2013

Fixed parameter algorithms for compatible and agreement supertree problems

Sudheer Reddy Vakati
Iowa State University

Fixed parameter algorithms for compatible and agreement supertree problems

by

Sudheer Vakati

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:

David Fernández-Baca, Major Professor

Oliver Eulenstein

Srinivas Aluru

Stephen Willson

Xiaoqiu Huang

Iowa State University

Ames, Iowa

2013

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. David Fernández-Baca for his support and guidance for my research work. His nice personality made our weekly meetings very pleasant. My research results would not have been possible without his insights. I thank him for supporting me through NSF grants CCF-1017189 and DEB-0829674.

I would like to thank Prof. Oliver Eulenstein for his guidance and willingness to work with me everytime I asked. I would like to thank my committee members Dr. Srinivas Aluru, Dr. Stephen Willson and Dr. Xiaoqui Huang for their time.

I would like to thank the extremely smart Dr. Sylvain Guillemot with whom I had the pleasure of working with. The later half of my research work would not have been possible without his deep insights.

Lastly, I would like to thank my family, friends and everyone else who helped me at various times in my life.

ABSTRACT

Biologists represent evolutionary history of species through phylogenetic trees. Leaves of a phylogenetic tree represent the species and internal vertices represent the extinct ancestors. Given a collection of input phylogenetic trees, a common problem in computational biology is to build a supertree that captures the evolutionary history of all the species in the input trees, and is consistent with each of the input trees. In this document we study the tree compatibility and agreement supertree problems.

Tree compatibility problem is NP-complete but has been shown to be fixed parameter tractable when parametrized by number of input trees. We characterize the compatible supertree problem in terms of triangulation of a structure called the display graph. We also give an alternative characterization in terms of cuts of the display graph. We show how these characterizations are related to characterization given in terms of triangulation of the edge label intersection graph. We then give a characterization of the agreement supertree problem.

In real world data, consistent supertrees do not always exist. Inconsistencies can be dealt with by contraction of edges or removal of taxa. The agreement supertree edge contraction (AST-EC) problem asks if a collection of k rooted trees can be made to agree by contraction of at most p edges. Similarly, the agreement supertree taxon removal (AST-TR) problem asks if a collection of k rooted trees can be made to agree by removal of at most p taxa. We give fixed parameter algorithms for both cases when parametrized by k and p.

We study the long standing conjecture on the perfect phylogeny problem; there exists a function $f(r)$ such that a given collection $\mathcal{C}$ of r-state characters is compatible if and only if every $f(r)$ subset of $\mathcal{C}$ is compatible. We will show that for $r \geq 2$, $f(r) \geq \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$.

CHAPTER 1. INTRODUCTION

In Chapter 2, we give various definitions and notations. In Chapter 7, we give open problems for further research. The rest of the thesis (Chapters 3-6) can broadly be organized into three parts.

1.1 Chapters 3 & 4

Tree compatibility is a fundamental problem in phylogenetics. Given a collection of phylogenetic trees, the tree compatibility problem asks if there exists a supertree that is consistent with all the input trees. Agreement supertree problem is a more restrictive version of the compatible supertree problem. Both agreement and tree compatibility problems are NP-complete [Steel (1992)] but polynomial time solvable when all the trees in the collection are rooted [Aho et al. (1981); Ng and Wormald (1996)]. Unrooted tree compatibility problem is fixed parameter tractable (FPT) when parametrized by the number of input trees [Bryant and Lagergren (2006)]. Fixed parameter tractability of agreement supertree problem is unknown.

The result in [Bryant and Lagergren (2006)] is derived by transforming the problem into a bounded treewidth graph problem in monadic second order logic and then using the results of Courcelle (1990) and Arnborg et al. (1991). This is done by making using of a structure called the display graph and showing that its treewidth is bounded by the number of input trees. This result does not easily translate into an algorithm. Though an explicit algorithm can be derived using the result, it has huge running constants.

A goal of our research is to derive explicit and practical FPT algorithms for tree compatibility and agreement supertree problems using graph theoretic concepts. In Chapter 1, we derive a characterization of the tree compatibility problem in terms of triangulation of the display

graph. We then study various properties of the display graph. In Chapter 2, we derive an alternative characterization of the tree compatibility problem in terms of cuts of the display graph. We show how this characterization is related to an alternative characterization given in [Gysel et al. (2012)]. We also give a characterization of the agreement supertree problem in terms of cuts of the display graph.

1.2 Chapter 5

Real world data is inconsistent. More often than not, real world collections of input phylogentic trees will not have an agreement supertree. These inconsistent collections can be made consistent by contraction of internal edges or by removal of taxons (leaves) from input trees. We specifically consider the following problems.

Given a collection $\mathcal{T}$ of k rooted trees, can the trees in $\mathcal{T}$ be made to agree by contraction of at most p edges. We call this the agreement supertree edge contraction *AST-EC* problem. Similarly, given a collection $\mathcal{T}$ of k rooted trees, the agreement supertree taxon removal *AST-TR* problem asks if the trees in $\mathcal{T}$ be made to agree by removal of at most p taxons. An FPT algorithm for the taxon removal problem when the trees are all binary is given in Guillemot and Berry (2010). We will give FPT algorithms for both problems when parametrized by k and p.

1.3 Chapter 6

Given a collection of full characters with at most r states, the perfect phylogeny problem asks if the characters in the collection are compatible. Perfect phylogeny problem is NP-complete [Bodlaender et al. (1992); Steel (1992)] but fixed parameter tractable when parametrized by number of states [Agarwala and Fernández-Baca (1994); Dress and Steel (1992); Gusfield (1991); Kannan and Warnow (1994)]. We study a long standing conjecture which says, there exists a function $f(r)$ such that a collection of full characters with at most r states is compatible if and only if every r-subset of the collection is compatible.

For long, the known lower bound on $f(r)$ was r [Meacham (1983)]. Recently, Lam et al.

(2011) showed that $f(3) = 3$. Habib and To [Habib and To (2011)] gave a construction which showed that $f(4) \geq 5$ thus improving on the long known lower bound of r. We will relate quartet compatibility to character compatibility and show that for $r \geq 2$, $f(r) \geq \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$.

CHAPTER 2. PRELIMINARIES

In this section, we review the notions of separators, cuts and triangulations in graphs. We then define tree compatibility, agreement supertrees, and splits compatibility. For every non negative integer m, we denote the set $\{1, \ldots, m\}$ by $[m]$.

2.1 Separators, Cuts, and Triangulations

Let G be a graph. We represent the vertices and edges of G by $V(G)$ and $E(G)$ respectively. Given a vertex v of G, we represent the neighbors of v in G by $N_G(v)$. For any $U \subseteq V(G)$, $G - U$ represents the graph derived by removing vertices of U and their incident edges from G. Similarly, for any $F \subseteq E(G)$, $G - F$ represents the graph with vertex set $V(G)$ and edge set $E(G) \setminus F$. A *clique* of G is a complete subgraph of G. A clique C is *maximal* if there does not exist another clique C' of G where $V(C) \subset V(C')$. We denote the set of all maximal cliques of G by $MC(G)$.

2.1.1 Separators and cuts

For any two nonadjacent vertices a and b of G, an *a-b separator* $U \subseteq V(G)$ is a set of vertices such that a and b are in different connected components of $G - U$. An a-b separator U is *minimal* if for every $U' \subset U$, U' is not an a-b separator. A set $U \subseteq V(G)$ is a *minimal separator* if U is a minimal a-b separator for some nonadjacent vertices a and b of G. A connected component H of $G - U$ is *full* if for every $u \in U$ there exists some vertex $v \in H$ where $\{u, v\} \in E(G)$.

Lemma 1. *[Parra and Scheffler (1997)] For a graph G and any $U \subset V(G)$, U is a minimal separator of G if and only if $G - U$ has at least two full components.*

Two minimal separators U and U' are *parallel* if $G-U$ contains at most one component H where $V(H) \cap U' \neq \emptyset$. We represent the set of all minimal separators of graph G by $\triangle_G$.

Assume that G is connected. A *cut* is a subset of edges of G whose removal disconnects G. A cut F is *minimal* if there does not exist $F' \subset F$ where F' is also a cut of G. Note that if F is minimal, there will exactly be two connected components in $G - F$.

2.1.2 Triangulation, treewidth, and tree decomposition

A *chord* is an edge between two nonadjacent vertices of a cycle. A graph H is *chordal* if and only if every cycle of length four or greater in H has a chord. A chordal graph H is a *triangulation* of graph G if and only if $V(G) = V(H)$ and $E(G) \subseteq E(H)$. The set $E(H) \setminus E(G)$ is called a *fill-in* for G and the edges in it are called *fill-in edges*. We denote the fill-in of G with respect the triangulation H of G by $\xi(G, H)$. Triangulation H of G is a *minimal triangulation* if for every edge $e \in \xi(G, H)$, $H - e$ is not a triangulation of G.

The *width* of triangulation H is defined as the maximum value of $|V(C)| - 1$ over all cliques C of H. The *treewidth* of graph G, denoted by $\text{tw}(G)$, is the smallest width among all possible triangulations of G.

A *tree decomposition* for a graph G is a pair (T, B), where T is a tree and B is a mapping from $V(T)$ to subsets of $V(G)$ that satisfies the following three properties.

(TD1) (*Vertex Coverage*) For every $v \in V(G)$ there is an $x \in V(T)$ such that $v \in B(x)$.

(TD2) (*Edge Coverage*) For every edge $\{u, v\} \in E(G)$ there exists an $x \in V(T)$ such that $\{u, v\} \subseteq B(x)$.

(TD3) (*Coherence*) For every $u \in V(G)$ the set of vertices $\{x \in V(T) : u \in B(x)\}$ forms a subtree of T.

It is well known that if G is chordal, G has a tree-decomposition (T, B) where (i) there is a one-to-one mapping C from the vertices of T to the maximal cliques of G and (ii) for each vertex x in T, $B(x)$ consists precisely of the vertices in the clique $C(x)$ [Heggernes (2005)]. This sort of tree decomposition is called a *clique tree* for G. Conversely, let (T, B) be a tree

decomposition of a graph G and let F be the set of all $\{u, v\} \notin E(G)$ such that $\{u, v\} \subseteq B(x)$ for some $x \in V(T)$. Then, F is a chordal fill-in for G [Heggernes (2005)]. We shall refer to this set F as the *chordal fill-in of G associated with tree-decomposition* (T, B) and to the graph G' obtained by adding the edges of F to G as the *triangulation of G associated with* (T, B).

2.1.3 Relation between minimal separators and minimal triangulations

Let G be a graph and let $\mathcal{F}$ be a collection of subsets of $V(G)$. We represent by $G_{\mathcal{F}}$ the graph derived from G by making the set of vertices of X a clique in G for every $X \in \mathcal{F}$.

Theorem 1. *[Bouchitté and Todinca (2001); Heggernes (2006); Parra and Scheffler (1997)] Let $\mathcal{F}$ be a maximal set of pairwise parallel minimal separators of G and let H be a minimal triangulation of G. Then, the following statements hold.*

1. *$G_{\mathcal{F}}$ is a minimal triangulation of G.*
2. *Let (T, B) be a clique tree of $G_{\mathcal{F}}$. There exists a minimal separator $F \in \mathcal{F}$ if and only if there exist two adjacent vertices x and y in T where $B(x) \cap B(y) = F$.*
3. *$\triangle_H$ is a maximal set of pairwise parallel minimal separators of G and $G_{\triangle_H} = H$.*

2.2 Compatibility and Agreement Supertrees

2.2.1 Phylogenetic trees

An *unrooted phylogenetic tree* T (or just *unrooted tree*) is a tree whose leaves are bijectively mapped to a label set $\mathcal{L}(T)$ and has no vertex of degree two. An unrooted tree is binary if every internal vertex has degree exactly three. A *quartet* is a binary unrooted tree with exactly four leaves. A quartet with with label set $\{a, b, c, d\}$ is denoted $ab|cd$ if the path between the leaves labelled a and b does not have any vertex in common with the path between the leaves labelled c and d.

A *rooted phylogenetic tree* (or just *rooted tree*) is a tree whose leaves are bijectively mapped to a label set $\mathcal{L}(T)$, has a distinguished vertex called the *root*, and no vertex other than the root has degree two. A rooted tree is *binary* if the root vertex has degree two and every other

internal vertex has degree three. A *triplet* is a rooted binary tree with exactly three leaves. A triplet with label set $\{a, b, c\}$ is denoted $ab|c$ if the path between the leaves labeled a and b avoids the path between the leaf labeled c and the root vertex.

Given a tree T, we denote the internal vertices, internal edges and leaf vertices of a tree T by $\mathcal{I}(T)$, $\hat{E}(T)$, $L(T)$ respectively. Let $\mathcal{P}$ be a collection of unrooted (rooted) trees. We call $\mathcal{P}$ a *profile* and denote $\bigcup_{T\in\mathcal{P}} \mathcal{L}(T)$ by $\mathcal{L}(\mathcal{P})$. We will use the words profile and collection interchangeably in this document. A *supertree* for $\mathcal{P}$ is an unrooted (rooted) tree S whose label set is $\mathcal{L}(\mathcal{P})$.

2.2.2 Splits compatibility

A *split* of a label set L is a bipartition of L consisting of non-empty sets. We denote a split $\{X, Y\}$ by $X|Y$. Let T be an unrooted tree. Consider an internal edge e of T. Deletion of e disconnects T into two subtrees T_1 and T_2. If L_1 and L_2 denote the set of all labels in T_1 and T_2, respectively, then $L_1|L_2$ is a split of $\mathcal{L}(T)$. We denote the split corresponding to edge e of T by $\sigma_e(T)$ and we denote by $\Sigma(T)$ the set of all splits corresponding to all internal edges of T.

An unrooted tree T *displays* a split $X|Y$ if there exists an internal edge e of T where $\sigma_e(T) = X|Y$. Then, we also say T is *compatible* with $X|Y$. A set of splits is compatible if there exists an unrooted tree which displays all the splits in the set. Two splits $A_1|A_2$ and $B_1|B_2$ are compatible if and only if at least one of $A_1 \cap B_1$, $A_1 \cap B_2$, $A_2 \cap B_1$ and $A_2 \cap B_2$ is empty [Semple and Steel (2003)]. By the Splits Equivalence Theorem [Buneman (1971); Semple and Steel (2003)], a collection of splits is compatible if and only if every pair is compatible.

Theorem 2. *(Splits-Equivalence Theorem [Buneman (1971); Semple and Steel (2003)]) Let Σ be a collection of splits over a label set L. Then, $\Sigma = \Sigma(S)$ for some unrooted tree S if and only if the splits in Σ are pairwise compatible. Such tree is unique up to isomorphism.*

By the Splits Equivalence Theorem, two phylogenetic trees T_1 and T_2 are *isomorphic* if $\Sigma(T_1) = \Sigma(T_2)$.

2.2.3 Tree compatibility

Let S be an unrooted tree. For any $Y \subseteq \mathcal{L}(S)$, $S[Y]$ denotes minimal subtree of S connecting the leaves in Y. Tree $S|Y$ denotes the tree obtained by suppressing any degree two vertices in $S[Y]$. Let T be an unrooted tree where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$. We say that S *displays* an unrooted tree T if T can be derived from $S|\mathcal{L}(T)$ by contraction of edges. Note that, S displays T if and only if $\Sigma(T) \subseteq \Sigma(S|\mathcal{L}(T))$.

Similarly, let S be a rooted tree. For any $Y \subseteq \mathcal{L}(S)$, $S[Y]$ denotes the tree derived from the minimal subtree of S connecting the labels in Y by distinguishing the vertex closest to the root of S as the root. Tree $S|Y$ denotes the tree derived from $S[Y]$ by suppressing any degree two vertices other than the root. We say that S *displays* a rooted tree T if T can be derived from $S|\mathcal{L}(T)$ by contraction of edges.

Given a profile $\mathcal{P}$ of unrooted (rooted) trees, the *unrooted (rooted) tree compatibility* problem asks if there exists a supertree of $\mathcal{P}$ that displays all the trees in $\mathcal{P}$. If such a supertree S exists, we say that $\mathcal{P}$ is *compatible* and S is a *compatible tree* of $\mathcal{P}$.

2.2.4 Agreement supertrees

Let S and T be two unrooted (rooted) trees where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$. Tree T is an *induced subtree* of S if and only if $S|\mathcal{L}(T) = T$. Let $\mathcal{P}$ be a profile of unrooted (rooted) trees. An unrooted (rooted) supertree S of $\mathcal{P}$ is an *agreement supertree* (or just *AST*) of $\mathcal{P}$ if and only if every input tree of $\mathcal{P}$ is an induced subtree of S. Note that, if S and T are unrooted trees, then T is an induced subtree of S if and only if $\Sigma(T) = \Sigma(S|\mathcal{L}(T))$. If $\mathcal{P}$ has an agreement supertree, we say that the trees in $\mathcal{P}$ *agree*.

2.2.5 Display graphs

Let T be a phylogenetic tree over label set $\mathcal{L}(T)$. Since there exists a bijective function from leaves of T to $\mathcal{L}(T)$, we will represent leaves of T by their labels. Let $\mathcal{P} = \{T_1, T_2, \cdots, T_k\}$ be a profile of k unrooted trees. For any two trees T_i and T_j in $\mathcal{P}$, we assume that the sets of internal vertices of T_i and T_j are disjoint. The *display graph* [Bryant and Lagergren (2006)] of profile $\mathcal{P}$,

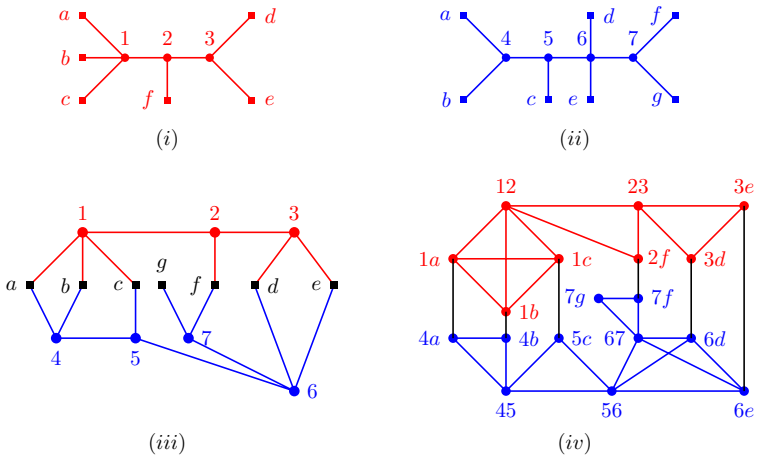


Figure 2.1: (i) First input tree. (ii) Second input tree, which is compatible with the first. (iii) Display graph of the input trees. (iv) Edge label intersection graph of the input trees. For every vertex, uv represents label $\{u, v\}$.

denoted by $G(\mathcal{P})$, is a graph whose vertex set is $\bigcup_{i\in[k]} V(T_i)$ and edge set is $\bigcup_{j\in[k]} E(T_j)$ (see Fig. 2.1). A vertex v of $G(\mathcal{P})$ is a *leaf* if $v \in \mathcal{L}(\mathcal{P})$. Every other vertex of $G(\mathcal{P})$ is an *internal*. An edge of $G(\mathcal{P})$ is *internal* if both its endpoints are internal; otherwise, it is *noninternal*. Let H be a subgraph of $G(\mathcal{P})$. We represent by $\mathcal{L}(H)$, the set of all leaf vertices of H.

2.2.6 Edge label intersection graphs

The *line graph* of a graph G, denoted by $LG(G)$, is a graph whose vertex set is $E(G)$ and two vertices of $LG(G)$ are adjacent if the corresponding edges in G share an endpoint. For rest of the paper we denote the line graph $LG(G(\mathcal{P}))$ of $G(\mathcal{P})$ by $LG(\mathcal{P})$. Graph $LG(\mathcal{P})$ is the modified edge label intersection graph defined in [Gysel et al. (2012)]. Note that if $G(\mathcal{P})$ is connected, then so is $LG(\mathcal{P})$. The line graph of a display graph can be seen in Figure 2.1.

2.3 Character Compatibility

A *character* on a label set L is a partition χ of a subset $L_\chi \subseteq L$; each subset in χ is called a *state*. If $L_\chi = L$, we call χ a *full character*. Otherwise χ is a *partial character*. Note that, a split of a label set L is a full character with exactly two states. A character with at most

r states is called a r-state character. A character χ is *convex* on a phylogenetic tree T if, for every $\{A, B\} \subseteq \chi$, subtrees $T[A]$ and $T[B]$ have no vertex in common.

Let $\mathcal{C} = \{\chi_1, \chi_2, \ldots, \chi_k\}$ be a collection of characters on label set L. We say $\mathcal{C}$ is *compatible*, if there exists a phylogenetic tree S on L where, every $\chi \in \mathcal{C}$ is convex on S. Given a collection $\mathcal{C}$ of full characters, the *perfect phylogeny problem* asks if $\mathcal{C}$ is compatible. The *partition intersection graph* of $\mathcal{C}$ is the graph $\text{Int}(\mathcal{C})$ where $V(\text{Int}(\mathcal{C})) = \{(\chi, A) : \chi \in \mathcal{C}, A \in \chi\}$ and $\{(\chi_i, A), (\chi_j, B)\} \in E(\text{Int}(\mathcal{C}))$ if and only if $A \cap B \neq \emptyset$. A triangulation of $\text{Int}(\mathcal{C})$ is *legal* if and only if there is no fill-in edge of type $\{(\chi, A), (\chi, B)\}$ for any $\chi \in \mathcal{C}$. The following result is well known.

Theorem 3. *[Buneman (1974)] A collection of characters $\mathcal{C}$ is compatible if and only if* $\text{Int}(\mathcal{C})$ *has a legal triangulation.*

Corollary 1. *Let C be a collection of two characters. Then C is compatible if and only if $G(C)$ is acyclic.*

There is a close connection between character compatibility and tree compatibility. Let $\mathcal{P}$ be a profile of unrooted trees. The *character representation* of $\mathcal{P}$ is the set of characters $\mathcal{C}_\mathcal{P} = \bigcup_{T \in \mathcal{P}} \Sigma(T)$. It is straightforward to see that a set of trees $\mathcal{P}$ is compatible if and only if its character representation is compatible.

CHAPTER 3. GRAPH TRIANGULATIONS AND THE COMPATIBILITY OF UNROOTED PHYLOGENETIC TREES

Sudheer Vakati, David Fernández-Baca

Modified from a paper published in the journal *Appl. Math. Lett.*

Abstract

We characterize the compatibility of a collection of unrooted phylogenetic trees as a question of determining whether a graph derived from these trees — the display graph — has a specific kind of triangulation, which we call legal. Our result is a counterpart to the well known triangulation-based characterization of the compatibility of undirected multi-state characters. We then derive a more compact characterization using a modified version of the display graph. The modified display graph is a structure of interest in its own right. We study the relation between unrooted tree compatibility and the treewidth of the modified display graphs.

A collection of unrooted trees is compatible if there is no contradiction among them, and there is a tree that represents all the evolutionary relationships among species present in them. While compatibility cannot be guaranteed, it is always possible to make an incompatible collection of trees compatible by either contracting certain edges in the input trees or by eliminating some subset of the input trees. We show that the problem of finding such sets of edges or trees to contract or remove is closely related to the triangulation-based characterization of the modified display graph.

3.1 Introduction

Given a profile $\mathcal{P} = \{T_1, T_2, \cdots, T_k\}$ of phylogenetic trees, the *phylogenetic tree compatibility problem* asks whether or not $\mathcal{P}$ is compatible. This question arises when trying to assemble a

collection of phylogenies for different sets of species into a single phylogeny (a supertree) for all the species [Gordon (1986)]. The phylogenetic tree compatibility problem asks whether or not it is possible to do so via a supertree that displays each of the input trees. In this chapter unless otherwise mentioned we only consider unrooted phylogenetic trees.

Phylogenetic tree compatibility is NP-complete [Steel (1992)] (but the problem is polynomially-solvable for rooted trees [Aho et al. (1981)]. Nevertheless, Bryant and Lagergren have shown that the problem is fixed-parameter tractable for fixed k [Bryant and Lagergren (2006)]. Their argument relies on a partial characterization of compatibility in terms of tree-decompositions and tree-width of the display graph of a profile. Here we build on their argument to produce a complete characterization of compatibility in terms of the existence of a special kind of triangulation of the display graph. These *legal* triangulations (defined in Section 3.2) only allow certain kinds of edges to be added. Our result is a counterpart to the well-known characterization of character compatibility in terms of triangulations of a class of intersection graphs [Buneman (1974)], which has algorithmic consequences [Gusfield (2009); McMorris et al. (1994)]. Our characterization of tree compatibility may have analogous implications.

Tree compatibility can determined by breaking down input trees into quartets and then determining the compatibility of the smaller trees. Grunewald et al. (2008) provide a way to determine quartet compatibility by building a special graph called the quartet graph. Tree compatibility can also be determined in terms of character compatibility by converting the input trees into characters [Steel (1992)]. These methods have some redundancy, since the quartets or characters that are built from single tree are already known to be compatible. Our characterization in terms of triangulation of display graph provides a more direct way to determine compatibility.

Inconsistencies are common in real world data; hence, collections of incompatible trees are prevalent. These incompatibilities are dealt with in different ways. One is by identifying a subset of trees in the collection whose removal makes the collection compatible. The *tree removal problem* asks whether a collection of phylogenetic trees can be made compatible by removal of at most p trees. An alternative and more fine-grained way to achieve compatibility is to identify a set of internal edges in the input trees whose contraction makes the collection

compatible. The *edge contraction problem* asks if a collection of phylogenetic trees can be made compatible by contracting at most p internal edges. The edge contraction and tree removal problems are NP-hard even for $p = 0$.

This chapter is organized as follows. We characterize unrooted tree compatibility using triangulations of display graph in Section 3.2. In Section 3.3, we modify the display graph to derived a more concise characterization of tree compatibility. In the same section we also study various properties of this modified display graph. We relate treewidth of the modified display graph to tree compatibility in Section 3.4. Lastly, we give characterizations of the edge contraction and tree removal problems in Section 3.5.

3.2 Legal Triangulations and Compatibility

Let $\mathcal{P}$ be a profile of unrooted trees. A triangulation G' of the display graph $G(\mathcal{P})$ is *legal* if it satisfies the following conditions.

(LT1) Suppose a clique in G' contains an internal edge. Then, this clique can contain no other edge from $G(\mathcal{P})$ (internal or non-internal).

(LT2) Fill-in edges can only have internal vertices as their endpoints.

Note that the above conditions rule out a chord between vertices of the same tree. Also, in any legal triangulation of $G(\mathcal{P})$, any clique that contains a non-internal edge cannot contain an internal edge from any tree. See Figure 3.1.

The importance of legal triangulations derives from the next results, which are proved in the next section.

Lemma 2. *Suppose a profile $\mathcal{P} = \{T_1, \ldots, T_k\}$ of unrooted phylogenetic trees is compatible. Then $G(\mathcal{P})$ has a legal triangulation.*

Lemma 3. *Suppose the display graph of a profile $\mathcal{P} = \{T_1, \ldots, T_n\}$ of unrooted trees has a legal triangulation. Then $\mathcal{P}$ is compatible.*

The preceding lemmas immediately imply our main result.

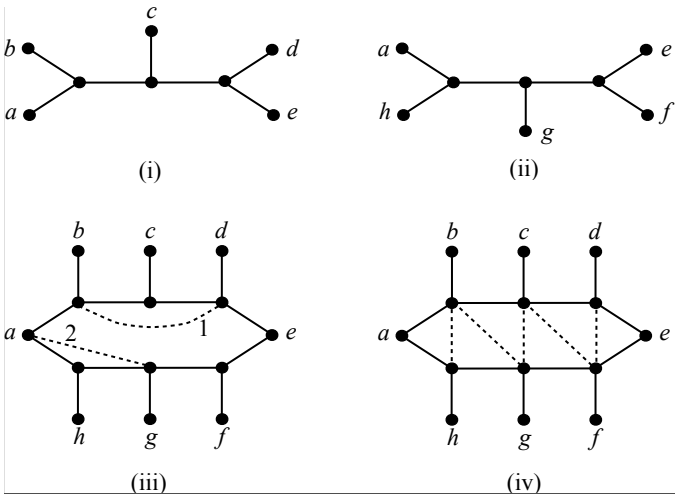


Figure 3.1: (i) First input tree. (ii) Second input tree. (iii) The display graph of the input tress with two fill-in edges, indicated by dashed lines. Edge 1 cannot appear in a legal triangulation, since the result would violate (LT1). Edge 2 is not allowed, because it would result in a violation of (LT2). (iv) The display graph with a legal triangulation, indicated by dashed lines.

Theorem 4. *A profile $\mathcal{P} = \{T_1, \ldots, T_k\}$ of unrooted trees is compatible if and only if $G(\mathcal{P})$ has a legal triangulation.*

3.2.1 Proofs

The proofs of Lemmas 2 and 3 rely on a new concept. Suppose T_1 and T_2 are phylogenetic trees such that $\mathcal{L}(T_2) \subseteq \mathcal{L}(T_1)$. An *embedding function from T_1 to T_2* is a surjective map ϕ from a subgraph of T_1 to T_2 satisfying the following properties.

(EF1) For every $\ell \in \mathcal{L}(T_2)$, ϕ maps the leaf ℓ in T_1 to the leaf ℓ in T_2.

(EF2) For every vertex v of T_2 the set $\phi^{-1}(v)$ is a connected subgraph of T_1.

(EF3) For every edge $\{u, v\}$ of T_2 there is a unique edge $\{u', v'\}$ in T_1 such that $\phi(u') = u$ and $\phi(v') = v$.

The next result extends Lemma 1 of [Bryant and Lagergren (2006)].

Lemma 4. *Let T_1 and T_2 be phylogenetic trees and $\mathcal{L}(T_2) \subseteq \mathcal{L}(T_1)$. Tree T_1 displays tree T_2 if and only if there exists an embedding function ϕ from T_1 to T_2.*

Proof. The "only if" part was already observed by Bryant and Lagergren (see Lemma 1 of [Bryant and Lagergren (2006)]). We now prove the other direction.

To prove that T_1 displays T_2, we argue that T_2 can be obtained from $T_1|\mathcal{L}(T_2)$ by a series of edge contractions, which are determined by the embedding function ϕ from T_1 to T_2. Let T_1' be the graph obtained from $T_1|\mathcal{L}(T_2)$ by considering each vertex v of T_2 and identifying all vertices of $\phi^{-1}(v)$ in $T_1|\mathcal{L}(T_2)$ to obtain a single vertex u' with $\phi(u') = v$. Property (EF2) ensures that, each such operation is well defined and yields a tree. By properties (EF1)–(EF3), each vertex v of $T_1|\mathcal{L}(T_2)$ is in the domain of ϕ. Thus, function ϕ is now a bijection between T_2 and T_1' that satisfies (EF1)–(EF3). We now prove that T_1' is isomorphic to T_2. It then follows from property (EF1) that T_1 displays T_2.

We claim that for any two vertices $u, v \in V(T_2)$, there is an edge $\{u, v\} \in E(T_2)$ if and only if there is an edge $\{\phi^{-1}(u), \phi^{-1}(v)\} \in E(T_1')$. The "only if" part follows from property (EF3). For the other direction, assume by way of contradiction that $\{x, y\} \notin E(T_2)$, but that $\{\phi^{-1}(x), \phi^{-1}(y)\} \in E(T_1')$. Let P be the path between vertices x and y in T_2. By property (EF3), there is a path between nodes $\phi^{-1}(x)$, $\phi^{-1}(y)$ in tree T_1' that does not include the edge $\{\phi^{-1}(x), \phi^{-1}(y)\}$. This path along with the edge $\{\phi^{-1}(x), \phi^{-1}(y)\}$ forms a cycle in T_1', which gives the desired contradiction. Thus, the bijection ϕ between T_2 and T_1' is an isomorphism between the two trees. □

The preceding lemma immediately implies the following characterization of compatibility.

Lemma 5. *Profile $\mathcal{P} = \{T_1, \ldots, T_k\}$ is compatible if and only if there exist a supertree S for $\mathcal{P}$ and functions $\phi_1, \ldots, \phi_k$, where, for $i = 1, \ldots, k$, ϕ_i is an embedding function from S to T_i.*

Proof of Lemma 2. If $\mathcal{P}$ is compatible, there exists a supertree for $\mathcal{P}$ that displays T_i for $i = 1, \ldots, k$. Let S be any such supertree. By Lemma 5, for $i = 1, \ldots, k$, there exists an embedding function ϕ_i from S to T_i. We will use S and the ϕ_is to build a tree decomposition (T, B) corresponding to a legal triangulation of $G(\mathcal{P})$. The construction closely follows that given by Bryant and Lagergren in their proof of Theorem 1 of [Bryant and Lagergren (2006)]; thus, we only summarize the main ideas.

Initially we set $T = S$ and, for every $v \in V(T)$, $B(v) = \{\phi_i(v) : v \text{ in the domain of } \phi_i; 1 \leq i \leq k\}$. Now, (T, B) satisfies the vertex coverage property and the coherence property, but not edge coverage [Bryant and Lagergren (2006)]. To obtain a pair (T, B) that satisfies all three properties, subdivide the edges of T and extend B to the new vertices. Do the following for each edge $\{x, y\}$ of T. Let $F = \{\{u_1, v_1\}, \ldots, \{u_m, v_m\}\}$ be set of edges of $G(\mathcal{P})$ such that $u_i \in B(x)$ and $v_i \in B(y)$. Observe that F contains at most one edge from T_i, for $i = 1, \ldots, k$ (thus, $m \leq k$). Replace edge $\{x, y\}$ by a path $x, z_1, \ldots, z_m, y$, where $z_1, \ldots, z_m$ are new vertices. For $i = 1, 2, \ldots, m$, let $B(z_i) = (B(x) \cap B(y)) \cup \{v_1, \ldots, v_i, u_i, \ldots, u_m\}$. The resulting pair (T, B) can be shown to be a tree decomposition of $G(\mathcal{P})$ of width k (see [Bryant and Lagergren (2006)]).

The preceding construction guarantees that (T, B) satisfies two additional properties:

(i) For any $x \in V(T)$, if $B(x)$ contains both endpoints of an internal edge of T_i, for some i, then $B(x)$ cannot contain both endpoints of any other edge, internal or not.

(ii) Let $x \in V(T)$ be such that $B(x)$ contains a labeled vertex $v \in V(G(\mathcal{P}))$. Then, for every $u \in B(x) \setminus \{v\}$, $\{v, u\} \in E(G(\mathcal{P}))$.

Properties (i) and (ii) imply that the triangulation of $G(\mathcal{P})$ associated with (T, B) is legal. □

Next, we prove Lemma 3. For this, we need some definitions and auxiliary results. Assume that $G(\mathcal{P})$ has a legal triangulation H. Let (T, B) be a clique tree for H. For each vertex $v \in V(G(\mathcal{P}))$, let $C(v)$ denote the set of all nodes in the clique tree T that contain v. Observe that the coherence property implies that $C(v)$ induces a subtree of T.

Lemma 6. *Suppose vertex v is a leaf in tree T_i, for some $i \in \{1, \ldots, k\}$. Let $U(v) = \bigcup_{x \in C(v)} B(x)$. Then, for any $j \in \{1, \ldots, k\}$, at most one internal vertex u from input tree T_j is present in $U(v)$. Furthermore, for any such a vertex u we must have that $\{u, v\} \in E(G(\mathcal{P}))$.*

Proof. Follows from condition (LT2). □

Lemma 7. *Suppose $e = \{u, v\}$ is an internal edge from input tree T_i, for some $i \in \{1, \ldots, k\}$. Let $U(e) = \bigcup_{x \in C(u) \cap C(v)} B(x)$. Then,*

(i) $U(e)$ *contains at most one vertex of* T_j*, for any* $j \in \{1, \ldots, k\}$*,* $j \neq i$*, and*

(ii) $V(T_i) \cap U(e) = \{u, v\}$.

Proof. Part (ii) follows from condition (LT1). We now prove part (i).

Assume by way of contradiction that the claim is false. Then, there exists a $j \neq i$ and an edge $\{x, y\} \in T$ such that $e \subseteq B(x)$, $e \subseteq B(y)$, and there are vertices $a, b \in V(T_j)$, $a \neq b$, such that $a \in B(x)$ and $b \in B(y)$.

Deletion of edge $\{x, y\}$ partitions $V(T)$ into two sets X and Y. Let $P = \{a \in V(T_j) : a \in B(z) \text{ for some } z \in X\}$ and $Q = \{b \in V(T_j) : b \in B(z) \text{ for some } z \in Y\}$. By the coherence property, (P, Q) is a partition of $V(T_j)$. There must be a vertex p in set P and a vertex q in set Q such that $\{p, q\} \in E(T_j)$. Since H is a legal triangulation, there must be a node z in T such that $p, q \in B(z)$. Irrespective of whether z is in set X or Y, the coherence property is violated, a contradiction. □

A legal triangulation of the display graph of a profile is *concise* if

(C1) each internal edge is contained in exactly one maximal clique in the triangulation and

(C2) every vertex that is a leaf in some tree is contained in exactly one maximal clique of the triangulation.

Lemma 8. *Let* $\mathcal{P}$ *be a profile. If* $G(\mathcal{P})$ *has a legal triangulation, then* $G(\mathcal{P})$ *has a concise legal triangulation.*

Proof. Let H be a legal triangulation of $G(\mathcal{P})$ that is not concise. Let (T, B) be a clique tree for H. We will build a concise legal triangulation for $G(\mathcal{P})$ by repeatedly applying contraction operations on (T, B). The *contraction* of an edge $e = \{x, y\}$ in T is the operation that consists of (i) replacing x and y by a single (new) node z, (ii) adding edges from node z to every neighbor of x and y, and (iii) making $B(z) = B(x) \cup B(y)$. Note that the resulting pair (T', B') is a tree decomposition for $G(\mathcal{P})$ (and H); however, it is not guaranteed to be a clique tree for H.

We proceed in two steps. First, for every leaf v of $G(\mathcal{P})$ such that $|C(v)| > 1$, contract each edge $e = \{x, y\}$ in T such that $x, y \in C(v)$. In the second step, we consider each edge $e = \{u, v\}$

of $G(\mathcal{P})$ such that $|C(u) \cap C(v)| > 1$, contract each edge $\{x, y\}$ in T such that $x, y \in C(u) \cap C(v)$. Lemma 6 (respectively, Lemma 7) ensures that each contraction done in the first (respectively, second) step leaves us with a new tree decomposition whose associated triangulation is legal. Furthermore, the triangulation associated with the final tree decomposition is concise. □

Proof of Lemma 3. We will show that, given a legal triangulation H of $G(\mathcal{P})$, we can generate a supertree S for $\mathcal{P}$ along with an embedding function ϕ_i from S to T_i, for $i = 1, \ldots, k$. By Lemma 5, this immediately implies that $\mathcal{P}$ is compatible

By Lemma 8, we can assume that H is concise. Let (T, B) be a clique tree for H. Initially, we make $S = T$. Next, for each node x of S, we consider three possibilities:

Case 1: *$B(x)$ contains a labeled vertex ℓ of $G(\mathcal{P})$.* Then, ℓ is a leaf in some input tree T_i; further, by conciseness, x is the unique node in S such that $v \in B(x)$, and, by the edge coverage property, if u is the neighbor of ℓ in T_i, $u \in B(x)$. Now, do the following.

(i) Add a new node ℓ and a new edge $\{x, \ell\}$ to S.

(ii) For each $i \in \{1, \ldots, k\}$ such that ℓ is a leaf in T_i, make $\phi_i(\ell) = \ell$ and $\phi_i(x) = u$, where u is the neighbor of v in T_i.

Case 2: *$B(x)$ contains both endpoints of an internal edge $e = \{u, v\}$ of some input tree T_i.* By legality, $B(x)$ does not contain both endpoints of any other edge of any input tree, and, by conciseness, x is the only node of S that contains both endpoints of e. Now, do the following.

(i) Replace node x with nodes x_u and x_v, and add edge $\{x_u, x_v\}$.

(ii) Add an edge between node x_u and every node neighbor y of x such that $u \in B(y)$.

(iii) Add an edge between node x_v and every neighbor y of x such that $v \in B(y)$.

(iv) For each neighbor y of x such that $u \notin B(y)$ and $v \notin B(y)$, add an edge from y to node x_u or node x_v, but not to both (the choice of which edge to add is arbitrary).

(v) For every $j \in \{1, \ldots, k\}$, $(j \neq i)$ such that $B(x) \cap V(T_j) \neq \emptyset$, make $\phi_j(x_u) = \phi_j(x_v) = z$ where, z is the vertex of T_j contained in $B(x)$. Also, make $\phi_i(x_u) = u$ and $\phi_i(x_v) = v$.

Case 3: *$B(x)$ contains at most one internal vertex from T_i for $i \in \{1, \ldots, k\}$.* Then, for every i such that $B(x) \cap V(T_i) \neq \emptyset$ make $\phi_i(x) = v$, where v is the vertex of T_i contained in $B(x)$.

By construction (Case 1) and the legality and conciseness of (T, B), for every $\ell \in \bigcup_{i=1}^{k} \mathcal{L}(T_i)$ there is exactly one leaf $x \in V(S)$ that is labeled ℓ. Thus, S is a supertree of profile $\mathcal{P}$. Property (TD1) also ensures that the function ϕ_i is a surjective map from a subgraph of S to T_i. Furthermore, the handling of Case 1 guarantees that ϕ_i satisfies (EF1). The coherence of (T, B) and the handling of all cases ensures that ϕ_i satisfies (EF2). The handling of Case 2 and conciseness ensure that ϕ_i satisfies (EF3). Thus, ϕ_i is an embedding function, and, by Lemma 5, profile $\mathcal{P}$ is compatible. □

3.3 Modified Display Graph

Let T be a phylogenetic tree. We define a function θ from $\mathcal{L}(T)$ to $\mathcal{I}(T)$, where $\theta(\ell) = v$ if and only if $\{v, \ell\} \in E(T)$. We call θ the *label mapping function* of T.

Let $\mathcal{P} = \{T_1, T_2, \ldots, \mathcal{T}_k\}$ be a collection of unrooted trees and for every $i \in [k]$ let θ_i be the label mapping function of T_i. The *modified display graph* of $\mathcal{P}$, denoted by $\mathcal{G}(\mathcal{P})$, is the graph whose vertex set is $\bigcup_{i \in [k]} \mathcal{I}(T_i)$ and there is an edge between vertices u and v if and only if one of the following conditions hold.

1. $\{u, v\} \in E(T_i)$ for some $i \in [k]$
2. $\theta_i^{-1}(u) \cap \theta_j^{-1}(v) \neq \emptyset$ where, $1 \leq i \neq j \leq k$, $u \in V(T_i)$ and $v \in V(T_j)$

An edge in $E(\mathcal{G}(\mathcal{P}))$ is a *tree edge* if it is an edge in some input tree. Any edge which is not a tree edge is called an *added edge*. Note that every vertex in $\mathcal{G}(\mathcal{P})$ is an internal vertex in some input tree and every tree edge of $\mathcal{G}(\mathcal{P})$ is an internal edge of some input tree. We represent the

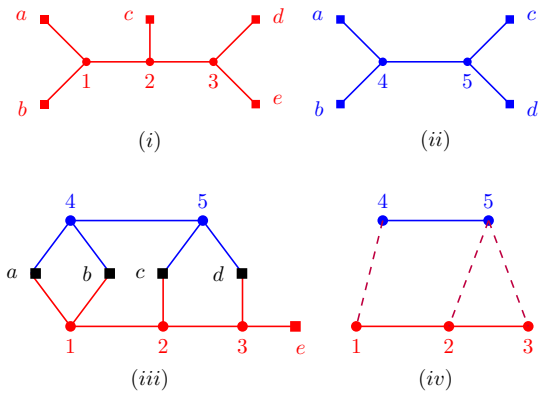


Figure 3.2: (i) First input tree. (ii) Second input tree. (iii) Display graph of the input trees. (iv) Modified display graph of the input trees. Solid lines represent the tree edges and the dashed lines represent the added edges.

tree edges of $\mathcal{G}(\mathcal{P})$ by $\hat{E}(\mathcal{G}(\mathcal{P}))$. A vertex of $\mathcal{G}(\mathcal{P})$ is a *labelled vertex* if it has an added edge incident on it. Display graphs and modified display graphs are illustrated in Figure 3.2.

A modified display graph of $\mathcal{P}$ can be alternatively be defined as follows. For every leaf ℓ in $G(\mathcal{P})$, make all its neighbors a clique and delete ℓ. The resulting graph is the modified display graph of $\mathcal{P}$.

3.3.1 Compatibility with modified display graphs

A triangulation H of $\mathcal{G}(\mathcal{P})$ is *legal* if there no clique in H which contains more than one tree edge. The following lemma characterizes unrooted tree compatibility in terms of triangulation of modified display graphs.

Lemma 9. *Let $\mathcal{P}$ be a profile of unrooted trees. Then, $G(\mathcal{P})$ has a legal triangulation if and only if $\mathcal{G}(\mathcal{P})$ has a legal triangulation.*

Proof. Let G' be a legal triangulation of $\mathcal{G}(\mathcal{P})$. For every leaf $\ell \in G(\mathcal{P})$, make the neighbors of ℓ a clique. Let H' be the modified graph. There is a chordless cycle C in H' if and only if $\mathcal{G}(\mathcal{P})$ also has chordless cycle C. Add the fill-in edges of $\xi(\mathcal{G}(\mathcal{P}), G')$ to H'. The resulting graph H'' is a triangulation of $G(\mathcal{P})$. Since G' is a legal triangulation of $\mathcal{G}(\mathcal{P})$ and there is no chord in H'' with a leaf vertex as an endpoint, H'' is a legal triangulation of $G(\mathcal{P})$.

Let H' be a legal triangulation of $G(\mathcal{P})$. Consider any chordless cycle C in $G(\mathcal{P})$ which has a leaf vertex ℓ. To triangulate C, there should be a fill in edge with either ℓ as an endpoint or there should be an edge with the neighbors of ℓ as endpoints. Since the former gives an illegal fill-in, any legal fill-in of C, will have a fill-in edge with neighbors of ℓ as endpoints. Consider any two neighbors v_1 and v_2 of a leaf vertex ℓ in H' which do not have a fill-in edge between them. Adding a fill in edge between v_1 and v_2 will not create any new chordless cycles. Let F represent the set of edges required to make the neighbors of any leaf in $G(\mathcal{P})$ a clique. Let F' represent the set of fill-in edges in H' with both endpoints as neighbors of some leaf vertex. Add the edges of $F \setminus F'$ to H' and let H'' be the resulting graph. Graph H'' is triangulated. Delete the leaf vertices from H''. The resulting graph is a triangulation of $\mathcal{G}(\mathcal{P})$ and is legal. □

A legal triangulation G' of $\mathcal{G}(\mathcal{P})$ is *concise*, if for every tree edge in $\mathcal{G}(\mathcal{P})$ there is exactly one maximal clique in G' which contains both its endpoints. By a slight modification of the proof of Lemma 8, we can conclude the following.

Lemma 10. *If a modified display graph has a legal triangulation, it also has a concise legal triangulation.*

Given a modified display graph $\mathcal{G}(\mathcal{P})$ of a collection of phylogenetic trees $\mathcal{P}$, the only information required to determine the compatibility of the collection is the partitioning of the edges in $\mathcal{G}(\mathcal{P})$ into tree and added edges. Note that, the added edges of $\mathcal{G}(\mathcal{P})$ capture the dependencies between the labels of various trees. Given two collections $\mathcal{P}_1$ and $\mathcal{P}_2$ of phylogenetic trees, if collection $\mathcal{P}_1$ is compatible if and only if collection $\mathcal{P}_2$ is compatible, we say $\mathcal{P}_1$ and $\mathcal{P}_2$ are *equivalent*.

Let $\mathcal{P}$ be a profile of k unrooted trees. The *added edge clique set* of $\mathcal{G}(\mathcal{P})$, denoted $AC(\mathcal{P})$, is the set of all maximal cliques in $\mathcal{G}(\mathcal{P})$ where, there is no clique in $AC(\mathcal{P})$ that contains two vertices from the same tree in $\mathcal{P}$ and, for every added edge in $\mathcal{G}(\mathcal{P})$, there is at least one clique in $AC(\mathcal{P})$ that contains both its end points.

We apply the following transformation to the trees of $\mathcal{P}$.

1. For every $i \in [k]$ let T_i' be the tree where $V(T_i') = \mathcal{I}(T_i)$ and every internal edge of T_i is also an edge of T_i'.

2. For every clique C in $AC(\mathcal{P})$ we create a new label ℓ and do the following. If v is a vertex in C and $v \in V(T_i)$ for some $i \in [k]$, add a new vertex ℓ to $V(T'_i)$ and add the edge $\{v, \ell\}$ to $E(T'_i)$

3. For every $i \in [k]$, suppress degree two vertices in T'_i.

Let $\mathcal{P}'$ represent the profile $\{T'_1, T'_2, \ldots, T'_k\}$. Note that, by the way of construction, every tree in $\mathcal{P}'$ is a phylogenetic tree. We call $\mathcal{P}'$ the reduced profile of $\mathcal{P}$. We can prove the following results.

Lemma 11. *Profiles $\mathcal{P}$ and $\mathcal{P}'$ are equivalent.*

Proof. After step 2 in the transformation, for every $i \in [k]$, do the following. For every degree 2 vertex v in tree T'_i, add a new label ℓ_v to $\mathcal{L}(T'_i)$ and add the edge $\{v, \ell_v\}$ to $E(T'_i)$. Let T''_i be the resulting tree and let $\mathcal{P}''$ represent the profile $\{T''_1, T''_2, \ldots, T''_k\}$. We prove the lemma, by first showing that, profiles $\mathcal{P}$ and $\mathcal{P}''$ are equivalent. We then show that profiles $\mathcal{P}'$ and $\mathcal{P}''$ are equivalent, thus proving the lemma.

For every $i \in [k]$, let θ_i, θ''_i represent the label mapping functions of T_i, T''_i respectively. By construction, the vertices and tree edges of both $\mathcal{G}(\mathcal{P})$ and $\mathcal{G}(\mathcal{P}'')$ are the same. We now show that the graphs $\mathcal{G}(\mathcal{P})$ and $\mathcal{G}(\mathcal{P}'')$ are the same by showing that the added edges in $\mathcal{G}(\mathcal{P})$ and $\mathcal{G}(\mathcal{P}'')$ are the same. Consider an added edge $\{u, v\}$ in $\mathcal{G}(\mathcal{P})$ where $u \in V(T_i)$ and $v \in V(T_j)$ for some $1 \leq i \neq j \leq k$. Then, there exists a clique C in $AC(\mathcal{P})$ which contains both u and v and thus, there will exist a label ℓ in $\mathcal{L}(\mathcal{P}'')$ where $\theta''_i(\ell) = u$ and $\theta''_j(\ell) = v$. Hence, there will exist an edge $\{u, v\}$ in $\mathcal{G}(\mathcal{P}'')$.

Now assume that there exists an added edge $\{u, v\}$ in $\mathcal{G}(\mathcal{P}'')$ where $u \in V(T''_i)$ and $v \in V(T''_j)$ for some $1 \leq i \neq j \leq k$. Thus there exists an ℓ in $\mathcal{L}(\mathcal{P}'')$ where, $\theta''_i(\ell) = u$ and $\theta''_j(\ell) = v$, and hence, there exists a clique C in $AC(\mathcal{P})$ which contains both u and v. The graph $\mathcal{G}(\mathcal{P})$ thus contains the added edge $\{u, v\}$. Since the vertices and edges of $\mathcal{G}(\mathcal{P})$ and $\mathcal{G}(\mathcal{P}'')$ are the same, graphs $\mathcal{G}(\mathcal{P})$ and $\mathcal{G}(\mathcal{P}'')$ are the same. Profiles $\mathcal{P}$ and $\mathcal{P}''$ are compatible if and only if $\mathcal{G}(\mathcal{P})$ has a legal triangulation. Thus, profiles $\mathcal{P}$ and $\mathcal{P}''$ are equivalent.

For every $i \in [k]$, tree T'_i can be derived from T''_i by deletion of zero or more leaf vertices and suppression of degree two vertices. Any supertree which displays T''_i also displays T'_i. Thus, if

$\mathcal{P}''$ is compatible then $\mathcal{P}'$ is compatible. Similarly, for every $i \in [k]$, T_i'' can be derived from T_i' by repeatedly breaking the edges and adding leaf vertices labelled by labels in $\mathcal{L}(T_i'') \setminus \mathcal{L}(T_i')$. If there exists a supertree S that displays tree T_i', then S can be modified to display tree T_i'' by repeated breaking of edges and addition of labelled leaf vertices. This addition of these leaf vertices would not have any effect on other input trees being displayed since the labels being added are unique to a single tree. Thus, if $\mathcal{P}'$ is compatible, then $\mathcal{P}''$ is compatible. Profiles $\mathcal{P}'$ and $\mathcal{P}''$ are thus equivalent. □

Corollary 2. *If $AC(\mathcal{P})$ contains a k-clique, the compatibility of $\mathcal{P}$ can be determined in polynomial time.*

Proof. Consider the reduced profile $\mathcal{P}'$ of $\mathcal{P}$. Since $AC(\mathcal{P})$ contains a k-clique, there exists a label $\ell \in \mathcal{L}(\mathcal{P}')$ where, for every tree $T \in \mathcal{P}'$, $\ell \in \mathcal{L}(T)$. Convert every tree $T \in \mathcal{P}'$ into a rooted tree as follows. Let v be the vertex adjacent to leaf vertex ℓ in T. Make v the root of T and delete ℓ from T. Let $\mathcal{R}$ represent the profile of resulting rooted phylogenetic trees. Profiles $\mathcal{R}$ and $\mathcal{P}'$ are equivalent [Steel (1992)]. By equivalence of profiles $\mathcal{P}$ and $\mathcal{P}'$ from Lemma 11, profiles $\mathcal{P}$ and $\mathcal{R}$ are equivalent. There exists a polynomial time algorithm to determine the compatibility of rooted trees [Aho et al. (1981)] and thus compatibility of $\mathcal{P}$ can be determined in polynomial time. □

The above lemmas provide the advantages of determining unrooted phylogenetic tree compatibility by triangulation of modified display graphs. By lemma 11, any profile $\mathcal{P}$ of phylogenetic trees can be reduced to an equivalent profile $\mathcal{P}'$ where, the number of labels of $\mathcal{P}'$ is equal to the number of elements of the added edge clique set of $\mathcal{G}(\mathcal{P})$. If there exists a label $\ell \in \mathcal{L}(\mathcal{P})$ where every tree in $\mathcal{P}$ has a leaf labelled ℓ, compatibility of $\mathcal{P}$ can be determined in polynomial time [Steel (1992)]. Corollary 2 identifies special cases of unrooted phylogenetic tree profiles whose compatibility can be determined in polynomial time, though there is no common label which is in every input tree in the profile. An example of such case is given in Figure 3.3.

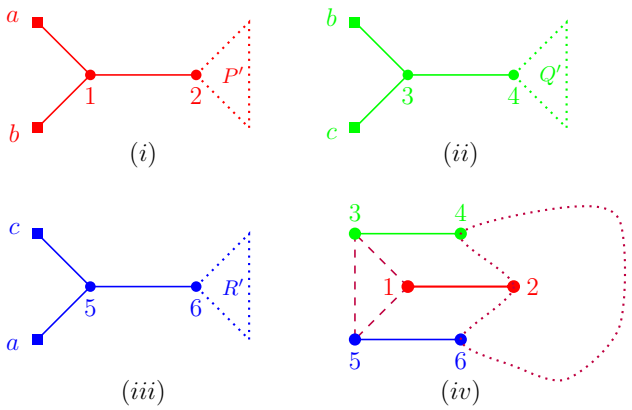


Figure 3.3: (i), (ii), and (iii) : Input trees P,Q, and R, where $\mathcal{L}(P) \cap \mathcal{L}(Q) \cap \mathcal{L}(R) = \emptyset$. (iv) Modified display graph of P, Q, and R. Vertices 1, 3, and 5 form a 3-clique. Hence the compatibility of these input trees can be determined in polynomial time.

3.3.2 Modified minor operation

Let G and G' be two modified display graphs of some profiles. Display graph G' is a *modified minor* of display graph G, if it can be derived from G by zero or more of the following operations: (a) Deletion of an added edge. (b) Contraction of a tree edge. (c) Deletion of vertices $U \subseteq V(G)$ where, (i) the subgraph $G[U]$ of G induced by U is connected, (ii) every edge in $G[U]$ is a tree edge and (iii) There is no tree edge $\{u, v\} \in E(G)$ where $u \in U$ and $v \in \{V(G) \setminus U\}$.

Theorem 5. *If there exists a legal triangulation for a modified display graph G, then there exists a legal triangulation for any modified minor of G.*

Proof. Let G' be a modified minor of G. We use induction on m, the number of modified minor operations required to derive G' from G. When $m = 0$ both G and G' are the same and hence the theorem holds. Let G'' be the modified minor of G after the first $m - 1$ operations. By induction hypothesis, assume that there exists a legal triangulation H of G''. We now have the following cases.

Edge Deletion: Suppose the mth operation is the deletion of an added edge e from G''. The fill-in $\xi(G'', H) \cup e$ is a legal fill-in of G'

Edge Contraction: Suppose the mth operation contracts a tree edge $e = \{u, v\}$. Let $w = V(G') \setminus V(G'')$ and let (T, B) be the clique tree of H. For every vertex $x \in V(T)$ where $B(x)$ contains u or v, set $B(x) = (B(x) \setminus \{u, v\}) \cup \{w\}$. The resulting pair (T, B') is a valid tree decomposition of G'. Since every clique in H is legal, there is no vertex $x \in V(T)$ where $B'(x)$ contains more than one tree edge. Thus the triangulation corresponding to (T, B') is a legal triangulation of G'.

Vertex Deletion: Suppose the mth operation deletes vertices in U. Delete the vertices of U in H and let H' be the resulting graph. Since H is legally triangulated, any induced subgraph of H is also legally triangulated. Thus, H' is a legal triangulation of G'.

□

Intuitively, contraction of a tree edge in a display graph G corresponds to contraction of an internal edge in some input tree. Similarly, deletion of a vertices in U in operation 3 correspond to deleting an input tree from the collection. In both these cases, if $\mathcal{P}$ is compatible, the resulting collection is also compatible. Deletion of an added edge e from the display graph G corresponds to renaming of certain labels such that if $\mathcal{P}$ is compatible, the resulting collection also compatible and has a display graph same as G except for edge e. The significance of modified minors can be seen from the following lemma.

Lemma 12. *Let $\mathcal{P} = \{T_1, T_2\}$. Then, $\mathcal{P}$ is incompatible if and only if $\mathcal{G}(\mathcal{P})$ has a K_4 with two non-adjacent tree edges as a modified minor.*

Proof. The "if" part follows from Theorem 5. To prove the "only if" part, assume that, $\mathcal{P}$ is incompatible. Then, there will exist four labels $\{a, b, c, d\} \subseteq \{\mathcal{L}(T_1) \cap \mathcal{L}(T_2)\}$ where, the quartets induced by the vertices a, b, c, d in T_1 and T_2 are incompatible.

Without loss of generality, let $ab|cd$, $ac|bd$ be the quartets induced by vertices a, b, c, d in T_1 and T_2 respectively. For every $i \in [2]$, let θ_i be the label mapping function of tree T_i. Let P be the path of tree edges between vertices $\theta_1(a)$ and $\theta_1(b)$ in $\mathcal{G}(\mathcal{P})$. Similarly, let Q be the path of tree edges between vertices $\theta_1(c)$ and $\theta_1(d)$ in $\mathcal{G}(\mathcal{P})$. Paths P and Q do not intersect

in $\mathcal{G}(\mathcal{P})$ and are separated by a path of tree edges $R = \{r_1, \ldots, r_i\}$ where $i \geq 2$, $r_1 \in P$, and $r_i \in Q$. Contract all the tree edges of T_1 in $\mathcal{G}(\mathcal{P})$ except for $\{r_1, r_2\}$.

Similarly, let P' be the path of tree edges between vertices $\theta_2(a)$ and $\theta_2(c)$ in $\mathcal{G}(\mathcal{P})$ and let Q' be the path of tree edges between vertices $\theta_2(b)$ and $\theta_2(d)$ in $\mathcal{G}(\mathcal{P})$. Paths P' and Q' do not intersect in $\mathcal{G}(\mathcal{P})$ and are separated by a path of tree edges $R' = \{r'_1, \ldots, r'_j\}$ where $j \geq 2$, r'_1 in path P', and r'_j in path Q'. Contract all the tree edges of T_2 in $\mathcal{G}(\mathcal{P})$ except for $\{r'_1, r'_2\}$. The resulting graph G' is a K_4 with two tree edges $\{r_1, r_2\}$ and $\{r'_1, r'_2\}$. Since G' is derived from $\mathcal{G}(\mathcal{P})$ by contraction of tree edges, G' is a modified minor of $\mathcal{G}(\mathcal{P})$. □

3.3.3 Relation to partition intersection graphs

Let $\mathcal{P} = \{T_1, \ldots, T_k\}$ be a profile of unrooted trees where, for every $i \in [k]$, T_i has at most one internal edge. Consider the graph $G_{\mathcal{P}} = (V(\mathcal{G}(\mathcal{P})), E(\mathcal{G}(\mathcal{P})) \setminus \hat{E}(\mathcal{G}(\mathcal{P})))$. We have the following.

Lemma 13. *Graphs $G_{\mathcal{P}}$ and $\mathrm{Int}(\mathcal{C}_{\mathcal{P}})$ are isomorphic.*

Proof. For every $i \in [k]$, let θ_i be the label mapping function of tree T_i. For every $i \in [k]$, $\Sigma(T_i)$ contains exactly one split and let χ_i be that split. If $\{u, v\}$ are the internal vertices of tree T_i, then $\chi_i = \{\theta_i^{-1}(u), \theta_i^{-1}(v)\}$. Let f be a function from $V(\mathrm{Int}(\mathcal{C}_{\mathcal{P}}))$ to $V(G_{\mathcal{P}})$ defined as follows. For every $i \in [k]$ and $A \in \chi_i$, set $f((\chi_i, A))$ to $\theta_i(\ell)$ for some $\ell \in A$. Note that, for any $\{\ell_1, \ell_2\} \subseteq A$, $\theta_i(\ell_1) = \theta_i(\ell_2)$. Since, $\theta_i^{-1}(u) \in \chi_i$ for any internal vertex u in T_i, $f^{-1}(u)$ is unique and hence, f is bijective.

Let $\{(\chi_i, A), (\chi_j, B)\}$ be an edge in $\mathrm{Int}(\mathcal{C}_{\mathcal{P}})$. Let $\ell \subseteq A \cap B$. Then, $f((\chi_i, A)) = \theta_i(\ell)$ and $f((\chi_j, B)) = \theta_j(\ell)$. Since, $\theta_i^{-1}(\theta_i(\ell)) \cap \theta_j^{-1}(\theta_j(\ell)) \neq \emptyset$, there will exist an edge $\{f((\chi_i, A)), f((\chi_j, B))\}$ in $G_{\mathcal{P}}$. Similarly, let $\{u, v\}$ be an edge in $G_{\mathcal{P}}$ where $u \in V(T_i)$ and $v \in V(T_j)$ for $1 \leq i \neq j \leq k$. Then $f^{-1}(u) = (\chi_i, \theta_i^{-1}(u))$ and $f^{-1}(v) = (\chi_j, \theta_j^{-1}(v))$. Since $\theta_i^{-1}(u) \cap \theta_j^{-1}(v) \neq \emptyset$, there exists an edge $\{(\chi_i, \theta_i^{-1}(u)), (\chi_j, \theta_j^{-1}(v))\}$ in $\mathrm{Int}(\mathcal{C}_{\mathcal{P}})$. □

From Lemma 13, there exists a bijection $f : V(\mathrm{Int}(\mathcal{C}_{\mathcal{P}})) \to V(G_{\mathcal{P}})$ where is an edge $\{u, v\}$ in $\mathrm{Int}(\mathcal{C}_{\mathcal{P}})$ if and only if there is an edge $\{f(u), f(v)\}$ in $G_{\mathcal{P}}$. Lemmas 14 and 15 give the relationship between the legal triangulations of $\mathcal{G}(\mathcal{P})$ and $\mathrm{Int}(\mathcal{C}_{\mathcal{P}})$. Given a legal triangulation

of $\mathcal{G}(\mathcal{P})$ ($\text{Int}(\mathcal{C}_\mathcal{P})$ resp.), it is possible to transform it into a legal triangulation of $\text{Int}(\mathcal{C}_\mathcal{P})$ ($\mathcal{G}(\mathcal{P})$ resp.). The proofs of Lemmas 14 and 15 give those transformations.

Lemma 14. *Given a concise legal triangulation G' of $\mathcal{G}(\mathcal{P})$, there exists a legal triangulation H of int$(\mathcal{C}_\mathcal{P})$ where, for every edge $\{u, v\}$ in $\xi(\text{Int}(\mathcal{C}_\mathcal{P}), H)$, there exists an edge $\{f(u), f(v)\}$ in $\xi(\mathcal{G}(\mathcal{P}), G')$.*

Proof. Let (T, B) be a clique tree of G'. For every internal edge $\{u, v\} \in E(\mathcal{G}(\mathcal{P}))$, we do the following transformation on (T, B). Since G' is a concise legal triangulation, there is exactly one vertex $x \in V(T)$ where $\{u, v\} \subseteq B(x)$.

1. Add two vertices x_u and x_v to $V(T)$ and set $B(x_u)$ to $B(x) \setminus v$ and $B(x_v)$ to $B(x) \setminus u$.
2. For every vertex $x' \in N_T(x)$, if $u \in B(x')$, add an edge $\{x', x_u\}$ to $E(T)$. Otherwise, add an edge $\{x', x_v\}$. Delete edge $\{x, x'\}$ from $E(T)$
3. Delete x from $V(T)$. Add an edge $\{x_u, x_v\}$ to $E(T)$

Let (T', B') denote the transformed tree. Note that, (T', B') satisfies the vertex coverage and coherence properties. We build a legal triangulation H of $\text{Int}(\mathcal{C}_\mathcal{P})$ by first building the tree decomposition (T'', B'') corresponding to H. First, set $T'' = T'$ and $B'' = B'$. For every $x \in V(T'')$ and $v \in B''(x)$, substitute v in $B''(x)$ with $f^{-1}(v)$. Since f is a bijective function, (T'', B'') also satisfies vertex coverage and coherence properties. For every edge $\{u, v\}$ in $\text{Int}(\mathcal{C}_\mathcal{P})$ there exists an edge $\{f(u), f(v)\}$ in $\mathcal{G}(\mathcal{P})$. Since there exists a vertex $x \in V(T')$ where $\{f(u), f(v)\} \subseteq B'(x)$, $\{u, v\} \subseteq B''(x)$. Thus, (T'', B'') satisfies the edge coverage property and is a tree decomposition of $\text{Int}(\mathcal{C}_\mathcal{P})$.

Let H be the triangulation corresponding to the tree decomposition (T'', B'') of $\text{Int}(\mathcal{C}_\mathcal{P})$. For any $i \in [k]$, let u, v be the two vertices in $\text{Int}(\mathcal{C}_\mathcal{P})$ that correspond to the sets of partition χ_i. Note that, $f(u)$ and $f(v)$ are the internal vertices of tree T_i. Since G' is concise, and by construction, there is no vertex $x \in V(T')$ where $\{f(u), f(v)\} \subseteq B'(x)$. Thus, there is no vertex $x \in V(T'')$ where $\{u, v\} \subseteq B''(x)$. Hence H is a legal triangulation of $\text{Int}(\mathcal{C}_\mathcal{P})$. Also the transformation from (T', B') to (T'', B'') makes sure that, for every edge $\{u, v\}$ in $\xi(\text{Int}(\mathcal{C}_\mathcal{P}), H)$ there is an fill-in edge $\{f(u), f(v)\}$ in $\xi(\mathcal{G}(\mathcal{P}), G')$. □

Lemma 15. *Given a legal triangulation H of $\mathrm{Int}(\mathcal{C}_\mathcal{P})$, there exists a concise legal triangulation G' of $\mathcal{G}(\mathcal{P})$ where, for every edge $\{u, v\}$ in $\xi(\mathrm{Int}(\mathcal{C}_\mathcal{P}), H)$, there exists an edge $\{f(u), f(v)\}$ in $\xi(\mathcal{G}(\mathcal{P}), G')$.*

Proof. Let (T, B) be a clique tree of H. We do the following transformation on (T, B) for every character $\chi \in \mathcal{C}_\mathcal{P}$. Let u, v be the vertices in $\mathrm{Int}(\mathcal{C}_\mathcal{P})$ representing the sets of the partition χ. Consider the path $x, x_1, \ldots, x_m, y$ in T where $u \in B(x)$, $v \in B(y)$ and, for any $j \in [m]$, $u \notin B(x_j)$, and $v \notin B(x_j)$. There exists exactly one such path. Note that, the subpath $x_1, x_2, \ldots, x_m$ can be empty. For every $j \in [m]$, add u to $B(x_j)$. The resulting tree (T', B') is still a tree decomposition of H.

We build the legal triangulation G' of $\mathcal{G}(\mathcal{P})$ by building the tree decomposition (T'', B'') corresponding to G'. First set $T'' = T'$ and $B'' = B'$. Transform (T'', B'') as follows. For every $x \in V(T'')$ and $v \in B''(x)$, substitute v in $B''(x)$ with $f(v)$. Since f is bijective (T'', B'') satisfies the vertex coverage and coherence properties. To satisfy the edge coverage property we do the following. Subdivide every edge $\{x, y\} \in E(T'')$ as follows. Let $\{x_1, x_2, \ldots, x_p\} \subseteq B''(x)$ and $\{y_1, y_2, \ldots, y_p\} \subseteq B''(y)$ such that for every $i \in [p]$, $\{x_i, y_i\} = \hat{E}(T_j)$ for some $j \in [k]$ and p is the maximum such value. The edge $\{x, y\}$ is subdivided p times into vertices $v_1, v_2, \ldots, v_p$ where $B''(v_i) = \{B(x) \cap B(y)\} \cup \{x_j : i \leq j \leq p\} \cup \{y_j : 1 \leq j \leq i\}$. For every $i \in [p]$, $\{x_i, y_i\}$ is the only edge of any input tree present in $B(v_i)$. Similarly, for every $z \in T''$ and $i \in [p]$ where $z \neq v_i$, $\{x_i, y_i\} \not\subset B(z)$. The pair (T'', B'') now satisfies the edge coverage property and is a valid tree decomposition of $\mathcal{G}(\mathcal{P})$. The triangulation G' of $\mathcal{G}(\mathcal{P})$ corresponding to the tree decomposition (T'', B'') is a concise legal triangulation. Also the transformation from (T, B) to (T'', B'') ensures that, for every edge $\{u, v\}$ in $\xi(\mathrm{Int}(\mathcal{C}_\mathcal{P}), H)$, there exists an edge $\{f(u), f(v)\}$ in $\xi(\mathcal{G}(\mathcal{P}), G')$. □

Let $\mathcal{P}$ be a collection of phylogenetic trees. Using transformations similar to the ones used in the proofs of Lemmas 14 and 15, it is possible to convert a legal triangulation of $\mathcal{G}(\mathcal{P})$ to legal triangulation of $\mathrm{Int}(\mathcal{C}_\mathcal{P})$ and vice versa. In either case, the supertree need not be built.

3.4 Treewidth and Tree Compatibility

Let $\mathcal{P} = \{T_1, T_2, \ldots, T_k\}$ be a profile of unrooted trees. Bryant and Lagergren have shown the following in [Bryant and Lagergren (2006)].

Lemma 16. *If $\mathcal{P}$ is compatible,* $\text{tw}(G(\mathcal{P})) \leq k$.

By a similar argument, it can be shown that,

Lemma 17. *If $\mathcal{P}$ is compatible,* $\text{tw}(\mathcal{G}(\mathcal{P})) \leq k$.

The converse of Lemma 17 is not necessarily true as can be seen from Fig. 3.4. We will first show the relation between treewidth and compatibility for a special class of phylogenetic trees.

Theorem 6. *Let $\mathcal{P} = \{T_1, T_2, \ldots, T_k\}$ be a profile of unrooted trees where for any $1 \leq i \neq j \leq k$, $\mathcal{L}(T_i) = \mathcal{L}(T_j)$. Profile $\mathcal{P}$ is compatible if and only if $\mathcal{G}(\mathcal{P})$ has treewidth at most k.*

Proof. The 'only if" part follows from Lemma 17. To prove the "if" part, we first prove the following characterization of the minimal separators in $\mathcal{G}(\mathcal{P})$. Every minimal separator U in $\mathcal{G}(\mathcal{P})$ will satisfy at least one of the following conditions.

1. U contains at least one vertex from every tree in $\mathcal{P}$. If U contains vertices from two different trees, then it will contain a vertex from every tree in $\mathcal{P}$.

2. There are at least 3 vertices from some tree $T \in \mathcal{P}$.

For every $i \in [k]$ let θ_i be the label mapping function of T_i. Consider any minimal $a - b$ separator U. Let $a \in V(T_i)$ and $b \in V(T_j)$. Assume that, in $\mathcal{G}(\mathcal{P}) \setminus U$, there exists a path from a to $a' \in V(T_i)$ where $\theta_i(\ell) = a'$ for some $\ell \in \mathcal{L}(\mathcal{P})$, and, there exists a path from b to $b' \in V(T_j)$ where $\theta_j(\ell') = b'$ for some $\ell \in \mathcal{L}(\mathcal{P})$. In case, vertex a (b) is labelled, then both a (b) and a' (b') would be the same. Note that, ℓ and ℓ' cannot be the same. For any h where $i \neq h$ and $j \neq h$, there is a path $P = a - a' - c - d - b' - b$ in $\mathcal{G}(\mathcal{P})$ where $\theta_h(\ell) = c$ and $\theta_h(\ell') = d$. The subpath $c - d$ in P passes only through vertices of tree T_h. So, U should contain at least one vertex each from every tree T_h where $i \neq h$ and $j \neq h$. There is a path $P' = a - a' - b'' - b$ in $\mathcal{G}(\mathcal{P})$ where $\theta_j(\ell) = b''$. Similarly, there is a path $P'' = a - a'' - b' - b$

in $\mathcal{G}(\mathcal{P})$ where $\theta_i(\ell') = a''$. To break paths P' and P'', U should contain at least a vertex each from T_i and T_j. So, U will contain at least one vertex from every tree in $\mathcal{P}$. If U contains two vertices from two different trees, then there will exist paths from a to a labelled vertex in T_i and from b to a labelled vertex in T_j. Thus, U will contain a vertex from every input tree.

Now, assume that there is no path from a to any labelled vertex belonging to T_i in $\mathcal{G}(\mathcal{P}) - U$. This implies that a is not labelled. Any unlabelled vertex will have degree 3. Thus, U will contain at least 3 vertices from T_i and will not contain vertices from any other tree T_h where $i \neq h$.

Consider any minimal triangulation H of $\mathcal{G}(\mathcal{P})$ whose width is at most k. Let (T, B) be a clique tree of the minimal triangulation. By way of contradiction, assume that the triangulation H is illegal. Then, there is a vertex x in T where $B(x)$ contains two tree edges. If $B(x)$ contains a vertex from every input tree then the size of $B(x)$ would be at least $k + 2$. That would contradict the assumption that treewidth of $\mathcal{G}(\mathcal{P})$ is at most k. Thus, $B(x)$ cannot contain a vertex from every tree. Consider a neighbor y of x. The intersection of $B(x)$ and $B(y)$ is a minimal separator U of $\mathcal{G}(\mathcal{P})$. If U contains two vertices from different trees, U would contain vertices from every input tree and the size of $B(x)$ would be greater than or equal to $k+2$. So, U can only contain vertices from some tree T_i and has at least 3 vertices from tree T_i. Note that $B(y)$ will also contain vertices of U and thus cannot contain a vertex each from every tree. Repeatedly applying the same argument, we can conclude that there is no vertex $z \in V(T)$ where $B(z)$ contains a vertex each from every tree. But, this is not possible, since, for every label $\ell \in \mathcal{L}(\mathcal{P})$ there is a clique in $\mathcal{G}(\mathcal{P})$ with a vertex each from every tree and thus, there should be a vertex z in T where $B(z)$ contains a vertex each from every tree. □

Theorem 7. *Two phylogenetic trees are compatible if and only if their modified display graph has treewidth at most 2.*

Proof. The "only if" part follows from Lemma 17. To prove the "if" part, assume by contradiction that T_1 and T_2 are incompatible. Then, from Lemma 12, $\mathcal{G}(\mathcal{P})$ contains a K_4 with two tree edges as a modified minor. Any modified minor of $\mathcal{G}(\mathcal{P})$ is also a minor of $\mathcal{G}(\mathcal{P})$. Thus, $\mathcal{G}(\mathcal{P})$ contains a K_4 as a minor and $\text{tw}(\mathcal{G}(\mathcal{P})) > 2$. Hence, if treewidth of $\mathcal{G}(\mathcal{P})$ is at most 2,

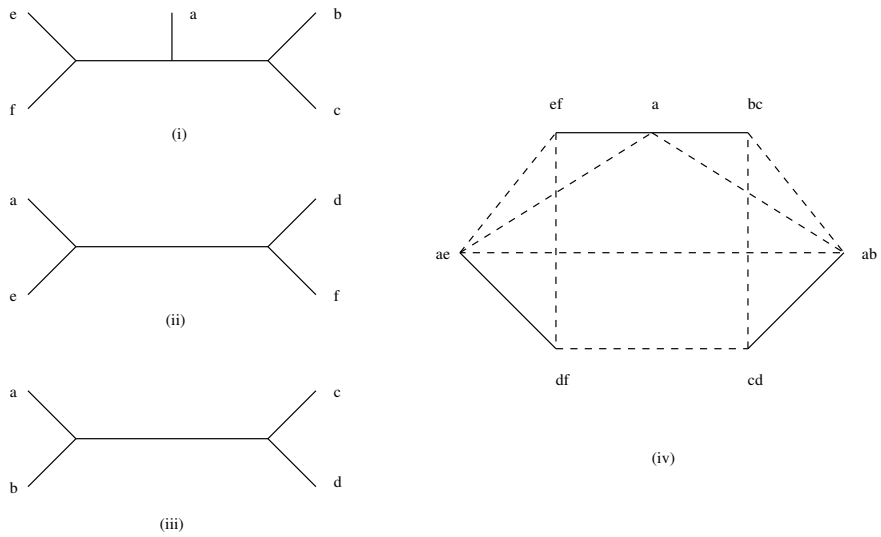


Figure 3.4: (i), (ii), and (iii) are three incompatible input trees. (iv) Modified display graph of input trees (i), (ii) and (iii); the graph has treewidth three.

T_1 and T_2 are compatible. □

The above theorem cannot be generalized for any number of trees. An example of three trees which are incompatible but whose display graph has treewidth 3 is given in Figure 3.4.

3.5 Edge Contraction and Tree Removal Problems

Given a profile $\mathcal{P}$ of unrooted trees, the *edge contraction problem* asks, if $\mathcal{P}$ can be made compatible by contraction of at most p internal edges from the input trees. Similarly, given a profile $\mathcal{P}$ of phylogenetic trees, the *tree removal problem* asks if $\mathcal{P}$ can be made compatible by removal of at most p trees from $\mathcal{P}$. By setting $p = 0$, there is a reduction from the quartet compatibility problem [Steel (1992)] to the edge contraction and tree removal problems. Hence, the edge contraction and tree removal problems are *NP-Hard*.

The above two problems are similar to the character removal problem. Given a collection $\mathcal{C}$ of characters, the *character removal* problem asks if $\mathcal{C}$ can be made compatible by removal of at most p characters. A characterization for finding the minimum number of characters to be removed to make $\mathcal{C}$ compatible was given by Bordewich et al. (2005). A character is broken

with respect to a specific triangulation of $Int(\mathcal{C})$ if there is fill-in edge between the states of the same character. Bordewich's characterization requires finding the triangulation of $Int(\mathcal{C})$ with minimum number of broken characters.

Bordewich's characterization can be used to solve the edge contraction and tree removal problems as follows. Profile $\mathcal{P}$ is compatible if and only if collection $\mathcal{C}_{\mathcal{P}}$ is compatible. Consider any triangulation H of $Int(\mathcal{C}_{\mathcal{P}})$. Deleting a broken character from $\mathcal{C}_{\mathcal{P}}$ corresponds to contracting the internal edge which induced it. Profile $\mathcal{P}$ can be made compatible by contraction of at most p edges if and only if there exists a triangulation of $Int(\mathcal{C}_{\mathcal{P}})$ with at most p broken characters. A similar characterization can be used to solve the tree removal problem. We provide a different characterization of edge contraction and tree removal problems using the modified display graphs. The size of $\mathcal{G}(\mathcal{P})$, in terms of number of vertices and edges, is smaller than the size of the $Int(\mathcal{C}_{\mathcal{P}})$. Also, the characterization we give in terms of triangulation of the modified display graphs for the edge contraction problem, allows us to bound the size of the treewidth of the modified display graphs.

Let $\mathcal{P} = \{T_1, T_2, \ldots, T_k\}$ be a profile of unrooted trees. A triangulation H of $\mathcal{G}(\mathcal{P})$ is *coherent*, if for every clique C in H, and for every $T \in \mathcal{P}$, $V(T) \cap V(C)$ induces a subtree in T. Note that, every legal triangulation is coherent. Consider any coherent triangulation H of $\mathcal{G}(\mathcal{P})$. Let $Q \subseteq \hat{E}(\mathcal{G}(\mathcal{P}))$ be a set of tree edges where, for every $C \in MC(H)$, $E(C) \setminus Q$ has at most one tree edge. We call the edges in Q, *blocking edges* of H. We denote the profile $\{T_1/(Q \cap E(T_1)), \ldots, T_k/(Q \cap E(T_k))\}$ by $\mathcal{P}/Q$.

Consider the set $P \subseteq \mathcal{P}$ of trees where $T \in P$ if and only if for every clique $C \in MC(H)$ there is at most one edge of T in C. Let $Y_1 \subseteq P$ be a set a trees where, for every clique $C \in MC(H)$, there is at most one tree which is not in Y_1 and has exactly one tree edge in C. Consider the set $Y_2 \subseteq \mathcal{P}$ where, $T \in Y_2$ if and only if there exists a clique in $MC(H)$ with more than one edge from T. Let $Y = Y_1 \cup Y_2$. We call the trees in Y, the *blocking trees* of H.

We then have the following theorems.

Theorem 8. *A profile $\mathcal{P}$ of unrooted trees can be made compatible by contraction of at most p internal edges, if and only if there exists a coherent triangulation of $\mathcal{G}(\mathcal{P})$ with at most p blocking edges.*

Corollary 3. *If a profile $\mathcal{P}$ of unrooted trees can be made compatible by contraction of at most p internal edges, then $tw(\mathcal{G}(\mathcal{P})) \leq k + p$.*

Theorem 9. *A profile $\mathcal{P}$ of unrooted trees can be made compatible by removal of at most p input trees, if and only if there exists a coherent triangulation of $\mathcal{G}(\mathcal{P})$, with at most p blocking trees.*

From Theorem 8, the answer to the edge contraction problem is "yes" if and only if there exists a coherent triangulation of $G(\mathcal{P})$ with at most p blocking edges. Similarly, by Theorem 9, the answer to the tree removal problem is "yes" if and only if there exists a coherent triangulation of $G(\mathcal{P})$ with at most p blocking trees.

For rest of the section we give the proofs of Theorems 8 and 9. Proof of Theorem 8 follows from Lemmas 18 and 19. Similarly, proof of Theorem 9 follows from Lemmas 20 and 21. We first give few definitions.

Let $G = (V, E)$ be a graph and let F be a subset of edges in G. Let $V' = \bigcup_{e \in F} e$. Let $h_1, h_2, \ldots, h_z$ represent the connected components of the graph $H = (V', F)$. Then, for every $i \in [z]$ there exists a vertex v_i in $V(G/F) \setminus V(G)$ satisfying the following condition. There exists an edge $\{v_i, v\} \in E(G/F)$ if and only if,

(1) $v \in \{N_G(V(h_i)) \setminus V'\}$ or

(2) $v = v_j$ for some $j \in [z]$ where $i \neq j$ and there exists a $v' \in \{V' \cap N_G(V(h_i))\}$ with $v' \in V(h_j)$

We define a surjective function σ from $V(G)$ to $V(G/F)$ as follows:

1. For every $v \in \{V(G) \setminus V'\}$, $\sigma(v) = v$
2. For every $v \in V'$ where $v \in V(h_i)$ for some $i \in [z]$, $\sigma(v) = v_i$

We call σ the *vertex mapping* function.

Lemma 18. *Let H be a coherent triangulation of $\mathcal{G}(\mathcal{P})$ and let Q be the blocking edges of H. Profile $\mathcal{P}/Q$ is compatible.*

Proof. For every $i \in [k]$, let $T_i' = T_i/(E(T_i) \cap Q)$ and let σ_i represent the vertex mapping function from T_i to T_i'. Consider a clique tree (T, B) of H. We do the following transformation on (T, B) for every input tree T_i where $i \in [k]$. For every $x \in V(T)$ and $v \in B(x)$ where $v \in V(T_i)$, replace v in $B(x)$ with $\sigma_i(v)$. Let (T, B') represent the resulting pair.

We now show that (T, B') is a valid tree decomposition of $\mathcal{G}(\mathcal{P}/Q)$. For every vertex $v \in \mathcal{I}(T_i')$, consider any vertex $v' \in \sigma_i^{-1}(v)$. There should exist an $x \in V(T)$ where $v' \in B(x)$ and so, $B'(x)$ will contain v. Thus (T, B') satisfies the vertex coverage property. Consider any edge $\{u, v\} \in \hat{E}(\mathcal{G}(\mathcal{P}/Q))$ where $\{u, v\} \subseteq V(T_i')$ for some $i \in [k]$. There exists, a vertex $u' \in \sigma_i^{-1}(u)$ and a vertex $v' \in \sigma_i^{-1}(v)$ where, $\{u', v'\} \in E(T_i)$. There must exist a vertex $x \in V(T)$ where $\{u', v'\} \subseteq B(x)$ and so, $\{u, v\} \subseteq B'(x)$. Consider any added edge $\{u, v\} \in E(\mathcal{G}(\mathcal{P}/Q))$ where, $u \in V(T_i')$ and $v \in V(T_j')$, $1 \leq i \neq j \leq k$. There exists, a vertex $u' \in \sigma_i^{-1}(u)$ and a vertex $v' \in \sigma_j^{-1}(v)$ where, $\{u', v'\} \in E(\mathcal{G}(\mathcal{P}))$. There exists an $x \in V(T)$ where $\{u', v'\} \subseteq B(x)$ and so, $\{u, v\} \subseteq B'(x)$. Thus, (T, B') satisfies the edge coverage property. Consider any vertex $v \in \{V(\mathcal{G}(\mathcal{P})) \cap V(\mathcal{G}(\mathcal{P}/Q))\}$. Since the vertex set $\{x : v \in B(x)\}$ induces a subtree in T, the vertex set $\{x : v \in B'(x)\}$ also induces a subtree in T. Now, consider any vertex $v \in \{V(\mathcal{G}(\mathcal{P}/Q)) \setminus V(\mathcal{G}(\mathcal{P}))\}$. Let $v \in V(T_i')$ for some $i \in [k]$ and let h be the subtree induced by vertices of $\sigma_i^{-1}(v)$ in T_i. The vertex set $\{x : \{V(h) \cap B(x)\} \neq \emptyset\}$ induces a subtree in T. Thus, the vertex set $\{x : v \in B'(x)\}$ also induces a subtree in T satisfying the coherence property. The pair (T, B') is hence a valid tree decomposition of $\mathcal{G}(\mathcal{P}/Q)$.

We now prove that the triangulation H' of $\mathcal{G}(\mathcal{P}/Q)$ corresponding to (T, B') is legal. By way of contradiction, assume that there exists a clique in H' which contains two tree edges, $\{p, q\} \in \hat{E}(T_i')$ and $\{r, s\} \in \hat{E}(T_j')$ for some $i \in [k]$ and $j \in [k]$. Then, there exists an $x \in V(T)$ where, $\{p, q, r, s\} \subseteq B'(x)$. There will exist, $p' \in \sigma_i^{-1}(p)$, $q' \in \sigma_i^{-1}(q)$, $r' \in \sigma_j^{-1}(r)$, $s' \in \sigma_j^{-1}(s)$ where $\{p', q'\} \in E(T_i)$, $\{r', s'\} \in E(T_j)$ and a clique C in H which contains $\{p', q', r', s'\}$. The set $E(C) \setminus Q$ will then have at least two tree edges which is a contradiction. Thus, H' is a legal triangulation of $\mathcal{G}(\mathcal{P}/Q)$ and the profile $\mathcal{P}/Q$ is compatible. □

Lemma 19. *If $\mathcal{P}/Q$ is compatible for some $Q \subseteq \hat{E}(\mathcal{G}(\mathcal{P}))$, then there exists a coherent triangulation H of $\mathcal{G}(\mathcal{P})$ where, the edges of Q are the blocking edges of H.*

Proof. For every $i \in [k]$, let $T_i' = T_i/Q$ and let σ_i represent the vertex mapping function from $V(T_i)$ to $V(T_i')$. Let H' be a legal triangulation of $\mathcal{G}(\mathcal{P}/Q)$. Consider a clique tree (T, B) of H'. We do the following transformation on (T, B) for every $i \in [k]$. For every $x \in V(T)$ and $v \in B(x)$ where $v \in V(T_i)$, replace v in $B(x)$ with $\sigma_i^{-1}(v)$. Let (T, B') be the resulting pair. We now show that (T, B') is a tree decomposition of $\mathcal{G}(\mathcal{P})$.

For every, $v \in T_i$ and $i \in [k]$, there exists a vertex $x \in V(T)$ such that $\sigma_i(v) \in B(x)$ and thus, $v \in B'(x)$. Hence, the pair (T, B') satisfies the vertex coverage property. Consider any tree edge $\{u, v\} \in E(T_i)$ of $\mathcal{G}(\mathcal{P})$. There exists a vertex $x \in V(T)$ where $\{\sigma_i(u), \sigma_i(v)\} \subseteq B(x)$ and thus, $\{u, v\} \subseteq B'(x)$. Consider any added edge $\{u, v\} \in E(\mathcal{G}(\mathcal{P}))$ where $u \in V(T_i)$, $v \in V(T_j)$ for some $1 \leq i \neq j \leq k$. There exists a vertex $x \in V(T)$ where $\{\sigma_i(u), \sigma_j(v)\} \subseteq B(x)$. Thus, $\{u, v\} \subseteq B'(x)$ satisfying the edge coverage property. Consider any vertex $v \in V(T_i)$ for some $i \in [k]$. The set $\{x : \sigma_i(v) \in B(x)\}$ induces a subtree in T. Thus the set $\{x : \sigma_i^{-1}(\sigma_i(v)) \subseteq B'(x)\}$ also induces a subtree in T, satisfying the coherence property. The pair (T, B') is thus a tree decomposition of $\mathcal{G}(\mathcal{P})$. Let H be the triangulation of $\mathcal{G}(\mathcal{P})$ corresponding to tree decomposition (T, B').

We will now prove that H is a coherent triangulation. By way of contradiction, assume that there exists a maximal clique C in H and $i \in [k]$ where the the vertices of T_i in C induce two non-adjacent subtrees h and h' in T_i such that $V(h) \cap V(h') = \emptyset$. Consider any two vertices $u \in V(h)$ and $v \in V(h')$. There exists a vertex $x \in V(T)$ where $\{u, v\} \subseteq B'(x)$. Since, $V(h) \cap V(h') = \emptyset$, vertices $\sigma_i(u)$, $\sigma_i(v)$ are different and non adjacent in T_i'. The clique induced by vertices of $B(x)$ in H' contains two non-adjacent vertices from tree T_i' and hence the triangulation H' is illegal, which is a contradiction. Thus, H is a coherent triangulation of $\mathcal{G}(\mathcal{P})$.

We now prove that the edges in Q are the blocking edges of H. By way of contradiction, assume that there exists a clique C in H where $E(C) \setminus Q$ has more than one tree edge. Let e, e' be the tree edges in $E(C) \setminus Q$. We have the following two cases.

1. Edges e and e' share an endpoint. Let $e = \{p, q\}$ and $e' = \{q, r\}$. Since e and e' share an endpoint and are tree edges, both the edges belong to same tree T_j for some $j \in [k]$. There

exists a vertex $x \in V(T)$ where $\{p, q, r\} \subseteq B'(x)$ and thus, $\{\sigma_j(p), \sigma_j(q), \sigma_j(r)\} \subseteq B(x)$. Since, $\{e, e'\} \not\subseteq Q$, $\sigma_j(p) \neq \sigma_j(q) \neq \sigma_j(r)$. The clique induced by the vertices of $B(x)$ in H' contains the tree edges $\{\sigma_j(p), \sigma_j(q)\}$, $\{\sigma_j(q), \sigma_j(r)\}$. The triangulation H' is thus illegal which is a contradiction.

2. Edges e and e' do not share an endpoint. Let $e = \{p, q\}$, $e' = \{r, s\}$ where $e \in \hat{E}(T_i)$ and $e' \in \hat{E}(T_j)$ for some $i \in [k]$ and $j \in [k]$. There exists a vertex $x \in V(T)$ where $\{p, q, r, s\} \subseteq B'(x)$ and thus, $\{\sigma_x(p), \sigma_x(q), \sigma_y(r), \sigma_y(s)\} \subseteq B(x)$. Since $\{e, e'\} \not\subseteq Q$, $\sigma_i(p) \neq \sigma_i(q) \neq \sigma_j(r) \neq \sigma_j(s)$. The clique induced by vertices of $B(x)$ in H' contains tree edges $\{\sigma_i(p), \sigma_i(q)\}$, $\{\sigma_j(r), \sigma_j(s)\}$. The triangulation H' is thus illegal which is a contradiction.

□

Lemma 20. *Let Y be the blocking trees of a coherent triangulation H of $\mathcal{G}(\mathcal{P})$. The profile $\mathcal{P} \setminus Y$ is compatible.*

Proof. Delete the vertices of trees in Y from H. Let H' be the resulting graph. Any induced subgraph of a triangulated graph is also triangulated. So, H' is triangulated. Every edge present in $\mathcal{G}(\mathcal{P} \setminus Y)$ is also present in H'. Hence, H' is a triangulation of $\mathcal{G}(\mathcal{P} \setminus Y)$. To see that H' is a legal triangulation of $\mathcal{G}(\mathcal{P} \setminus Y)$, by way of contradiction, assume that there is a clique C in H' which contains two tree edges e and e'. Since, H' is an induced subgraph of H, there exists a clique C' in H which contains both e and e'. Edges e and e' cannot be from the same tree, since any tree which contains more than one edge in a clique is contained in Y. Since e and e' are from different trees, the clique C' in H contains at least an edge each from two different trees, both of which are not in Y, a contradiction. Thus, H' is a legal triangulation of $\mathcal{G}(\mathcal{P} \setminus Y)$. □

Lemma 21. *Let $Y \subseteq \mathcal{P}$ be a set of unrooted trees where, the profile $\mathcal{P} \setminus Y$ is compatible. Then, there exists a coherent triangulation H of $\mathcal{G}(\mathcal{P})$ where, the trees of Y are the blocking trees of H.*

Proof. Let H' be a legal triangulation of $\mathcal{G}(\mathcal{P} \setminus Y)$. Let $V' = \{v : v \in \hat{E}(T), T \in Y\}$. Add V' to graph H' and make V' a clique. For every clique $C \in MC(H')$, make $V' \cup V(C)$ a clique. Let H be the resulting graph. Since, H' is a legal triangulation of $\mathcal{G}(\mathcal{P} \setminus Y)$, graph H is a coherent triangulation of $\mathcal{G}(\mathcal{P})$. There exists a clique $C \in MC(H)$ if and only if $C \setminus V'$ is a clique in $MC(H')$. It can be seen easily that the trees in Y are the blocking trees of triangulation H. □

3.6 Concluding Remarks

We characterized the unrooted tree compatibility problem in terms of triangulations of the display graph. We then modified the display graphs to derive a concise characterization for the unrooted tree compatibility problem. Every modified display graph defines an equivalence class of profiles of phylogenetic trees. Profiles of phylogenetic trees which have the same modified display graph belong to the same class. We identified a special case of unrooted phylogenetic tree problem which can be solved in polynomial time. We also proved that any modified display graph which contains a K_4 with two non adjacent tree edges as a modified minor will not have a legal triangulation. The K_4 with non adjacent tree edges is thus an obstruction for unrooted tree compatibility. It needs to be investigated, if the number of such obstructions is finite, and polynomial in size for profiles with fixed number of unrooted phylogenetic trees. We defined the edge contraction and tree removal problems and provided characterizations in terms of the modified display graph for the same. We provided an upper bound on the treewidth of the modified display graph for the edge contraction problem. This leaves open the question if the edge contraction problem is fixed parameter tractable.

CHAPTER 4. CHARACTERIZING COMPATIBILITY AND AGREEMENT OF UNROOTED TREES WITH CUTS

Sudheer Vakati, David Fernández-Baca

Deciding whether a collection of unrooted trees is compatible is a fundamental problem in phylogenetics. Two different graph-theoretic characterizations of tree compatibility have recently been proposed. In one of these, tree compatibility is characterized in terms of the existence of a specific kind of triangulation in a structure known as the display graph. An alternative characterization expresses the tree compatibility problem as a chordal graph sandwich problem in a structure known as the edge label intersection graph. Here, we show that the characterization using edge label intersection graphs yields to a characterization in terms of minimal cuts of the display graph. We show how these two characterizations are related to compatibility of splits. We also derive characterizations for the agreement supertree problem in terms of minimal cuts and minimal separators of display and edge label intersection graphs respectively.

4.1 Introduction

Vakati and Fernández-Baca characterized the tree compatibility problem in terms of finding a legal triangulation [Vakati and Fernández-Baca (2011)] of the display graph of a profile, a graph introduced by Bryant and Lagergren [Bryant and Lagergren (2006)]. Gysel et al. introduced the edge label intersection graph for a profile of phylogenetic trees and used this graph to characterize tree compatibility as a chordal sandwich problem [Gysel et al. (2012)].

In this paper we explore the connection between separators in the edge label intersection graph of a profile and cuts in the display graph of the profile (Section 4.2). We show that

this leads to a new, and very natural, characterization of compatibility in terms of minimal cuts in the display graph (Section 4.3). We then show how such cuts are closely related to the splits of the compatible supertree (Section 4.4). We give a characterization of the agreement supertree problem in terms of minimal cuts of the display graph (Section 4.5). This characterization also translates to a characterization in terms of minimal separators of the edge label intersection graph. To the best of our knowledge, there is no other known characterization of the agreement supertree problem for unrooted trees. Lastly, we give a transformation from the cuts characterization of the display graph to the characterization in terms of legal triangulation (Section 4.6).

4.2 Display Graphs and Edge Label Intersection Graphs

Let $\mathcal{P}$ be a profile. In what follows, we assume that $G(\mathcal{P})$ is connected. If it is not, the connected components of $G(\mathcal{P})$ induce a partition of $\mathcal{P}$ into sub-profiles such that for each sub-profile $\mathcal{P}'$, $G(\mathcal{P}')$ is a connected component of $G(\mathcal{P})$. It is easy to see that $\mathcal{P}$ is compatible if and only if each sub-profile is compatible.

The *line graph* of a graph G, denoted by $LG(G)$, is a graph whose vertex set is $E(G)$ and two vertices of $LG(G)$ are adjacent if the corresponding edges in G share an endpoint. For rest of the paper we denote the line graph $LG(G(\mathcal{P}))$ of $G(\mathcal{P})$ by $LG(\mathcal{P})$. Graph $LG(\mathcal{P})$ is the modified edge label intersection graph defined in Gysel et al. (2012). Note that if $G(\mathcal{P})$ is connected, then so is $LG(\mathcal{P})$. Indeed the following useful fact is easy to prove.

Observation 1. *Let I be a set of edges of $G(\mathcal{P})$ and let $\{v_1, v_2, \cdots, v_m\} \subseteq V(G(\mathcal{P}))$ where $m \geq 2$. Then, there exists a path $v_1, v_2, \cdots, v_m$ in $G(\mathcal{P}) - I$ if and only if there exists a path $\{v_1, v_2\}, \cdots, \{v_{m-1}, v_m\}$ in $LG(\mathcal{P}) - I$.*

Thus, in what follows, we assume that $LG(\mathcal{P})$ is connected.

For an unrooted tree T, we use $LG(T)$ to denote $LG(\{T\})$. A fill-in edge of $LG(\mathcal{P})$ is *valid* if both its endpoints are not in $LG(T)$ for every $T \in \mathcal{P}$. A triangulation H of $LG(\mathcal{P})$ is *restricted* if every fill-in edge of H is valid.

Theorem 10. *[Gysel et al. (2012)] A profile $\mathcal{P}$ of unrooted phylogenetic trees is compatible if and only if there exists a restricted triangulation of $LG(\mathcal{P})$.*

A minimal separator F of $LG(\mathcal{P})$ is *legal* if for every $T \in \mathcal{P}$, all the edges of T in F share a common endpoint; i.e., $F \cap E(T)$ is a clique in $LG(T)$. The following theorem was first mentioned in Gysel et al. (2012). For future reference, we formally state it and prove it here.

Theorem 11. *A profile $\mathcal{P}$ is compatible if and only if there exists a maximal set $\mathcal{F}$ of pairwise parallel minimal separators in $LG(\mathcal{P})$ where every separator in $\mathcal{F}$ is legal.*

Proof. We use the same technique in Gusfield (2009) to derive a characterization of the perfect phylogeny problem in terms of minimal separators of the partition intersection graph.

Assume that $\mathcal{P}$ is compatible. From Theorem 10, there exists a restricted triangulation H of $LG(\mathcal{P})$. We can assume that H is minimal triangulation, since if it is not, a restricted minimal triangulation of $LG(\mathcal{P})$ can be obtained by repeatedly deleting fill-in edges from H until it is a minimal triangulation. Let $\mathcal{F} = \triangle_H$. From Theorem 1, $\mathcal{F}$ is a maximal set of pairwise parallel minimal separators of $LG(\mathcal{P})$ and $LG(\mathcal{P})_{\mathcal{F}} = H$. Assume that $\mathcal{F}$ contains a separator F that is not legal. Let $\{e, e'\} \subseteq F$ where $\{e, e'\} \subseteq E(T)$ for some input tree T and $e \cap e' = \emptyset$. The vertices of F form a clique in H. Thus, H contains the edge $\{e, e'\}$. Since $\{e, e'\}$ is not a valid edge, H is not a restricted triangulation, which is a contradiction. Hence, every separator in $\mathcal{F}$ is legal.

Let $\mathcal{F}$ be a maximal set of pairwise parallel minimal separators of $LG(\mathcal{P})$ where every separator in $\mathcal{F}$ is legal. From Theorem 1, $LG(\mathcal{P})_{\mathcal{F}}$ is a minimal triangulation of $LG(\mathcal{F})$. If $\{e, e'\} \in E(LG(\mathcal{P})_{\mathcal{F}})$ is a fill-in edge, then $e \cap e' = \emptyset$ and there exists a minimal separator $F \in \mathcal{F}$ where $\{e, e'\} \subseteq F$. Since separator F is legal, if $\{e, e'\} \subseteq E(T)$ for some input tree T then $e \cap e' \neq \emptyset$. Thus, both e and e' are not from $LG(T)$ for any input tree T. Hence, every fill-in edge in $LG(\mathcal{P})_{\mathcal{F}}$ is valid, and $LG(\mathcal{P})_{\mathcal{F}}$ is a restricted triangulation. □

Let u of be a vertex of some input tree, Then, $\text{Inc}(u)$ denotes the set of all vertices e of $LG(\mathcal{P})$ such that $u \in e$. Equivalently, $\text{Inc}(u)$ is the set of all edges of $G(\mathcal{P})$ incident on u.

Lemma 22. *Let F be any minimal separator of $LG(\mathcal{P})$ and u be any vertex of any input tree. Then,* $\text{Inc}(u) \not\subseteq F$.

Proof. Suppose F is a minimal a-b separator of $LG(\mathcal{P})$ and u is a vertex of some input tree such that $\text{Inc}(u) \subseteq F$. Consider any vertex $e \in \text{Inc}(u)$. Then, there exists a path π from a to b in $LG(\mathcal{P})$ where e is the only vertex of F in π. If such a path π did not exist, then $F - e$ would still be a a-b separator, and F would not be minimal, a contradiction. Let e_1 and e_2 be the neighbors of e in π and let $e = \{u, v\}$. Since $\text{Inc}(u) \subseteq F$, π does not contain any other vertex e' where $u \in e'$. Thus, $e \cap e_1 = \{v\}$ and $e \cap e_2 = \{v\}$. Let $\pi = a, \cdots, e_1, e, e_2, \cdots, b$. Then $\pi' = a, \cdots, e_1, e_2, \cdots, b$ is also a path from a to b. But π' does not contain any vertex of F thus contradicting the assumption that F is a separator of $LG(\mathcal{P})$. Hence, neither such a minimal separator F nor such a vertex u can exist. □

Lemma 23. *If F is a minimal separator of $LG(\mathcal{P})$, then $LG(\mathcal{P}) - F$ has exactly two connected components.*

Proof. Assume that $LG(\mathcal{P}) - F$ has more than two connected components. From Lemma 1, $LG(\mathcal{P}) - F$ has at least two full components. Let H_1 and H_2 be two full components of $LG(\mathcal{P}) - F$. Let H_3 be a connected component of $LG(\mathcal{P}) - F$ different from H_1 and H_2. Then, there exists an edge $\{e, e_3\}$ in $LG(\mathcal{P})$ where $e \in F$ and $e_3 \in V(H_3)$.

Since H_1 and H_2 are full components, there exist edges $\{e, e_1\}$, $\{e, e_2\}$ in $LG(\mathcal{P})$ where, $e_1 \in V(H_1)$ and $e_2 \in V(H_2)$. Let $e = \{u, v\}$. Without loss of generality, let $u \in e \cap e_3$. Then, there will not exist a vertex $f \in V(H_1)$ where $u \in e \cap f$. Thus, $v \in e \cap e_1$. Similarly, H_1, H_2, H_3 are different connected components of $LG(\mathcal{P}) - F$, and hence, there will not exist a $f \in V(H_2)$ where, $u \in f \cap e$ or $v \in f \cap e$. Since H_2 does not contain a vertex adjacent to e, H_2 is not a full component which is a contradiction. □

Corollary 4. *If F is a minimal separator of $LG(\mathcal{P})$, then $LG(\mathcal{P}) - F'$ is connected for any $F' \subset F$.*

Lemma 24. *Let F be a subset of $E(G(\mathcal{P}))$. Then, F is a legal minimal separator of $LG(\mathcal{P})$ if and only if F is a legal minimal cut of $G(\mathcal{P})$.*

Proof. We will prove that if F is a legal minimal separator of $LG(\mathcal{P})$ then F is a legal minimal cut of $G(\mathcal{P})$. The proof for the other direction is similar and is omitted.

First, we show that F is a cut of $G(\mathcal{P})$. Assume the contrary. Let $\{u, v\}$ and $\{p, q\}$ be vertices in different components of $LG(\mathcal{P}) - F$. Since $G(\mathcal{P}) - F$ is connected, there exists a path between vertices u and q. Also, $\{u, v\} \notin F$ and $\{p, q\} \notin F$. Thus, by Observation 1 there also exists a path between vertices $\{u, v\}$ and $\{p, q\}$ of $LG(\mathcal{P}) - F$. This implies that $\{u, v\}$, $\{p, q\}$ are in the same connected component of $LG(\mathcal{P}) - F$, a contradiction. Thus F is a cut of $G(\mathcal{P})$.

Next we show that separator F of $LG(\mathcal{P})$ is a legal cut of $G(\mathcal{P})$. For every $T \in \mathcal{P}$ all the vertices of $LG(T)$ in F form a clique in $LG(T)$. Thus, all the edges of T in F are incident on a common vertex. Assume that $G(\mathcal{P}) - F$ has a connected component with no edge and let u be the vertex in one such component. Then, $\text{Inc}(u) \subseteq F$. But, F is a minimal separator of $LG(\mathcal{P})$ and by Lemma 22, $\text{Inc}(u) \not\subseteq F$ which is a contradiction. Thus, F is a legal cut of $G(\mathcal{P})$.

Lastly, we show that F is a minimal cut of $G(\mathcal{P})$. Assume the contrary. Then, there exists $F' \subset F$ where $G(\mathcal{P}) - F'$ is disconnected. Since $F' \subset F$ and every connected component of $G(\mathcal{P}) - F$ has at least one edge, every connected component of $G(\mathcal{P}) - F'$ also has at least one edge. Let $\{u, v\}$ and $\{p, q\}$ be the edges in different components of $G(\mathcal{P}) - F'$. By corollary 4, $LG(\mathcal{P}) - F'$ is connected and thus, there exists an path between $\{u, v\}$ and $\{p, q\}$ in $LG(\mathcal{P}) - F'$. Then, by Observation 1 there also exists a path between vertices u and p in $G(\mathcal{P}) - F'$. Hence, edges $\{u, v\}$ and $\{p, q\}$ are in the same connected component of $G - F'$ which is a contradiction. Thus, F is also a minimal cut of $G(\mathcal{P})$. □

Lemma 25. *Two legal minimal separators F and F' of $LG(\mathcal{P})$ are parallel if and only if the legal minimal cuts F and F' are parallel in $G(\mathcal{P})$.*

Proof. Assume that legal minimal separators F and F' of $LG(\mathcal{P})$ are parallel, but legal minimal cuts F and F' of $G(\mathcal{P})$ are not. Then, there exists $\{\{u, v\}, \{p, q\}\} \subseteq F'$ where $\{u, v\}$ and $\{p, q\}$ are present in different components of $G(\mathcal{P}) - F$. Since F and F' are parallel separators in $LG(\mathcal{P})$, and F does not contain $\{u, v\}$ and $\{p, q\}$, there exists a path between vertices $\{u, v\}$ and $\{p, q\}$ in $LG(\mathcal{P}) - F$. Then, by Observation 1 there also exists a path between vertices u and q in $G(\mathcal{P}) - F$. Thus, edges $\{u, v\}$ and $\{p, q\}$ are in the same connected component of $G(\mathcal{P}) - F$ which is a contradiction.

The other direction can be proved similarly using Observation 1. □

The next lemma follows from the definition of restricted triangulation and is from Gysel et al. (2012).

Lemma 26. *Let H be a restricted triangulation of $LG(\mathcal{P})$ and let (T, B) be a clique tree of H. Let $e = \{u, v\}$ be any vertex in $LG(\mathcal{P})$. Then, there does not exist a node $x \in V(T)$ where $B(x)$ contains vertices from both $\mathrm{Inc}(u) \setminus e$ and $\mathrm{Inc}(v) \setminus e$.*

Lemma 27. *Let T be a tree in $\mathcal{P}$ and suppose F is a minimal cut of $G(\mathcal{P})$ that contains precisely one edge e of T. Then, the edges of the two subtrees of $T - e$ are in different connected components of $G(\mathcal{P}) - F$.*

Proof. Since F is a minimal cut of $G(\mathcal{P})$, endpoints of e are in different connected components of $G(\mathcal{P}) - F$. Let $e = \{u, v\}$. For every $x \in e$, let T_x represent the subtree containing vertex x in $T - e$. Edge e is the only edge of T in F. Thus, for every $x \in e$ all the edges of T_x are in the same connected component of $G(\mathcal{P}) - F$ as vertex x. Since the endpoints of e are in different connected components of $G(\mathcal{P}) - F$, the edges of T_u and T_v are also in different connected components of $G(\mathcal{P}) - F$. □

4.3 Characterizing Tree Compatibility via Cuts

A cut F of the display graph $G(\mathcal{P})$ of a profile $\mathcal{P} = \{T_1, T_2, \cdots, T_k\}$ is *legal* if it satisfies the following:

(LC1) For every tree $T \in \mathcal{P}$, the edges of T in F are incident on a common vertex.

(LC2) There is at least one edge in each of the connected components of $G(\mathcal{P}) - F$.

A set $\mathcal{F}$ of pairwise parallel legal minimal cuts of $G(\mathcal{P})$ is *complete*, if for every input tree $T \in \mathcal{P}$ and for every internal edge e of T, there exists a cut $F \in \mathcal{F}$ where e is the only edge of T in F.

Example 1. For the display graph $G(\mathcal{P})$ of Fig. 2.1, let $\mathcal{F} = \{F_1, F_2, F_3, F_4\}$, where $F_1 = \{\{1, 2\}, \{5, 6\}\}$, $F_2 = \{\{2, 3\}, \{6, 7\}, \{5, 6\}\}$, $F_3 = \{\{4, 5\}, \{1, 2\}, \{1, c\}\}$ and $F_4 = \{\{6, 7\}, \{2, f\}\}$. Then, $\mathcal{F}$ is a complete set of pairwise parallel legal minimal cuts.

We characterize the compatibility of profile $\mathcal{P}$ in terms of minimal cuts of $G(\mathcal{P})$ as follows.

Theorem 12. *A profile $\mathcal{P}$ of unrooted phylogenetic trees is compatible if and only if there exists a complete set of pairwise parallel legal minimal cuts for $G(\mathcal{P})$.*

Theorem 12 and Lemmas 24 and 25 imply the following analogous result for $LG(\mathcal{P})$. A set $\mathcal{F}$ of legal minimal separators of $LG(\mathcal{P})$ is *complete*, if for every internal edge e of an input tree T, there exists a separator $F \in \mathcal{F}$ where e is the only vertex of $LG(T)$ in F.

Theorem 13. *A profile $\mathcal{P}$ of unrooted phylogenetic trees is compatible if and only if there exists a complete set of pairwise parallel legal minimal separators for $LG(\mathcal{P})$.*

Theorem 12 follows from Theorem 11 and the following lemma.

Lemma 28. *The following two statements are equivalent.*

(i) There exists a maximal set $\mathcal{F}$ of pairwise parallel minimal separators of $LG(\mathcal{P})$ where every separator in $\mathcal{F}$ is legal.

(ii) There exists a complete set of pairwise parallel minimal cuts for $G(\mathcal{P})$.

Proof. (i) $\Rightarrow$ (ii): We will show that for every internal edge $e = \{u, v\}$ of an input tree T there exists a minimal separator in $\mathcal{F}$ that contains only vertex e from $LG(T)$. Then from Theorem 24 and Lemma 25 it follows that $\mathcal{F}$ is a complete set of pairwise parallel legal minimal cuts for display graph $G(\mathcal{P})$.

As shown in the proof of Theorem 11, $LG(\mathcal{P})_{\mathcal{F}}$ is a restricted minimal triangulation of $LG(\mathcal{P})$. Let (S, B) be a clique tree of $LG(\mathcal{P})_{\mathcal{F}}$. By definition, the vertices in each of the sets $\text{Inc}(u)$ and $\text{Inc}(v)$ form a clique in $LG(\mathcal{P})$. Consider any vertex p of S where $\text{Inc}(u) \subseteq B(p)$ and any vertex q of S where $\text{Inc}(v) \subseteq B(q)$. Since (S, B) is a clique tree of $LG(\mathcal{P})_{\mathcal{F}}$, there will always exist such vertices p and q. Also, by Lemma 26, $p \neq q$, $B(p) \cap (\text{Inc}(v) \setminus \{e\}) = \emptyset$ and $B(q) \cap (\text{Inc}(u) \setminus \{e\}) = \emptyset$.

Let $\pi = p, x_1, x_2, \cdots, x_m, q$ be the path from p to q in S where $m \geq 0$. Let $x_0 = p$ and $x_{m+1} = q$. Let x_i be the vertex nearest to p in path π where $i \in [m+1]$ and $B(x_i) \cap (\text{Inc}(u) \setminus \{e\}) = \emptyset$. Let $F = B(x_{i-1}) \cap B(x_i)$. Then by Theorem 1, $F \in \mathcal{F}$. Since

$\mathrm{Inc}(u) \cap \mathrm{Inc}(v) = \{e\}$, by the coherence property of the clique tree, $e \in B(x_j)$ for every $j \in [m]$. Thus, $e \in F$. By Lemma 26, $B(x_{i-1}) \cap (\mathrm{Inc}(v) \setminus \{e\}) = \emptyset$. Since $B(x_i) \cap (\mathrm{Inc}(u) \setminus \{e\}) = \emptyset$, $F \cap \mathrm{Inc}(u) = \{e\}$ and $F \cap \mathrm{Inc}(v) = \{e\}$. Thus, for every vertex $e' \in LG(T)$ where $e \neq e'$ and $e \cap e' \neq \emptyset$, $e' \notin F$. Also, since every separator in $\mathcal{F}$ is legal, for every vertex $f \in LG(T)$ where $f \cap e = \emptyset$, $f \notin F$. Thus, e is the only vertex of $LG(T)$ in F.

(i) $\Leftarrow$ *(ii):* Consider any complete set of pairwise parallel legal minimal cuts $\mathcal{F}'$ of $G(\mathcal{P})$. By Theorem 24 and Lemma 25, $\mathcal{F}'$ is a set of pairwise parallel legal minimal separators of $LG(\mathcal{P})$. There exists a maximal set $\mathcal{F}$ of pairwise parallel minimal separators where $\mathcal{F}' \subseteq \mathcal{F}$. Assume that there exists a minimal separator F in $\mathcal{F} \setminus \mathcal{F}'$ which is not legal.

Since, by assumption, minimal separator F of $LG(\mathcal{P})$ is not legal, there exists a tree $T \in \mathcal{P}$ where at least two nonincident edges of T are in F. Let $e_1 = \{x, y\}$ and $e_2 = \{x', y'\}$ be those nonincident edges. Consider any internal edge e_3 in T where e_1 and e_2 are in different components of $T - e_3$. Such an edge exists because e_1 and e_2 are nonincident. Set $\mathcal{F}'$ is a complete set of pairwise parallel legal minimal cuts of G. Thus, there exists minimal cut $F' \in \mathcal{F}'$ where e_3 is the only edge of T in F'. Since, minimal separators F and F' are in $\mathcal{F}$, they are parallel to each other and vertices e_1 and e_2 are in the same connected component of $LG(\mathcal{P}) - F'$. Thus, by Observation 1, there exists a path between vertices x and x' in $G(\mathcal{P}) - F'$ and edges e_1 and e_2 are also in the same connected component of $G(\mathcal{P}) - F'$. But by Lemma 27 that is impossible.

Thus, every separator of $\mathcal{F} \setminus \mathcal{F}'$ is legal and the set $\mathcal{F}$ is a maximal set of pairwise minimal separators of $LG(\mathcal{P})$ where every separator in $\mathcal{F}$ is legal. □

4.4 Splits and Cuts

Theorem 12 is closely related to the characterization of the compatibility of splits. The following lemma shows that for every legal minimal cut of $G(\mathcal{P})$ we can derive a split of $\mathcal{L}(\mathcal{P})$.

Lemma 29. *Let F be a legal minimal cut of $G(\mathcal{P})$ and let G_1 and G_2 be the two connected components of $G(\mathcal{P}) - F$. Then, $\mathcal{L}(G_1)|\mathcal{L}(G_2)$ is a split of $\mathcal{L}(\mathcal{P})$.*

Proof. Consider G_i for any $i \in \{1, 2\}$. We will show that $\mathcal{L}(G_i)$ is non-empty. Since F is a legal minimal cut, G_i contains at least one edge e of $G(\mathcal{P})$. If e is a non internal edge, then $\mathcal{L}(G_i)$ is non-empty. Assume that $e = \{u, v\}$ is an internal edge of some input tree T. If F does not contain an edge of T, then $\mathcal{L}(T) \subseteq \mathcal{L}(G_i)$ and thus $\mathcal{L}(G_i)$ is non-empty. Assume that F contains one or more edges of T. Let T_u, T_v be the two subtrees of $T - e$. Since F is a legal minimal cut, F contains edges from either T_u or T_v but not both. Without loss of generality assume that F does not contain edges from T_u. Then, every edge of T_u will be in the same component as e. Since T_u contains at least one leaf vertex, $\mathcal{L}(G_i)$ is non-empty. Thus, $\mathcal{L}(G_1)|\mathcal{L}(G_2)$ is a split of $\mathcal{L}(\mathcal{P})$. □

For any legal minimal cut F of $G(\mathcal{P})$, we denote by $\sigma(F)$ the split of $\mathcal{L}(\mathcal{P})$ induced by F. If $\mathcal{F}$ is a set of legal minimal cuts of $G(\mathcal{P})$, we denote the set of all the non-trivial splits in $\bigcup_{F \in \mathcal{F}} \sigma(F)$ by $\Sigma(\mathcal{F})$. The relationship between splits compatibility and complete set of legal minimal cuts is given by the following lemma.

Theorem 14. *Suppose of $G(\mathcal{P})$ has a complete set of pairwise parallel legal minimal cuts $\mathcal{F}$. The following statements hold true.*

(i) $\Sigma(\mathcal{F})$ is compatible.

(ii) If S is a compatible tree for $\Sigma(\mathcal{F})$, then S is a compatible tree for $\mathcal{P}$.

(iii) There exists a compatible tree S of $\mathcal{P}$ where $\Sigma(S) = \Sigma(\mathcal{F})$.

Example 2. For the cuts of the display graph in Fig. 2.1 given in Example 1, we have $\sigma(F_1) = abc|defg$, $\sigma(F_2) = abcfg|de$, $\sigma(F_3) = ab|cdefg$, and $\sigma(F_4) = abcde|fg$. Note that these splits are pairwise compatible.

The proof of Theorem 14 uses the following lemma.

Lemma 30. *Let F_1 and F_2 be two parallel legal minimal cuts of $G(\mathcal{P})$. Then, $\sigma(F_1)$ and $\sigma(F_2)$ are compatible.*

Proof. Let $\sigma(F_1) = U_1|U_2$ and $\sigma(F_2) = V_1|V_2$. Assume that $\sigma(F_1)$ and $\sigma(F_2)$ are incompatible. Thus, the intersection of U_i and V_j for every $i \in \{1, 2\}$ and $j \in \{1, 2\}$ is non-empty. Let

$a \in U_1 \cap V_1$, $b \in U_1 \cap V_2$, $c \in U_2 \cap V_1$ and $d \in U_2 \cap V_2$. Since $\{a, b\} \subseteq U_1$, there exists a path π_1 between leaf vertices a and b in $G(\mathcal{P}) - F_1$. But a and b are in different components of $G(\mathcal{P}) - F_2$. Thus, an edge e_1 of path π_1 is in the cut F_2. Similarly, $\{c, d\} \subseteq U_2$ and there exists a path π_2 between labels c and d in $G(\mathcal{P}) - F_1$. Since c and d are in different components of $G(\mathcal{P}) - F_2$, cut F_2 contains an edge e_2 of path π_2. But paths π_1 and π_2 are in different components of $G(\mathcal{P}) - F_1$. So, edges e_1 and e_2 are in different components of $G(\mathcal{P}) - F_1$. Since $\{e_1, e_2\} \subseteq F_2$, the cuts F_1 and F_2 are not parallel, which is a contradiction. □

Proof of Theorem 14. (i) The statement follows from Lemma 30 and the splits equivalence theorem.

(ii) Let T be an input tree of $\mathcal{P}$ and let $S' = S|\mathcal{L}(T)$. We will show that S' displays $\sigma(e)$ for every internal edge e of T. Let $\sigma(e) = A|B$. There exists a cut $F \in \mathcal{F}$ where e is the only edge of T in F. Since F is a minimal cut, by Lemma 27, leaves of sets A and B are in different components of $G(\mathcal{P}) - F$. Thus, if $\sigma(F) = A'|B'$ then up to relabeling of sets we have $A \subseteq A'$ and $B \subseteq B'$. Because S displays $\sigma(F)$, S' also displays $\sigma(e)$. Since S' displays all the splits of T, T can be obtained from S' by contraction of zero or more edges [Semple and Steel (2003)]. Thus, S displays T. Since S displays every tree in $\mathcal{P}$, S is a compatible tree of $\mathcal{P}$.

(iii) This is a consequence of the well-known fact (see, e.g., [Semple and Steel (2003)]) that if X is a set of compatible non-trivial splits, there exists a tree T where $\Sigma(T) = X$. □

4.5 Characterizing Agreement via Cuts

We now characterize the agreement supertree problem in terms of minimal cuts in the display graph. This characterization is similar to the one for tree compatibility given by Theorem 12, except for an additional restriction on the minimal cuts.

Theorem 15. *A profile $\mathcal{P}$ has an agreement supertree if and only if $G(\mathcal{P})$ has a complete set $\mathcal{F}$ of pairwise parallel legal minimal cuts where, for every cut $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$ there is at most one edge of T in F.*

Example 3. One can verify that the display graph of Fig. 2.1 does not meet the conditions of Theorem 15 and, thus, the associated profile does not have an AST. On the other hand, for the

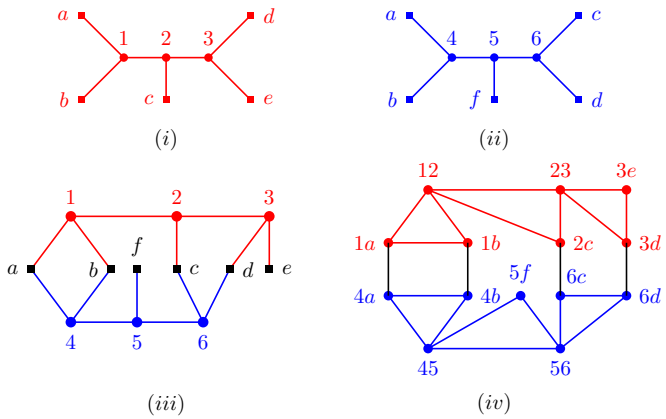


Figure 4.1: (i) First input tree. (ii) Second input tree, which agrees with the first. (iii) Display graph of the input trees. (iv) Edge label intersection graph of the input trees. For every vertex, uv represents label $\{u, v\}$.

display graph of Fig. 4.1, let $\mathcal{F} = \{F_1, F_2, F_3\}$, where $F_1 = \{\{1,2\},\{4,5\}\}$, $F_2 = \{\{1,2\},\{5,6\}\}$ and $F_3 = \{\{2,3\},\{6,c\}\}$. For any given input tree T, every cut in $\mathcal{F}$ has at most one edge of T. Also, $\mathcal{F}$ is a complete set of pairwise parallel legal minimal cuts. Thus, by Theorem 15, the input trees of Figure 4.1 have an AST.

The analogue of Theorem 15 for $LG(\mathcal{P})$ is as follows.

Theorem 16. *A profile $\mathcal{P}$ has an agreement supertree if and only if $LG(\mathcal{P})$ has a complete set $\mathcal{F}$ of pairwise parallel legal minimal separators where, for every $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$ there is at most one vertex of $LG(T)$ in F.*

Theorem 16 follows from Theorem 15 and Lemmas 24 and 25. Thus, the section is devoted to the proof of Theorem 15.

Let S be an AST of $\mathcal{P}$ and let $e = \{u, v\}$ be an edge of S. For each $x \in e$, L_x denotes the set of leaves of the subtree in $S - e$ that contains vertex x. Thus, $\sigma_e(S) = L_u|L_v$. Assume that there exists an input tree T where $\mathcal{L}(T) \cap L_x \neq \emptyset$ for every $x \in e$. Then there exists an edge $f \in E(T)$ where, if $\sigma_f(T) = A_1|A_2$ then $A_1 \subseteq L_u$ and $A_2 \subseteq L_v$. Otherwise $S|\mathcal{L}(T)$ will contain a split which is not in T and is thus not isomorphic to T. We call e an *agreement edge* of S corresponding to edge f of T. Note that there does not exist any other edge f' of T where e is

also an agreement edge of S with respect to edge f' of T.

Given an AST S of $\mathcal{P}$, we define a function Ψ from $E(S)$ to subsets of edges of $G(\mathcal{P})$ as follows. For every $e \in E(S)$, an edge f of an input tree T is in $\Psi(e)$ if and only if e is an agreement edge of S corresponding to edge f of T. Observe that Ψ is uniquely defined. We call Ψ the *cut function* of S. Given an edge $e \in E(S)$, we define a set V_x for every $x \in e$ as follows. For every $T \in \mathcal{P}$, V_x contains all the vertices of the minimal subtree of T connecting the labels in $\mathcal{L}(T) \cap L_x$. Note that if $e = \{u, v\}$ then $\{V_u, V_v\}$ is a partition of $V(G(\mathcal{P}))$.

Observation 2. *Let S be an AST of $\mathcal{P}$ and let Ψ be the cut function of S. If $\{e_1, e_2\} \subseteq E(S)$, then $\Psi(e_1) \neq \Psi(e_2)$.*

Lemma 31. *Let S be an AST of $\mathcal{P}$ and let Ψ be the cut function of S. The following statements hold true.*

(i) For every edge $e \in E(S)$, $\Psi(e)$ is a cut of $G(\mathcal{P})$.

(ii) For any edge $e \in E(S)$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$ if and only if $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components.

Proof. (i) Let $e = \{u, v\}$. We will show that $G(\mathcal{P}) - \Psi(e)$ does not contain an edge whose endpoints are in distinct sets of $\{V_u, V_v\}$. Assume the contrary. Let $f = \{x, y\}$ be an edge of $G(\mathcal{P}) - \Psi(e)$ where $x \in V_u$ and $y \in V_v$. Since $f \in G(\mathcal{P}) - \Psi(e)$, $f \notin \Psi(e)$. Let f be an edge of input tree T. There are two cases.

Case 1: There does not exist an edge of T in $\Psi(e)$. Then, there exists an endpoint p of e where $\mathcal{L}(T) \subseteq L_p$. Without loss of generality, let $u = p$. Then, $V(T) \subseteq V_u$ and thus $y \in V_u$, a contradiction.

Case 2: There exists an edge $f' \neq f$ of T in $\Psi(e)$. Let $f' = \{r, s\}$ and let $L_r \subseteq L_u$ and $L_s \subseteq L_v$. Let x,r be the vertices of f and f' where $L_x \subset L_r$. Since T is a phylogenetic tree, such vertices x and r exist. Since $L_r \subseteq L_u$, both the endpoints of f are in V_u which is a contradiction.

Thus, $G(\mathcal{P}) - \Psi(e)$ does not contain an edge whose endpoints are in different sets of $\{V_u, V_v\}$. Since V_u and V_v are non-empty, it follows that $\Psi(e)$ is a cut of $G(\mathcal{P})$.

(ii) The only if part follows from the definition of a minimal cut. We will prove the if part. Let $e = \{u, v\}$. Assume that $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components. From proof of (i), V_u and V_v are the vertex sets of those two connected components. Consider any edge $f \in \Psi(e)$. Endpoints of f are in different sets of $\{V_u, V_v\}$ and thus are in different connected components of $G(\mathcal{P}) - \Psi(e)$. This implies that $G(\mathcal{P}) - (\Psi(e) \setminus \{f\})$ is connected. Thus, if $G(\mathcal{P}) - \Psi(e)$ has exactly two connected components, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$. □

Let S be an AST of $\mathcal{P}$ and let $e = \{u, v\}$ be an edge of S. We define a *splitting operation* on edge e at u as follows. Let $\{L_1, \cdots, L_m\}$ be the partition of L_v where for every $i \in [m]$, $\mathcal{L}(C) \cap L_v = L_i$ for some connected component C in $G(\mathcal{P}) - \Psi(e)$. Let R be the rooted tree derived from the subtree containing v in $S - e$ by distinguishing vertex v as the root. For any $L \subseteq L_v$, let R^L be the minimal subtree of R connecting the labels in L. We denote by $R|L$, the tree derived from R^L by distinguishing the vertex closest to the root of R as the root and suppressing every other vertex which has degree two. Delete vertices of R from S. For every $i \in [m]$, add an edge from u to the root of $R|L_i$.

Let S' be the derived tree. If $m = 1$, then S and S' are isomorphic and we say the split operation is *invalid*. Otherwise, we say the split operation is *valid*. Let S_u be the subtree of $S - e$ containing vertex u. Note that, edges of $E(S_u)$ are present in both S and S'. We denote the edges in $E(S) \setminus E(S_u)$ and $E(S') \setminus E(S_u)$ by $E(S) \setminus E(S')$ and $E(S') \setminus E(S)$ respectively. We first make the following the following two observations.

Observation 3. *For any input tree T where $\mathcal{L}(T) \cap L_v \neq \emptyset$, all the labels of $\mathcal{L}(T) \cap L_v$ are in the same connected component of $G(\mathcal{P}) - \Psi(e)$.*

Observation 4. *Consider any connected component C of $G(\mathcal{P}) - \Psi(e)$ where $\mathcal{L}(C) \cap L_v \neq \emptyset$. Then, for every $X \subseteq (\mathcal{L}(C) \cap L_v)$, $S|X$ and $S'|X$ are isomorphic.*

The next observation follows from the definition of agreement supertrees and we make use of it in the next lemma.

Observation 5. *Let S and T be two phylogenetic trees where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$ and T is an induced subtree of S. For every $U \subseteq \mathcal{L}(S)$ where $\mathcal{L}(T) \subseteq U$, T is an induced subtree of $S|U$.*

Lemma 32. *Let S be an AST of $\mathcal{P}$ and let $e = \{u, v\}$ be an edge of S. Let S' be the tree derived by splitting edge e at u. Tree S' is an AST of $\mathcal{P}$.*

Proof. From construction, it can be seen that S' is a phylogenetic tree over $\mathcal{L}(\mathcal{P})$. Let $\{L_1, \cdots, L_m\}$ be the partition of L_v where for every $i \in [m]$, $\mathcal{L}(C) \cap L_v = L_i$ for some connected component C in $G(\mathcal{P}) - \Psi(e)$. Consider any input tree T of profile $\mathcal{P}$. We will prove that T is an induced subtree of S'. There are three cases.

Case 1: $\mathcal{L}(T) \subseteq L_u$. Since $\mathcal{L}(T) \subseteq L_u$, by Observation 5, T is an induced subtree of $S|L_u$. By construction of the split operation, trees $S|L_u$ and $S'|L_u$ are isomorphic. Thus, T is an induced subtree of S'.

Case 2: $\mathcal{L}(T) \subseteq L_v$. By Observation 3, $\mathcal{L}(T) \subseteq L_i$ for some $i \in [m]$. Since T is an induced subtree of S and $\mathcal{L}(T) \subseteq L_i$, by Observation 5, T is an induced subtree of $S|L_i$. By construction, trees $S|L_i$ and $S'|L_i$ are isomorphic. Thus, T is an induced subtree of S'.

Case 3: $(\mathcal{L}(T) \cap L_u \neq \emptyset) \wedge (\mathcal{L}(T) \cap L_v \neq \emptyset)$. By Observation 3, $\mathcal{L}(T) \cap L_v \subseteq L_i$ for some $i \in [m]$. Since T is an induced subtree of S and $\mathcal{L}(T) \subseteq (L_u \cup L_i)$, by Observation 5, T is also an induced subtree of $S|(L_u \cup L_i)$. By construction, trees $S|(L_u \cup L_i)$ and $S'|(L_u \cup L_i)$ are isomorphic. Thus, T is an induced subtree of $S'|(L_u \cup L_i)$. It follows that T is an induced subtree of S'.

Thus, S' is an AST of $\mathcal{P}$. □

Lemma 33. *Let S be an AST of $\mathcal{P}$ and let $e = \{u, v\}$ be an edge of S. Let S' be the tree derived by splitting edge e at u. Let Ψ, Ψ' be the cut functions of S and S' respectively. Consider any edge $f \in E(S') \setminus E(S)$. There exists an edge $e' \in E(S) \setminus E(S')$ where $\Psi'(f) \subseteq \Psi(e')$. Furthermore, if $\Psi(e')$ is a minimal cut of $G(\mathcal{P})$ then $\Psi'(f) = \Psi(e')$ and $\Psi'(f)$ is a minimal cut of $G(\mathcal{P})$.*

Proof. Let $f = \{x, y\}$ and let x be the vertex of f where $L_x \subseteq L_v$. Let S_p be the minimal subtree of S connecting the labels in L_x. Let p be the vertex of S_p closest to u in S. Let q be

the vertex adjacent to p in the path from p to u. Let $e' = \{p, q\}$. Note that, $L_x \subseteq L_p$. Since $L_x \subseteq L_v$, e' is an edge of $E(S) \setminus E(S')$. Consider any tree T which has an edge f_1 in $\Psi'(f)$. We will show that $\mathcal{L}(T) \cap L_x = \mathcal{L}(T) \cap L_p$. It then follows that $f_1 \in \Psi(e')$ and thus, $\Psi'(f) \subseteq \Psi(e')$.

Since $L_x \subseteq L_p$, $(L_x \cap \mathcal{L}(T)) \subseteq (\mathcal{L}(T) \cap L_p)$. By Observation 3, all the labels in $\mathcal{L}(T) \cap L_v$ are in the same connected component of $G(\mathcal{P}) - \Psi(e)$. Thus, all the labels in $L_x \cup (L_p \cap \mathcal{L}(T))$ are in the same connected of $G(\mathcal{P}) - \Psi(e)$. If $(L_p \cap \mathcal{L}(T)) \not\subseteq (L_x \cap \mathcal{L}(T))$, then $S|(L_x \cup (L_p \cap \mathcal{L}(T))$ and $S'|(L_x \cup (L_p \cap \mathcal{L}(T))$ are not isomorphic, a contradiction of Observation 4. Thus, $(L_p \cap \mathcal{L}(T)) \subseteq (L_x \cap \mathcal{L}(T))$.

Assume that $\Psi(e')$ is a minimal cut of $G(\mathcal{P})$. Then, all the labels in L_p are in the same connected component of $G(\mathcal{P}) - \Psi(e')$. By Observation 4, it follows that $L_p = L_x$. Thus, $\Psi'(f)$ is also a minimal cut of $G(\mathcal{P})$. □

Let S be an AST of $\mathcal{P}$ and let Ψ be the cut function of S. Let e be an edge of S where $\Psi(e)$ is not a minimal cut of $G(\mathcal{P})$ and $|\Psi(e)|$ is the maximum of all such edges. Then, there exists a vertex $u \in e$ where the split of edge e at u is valid. Split e at u. Let S' be the derived tree and let Ψ' be the cut function of S'. Let P be the set of all edges in S such that for every $x \in P$, $\Psi(x)$ is not a minimal cut and $|\Psi(x)| = |\Psi(e)|$. Similarly let P' be the set of all edges in S' such that for every $x \in P'$, $\Psi'(x)$ is not a minimal cut and $|\Psi'(x)| = |\Psi(e)|$. We have the following two lemmas.

Lemma 34. *For every edge $f \in E(S')$, if $|\Psi'(f)| > |\Psi(e)|$ then $\Psi'(f)$ is a minimal cut of $G(\mathcal{P})$.*

Proof. Consider any edge $f \in E(S')$ where $|\Psi'(f)| > |\Psi(e)|$. If $f \in E(S) \cap E(S')$, then $\Psi(f) = \Psi'(f)$. Since $|\Psi(f)| > |\Psi(e)|$, by assumption $\Psi(f)$ is a minimal cut of $G(\mathcal{P})$. Thus, $\Psi'(f)$ is also a minimal cut of $G(\mathcal{P})$. Assume that $f \in E(S') \setminus E(S)$. By Lemma 33, there exists an edge $e' \in E(S)$ where $\Psi'(f) \subseteq \Psi(e')$. Since $|\Psi'(f)| > |\Psi(e)|$, $|\Psi(e')| > |\Psi(e)|$. Thus, by assumption $\Psi(e')$ is a minimal cut of $G(\mathcal{P})$. From Lemma 33, it follows that $\Psi(e') = \Psi'(f)$ and $\Psi'(f)$ is a minimal cut of $G(\mathcal{P})$. □

Lemma 35. $|P'| < |P|$.

Proof. Let $Q = P \cap (E(S) \setminus E(S'))$ and $Q' = P' \cap (E(S') \setminus E(S))$. It suffices to show that $|Q'| < |Q|$. Consider any edge $f \in Q'$. By Lemma 33, there exists an edge $e' \in E(S) \setminus E(S')$ where $\Psi'(f) \subseteq \Psi(e')$. Thus, $|\Psi(e')| \geq |\Psi'(f)|$. If $|\Psi(e')| > |\Psi'(f)|$, then by assumption $\Psi(e')$ is a minimal cut and thus by Lemma 33 $|\Psi(e')| = |\Psi'(f)|$, a contradiction. Thus, $\Psi(e') = \Psi'(f)$. Also, since $\Psi'(f)$ is not a minimal cut, by Lemma 33, $\Psi(e')$ is not a minimal cut. If $e' = e$, it would imply that all vertices of V_v are in the same connected component of $G(\mathcal{P}) - \Psi(e)$ and thus contradicts the assumption that the split of e at u is valid. Thus, $e' \neq e$. Hence, we can conclude that for every edge $f \in Q'$, there exists an edge $e' \in (Q \setminus \{e\})$, where $\Psi'(f) = \Psi(e')$.

Let f_1 and f_2 be any two distinct edges in Q'. Let e_1 and e_2 be the edges of $Q \setminus \{e\}$ where $\Psi'(f_1) = \Psi(e_1)$ and $\Psi'(f_2) = \Psi(e_2)$. If $e_1 = e_2$, then $\Psi'(f_1) = \Psi'(f_2)$, which is a contradiction of Observation 2. Thus, $e_1 \neq e_2$. Since $e \in Q$ and $e \notin Q'$, it then follows that $|Q'| \leq |Q| - 1$ and thus, $|Q'| < |Q|$. □

Lemma 36. *If $\mathcal{P}$ has an AST, then there exists an AST S of $\mathcal{P}$ satisfying the following. Let Ψ be the cut function of S. For every edge $e \in S$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$.*

Proof. Let S be an AST of $\mathcal{P}$ and let Ψ be the cut function of S. Assume that there exists an edge f of S where $\Psi(f)$ is not a minimal cut of $G(\mathcal{P})$. We do the following repeatedly till $\Psi(f)$ is a minimal cut for every edge f of S. Let e be an edge of S where, $\Psi(e)$ is not a minimal cut of $G(\mathcal{P})$ and $|\Psi(e)|$ is the maximum of all such edges. Then, there exists a vertex $x \in e$ where the split of edge e at x is valid. Split e at x. Let S' be the derived tree and let Ψ' be the cut function of S'. Set S to S' and Ψ to Ψ'.

Let s be the total number of iterations. By Lemma 32, in each iteration the derived tree S' is an AST of $\mathcal{P}$. We only need to prove that s is finite. Number of vertices in an AST of $\mathcal{P}$ is at most $2|\mathcal{L}(\mathcal{P})|$. Also, the minimum size of $\Psi(e)$ for an edge e of S is 1. It thus follows from Lemmas 34 and 35 that s is finite. □

Proof of Theorem 15. ($\leftarrow$) Assume that $\mathcal{P}$ has an AST. By Lemma 36, there exists an AST S of $\mathcal{P}$ satisfying the following. Let Ψ be the cut function of S. For every edge $e \in E(S)$, $\Psi(e)$ is a minimal cut of $G(\mathcal{P})$. Let $\mathcal{F}$ be the set of all minimal cuts of $G(\mathcal{P})$ where if e is an internal edge of S then $\Psi(e) \in \mathcal{F}$. For every $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$, by definition of Ψ, at most

one edge of T is in F. We will now prove that $\mathcal{F}$ is a complete set of pairwise parallel legal minimal cuts of $G(\mathcal{P})$.

We first prove that every cut in $\mathcal{F}$ is legal. Consider any $F \in \mathcal{F}$. Let $e = \{u, v\}$ be the internal edge of S where $\Psi(e) = F$. Let T be an input tree which has an internal edge f in $\Psi(e)$. Since e is an internal edge at least one such input tree exists. Otherwise $\Psi(e)$ is not a minimal cut. By definition of a cut function, f is the only edge of T in $\Psi(e)$. Thus, each of the two connected components of $G(\mathcal{P}) - \Psi(e)$ have at least one non-internal edge of T. So, F is a legal minimal cut of $G(\mathcal{P})$.

To prove that cuts in $\mathcal{F}$ are pairwise parallel, we will prove that for any two distinct internal edges e_1 and e_2 of S, $\Psi(e_1)$ and $\Psi(e_2)$ are parallel. There exist vertices $x \in e_1$ and $y \in e_2$ where $L_x \subseteq L_y$. We will show that for every $f \in \Psi(e_1)$, $f \in \Psi(e_2)$ or $f \subseteq V_y$. It then follows that $\Psi(e_1)$ and $\Psi(e_2)$ are parallel. Let f be an edge of input tree T in $\Psi(e_1)$. Then there exists $z \in f$ where $L_z \subseteq L_x$. Thus, $L_z \subseteq L_y$ and $z \in V_y$. By Lemma 31, all the vertices of V_y are in the same connected component of $G(\mathcal{P}) - \Psi(e_2)$. Thus, $f \in \Psi(e_2)$ or $f \subseteq V_y$.

Lastly, we will show that $\mathcal{F}$ is complete. Consider any internal edge $f = \{p, q\}$ of some input tree T. Since S is an AST of $\mathcal{P}$, there exists an edge $e = \{u, v\}$ where up to relabeling of sets $L_p \subseteq L_u$ and $L_q \subseteq L_v$. Thus, e is an agreement edge of S corresponding to f and so, $f \in \Psi(e)$. Since f is an internal edge, e will also be an internal edge of S and thus $\Psi(e) \in \mathcal{F}$. Hence, for every internal edge f of an input tree there exists a cut $F \in \mathcal{F}$ where $f \in F$. So, S is a complete set of pairwise parallel legal minimal cuts of $G(\mathcal{P})$.

($\rightarrow$) Assume that there exists a complete set $\mathcal{F}$ of pairwise parallel legal minimal cuts of $G(\mathcal{P})$ where for every $F \in \mathcal{F}$ and for every $T \in \mathcal{P}$, there is at most one edge of T in F. By Theorem 14, there exists an unrooted tree S where $\Sigma(\mathcal{F}) = \Sigma(S)$. We will prove that S is an AST of $\mathcal{P}$ by showing that $\Sigma(S|\mathcal{L}(T)) = \Sigma(T)$ for every input tree $T \in \mathcal{P}$.

Consider an input tree T of $\mathcal{P}$. Let $X_1|X_2$ be a non-trivial split of T corresponding to edge $f \in E(T)$. Since $\mathcal{F}$ is complete, there exists a cut $F \in \mathcal{F}$ where $f \in \mathcal{F}$. If $\sigma(F) = Y_1|Y_2$, by Lemma 27, up to relabeling of sets $X_i \subseteq Y_i$ for every $i \in \{1, 2\}$. Since $\sigma(F)$ is a split of S, it implies that $\Sigma(T) \subseteq \Sigma(S|\mathcal{L}(T))$.

Consider any non-trivial split $P_1|P_2$ of $\Sigma(S)$ where $P_i \cap \mathcal{L}(T) \neq \emptyset$ for every $i \in \{1, 2\}$. Let

$Q_i = P_i \cap \mathcal{L}(T)$ for every $i \in \{1, 2\}$. Since $\Sigma(S) = \Sigma(\mathcal{F})$, there exists a cut $F \in \mathcal{F}$ where $\sigma(F) = P_1|P_2$. Since P_1 and P_2 are in different connected components of $G(\mathcal{P}) - F$, Q_1 and Q_2 are also in different connected components of $G(\mathcal{P}) - F$. Thus, there exists an edge f' of T in F. Since F does not contain any other edge of T, $\sigma(f') = Q_1|Q_2$ and thus $\Sigma(S|\mathcal{L}(T)) \subseteq \Sigma(T)$ □

4.6 Relationship to Legal Triangulation

Let $\mathcal{P}$ be a profile of phylogenetic trees. Theorems 4 and 12 together imply that if $G(\mathcal{P})$ has a complete set $\mathcal{F}$ of pairwise parallel legal minimal cuts, there also exists a legal triangulation of $G(\mathcal{P})$. As shown in Vakati and Fernández-Baca (2011), a legal triangulation of $G(\mathcal{P})$ can be derived from a compatible tree of $\mathcal{P}$. In this section, we show how to derive a legal triangulation of $G(\mathcal{P})$ directly from $\mathcal{F}$ without building a compatible tree. This shows the relationship between complete sets of pairwise parallel legal minimal cuts and legal triangulations of display graphs. By Theorem 11 and Lemma 25, this also shows the relationship between restricted triangulations of edge $LG(\mathcal{P})$ and legal triangulations of $G(\mathcal{P})$.

A complete set $\mathcal{F}$ of pairwise parallel legal minimal cuts of $G(\mathcal{P})$ is *minimal* if no proper subset of $\mathcal{F}$ is also complete. Let $\mathcal{F}$ be a minimal complete set of pairwise parallel legal minimal cuts of $G(\mathcal{P})$.

At a high level, we construct a legal triangulation of $G(\mathcal{P})$ from $\mathcal{F}$ as follows. Consider any cut $F \in \mathcal{F}$. We build a pair $D_F = (X, Y)$ where X and Y are subsets of $E(F)$ and are vertex separators of $G(\mathcal{P})$. Let A and B be the connected components of $G(\mathcal{P}) - F$. Also, let A', B' be the subgraphs induced in $G(\mathcal{P})$ by the vertex sets $V(A) \cup \{X \cap Y\}$ and $V(B) \cup \{X \cap Y\}$ respectively. To legally triangulate $G(\mathcal{P})$ we first triangulate the subgraph of $G(\mathcal{P})$ induced by the vertex set $X \cup Y$ and then triangulate the subgraphs A' and B'. To triangulate either of those subgraphs, we again use vertex separators built from endpoints of a different cut. We make sure that, for every set D_I for some $I \in \mathcal{F}$ built after D_F, both the sets of D_I are subsets of either $V(A')$ or $V(B')$ but not both.

We now give the details of our construction. We consider the elements of $\mathcal{F}$ in some arbitrary, but fixed order, and use a set W to record all such cuts $F \in \mathcal{F}$ for which D_F has already been constructed. Initially W is empty. For each successive cut F in $\mathcal{F}$, we do the

following. Let $F' \subseteq F$ be the set of all internal edges $e \in F$ such that e is the only edge of the tree containing e that is in F. Let A and B be the two connected components of $G(\mathcal{P}) - F$. Let $X = V(A) \cap V(F')$ and $Y = V(B) \cap V(F')$. For every edge e of F' whose endpoints are in different sets of some set D_I where $I \in W$, we do the following. Let Q be the connected component of $G(\mathcal{P}) - I$ where $E(Q) \cap F \neq \emptyset$. Note that Q is the only such component of $G(\mathcal{P}) - I$. Let v be the vertex of e in Q. Replace the endpoints of e in sets X and Y by v. For every non internal edge $f \in F$ where f is the only edge of the tree containing f that is in F, add the internal vertex of f to both sets X and Y. If there exists a tree T where more than one edge of T is in F, add the common endpoint of all the edges of T in F to both sets X and Y. Set D_F to (X, Y). Add F to W.

For every $F \in \mathcal{F}$, let O_F be the set defined as follows. Let $D_F = (X, Y)$ and let $X = \{x_1, \cdots, x_m, z_1, \cdots, z_p\}$ and $Y = \{y_1, \cdots, y_m, z_1, \cdots, z_p\}$, where $m > 0$, $p \geq 0$ and for every $i \in [m]$, $\{x_i, y_i\}$ is an internal edge of $G(\mathcal{P})$. Then, O_F consists of sets $\{x_1, \cdots, x_j, y_j, \cdots, y_m, z_1, \cdots, z_p\}$ for every $j \in [m]$.

Let G' be the graph derived from $G(\mathcal{P})$ as follows. For every cut $F \in \mathcal{F}$ where $D_F = (X, Y)$, add edges to make each of the sets X and Y a clique. For every cut $F \in \mathcal{F}$ and for every $Y \in O_F$, add edges to make Y a clique. For every leaf ℓ, make the vertices of $N_{G(\mathcal{P})}(\ell)$ a clique.

Theorem 17. *G' is a legal triangulation of $G(\mathcal{P})$.*

To prove Theorem 17 we first prove few useful lemmas. For every cut $F \in \mathcal{F}$ where $D_F = (X, Y)$, we denote the sets $X \cup Y$, $X \cap Y$ by $F_\cup$ and $F_\cap$ respectively. For any internal edge e, we call the cut $F \in \mathcal{F}$ a *differentiating* cut of e if e's endpoints are in different sets of D_F. Note that, since $\mathcal{F}$ is minimal, every cut in $\mathcal{F}$ is a differentiating cut of some internal edge. A clique of G' is *illegal* if it contains a fill-in edge with a leaf vertex as an endpoint or if it contains an internal edge along with any another edge of $G(\mathcal{P})$. Graph G' is a legal triangulation if and only if G' does not contain an illegal clique.

Lemma 37. *Let F and I be two distinct cuts of $\mathcal{F}$. Let x be a vertex where $x \in F_\cup$ and x is in the connected component of $G(\mathcal{P}) - I$ which does not contain edges of F. Then, $x \in I_\cap$.*

Proof. Let E_F be the set of all edges of F that have x as an endpoint and let E_I be the set of all edges of I that have x as an endpoint. Since x is in $F_\cup$ and in the component of $G(\mathcal{P}) - I$ which does not contain edges of F, $E_F \subseteq E_I \subseteq I$. If $|E_I| > 1$, then $x \in I_\cap$. Assume that $|E_I| = 1$ and let $e = \{x, y\}$ be the edge with endpoint x in I. Since $E_F \subseteq E_I$ and $E_F \geq 1$, $e \in F$ and $|E_F| = 1$.

If y is a leaf vertex, then $x \in I_\cap$, so assume that y is not a leaf vertex. Let E_y represent the set of edges of I with y as an endpoint. If $|E_y| > 1$, then $x \notin F_\cup$ since $E_y \subseteq F$. Thus, $|E_y| = 1$. Let J be the cut that differentiates edge e. If $F = J$ then by construction, $x \in I_\cap$. Thus, assume that $F \neq J$. If J is in the same connected component of $G(\mathcal{P}) - F$ as I, then by construction $x \notin F_\cup$, which is a contradiction. Thus, J is in the connected component of $G(\mathcal{P}) - F$ which does not contain I and by construction, $x \in I_\cap$. □

Lemma 38. *Let $D_F = (X, Y)$ for some $F \in \mathcal{F}$. Let A and B be the connected components of $G(\mathcal{P}) - F$ where $\{X \setminus F_\cap\} \subseteq V(A)$ and $\{Y \setminus F_\cap\} \subseteq V(B)$. There does not exist an edge $\{u, v\}$ in G' where*

1. *$u \in V(A) \setminus F_\cap$ and $v \in V(B) \setminus Y$, or*
2. *$u \in V(B) \setminus F_\cap$ and $v \in V(A) \setminus X$*

Proof. Assume that there exists an edge $e = \{u, v\}$ in G' which satisfies one of the two conditions. Without loss of generality, assume that $u \in V(A) \setminus F_\cap$ and $v \in V(B) \setminus Y$. If $e \in E(G(\mathcal{P}))$, then $e \in F$ and hence, by construction, at least one of u and v should be in $F_\cup$. But $v \notin Y$ and so, $u \in F_\cap$ which is a contradiction. Thus, e is a fill-in edge. Note that $e \not\subseteq F_\cup$. So, by construction there exists a cut $I \in \mathcal{F}$ where $I \neq F$ and $e \subseteq I_\cup$.

If $E(A) \cap I \neq \emptyset$, then by Lemma 37, $v \in F_\cap$, which is a contradiction. Thus, assume that $E(B) \cap I \neq \emptyset$. Then, by Lemma 37, $u \in F_\cap$ which is a contradiction. Thus, such an edge e cannot exist. □

Lemma 39. *Let F be a cut of $\mathcal{F}$ and let H represent the subgraph of G' induced by vertices of $F_\cup$. Then, we have the following.*

1. *H is triangulated.*

2. The is no illegal clique in H.

Proof. Let $D_F = (X, Y)$ and let A, B be the connected components of $G(\mathcal{P}) - F$. Let $X = \{x_1, \cdots, x_m, z_1, \cdots, z_p\}$ and $Y = \{y_1, \cdots, y_m, z_1, \cdots, z_p\}$ where for every $i \in [m]$, $x_i \in V(A)$, $y_i \in V(B)$ and $\{x_i, y_i\}$ is an internal edge of $G(\mathcal{P})$.

We will first prove that for any $i \in [m]$, $j \in [m]$ where $i > j$, there does not exist an edge $e = \{x_i, y_j\}$ in H. Assume that $e \in E(H)$. Edges $e_1 = \{x_i, y_i\}$ and $e_2 = \{x_j, y_j\}$ are differentiated by F. Thus, e is not in $E(G(\mathcal{P}))$ and is a fill-in edge. Since there does not exist a set in O_F which contains both x_i and y_j, there exists a cut $I \in \mathcal{F}$ where $e \subseteq I_\cup$. Since F and I are parallel, edges of I are either in component A or B but not both. Assume that $I \cap E(A) \neq \emptyset$. Then by Lemma 37, $y_j \in F_\cap$, which is a contradiction. Similarly, if $I \cap E(B) \neq \emptyset$, then by Lemma 37, $x_i \in F_\cap$ which is a contradiction. Thus, there cannot be such a fill-in edge e.

Let C be a chordless cycle of length at least four in H. Sets X and Y are cliques in G'. Thus, if C contains more than two vertices from one of X or Y, C must contain a chord. Hence, C has exactly four vertices and contains exactly two vertices each from X and Y. Note that C cannot contain vertex z_i for any $i \in [p]$. Let x_i,x_j be the vertices of X in C where $1 \leq i < j \leq m$. Similarly, let $y_{i'}$,$y_{j'}$ be the vertices of Y in C where $1 \leq i' < j' \leq m$. We have the following cases. If $i \leq i'$, then $\{x_1, \cdots, x_i, y_i, \cdots, y_m, z_1, \cdots, z_p\} \in O_F$ and thus vertices $x_i, y_{i'}, y_{j'}$ form a clique. Hence, C is not chordless which is a contradiction. If $i > i'$, then from the above argument neither of the edges $\{x_i, y_{i'}\}$ and $\{x_j, y_{i'}\}$ can exist. Thus, vertex $y_{i'}$ cannot be in C which is a contradiction.

Assume that H contains an illegal clique H'. Thus, H' contains two internal edges e and e'. By construction, $F_\cup$ cannot contain a leaf vertex. By legality of cuts and from the construction of $F_\cup$, edges e and e' are from different input trees and both are differentiated by F. Let $e = \{x_i, y_i\}$ for some $i \in [m]$ and let $e' = \{x_j, y_j\}$ for some $j \in [m]$. Without loss of generality, assume that $i < j$. As proved above, there cannot exist an edge between vertices x_j and y_i in H and thus H' is not a clique which is a contradiction. Thus, H does not contain an illegal clique. □

Lemma 40. *G' is chordal.*

Proof. Assume the contrary. Let C be a chordless cycle of length at least 4 in G'. By construction, C cannot contain a leaf vertex. We have the following cases.

Suppose that there exist vertices $\{u, v\} \subset V(C)$ and a cut $F \in \mathcal{F}$ where if $D_F = (X, Y)$, then $u \in X \setminus F_\cap$ and $v \in Y \setminus F_\cap$. Let A, B be the connected components of $G(\mathcal{P}) - F$ where $u \in V(A)$ and $v \in V(B)$. We have two cases.

Case 1: C contains a vertex $x \in F_\cap$. Then, there exists a path u, x, v in C. Because C is a cycle, there must exist an edge between a vertex $u' \in V(A) \setminus x$ and $v' \in V(B) \setminus x$. Since C is chordless, $u' \notin F_\cap$ and $v' \notin F_\cap$. Thus, $u' \in V(A) \setminus F_\cap$ and $v' \in V(B) \setminus F_\cap$. By Lemma 38, if $u' \in V(A) \setminus X$ then there cannot exist an edge between u' and v'. Thus, $u' \in X \setminus F_\cap$. Similarly, $v' \in Y \setminus F_\cap$. If $u \neq u'$ or $v \neq v'$, C cannot be chordless. Thus, $u = u'$ and $v = v'$ and C is a chordless cycle of length 3 which is a contradiction.

Case 2: C does not contain a vertex of $F_\cap$. Since $u \in V(A) \setminus F_\cap$, $v \in V(B) \setminus F_\cap$ and F is a cut, there must exist two edges $e_1 = \{x_1, y_1\}$ and $e_2 = \{x_2, y_2\}$ in C where $\{x_1, x_2\} \subseteq V(A) \setminus F_\cap$ and $\{y_1, y_2\} \subseteq V(B) \setminus F_\cap$. If $x_1 \in V(A) \setminus X$, then by Lemma 38 there cannot exist an edge between x_1 and y_1. Thus, $x_1 \in X \setminus F_\cap$. Similarly, $x_2 \in X \setminus F_\cap$ and $\{y_1, y_2\} \subseteq Y \setminus F_\cap$. Since vertex sets of X and Y are cliques in G', there exist edges $\{x_1, x_2\}$ and $\{y_1, y_2\}$. Thus, there cannot exist any other vertex in C and hence $V(C) \subseteq F_\cup$. But, by Lemma 39 subgraph of G' induced by vertices of $F_\cup$ is triangulated. Thus, C is not chordless which is a contradiction.

Now assume that for every cut $F \in \mathcal{F}$, where $D_F = (X, Y)$, there do not exist two vertices $u, v \in V(C)$ where $u \in X \setminus F_\cap$ and $v \in Y \setminus F_\cap$. This also implies that, for every cut $F \in \mathcal{F}$ at most two vertices of $V(C)$ are in $F_\cup$. Let x_1, x_2, x_3, x_4 be a path of length four in C. Also, for every $i \in \{1, 2, 3\}$, let $F^{(i)} \in \mathcal{F}$ be the cut where $\{x_i, x_{i+1}\} \subseteq F^{(i)}_\cup$. Note that such cuts must exist and must be distinct. For every $i \in \{1, 2, 3\}$, let A_i and B_i be the connected components of $G(\mathcal{P}) - F^{(i)}$. Without loss of generality, assume that $E(A_1) \cap F^{(2)} \neq \emptyset$ and $E(B_2) \cap F^{(1)} \neq \emptyset$. We have the following cases.

Case 1: $F^{(3)} \cap E(A_2) \neq \emptyset$. If $x_1 \in A_2$, then by Lemma 37, $x_1 \in F^{(2)}_{\cap}$ and C is not chordless, which is a contradiction. Thus, $x_1 \in B_2$. Similarly, if $x_4 \in B_2$, by Lemma 37, $x_4 \in F^{(2)}_{\cap}$ and C is not chordless, which is a contradiction. Thus, $x_4 \in A_2$. Since C is a cycle, $F^{(2)}$ is a minimal cut and $\{F^{(2)}_{\cup} \setminus \{x_2, x_3\}\} \cap V(C) = \emptyset$, there exists an edge $\{v_1, v_2\}$ in C where $v_1 \in V(A_2) \setminus F^{(2)}_{\cup}$ and $v_2 \in V(B_2) \setminus F^{(2)}_{\cup}$. But, by Lemma 38, such an edge cannot exist.

Case 2: $F^{(3)} \cap E(A_1) \neq \emptyset$ and $F^{(3)} \cap E(B_2) \neq \emptyset$. Without loss of generality let A_3, B_3 be the connected components of $G(\mathcal{P}) - F^{(3)}$ that contain $F^{(2)}$ and $F^{(1)}$ respectively. Assume that $x_2 \in A_3$. Since $x_2 \in F^{(1)}_{\cup}$, by Lemma 37, $x_2 \in F^{(3)}_{\cap}$. Then, there exists an edge $\{x_2, x_4\}$ and C is not chordless, which is a contradiction. Thus, $x_2 \in B_3$. But $x_2 \in F^{(2)}_{\cup}$ and thus, by Lemma 37, $x_2 \in F^{(3)}_{\cap}$. Hence, there exists a chord $\{x_2, x_4\}$ and C is not chordless, which again is a contradiction.

Case 3: $F^{(3)} \cap E(B_1) \neq \emptyset$. Renaming vertices x_1, x_2, x_3 and x_4 as, x_4, x_3, x_2 and x_1, respectively, brings us back to case 2.

Thus, G' does not contain a chordless cycle of length 4 or greater and hence G' is chordal. □

Proof of Theorem 17. From Lemma 40, G' is triangulated. We now prove that triangulation G' is legal.

By construction, we do not add any fill-in edge with a leaf vertex as an endpoint. Thus, condition $(LT2)$ is true for G'. Assume that there exists a clique H with two internal edges $e = \{x_1, y_1\}$ and $e' = \{x_2, y_2\}$. Let F be the cut which differentiates e. Let A and B be the connected components of $G(\mathcal{P}) - F$ where $x_1 \in V(A)$ and $y_1 \in V(B)$. By Lemma 39, both the endpoints of e' cannot be in $F_{\cup}$. Without loss of generality, assume that $x_2 \notin F_{\cup}$ and $x_2 \in A$. Since $x_2 \notin F_{\cup}$ and $y_1 \notin F_{\cap}$, by Lemma 38, there cannot exist an edge between x_2 and y_1 in G'. Thus, H is not a clique of G' which is a contradiction. □

4.7 Conclusion

We have shown that the characterization of tree compatibility in terms of restricted triangulations of the edge label intersection graph transforms into a characterization in terms of minimal cuts in the display graph. These two characterizations are closely related to the legal triangulation characterization of Vakati and Fernández-Baca (2011). Given a complete set of pairwise parallel legal minimal cuts of a display graph, a legal triangulation of the display graph can derived without building the compatible tree. We also derived characterizations of the agreement supertree problem in terms of minimal cuts and minimal separators of the display and edge label intersection graphs respectively.

It is not known if the agreement supertree problem is fixed parameter tractable when parametrized by the number of input trees. It remains to be seen whether any of these characterizations can be exploited to derive an explicit fixed parameter algorithm for the tree compatibility and agreement supertree problems when parametrized by the number of trees.

CHAPTER 5. FIXED-PARAMETER ALGORITHMS FOR AGREEMENT SUPERTREES

David Fernández-Baca, Sylvain Guillemot, Brad Shutters, Sudheer Vakati

Modified from a paper submitted to the journal *SIAM. J. Comput.*

Abstract

We study the agreement supertree approach for combining rooted phylogenetic trees when the input trees do not fully agree on the relative positions of the taxa. Two approaches to dealing with such conflicting input trees are considered. The first is to contract a set of edges in the input trees so that the resulting trees have an agreement supertree. We show that this problem is NP-complete and give an FPT algorithm for the problem parameterized by the number of input trees and the number of edges contracted. The second approach is to remove a set of taxa from the input trees so that the resulting trees have an agreement supertree. An FPT algorithm for this problem when the input trees are all binary was given by Guillemot and Berry (2010). We give an FPT algorithm for the more general case when the input trees have arbitrary degree.

5.1 Introduction

A phylogeny, or evolutionary tree, is a tree representing the evolutionary history of a set of species. The leaves of the tree represent the current species (taxa), and the internal nodes of the tree represent the hypothetical ancestors. A fundamental problem in phylogenetics is to construct a supertree from smaller input trees with overlapping taxa in such a way that the inferred supertree complies as closely as possible with the topological information of the input

trees. This problem is motivated by the biological and computational constraints on constructing large scale phylogenies. The supertree problem was introduced in Gordon (1986), and a variety of supertree construction methods have been proposed. See [Bininda-Emonds (2004); Scornavacca (2009); Aho et al. (1981); Steel (1992); Bryant (2001)] for more on supertrees.

In this paper we use the agreement supertree approach for combining rooted phylogenetic trees. The goal of this approach is to search for a supertree such that each of the input trees is a restriction of the supertree to a subset of its taxa. Formally, we have the following decision problem.

Agreement Supertree (AST)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa.

Question: Does there exist an agreement supertree for $\mathcal{T}$?

The answer to an instance of AST is "yes" if and only if the input trees fully agree on the relative positions of the taxa, in which case the input trees are said to agree. There is a polynomial time algorithm for the AST problem, which returns an agreement supertree if one exists [Ng and Wormald (1996)].

The input trees may fail to have an agreement supertree because of conflicts with respect to the relative positions of some taxa. Such conflicts arise due to errors in the inference process, or due to biological processes, e.g., lateral gene transfer, gene duplication, and others [Maddison (1997); Linder and Rieseberg (2004)]. Here we consider two approaches for dealing with conflicting input trees. The first addresses the case where conflict is due to unnecessary edges in the input trees. The goal is to find a subset of the edges of the input trees to contract so that the resulting collection of trees agree. Formally, we focus on the following decision problem.

Agreement Supertree Edge Contraction (AST-EC)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa, and an integer p.

Question: Can we contract at most p internal edges of $\mathcal{T}$ so that the trees in $\mathcal{T}$ agree?

The AST-EC problem does not seem to have been considered before. We prove that problem is NP-complete, and show that it is fixed-parameter tractable for parameters k and p.

The second approach we study addresses the case where disagreement is due to misplaced taxa. The goal is to find a subset of the taxa to remove from the input trees so that the resulting collection of trees agree. Formally, we focus on the following decision problem.

Agreement Supertree Taxon Removal (AST-TR)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa, and an integer p.

Question: Can we remove at most p taxa so that the input trees agree?

The AST-TR problem is NP-complete [Jansson et al. (2005); Berry and Nicolas (2007)], but was shown to be fixed-parameter tractable in k and p when restricted to the case when the input trees are all binary [Guillemot and Berry (2010)]. Our contribution is to show that the more general AST-TR problem, where the input trees are allowed to have arbitrary degree, is fixed-parameter tractable in k and p. It was also shown in Berry and Nicolas (2007) that if AST-TR is parameterized by only k or p, then the problem is fixed-parameter intractable. We also note that the optimization version of AST-TR, i.e., finding a minimum set of taxa to remove, is the dual of the Maximum Agreement Supertree (Smast) problem [Berry and Nicolas (2007); Jansson et al. (2005); Kao (2007)]. Exact algorithms for Smast on binary trees are known that run in time $O(6^k n^k)$ [Guillemot and Berry (2010); Hoang and Sung (2011)] and, when the maximum degree of the input trees is d, Hoang and Sung (2011) gives an $O((kd)^{kd+3}2^k n^k)$ time algorithm for Smast.

The rest of this paper proceeds as follows. In Section 5.2, we give basic definitions needed for the remainder of the paper. In Section 5.3, we develop a characterization of when a set of input trees agree. We then use this characterization to develop an algorithm for testing agreement that solves the AST problem. We remark here that this algorithm could be easily modified to produce an agreement supertree when the set of input trees agree. If the algorithm answers in the negative, it returns a subset of the internal nodes of the trees in $\mathcal{T}$ encapsulating the taxa on which the trees in $\mathcal{T}$ disagree. In Section 5.4 we use these internal nodes to develop

$O((2k)^p kn^2)$ time algorithms to solve the AST-EC and AST-TR problems. We also prove the NP-completeness of the AST-EC problem by giving a reduction from MULTICUT to AST-EC. Section 5.5 contains proofs of some results from Sections 5.3 and 5.4, that were deferred for readability. We conclude in Section 5.6 with some ideas for future research.

5.2 Definitions

Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a collection of rooted phylogenetic trees, and let T be some arbitrary tree in $\mathcal{T}$. We use $V(T)$, $E(T)$, $\mathcal{I}(T)$, $\hat{E}(T)$, and $r(T)$ to denote the vertices, edges, internal vertices, internal edges, and root vertex of T respectively. We use $L(T)$ and $\mathcal{L}(T)$ to denote the leaves of T and the set of labels mapped to the leaves of T respectively. We write $\mathcal{L}(\mathcal{T})$ for $\bigcup_{i\in[k]} \mathcal{L}(T_i)$ and $\hat{E}(\mathcal{T}) = \bigcup_{i\in[k]} \hat{E}(T_i)$, where $[k]$ stands for $\{1, \ldots, k\}$. We represent that a vertex v is an ancestor of u in T by $u \leq_T v$. For two vertices u and v such that $u <_T v$, we write $\text{child}_T(v, u)$ to denote the child of v along the path from v to u in T. For each $u \in V(T)$, we use $\text{parent}(u)$, $\text{Ch}(u)$, $T(u)$, and $\mathcal{L}(u)$ to denote the parent of u, the children of u, the subtree of T rooted at u, and the set of labels mapped to the leaves of $T(u)$, respectively.

For a label set L, the *restriction* of T to L, denoted by $T|L$, is the minimal homeomorphic subtree of T connecting leaves with labels in L. For a set $L \subseteq \mathcal{L}(\mathcal{T})$, we write $\mathcal{T}|L$ for the collection $\{T_1|L, \ldots, T_k|L\}$ of trees in $\mathcal{T}$ restricted to L. For a set $F \subseteq \hat{E}(T)$ we use T/F to denote the tree obtained from T by contracting the edges of F. For a set $F \subseteq \hat{E}(\mathcal{T})$, we denote the set $\{T_1/F, ..., T_k/F\}$ by $\mathcal{T}/F$. Given two trees S and T where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$, T is an *induced subtree* of S if and only if $S|\mathcal{L}(T) = T$. Note that all degree two vertices in $S|\mathcal{L}(T)$ other than the root are assumed to be suppressed. An *agreement supertree* for $\mathcal{T}$ is a tree S such that $\mathcal{L}(S) = \mathcal{L}(\mathcal{T})$, and each T_i is an induced subtree of S. We say that the trees in $\mathcal{T}$ *agree* if and only if there is an agreement supertree for $\mathcal{T}$.

5.3 Solving the Agreement Supertree Problem

The main result of this section, presented in Section 5.3.3, is an $O(kn^2)$ time algorithm, called TESTAGREEMENT, that determines whether or not a collection $\mathcal{T}$ of phylogenetic trees

has an agreement supertree. The algorithm relies on a recursive characterization of agreement, based on an intersection graph that we define in Section 5.3.1. In Section 5.3.2, we use this graph to develop an algorithm GetSuccessors that decomposes an agreement problem into smaller subproblems, or reports that no such decomposition is possible. In the latter case, GetSuccessors returns a small set of internal nodes from the input collection $\mathcal{T}$ that, in a sense, obstruct agreement. GetSuccessors is thus at the core of TestAgreement, and the information it produces is essential for our algorithms for the AST-EC and AST-TR problems.

5.3.1 An auxiliary graph

A *position* π in $\mathcal{T}$ is a tuple $(v_1, v_2, \ldots, v_k)$ where each v_i is either a vertex from the tree T_i or the symbol $\perp$. A *reduced position* is a position where each component is an internal node or $\perp$. The *reduction of a position* π, denoted by $\pi\downarrow$, is derived by substituting every leaf vertex in π by $\perp$. We use $\pi_\top$, respectively $\pi_\perp$, to denote the *initial*, respectively *final*, positions where $v_i = r(T_i)$, respectively $v_i = \perp$, for each $i \in [k]$. We write $\mathcal{L}(\pi)$ for $\bigcup_{i\in[k]} \mathcal{L}(\pi[i])$. By an agreement supertree for π, we mean an agreement supertree for the collection of trees $\{T_1(\pi[1]), \ldots, T_k(\pi[k])\}$.

We now introduce an auxiliary graph $G(\mathcal{T}, \pi)$, defined in Guillemot and Berry (2010), which is useful for identifying an agreement supertree for the position π. We will look for a specific partition of this graph, called a *nice partition*, that allows us to break the problem into smaller subproblems, or to conclude that there is no solution. The vertex set of $G(\mathcal{T}, \pi)$ consists of the children of all the vertices in π, and there is an edge between two vertices u and v if and only if $\mathcal{L}(u) \cap \mathcal{L}(v) \neq \emptyset$. Note that the graph $G(\mathcal{T}, \pi)$ is only defined when π is a reduced position. In the rest of this paper, $G(\mathcal{T}, \pi)$ is denoted by $G = (V, E)$ and $V_i = \mathrm{Ch}(\pi[i])$ for each $i \in [k]$.

A subset $U \subseteq V$ is *nice* if, for each $i \in [k]$, U contains either zero, one, or all of the elements of V_i. A partition P of V is a *nice partition* of G if every set of P is nice, and, for every $\{C, C'\} \subseteq P$, C and C' are disconnected in G. The *successor of* π *w.r.t. a nice set* U, denoted

by π_U, is defined as:

$$\pi_U[i] = \begin{cases} \bot & \text{if } V_i \cap U = \emptyset \\ p & \text{if } V_i \cap U = \{p\} \\ \pi[i] & \text{if } |V_i \cap U| \geq 2 \end{cases}$$

for each $i \in [k]$. We write $\mathcal{L}(U)$ for $\bigcup_{v \in U} \mathcal{L}(v)$. We have the following relationship between $\mathcal{L}(U)$ and $\mathcal{L}(\pi_U)$ which follows from the definition of a successor position.

Lemma 41. *Let $U \subseteq V$ be a nice set, then $\mathcal{L}(U) = \mathcal{L}(\pi_U)$.*

In the following, we will focus on a specific nice partition of G that is minimal in a certain sense. For partitions P and Q of V, we say P is *finer* than Q, denoted $P \sqsubseteq Q$, if and only if, for every $C \in P$, there exists a $D \in Q$ such that $C \subseteq D$. Let $\mathcal{P}$ represent the set of all nice partitions of G, and let $(\mathcal{P}, \sqsubseteq)$ represent the poset formed by partitions of $\mathcal{P}$ ordered under $\sqsubseteq$.

Lemma 42. *$(\mathcal{P}, \sqsubseteq)$ has a unique minimal element.*

Proof. Assume, towards a contradiction, that P and Q are distinct minimal elements of $(\mathcal{P}, \sqsubseteq)$. Consider the set $P \sqcap Q$ defined as:

$$P \sqcap Q = \{C \cap D : C \in P, D \in Q\} \setminus \{\emptyset\}. \tag{5.1}$$

It is known that $P \sqcap Q$ is also a partition of V, s.t. $P \sqcap Q \sqsubset P$. We show that $P \sqcap Q$ is also in $\mathcal{P}$, which will contradict the minimality of P, Q.

Consider any $X \in P \sqcap Q$, and let us show that X is a nice set. We have $X = C \cap D$ for some $C \in P, D \in Q$. Fix $i \in [k]$, and assume that $|X \cap V_i| \geq 2$. Since C and D are nice sets, $V_i \subseteq C$ and $V_i \subseteq D$, and thus, $V_i \subseteq X$. As this holds for any $i \in [k]$, we conclude that X is a nice set.

Consider two distinct classes $X, X' \in P \sqcap Q$, and let us show that they are disconnected in G. We have $X = C \cap D$ for some $C \in P, D \in Q$, and $X' = C' \cap D'$ for some $C' \in P, D' \in Q$. As $X \neq X'$, we have either $C \neq C'$ or $D \neq D'$. In the first case, C, C' are disconnected, and in the second case D, D' are. We conclude that X, X' are disconnected. Hence, $P \sqcap Q$ is a nice partition of V. $\square$

We call the unique minimal element of $(\mathcal{P}, \sqsubseteq)$ the *minimum nice partition* of G. Suppose that the minimum nice partition P of G is a singleton. Let $K = \{i \in [k] : \pi[i] \neq \perp\}$. We say that a set $I \subseteq V$ is *interesting* for a reduced position π of $\mathcal{T}$ if both $|I \cap V_i| = 2$ for each $i \in K$, and there is a set $F \subseteq E$ such that all of the following conditions hold: (i) $|F| \leq 2|K| - 1$; (ii) for each $v \in I$, there exists an $e \in F$ such that $v \in e$; and (iii) the subgraph of G induced by F has a minimum nice partition that is a singleton. Intuitively, a set of interesting vertices certifies that the minimum nice partition of G has a unique class. As we will see in Section 5.3.3, this condition will guarantee that there is no agreement supertree for π. In this case, we say that π is an *obstructing position* for $\mathcal{T}$.

5.3.2 Finding successor positions and interesting vertices

We now present algorithm GetSuccessors, which takes as input a position π in a collection $\mathcal{T}$ of rooted phylogenetic trees, and finds the set Π of successor positions for each class in the minimum nice partition of G (see Algorithm 1). When the minimum nice partition of G is a singleton, the algorithm returns a set I of interesting vertices for π.

The central idea behind the GetSuccessors algorithm is that, for a given $\ell \in \mathcal{L}(\pi)$, all of the vertices in the set $S_\ell = \{v \in V : \ell \in \mathcal{L}(v)\}$ are connected, and hence, must be in the same class of the minimum nice partition. The algorithm builds the successor positions by iterating over each label ℓ and building a position for ℓ which is denoted by π_ℓ, by examining each vertex $v \in S_\ell$. If v is already covered by a position π', then π' will need to be merged with π_ℓ. Once this merge is completed, the position π' is no longer needed. For implementation efficiency, instead of deleting positions, we keep a flag $\mathsf{active}(\pi_\ell)$ for each position π_ℓ. If $\mathsf{active}(\pi_\ell)$ is set to true, then π_ℓ is one of the successor positions of the graph G restricted to only those labels which have already been processed by the algorithm. If $\mathsf{active}(\pi_\ell)$ is set to false, then it is no longer used by the algorithm. If v is not already covered by some position, then we simply add v to π_ℓ.

After merging a position π' with π_ℓ, it may be the case that π_ℓ contains multiple vertices from some input tree T. In such a case, the algorithm needs to merge with π_ℓ all of the positions covering any of the vertices from T. Furthermore, since each $\ell \in \mathcal{L}(\pi)$ can be in the subtree of

Input: A position π in a collection $\mathcal{T}$ trees.
Output: A tuple (Π, I) where Π is the set of successor positions of each class in the minimum nice partition of G, and when Π is a singleton, I is a set of interesting vertices for the unique $\pi \in \Pi$.

foreach $\ell \in \mathcal{L}(\pi)$ **do** $S_\ell \leftarrow \emptyset$
foreach $i \in [k]$ **do**
  $\mathsf{position}(\pi[i]) \leftarrow \emptyset$
  foreach $v \in V_i$ **do**
    $\mathsf{position}(v) \leftarrow \emptyset$
    foreach $\ell \in \mathcal{L}(v)$ **do** $S_\ell \leftarrow S_\ell \cup \{v\}$
$\Pi \leftarrow \emptyset$; $I \leftarrow \emptyset$
foreach $\ell \in \mathcal{L}(\pi)$ **do**
  $\pi_\ell \leftarrow \pi_\perp$; $Z \leftarrow \emptyset$
  foreach $v \in S_\ell$ **do**
    if $\mathsf{position}(\pi[\mathsf{tree}(v)]) \neq \emptyset$ **then** $Z \leftarrow Z \cup \mathsf{position}(\pi[\mathsf{tree}(v)])$
    else if $\mathsf{position}(v) \neq \emptyset$ **then** $Z \leftarrow Z \cup \mathsf{position}(v)$
    else $\pi_\ell[\mathsf{tree}(v)] \leftarrow v$; $\mathsf{position}(v) \leftarrow \{\pi_\ell\}$
  while *there is a* $\pi_p \in Z$ *with* $\mathsf{active}(\pi_p) = \mathsf{true}$ **do**
    $\mathsf{active}(\pi_p) \leftarrow \mathsf{false}$
    foreach $i \in [k]$ *such that* $\pi_\ell[i] \neq \pi[i]$ *and* $\pi_p[i] \neq \perp$ **do**
      if $\pi_\ell[i] = \perp$ **then** $\pi_\ell[i] \leftarrow \pi_p[i]$; $\mathsf{position}(\pi_p[i]) \leftarrow \{\pi_\ell\}$
      else
        $I \leftarrow I \cup \{\pi_\ell[i], \pi_p[i]\}$; $\pi_\ell[i] \leftarrow \pi[i]$
        $\mathsf{position}(\pi[i]) \leftarrow \{\pi_\ell\}$
        foreach $w \in V_i$ **do** $Z \leftarrow Z \cup \mathsf{position}(w)$
  $\mathsf{active}(\pi_\ell) \leftarrow \mathsf{true}$
  $\Pi \leftarrow \Pi \cup \{\pi_\ell\}$
return $(\{\pi_\ell \in \Pi : \mathsf{active}(\pi_\ell) = \mathsf{true}\}, I)$

Algorithm 1: GetSuccessors$(\mathcal{T}, \pi)$

at most one v per V_i, it follows that the first time two vertices from the same input tree end up in the same partition, those two vertices are unique and we add them to the set I of interesting vertices.

If GetSuccessors determines that the minimum nice partition of G is a singleton, then the set Π returned has π as its only element. In this case, the set I of vertices returned by the algorithm is a set of interesting vertices for π.

For each vertex v that is either an element of position π or the child of a vertex in π, the algorithm keeps a reference $\mathsf{position}(v)$ that points to a set containing the active position containing v. This is determined in the following manner:

- if v is an element of the position π, then $\mathsf{position}(v)$ points to an active position that contains all of the children of v; and
- if v is a child of a vertex in π, then $\mathsf{position}(v)$ points to an active position that contains v.

Also, note that the set $\mathsf{position}(v)$ is pointing to is always either the empty set or a singleton. The purpose for using sets for $\mathsf{position}(v)$ is to simplify the code of lines 11, 12 and 21.

In the initialization phase of the algorithm (lines 1-6), for each label $\ell \in \mathcal{L}(\mathcal{T})$ we construct a set S_ℓ that contains all of the vertices $v \in V(\mathcal{T})$ such that both of the following conditions hold:

(i) $v \in V_i$ for some $i \in [k]$, i.e., v is a child of some vertex in position π; and

(ii) $\ell \in \mathcal{L}(v)$, i.e., ℓ is a label of the subtree rooted at v.

Also, since at the initialization phase, there are no active positions, the $\mathsf{position}$ references are all set to $\emptyset$.

The algorithm then turns to the construction phase (lines 7-23), where the positions π_ℓ are constructed. Note that for each vertex $v \in V(\mathcal{T})$, the algorithm also uses a function $\mathsf{tree}(v)$ that returns the unique index i such that $v \in V(T_i)$. The algorithm maintains two sets. Set Π will hold all of the successor positions created, and set I will hold the interesting vertices. The position π_ℓ is built in two phases. Naturally, each position π_ℓ is going to hold all those vertices of G that contain ℓ in their subtree, and these are precisely the vertices in the set S_ℓ. Thus, in the first phase (the loop in lines 10-13) the algorithm iterates over the elements of S_ℓ to ensure that they are included in the position. While doing so, the algorithm may discover new positions that need to be merged with π_ℓ and it stores these positions in a buffer Z. Now, for each $v \in S_\ell$ it performs the following tests in the specified order:

1. If $\mathsf{position}(\pi[\mathsf{tree}(v)])$ is non-empty, then it points to some position π' that contains all of the elements of V_i. Since $v \in V_i$, π' also contains v. Hence π' needs to be merged into π_ℓ, and so π' is added to Z.

2. If $\mathsf{position}(\pi[\mathsf{tree}(v)])$ is empty, but $\mathsf{position}(v)$ is non-empty, then $\mathsf{position}(v)$ points to some other position π' containing v, and hence π' needs to be merged into π_ℓ. Thus, π' is added to Z.

3. If both $\mathsf{position}(\pi[\mathsf{tree}(v)])$ and $\mathsf{position}(\pi[\mathsf{tree}(v)])$ are empty, then no position currently contains v, so we add v to π_ℓ.

Note that if tests 1 or 2 succeed, we do not immediately add v to position π_ℓ. That is, we add some position π' to Z, and so eventually π' will be merged with π_ℓ. After this happens, π_ℓ will contain v.

After processing the set S_ℓ, it may be the case that there are active components in Z. These components need to be merged with π_ℓ. This is done in the merge phase of the algorithm (lines 14-21). Let π_p be the position being merged with π_ℓ. Since π_p is being merged with π_ℓ, π_p will no longer represent a successor position, so the algorithm first sets $\mathsf{active}(\pi_p)$ to false. We now compare positions π_ℓ and π_p index by index. If $\pi_\ell[i] = \pi[i]$, then π_ℓ already contains all of the children of V_i, so no work needs to be done for index i. If $\pi_p[i] = \perp$, then no new vertices need to be added to π_ℓ for index i. Otherwise, either $\pi_\ell[i] = \perp$ or $\pi_\ell[i] \neq \perp$

Case 1. If $\pi_\ell[i] = \perp$, then either $\pi_p[i]$ is a single element of V_i, or $\pi_p[i] = \pi[i]$. In either case, to merge the two positions, we only need to copy the value of $\pi_p[i]$ to $\pi_\ell[i]$. Then, we need to update the $\mathsf{position}(\pi_p[i])$ to now refer to π_ℓ instead of π_p.

Case 2. If $\pi_\ell[i] \neq \perp$, then it must be the case that both $\pi_\ell[i]$ and $\pi_p[i]$ each contain an element of V_i and that these vertices are different. Thus, $\pi_\ell[i]$ now contains two elements of V_i and hence must contain all elements of V_i, so we set $\mathsf{position}(\pi[i])$ to π_ℓ, add the two vertices in $\pi_\ell[i]$ and $\pi_p[i]$ to the set of interesting vertices, and add any positions containing a child of V_i to Z, as they now also need to be merged with π_ℓ.

Lines 22 and 23 complete the process of constructing π_ℓ. This is done by first setting $\mathsf{active}(\pi_\ell)$ to true since it represents a successor position of G restricted to the labels processed by the algorithm so far. It then adds π_ℓ to the set Π containing all positions constructed so far.

The algorithm finishes in line 24, after all labels have been processed, by returning the set

of positions that remain active. We later formally prove (Theorem 18) that these are indeed the successor positions of G. The algorithm also returns the set I of vertices collected during the execution of the algorithm. If there is more than one active position in Π, then the set I of vertices has no meaningful value to us. However, as we show later (Theorem 19), if there is only one active position in Π, I is indeed a set of interesting vertices for π. The next two theorems summarize the essential properties of algorithm GetSuccessors. For technical reasons, their proofs are deferred to Section 5.5.1

Theorem 18. GetSuccessors *can be implemented to run in* $O(kn)$ *time, and in the tuple* (Π, I) *returned,* Π *is exactly the successor positions of each class of the minimum nice partition* P *of* G.

Theorem 19. *If the set* Π *returned by* GetSuccessors *is a singleton with* $\Pi = \{\pi\}$*, then* I *is a set of interesting vertices for* π.

5.3.3 Testing for an agreement supertree

We now present algorithm TestAgreement, which takes a position π in a collection $\mathcal{T}$ of phylogenetic trees, and decides if there is an agreement supertree for π (see Algorithm 2). If there is no agreement supertree for π, the algorithm returns an obstructing position π' and a set of interesting vertices for π'. To test for the existence of an agreement supertree for $\mathcal{T}$, it suffices to call TestAgreement on the initial position $\pi_\top$. The set of interesting vertices returned by TestAgreement will be used in the remainder of the algorithms discussed in this paper.

The algorithm TestAgreement proceeds in a recursive top-down fashion. If the position π is not reduced, it considers instead the reduced position $\pi\downarrow$, as justified by Lemma 43 below. Then, it calls GetSuccessors to compute the set Π of successor positions corresponding to the minimum nice partition of $G(\mathcal{T}, \pi)$. If Π has a single class, the algorithm concludes that there is no agreement supertree for π. Otherwise, it recursively checks for agreement on the successor positions $\pi' \in \Pi$. The correctness of this step follows from Lemma 45 below.

Lemma 43. *The following statements hold:*

Input: A position π in a collection $\mathcal{T}$ of trees.
Output: A tuple (B, π', I) where B is a boolean indicating whether there is an agreement supertree for π, and, when B is no, π' is an obstructing position and I is a set of interesting vertices for π'.

$\pi \leftarrow \pi\downarrow$
if $\pi = \pi_\perp$ **then return** $(\mathsf{yes}, \emptyset, \emptyset)$
$(\Pi, I) \leftarrow$ GetSuccessors$(\mathcal{T}, \pi)$
if $|\Pi| = 1$ **then return** (no, π, I)
foreach $\pi' \in \Pi$ **do**
  $(B, \pi'', I) \leftarrow$ TestAgreement$(\mathcal{T}, \pi')$
  if $B = no$ **then return** (no, π'', I)
return $(\mathsf{yes}, \emptyset, \emptyset)$

Algorithm 2: TestAgreement$(\mathcal{T}, \pi)$

1. *There is an agreement supertree for $\mathcal{T}$ if and only if there is an agreement supertree for every position π of $\mathcal{T}$.*

2. *There is an agreement supertree for a position π of $\mathcal{T}$ if and only if there is an agreement supertree for $\pi\downarrow$.*

Proof. Statement 1 holds because ($\Rightarrow$) for any agreement supertree S of $\mathcal{T}$ and any position π of $\mathcal{T}$, $S|\mathcal{L}(\pi)$ is an agreement supertree for π; and ($\Leftarrow$) any agreement supertree for $\pi_\top$ is an agreement supertree for $\mathcal{T}$. Statement 2 holds since ($\Rightarrow$) $\mathcal{L}(\pi\downarrow) \subseteq \mathcal{L}(\pi)$, so for any supertree S for π, $S|\mathcal{L}(\pi\downarrow)$ is a supertree for $\pi\downarrow$; and ($\Leftarrow$) for each $i \in [k]$ for which $\pi\downarrow[i] \neq \pi[i]$, we have that $\pi[i]$ is a leaf whose label can be added to a supertree for $\pi\downarrow$ (that is, we simply add each such leaf as child of the root, to get a supertree for π). □

We will need the following characterization of induced subtrees in terms of embeddings. This lemma follows from the definitions and is stated without proof.

Lemma 44. *Let S and T be two phylogenetic trees where $\mathcal{L}(T) \subseteq \mathcal{L}(S)$. The following statements are equivalent*

1. *T is an induced subtree of S.*

2. *There exists an injective function $\phi : V(T) \to V(S)$ with the following properties.*

   (a) *For every $p \in L(T)$, $\phi(p)$ is a leaf of S with the same label.*

(b) For every $p \in \mathcal{I}(T)$, if $q \in \mathrm{Ch}(p)$, then $\phi(q) <_S \phi(p)$. Also, for every $\{u, v\} \subseteq \mathrm{Ch}(p)$, $\mathrm{child}_S(\phi(p), \phi(u)) \neq \mathrm{child}_S(\phi(p), \phi(v))$.

We call ϕ an embedding of T into S.

The following lemma is the central result on which our recursive algorithm is based. Note that we only have to consider a reduced position π, according to Lemma 43.

Lemma 45. *Let π be a reduced position such that $\pi \neq \pi_\perp$. The following statements are equivalent.*

1. *There is an agreement supertree for π.*
2. *There exists a nice partition P of G where P has at least two classes, and, for every $X \in P$, π_X has an agreement supertree.*

Proof. ($\Longrightarrow$) Suppose that π has an agreement supertree S with $\mathcal{L}(S) = \mathcal{L}(\pi)$. By Lemma 44, for every $i \in [k]$ there exists an embedding ϕ_i of $T_i(\pi[i])$ into S. Let $r = r(S)$. As π is a reduced position different from $\pi_\perp$, we have $|\mathcal{L}(S)| = |\mathcal{L}(\pi)| \geq 2$, and thus $\mathrm{Ch}(r)$ is not empty. Let $\mathrm{Ch}(r) = \{u_1, \ldots, u_m\}$. We build a partition $P = \{C_1, \ldots, C_m\}$ of V as follows. For every $v \in V_i$ and $i \in [k]$, if $\phi_i(v) \leq_S u_{i'}$ for some $i' \in [m]$, then add v to $C_{i'}$. Note that there will always exist one such $u_{i'}$. We now show that P is a nice partition with at least two classes, and that for every $X \in P$, π_X has an agreement supertree.

As S is a phylogenetic tree, we have $m \geq 2$ and thus $|P| \geq 2$. Let us now show that P is a nice partition of G. Fix $i \in [k]$, and let $r_i = \pi[i]$. If $\phi_i(r_i) = r$, then by definition of an embedding the nodes $\phi_i(u)$ ($u \in V_i$) belong to distinct classes of P. On the other hand, if $\phi_i(r_i) \leq_S u_j$, then the nodes $\phi_i(u)$ ($u \in V_i$) all belong to C_j. Thus, every class of P is nice. Consider any $u \in C_i$ and $v \in C_j$ where $i \neq j$, with $u \in V_a$ and $v \in V_b$. Since $\mathcal{L}(u_i) \cap \mathcal{L}(u_j) = \emptyset$, $\phi_a(u) \leq_S u_i$ and $\phi_b(v) \leq_S u_j$, then $\mathcal{L}(u) \cap \mathcal{L}(v) = \emptyset$. Thus, vertices of C_i and C_j will be disconnected in G and P is a nice partition of G.

Lastly, for any $i \in [m]$, we show that $S(u_i)$ is an agreement supertree for π_{C_i}. Observe that $\mathcal{L}(S(u_i)) = \mathcal{L}(C_i)$, which is equal to $\mathcal{L}(\pi_{C_i})$ by Lemma 41. For any $j \in [k]$, we define an embedding ϕ'_j from $T_j(\pi_{C_i}[j])$ to $S(u_i)$ as follows. For every $v \in V(T_j(\pi_{C_i}[j]))$, set $\phi'_j(v) =$

$\phi_j(v)$. Since C_i is a nice set and ϕ_j is an embedding, ϕ'_j also satisfies all the properties of an embedding. Thus, $T_j(\pi_{C_i}[j])$ is an induced subtree of $S(u_i)$.

($\Longleftarrow$) Let $P = \{C_1, \ldots, C_m\}$ and let S_i represent the agreement supertree for π_{C_i} for every $i \in [m]$, such that $\mathcal{L}(S_i) = \mathcal{L}(\pi_{C_i})$. We build an agreement supertree S for π as follows. Add an edge from a vertex r to $r(S_i)$ for every $i \in [m]$. Set r to be the root of S. For any $1 \leq i \neq j \leq m$, the sets C_i and C_j are disconnected in G. Thus, $\mathcal{L}(C_i) \cap \mathcal{L}(C_j) = \emptyset$, and by Lemma 41 it follows that $\mathcal{L}(S_i) \cap \mathcal{L}(S_j) = \emptyset$. Furthermore, since P is a partition of V, we have $\bigcup_i \mathcal{L}(C_i) = \mathcal{L}(V)$. By Lemma 41, we deduce that $\bigcup_i \mathcal{L}(S_i) = \mathcal{L}(\pi)$. Thus, S is a phylogenetic tree with label set $\mathcal{L}(\pi)$. We will prove that S is an agreement supertree for π by showing that for any $i \in [k]$, $T_i(\pi[i])$ is an induced subtree of S. We have the following two cases for vertex $\pi[i]$.

Case 1: $\pi[i] = \pi_{C_j}[i]$ for some $j \in [m]$. As S_j is an agreement supertree for π_{C_j}, it follows that the tree $T_i(\pi[i])$ is an induced subtree of S_j. Since S_j is an induced subtree of S, $T_i(\pi[i])$ is also an induced subtree of S.

Case 2: $\pi[i] \neq \pi_{C_j}[i]$ for every $j \in [m]$. We build an embedding ϕ_i of $T_i(\pi[i])$ into S as follows. By Lemma 44, there exists an embedding $\phi_{p,q}$ from $T_p(\pi_{C_q}[p])$ to S_q for every $p \in [k]$ and $q \in [m]$. For every $v \in V(T_i(\pi[i]))$, set $\phi_i(v) = \phi_{i,j}(v)$ if $v \leq_{T_i} \pi_{C_j}[i]$. Now, set $\phi_i(\pi[i]) = r$. Let us show that ϕ_i is an embedding of $T_i(\pi[i])$ into S. First, observe that ϕ_i satisfies Conditions (a)-(b) of Lemma 44 for any $v <_{T_i} \pi[i]$. Indeed, for such a v we have $v \leq_{T_i} \pi_{C_j}[i]$ for some $j \in [m]$, and Conditions (a)-(b) hold for $\phi_{i,j}$ as it is an embedding of $T_i(\pi_{C_j}[i])$ into S_j. It remains to verify Condition (b) of Lemma 44 for $v = \pi[i]$. For every u child of v, we have $\phi_i(u) = r(S_j)$ for some $j \in [m]$, and thus $\phi_i(u) <_S \phi_i(v)$. Moreover, given two children u, u' of v, as they belong to different classes of P, $\text{child}_S(\phi_i(v), \phi_i(u)) \neq \text{child}_S(\phi_i(v), \phi_i(u'))$. Thus, ϕ_i is an embedding of $T_i(\pi[i])$ into S and hence, $T_i(\pi[i])$ is an induced subtree of S. □

Theorem 20 states the runtime and correctness of the TestAgreement algorithm (Algorithm 2).

Theorem 20. TestAgreement *can be implemented to run in* $O(kn^2)$ *time and correctly decides if there is an agreement supertree for a position* π *in* $\mathcal{T}$*. If there is no agreement supertree for* π*, it returns an obstructing position* π' *and a set* I *of interesting vertices for* π'*.*

Proof. We first justify the running time of the algorithm. Since at each execution of a recursive call, the label sets in each position returned by GetSuccessors are disjoint, it follows that the recursion tree has $O(n)$ leaves. So there are $O(n)$ recursive calls to TestAgreement. Since each execution of the loop in line 5 results in a recursive call, and the body of the loop takes $O(1)$ time outside of the recursive call, it follows that the algorithm spends, over all recursive calls, a total of $O(n)$ time executing lines 5-7. Thus, it suffices to show that each recursive call spends at most $O(kn)$ time outside of lines 5-7. Clearly, lines 1 and 2 can be done in $O(k)$ time. The call to GetSuccessors takes $O(kn)$ time by Theorem 18. Line 4 and 8 can clearly be done in $O(1)$ time.

We now argue for the correctness of the algorithm. We show by induction on the height of the recursion tree, that TestAgreement$(\mathcal{T}, \pi)$ correctly decides if π has an agreement supertree, and, in case there is no agreement supertree for π, the position π'' and set I returned on line 7 are an obstructing position for $\mathcal{T}$ and a set of interesting vertices for π''. Let P be the minimum nice partition of G.

There are two base cases: (i) when $\pi = \pi_\perp$; and (ii) when P has a single class. In case (i) there is an agreement supertree for π and the algorithm returns "yes" on line 2. In case (ii), by Lemma 45, there is no agreement supertree for π. By Theorem 18, the set Π returned in line 3 will be a singleton and π is an obstructing position. By Theorem 19, I is a set of interesting vertices for π. Line 4 returns π along with the set of interesting vertices returned by the call to GetSuccessors.

Now suppose that P has more than one class and Π is the set of successor positions returned by GetSuccessors. Then by induction hypothesis, for each $\pi' \in \Pi$, TestAgreement correctly decides whether there is an agreement supertree for π'. If there is an agreement supertree for each position in Π, then by Lemma 45, there is an agreement supertree for π and the algorithm returns "yes" on line 8. If there is no agreement supertree for some position $\pi' \in \Pi$, then by the inductive hypothesis, TestAgreement$(\mathcal{T}, \pi')$ will answer in the negative, and also return an obstructing position π'' and a set of interesting vertices for π''. By Lemma 43, π'' is an obstructing position for $\mathcal{T}$, so the algorithm returns π'' along with the set I of interesting vertices returned by GetSuccessors. □

5.4 Solving the AST-EC and AST-TR Problems

In this section we show that both the AST-EC and AST-TR problems are fixed-parameter tractable for parameters k and p. Our two algorithms build upon the results of Section 5.3, in the sense that we use the interesting vertices found by TestAgreeement to identify small obstructions. These obstructions allow us to solve the problems by a bounded search tree approach, giving rise to FPT algorithms with running time $O((2k)^p kn^2)$. This section is organized as follows. Section 5.4.1 presents an auxiliary algorithm, called Merge, on which we will rely to construct and prove the correctness of the obstructions for both problems. In Section 5.4.2, we prove that AST-EC is NP-complete and we give an FPT algorithm for the problem. In Section 5.4.3, we give an FPT algorithm for AST-TR.

5.4.1 An auxiliary algorithm

We define the *closure* of a set $C \subseteq V$ as the set $\langle C \rangle_G \subseteq V$ such that: (i) if $C \cap V_i = \emptyset$ then $\langle C \rangle_G \cap V_i = \emptyset$; (ii) if $C \cap V_i = \{v\}$, then $\langle C \rangle_G \cap V_i = \{v\}$; and (iii) if $|C \cap V_i| \geq 2$ then $\langle C \rangle_G \cap V_i = V_i$. Our auxiliary algorithm, called Algorithm Merge, takes as input G and a set $I \subseteq V$, and proceeds as follows. We maintain a partition P of I. Initially P contains a class $\{v\}$ for each $v \in I$. At a given step, suppose that $P = \{C_1, \ldots, C_p\}$. An edge of G is a *transverse edge* if it connects two nodes in the closures of two different sets in P. If G contains a transverse edge joining $\langle C_i \rangle_G$ to $\langle C_j \rangle_G$ for some i, j, then we replace P by $P \backslash \{C_i, C_j\} \cup \{C_i \cup C_j\}$, and we continue.

Lemma 46. *Let Q be the minimum nice partition of G. At a given step of Algorithm* Merge*, it holds that: for each $C \in P$, there is a $K \in Q$ such that $\langle C \rangle_G \subseteq K$.*

Proof. By induction on the number of steps executed by the algorithm. This is clear initially. Suppose that this holds at the beginning of the pth step, and let us show that this holds at the end of the pth step. Suppose that this step replaces P by $P' = P \backslash \{C_i, C_j\} \cup \{C_i \cup C_j\}$. By induction hypothesis, there exist $A, B \in Q$ such that $\langle C_i \rangle_G \subseteq A$ and $\langle C_j \rangle_G \subseteq B$. By assumption, G contains a transverse edge between $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$. As two classes of Q are

disconnected in G, it follows that $A = B$, and thus $C_i \cup C_j \subseteq A$. As A is a nice set, we obtain that $\langle C_i \cup C_j \rangle_G \subseteq A$. □

The crucial property of Algorithm MERGE is that, by starting with the interesting vertices, it will end with P consisting of a single class. This property is stated in the following Lemma, whose proof is given in Section 5.5.2. The proof relies on an alternative formulation of the GETSUCCESSORS algorithm, and on the notion of *merge forest* representing the sequence of merges performed by the algorithm.

Lemma 47. *Suppose that* TESTAGREEMENT$(\mathcal{T}, \pi_\top)$ *has returned* (no, π, I)*. Then Algorithm* MERGE *run on* G, I *ends with the partition* $\{I\}$*.*

Note that this fact certifies that the minimum nice partition Q of G is a singleton: by Lemma 46, it follows that $\langle I \rangle_G = V$ is included in a same class of Q. To check that Q has a unique class, we can thus use I as the certificate, and MERGE as the verification algorithm. We will repeatedly use this fact in the following sections. Additionally, for settings where we actually want to use MERGE for verification purposes, we will give a $O(kn)$ implementation in Section 5.4.3 under the name FINDOBSTRUCTION.

5.4.2 Solving the AST-EC problem

The computational complexity of AST-EC does not seem to have been studied before. To motivate the development of a fixed-parameter algorithm to solve the problem, we first prove the problem is NP-complete.

We use a recursive parenthesized notation for trees: if ℓ is a label, ℓ represents a tree with a single leaf labelled ℓ; if $T_1, \ldots, T_k$ are trees, then $(T_1, \ldots, T_k)$ represents the tree whose root is unlabelled and has $T_1, \ldots, T_k$ as child subtrees.

Theorem 21. AST-EC *is* NP*-complete.*

Proof. Membership in NP is clear. The NP-hardness is shown by a reduction from the MULTICUT problem, which is defined as follows. Given a graph $G = (V, E)$, a set of requests $R \subseteq V \times V$, and an integer p, the MULTICUT problem asks if there exists a set S of at most p edges in E, where, for every $uv \in R$, u and v are in different components of $G \backslash S$.

Given an instance $I = (G, R, p)$ of MULTICUT, we construct an instance $I' = (\mathcal{T}, p)$ of AST-EC as follows. The collection $\mathcal{T}$ is defined over $L := V \cup \{x\}$ and consists of: (i) for each edge $e = uv \in E$, a tree $t_e = ((u, v), x)$; (ii) for each pair $p = uv \in R$, a tree $f_p = (u, v, x)$. For each $e \in E$, we let $\gamma(e)$ denote the internal edge of t_e, we let $\alpha(e)$ denote the parent of u, v in t_e, and we let $\beta(e)$ denote the parent of $\alpha(e), x$ in t_e. The notation $\gamma(e)$ induces a bijection $\gamma : E \to \hat{E}(\mathcal{T})$; we will use the notation $\gamma(S)$ and $\gamma^{-1}(S)$ to denote, respectively, the image and the inverse image of a set S under γ.

The reduction is clearly doable in polynomial time. To prove its correctness, we show that: I is a positive instance of MULTICUT iff I' is a positive instance of AST-EC.

Suppose that (G, R) has a minimum multicut $S \subseteq E$ with $|S| \leq p$. Let $S' = \gamma(S)$, we show that $\mathcal{T}/S'$ has an agreement supertree. Let $C_1, \ldots, C_m$ denote the connected components of $G \backslash S$. By minimality of S, each edge of S is between two distinct components of G. For every $1 \leq i \leq m$, let T_i denote a star-tree with leaves labeled by C_i, and let $T = (T_1, \ldots, T_m, x)$. We show that T is an agreement supertree of $\mathcal{T}/S'$. First, for each $e = uv \in E \backslash S$, we have u, v in a same connected component of $G \backslash S$, and thus $t_e/S' = ((u, v), x)$ is a subtree of T. Second, for each $e = uv \in S$, we have u, v in different connected components of $G \backslash S$, and thus $t_e/S' = (u, v, x)$ is a subtree of T. Third, for each $p = uv \in R$, we have u, v in different connected components of $G \backslash S$, and thus $f_e/S' = (u, v, x)$ is a subtree of T. We conclude that T is an agreement supertree of $\mathcal{T}/S'$.

Conversely, suppose that there exists $S \subseteq \hat{E}(\mathcal{T})$ such that $|S| \leq p$ and $\mathcal{T}/S$ has an agreement supertree T. Let $S' = \gamma^{-1}(S)$, we show that S' is a multicut of (G, R). Suppose by contradiction that $G \backslash S'$ contains a path P between two endpoints of a request $uv \in R$. Then $P = u_0 e_1 u_1 \ldots u_{r-1} e_r u_r$ with $u = u_0, v = u_r$. It follows from Lemma 44 that for every $1 \leq i \leq r$, there is an embedding ϕ_i of t_{e_i} into T; let $a_i = \phi_i(\alpha(e_i))$ and $b_i = \phi_i(\beta(e_i))$. Since ϕ_i is an embedding, the nodes $\text{child}_T(b_i, a_i)$ and $\text{child}_T(b_i, x)$ are distinct; let us denote them by p_i, q_i. As u_i occurs in t_{e_i} and $t_{e_{i+1}}$, it follows that $b_{i+1} = b_i, p_{i+1} = p_i, q_{i+1} = q_i$. Denote these nodes by b, p, q. We then have $u_0 <_T p$, $u_r <_T p$ and $x \leq_T q$, which implies that $((u_0, u_r), x)$ is a subtree of T. But by definition of T, it contains $f_{uv} = (u, v, x)$ as a subtree, contradiction. We conclude that there is no such path P, and thus S' is a multicut of (G, R). □

In the remainder of this subsection, we show that AST-EC is fixed-parameter tractable in parameters k and p. Lemma 48 shows that if a call to TestAgreement$(\mathcal{T}, \pi_\top)$ answers negatively, we must contract at least one edge joining an interesting vertex to its parent.

Lemma 48. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *has returned a tuple* (no, π, I). *In order to obtain a collection having an agreement supertree, we need to contract one edge* $\{v,\ parent(v)\}$ *with* $v \in I$.

Proof. Assume that we have a set $S \subseteq \hat{E}(\mathcal{T})$ which contains none of the edges $\{v,\ parent(v)\}$ with $v \in I$; we show that $\mathcal{T}/S$ has no agreement supertree. We use the following convention: when contracting an edge $\{u, v\}$ with $v = parent(u)$, the resulting vertex is identified with v. Then, by definition of S, each element of I is still a node of $\mathcal{T}/S$. Define the position π' in $\mathcal{T}/S$ as follows. If $\pi[i] = \perp$, then $\pi'[i] = \perp$. If $\pi[i] = u$ is the parent of two vertices $v, w \in I$, then $\pi'[i]$ is the common parent of v, w in T_i/S. Let $G' = G(\mathcal{T}/S, \pi') = (V', E')$.

Let us consider an execution $\mathcal{E}$ of Algorithm Merge on G and I. We build an execution $\mathcal{E}'$ of Algorithm Merge on G' and I that ends with $\{I\}$ by mirroring each step of $\mathcal{E}$. Let $P_\mathcal{E}, P_{\mathcal{E}'}$ denote the values of P during $\mathcal{E}, \mathcal{E}'$ respectively. We define $\mathcal{E}'$ by induction such that $P_\mathcal{E} = P_{\mathcal{E}'}$ holds at each step. Clearly, this holds at the beginning of $\mathcal{E}, \mathcal{E}'$. Suppose that this holds at the beginning of step s. Then $\mathcal{E}$ picks a transverse edge induced by some label $\ell \in \mathcal{L}(\langle C_i \rangle_G) \cap \mathcal{L}(\langle C_j \rangle_G)$. The important observation is that given $C \subseteq I$, we have $\mathcal{L}(\langle C \rangle_G) \subseteq \mathcal{L}(\langle C \rangle_{G'})$ (as $\mathcal{L}(\pi[i]) \subseteq \mathcal{L}(\pi'[i])$ for every $i \in [k]$). Hence, $\ell \in \mathcal{L}(\langle C_i \rangle_{G'}) \cap \mathcal{L}(\langle C_j \rangle_{G'})$, and thus $\mathcal{E}'$ can pick a transverse edge joining $\langle C_i \rangle_{G'}$ and $\langle C_j \rangle_{G'}$. This leads to the merge of C_i and C_j, and thus $P_\mathcal{E} = P_{\mathcal{E}'}$ at the end of step s.

Applying the induction hypothesis at the last step of $\mathcal{E}$, and using that $\mathcal{E}$ ends with the partition $\{I\}$ (Lemma 47), we obtain that $\mathcal{E}'$ ends with the partition $\{I\}$. By Lemma 46, if Q is the minimum nice partition of G', we have $\langle I \rangle_{G'} = V'$ in a same component of Q, and thus $\mathcal{T}/S$ has no agreement supertree by Lemmas 43 and 45. □

Since TestAgreement returns a set of at most $2k$ interesting vertices, Lemma 48 leads to an FPT algorithm for AST-EC using a bounded search tree technique.

Theorem 22. AST-EC *can be solved in* $O((2k)^p kn^2)$ *time.*

Proof. We use a recursive algorithm SolveAST-EC$(\mathcal{T}, p)$. The algorithm answers "no" if $p < 0$. Otherwise, it runs TestAgreement$(\mathcal{T}, \pi_\top)$ to decide in $O(kn^2)$ if $\mathcal{T}$ has an agreement supertree. It answers "yes" in case in positive answer. In case of negative answer, it obtains a set I of nodes of $\mathcal{T}$; for each non-leaf vertex $v \in I$, it recursively calls SolveAST-EC$(\mathcal{T}/\{v, parent(v)\}, p-1)$. The algorithm then answers "yes" iff one of the recursive calls does. The correctness of the algorithm follows from Lemma 48, and the running time is $O((2k)^p kn^2)$. □

5.4.3 Solving the AST-TR problem

We will say that a set $C \subseteq \mathcal{L}(\mathcal{T})$ is a *conflict among* $\mathcal{T}$ if $\mathcal{T}|C$ has no agreement supertree. Lemma 49 shows that if TestAgreement$(\mathcal{T}, \pi_\top)$ answers negatively, we can obtain a conflict among $\mathcal{T}$ from the set of interesting vertices.

Lemma 49. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *has returned a tuple* (no, π, I)*. We can then obtain a set* $C \subseteq L$ *of size at most* $2k-1$ *such that* $\mathcal{T}|C$ *has no agreement supertree.*

Proof. Consider an execution $\mathcal{E}$ of Algorithm Merge on G and I. For each transverse edge $e = uv$ found by $\mathcal{E}$, pick a label $\ell_e \in \mathcal{L}(u) \cap \mathcal{L}(v)$, and let C be the resulting set of labels. Clearly $|C| \leq 2k-1$. Consider a vertex $v \in I \cap V_i$. Then, during $\mathcal{E}$ consider the first time that $\{v\}$ is merged with another component. This merge corresponds to some label $\ell_v \in \mathcal{L}(v) \cap C$. It follows that $\pi[i]$ is still a node of $T_i|C$, let v' denote the child of $\pi[i]$ in $T_i|C$ that contains ℓ_v. Let $I' = \{v' : v \in I\}$, and let $G' = G(\mathcal{T}|C, \pi) = (V', E')$.

We claim that (as in the proof of Lemma 48) the execution $\mathcal{E}$ can be simulated by an execution $\mathcal{E}'$ of Algorithm Merge on G' and I'. Let $P_\mathcal{E}, P_{\mathcal{E}'}$ denote the values of P during $\mathcal{E}, \mathcal{E}'$ respectively. We define $\mathcal{E}'$ by induction such that the following holds at each step: if $P_\mathcal{E} = \{C_1, \ldots, C_p\}$, then $P_{\mathcal{E}'} = \{C'_1, \ldots, C'_p\}$ with $C'_i = \{v' : v \in C_i\}$. Clearly, this holds at the beginning of $\mathcal{E}, \mathcal{E}'$. Suppose that this holds at the beginning of step s. Then $\mathcal{E}$ picks a transverse edge $e = uv$ with $u \in \langle C_i \rangle_G, v \in \langle C_j \rangle_G$. Suppose that $u \in V(T_p)$ and $v \in V(T_q)$. In $T_p|C$ (resp. $T_q|C$), there is a child u' of $\pi[p]$ (resp. a child v' of $\pi[q]$) that contains ℓ_e. The induction hypothesis implies that $u' \in \langle C'_i \rangle_{G'}$ and $v' \in \langle C'_j \rangle_{G'}$, thus $\mathcal{E}'$ can pick the transverse edge $e' = u'v'$ induced by label ℓ_e. This leads to merge C'_i and C'_j and thus the induction hypothesis holds at

the end of step s.

Applying the induction hypothesis at the last step of $\mathcal{E}$, and since $\mathcal{E}$ ends with the partition $\{I\}$ (Lemma 47), we conclude that $\mathcal{E}'$ ends with the partition $\{I'\}$. By Lemma 46, if Q is the minimum nice partition of G', we have $\langle I' \rangle_{G'} = V'$ in a same component of Q, and thus $\mathcal{T}|C$ has no agreement supertree by Lemmas 43 and 45. □

We outline an algorithm FindObstruction that takes as input a set of interesting vertices and returns a conflict among $\mathcal{T}$ of size at most $2k-1$. Suppose that $I = \{v_1, \ldots, v_r\}$. We initialize components $C_1, \ldots, C_r$ with $C_i = \{v_i\}$, and we let $J = \{1, \ldots, r\}$. We use a main loop which performs the $r-1$ steps of Algorithm Merge. At each step, we have $J \subseteq \{1, \ldots, r\}$, and the current partition is represented by the components C_i $(i \in J)$, which are called the *active* components. We maintain for each $\ell \in L$ a variable $J(\ell) = \{i \in J : \ell \in \mathcal{L}(\langle C_i \rangle_G)\}$. At a given step, we have to find a label inducing a transverse edge which joins the closure of two active components. This amounts to looking for an ℓ such that $|J(\ell)| \geq 2$. Once such an ℓ has been found, we pick two indices $i, j \in J(\ell)$, and we merge the components C_i, C_j, letting C_i be the newly created component, and updating the variables $J(\ell)$ accordingly. An implementation of FindObstruction is given in the listing of Algorithm 3.

Lemma 50. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *returns* (no, π, I). *Then Algorithm* FindObstruction$(\mathcal{T}$ *returns in* $O(kn)$ *time a conflict among* $\mathcal{T}$ *of size at most* $2k-1$.

Proof. We first argue for the correctness. If $2k \geq n$, then the set L returned by the algorithm is a conflict, as by assumption $\mathcal{T}$ has no agreement supertree. Suppose now that $2k < n$. By Lemma 49, it suffices to show that an execution $\mathcal{E}$ of Algorithm 3 simulates an execution $\mathcal{E}'$ of Algorithm Merge. More precisely, we show the following holds after each step s.

1. There is an execution $\mathcal{E}'$ of s steps of Algorithm Merge which produces the set of components C_i $(i \in J)$ and, the set of labels R is exactly the set of labels corresponding to the transverse edges which induced all the s merges in $\mathcal{E}'$.

2. For each $\ell \in L$, $J(\ell) = \{i \in J : \ell \in \mathcal{L}(\langle C_i \rangle_G)\}$.

Input: A collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of rooted trees, an obstructing position π in $\mathcal{T}$, a set $I = \{v_1, \ldots, v_r\}$ of interesting vertices for π.
Output: A conflict among $\mathcal{T}$.

```
1  if 2k ≥ n then return L
2  R ← ∅ ; J ← {1, ..., r}
3  for i from 1 to r do
4  |  C_i ← {v_i}
5  |  foreach ℓ ∈ L(v_i) do J(ℓ) ← J(ℓ) ∪ {i}
6  for s from 1 to r − 1 do
7  |  Pick ℓ ∈ L such that |J(ℓ)| ≥ 2, and choose i, j ∈ J(ℓ)
8  |  R ← R ∪ {ℓ}
9  |  for ℓ ∈ L do
10 |  |  if j ∈ J(ℓ) then J(ℓ) ← J(ℓ)\{j} ∪ {i}
11 |  for p from 1 to k do
12 |  |  if C_i ∩ V_p ≠ ∅ and C_j ∩ V_p ≠ ∅ then
13 |  |  |  foreach ℓ ∈ L(π[p]) do J(ℓ) ← J(ℓ) ∪ {i}
14 |  C_i ← C_i ∪ C_j, J ← J\{j}
15 return R
```

Algorithm 3: FindObstruction$(\mathcal{T}, \pi, I)$

This is shown by induction on s. The initialization of the variables C_i and $J(\ell)$ in Lines 3-5 ensure that this is true initially. Suppose that this holds at the beginning of step s. The choice of ℓ and i, j in Line 7 ensures that $\ell \in \mathcal{L}(\langle C_i \rangle_G) \cap \mathcal{L}(\langle C_j \rangle_G)$, and thus there exists a transverse edge e between $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$, induced by the label ℓ. The update of $C_i \leftarrow C_i \cup C_j$ and $J \leftarrow J \backslash \{j\}$ reflect the merge of C_i and C_j, and thus we can simulate step s of Algorithm Merge which would choose transverse edge e and merge C_i and C_j. This establishes Point 1, and the update of $J(\ell)$ in Lines 9-13 ensures that Point 2 is preserved.

We now justify the running time. Let us assume that $2k < n$, as otherwise the algorithm takes $O(1)$ time. We implement the sets $J(\ell)$ by bit arrays, allowing in constant time the following operations: (i) insertion or deletion of an element, (ii) obtaining the size of the set. It follows that Lines 3-5 take $O(rn) = O(kn)$ time. Let us now analyze the time taken by the s-th iteration of the loop in Lines 6-14. Let K_s denote the set of indices p for which the condition of Line 12 holds. Then Lines 7-10 take $O(n)$ time, Lines 11-13 take $O(k + |K_s|n)$ time, and Line 14 takes $O(k)$ time. Overall, Lines 7-14 take $O(n + |K_s|n)$ time as $k = O(n)$. Observe that the sets K_s are disjoint for $s = 1, \ldots, r-1$, and thus $\sum_s |K_s| = O(k)$. It follows that the

loop of Lines 6-14 take $O(rn + kn) = O(kn)$ time, and we conclude that the whole algorithm runs in $O(kn)$ time. □

Theorem 23. AST-TR *can be solved in* $O((2k)^p kn^2)$ *time.*

Proof. We use a recursive algorithm SolveAST-TR$(\mathcal{T}, p)$. The algorithm answers "no" if $p < 0$. Otherwise, it runs TestAgreement$(\mathcal{T}, \pi^\top)$ to decide in $O(kn^2)$ if $\mathcal{T}$ has an agreement supertree. It answers "yes" in case of positive answer. In case of negative answer, it obtains a position π and a set I of interesting nodes for π. It calls FindObstruction$(\mathcal{T}, \pi, I)$ to obtain in $O(kn)$ time a conflict C among $\mathcal{T}$ of size at most $2k - 1$. Then, for each $\ell \in C$, it recursively calls SolveAST-TR$(\mathcal{T}|(\mathcal{L}(\mathcal{T}) \setminus \{\ell\}), p - 1)$, and it answers "yes" if and only if one of the recursive calls does. The correctness follows from Lemma 50, and the running time is $O((2k)^p kn^2)$. □

5.5 Deferred Proofs

5.5.1 Proofs of Section 5.3.2

The proof of Theorem 18 relies on three lemmas. Lemma 51 justifies the running time of the algorithm, while Lemmas 52 and 53 prove its correctness.

Lemma 51. *The implementation of* GetSuccessors *given in Algorithm 1 runs in* $O(kn)$ *time.*

Proof. Clearly, line 1 takes $O(n)$ time. Each iteration of the outer loop in lines 2-6 takes a total of $O(n)$ time, since each label can be in the subtree of at most one child of $\pi[i]$, and there are k iterations, for a total of $O(kn)$ time spent on lines 2-6. Line 7 takes $O(1)$ time.

We now argue that the algorithm spends a total of $O(kn)$ time on lines 8-24. The loop in line 8 executes $O(n)$ times. Since any given label can be in a given tree at most once, we have that each $|S_\ell| \leq k$ for each $\ell \in \mathcal{L}(\pi)$. Hence the loop in line 10 executes $O(k)$ times for each execution of the outer loop in line 8, and takes constant time per iteration. Since lines 9, 22, and 23 take constant time, we have that the algorithm spends a total of $O(kn)$ time on lines 8-13, and lines 22-23.

We consider lines 14-21 separately. Since we create a total of n positions, and once a position's active flag is set to false it is never again set to true, we have that the while loop of line 14 executes a total of at most n times during the execution of the entire algorithm. The loop in line 16 executes k times and takes $O(1)$ time if the test in line 17 is true, and $O(n)$ time if the test in line 17 is false. However, the test in line 17 can be false at most k times during the entire execution of the algorithm. Hence the algorithm spends a total of $O(kn)$ executing lines 14-17 and $O(kn)$ time executing lines 18-21.

Since there are at most n positions created, line 24 takes $O(n)$ time. This completes the analysis of the runtime of GetSuccessors, showing that it takes $O(kn)$ time. □

Consider an execution of GetSuccessors$(\mathcal{T}, \pi)$, and let $G = G(\mathcal{T}, \pi)$ as before. Observe that Lines 1-6 ensure that for each $\ell \in L$, $S_\ell = \{v \in V : \ell \in \mathcal{L}(v)\}$. Suppose that Loop 8-21 examines the labels in the order $\ell_1, \ldots, \ell_n$. Given $0 \leq i \leq n$, let $L_i = \{\ell_1, \ldots, \ell_i\}$. Let G_i denote the graph with vertex set V, and which contains an edge uv iff $\mathcal{L}(u) \cap \mathcal{L}(v)$ intersects L_i. Let Q_i denote the minimum nice partition of G_i, and let $\mathcal{P}_i$ denote the set of nice partitions of G_i. In the following, we let $\ell = \ell_{i+1}$.

Lemma 52. *Q_i is finer than Q_{i+1}. Furthermore, let S denote the set of classes $C \in Q_i$ which contain a vertex in S_ℓ. Then the classes of S are included in a same class K_ℓ of Q_{i+1}, and $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$, where Q' is the set of classes of Q_i included in K_ℓ.*

Proof. We first show that Q_i is finer than Q_{i+1}. Observe that a nice partition of G_{i+1} is also a nice partition of G_i, as $E(G_i) \subseteq E(G_{i+1})$. It follows that $Q_i = \sqcap_{P \in \mathcal{P}_i} P \sqsubseteq \sqcap_{P \in \mathcal{P}_{i+1}} P = Q_{i+1}$, where the inclusion holds as $\mathcal{P}_{i+1} \subseteq \mathcal{P}_i$. We deduce that each class of Q_i is included in a class of Q_{i+1}. Now, the classes of S must be included in a same class K_ℓ of Q_{i+1}, as two classes of Q_{i+1} are disconnected. Let Q' be the set of classes of Q_i included in K_ℓ. Then, Q_{i+1} is obtained from Q_i by merging together the classes of Q', and possibly some other classes. Suppose by contradiction that $Q_{i+1} \neq Q_i \backslash Q' \cup \{K_\ell\}$, then there is a class K' of Q_{i+1} distinct of K_ℓ and which is not a class of Q_i. Let $C_1, \ldots, C_m$ be the classes of Q_i included in K', with $m \geq 2$. Let $R = Q_{i+1} \backslash \{K'\} \cup \{C_1, \ldots, C_m\}$. Then Q_i is finer than R and thus R is a nice partition of G_i. On the other hand, R is not a nice partition of G_{i+1} by minimality of Q_{i+1}. It follows

that G_{i+1} contains an edge joining two classes C_p, C_q; as R is a nice partition of G_i, this edge must be induced by ℓ. But K_ℓ is the only class of Q_{i+1} which intersects S_ℓ, contradiction. We conclude that $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$. □

A *successor* of π is a position π' such that for every $i \in [k]$, either $\pi'[i] = \perp$, or $\pi'[i] \in V_i$, or $\pi'[i] = \pi[i]$. Given π' successor of π, we define the corresponding component $C_\pi \subseteq V$, such that: if $\pi'[i] = \perp$ then $C_\pi \cap V_i = \emptyset$; if $\pi'[i] = v \in V_i$ then $C_\pi \cap V_i = \{v\}$; if $\pi'[i] = \pi[i]$ then $C_\pi \cap V_i = V_i$. Let Π_i denote the set of active positions of Π at the end of step i of Loop 8-21. The construction of each position π_ℓ in Lines 14-21 ensures that Π_i is a set of successors of π. Let P_i denote the family containing (i) the sets $C_{\pi'}$ for $\pi' \in \Pi_i$, (ii) the singletons $\{v\}$ for $v \in V$ such that there is no $\pi' \in \Pi_i$ with $v \in C_{\pi'}$.

Lemma 53. *For every i $(0 \leq i \leq n)$, it holds that $P_i = Q_i$ at the end of step i of the loop in lines 8-21.*

Proof. Let us show that this holds initially. We have $\Pi_0 = \emptyset$, $V_0 = \emptyset$, and thus P_0 consists of the singletons $\{v\}$ for $v \in V$. This is clearly equal to Q_0 as G_0 is the empty graph. Let us now assume that this holds at the end of step i, let us consider step $i+1$ and let $\ell = \ell_{i+1}$. By the induction hypothesis, we have $P_i = Q_i$. By Lemma 52, we have $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$, where Q' is the set of classes of Q_i included in K_ℓ. Let S denote the set of components of Q_i corresponding (i) to positions added to Z in Lines 11-12, (ii) to singleton components $\{v\}$ for v examined at Line 13. Consider step j of Loop 14-21. At a given step, let π_ℓ^j denote the current value of π_ℓ, let C_ℓ^j denote the corresponding component, let Z^j be the set of elements of Z, and let Z_f^j be the set of elements $\pi \in Z$ with $\mathsf{active}(\pi) = \mathsf{false}$. Let D_ℓ^j denote the set of vertices $v \in C_\ell^j$ such that there is no $\pi' \in \Pi_i$ with $v \in C_{\pi'}$.

Claim 1. At step j of Loop 14-21, we have:

(a) $D_\ell^j \cup \bigcup_{\pi' \in Z_f^j} C_{\pi'}$ is included in C_ℓ^j;

(b) C_ℓ^j is included in $D_\ell^j \cup \bigcup_{\pi' \in Z^j} C_{\pi'}$ and in K_ℓ;

(c) for each $\pi' \in Z^j$, $C_{\pi'} \subseteq K_\ell$.

Proof. Let us verify that this holds for $j = 0$. Observe that $C_\ell^0 = D_\ell^0$, as each vertex v added to C_ℓ^0 is obtained in Line 13 and has $\mathsf{position(v)} = \emptyset$. Point (a) is an equality. Points (b) and (c) follow from the fact that the components of S are included in K_ℓ, according to the definition of K_ℓ in Lemma 52.

Let us now verify it for $j+1$, assuming that it holds for j. Suppose that step $j+1$ examines position $\pi' \in Z^j$. Let Z' be the set of positions added at Line 21 during this step. Lines 16-21 compute π_ℓ^{j+1} such that $C_\ell^{j+1} = \langle C_\ell^j \cup C_{\pi'} \rangle_{G_{i+1}}$. By the induction hypothesis, $D_\ell^j \cup \bigcup_{\pi' \in Z_f^j} C_{\pi'} \subseteq C_\ell^j \subseteq C_\ell^{j+1}$; by definition, $D_\ell^{j+1} \subseteq C_\ell^{j+1}$ and $C_{\pi'} \subseteq C_\ell^{j+1}$; as $Z_f^{j+1} = Z_f^j \cup \{\pi'\}$, we deduce that Point (a) holds. Let us show Point (b). As $C_\ell^j \subseteq K_\ell$ and $C_{\pi'} \subseteq K_\ell$ by the induction hypothesis, it follows that $C_\ell^j \cup C_{\pi'} \subseteq K_\ell$, and thus $C_\ell^{j+1} \subseteq K_\ell$ as K_ℓ is a nice set. Now, observe that for every $v \in C_\ell^{j+1} \backslash C_\ell^j$ we have either $v \in D_\ell^{j+1}$ (if $\mathsf{position(v)} = \emptyset$ at Line 21) or $v \in C_{\pi'}$ for some $\pi' \in Z'$ (if $\mathsf{position(v)} = \{\pi'\}$ at Line 21). It follows that $C_\ell^{j+1} \subseteq C_\ell^j \cup D_\ell^{j+1} \cup \bigcup_{\pi' \in Z'} C_{\pi'} \subseteq D_\ell^{j+1} \cup \bigcup_{\pi' \in Z^j} C_{\pi'}$ (using the induction hypothesis), and thus Point (b) holds. Let us show Point (c). Observe that for each $\pi' \in Z'$, $C_{\pi'}$ intersects C_ℓ^{j+1}. As $C_\ell^{j+1} \subseteq K_\ell$ and as $C_{\pi'}$ is a class of Q_i, it follows that $C_{\pi'} \subseteq K_\ell$. ◇

Suppose that we have reached Line 23. Let Π' be the set of positions in Z at this step, then $\Pi_{i+1} = \Pi_i \backslash \Pi' \cup \{\pi_\ell\}$. Thus, we have $P_{i+1} = P_i \backslash P' \cup \{C_\ell\}$, where C_ℓ is the component corresponding to π_ℓ. Recall that $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$ and that $P_i = Q_i$. To show that $P_{i+1} = Q_{i+1}$, we will prove that (i) $C_\ell \subseteq K_\ell$, (ii) P_{i+1} is a nice partition of G_{i+1}.

(i) $C_\ell \subseteq K_\ell$. Indeed, by applying Claim 1 at the last step of Loop 14-21, we obtain that $C_\ell \subseteq K_\ell$ by Point (b).

(ii) P_{i+1} is a nice partition of G_{i+1}. We first show that P_{i+1} is a partition of V. Let D_ℓ denote the value of D_ℓ^j at the last step of Loop 14-21. Note that $P_{i+1} = P_i \backslash P' \cup \{C_\ell\}$, where P' contains (i) the singleton components $\{v\}$ for $v \in D_\ell$, (ii) the set of components corresponding to the positions in Z. Let X denote the union of the components in P', observe that $X = D_\ell \cup \bigcup_{\pi \in Z} C_\pi$. By Points (a) and (b) of Claim 1, we have $X = C_\ell$. As P_i is a partition of V, we obtain that P_{i+1} is a partition of V.

We now show that P_{i+1} is nice. Each class of P_{i+1} is nice, as it is either a class of P_i (which is a nice partition of G_i), or the set C_ℓ (which is nice). Suppose that P_{i+1} contains

two classes C, C' that are connected. As P_i is a nice partition of G_i finer than P_{i+1}, it holds that P_{i+1} is a nice partition of G_i. Thus, C and C' are disconnected in G_i, and they must be connected in G_{i+1} by an edge induced by ℓ. But C_ℓ is the only class of P_{i+1} which intersects S_ℓ, a contradiction. □

The proof of Theorem 18 follows directly from Lemmas 51, 52, and 53.

We conclude this section with a proof of Theorem 19. Recall that when the set of successor positions returned by GetSuccessors is a singleton, the algorithm returns a set of vertices I along with the obstructing position π. Theorem 19 states that, in this case, the vertices in I are interesting for π. We have delayed the proof of this theorem until now for expository reasons: It is not until Section 5.4 that we developed the machinery required for the proof.

of Theorem 19. The following properties of Algorithm Merge demonstrate that the set I satisfies the definition of interesting vertices:

- Each vertex in I starts in its own class.
- At each step of the algorithm a transverse edge is found and two of the classes are merged.

Let $K = \{i \in [k] : \pi[i] \neq \perp\}$. By Lemma 47, Algorithm Merge run on G and I returns $\{I\}$. Since there are $|I| = 2|K|$ initial classes, it follows that $2|K| - 1$ transverse edges are needed to merge them into a single class. Furthermore, since each interesting vertex is initially the unique element in its class, it must be an endpoint of the transverse edge found that merged that class with the final class. Thus, the set of transverse edges used by Algorithm Merge prove that I is indeed a set of interesting vertices. □

5.5.2 Proof of Lemma 47

For convenience, we introduce an abstract version of GetSuccessors called Algorithm $\mathcal{A}$. The algorithm takes as input $\mathcal{T}$ and the position π, and it returns a partition P of V, and a set I of interesting vertices. For each $\ell \in L$, the algorithm constructs the set $S_\ell = \{v \in V : \ell \in \mathcal{L}(v)\}$. Initially, P consists of the singletons $\{v\}$ for $v \in V$, and $I = \emptyset$. Then, the algorithm successively

examines each label $\ell \in L$. When examining ℓ, it constructs a set $Z \subseteq P$ and $K \subseteq V$ in two steps. It starts with $Z = \emptyset$ and $K = \emptyset$, and it does the following.

- Phase A: for each $C \in P$ intersecting S_ℓ, add C to Z.
- Phase B: while Z contains an unprocessed class C, extract C from Z and do the following. For every $i \in [k]$, if K and C respectively contain a vertex x and a vertex y where $x, y \in V_i$, then add x, y to I, and for every $v \in V_i \backslash \{x, y\}$ add to Z the class $C' \in P$ that contains v. Then let $K \leftarrow \langle K \cup C \rangle_G$.
- At the end of Phase B, let P' be the set of classes of P which have been added to Z. Let $P \leftarrow P \backslash P' \cup \{K\}$.

It is not difficult to see that Algorithm $\mathcal{A}$ is equivalent to Algorithm GetSuccessors, so for every execution of GetSuccessors producing the set of vertices I, there is a corresponding execution of Algorithm $\mathcal{A}$ which produces I. An execution of Algorithm $\mathcal{A}$ can be represented by a *merge forest*. This is an ordered rooted forest $\mathcal{M}$, where each node u of $\mathcal{M}$ is associated to a component K_u produced by the algorithm, such that at each step of Algorithm $\mathcal{A}$ the roots of $\mathcal{M}$ correspond to the classes of P. $\mathcal{M}$ is constructed as follows. At the beginning of Algorithm $\mathcal{A}$, $\mathcal{M}$ consists of one isolated node for each class of P. When Algorithm $\mathcal{A}$ examines label ℓ, let P' be the set produced at the end of Phase B and let K be the class added to P, then $\mathcal{M}$ is updated by adding a vertex u with $K_u = K$, and by adding an arc (u, v) for each root v of $\mathcal{M}$ corresponding to a class $K_v \in P'$. Then, the new children of u correspond to the classes added to Z, and they are ordered according to the first time when they were added to Z.

We state below some simple properties of the merge forest. The following lemma can be proved by a similar argument as in the proof of Lemma 53.

Lemma 54. *Let u be an internal node of $\mathcal{M}$ with children $u_1, \ldots, u_p$. Then $\{K_{u_1}, \ldots, K_{u_p}\}$ is a partition of K_u.*

Let $i \in [k]$, and let u be a node of $\mathcal{M}$. We say that u *merges* V_i if $V_i \subseteq K_u$ but there is no child v of u with $V_i \subseteq K_v$. Suppose that u merges V_i. Given $x \in V_i$, we let $C_x(u)$ denote the child v of u such that $x \in K_v$. We let $Ch(u)$ denote the ordered list of children of u in $\mathcal{M}$.

Lemma 55. *Suppose that u merges V_i. Then I contains two vertices $x, y \in V_i$, and for every $z \in V_i \backslash I$, $C_x(u), C_y(u)$ precede $C_z(u)$ in $Ch(u)$.*

We are now in position to prove Lemma 47.

of Lemma 47. By definition of π and I, it holds that GetSuccessors$(\mathcal{T}, \pi)$ has returned $(\{\pi\}, I)$. It follows that there is a corresponding execution of Algorithm $\mathcal{A}$ on $\mathcal{T}, \pi$ which returns $(\{V\}, I)$. Let $\mathcal{M}$ be the merge forest corresponding to this execution, then $\mathcal{M}$ has a single root r with $K_r = V$. Consider an execution of Algorithm Merge on G, I. We show that as long as P contains at least two classes, the algorithm finds a transverse edge. Suppose that $P = \{C_1, \ldots, C_p\}$ with $p \geq 2$, we thus need to find a transverse edge joining $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$ for some i, j.

We will need the following definitions. Let v be a node of $\mathcal{M}$. Given $x \in K_v$, we color x with color $1 \leq i \leq p$ if $x \in \langle C_i \rangle_G$. We say that v is *split* if K_v contains two colored vertices of different colors. We say that v is *full* if for each $i \in [k]$, if $K_v \cap V_i = \{x\}$ then x is colored. For every $i \in [k]$ such that $V_i \subseteq K_v$, the vertices of $V_i \backslash I$ are called *secondary vertices* of K_v. Given v' child of v in $\mathcal{M}$, we say that v' is a *secondary child* of v if there exists $i \in [k]$ such that $K_{v'} \cap V_i = \{x\}$ with x secondary vertex of K_v; otherwise, we say that v' is a *primary child* of v.

Let S be the set of nodes of $\mathcal{M}$ that are split and full. Observe that the root r of $\mathcal{M}$ is in S. Indeed, r is split as K_r contains all interesting nodes and as $p \geq 2$; r is full as there is no $i \in [k]$ such that $|K_r \cap V_i| = 1$. Let u be a deepest node of S, and let $u_1, \ldots, u_m$ be its ordered list of children. We say that a node v of $\mathcal{M}$ is *monochromatic* (with color c) iff all vertices of K_v have the same color c.

Claim 1. If v is a primary child of u, then v is added during Phase A of Algorithm $\mathcal{A}$. Furthermore, v is full and monochromatic.

Proof. For the first point, observe that if v is added during Phase B of Algorithm $\mathcal{A}$ then K_v contains a secondary vertex of K_u and thus v is a secondary child of u. Let us now show the second point. Suppose by contradiction that v is not full. Then there is some $i \in [k]$ such that $K_v \cap V_i = \{x\}$ with x uncolored. As u is full, we cannot have $K_u \cap V_i = \{x\}$ and thus $V_i \subseteq K_u$.

As x is uncolored, we have $x \in V_i \backslash I$, and thus x is a secondary vertex of K_u. We conclude that v is a secondary child of u, contradiction. It follows that v is full, and by choice of u it cannot be split. Thus, all colored vertices of K_v have the same color c. Now, if $K_v \cap V_i = \{x\}$ then x has color c (as K_v is full), and if $V_i \subseteq K_v$ then the two vertices of $V_i \cap I$ have color c, which implies that all vertices of V_i have color c. It follows that v is monochromatic with color c. $\diamond$

By Claim 1, each primary child u_i of u is monochromatic with color c_i.

Claim 2. There exist two primary children u_i, u_j of u such that $c_i \neq c_j$.

Proof. By way of contradiction, assume that all primary children have the same color c. We show by induction on $1 \leq j \leq m$ that u_j is monochromatic with color c. This holds if u_j is a primary child of u, so let us assume that u_j is a secondary child of u. Let $J \subseteq K_{u_j}$ be the set of vertices $x \in K_{u_j}$ such that $K_{u_j} \cap V_i = \{x\}$ for some $i \in [k]$, and let $J' \subseteq J$ be the set of vertices of J that are secondary vertices of K_u. Then $J' \neq \emptyset$ as u_j is a secondary child of u. Consider $z \in J'$ and suppose that $K_{u_j} \cap V_i = \{z\}$. As z is a secondary vertex of K_u, we then have $V_i \subseteq K_u$. Then u merges V_i, and by Lemma 55 it follows that I contains two elements $x, y \in V_i$. Let u_p, u_q be the children of u such that $x \in K_{u_p}, y \in K_{u_q}$. We have $p, q < j$ by Lemma 55. We can thus apply the induction hypothesis to obtain that x, y have color c, which implies that z has color c. We obtain that all vertices of J' have color c. On the other hand, all vertices of $J \backslash J'$ are colored, as u is full. We conclude that u_j is full, with some vertex having color c. By choice of u, its child u_j cannot be split. A similar reasoning as in the proof of Claim 1 shows that u_j is monochromatic with color c. This concludes the induction, and we obtain that all children of u are monochromatic with color c. By Lemma 54, we obtain that u is monochromatic with color c, contradicting the assumption that u is split. $\diamond$

Claim 2 yields two primary children u_i, u_j of different colors c, c'. By Claim 1, they are added during Phase A of Algorithm $\mathcal{A}$, and thus the label that is examined in the iteration of Algorithm $\mathcal{A}$ when K_u is built induces an edge between K_{u_i} and K_{u_j}. We have thus shown the existence of a transverse edge between $\langle C_c \rangle_G$ and $\langle C_{c'} \rangle_G$, which concludes the proof. $\square$

5.6 Concluding Remarks

We have given $O((2k)^p kn^2)$ time algorithms for both the AST-EC and AST-TR problems, thus showing they are fixed-parameter tractable for parameters k and p. We remark here that the bound of $2k-1$ given for the obstruction set of AST-TR (Lemmas 49 and 50) is tight.

Our proof that AST-EC is NP-hard relies on a reduction from the parameterized Multicut problem to the AST-EC problem parameterized by p. As Multicut is fixed-parameter tractable [Bousquet et al. (2011); Marx and Razgon (2011)], this leaves open the question of whether AST-EC could be fixed-parameter tractable in p only. It is known that AST-TR is fixed-parameter intractable for parameter p [Berry and Nicolas (2007)].

Our focus here was on agreement supertrees. A compatible supertree is one that contains a refinement of each of the input trees. There are natural analogs of AST-EC and AST-TR for compatible supertrees. For binary input trees, compatibility is equivalent to agreement, so the results of Guillemot and Berry (2010) imply fixed-parameter tractability. However, for input trees of arbitrary degree, we have established that any upper bound on the cardinality of an obstruction set is at least $c2^k$. Hence, the techniques given here are unlikely to imply efficient fixed-parameter tractability for the analogs of AST-TR and AST-EC to compatible supertrees.

There are also analogs of both AST-EC and AST-TR to unrooted trees. Although Maximum Agreement Supertree (Smast) has been studied for unrooted trees [Berry and Nicolas (2007); Hoang and Sung (2011)], the AST-EC and AST-TR problems for unrooted trees do not seem to have been studied before.

CHAPTER 6. INCOMPATIBLE SETS OF QUARTETS, TRIPLETS, AND CHARACTERS

Brad Shutters, Sudheer Vakati, David Fernández-Baca

Modified from a paper submitted to the journal *Algorithm. Mol. Biol.*

Abstract

We study a long standing conjecture on the necessary and sufficient conditions for the compatibility of full multi-state characters: There exists a function $f(r)$ such that, for any set C of r-state characters, C is compatible if and only if every subset of $f(r)$ characters of C is compatible. We show that for every $r \geq 2$, there exists an incompatible set C of $\Omega(r^2)$ r-state characters such that every proper subset of C is compatible. This improves the previous lower bound of $f(r) \geq r$ given by Meacham (1983), and $f(4) \geq 5$ given by Habib and To (2011). For the case when $r = 3$, Lam, Gusfield and Sridhar (2011) recently showed that $f(3) = 3$. We give an independent proof of this result and completely characterize the sets of pairwise compatible 3-state characters by a single forbidden intersection pattern.

Our lower bound on $f(r)$ is proven via a result on quartet compatibility that may be of independent interest: For every $n \geq 4$, there exists an incompatible set Q of $\Omega(n^2)$ quartets over n labels such that every proper subset of Q is compatible. We show that such a set of quartets can have size at most 3 when $n = 5$, and at most $O(n^3)$ for arbitrary n. We contrast our results on quartets with the case of rooted triplets: For every $n \geq 3$, if R is an incompatible set of more than $n-1$ triplets over n labels, then some proper subset of R is incompatible. We show this bound is tight by exhibiting, for every $n \geq 3$, a set of $n-1$ triplets over n taxa such that R is incompatible, but every proper subset of R is compatible.

6.1 Introduction

The multi-state character compatibility (or perfect phylogeny) problem is a basic question in computational phylogenetics [Semple and Steel (2003)]. Given a set C of full characters, we are asked whether there exists a phylogenetic tree that displays every character in C; if so, C is said to be compatible, and incompatible otherwise. The problem is known to be NP-complete [Bodlaender et al. (1992); Steel (1992)], but certain special cases are known to be polynomially-solvable [Agarwala and Fernández-Baca (1994); Dress and Steel (1992); Gusfield (1991); Kannan and Warnow (1994, 1997); Lam et al. (2011); Shutters and Fernández-Baca (2012)]. See Fernández-Baca (2001) for more on the perfect phylogeny problem.

In this paper we study a long standing conjecture on the necessary and sufficient conditions for the compatibility of full multi-state characters. For rest of this chapter when we say character we are only referring to full characters.

Conjecture 1. *There exists a function $f(r)$ such that, for any set C of r-state characters, C is compatible if and only if every subset of $f(r)$ characters of C is compatible.*

If Conjecture 1 is true, it would follow that we can determine if any set C of r-state characters is compatible by testing the compatibility of each subset of $f(r)$ characters of C, and, in case of incompatibility, output a subset of at most $f(r)$ characters of C that is incompatible. This would allow us to reduce the character removal problem (i.e., finding a subset of characters to remove from C so that the remaining characters are compatible) to $f(r)$-hitting set which is fixed-parameter tractable [Niedermeier and Rossmanith (2003)].

A classic result on binary character compatibility shows that $f(2) = 2$; see [Buneman (1971); Estabrook et al. (1976); Gusfield (1991); Meacham (1983); Semple and Steel (2003)]. In 1975, Fitch [Fitch (1975, 1977)] gave an example of a set C of three 3-state characters such that C is incompatible, but every pair of characters in C is compatible; showing that $f(3) \geq 3$. In 1983, Meacham [Meacham (1983)] generalized this example to r-state characters for every $r \geq 3$ demonstrating a lower bound of $f(r) \geq r$ for all r; see also [Lam et al. (2011)]. For the case of $r = 3$, Lam, Gusfield, and Sridhar [Lam et al. (2011)] recently established that $f(3) = 3$.

While the previous results could lead one to conjecture that $f(r) = r$ for all r, Habib and

To [Habib and To (2011)] recently disproved this possibility by exhibiting a set C of five 4-state characters such that C is incompatible, but every proper subset of the characters in C are compatible, showing that $f(4) \geq 5$. They conjectured that $f(r) \geq r + 1$ for every $r \geq 4$.

The main result of this paper is to prove the conjecture stated in Habib and To (2011) by giving a quadratic lower bound on $f(r)$. Formally, we show that for every $r \geq 2$, there exists a set C of r-state characters such that all of the following conditions hold.

1. C is incompatible.
2. Every proper subset of C is compatible.
3. $|C| = \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$.

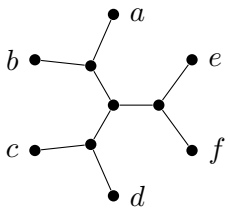
Therefore, $f(r) \geq \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$ for every $r \geq 2$.

Our proof relies on a new result on quartet compatibility we believe is of independent interest. We show that for every $n \geq 4$, there exists a set Q of quartets over a set of n labels such that all of the following conditions hold.

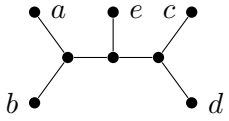
1. Q is incompatible.
2. Every proper subset of Q is compatible.
3. $|Q| = \lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$.

This is an improvement over the previous lower bound on the maximum cardinality of such an incompatible set of quartets of $n-2$ given in Steel (1992). We show that such a set of quartets can have size at most 3 when $n = 5$, and at most $O(n^3)$ for arbitrary n. We note here that the construction given in Habib and To (2011) showing that $f(4) \geq 5$ can be viewed as a special case of the construction given here when $n = 6$.

We study the compatibility of three-state characters further. The work of Lam et al. (2011) completely characterized the sets of pairwise compatible 3-state characters by the existence of one of four forbidden intersection patterns. An alternative characterization of this result was given in Shutters and Fernández-Baca (2012) and was partially derived using the results of Lam et al. (2011). In this paper, we give a proof that $f(3) = 3$ that is independent of the results



(a)



(b)

Figure 6.1: (a) shows a tree T witnessing that the quartets $q_1 = ab|ce$, $q_2 = cd|bf$, and $q_3 = ad|ef$ are compatible; T is also a witness that the characters $\chi_{q_1} = ab|ce|d|f$, $\chi_{q_2} = cd|bf|a|e$, and $\chi_{q_3} = ad|ef|b|c$ are compatible; (b) shows $T|\{a, b, c, d, e\}$.

in Lam et al. (2011), and we completely characterize the sets of pairwise compatible 3-state characters by a single forbidden intersection pattern.

We contrast our result on quartet compatibility with a result on the compatibility of rooted triplets: For every $n \geq 3$, if R is an incompatible set of triplets over n labels, and $|R| > n - 1$, then some proper subset of R is incompatible. We show this bound is tight by exhibiting, for every $n \geq 3$, a set of $n - 1$ triplets over n labels such that R is incompatible, but every proper subset of R is compatible.

6.2 Preliminaries

6.2.1 Quartet Rules

We now introduce *quartet (closure) rules* which were originally used in the contexts of psychology [Colonius and Schulze (1981)] and linguistics [Dekker, M. C. H. (1986)]. The idea is that for a collection Q of quartets, any tree that displays Q may also necessarily display another quartet $q \notin Q$, and if so we write $Q \vdash q$.

Example 4. Let $Q = \{ab|ce, ae|cd\}$. Then the tree of Figure 6.1(b) displays Q, and furthermore, it is easy to see that it is the only tree that displays Q. Hence, $Q \vdash ab|de$, $Q \vdash ab|cd$, and $Q \vdash be|cd$.

We use the following quartet rules in this paper:

$$\{ab|cd, ab|ce\} \vdash ab|de \tag{R1}$$

$$\{ab|cd, ac|de\} \vdash ab|ce \tag{R2}$$

For the purposes of this paper, we define the *closure* of an arbitrary collection Q of quartets, denoted Q^*, as the minimal set of quartets that contains Q, and has the property that if for some $q_1, q_2 \in Q^*$, $\{q_1, q_2\} \vdash q_3$ using either (R1) or (R2), then $q_3 \in Q^*$. Clearly, any tree that displays Q must also display Q^*. We will use the following lemma which follows by repeated application of (R1) and is formally proven in Dietrich et al. (2012).

Lemma 56. *Let Q be an arbitrary set of quartets with $\{x, y, z_1, \ldots, z_k\} \subseteq \mathcal{L}(Q)$. If*

$$\bigcup_{i=1}^{k-1} \{xy|z_i z_{i+1}\} \subseteq Q^* \ ,$$

then $xy|z_1 z_k \in Q^*$.

We refer the reader to [Semple and Steel (2003)] and [Grünewald and Huber (2007)] for more on quartet rules.

6.3 Incompatible Quartets

For every $s, t \geq 2$, we fix a set of labels $\mathcal{L}_{s,t} = \{a_1, a_2, \ldots, a_s, b_1, b_2, \ldots, b_t\}$ and define the set

$$Q_{s,t} = \{a_1 b_1 | a_s b_t\} \cup \bigcup_{i=1}^{s-1} \bigcup_{j=1}^{t-1} \{a_i a_{i+1} | b_j b_{j+1}\}$$

of quartets with $\mathcal{L}(Q_{s,t}) = \mathcal{L}_{s,t}$. We denote the quartet $a_1 b_1 | a_s b_t$ by q_0, and a quartet of the form $a_i a_{i+1} | b_j b_{j+1}$ by $q_{i,j}$.

Observation 6. *For all $s, t \geq 2$, $|Q_{s,t}| = (s-1)(t-1) + 1$.*

Lemma 57. *For all $s, t \geq 2$, $Q_{s,t}$ is incompatible.*

Proof. For each $i \in [s-1]$,

$$\bigcup_{j=1}^{t-1} \{a_i a_{i+1} | b_j b_{j+1}\} \subseteq Q_{s,t} \subseteq Q_{s,t}^*.$$

Then, by Lemma 56, it follows that for each $i \in [s-1]$, $a_i a_{i+1}|b_1 b_t \in Q^*_{s,t}$. So,

$$\bigcup_{i=1}^{s-1} \{b_1 b_t | a_i a_{i+1}\} \subseteq Q^*_{s,t}.$$

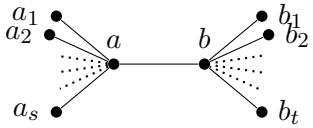
Then, again by Lemma 56, it follows that $b_1 b_t | a_1 a_s \in Q^*_{s,t}$. But then $\{a_1 b_1 | a_s b_t, b_1 b_t | a_1 a_s\} \subseteq Q^*_{s,t}$. It follows that any tree that displays $Q_{s,t}$ must display both $a_1 b_1 | a_s b_t$ and $b_1 b_t | a_1 a_s$. However, no such tree exists. Hence, $Q_{s,t}$ is incompatible. □

Lemma 58. *For all $s, t \geq 2$, every proper subset of $Q_{s,t}$ is compatible.*

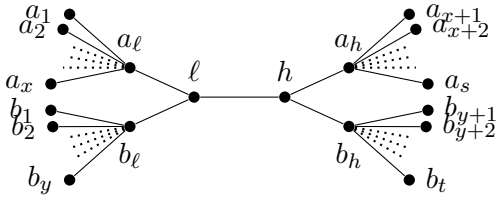
Proof. Since every subset of a compatible set of quartets is compatible, it suffices to show that for every $q \in Q_{s,t}$, $Q_{s,t} \setminus \{q\}$ is compatible. Let $q \in Q_{s,t}$. Either $q = q_0$ or $q = q_{x,y}$ for some $1 \leq x < s$ and $1 \leq y < t$. In either case, we exhibit a tree witnessing that $Q_{s,t} \setminus \{q\}$ is compatible.

Case 1. Suppose $q = q_0$. We build the tree T as follows: There is a node ℓ for each label $\ell \in \mathcal{L}_{s,t}$ and two additional nodes a and b along with the edge ab. There is an edge $a_x a$ for every $a_x \in \mathcal{L}_{s,t}$, and an edge $b_x b$ for every $b_x \in \mathcal{L}_{s,t}$. There are no other nodes or edges in T. See Figure 6.2(a) for an illustration. Now consider any quartet $q \in Q_{s,t} \setminus \{q_0\}$. Then $q = a_i a_{i+1}|b_j b_{j+1}$ for some $1 \leq i < s$ and $1 \leq j < t$. Then, the minimal subgraph of T connecting leaves with labels in $\{a_i, a_{i+1}, b_j, b_{j+1}\}$ is the quartet q. Hence T displays q.

Case 2. Suppose $q = q_{x,y}$ for some $1 \leq x < s$ and $1 \leq y < t$. We build the tree T as follows: There is a node ℓ for each label $\ell \in \mathcal{L}_{s,t}$ and six additional nodes a_ℓ, b_ℓ, ℓ, h, a_h,



(a) Case 1: a tree that displays $Q_{s,t} \setminus \{q_0\}$.



(b) Case 2: a tree that displays $Q_{s,t} \setminus \{q_{x,y}\}$.

Figure 6.2: Illustrating the proof of Lemma 58.

and b_h. There are edges $a_\ell\ell$, $b_\ell\ell$, ℓh, ha_h, and hb_h. For every $a_i \in \mathcal{L}_{s,t}$, there is an edge $a_i a_\ell$ if $i \leq x$, and an edge $a_i a_h$ if $i > x$. For every $b_j \in \mathcal{L}_{s,t}$ there is an edge $b_j b_\ell$ if $j \leq x$, and an edge $b_j b_h$ if $j > y$. There are no other nodes or edges in T. See Figure 6.2(b). Now consider any quartet $q \in Q_{s,t} \setminus \{q_{x,y}\}$. Either $q = q_0$ or $q = q_{i,j}$ where $i \neq x$ or $j \neq y$. If $q = q_0$, then the minimal subgraph of T connecting leaves with labels in $\{a_1, b_1, a_s, b_t\}$ is the subtree of T induced by the nodes in $\{a_1, a_\ell, \ell, b_\ell, b_1, a_s, a_h, h, b_h, b_t\}$. Suppressing all degree two vertices results in a tree that is the same as q_0. So T displays q. So assume that $q = a_i a_{i+1}|b_j b_{j+1}$ where $i \neq x$ or $j \neq y$. We define the following subset of the nodes in T:

$$V = \begin{cases} \{a_i, a_{i+1}, a_\ell, \ell, b_\ell, b_j, b_{j+1}\} & \text{if } i < x \text{ and } j < y, \\ \{a_i, a_{i+1}, a_\ell, \ell, b_y, b_\ell, h, b_h, b_{y+1}\} & \text{if } i < x \text{ and } j = y, \\ \{a_i, a_{i+1}, a_\ell, \ell, h, b_h, b_j, b_{j+1}\} & \text{if } i < x \text{ and } j > y, \\ \{a_x, a_\ell, \ell, h, a_h, a_{x+1}, b_\ell, b_j, b_{j+1}\} & \text{if } i = x \text{ and } j < y, \\ \{a_x, a_\ell, \ell, h, a_h, a_{x+1}, b_h, b_j, b_{j+1}\} & \text{if } i = x \text{ and } j > y, \\ \{a_j, a_{j+1}, a_h, h, \ell, b_\ell, b_j, b_{j+1}\} & \text{if } i > x \text{ and } j < y, \\ \{a_j, a_{j+1}, a_h, h, b_y, b_\ell, \ell, b_h, b_{y+1}\} & \text{if } i > x \text{ and } j = y, \\ \{a_j, a_{j+1}, a_h, h, b_h, b_j, b_{j+1}\} & \text{if } i > x \text{ and } j > y. \end{cases}$$

Now, the subgraph of T induced by the nodes in V is the minimal subgraph of T connecting leaves with labels in q. Suppressing all degree two vertices gives q. Hence, T displays q. □

With $s = \lfloor \frac{n}{2} \rfloor$ and $t = \lceil \frac{n}{2} \rceil$, Observation 6 and Lemmas 57 and 58 imply the following theorem.

Theorem 24. *For every integer $n \geq 4$, there exists a set Q of quartets over n taxa such that all of the following conditions hold.*

1. *Q is incompatible.*

2. *Every proper subset of Q is compatible.*

3. $|Q| = \lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$.

6.3.1 Incompatible Quartets on Five Taxa

When Q is a set of quartets over five taxa, we show that the set of quartets given by Theorem 24 is as large as possible. We hope that the technique used in the proof of the following theorem might be useful in proving tight bounds for $n > 5$.

Theorem 25. *If Q is an incompatible set of quartets over five taxa such that every proper subset of Q is compatible, then $|Q| \leq 3$.*

Proof. Let Q be an incompatible set of quartets with $\mathcal{L}(Q) = \{a, b, c, d, e\}$ and $q_0 = ab|cd \in Q$. We will show that Q contains an incompatible subset of at most three quartets. If Q contains two different quartets on the same four taxa, then Q must contain an incompatible pair of quartets. So, we may assume that each quartet is on a unique subset of four of the five taxa. Hence, every pair of quartets in Q shares three taxa in common. We have the following two cases.

> *Case 1:* Q contains at least one of the quartets $ac|be$, $ac|de$, $ad|be$, $ad|ce$, $ae|bc$, $ae|bd$, $bc|de$, or $bd|ce$. W.l.o.g. we may assume that Q contains $q_1 = ac|de$, as all other cases are symmetric. By (R2), $\{q_0, q_1\} \vdash ab|ce$. Then, by (R1), $\{q_0, q_1, ab|ce\} \vdash ab|de$. Then, again by (R1), $\{q_0, q_1, ab|ce, ab|de\} \vdash bc|de$. Now let $Q' = \{q_0, q_1, ab|ce, ab|de, bc|de\}$. Now, any quartet in Q must be either in Q' or be pairwise incompatible with a quartet in Q'. Since Q' is compatible, but by assumption, Q is incompatible, Q must contain a quartet q_2 that is pairwise incompatible with some quartet in Q'. Hence, $\{q_0, q_1, q_2\}$ is an incompatible subset of Q.
>
> *Case 2:* Q contains none of the quartets $ac|be$, $ac|de$, $ad|be$, $ad|ce$, $ae|bc$, $ae|bd$, $bc|de$, or $bd|ce$. Then every quartet in Q is either of the form $ab|xy$ where $\{x, y\} \neq \{c, d\}$, or $cd|xy$ where $\{x, y\} \neq \{a, b\}$. But then Q is compatible, contradicting our assumption that Q is incompatible.

In either case, the theorem holds. □

6.3.2 Incompatible Quartets on Arbitrarily Many Taxa

We say a set Q of compatible quartets is *redundant* if for some $q \in Q$, $Q \setminus \{q\} \vdash q$; otherwise, we say that Q is *irredundant*. The following lemma establishes a connection between sets of irredundant quartets and minimal sets of incompatible quartets.

Lemma 59. *If Q is incompatible, but every proper subset of Q is compatible, then every proper subset of Q is irredundant.*

Proof. Suppose that Q is incompatible and every proper subset of Q is compatible. Furthermore, suppose that some proper subset Q' of Q is redundant. Since every compatible superset of a redundant set of quartets is also redundant, we may assume w.l.o.g., that there is a unique quartet $q \in Q \setminus Q'$ (i.e., $|Q| = |Q'| + 1$). Since Q' is redundant, there exists a $q' \in Q'$ such that $Q' \setminus \{q'\} \vdash q'$. But then $(Q' \setminus \{q'\}) \cup \{q\}$ is incompatible, contradicting that every proper subset of Q is compatible. □

It follows from Lemma 59 that any upper bound on the maximum cardinality of an irredundant set of quartets can be used to place an upper bound on the maximum cardinality of a set of quartets satisfying the first two conditions of Theorem 24. The theorem follows from Dietrich et al. (2012).

Theorem 26. *Let Q be a set of quartets over a set of n taxa. If Q is irredundant, then Q has cardinality at most $(n-3)(n-2)^2/3$.*

Lemma 59 together with Theorem 26 gives the following upper bound on the maximum cardinality of a set Q of quartets over $n > 5$ taxa that satisfies the first two conditions of Theorem 24.

Theorem 27. *Let Q be a set of incompatible quartets over a set of n taxa such that every proper subset of Q is compatible. Then $|Q| \leq (n-3)(n-2)^2/3 + 1$.*

6.4 Incompatible Characters

There is a natural correspondence between quartet compatibility and character compatibility that we now describe. Let Q be a set of quartets, $n = |\mathcal{L}(Q)|$, and $r = n - 2$. For each

$q = ab|cd \in Q$, we define the r-state character corresponding to q, denoted χ_q, as the character where a and b have state 0 for χ_q; c and d have state 1 for χ_q; and, for each $\ell \in \mathcal{L}(Q) \setminus \{a, b, c, d\}$, there is a state s of χ_q such that ℓ is the only label with state s for character χ_q (see Example 5). We define the set of r-state characters corresponding to Q by $C_Q = \bigcup_{q \in Q} \{\chi_q\}$.

Example 5. Consider the quartets and characters given in Figure 6.1(a): χ_{q_1} is the character corresponding to q_1, χ_{q_2} is the character corresponding to q_2, and χ_{q_3} is the character corresponding to q_3.

The following lemma relating quartet compatibility to character compatibility is well known [Steel (2012)], and its proof is omitted here.

Lemma 60. *A set Q of quartets is compatible if and only if C_Q is compatible.*

The next theorem allows us to use our result on quartet compatibility to establish a lower bound on $f(r)$.

Theorem 28. *Let Q be a set of incompatible quartets over n labels such that every proper subset of Q is compatible, and let $r = n - 2$. Then, there exists a set C of $|Q|$ r-state characters such that C is incompatible, but every proper subset of C is compatible.*

Proof. We claim that C_Q is such a set of incompatible r-state characters. Since for two quartets $q_1, q_2 \in Q$, $\chi_{q_1} \neq \chi_{q_2}$, it follows that $|C_Q| = |Q|$. Since Q is incompatible, it follows by Lemma 60 that C_Q is incompatible. Let C' be any proper subset of C. Then, there is a proper subset Q' of Q such that $C' = C_{Q'}$. Then, since Q' is compatible, it follows by Lemma 60 that C' is compatible. □

Theorem 24 together with Theorem 28 gives the main theorem of this paper.

Theorem 29. *For every integer $r \geq 2$, there exists a set C of r-state characters such that all of the following hold.*

1. *C is incompatible.*

2. *Every proper subset of C is compatible.*

3. $|C| = \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$.

Proof. By Theorem 24 and Observation 6, there exists a set Q of $\lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$ quartets over $r+2$ labels that that are incompatible, but every proper subset is compatible, namely $Q_{\lfloor \frac{r+2}{2} \rfloor, \lceil \frac{r+2}{2} \rceil}$. The theorem follows from Theorem 28. □

The quadratic lower bound on $f(r)$ follows from Theorem 29.

Corollary 5. $f(r) \geq \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$.

6.4.1 Three-State Characters

In the remainder of this section we focus on the case when $r = 3$, and thus, fix C to be an arbitrary set of 3-state characters over a set S of taxa. Lam, Gusfield, and Sridhar [Lam et al. (2011)] recently established that $f(3) = 3$, and they completely characterized the sets of pairwise compatible 3-state characters by the existence of one of four forbidden intersection patterns. We give an independent proof that $f(3) = 3$. We then completely characterize the sets of pairwise compatible 3-state characters by a single forbidden intersection pattern. Our proof uses several structural results from the algorithm for the three-state perfect phylogeny problem given by Kannan and Warnow [Kannan and Warnow (1994)].

6.4.1.1 The Algorithm of Kannan and Warnow

The algorithm of Kannan and Warnow (1994) takes a divide and conquer approach to determining the compatibility of a set of three-state characters. An instance is reduced to subproblems by finding a partition S_1, S_2 of the taxon set S of C with both of the following properties:

1. $2 \leq |S_i| \leq n - 2, i = 1, 2$.

2. Whenever C is compatible S there is a perfect phylogeny P that contains an edge e whose removal breaks P into subtrees P_1 and P_2 with $\mathcal{L}(P_i) = S_i, i = 1, 2$.

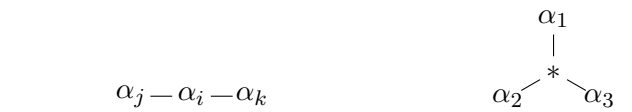
A partition of S satisfying both of these properties is a *legal partition*, and the following theorem shows that finding such a partition for a given set of characters is the crux of the algorithm.

Theorem 30. *[Kannan and Warnow (1994)] Given a set C of three state characters, we can in $O(nk)$ time either find a legal partition of S of determine that the set of characters is incompatible.*

6.4.1.2 Finding a Legal Partition

We now discuss the manner in which such a legal partition is found for a set of three-state characters C. Let T be a tree witnessing that C is compatible. The *canonical labeling* of T is the labeling where, for each internal node v of T, and each character $\alpha \in C$, if there are leaves x and y in different components of $T - \{v\}$ such that $\alpha(x) = \alpha(y)$, then $\alpha(v) = \alpha(x)$; otherwise $\alpha(v) = *$ where $*$ denotes a *dummy* state for C. Note that such a labeling of T always exists and is unique. We will assume that every compatible tree for C is canonically labeled.

The *tree-structure* for a character α in T is formed by repeatedly contracting edges of T connecting nodes that have the same state (other than $*$) for α. Note that this tree does not depend on the sequence of edge-contractions and is thus well defined. Furthermore, there is exactly one node for each state (other than the dummy state) of α, and each node labeled by $*$ has degree at least three. A tree-structure for α that is formed from some compatible tree for C is called a *realizable tree-structure* for α. There are four possible realizable tree-structures for a three-state character α which are shown in Figure 6.3.



(a) A path P^i for each $i \in \{1, 2, 3\}$. (b) A star S^*.

Figure 6.3: The four possible realizable tree-structures for a three-state character α.

To find a realizable tree structure for a character α, the algorithm examines the pairwise intersection patterns of α with every other character $\beta \in C$, and applies the following rules to

rule out possible tree structures for α.

Rule 1 *Let α and β be two characters of C. If, under some relabeling of the states of α and β, we have that $\alpha_1 \subseteq \beta_1$, $\alpha_2 \cap \beta_2 \neq \emptyset$, and $\alpha_3 \cap \beta_2 \neq \emptyset$, then P^1 is not a realizable tree-structure for α. If this is the case, we say that α and β match Rule 1 with respect to α_1.*

Rule 2[1] *Let α and β be two characters of C. If, under some relabeling of the states of α and β, we have that $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_2 \cap \beta_1 \neq \emptyset$, $\alpha_2 \cap \beta_2 \neq \emptyset$, and $\alpha_3 \cap \beta_2 \neq \emptyset$, then P^2 is the only possible realizable tree-structure for α. If this is the case, we say that α and β match Rule 2 with respect to α_2.*

The set Q_α^C of *candidate* tree-structures for α are all of those possible tree-structures for α that are not ruled out after comparing the intersection pattern of α with every other character in C and applying Rules 1 and 2.

The following theorem which follows from Kannan and Warnow (1994) shows that a legal partition is found by choosing an arbitrary $\alpha \in C$ for which $Q_\alpha^C \neq \emptyset$. Furthermore, if there is an $\alpha \in C$ for which $Q_\alpha^C = \emptyset$, then C is incompatible.

Theorem 31. *[Kannan and Warnow (1994)] If $Q_\alpha^C \neq \emptyset$, then we can find a legal partition of S.*

Corollary 6. *A set C of 3-state characters is compatible if and only if $Q_\alpha^C \neq \emptyset$ for every $\alpha \in C$.*

6.4.1.3 Tight Bounds on Three-State Character Compatibility

We use Corollary 6 to give upper bounds on the maximum cardinality of a minimal set of incompatible three-state characters.

Theorem 32. *Let C be a set of three-state characters on species set S. Then C is incompatible if and only if there exists a character $\alpha \in C$, and two distinct states α_i and α_j of α, such that both of the following hold:*

[1] *Rule 2 was state incorrectly in Kannan and Warnow (1994)*

1. *There is a $\beta \in C$ where the intersection pattern of α and β matches Rule 2 with respect to α_i.*

2. *There is a $\gamma \in C$ where the intersection pattern of α and γ matches Rule 2 with respect to α_j.*

Proof. ($\Rightarrow$) If C is pairwise incompatible, then by Corollary 1, there is a pair $\alpha, \beta \in C$ whose intersection graph contains a cycle. Since the intersection graph is bipartite, this cycle must have length at least four and contain at least two states of each character. Let α_i and α_j be the two states of α on this cycle. Then, the intersection pattern of α and β matches Rule 2 with respect to both α_i and α_j, and so the theorem holds. So we may assume that C is incompatible but pairwise compatible.

It follows from Corollary 6 that there exists an $\alpha \in C$ such that $Q_\alpha^C = \emptyset$. Then there must exist a character $\beta \in C$ such that the intersection pattern of α and β matches Rule 2 with respect to some state α_i of α; otherwise $S^* \in Q_\alpha^C$. Hence, $Q_\alpha^C \subseteq \{P^i\}$. Then, since $Q_\alpha^C = \emptyset$, there must be a character $\gamma \in C$ such that the intersection pattern of α and γ places a constraint on Q_α^C that prevents Q_α^C from containing P^i. There are two possibilities.

Case 1: There is a state α_j of α where $j \neq i$ and the intersection pattern of α and γ matches Rule 2 with respect to α_j. In this case the theorem holds.

Case 2: The intersection pattern of α and γ matches Rule 1 with respect to α_i. W.l.o.g., we fix $i = 1$, and relabel the states of α, β, and γ so that $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_1 \cap \beta_2 \neq \emptyset$, $\alpha_2 \cap \beta_1 \neq \emptyset$, $\alpha_3 \cap \beta_2 \neq \emptyset$, $\alpha_1 \subseteq \gamma_1$, $\alpha_2 \cap \gamma_2 \neq \emptyset$, and $\alpha_3 \cap \gamma_2 \neq \emptyset$. Such a labeling exists since, by assumption, α and β matches Rule 2 with respect to α_1, and α and γ matches Rule 1 with respect to α_1.

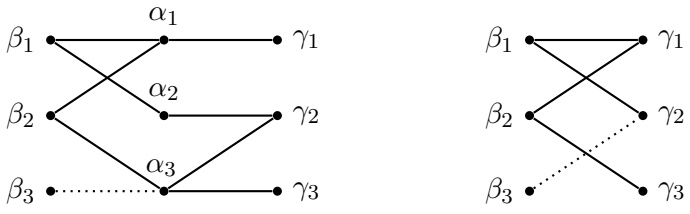


Figure 6.4: Illustrating the proof of Theorem 32.

If $\alpha_2 \cap \gamma_1 \neq \emptyset$, then the intersection pattern of α and γ matches Rule 2 with respect to α_2, in which case the theorem holds. If $\alpha_3 \cap \gamma_1 \neq \emptyset$, then the intersection pattern of α and γ matches Rule 2 with respect to α_3, in which case the theorem holds. So we may assume hat $\alpha_1 = \gamma_1$. Now, since $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_1 \cap \beta_2 \neq \emptyset$, and $\alpha_1 = \gamma_1$, we have that both $\beta_1 \cap \gamma_1 \neq \emptyset$ and $\beta_2 \cap \gamma_2 \neq \emptyset$.

γ_3 must have a nonempty intersection with at least one state of α, and since $\alpha_1 = \gamma_1$, we have that $\alpha_1 \cap \gamma_3 = \emptyset$. So γ_3 has a nonempty intersection with either α_2 or α_3. Due to the symmetry of the intersection graph of α and β, we may assume, w.l.o.g., that $\alpha_3 \cap \gamma_3 \neq \emptyset$.

By assumption, $\alpha_2 \cap \gamma_1 = \emptyset$, and if $\alpha_2 \cap \gamma_3 \neq \emptyset$, then the intersection graph of α and β contains a cycle, contradicting our assumption that C is pairwise compatible. So we may assume that $\alpha_2 \subset \gamma_2$. Then, since $\beta_1 \cap \alpha_2 \neq \emptyset$, we have that $\beta_1 \cap \gamma_2 \neq \emptyset$.

Let $s \in \alpha_3 \cap \beta_2$. Since, by assumption, $\alpha_3 \cap \gamma_1 = \emptyset$, we have that either $s \in \gamma_2$ or $s \in \gamma_3$. However, if $s \in \gamma_2$, then $\beta_2 \cap \gamma_2 \neq \emptyset$ and intersection graph of β and γ contains a cycle, contradicting our assumption that C is pairwise compatible. Hence $s \in \gamma_3$ and $\beta_2 \cap \gamma_3 \neq \emptyset$.

We have now established all of the edges of the intersection graph of α, β, and γ represented by the solid edges in Figure 6.4. Now, let $s_5 \in \alpha_3 \cap \gamma_2$. Now s_5 must be in some state of β. If $s_5 \in \beta_1$, then $s_5 \in \beta_1 \cap \alpha_3$ and the intersection graph of β and α contains a cycle, contradicting our assumption that C is pairwise compatible. If $s_5 \in \beta_2$, then $s_5 \in \beta_2 \cap \gamma_2$, and the intersection graph of β and γ contains a cycle, again contradicting our assumption that C is pairwise compatible. Hence $s_5 \in \beta_3$. Then, we have that $s_5 \in \beta_3 \cap \alpha_3$ and $s_5 \in \beta_3 \cap \gamma_2$, witnessing the dotted edges in Figure 6.4. So we have that the intersection pattern of β and α matches Rule 2 with β_2 as witness, and the intersection pattern of β and γ matches Rule 2 with β_1 as witness. Hence the theorem holds. □

Note that in the statement of Theorem 32, the characters β and γ are not necessarily distinct. In cases where they are not distinct, C contains an incompatible pair.

Corollary 7. *A set C of 3-state characters is compatible if and only if every subset of at most three characters of C is compatible.*

In Lam et al. (2011), it was also shown that we can determine the compatibility of a

pairwise compatible set C of three-state characters by testing the intersection patterns of C for the existence of one of a set of four forbidden patterns. As a corollary to Theorem 32, we have that a single forbidden pattern suffices to determine the compatibility of C.

Corollary 8. *A pairwise compatible set C of 3-state characters is compatible if and only if the partition intersection graph of C does not contain, up to relabeling of characters and states, the subgraph of Figure 6.5.*

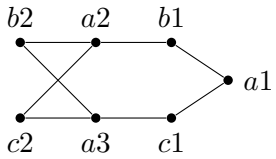


Figure 6.5: The forbidden subgraph for 3-state character compatibility.

Note that each edge of the graph of Figure 6.5 has one endpoint which is a state in α. It follows that we can find such a subgraph in the partition intersection graph of C by testing the intersection pattern of each pair of characters in C [Shutters and Fernández-Baca (2012)]. Furthermore, all p occurrences of the forbidden subgraph in the intersection graph of m characters on n taxa can be found in $O(m^2n+p)$ time. Whereas the forbidden subgraph given here is witnessed by eight taxa (or edges), each of the four forbidden subgraphs of Lam et al. (2011) are witnessed by five taxa, making them better suited for taxon removal problems.

6.5 Incompatible Triplets

The following theorems follow from the connection between collections of unrooted trees with at least one common label across all the trees, and collections of rooted trees [Steel (1992)].

Theorem 33. *Let Q be a collection of quartets where every quartet in Q shares a common label ℓ. Let R be the set of triplets such that there exists a triplet $ab|c$ in R if and only if there exists a quartet $ab|c\ell$ in Q. Then, Q is compatible if and only if R is compatible.*

Let R be a collection of triplets. For a subset $S \subseteq \mathcal{L}(R)$, we define the graph $[R, S]$ as the graph having a vertex for each label in S, and an edge $\{a, b\}$ if and only if $ab|c \in R$ for some

$c \in S$. The following theorem is from page 439 of Bryant and Steel (1995).

Theorem 34. *A collection R of rooted triplets is compatible if and only if $[R, S]$ is not connected for every $S \subseteq \mathcal{L}(R)$ with $|S| \geq 3$.*

Corollary 9. *Let R be a set of rooted triplets such that R is incompatible but every proper subset of R is compatible. Then, $[R, \mathcal{L}(R)]$ is connected.*

We now contrast our result on quartet compatibility with a result on triplets.

Theorem 35. *For every $n \geq 3$, if R is an incompatible set of triplets over n labels, and $|R| > n - 1$, then some proper subset of R is incompatible.*

Proof. For sake of contradiction, let R be a set of triplets such that R is incompatible, every proper subset of R is compatible, $|\mathcal{L}(R)| = n$, and $|R| > n - 1$. The graph $[R, \mathcal{L}(R)]$ will contain n vertices and at least n edges. Since each triplet in R is distinct, there will be a cycle C of length at least three in $[R, \mathcal{L}(R)]$. Since R is incompatible but every proper subset of R is compatible, by Corollary 9, $[R, \mathcal{L}(R)]$ is connected.

Consider any edge e in the cycle C. Let t be the triplet that contributed edge e in $[R, \mathcal{L}(R)]$. Let $R' = R \setminus t$. Since the graph $[R, \mathcal{L}(R)] - e$ is connected, $[R', \mathcal{L}(R')]$ is connected. By Theorem 34, R' is incompatible. But $R' \subset R$, contradicting that every proper subset of R is compatible. □

To show the bound is tight, we first prove a more restricted form of Theorem 24.

Theorem 36. *For every $n \geq 4$, there exists a set of quartets Q with $|\mathcal{L}(Q)| = n$, and a label $\ell \in L(Q)$, such that all of the following hold.*

1. *Every $q \in Q$ contains a leaf labeled by ℓ.*
2. *Q is incompatible.*
3. *Every proper subset of Q is compatible.*
4. *$|Q| = n - 2$.*

Proof. Consider the set of quartets $Q_{2,n-2}$. From Lemmas 57 and 58, $Q_{2,n-2}$ is incompatible but every proper subset of $Q_{2,n-2}$ is compatible. The set $Q_{2,n-2}$ contains exactly $n-2$ quartets. From the construction, there are two labels in $\mathcal{L}_{2,n-2}$ which are present in all the quartets in $Q_{2,n-2}$. Set one of them to be ℓ. □

The following is a consequence of Theorems 36 and 33.

Corollary 10. *For every $n \geq 3$, there exists a set R of triplets with $|\mathcal{L}(R)| = n$ such that all of the following hold.*

1. *R is incompatible.*
2. *Every proper subset of R is compatible.*
3. *$|R| = n - 1$.*

The generalization of the Fitch-Meacham examples given in Lam et al. (2011) can also be expressed in terms of triplets. For any $r \geq 2$, let $L = \{a, b_1, b_2, \cdots, b_r\}$. Let

$$R_r = ab_r|b_1 \cup \bigcup_{i=1}^{r-1} ab_i|b_{i+1}$$

Let $Q = \{ab|c\ell : ab|c \in R_r\}$ for some label $\ell \notin L$. The set C_Q of r-state characters corresponding to the quartet set Q is exactly the set of characters built for r in Lam et al. (2011). In the partition intersection graph of C_Q, (following the terminology in Lam et al. (2011)) labels ℓ and a correspond to the end cliques and the rest of the r labels $\{b_1, b_2, \cdots, b_r\}$ correspond to the r tower cliques. From Lemma 60 and Theorem 33, R_r is compatible if and only of Q is compatible.

6.6 Conclusion

We have shown that for every $r \geq 2$, $f(r) \geq \lfloor \frac{r}{2} \rfloor \cdot \lceil \frac{r}{2} \rceil + 1$, by showing that for every $n \geq 4$, there exists an incompatible set Q of $\lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$ quartets over a set of n labels such that every proper subset of Q is compatible. Previous results [Buneman (1971); Estabrook et al. (1976); Gusfield (1991); Lam et al. (2011); Meacham (1983); Semple and Steel (2003)],

along with our discussion in Section 6.4, show that our lower bound on $f(r)$ is tight for $r = 2$ and $r = 3$. For quartets, our discussion in Section 6.3 gives an upper bound on the maximum cardinality of a minimal set of incompatible quartets. However, this argument does not extend to multi-state characters. Indeed, an upper bound on the maximum cardinality of a minimal set of incompatible r-state characters remains a central open question. We give the following conjecture.

Conjecture 2. $f(r) \in \Theta(r^2)$.

A less ambitious goal would be to narrow the gap between the upper bound of $O(n^3)$ and lower bound of $\Omega(n^2)$ on the maximum cardinality of a minimal incompatible set of quartets over n taxa given in Section 6.3. Note that, due to Theorem 28, a proof of Conjecture 2 would also show that the number of incompatible quartets given in the statement of Theorem 24 is also as large as possible.

Acknowledgements

We thank Sylvain Guillemot, Mike Steel, and Rob Gysel for valuable comments. This work was supported in part by the National Science Foundation under grants CCF-1017189 and DEB-0829674.

CHAPTER 7. FURTHER RESEARCH

7.1 Tree Compatibility and Agreement Supertrees

We characterized the tree compatibility problem in terms of triangulations of the display graph. We introduced the modified display graph and studied it's properties. We gave the obstruction for legal triangulation of modified display graphs of two trees. We then characterized both compatible and agreement supertree problems in terms of cuts of the display graph. We gave FPT algorithms for the *AST-EC* and *AST-TR* problems. The following problems remain open.

1. A linear time fixed parameter tractable algorithm for tree compatibility and agreement supertree problems using any of the derived characterizations.

2. Enumeration of all possible tree compatibility obstructions in display graphs for three trees.

3. For any given profile $\mathcal{P}$ with a finite number of input trees, does $G(\mathcal{P})$ have finite number of obstructions for tree compatibility ?

4. The following is closely related to the previous question. Let $\mathcal{P} = \{T_1, T_2, \cdots, T_k\}$ be a profile of incompatible trees. Does there exist a function $g(k)$ such that for some $U \subseteq \mathcal{L}(\mathcal{P})$ and $|U| \leq g(k)$, $\{T_1|U, T_2|U, \cdots, T_k|U\}$ is incompatible.

5. Analogs of *AST-EC* and *AST-TR* problems for unrooted trees. Similarly, analogs of *AST-EC* and *AST-TR* problems for tree compatibility.

7.2 Perfect Phylogeny Conjecture

We showed the relationship of triplet and quartet compatibility to perfect phylogeny problem. A collection $\mathcal{Q}$ of quartets is a *quartet obstruction*, if $\mathcal{Q}$ is incompatible but every subset of $\mathcal{Q}$ is compatible. We have shown that the lower bound of a quartet obstruction with n labels is $\Omega(n^2)$. We used the constructed obstruction to show that $f(r) \geq r^2$. The following problems remain open.

1. $f(r) \in \theta(r^2)$?
2. $f(4) = 5$?
3. Is the derived bound on quartet obstructions tight ?

LIST OF PUBLICATIONS

To be submitted:

1. **Sudheer Vakati**, David Fernández-Baca. Graph-theoretic characterizations of agreement and compatibility of unrooted trees.

2. **Sudheer Vakati**, David Fernández-Baca. Tree compatibility with display graphs.

Journal:

1. David Fernández-Baca, Sylvain Guillemot, Brad Shutters, **Sudheer Vakati**. Fixed-parameter algorithms for finding agreement supertrees. *SIAM Journal of Computing*, under review (2013).

2. Brad Shutters, **Sudheer Vakati**, David Fernández-Baca. Incompatible quartets, triplets, and characters. *Algorithms for Molecular Biology*, 8:11 (2013).

3. **Sudheer Vakati**, David Fernández-Baca. Graph triangulations and the compatibility of unrooted phylogenetic trees. *Applied Mathematics Letters*, 24(5):719-723 (2011).

Conference:

1. David Fernández-Baca, Sylvain Guillemot, Brad Shutters, **Sudheer Vakati**. Fixed-parameter algorithms for finding agreement supertrees. In *23rd Annual Symposium on Combinatorial Pattern Matching*, volume 7354 of *Lecture Notes in Computer Science*, pages 373-384, 2012.

2. Brad Shutters, **Sudheer Vakati**, David Fernández-Baca. Incompatible quartets, triplets, and characters. In *12th Workshop on Algorithms in Bioinformatics*, volume 7534 of *Lecture Notes in Computer Science*, pages 190-200,2012.

BIBLIOGRAPHY

Agarwala, R. and Fernández-Baca, D. (1994). A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224. 2, 94

Aho, A. V., Sagiv, Y., Szymanski, T. G., and Ullman, J. D. (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.*, 10(3):405–421. 1, 12, 23, 63

Arnborg, S., Lagergren, J., and Seese, D. (1991). Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340. 1

Berry, V. and Nicolas, F. (2007). Maximum agreement and compatible supertrees. *J. Discrete Algorithms*, 5(3):564–591. 64, 92

Bininda-Emonds, O. R. P., editor (2004). *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 4 of *Ser. on Comp. Biol.* Springer. 63

Bodlaender, H., Fellows, M., and Warnow, T. (1992). Two strikes against perfect phylogeny. In Kuich, W., editor, *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 273–283. Springer. 2, 94

Bordewich, M., Huber, K. T., and Semple, C. (2005). Identifying phylogenetic trees. *Discrete Mathematics*, 300(1-3):30–43. 31

Bouchitté, V. and Todinca, I. (2001). Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232. 6

Bousquet, N., Daligault, J., and Thomassé, S. (2011). Multicut is FPT. In *STOC 2011*, pages 459–468. ACM. 92

Bryant, D. (2001). Optimal agreement supertrees. In *JOBIM 2000*, volume 2066 of *Lect. Notes in Comp. Sc.*, pages 24–31. Springer. 63

Bryant, D. and Lagergren, J. (2006). Compatibility of unrooted phylogenetic trees is FPT. *Theor. Comput. Sci.*, 351:296–302. 1, 8, 12, 14, 15, 16, 29, 38

Bryant, D. and Steel, M. (1995). Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16:425–453. 109

Buneman, P. (1971). The recovery of trees from measures of dissimilarity. In *Mathematics in the Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, Edinburgh. 7, 94, 110

Buneman, P. (1974). A characterisation of rigid circuit graphs. *Discrete Math.*, 9:205–212. 10, 12

Colonius, H. and Schulze, H. H. (1981). Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology*, 34(2):167–180. 96

Courcelle, B. (1990). The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75. 1

Dekker, M. C. H. (1986). Reconstruction methods for derivation trees. Master's thesis, Vrije Universiteit, Amsterdam, Netherlands. 96

Dietrich, M., McCartin, C., and Semple, C. (2012). Bounding the maximum size of a minimal definitive set of quartets. *Information Processing Letters*, 112(16):651–655. 97, 101

Dress, A. and Steel, M. (1992). Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6. 2, 94

Estabrook, G. F., Johnson, J., and McMorris, F. R. (1976). A mathematical foundation for the analysis of cladistic character compatibility. *Mathematical Biosciences*, 29(1-2):181–187. 94, 110

Fernández-Baca, D. (2001). The Perfect Phylogeny Problem. In *Steiner Trees in Industry*, pages 203–234. Kluwer. 94

Fitch, W. M. (1975). Toward finding the tree of maximum parsimony. In *Proceedings of the 8th International Conference on Numerical Taxonomy*, pages 189–230. 94

Fitch, W. M. (1977). On the problem of discovering the most parsimonious tree. *The American Naturalist*, 111(978):223–257. 94

Gordon, A. D. (1986). Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. *J. Classif.*, 9:335–348. 12, 63

Grünewald, S. and Huber, K. T. (2007). Identifying and defining trees. In Gascuel, O. and Steel, M., editors, *Reconstructing Evolution: New Mathematical and Computational Advances*. Oxford University Press. 97

Grunewald, S., Humphries, P. J., and Semple, C. (2008). Quartet compatibility and the quartet graph. *The Electronic Journal of Combinatorics*, 15(1):R103. 12

Guillemot, S. and Berry, V. (2010). Fixed-parameter tractability of the maximum agreement supertree problem. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 7(2):342–353. 2, 64, 66, 92

Gusfield, D. (1991). Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28. 2, 94, 110

Gusfield, D. (2009). The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. In Batzoglou, S., editor, *RECOMB*, volume 5541 of *Lecture Notes in Computer Science*, pages 236–252. Springer. 12, 40

Gysel, R., Stevens, K., and Gusfield, D. (2012). Reducing problems in unrooted tree compatibility to restricted triangulations of intersection graphs. In *WABI*, pages 93–105. 2, 9, 38, 39, 40, 43

Habib, M. and To, T.-H. (2011). On a conjecture of compatibility of multi-states characters. In Przytycka, T. and Sagot, M.-F., editors, *Algorithms in Bioinformatics*, volume 6833 of *Lecture Notes in Computer Science*, pages 116–127. Springer. 3, 95

Heggernes, P. (2005). Treewidth, partial k-trees, and chordal graphs. Partial curriculum in INF334 — Advanced algorithmical techniques, Department of Informatics, University of Bergen, Norway. 5, 6

Heggernes, P. (2006). Minimal triangulations of graphs: A survey. *Discrete Math.*, 306(3):297 – 317. 6

Hoang, V. T. and Sung, W.-K. (2011). Improved algorithms for maximum agreement and compatible supertrees. *Algorithmica*, 59(2):195–214. 64, 92

Jansson, J., Ng, J. H.-K., Sadakane, K., and Sung, W.-K. (2005). Rooted maximum agreement supertrees. *Algorithmica*, 43(4):293–307. 64

Kannan, S. and Warnow, T. (1994). Inferring Evolutionary History From DNA Sequences. *SIAM Journal on Computing*, 23(4):713–737. 2, 94, 103, 104, 105

Kannan, S. and Warnow, T. (1997). A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763. 94

Kao, M.-Y. (2007). *Encyclopedia of algorithms*. Springer, New York. 64

Lam, F., Gusfield, D., and Sridhar, S. (2011). Generalizing the Splits Equivalence Theorem and Four Gamete Condition: Perfect Phylogeny on Three-State Characters. *SIAM Journal on Discrete Mathematics*, 25(3):1144–1175. 2, 94, 95, 96, 103, 107, 108, 110

Linder, C. R. and Rieseberg, L. H. (2004). Reconstructing patterns of reticulate evolution in plants. *Am. J. Bot.*, 91(10):1700–8. 63

Maddison, W. P. (1997). Gene trees in species trees. *Systematic Biology*, 46(3):523–536. 63

Marx, D. and Razgon, I. (2011). Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *STOC 2011*, pages 469–478. ACM. 92

McMorris, F. R., Warnow, T. J., and Wimer, T. (1994). Triangulating vertex colored graphs. *SIAM Journal on Discrete Mathematics*, 7(2). Preliminary version in 4th Annual Symposium on Discrete Algorithms, Austin, Texas, 1993. 12

Meacham, C. A. (1983). Theoretical and computational considerations of the compatibility of qualitative taxonomic characters. In *Numerical Taxonomy*, volume G1 of *Nato ASI series*. Springer. 2, 94, 110

Ng, M. and Wormald, N. (1996). Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1–2):19–31. 1, 63

Niedermeier, R. and Rossmanith, P. (2003). An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102. 94

Parra, A. and Scheffler, P. (1997). Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Appl. Math.*, 79(1-3):171–188. 4, 6

Scornavacca, C. (2009). *Supertree methods for phylogenomics*. PhD thesis, Univ. of Montpellier II, Montpellier, France. 63

Semple, C. and Steel, M. (2003). *Phylogenetics*. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford. 7, 47, 94, 97, 110

Shutters, B. and Fernández-Baca, D. (2012). A simple characterization of the minimal obstruction sets for three-state perfect phylogenies. *Applied Mathematics Letters*, 25(9):1226–1229. 94, 95, 108

Steel, M. (2012). Personal communications. 102

Steel, M. A. (1992). The complexity of reconstructing trees from qualitative characters and subtrees. *J. Classif.*, 9:91–116. 1, 2, 12, 23, 31, 63, 94, 95, 108

Vakati, S. and Fernández-Baca, D. (2011). Graph triangulations and the compatibility of unrooted phylogenetic trees. *Appl. Math. Lett.*, 24(5):719–723. 38, 55, 61