance is trivially compatible.

6.3 Beyond treewidth 2

Two incompatible quartets induce a display graph with treewidth 3, so treewidth 3 cannot guarantee compatibility. However, it is natural to ask whether treewidth 3 guarantees compatibility if the number of input trees becomes sufficiently large. Unfortunately, the answer to that question is no. Namely, for any number of trees there exists a compatible instance with $tw(D) = 3$ and an incompatible instance with $tw(D) = 3$, as we now demonstrate.

Consider figures 6.5 and 6.6. They both show the display graph of k trees with leaves indicated by black circles and inner nodes of each tree indicated by filled circles in the same color as that of the tree they correspond to. As it can be easily verified either by any available software or by giving an optimal tree decomposition (see figure 6.7), the treewidth of both of these display graphs is exactly 3.

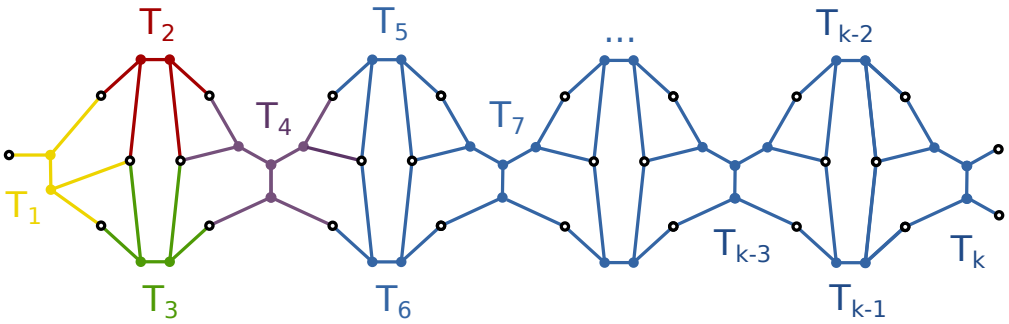

Figure 6.5: Display graph of an instance with k input trees. Leaves of all trees in this image are marked by black circles, while inner nodes are marked by filled circles in the same color as the tree they belong to. The treewidth of D is 3 and the instance is compatible.

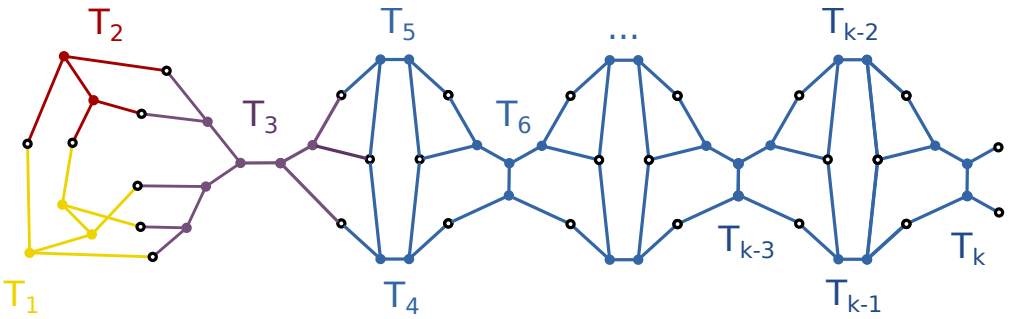

Figure 6.6: Display graph of an instance with k input trees. Leaves are marked by black circles, while inner nodes are marked by filled circles in the color of the corresponding tree. The treewidth of D is 3 and the instance is incompatible.

The display graph in figure 6.5 however shows an instance that is compatible. This can be verified without too much difficulty. We already argued that the cleaning up procedure preserves (in)compatibility; we will use this now. Notice then that we can remove two leafs of T_k with degree 1, making T_k a tree on three leaves. We can thus remove the whole tree, thereby making T_{k-1} and T_{k-2} trees on three leaves. Removing T_{k-1} and T_{k-2} makes T_{k-3} a tree on three leaves etc. In the end we are only left with a single tree, T_1. Since a single tree is trivially compatible it follows that all k trees are compatible.

Figure 6.6 on the other hand shows an incompatible instance. We can again start cleaning up this instance from T_k backwards until we are left with T_1, T_2 and T_3. Figure 6.8, first panel, shows the display graph of T_1, T_2 and T_3. We can do one more cleaning up step and end up with a graph in the middle panel (we abuse the notation slightly and call the cleaned up version of T_3, T_3 again). We argue that the three trees in the middle panel of Figure 6.8 are incompatible.

In the third panel of Figure 6.8 we try to build a supertree of the three trees in order to reach a contradiction. We start with T_3 and add T_1 to it. T_3 already contains leaves 1,2,3 of T_1 so we only have to decide where to add leaves 6 and 7. Notice that T_1 requires paths $p(1,6)$ and $p(2,7)$ to be disjoint. Furthermore, an inner vertex k of T_1 is (by definition of a tree) the only vertex with a property that paths $p(1,k), p(2,k), p(3,k)$ are edge-disjoint. If we want to map it to T_3 there is only one place where it can be (as indicated in the third panel of the Figure 6.8). From the condition that paths $p(1,6)$ and $p(2,7)$ must be disjoint, it follows that leaf 6 has to

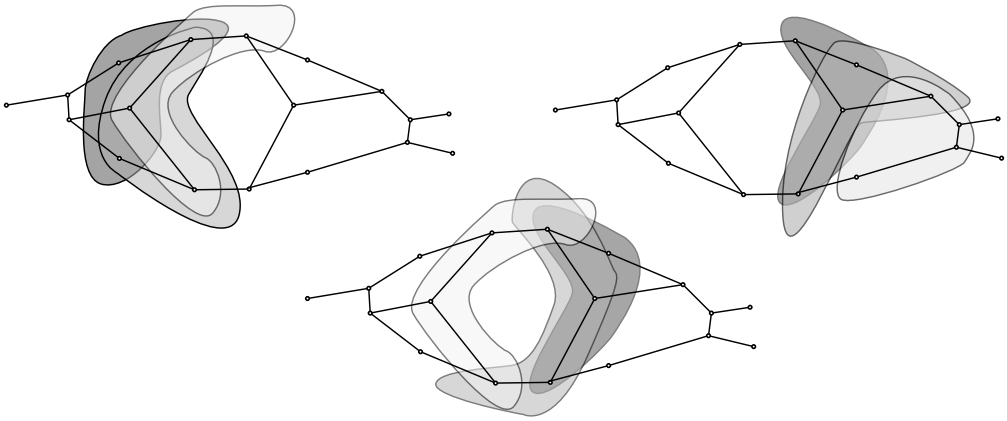

Figure 6.7: Tree decomposition of display graph of k compatible trees given in figure 6.5 such that each bag contains exactly four vertices. Due to the symmetry of the graph, we only give decomposition for the first part.

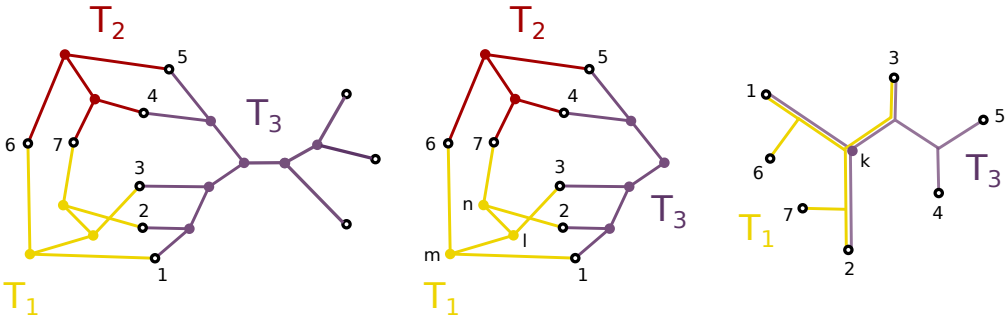

Figure 6.8: A display graph of three incompatible trees; A cleaned-up display graph of the same three trees; A unique supertree of T_1 and T_3

be attached somewhere on the path $p(1, k)$ and leaf 7 has to be attached somewhere on the path $p(2, k)$. Since these two paths are in fact edges in T_3 it follows that there only one place where we can attach leaves 6 and 7 respectively. So the supertree of T_1 and T_3 is unique. Furthermore, it contains a quartet $45|67$ which in incompatible with tree $T_2 = 47|56$. So the three trees T_1, T_2 and T_3 are incompatible, and thus all k trees from Figure 6.6 must be incompatible.

Figure 6.9 summarizes our results. The red area is due to result of Bryant and Lagergren which proves that that any instance on k trees whose display graph has treewidth strictly greater than k must be incompatible. The green area is due to our result. What we are left with is the gray area in which (as demonstrated by the constructions in the previous section) we cannot conclude anything about compatibility of the instances based only on treewidth of the display graph and the number of trees, at least not with the current results. An obvious open question is whether existing characterizations (such as legal triangulations [79]) can be specialized to yield simple and efficient combinatorial algorithms in the case of treewidth 3 or higher.

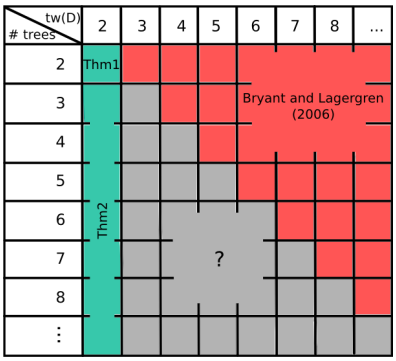

Figure 6.9: The green (respectively, red) area shows which combinations of (number of input trees, treewidth of display graph) are always compatible (respectively, incompatible). The gray area indicates that both compatible and incompatible instances exist for this combination of parameters.

Chapter 7

Conclusion

We looked at two well-known and well-studied problems in phylogenetics, minimum hybridization (Chapters 2-5) and compatibility (Chapter 6). Even though both of these problems deal with combining phylogenetic trees into larger structures, they are of very different nature. First of all their inputs are different: minimum hybridization takes rooted trees while compatibility (the variant we studied) is a problem defined on unrooted trees. Furthermore, minimum hybridization requires all input trees to have an identical leaf label set, while for compatibility trees can have overlapping label sets. The outputs of these two problems are different too: minimum hybridization combines trees into a network (DAG), while compatibility looks for a supertree. The reason underlying this is that in the unrooted setting checking whether trees can be merged into a bigger tree is a hard problem, which was not the case when we considered rooted trees.

This is why Chapter 6 is rather separate from the rest of the work done here. Our approach was to explore further the relationship between treewidth of a display graph and compatibility of the instance. This relationship has already been studied and it led to an efficient albeit nonconstructive algorithm for compatibility of unrooted trees. The motivation for our work was searching for a clean combinatorial algorithm for this problem. We wanted to know if incompatibility leaves some sort of a signal or a hint for us in the display graph; if presence (or absence) of certain structures can tell us something about weather partial trees all came from a same tree. And indeed we realized that absence of a K_4 minor in a display graph of any number of trees guarantees compatibility of that instance. We also gave a polynomial time algorithm to construct a supertree. A clean combinatorial algorithm that efficiently combines any number of unrooted trees into a supertree (if one exists) still remains an open question: our algorithm applies only to the case when the display graph has treewidth at most 2.

From the chapters that deal with minimum hybridization, the first three are very closely related. They were produced as a series of papers, each inspired by the previous one. We started with a theoretical result in Chapter 2: we showed that MH is in APX if and only if DFVS is in APX. This result also had an interesting consequence for the fixed parameter tractability of MH. We saw that the inflation factor of 6 in the reduction from DFVS to MH was very closely linked to a reduction described by Bordewich and Semple [10]. They showed that the input trees can be reduced to produce a weighted instance containing at most $14r$ taxa which we have sharpened to at most $9r$ taxa. Without this sharpening, the inflation factor we obtain would have been higher than 6. From this analysis it became clear that the kernel size has an important role to play in analyzing the approximability of MH.

This raised some interesting general questions about the linkages between MH and DFVS. For example, the reduction by Bordewich and Semple, which gives a linear kernel for a weighted

variant of MH, can be modified slightly (as is e.g. done in [8] for unrooted SPR distance) to obtain a quadratic kernel for MH (without weights). This contrasts sharply with DFVS. It is known that DFVS is fixed parameter tractable [15], but it is *not* known whether DFVS permits a polynomial-size kernel. Might MH give us new insights into the structure of DFVS (and vice-versa)? More generally: within which complexity frameworks is one of the two problems strictly harder than the other?

Another interesting consequence of this result is that approximating hybridization number by splitting it into MAF and DFVS instances yields an extremely competitive practical algorithm. This is the topic of Chapters 3 (approximating hybridization number for two binary trees) and Chapter 4 (approximating hybridization number for two nonbinary trees). In both chapters we presented an algorithm and tested its performance. Our experiments with binary trees show that CYCLE KILLER is much faster than available exact methods once the input trees become sufficiently large and/or discordant. In over 96% of the cases CYCLE KILLER finds the optimal solution and in the remaining cases it finds a solution very close to the optimum. We have shown that the most accurate mode of the program produces solutions that are at most a factor 2 from the optimum. In practice, the average-case approximation ratio of the most accurate mode that we observed was 1.003. The fastest mode of the algorithm on the other hand can be used on trees with thousands of leaves and provably constructs networks that are at most a factor of 4 from the optimum. The cycle-breaking technique also gives very good results when generalized to nonbinary trees. The algorithm is able to deal with very large instances that exact solvers will probably never be able to cope with. The practical worst-case approximation ratio observed in our experiments was 1.5 for the more accurate but slower mode of the algorithm and 4 for the faster but less accurate mode. Furthermore, in this chapter we have given improved FPT and polynomial-time approximation algorithms for nonbinary MAF.

Our methods showed how, with a d-approximation for DFVS and a c-approximation for MAF, we can obtain an approximation factor of $d(c+1)$ in the binary case and $d(c+3)$ in the nonbinary case. Might it be possible to also obtain $d(c+1)$ for the nonbinary case, or is the nonbinary case somehow inherently less approximable using these techniques? Also, as we wrote in the original papers, approximation ratios of this form have the advantage that they automatically improve (both theoretically and in practice) as techniques for solving DFVS and MAF improve. Since publishing the original papers we have seen this happen. For example, for a long time the best approximation for MAF on two rooted binary trees was 3, recently a polynomial-time approximation ratio of 2.5 was given in [77]. Even though we don't tackle unrooted MAF in this thesis, there has been recent progress on that front. In [76] authors study MAF on both multiple rooted and multiple unrooted trees and give a polynomial-time approximation algorithm with approximation ratio 3 and 4 respectively. In the case of nonbinary trees, the first FPT algorithm and the first constant-ratio approximation for unrooted MAF are given in [14].

Returning to hybridization number, an obvious next step is to try to build algorithms that can deal with multiple binary (and nonbinary) trees, which we presented in Chapter 5. However, this problem is relatively unexplored (at least when compared to the two tree case). When we are dealing with only two trees the standard approach to constructing a hybridization network is to puzzle it from the components of some AAF by wiring them appropriately. How this wiring should be done is completely determined by the AAF: each of the two hybridization edges above a component comes from a single tree. Furthermore, where these edges should be placed in the reconstruction of a network is also encoded in the AAF. In the case of more than two trees one must guess the wiring of the components. There is only a constant number of choices for wiring per component, so with $O(k)$ components and guessing the wiring of them, we get an $O(c^k \cdot \text{poly}(n))$ time algorithm for reconstructing a hybridization network.

The problem with this approach for multiple trees is the existence of what we called an invis-

ible component: a set of nodes of the network that is not represented in the AAF. Unfortunately, even for the case of only three trees we found examples where every optimal network contained an invisible component. In this particular case, the three trees instances, we realized that although these nodes might be invisible in AAF, we can identify them in the input trees. This eventually let to our $O(c^k \cdot \text{poly}(n))$-time algorithm. Sadly enough, the framework of our algorithm extends to any number of trees, except in this key observation that invisible nodes can be seen in the input trees afterall. With already just four trees we found examples of "truly invisible" nodes, nodes not found in AAF or in any of the input trees. Guessing the number and structure of these components seems very challenging and remains open.

Bibliography

[1] A. Aho, Y. Sagiv, T. Szymanski, and J. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[2] B. Albrecht, C. Scornavacca, A. Cenci, and D. Huson. Fast computation of minimum hybridization networks. *Bioinformatics*, 28(2):191–197, 2012.

[3] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[4] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L.Stougie, and J. Whitfield. Networks: expanding evolutionary thinking. *Trends in Genetics*, 29(8):439 – 441, 2013.

[5] M. Baroni, S. Grünewald, V. Moulton, and C. Semple. Bounding the number of hybridisation events for a consistent evolutionary history. *Journal of Mathematical Biology*, 51:171–182, 2005.

[6] M. Baroni, C. Semple, and M. Steel. A framework for representing reticulate evolution. *Annals of Combinatorics*, 8:391–408, 2004.

[7] H. Bodlaender and A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208(3):259–275, 2010.

[8] M.L. Bonet and K. St. John. On the complexity of uSPR distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):572–576, 2010.

[9] M. Bordewich, S. Linz, K. St. John, and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics*, 3:86–98, 2007.

[10] M. Bordewich and C. Semple. Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(3):458–466, 2007.

[11] M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.

[12] D. Bryant and J. Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351(3):296–302, 2006.

[13] P. Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205 – 212, 1974.

[14] Fan J. H. Chen, J. and S. H. Sze. Parameterized and approximation algorithms for maximum agreement forest in multifurcating trees. *Theoretical Computer Science*, 562(p 496-512), 2015.

[15] J. Chen, Y. Liu, S. Lu, B. O'sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), 2008.

[16] Z.Z. Chen and L. Wang. Hybridnet: a tool for constructing hybridization networks. *Bioinformatics*, 26(22):2912–2913, 2010.

[17] Z.Z. Chen and L. Wang. An ultrafast tool for minimum reticulate networks. *Journal of Computational Biology*, 20(1):38–41, 2013.

[18] J. Collins, S. Linz, and C. Semple. Quantifying hybridization in realistic time. *Journal of Computational Biology*, 18:1305–1318, 2011.

[19] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[20] R. Diestel. *Graph Theory*. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2000.

[21] I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 33–42, New York, NY, USA, 2002. ACM.

[22] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:2005, 2004.

[23] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[24] A. Dress, K. Huber, and J. Koolen. *Basic Phylogenetic Combinatorics*. Cambridge University Press, 2012.

[25] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[26] M. Fischer, L. van Iersel, S. Kelk, and C. Scornavacca. On computing the maximum parsimony score of a phylogenetic network. *SIAM Journal on Discrete Mathematics*, 29(1):559–585, 2015.

[27] A. Francis and M. Steel. Tree-like reticulation networkswhen do tree-like distances also support reticulate evolution? *Mathematical Biosciences*, 259:12–19, 2015.

[28] P. Gambette, V. Berry, and C. Paul. Quartets and unrooted phylogenetic networks. *Journal of Bioinfomatics and Computational Biology*, 10(4):1250004, 2012. http://hal.archives-ouvertes.fr/hal-00678046/en/.

[29] G. Ganapathy and T. Warnow. Approximating the complement of the maximum compatible subset of leaves of k trees. In Klaus Jansen, Stefano L.nardi, and Vijay Vazirani, editors, *Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 122–134. Springer Berlin Heidelberg, 2002.

[30] O. Gascuel. *Mathematics of Evolution and Phylogeny*. Oxford University Press, Inc., 2005.

[31] O. Gascuel and M. Steel, editors. *Reconstructing Evolution: New Mathematical and Computational Advances*. Oxford University Press, USA, 2007.

[32] S. Grünewald, P. Humphries, and C. Semple. Quartet compatibility and the quartet graph. *Electronic Journal of Combinatorics*, 15(1), 2008.

[33] D. Gusfield. *ReCombinatorics: The Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks*. MIT Press, 2014. http://mitpress.mit.edu/books/recombinatorics.

[34] R. Gysel, K. Stevens, and D. Gusfield. Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs. In Ben Raphael and Jijun Tang, editors, *Algorithms in Bioinformatics (Proceedings of WABI2012)*, volume 7534 of *Lecture Notes in Computer Science*, pages 93–105. Springer Berlin Heidelberg, 2012.

[35] K. Huber, L. van Iersel, V. Moulton, and T. Wu. How much information is needed to infer reticulate evolutionary histories? *Systematic Biology*, 2014.

[36] K. Huber and V. Moulton. Encoding and constructing 1-nested phylogenetic networks with trinets. *Algorithmica*, 66(3):714–738, 2013.

[37] K. Huber, L. van Iersel, V. Moulton, C. Scornavacca, and T. Wu. Reconstructing phylogenetic level-1 networks from nondense binet and trinet sets. *ArXiv e-prints*, November 2014.

[38] D. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2011.

[39] D. Huson and C. Scornavacca. Dendroscope 3 - a program for computing and drawing rooted phylogenetic trees and networks. Software available from: `www.dendroscope.org`, 2011.

[40] D. Huson and C. Scornavacca. A survey of combinatorial methods for phylogenetic networks. *Genome Biology and Evolution*, 3:23–35, 2011.

[41] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout. Constructing level-2 phylogenetic networks from triplets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(4):667–681, October 2009.

[42] L. van Iersel and S. Kelk. Constructing the simplest possible phylogenetic network from triplets. *Algorithmica*, 60(2):207–235, 2011.

[43] L. van Iersel and S. Kelk. When two trees go to war. *Journal of Theoretical Biology*, 269(1):245–255, 2011.

[44] L. van Iersel and S. Kelk. Kernelizations for the hybridization number problem on multiple nonbinary trees. In *WG14*, LNCS, 2014.

[45] L. van Iersel, S. Kelk, N. Lekić, and L. Stougie. Approximation algorithms for nonbinary agreement forests. *SIAM Journal on Discrete Mathematics*, 28(1):49–66, 2014.

[46] L. van Iersel and S. Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Information Processing Letters*, 113(9):318 – 323, 2013.

[47] L. van Iersel and V. Moulton. Trinets encode tree-child and level-2 phylogenetic networks. *Journal of Mathematical Biology*, 68(7):1707–1729, 2014.

[48] G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Efficient parsimony-based methods for phylogenetic network reconstruction. In *ECCB06*, volume 23(2) of *BIO*, pages e123–e128, 2006. http://www.cs.rice.edu/ nakhleh/Papers/eccb06.pdf.

[49] G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Maximum likelihood of phylogenetic networks. *Bioinformatics*, 22(21):2604–2611, 2006.

[50] G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Inferring phylogenetic networks by the maximum parsimony criterion: A case study. *Molecular Biology and Evolution*, 24(1):324–337, 2007. http://www.cs.rice.edu/ nakhleh/Papers/MBE06.pdf.

[51] V. Kann. *On the Approximability of NP-Complete Optimization Problems*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.

[52] R. Karp. *Complexity of Computer Computations*, chapter Reducibility among Combinatorial Problems, pages 85–103. Plenum Press, 1972.

[53] J. Keijsper and R. Pendavingh. Reconstructing a phylogenetic level-1 network from quartets. *Bulletin of Mathematical Biology*, 76(10):2517–2541, 2014.

[54] S. Kelk. TerminusEst. `http://skelk.sdf-eu.org/terminusest`.

[55] S. Kelk, L. van Iersel, and C. Scornavacca. Phylogenetic incongruence through the lens of monadic second order logic. *CoRR*, abs/1503.00368, 2015.

[56] S. Kelk and C. Scornavacca. Towards the fixed parameter tractability of constructing minimal phylogenetic networks from arbitrary sets of nonbinary trees. arXiv:1207.7034 [q-bio.PE], 2012.

[57] S. Kelk and C. Scornavacca. Constructing minimal phylogenetic networks from softwired clusters is fixed parameter tractable. *Algorithmica*, 68(4):886–915, 2014.

[58] S. Kelk, C. Scornavacca, and L. van Iersel. On the elusiveness of clusters. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2):517–534, 2012.

[59] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ϵ;. *Journal of Computer and System Sciences*, 74:335–349, May 2008.

[60] S. Linz and C. Semple. Hybridization in non-binary trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(1):30–45, 2009.

[61] C. McDiarmid, C. Semple, and D. Welsh. Counting phylogenetic networks. *Annals of Combinatorics*, 19(1):205–224, 2015.

[62] C. Meacham. Theoretical and computational considerations of the compatibility of qualitative taxonomic characters. In *Numerical taxonomy*, pages 304–314. Springer, 1983.

[63] D. Morrison. *Introduction to phylogenetic networks*. RJR Productions, Uppsala, 2011.

[64] L. Nakhleh. *The Problem Solving Handbook for Computational Biology and Bioinformatics*, chapter Evolutionary phylogenetic networks: models and issues. Springer, Berlin, 2009.

[65] R. Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006.

[66] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[67] H.J. Park and L. Nakhleh. Inference of reticulate evolutionary histories by maximum likelihood: The performance of information criteria. In *RECOMB-CG'12*, volume 13 of *BMCB*, page S12, 2012. http://dx.doi.org/10.1186/1471-2105-13-S19-S12.

[68] T. Piovesan and S. Kelk. A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees. 10(1):18–25, January 2013.

[69] E. Schroder. Vier combinatorische probleme. *Zeitschrift fur Mathematik and Physik*, 15:361–376, 1870.

[70] C. Scornavacca, S. Linz, and B. Albrecht. A first step towards computing all hybridization networks for two rooted binary phylogenetic trees. Submitted, preliminary version arXiv:1109.3268v1 [q-bio.PE].

[71] C. Semple. *Reconstructing Evolution - New Mathematical and Computational Advances*, chapter Hybridization Networks. Oxford University Press, 2007.

[72] C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105(1-3):147–158, 2000.

[73] C. Semple and M. Steel. A characterization for a set of partial partitions to define an x-tree. *Discrete Mathematics*, 247(13):169 – 186, 2002.

[74] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003.

[75] P Seymour. Packing directed circuits fractionally. *Combinatorica*, 15:281–288, 1995.

[76] F. Shi, J. Chen, Q. Feng, and J. Wang. Approximation algorithms for maximum agreement forest on multiple trees. *Computing and Combinatorics: 20th International Conference, COCOON Proceedings, Springer.*, 8591(p 381), 2014.

[77] Feng Q. You J. Shi, F. and J. Wang. Improved approximation algorithm for maximum agreement forest of two rooted binary phylogenetic trees. *Journal of Combinatorial Optimization*, (p 1-33), 2015.

[78] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.

[79] S. Vakati and D. Fernández-Baca. Graph triangulations and the compatibility of unrooted phylogenetic trees. *Applied Mathematics Letters*, 24(5):719–723, 2011.

[80] S. Vakati and D. Fernández-Baca. Characterizing compatibility and agreement of unrooted trees via cuts in graphs. *CoRR*, abs/1307.7828, 2013.

[81] C. Whidden. rSPR. `http://kiwi.cs.dal.ca/Software/RSPR`.

[82] C. Whidden, R. G. Beiko, and N. Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.

[83] C. Whidden, R. G. Beiko, and N. Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.

[84] C. Whidden, R.G. Beiko, and N. Zeh. Fast FPT algorithms for computing rooted agreement forests: Theory and experiments. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA)*, volume 6049 of *Lect Notes Comput Sc*, pages 141–153. 2010.

[85] C. Whidden and N. Zeh. A unifying view on approximation and FPT of agreement forests. In S. Salzberg and Tandy Warnow, editors, *Algorithms in Bioinformatics*, volume 5724 of *Lecture Notes in Computer Science*, pages 390–402. Springer Berlin / Heidelberg, 2009.

[86] S. J. Willson. Reconstruction of certain phylogenetic networks from their tree-average distances. *Bulletin of Mathematical Biology*, 75(10):1840–1878, 2013. http://www.public.iastate.edu/ swillson/Tree-AverageReconPaper9.pdf.

[87] Y. Wu. An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. *Journal of Computational Biology*, 20(10):792–804, 2013.

[88] J. Yang, S. Grünewald, Y. Xu, and X.F. Wan. Quartet-based methods to reconstruct phylogenetic networks. *BMC Systems Biology*, 80(21), 2014. http://dx.doi.org/10.1186/1752-0509-8-21.

[89] Y. Yu, R M. Barnett, and L. Nakhleh. Parsimonious inference of hybridization in the presence of incomplete lineage sorting. *Systematic Biology*, 62(5):738–751, 2013.

List of publications

1. **Cycle killer... qu'est-ce que c'est? On the comparative approximability of hybridization number and directed feedback vertex set**. Steven Kelk, Leo van Iersel, Nela Lekić, Simone Linz, Celine Scornavacca, Leen Stougie. *SIAM Journal on Discrete Mathematics* 2012, 26(4):1635.

2. **Approximation algorithms for nonbinary agreement forests**. Leo van Iersel, Steven Kelk, Nela Lekić, Leen Stougie. *SIAM Journal on Discrete Mathematics* 2014, 28(1):49.

3. **A practical approximation algorithm for solving massive instances of hybridization number for binary and nonbinary trees**. Leo van Iersel, Steven Kelk, Nela Lekić, Celine Scornavacca. *BMC Bioinformatics* 2014, 15(1):127.

4. **Hybridization number on three trees**. Leo van Iersel, Steven Kelk, Nela Lekić, Chris Whidden, Norbert Zeh. *SIAM Journal on Discrete Mathematics*, submitted.

5. **On low treewidth graphs and supertrees**. Alexander Grigoriev, Steven Kelk, Nela Lekić. *Journal of Graph Algorithms and Applications* 2015, 19(1):325.

6. **Satisfying ternary permutation constraints by multiple linear orders or phylogenetic trees**. Leo van Iersel, Steven Kelk, Nela Lekić, Simone Linz. *Theoretical Computer Science*, to appear.

Curriculum Vitae

Nela Lekić was born on December 19th in Novi Sad, Serbia. She attended Red Cross Nordic United World College and obtained an International Baccalaureate diploma in 2005. After obtaining her B.S. degree at Roosevelt Academy in Middelburg she was awarded Huygens Scholarship to complete a Masters degree in Mathematics at Utrecht University, where she obtained her diploma in 2011. In October 2011 she joined the Department of Knowledge Engineering at Maastricht University as a Ph.D. candidate under the supervision of Steven Kelk. The research carried out there is presented in this dissertation.