FIXED-PARAMETER ALGORITHM FOR HYBRIDIZATION NUMBER OF TWO MULTIFURCATING TREES

by

Zhijiang Li

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2014

Table of Contents

List of Tables

List of Figures

Abstract

Phylogenetic trees are used to represent the evolution of a set of species. However, this history may include reticulation events where taxa acquire genes from more than one ancestor or from contemporaries, which leads to non-tree-like ancestry relationships that cannot be represented using trees alone. Hybridization networks can be used to model such histories.

The hybridization number is NP-hard to compute and we present a fixed-parameter algorithm to compute the hybridization number (and a corresponding network) of two multifurcating trees. The running time of our algorithm is $O(4.83^k \cdot n)$, where k is the computed hybridization number, which is more efficient than two previous algorithms with complexity $O(2^n \cdot poly(n))$ and $O((6^k k!) \cdot poly(n))$, respectively. We verified the practicality of our algorithm by implementing it and evaluating its performance on real data. Our algorithm is obtained using a non-trivial combination of novel ideas with techniques from previous algorithms for multifurcating SPR distance and binary hybridization number.

List of Abbreviations Used

AAF Acyclic Agreement Forest

AF Agreement Forest

DFVS Directed Feedback Vertex Set

FPT Fixed-Parameter Tractability/Fixed-Parameter Tractable

LCA Lowest Common Ancester

LGT Lateral Gene Transfer

MAAF Maximum Acyclic Agreement Forest

MAF Maximum Agreement Forest

SPR Subtree Prune-and-Regraft

Acknowledgements

First and foremost, I wish to thank my supervisor, Norbert Zeh. I am grateful for his enlightening advice and detailed collaboration. Particularly, I would like to thank him for his attention to detail, dedication, passion for research, and most importantly, his patience with his students.

Special thanks should be given to Chris Whidden for helping with my thesis, including introducing me to important research background, providing test data, and offering advice on implementation.

I also want to thank Robert Beiko and the members of the Beiko lab for providing me with the biological background and some biological data.

Finally, I wish to thank my father and mother for their whole-hearted support of my graduate studies.

Chapter 1

Introduction

Phylogenetic trees (phylogenies, or evolutionary trees) are used to represent the inferred evolutionary relationships among various biological species. In such a tree, extant species are represented as leaves, common ancestors are represented as internal nodes. Generally, each node is called a taxonomic unit or taxon, and internal nodes are called hypothetical taxa because they cannot be directly observed. The distance between two nodes in an evolutionary tree is related to their evolutionary distance, such as time or gene mutations. For example, Figure 1.1 shows a phylogenetic tree constructed by Woese et al. [1] that separates life into the three domains Bacteria, Archaea and Eukaryota. For an introduction to phylogenetic trees, see [2–4].

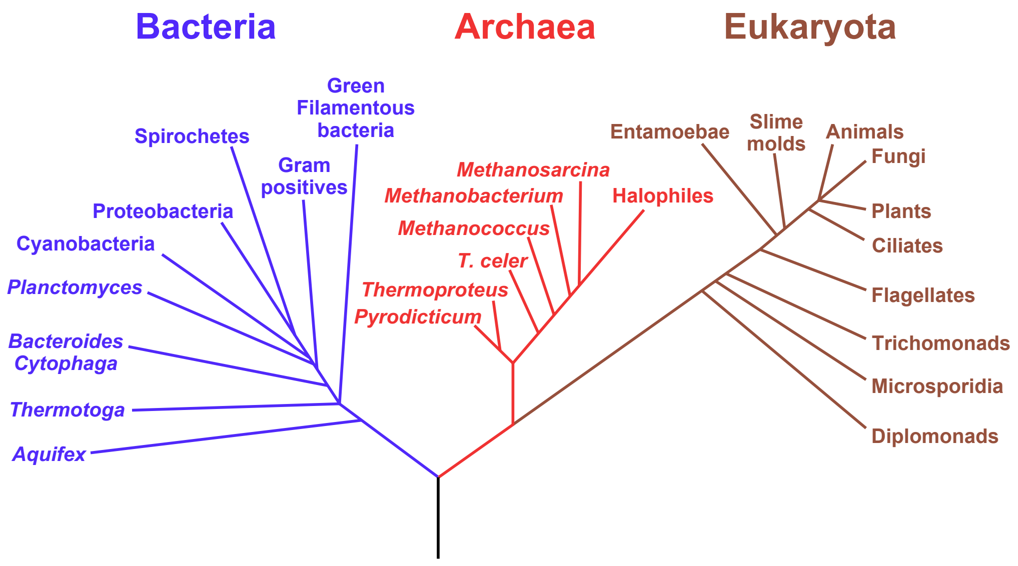


Figure 1.1: A phylogenetic tree built by Woese et al. [1] that separates life into the three domains Bacteria, Archaea and Eukaryota.

The most common methods of building phylogenetic trees are parsimony [5], maximum likelihood [6], and MCMC-based Bayesian inference [7]. All these methods depend upon a mathematical model describing the evolutionary relationships of the species involved. However, there is no way to measure whether a particular phylogenetic hypothesis is accurate or not, unless the true relationships among the taxa being examined are already known. This implies that there is no way to choose an optimal method to build a phylogenetic tree when different methods lead to different trees. Moreover, even good phylogenetic inference methods cannot guarantee that a constructed tree correctly represents their evolutionary history, since the ancestral history of species may be non-tree-like. This is caused by processes that include hybridization, lateral gene transfer (LGT), and recombination [8, 9]. Collectively, these processes are known as reticulation events. Thus, for each gene involved in reticulation events, phylogenetic methods create a different phylogenetic tree, representing the evolutionary history of that particular gene. Phylogenetic distance measures are used to quantify the dissimilarity of two phylogenetic trees, and thus capture how well the evolutionary hypotheses of two or more phylogenetic trees agree. They can often also be used to discover putative sequences of reticulation events that can explain the dissimilarity. To simultaneously display disagreeing tree topologies, phylogenetic networks (e.g., hybridization networks) can be used, which are a generalization of phylogenetic trees and allow species to inherit genetic material from more than one parent. An example of a phylogenetic network is shown in Figure 1.2.

A number of measures are commonly used to define the distance between phylogenies. Among them, the subtree prune-and-regraft (SPR) distance [10] and the hybridization number [11] have attracted particular attention due to their direct evolutionary interpretations. The SPR distance is the minimal number of SPR operations needed to transform one tree into the other. Each such operation detaches a subtree by cutting the parent edge of its root and then reattaches this subtree in a different

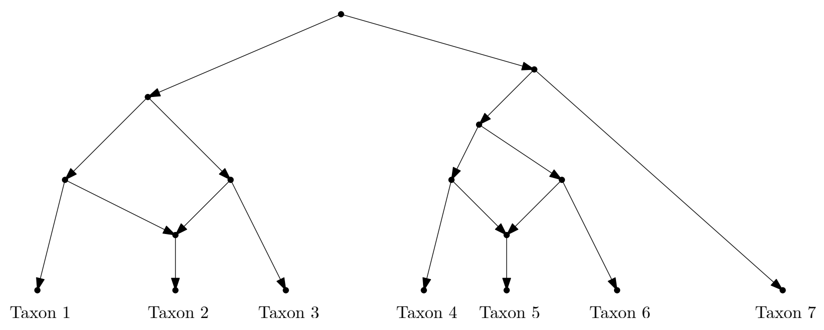


Figure 1.2: An imaginary phylogenetic network representing the history of 7 taxa.

location in the tree. This directly models the effect of lateral gene transfer because an LGT event between the endpoints of the SPR operation, in the opposite direction of the SPR operation, has the effect that, in the tree constructed from the gene involved in the LGT event, the moved subtree appears more similar to descendants of its new parent after the SPR operation than to its "real parent" in the initial tree. As such, the SPR distance models the minimum number of LGT events necessary to explain the difference between a gene tree and a reference tree. The SPR distance has also been used as an optimality criterion in the construction of supertrees [12, 13].

Hybrid networks represent the evolutionary history of a set of taxa under the assumption that the only type of reticulation events in their history is hybridization, which is common in plants. In this case, each gene common to the underlying set of extant species evolves in a tree-like fashion, while certain taxa may inherit their genes from different parents. Thus, while the history of the set of taxa is a network, the history of each gene is a tree and must be displayed by the network, that is, the tree must be constructible from the network by removing edges. For reconciling two different gene trees, this leads to the problem of finding a hybrid network that displays both trees and, following the parsimony principle, this network should contain the smallest possible number of hybrid nodes. The hybridization number of the two trees is this minimum number of hybrid nodes in any hybrid network displaying the two trees. Algorithms for computing the hybridization number can also construct the corresponding hybrid network [14] and thus are able to find a smallest set of possible hybridization events necessary to explain the differences between the two input trees [11]. Both SPR distance and hybridization number are NP-hard to compute [14–16].

To model these two distance measures, two appropriately defined types of agreement forests (AFs) are useful: maximum agreement forest (MAF) for SPR distance, and maximum acyclic agreement forest (MAAF) for hybridization number. An agreement forest of two phylogenies can be obtained from either tree by cutting an appropriate set of edges. Such a forest is called acyclic if the ancestry relationships of its components in the two trees do not form cycles. Their formal definitions will be introduced in Chapter 2. The number of components of an MAF is one more than the SPR distance [16], so the problem of computing the SPR distance of two trees is equivalent to the problem of constructing an MAF. Similarly, the number of components of an MAAF is one more than the hybridization number [11], so the problem of computing the hybridization number of two trees is equivalent to the problem of constructing an MAAF.

A number of previous results have concentrated on computing the SPR distance and hybridization number of two trees efficiently (see Section 1.2). Most of these algorithms are restricted to binary trees due to the inherent complexity of comparing non-binary trees.

Non-binary trees contain *multifurcations* (polytomies), which are vertices with three or more children. Often a distinction is made between soft and hard multifurcations. *Soft multifurcations* are the result of collapsing bipartitions with low statistical support: though the lineages diverged in binary phylogenies constructed using statistical methods (i.e., some descendants are closer relatives than others), the available data does not allow us to determine with confidence the exact order of these speciation events. *Hard multifurcations* on the other hand mean three or more lineages are created during a single speciation event, and the children are equally distant from each other [17]. Simultaneous speciation events are assumed to be rare, so the common assumption is that all multifurcations are soft. A binary phylogeny consistent with a given multifurcating one can be obtained by resolving each multifurcation into an arbitrary sequence of bipartitions. However, the inferred binary evolutionary relationships are not supported by the original data and may be incorrect. This may lead to the inference of spurious reticulation events from such binary phylogenies. Thus, since most phylogenies constructed using statistical inference methods include multifurcations, it is crucial to develop efficient algorithms to compare multifurcating trees using SPR distance and hybridization number directly. The computation of SPR distance and hybridization number is NP-hard even on binary trees. Their computation for non-binary trees is much harder because the number of possible resolutions grows exponentially with the out-degrees of multifurcating nodes.

1.1 Contribution

The contribution of this thesis is to develop a novel and fast fixed-parameter algorithm for computing the hybridization number k of two multifurcating phylogenetic trees, assuming that all multifurcations are soft. As previous algorithms for binary trees [18], we exploit its relationship with MAAFs to compute the hybridization number. Given an MAAF, a hybrid network with k hybridization events can be constructed quickly [14]. Similar to [18], we find an MAAF in two phases. We first use a 4-way branching algorithm to find a set of agreement forests with at most $k + 1$ components, and then do cut additional edges to break cycles in the ancestry relationships of the components of each forest. A first simple version of this algorithm has a running time of $O(16^k n)$, where k is the hybridization number and n is the number of taxa. A second improved version is obtained by applying a number of incremental improvements to reduce the running time to $O(4.83^k n)$. To the best of our knowledge, there are only two existing FPT algorithms for multifurcating hybridization number, one by van Iersel et al. [19]

with a complexity of $O(2^n \cdot poly(n))$, and the other by Piovesan and Kelk [20] with a complexity of $O((6^k k!) \cdot poly(n))$. The algorithm in [19] itself is not an FPT algorithm, but it can be combined with known kernelization rules [14] to obtain an algorithm with running time $O(2^{O(k)} + poly(n)) = O(c^k + poly(n))$. The constant c is large. In comparison, our solution with a complexity of $O(4.83^k \cdot poly(n))$ is indeed a more efficient FPT algorithm to compute the hybridization number of two multifurcating trees.

The work in this thesis is based on previous work on multifurcating SPR distance [21] and binary hybridization number [18], but it is not a simple combination of both algorithms. Substantial novel insights are necessary to make this combination work for multifurcating trees.

- The 4-way branching technique from the multifurcating SPR algorithm [21] gives us binary agreement forests that may or may not be the right starting point for computing an MAAF of the two input trees. We developed an efficient algorithm to compute minimally resolved multifurcating agreement forests that can be refined to an MAAF of the two input trees.
- In order to determine which edges to cut to break cycles in the AFs computed by the branching phase, Whidden et al. [18] introduced the notion of an expanded cycle graph. Their definition depends on the trees being binary, and we had to extend this definition to multifurcating trees in a way that captures the existence or non-extistence of cycles in an appropriate binary resolution.
- The refinement phase identifies certain nodes, called exit nodes, in the agreement forest found in the branching phase so that an MAAF can be found by cutting all edges on the paths from a subset of these nodes to the roots of their components. To make the refinement phase faster, Whidden et al. [18] introduced sophisticated tagging rules in the branching phase that allows them to cut the number of these candidates in half. We applied the same idea, but multifurcations forced us to

completely redesign these tagging rules.

- The correctness proof of the improved refinement phase in Whidden et al.'s algorithm [18] shows that there exists a “canonical” AF found in the branching phase so that the exit nodes whose ancestor edges need to be cut to obtain an MAAF are in fact tagged in this forest. This proof becomes substantially more technical in the case of multifurcating trees, and can easily be considered the main technical contribution of this thesis.
- We present the first implementation of an FPT algorithm for multifurcating hybridization number. Our experiments show its practicality for moderate hybridization numbers. On a 2.4GHz AMD Opteron system, our algorithm can compute hybridization numbers as large as 17 of trees with 74 taxa in 1 hour.

1.2 Related Work

Bordewich and Semple proved that the SPR distance is NP-hard to compute [16]. They also proved that the hybridization number is NP-hard to compute [14]. Thus, it is unlikely that either SPR distance or hybridization number can be computed exactly by an algorithm without exponential running time. Even so, due to their biological significance, much effort has been made to develop efficient algorithms to compute these distances. There are two standard algorithmic approaches for solving NP-hard problems: approximation algorithms and fixed-parameter algorithms. Before reviewing the previous work further, it is necessary to illustrate the close relationship between SPR distance and hybridization number.

While the SPR distance and the hybridization number can both be used as measures of dissimilarity between phylogenies, the SPR distance is also a lower bound on the hybridization number. It has been shown that the SPR distance can be obtained from an MAF [16] and that hybridization number can be obtained from an MAAF [11]. An MAAF is a more constraint agreement forest than an MAF. Thus, the algorithms to construct MAFs are very valuable as starting points for algorithms to construct MAAFs.

1.2.1 Approximation Algorithms

Approximation algorithms are often used to find approximate solutions to NP-hard problems. Since it is unlikely that there can ever be efficient polynomial-time exact algorithms solving NP-hard problems, we need to settle for approximate answers if we insist on polynomial running time. Unlike heuristics, approximation algorithms provide a guaranteed bound on the ratio between the optimal solution and the one they compute, which is called the *approximation ratio*. An introduction to approximation algorithms can be found in [22].

For the SPR distance problem, Hein et al. [23] first claimed a 3-approximation algorithm for computing SPR distances, but Rodrigues et al. [24] provided a counter example that shows that the claimed 3-approximation ratio is incorrect. Both the algorithms of [23] and [24] compute 5-approximations of the SPR distance between two rooted binary phylogenies, and the algorithms can be implemented in linear time. Later, the approximation ratio was improved to 3 by Bordewich et al. [25] using an algorithm with $O(n^5)$ running time, and by Rodrigues et al. [24] using an algorithm with $O(n^2)$ running time. Whidden and Zeh [26] improved the running time of this algorithm to linear. In the non-binary (multifurcating) case, a linear time 4-approximation algorithm was proposed by van Iersel et al. [27] and Whidden et al. [21] independently, but [21] also proposed an $O(n \log n)$ time 3-approximation algorithm for rooted non-binary MAF.

The hybridization number problem was proved to be APX-hard by Bordewich and Semple [14], which means it does not have a polynomial-time approximation scheme unless P=NP. Kelk et al. [28] made the first step towards the approximation of hybridization number for two binary phylogenies. They proved that this problem has a constant factor polynomial-time approximation if and only if the problem of computing a minimum-size feedback vertex set in a directed graph (DFVS) has a constant factor polynomial-time approximation. Whether DFVS has a polynomial-time constant factor approximation algorithm is a long standing open problem. Kelk et al. also provided an $O(\log k \log\log k)$-approximation algorithm for hybridization number, which is obtained using an existing approximation algorithm for DFVS and is the first non-trivial polynomial-time approximation algorithm for hybridization number. Van Iersel et al. [29] presented CycleKiller, an exponential-time 2-approximation (or 4-approximation in its fastest mode) algorithm. By implementing this algorithm, they claimed that their actual approximation ratio was often close to 1. In the non-binary case, van Iersel et al. [30] showed that a c-approximation algorithm for non-binary MAF and a d-approximation algorithm for the DFVS problem can be combined to yield a $d(c+3)$-approximation for non-binary MAAF, which is also a $d(c+3)$-approximation for hybridization number.

1.2.2 Fixed-parameter Algorithms

The main idea of fixed-parameter algorithms is to restrict the exponential growth of the running time to some parameter that is specific to the problem but independent of the input size. As these algorithms provide exact solutions, they are more appealing than approximate methods. For an introduction to fixed-parameter tractability (FPT), see [31].

For the SPR distance problem, the previously best algorithm for the rooted SPR distance of two binary phylogenies by Whidden and Zeh [13] runs in $O(2^k n)$ time. An earlier version runs in $O(2.42^k n)$ time [18]. Both algorithms are based on a 3-way branching algorithm [26] that runs in $O(3^k n)$ time. The improvements are achieved using more detailed branching rules [18] and a novel edge protection scheme [13], respectively. Before these, Bordewich et al. [25] gave an algorithm that runs in $O(4^k \cdot k^4 + n^3)$. Van Iersel et al. [27] expanded the algorithm of Whidden and Zeh [26] to compute the SPR distance of two non-binary phylogenies in $O(4^k n)$ time. At the same time, a preprint by Whidden and Zeh [21] proposed the same extension independently, and introduced improved branch rules to obtain a running time of $O(2.42^k n)$. Recently, Shi et al. [32] introduced algorithms to construct maximum agreement forest on multiple binary trees for both rooted and unrooted trees. Their algorithms can solve the rooted-MAF problem in $O(3^k n)$ time, and solve the unrooted-MAF problem in $O(4^k n)$ time. Note that n here refers to the number of vertices in all the input trees, which is different from the common definition as the number of species.

For the hybridization number problem, the hybridization number k is a natural parameter to choose. In most biological datasets, the number of reticulation events is relatively small, and thus k is much smaller than n, the number of taxa. The first fixed-parameter algorithm for the hybridization number of two binary trees was given by Bordewich and Semple [14] in 2007, with running time $O((28k)^k + n^3)$. Kelk et al. [28] provided an improved analysis of the kernel size for hybridization number, which reduces the running time of the algorithm by Bordewich and Semple to $O((18k)^k + n^3)$. Without kernelization, Whidden and Zeh [26] proposed a branching algorithm for binary hybridization number with running time $O(3^k n \log n)$, but it assumed that all cycles have length 2. Combining the $O(3^k n)$ branching technique for agreement forests from Whidden and Zeh [26] with an exhaustive search, Chen and Wang [33] proposed an algorithm for computing all MAAFs of two binary phylogenies. Their algorithm was implemented as a program named HybridNet that can construct optimal hybridization networks rapidly. Similarly, Scornavacca and Albrecht [34] combined the $O(3^k n)$ branching algorithm with reductions, and implemented a parallel solution [35] of computing hybridization networks for two binary trees. Collins et al. [36] improved the kernelization approach to $O((14k)^k \cdot n^3)$ using repeated kernelization steps (called "interleaving") that are applied throughout the exhaustive search. So far, the best theoretical result for two rooted binary phylogenetic trees is due to Whidden and Zeh [18], with a running time of $O(3.18^k n)$ or $O(3.18^k k + n^3)$ with kernelization. Algorithms for more than two trees have also been proposed recently [37–39]. In the non-binary (multifurcating) case, several fixed-parameter algorithms for hybridization number have been developed recently [19, 20, 40–42]. However, most of them are of only theoretical value in that they show hybridization number is in FPT. Only two of these solutions exist can be considered practical. Linz and Semple [40] used two reductions to prove that the hybridization number of two (not necessarily binary) trees is in FPT. Kelk et al. [41] proved that non-binary hybridization for multiple trees is in FPT assuming certain conditions hold. Van Iersel and Kelk [42] discussed two different kernelization techniques for multiple non-binary trees. The two practical solutions are an $O(2^n \cdot poly(n))$ time algorithm [19] and an $O((6^k k!) \cdot poly(n))$ time algorithm [20]. The $O(2^n \cdot poly(n))$ algorithm makes use of ST-sets (subsets of the set of taxa that are compatible with all clusters in a phylogeny) and dynamic programming, while the $O((6^k k!) \cdot poly(n))$ algorithm uses a different approach based on analyzing the structure of "terminals", which are maximal elements of a natural partial order on species set.

Table 1.1: The previous best algorithms for hybridization number. (Note: approximation ratio is used to compare approximation algorithms with polynomial running time, while time complexity is used to compare FPT algorithms.)

	Approximation algorithm	**FPT algorithm**
Binary case	4 [29]	$O(3.18^k n)$ [18]
Non-binary case	$O(\log k \log\log k)$ [30]	$O((6^k k!) \cdot poly(n))$ [20]

In summary, there are three common techniques for these FPT algorithms: branching, kernelization, and reduction. The branching technique (or bounded search tree algorithm) was used to compute MAFs [13,18,21,25–27,32] and MAAFs [13,18,33,34]. It was also combined with the kernelization technique [18] and reduction [34,35]. The kernelization technique was first proposed by Bordewich and Semple [16] for the SPR distance of two rooted binary trees, with a kernel size $28k$. Later, they also proposed a size $14k$ kernel for the hybridization number of two rooted binary trees [14]. Then this technique was used in [18, 28, 36, 39, 42]. The reduction technique first emerged with kernelization for SPR distance [15] and for hybridization number [14]. Bordewich et al. [43] introduced a pure reduction algorithm for hybridization number and provided the first experimental results. Compared with branching and kernelization, reduction is less powerful because it doesn't give an exact time bound, but it has been proven extremely powerful in practice for certain inputs because it divides the original inputs into several smaller inputs, which decreases the parameter size and the exponential complexity. In addition, different from these common techniques, there is a different approach to compute the exact SPR distance and hybridization number, which is integer linear programming [44, 45]. However, computing the hybridization number between two large and topologically far apart trees is still challenging.

Above we have discussed theoretical solutions to computing the SPR distance and hybridization number of two trees. Unfortunately, not each of them has an existing implementation. For the SPR distance problem, there are implementations for both binary and non-binary trees. The best three results for binary trees are due to Wu [44] using integer linear programming, Bonet and John [46] using satisfiability testing, and Whidden et al. [18] using a branching algorithm and reduction. Recently, van Iersel [47] provided a Java implementation to compute MAFs for non-binary trees, which can also be used to compute the SPR distance of non-binary trees. For hybridization number, experimental results are available only for binary trees to date. As the first program that can compute the binary hybridization number, *HybridNET* [43] was developed to construct optimal hybridization networks using Whidden's branching technique [48]. Later, some faster implementations followed using different techniques. *SPRDist* [45] used the GNU GLPK integer linear programming solver, or commercial CPLEX solver, to compute hybridization number quickly. Collins et al. [36] combined reduction and repeated kernelization. Albrecht et al. [35] applied parallelism and used a combination of reduction techniques and exhaustive search. These three implementations didn't vary much in performance and they were evaluated on different machines.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2 presents the necessary terminology and notation. Chapter 3 reviews the 4-way branching algorithm that is used to compute MAFs [21] and some structural results that form the theoretical basis for the branching phase of our hybridization algorithm for two multifurcating trees. Chapter 4 introduces an efficient algorithm to restore the multifurcation information lost in the branching phase. This algorithm is crucial to connect the branching phase and the refinement phase to form the final algorithm. Its linear time complexity is also essential to keep the final algorithm efficient. Chapter 5 discusses how to transplant the expanded cycle graph [18] from binary trees to multifurcating trees and describes a first complete fixed-parameter algorithm for multifurcating hybridization number. Chapter 6 develops an improved algorithm by adding tagging rules to the branching phase, and reducing the size of the search space of the refinement phase using these tags. Chapter 7 discusses the implementation of our algorithm and experimental results. Chapter 8 offers concluding remarks and discusses avenues for future research.

Chapter 2

Preliminaries

This chapter introduces terminology and notations used throughout this thesis. For the sake of consistency with previous work, we mostly use the definitions and notation from [16, 18, 21, 25].

2.1 Phylogenetic Trees

A (rooted phylogenetic) *X-tree* is a rooted tree T whose leaves are the elements of a label set X and whose non-root internal nodes have at least two children each. The root of T has label ρ and has only one child. We use n to denote the number of leaves, that is, the size of the label set X. T is *binary* (or *bifurcating*) if all internal nodes have exactly two children each, otherwise it is *multifurcating*. For a subset V of X, $T(V)$ is the smallest subtree of T that connects all nodes in V; see Figure 2.1(b). $T|V$ is the smallest tree that can be obtained from $T(V)$ by suppressing degree-2 internal nodes; see Figure 2.1(c). *Suppressing* a degree-2 node v deletes v and its incident edges, and then reconnects its parent u and child w using a new edge (u, w).

For two binary rooted X-trees T_1 and T_2, the *hybridization number* is defined in terms of hybrid networks of the two trees. A *hybrid network* of T_1 and T_2 is a directed acyclic graph H with a single source ρ which has in-degree 0 and out-degree 1, whose leaves (nodes with out-degree 0) are labelled bijectively with the labels in $X \backslash \{\rho\}$, and which displays both T_1 and T_2. In other words, both T_1 and T_2, with their edges directed away from the root, can be obtained from H by deleting and contracting edges. An example of such a hybrid network is shown in Figure 2.1(e).

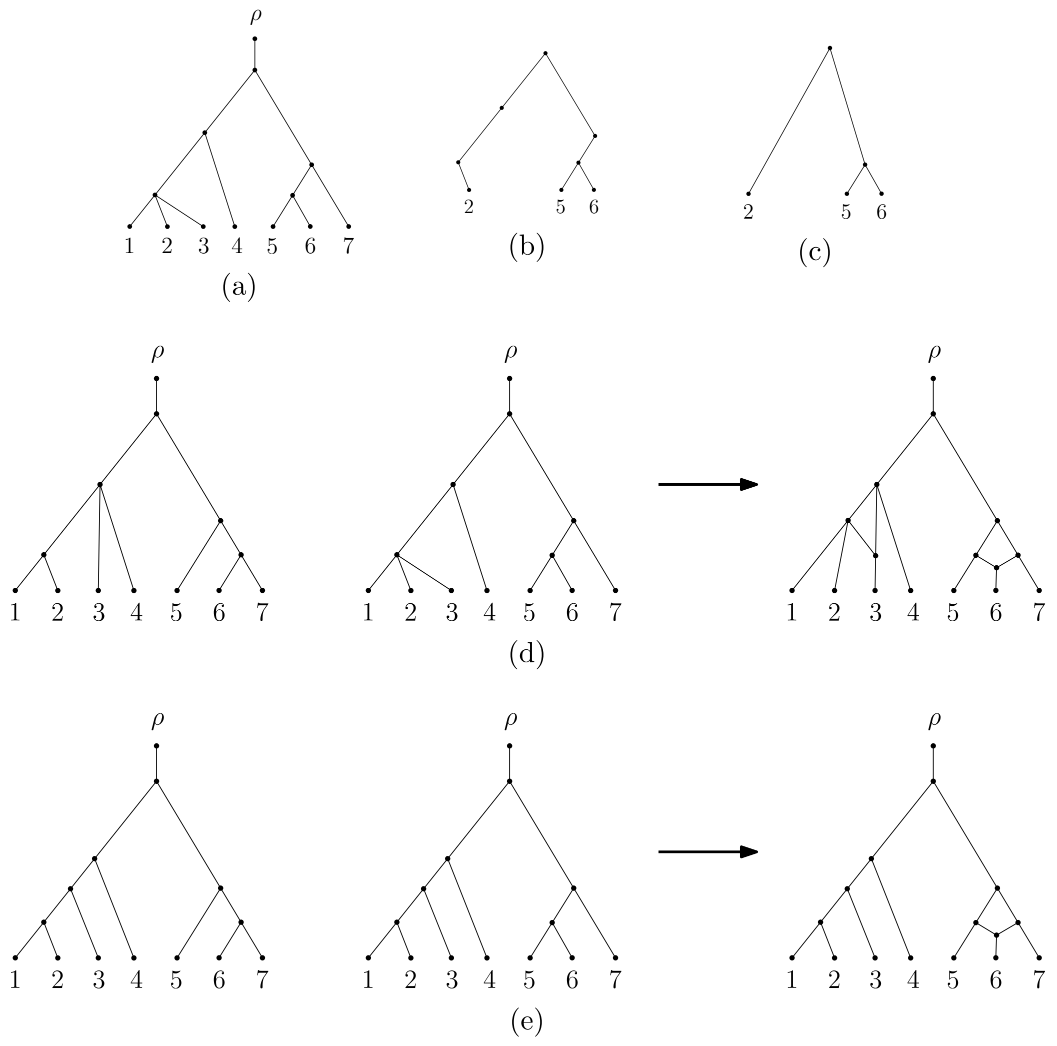


Figure 2.1: (a) An X-tree T. (b) The subtree $T(V)$ for $V = \{2, 5, 6\}$. (c) $T|V$. (d) A hybrid network of two multifurcating X-trees. (e) A hybrid network of two binary resolutions of the trees in (d).

The nodes of H with in-degree 2 are called *hybrid nodes*. (In a hybrid network of k trees, every node has at most k parent edges or the network contains useless edges that can be deleted without changing the property that the network displays the k trees.) The *hybridization number* of two X-trees T_1 and T_2 is the minimum number of hybrid nodes among all hybrid networks of the two trees. Since hybrid nodes represent hybridization events in the history of the set of taxa in X, minimizing the number of such nodes is equivalent to seeking a most parsimonious explanation of the differences between the two input trees.

Before defining the hybridization number of two multifurcating X-trees, the significance of multifurcations should be explained. In phylogenetic trees, multifurcations are vertices with more than two children. A multifurcation is *hard* if it indeed represents a common ancestor that has more than two species as direct descendants; it is *soft* if it represents a series of binary speciation events but the order of these events cannot be determined with sufficient confidence [17]. Since simultaneous speciation events are assumed to be rare, a common assumption is that all multifurcations are soft. Similar to the SPR distance of two multifurcating trees [21], hard and soft multifurcations lead to two different definitions of the hybridization number of two multifurcating trees. The *hard hybridization number* of two multifurcating X-trees T_1 and T_2 is defined as the minimum number of hybrid nodes among all hybrid networks from which T_1 and T_2 can be obtained by deleting edges and suppressing degree-2 nodes. These hybrid networks display both trees, including their multifurcations. The *soft hybridization number* of two multifurcating X-trees T_1 and T_2 is defined as the minimum hybridization number of all pairs of binary resolutions of the two trees. A *resolution* of a multifurcating tree T is a tree T' from which T can be obtained by contracting edges. T' is a *binary resolution* if it is a binary tree of two multifurcating trees simply. An example that hard hybridization number and soft hybridization number are different is shown in Figures 2.1(d) and 2.1(e). Based on the assumption that all multifurcations are soft, we focus on the soft hybridization number in this thesis, and refer to the soft hybridization number of two multifurcating trees as their hybridization number. We use $hyb(T_1, T_2)$ to denote this number.

2.2 Hybridization Number and Agreement Forests

The hybridization number is related to the size of appropriately defined agreement forests. For a forest F whose components are rooted phylogenetic trees $T_1, T_2, ..., T_k$ with label sets $X_1, X_2, ..., X_k$, we say F *yields* the forest with components $T_1|X_1$, $T_2|X_2$, ..., $T_k|X_k$. For a subset E of edges of F, let $F - E$ be the forest obtained by deleting the edges in E from F, and let $F \div E$ be the forest yielded by $F - E$. We say $F \div E$ is a *forest* of F.

F is an *agreement forest* (AF) of two forests F_1 and F_2, if F is a forest of both a binary resolution of F_1 and a binary resolution of F_2. If there is no AF of F_1 and F_2 with fewer components than F, then F is a *maximum agreement forest* (MAF) of F_1 and F_2. We use $m(F_1, F_2)$ to denote the number of components in an MAF of F_1 and F_2. For a forest F of F_2, we use $e(F_1, F_2, F)$ to denote the size of the smallest edge set E such that $F \div E$ is an AF of F_1 and F_2. For two rooted X-trees T_1 and T_2, we use $d_{SPR}(T_1, T_2)$ to denote the SPR distance between T_1 and T_2. Bordewich and Semple [16] showed that $d_{SPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$ for two binary rooted X-trees T_1 and T_2. Moreover, the equation also holds for two multifurcating rooted X-trees and their soft SPR distance [21]. This is true because $d_{SPR}(T_1, T_2)$, $e(T_1, T_2, T_2)$, and $m(T_1, T_2)$ are taken as the minimum over all binary resolutions of T_1 and T_2.

The hybridization number of two binary rooted X-trees T_1 and T_2 corresponds to a more constrained type of AF of T_1 and T_2. To define this AF, we first need to introduce the concept of cycle graph. For an AF $F = \{C_\rho, C_1, C_2, ..., C_k\}$ of T_1 and T_2, the *cycle graph* G_F of F has one node per component of F, and there is a directed edge from node C_i to node C_j if component C_i is an ancestor of component C_j in one of the two trees. An example of cycle graph is shown in Figure 2.2(d). Formally, for every node $x \in F$ and $i \in \{1, 2\}$, we define $\phi_i(x)$ to be the *lowest common ancestor* (LCA)

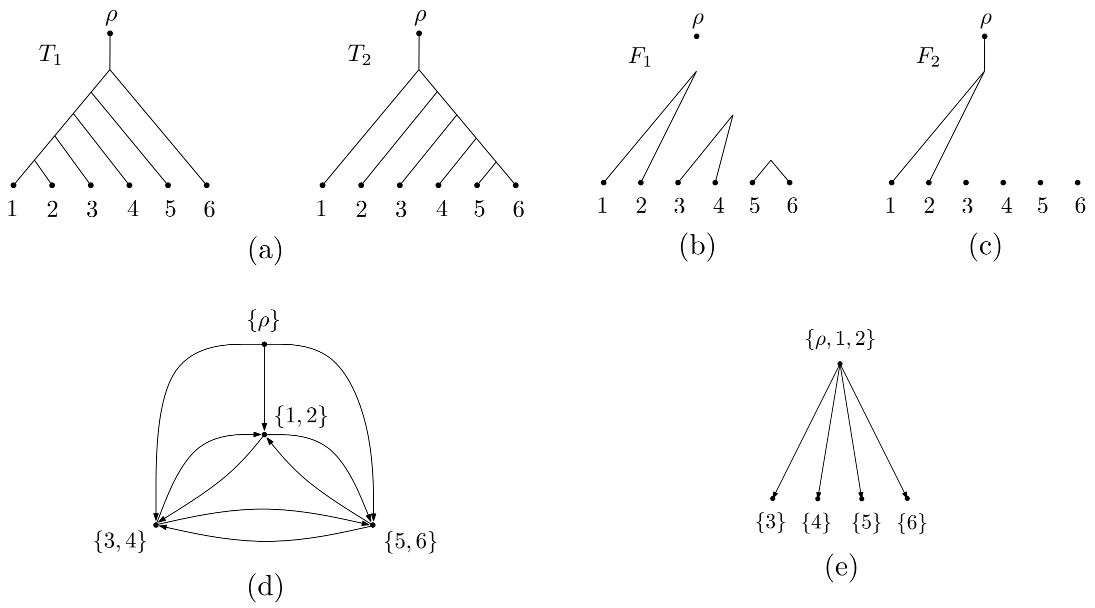
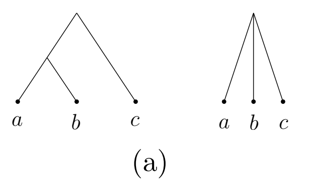


Figure 2.2: (a) Two binary rooted X-tree T_1 and T_2. (b) A cyclic AF F_1 of T_1 and T_2. (c) An acyclic AF F_2 of T_1 and T_2. (d) The cycle graph G_{F_1} of F_1. (e) The cycle graph G_{F_2} of F_2.

in T_i of all labelled leaves that are descendants of x in F. Based on this mapping, G_F contains a directed edge (C_i, C_j) if and only if either $\phi_1(r_i)$ is an ancestor of $\phi_1(r_j)$ or $\phi_2(r_i)$ is an ancestor of $\phi_2(r_j)$, where r_i is the root of component C_i and r_j is the root of component C_j. We say F is *cyclic* if G_F contains at least one cycle. Otherwise, F is an *acyclic* agreement forest (AAF) of T_1 and T_2. In Figure 2.2, F_1 is cyclic and F_2 is acyclic. If there is no AAF of F_1 and F_2 with fewer components than F, then F is a *maximum acyclic agreement forest* (MAAF) of F_1 and F_2. We use $\tilde{m}(F_1, F_2)$ to denote the number of components in an MAAF of F_1 and F_2, and $\tilde{e}(F_1, F_2, F)$ to denote the size of the smallest edge set E such that $F \div E$ is an AAF of F_1 and F_2. Baroni et al. [11] showed that $hyb(T_1, T_2) = \tilde{e}(T_1, T_2, T_2) = \tilde{m}(T_1, T_2) - 1$ for two binary rooted X-trees T_1 and T_2. Again, the equation also holds for two multifurcating rooted X-trees T_1 and T_2 and the soft hybridization number, because $hyb(T_1, T_2)$, $\tilde{e}(T_1, T_2, T_2)$, and $\tilde{m}(T_1, T_2) - 1$ are taken as the minimum over all binary resolutions of T_1 and T_2. Thus, to determine the hybridization number of two multifurcating rooted X-trees, we need to compute a binary MAAF of the two trees.

$\sim_F$ is a relation derived on the vertices of F where $a \sim_F b$ if there exists a path from a to b. And let $F(a, b)$ be this path from a to b. The nodes of F that are not in $F(a, b)$ and whose parents are in $F(a, b)$ are called *pendant nodes* of this path. Their parent edges are *pendant edges* of $F(a, b)$. For a node x of F, F^x denotes the subtree of F induced by all descendants of x, including x itself. For two rooted forests F_1 and F_2 and a node $a \in F_1$, we say that a exists in F_2 if there exists a node a' in F_2 such that $F_1^a = F_2^{a'}$. For simplicity, we refer to both a and a' as a. For forests F_1 and F_2 and nodes $a, c \in F_1$ with a common parent in F_1, we say $\{a, c\}$ is a *sibling pair* of F_1 if a and c exist in F_2. We say $\{a_1, a_2, ..., a_m\}$ is a *sibling group* if $\{a_i, a_j\}$ is a sibling pair of F_1, for all $1 \leq i < j \leq m$, and a_1 has no sibling not in the group.

A *triplet* $ab|c$ of a rooted forest F is defined by three leaves a, b, and c in the same component of F and such that the path from a to b in F is disjoint from the path from c to the root of the component. Multifurcating trees also allow triplets $a|b|c$ where a, b, and c share the same lowest common ancestor. Examples of triplets are shown in Figure 2.3(a). A triplet $ab|c$ of a forest F_1 is *compatible* (or *consistent*) with a forest F_2 if it is also a triplet of F_2 or F_2 contains the triplet $a|b|c$; otherwise it is *incompatible* (or *inconsistent*) with F_2. We say a forest F_1 is incompatible (or inconsistent) with another forest F_2 if there is at least one triplet of F_1 incompatible with F_2; otherwise F_1 is compatible (or consistent) with F_2 (but F_2 may be incompatible with F_1).



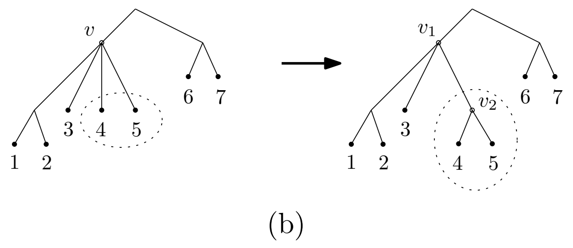


Figure 2.3: (a) Two triplets $ab|c$ and $a|b|c$. (b) Expansion of a subset of v's children.

2.3 Derived Properties

The correctness of our algorithms in the next chapters relies on the following two lemmas and two observations. Lemma 1 was shown by Bordewich et al. [25]. Given a forest F and a set of edges E, if there is an edge e of F such that $F - (E \cup \{e\})$ has a component without labelled nodes, it shows that we can replace some edge f in E by e. In other words, the forest $F \div (E \setminus \{f\} \cup \{e\})$ is the same as $F \div E$.

Lemma 1 (Shifting Lemma). *Let F be a forest of an X-tree, e and f edges of F, and E a subset of edges of F such that $f \in E$ and $e \notin E$. Let v_f be the end vertex of f closest to e, and v_e an end vertex of e. If (1) $v_f \sim_{F-E} v_e$ and (2) $x \not\sim_{F-(E\cup\{e\})} v_f$, for all $x \in X$, then $F \div E = F \div (E \backslash \{f\} \cup \{e\})$.*

The following lemma specifies when an expansion does not change the hybridization number. An *expansion* splits a multifurcating node v into two nodes v_1 and v_2, such that v_1 is the parent of v_2, and the children of v are divided into two subsets that become the children of v_1 and v_2, respectively. For brevity, we refer to this operation as expanding the subset of v's children that become v_2's children; an example is shown in Figure 2.3(b).

Lemma 2 (Expansion Lemma). *Given two rooted X-trees T_1 and T_2, let F_2 be a forest of T_2, and let $F \div E$ be an MAAF of T_1 and T_2, where F is a binary resolution of F_2. Let $a_1, a_2, ..., a_p, a_{p+1}, ..., a_m$ be the children of a node in F_2 and let F_2' be the result of expanding $\{a_{p+1}, a_{p+2}, ..., a_m\}$ in F_2. If $a_i' \not\sim_{F\div E} a_j'$, for all $1 \le i \le p < j \le m$ and all leaves $a_i' \in F_2^{a_i}$ and $a_j' \in F_2^{a_j}$, then $\tilde{e}(T_1, T_2, F_2) = \tilde{e}(T_1, T_2, F_2')$.*

Proof. We prove that $\tilde{e}(T_1, T_2, F_2) \le \tilde{e}(T_1, T_2, F_2')$ and $\tilde{e}(T_1, T_2, F_2) \ge \tilde{e}(T_1, T_2, F_2')$ separately.

The only difference between F_2 and F_2' is the expansion of $\{a_{p+1}, a_{p+2}, ..., a_m\}$. Thus, the resolutions of F_2' are a subset of the resolutions of F_2, and $\tilde{e}(T_1, T_2, F_2) \le \tilde{e}(T_1, T_2, F_2')$.

To prove that $\tilde{e}(T_1, T_2, F_2) \geq \tilde{e}(T_1, T_2, F_2')$, it suffices to prove that $F \div E$ is a forest of F_2' because it is an MAAF of T_1 and T_2. Assume the contrary. By Observation 2, at least one component of $F \div E$ is inconsistent with F_2', while it is consistent with F_2 because $F \div E$ is a forest of F_2. This component must contain leaves $a_i' \in F_2^{a_i}$ and $a_j' \in F_2^{a_j}$ for some $1 \leq i \leq p < j \leq m$ because the expansion is the only difference between F_2 and F_2'. However, this is a contradiction because $a_i' \nsim_{F \div E} a_j'$. □

The following observation states that agreement forests cannot contain incompatible triplets.

Observation 1. *[18] Let F_1 and F_2 be forests of rooted X-trees T_1 and T_2, and let F be an agreement forest of F_1 and F_2. If $ab|c$ is a triplet of F_1 incompatible with F_2, then $a \nsim_F b$ or $a \nsim_F c$.*

For two forests F_1 and F_2 with the same label set, two components C_1 and C_2 of F_2 with label sets X_1 and X_2, respectively, are said to *overlap* in F_1 if $F_1|X_1$ and $F_1|X_2$ are not edge-disjoint. In particular, there exist leaves $u, v \in X_1$ and $x, y \in X_2$ such that the two paths $u \sim_{F_1} v$ and $x \sim_{F_1} y$ share an edge. The following observation characterizes when one forest is a forest of another forest.

Observation 2. *[21] Let F_1 and F_2 be binary resolutions of forests of rooted X-trees T_1 and T_2, and denote the label sets of the components of F_1 by $X_1, X_2, ..., X_k$ and the label sets of the components of F_2 by $Y_1, Y_2, ..., Y_l$. F_2 is a forest of F_1 if and only if (1) for every Y_j, there exists an X_i such that $Y_j \subseteq X_i$, (2) no two components of F_2 overlap in F_1, and (3) no triplet of F_2 is incompatible with F_1.*

Chapter 3

A 4-Way Branching Algorithm for SPR Distance

Since the hybridization number of two multifurcating trees is one less than the size of an MAAF of the two trees, we compute the hybridization number by computing an MAAF. As in the binary case [18], we do this in two phases: First we compute an AF that can be refined to an MAAF by cutting additional edges. Then we identify this set of additional edges that need to be cut. For finding an AF that can be refined to an MAAF, we use a simple 4-way branching algorithm from [21, 27]. We review this algorithm in this chapter for completeness and because we need to show that the edges cut by the algorithm are part of an edge set E such that $T_2 \div E$ is an MAAF of T_1 and T_2, that is, the forest produced by the algorithm can be refined to an MAAF of T_1 and T_2 by cutting the remaining edges in E.

Starting with the two input trees T_1 and T_2, the algorithm cuts edges and resolves multifurcations in both trees until the two resulting forests are identical. The intermediate state is that T_1 and T_2 have been partially resolved and reduced to forests F_1 and F_2, respectively. F_1 consists of a tree $\dot{T}_1$ and a forest F_0 while F_2 consists of two sets of components $\dot{F}_2$ and F_0. F_0 is the part of F_1 that agrees with F_2 and will be part of the final agreement forest. $\dot{T}_1$ and $\dot{F}_2$ may not agree but share the same label set. The key in each iteration is deciding which edges in $\dot{F}_2$ to cut next or which nodes to expand, in order to make progress towards an MAAF of T_1 and T_2. The results in this chapter identify a set of 4 edges such that cutting one of them makes progress towards an agreement forest that can be refined to an MAAF. Some of these edges are introduced by expanding nodes.

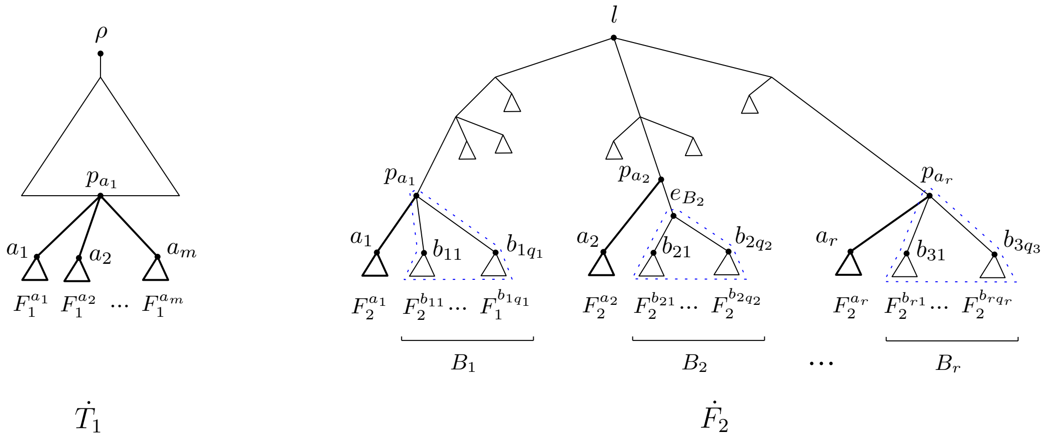

ρ
p_{a_1}
a_1 a_2 a_m
$F_1^{a_1}$ $F_1^{a_2}$ ... $F_1^{a_m}$
$\dot{T}_1$
l
p_{a_1} p_{a_2} p_{a_r}
e_{B_2}
a_1 b_{11} b_{1q_1} a_2 b_{21} b_{2q_2} a_r b_{31} b_{3q_3}
$F_2^{a_1}$ $F_2^{b_{11}}$... $F_1^{b_{1q_1}}$ $F_2^{a_2}$ $F_2^{b_{21}}$... $F_2^{b_{2q_2}}$ $F_2^{a_r}$ $F_2^{b_{r1}}$... $F_2^{b_{rq_r}}$
B_1 B_2 ... B_r
$\dot{F}_2$

Figure 3.1: A sibling group $\{a_1, a_2, ..., a_m\}$ in $\dot{T}_1$ such that $a_1, a_2, ..., a_r$ share a minimal LCA l in $\dot{F}_2$; $\dot{F}_2$ shows the result of expanding B_2.

3.1 Structural Results of Multifurcating Agreement Forests

Let $\{a_1, a_2, ..., a_m\}$ be a sibling group of $\dot{T}_1$. Before identifying an edge set to cut, the branching algorithm ensures a_i and a_j are not siblings in F_2, for all $1 \le i < j \le m$, and $F_2^{a_i}$ is not a component of F_2, for all $1 \le i \le m$. It does this using the following two transformations: If there exist indices $i \ne j$ such that a_i and a_j are also siblings in F_2, we expand this sibling pair $\{a_i, a_j\}$ and replace a_i and a_j with the resolving parent node (a_i, a_j) in the sibling group. If there exists an index i such that $F_2^{a_i}$ is a component of F_2, we cut a_i's parent edge in F_1, thereby removing a_i from the sibling group. It is easy to prove that these two transformations do not alter the number of edges that need to be cut in F_2 to obtain an MAAF of F_1 and F_2. Now, let $B_i = \{b_{i1}, b_{i2}, ..., b_{iq_i}\}$ be the set of siblings of a_i in F_2, for $1 \le i \le m$. We use e_x to denote the edge between a node x and its parent p_x, and e_{B_i} to denote the edge introduced by expanding B_i. For simplicity, we also use B_i to denote the parent node of $b_{i1}, b_{i2}, ..., b_{iq_i}$ introduced by expanding B_i. These definitions are illustrated in Figure 3.1. For a subset $\{a_{i_1}, a_{i_2}, ..., a_{i_r}\}$ of a sibling group $\{a_1, a_2, ..., a_m\}$, we say $a_{i_1}, a_{i_2}, ..., a_{i_r}$ share their lowest common ancestor l if $l = LCA_{F_2}(a_i, a_j)$ for all $i, j \in \{i_1, i_2, ..., i_r\}$, $i \ne j$. If, in addition, $LCA_{F_2}(a_i, a_j)$ is not a proper descendant of l for any $1 \leq i < j \leq m$, we say that $a_{i_1}, a_{i_2}, ..., a_{i_r}$ share a *minimal LCA* l; see Figure 3.1. For simplicity, we assume $\{a_{i_1}, a_{i_2}, ..., a_{i_r}\}$ is a maximal subset of $\{a_1, a_2, ..., a_m\}$ such that $\{a_{i_1}, a_{i_2}, ..., a_{i_r}\}$ share a minimal LCA and we orders these siblings in $\{a_1, a_2, ..., a_m\}$ so that $\{a_{i_1}, a_{i_2}, ..., a_{i_r}\} = \{a_1, a_2, ..., a_r\}$, $a_1, a_2, ..., a_{r-1}$ are not children of l and a_r may be child of l.

Our very first theorem shows that cutting at least one of the edges e_{a_1}, e_{a_2}, e_{B_1} and e_{B_2} makes progress towards an MAAF.

Theorem 1 (4-way Branch)**.** *Let F_1 and F_2 be forests of rooted X-trees T_1 and T_2, respectively. Suppose F_1 consists of a tree $\dot{T}_1$ and a set of components that also exist in F_2. Let $\{a_1, a_2, ..., a_m\}$ be a sibling group of $\dot{T}_1$ such that $a_1, a_2, ..., a_r$ share a minimal LCA l in F_2; neither a_1 nor a_2 is a child of l; a_i and a_j are not siblings in F_2, for all $1 \leq i < j \leq m$; and $F_2^{a_i}$ is not a component of F_2, for all $1 \leq i \leq m$. Then $\tilde{e}(T_1, T_2, F_2 \div \{e_x\}) = \tilde{e}(T_1, T_2, F_2) - 1$, for some $x \in \{a_1, a_2, B_1, B_2\}$.*

Proof. Consider an edge set E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E$ is an MAAF of T_1 and T_2 for some binary resolution F of F_2, and assume E is chosen so that $|E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}|$ is maximized. If $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \emptyset$, the lemma holds, so assume $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} = \emptyset$.

First we prove that there exist leaves $b'_1 \in F_2^{B_1}$ and $x \notin F_2^{B_1}$ such that $b'_1 \sim_{F \div E} x$. In particular, $b'_1 \sim_{F \div E} p_{a_1}$. Assume the contrary, that is, $b'_1 \not\sim_{F \div E} x$, for all leaves $b'_1 \in F_2^{B_1}$ and $x \notin F_2^{B_1}$. Then, by Lemma 2, expanding B_1 does not change $\tilde{e}(T_1, T_2, F_2)$, so we can assume F contains this expansion. Next we choose arbitrary leaves $b'_1 \in F_2^{B_1}$, $x \notin F_2^{B_1}$ and the first edge $f \in E$ on the path from b'_1 to x. By Lemma 1, $F \div E = F \div (E \setminus \{f\} \cup \{e_{B_1}\})$. This contradicts our choice of E.

We can prove that $a'_1 \sim_{F \div E} p_{a_1}$ for at least one leaf $a'_1 \in F_2^{a_1}$, $b'_2 \sim_{F \div E} p_{a_2}$ for at least one leaf $b'_2 \in F_2^{B_2}$, and $a'_2 \sim_{F \div E} p_{a_2}$ for at least one leaf $a'_2 \in F_2^{a_2}$, using similar arguments. Thus, we have $a'_1 \sim_{F \div E} p_{a_1} \sim_{F \div E} b'_1$ and $a'_2 \sim_{F \div E} p_{a_2} \sim_{F \div E} b'_2$.

Recall that a_1 and a_2 share a minimal LCA l, and neither a_1 nor a_2 is a child of l. Since the four subtrees $F_2^{a_1}$, $F_2^{B_1}$, $F_2^{a_2}$ and $F_2^{B_2}$ are disjoint, F_2 contains the triplet $a'_1b'_1|a'_2$, while F_1 contains the triplet $a'_1a'_2|b'_1$. By Observation 1, this implies that $a'_1 \sim_{F\div E} p_{a_1} \sim_{F\div E} b'_1 \not\sim_{F\div E} a'_2 \sim_{F\div E} p_{a_2} \sim_{F\div E} b'_2$. Since F_1 contains both $a'_1a'_2|b'_1$ and $a'_1a'_2|b'_2$, this in turn implies that the components of $F \div E$ containing a'_1, b'_1 and a'_2, b'_2 overlap in F_1. By Observation 2, this contradicts that $F \div E$ is an AF of T_1 and T_2. □

Theorem 1 assumes that some minimal LCA l of a subset of the current sibling group $\{a_1, a_2, ..., a_r\}$ exists, and that a_a and a_2 are not children of l. The following two lemmas cover the cases when l exists (in which case a_1 cannot be a child of l) and a_2 is a child of l, and when l does not exist. Note that a_2 being a child of l implies that $r = 2$.

Lemma 3 (One Child). *Let F_1 and F_2 be forests of rooted X-trees T_1 and T_2, respectively. Suppose F_1 consists of a tree $\dot{T}_1$ and a set of components that also exist in F_2. Let $\{a_1, a_2, ..., a_m\}$ be a sibling group of $\dot{T}_1$ such that $\{a_1, a_2\}$ is a maximal subset of nodes in $\{a_1, a_2, ..., a_m\}$ that share a minimal LCA l in F_2, a_1 is not a child of l, and a_2 is a child of l. Then $\tilde{e}(T_1, T_2, F_2 \div \{e_x\}) = \tilde{e}(T_1, T_2, F_2) - 1$, for some $x \in \{a_1, a_2, B_1\}$.*

Proof. Consider an edge set E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E$ is an MAAF of T_1 and T_2, for some binary resolution F of F_2, and assume E is chosen so that $|E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\}|$ is maximized. Assume $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\} = \emptyset$, because the lemma holds otherwise.

By the same arguments as in the proof of Theorem 1, there exist leaves $a'_1 \in F_2^{a_1}$, $a'_2 \in F_2^{a_2}$ and $b'_1 \in F_2^{B_1}$ such that $a'_1 \sim_{F\div E} p_{a_1} \sim_{F\div E} b'_1$ and $a'_2 \sim_{F\div E} p_{a_2}$.

Since a_1 and a_2 share a minimal LCA l and a_1 is not a child of l, the three subtrees $F_2^{a_1}$, $F_2^{B_1}$ and $F_2^{a_2}$ are disjoint. Thus, F_2 contains the triplet $a'_1b'_1|a'_2$, while

F_1 contains the triplet $a_1'a_2'|b_1'$. By Observation 1, this implies that $a_1' \nsim_{F \div E} a_2'$ (and thus $a_2' \nsim_{F \div E} a_1$) because $a_1' \sim_{F \div E} b_1'$. Thus, we also have $a_2' \nsim_{F \div E} x$, for all $x \in F_2^l \setminus F_2^{a_2}$. Indeed, if $x \in F_2^{a_1}$, then $a_2' \nsim_{F \div E} x$ because $a_2' \nsim_{F \div E} a_1$ and $a_2' \notin F_2^{a_1}$. If $x \notin F_2^{a_1}$, the components of $F \div E$ containing a_1', b_1' and a_2', x would overlap in F_1, which contradicts Observation 2 because $F \div E$ is an AF of T_1 and T_2.

Now we choose an arbitrary leaf $x \in F_2^l \setminus F_2^{a_2}$ and the first edge $f \in E$ on the path from a_2' to x. By Lemma 1, we have $F \div E = F \div (E \setminus \{f\} \cup \{e_{a_2}\})$, which contradicts the choice of E. $\square$

Lemma 4 (Isolated Siblings). *Let F_1 and F_2 be forests of rooted X-trees T_1 and T_2, respectively. Suppose F_1 consists of a tree $\dot{T}_1$ and a set of components that also exist in F_2. Let $\{a_1, a_2, ..., a_m\}$ $(m \geq 2)$ be a sibling group of $\dot{T}_1$ such that $a_1 \nsim_{F_2} a_i$ for all $i \neq 1$, $a_2 \nsim_{F_2} a_j$ for all $j \neq 2$, and assume $F_2^{a_i}$ is not a component of F_2, for all $1 \leq i \leq m$. Then $\tilde{e}(T_1, T_2, F_2 \div \{e_x\}) = \tilde{e}(T_1, T_2, F_2) - 1$, for some $x \in \{a_1, a_2\}$.*

Proof. Consider an edge set E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E$ is an MAAF of T_1 and T_2, for some binary resolution F of F_2, and assume E is chosen so that $|E \cap \{e_{a_1}, e_{a_2}\}|$ is maximized. Assume $E \cap \{e_{a_1}, e_{a_2}\} = \emptyset$ because the lemma holds otherwise.

By the same arguments as in the proof of Theorem 1, there exist leaves $a_1' \in F_2^{a_1}$ and $a_2' \in F_2^{a_2}$ such that $a_1' \sim_{F \div E} a_1$ and $a_2' \sim_{F \div E} a_2$. Since $\{a_1, a_2, ..., a_m\}$ is a sibling group of F_1 but $a_1 \nsim_{F_2} a_i$ for all $i \neq 1$, and $a_2 \nsim_{F_2} a_j$ for all $j \neq 2$, we must have $a_1' \nsim_{F \div E} x$, for all leaves $x \notin F_2^{a_1}$, or $a_2' \nsim_{F \div E} y$, for all leaves $y \notin F_2^{a_2}$. Otherwise, the components of $F \div E$ containing a_1', x and a_2', y would overlap in F_1, contradicting Observation 2. W.l.o.g., assume the former is true. Since $F_2^{a_1}$ is not a component of F_2, there exists a leaf $z \notin F_2^{a_1}$ such that $a_1 \sim_{F_2} z$ and, hence, $a_1' \sim_{F_2} z$. For each such leaf z, the path from a_1' to z in F contains an edge in E because $a_1' \nsim_{F \div E} z$, and this edge does not belong to $F_2^{a_1}$ because $a_1' \sim_{F \div E} a_1$. We pick an arbitrary such leaf z, and let f be the first edge in E on the path from a_1' to z. The edges e_{a_1} and f satisfy the conditions of Lemma 1, that is, $F \div E = F \div (E\backslash\{f\} \cup \{e_{a_1}\})$. This contradicts the choice of E. $\square$

For both Lemma 3 and Lemma 4, the candidate edge sets to cut are subsets of the candidate edge set in Theorem 1. Thus, Lemmas 3 and 4 can be considered special cases of Theorem 1. In other words, we can always cut one of e_{a_1}, e_{a_2}, e_{B_1} and e_{B_2} to make progress towards an MAAF of F_1 and F_2.

3.2 Algorithm for Branching Phase

Whidden et al. [21] and van Iersel et al. [27] used similar arguments as the ones in the previous section to prove that cutting one of e_{a_1}, e_{a_2}, e_{B_1} and e_{B_2} makes progress towards an MAF of T_1 and T_2. They used this to obtain a simple 4-way branching algorithm to compute the SPR distance between two multifurcating trees. We use this algorithm as the first phase of our hybridization algorithm and review it here to highlight the changes necessary to compute an MAAF instead of an MAF.

As is customary for FPT algorithms, we focus on the decision version of the problem: "Give two multifurcating rooted X-trees T_1 and T_2 and a parameter k_p, is it possible to get an AF of T_1 and T_2 by cutting no more than $k_p + 1$ edges from T_2?" To compute the MAF of two trees, we start with $k_p = 0$ and increase it until we receive an affirmative answer. This does not increase the running time of the algorithm by more than a constant factor because the running time depends exponentially on k_p.

The branching algorithm is recursive. Each invocation takes two forests F_1 and F_2 of T_1 and T_2 and a parameter k as inputs. We denote such an invocation by $\textsc{Maf}(F_1, F_2, k)$. (Or we denote such an invocation by $\textsc{Maaf}(T_1, T_2, F_1, F_2, k)$ with T_1 and T_2 for refinement phase, when we are computing an MAAF. The difference is shown in the algorithm's Step 2 below.) The invocation returns "Yes" if there exists an

MAF (MAAF) of F_1 and F_2 that can be obtained by cutting at most k edges in F_2. Otherwise it returns "No". We maintain two sets of labelled nodes: R_d (roots-done) contains the roots of F_0, and R_t (roots-todo) contains roots of (not necessarily maximal) subtrees that agree between $\dot{T}_1$ and $\dot{F}_2$. We refer to the nodes in these sets by their labels. For the top-level invocation, $F_1 = \dot{T}_1 = T_1$, $F_2 = \dot{F}_2 = T_2$, and $F_0 = \emptyset$; R_d is empty, and R_t contains all leaves of T_1. The following are the steps of the invocation $\textsc{Maf}(F_1, F_2, k)$ (or $\textsc{Maaf}(T_1, T_2, F_1, F_2, k)$).

1. (Failure) If $k < 0$, there is no edge set E of at most k edges such that $F_2 \div E$ is an AF of T_1 and T_2. Return "No" in this case.

2. (Success) If $|R_t| = 0$, then F_0 is an AF of T_1 and T_2.

   - If the algorithm is used to compute an MAF, return "Yes" because F_0 is such an MAF.
   - If the algorithm is used to compute an MAAF, invoke the refinement phase (discussed in Chapters 4, 5, and 6) to check whether F_0 can be refined to an MAAF. Return the result of the refinement phase.

3. (Prune maximal agreeing subtrees) If there is a node $r \in R_t$ that is a root of $\dot{F}_2$, remove r from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F_0; then cut the edge e_r in $\dot{T}_1$ and return to Step 2. If no such root r exists, proceed to Step 4.

4. (Choose sibling group) Choose a sibling group $\{a_1, a_2, ..., a_m\}$ in $\dot{T}_1$ such that $a_1, a_2, ..., a_m \in R_t$. If the sibling group chosen in the parent invocation still has two or more members in $\dot{T}_1$, choose that sibling group.

5. (Grow agreeing subtrees) While there exist indices $1 \leq i < j \leq m$ such that a_i and a_j are siblings in $\dot{F}_2$, merge a_i and a_j as follows: remove a_i and a_j from R_t; resolve a_i and a_j in $\dot{T}_1$ and $\dot{F}_2$; label their new parent in both forests with (a_i, a_j);

and add it to R_t. The new node (a_i, a_j) becomes a member of the current sibling group and m decreases by 1. If $m = 1$ after resolving all such sibling pairs $\{a_i, a_j\}$, contract the parent of the only remaining member of the sibling group and return to Step 2; otherwise, proceed to Step 6.

6. (Choose minimal LCA) If $a_i \not\sim a_j$, for all $1 \leq i < j \leq m$, proceed to Step 7. Otherwise there exists a node l in $\dot{F}_2$ that is a minimal LCA of a group of nodes in the current sibling group. If the minimal LCA in the parent invocation still has two or more members of the current sibling group as descendants, choose l to be this node; otherwise choose l arbitrarily. Now rename the nodes in the sibling group $\{a_1, a_2, ..., a_m\}$ so that, for some $r \geq 2$, $a_1, a_2, ..., a_r$ are descendants of l; a_r is the only sibling that may be a child of l; and for $1 \leq i \leq r < j \leq m$, either the LCA of a_i and a_j is a proper ancestor of l or $a_i \not\sim_{F_2} a_j$.

7. (Cut edges) Make four recursive calls (shown in Figure 3.2):

   (i) $\textsc{Maf}(F_1, F_2 \div \{e_{a_1}\}, k-1)$.

   (ii) $\textsc{Maf}(F_1, F_2 \div \{e_{a_2}\}, k-1)$.

   (iii) $\textsc{Maf}(F_1, F_2 \div \{e_{B_1}\}, k-1)$.

   (iv) $\textsc{Maf}(F_1, F_2 \div \{e_{B_2}\}, k-1)$.

   Return “Yes” if one of these recursive calls does; otherwise return “No”.

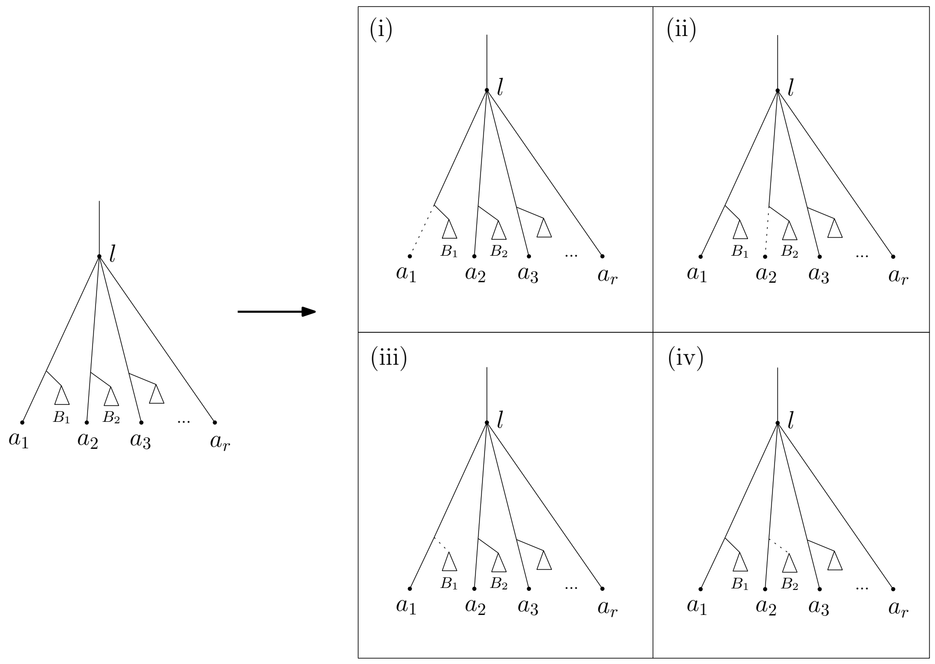


Figure 3.2: 4-way branching.

Chapter 4

Preparation for Refinement Phase

In the previous chapter, we proved that, if the branching phase finds an AF of T_1 and T_2 at all, then it finds an AF F that is obtained by cutting only edges that also belong to an edge set E such that $T_2 \div E$ is an MAAF of T_1 and T_2. However, it may be impossible to obtain an MAAF by cutting additional edges in F. The reason is that F is a binary resolution of the forest $T_2 \div E'$, where E' is the set of edges we cut to produce F, while $T_2 \div E$ may be a forest of a different binary resolution of $T_2 \div E'$.

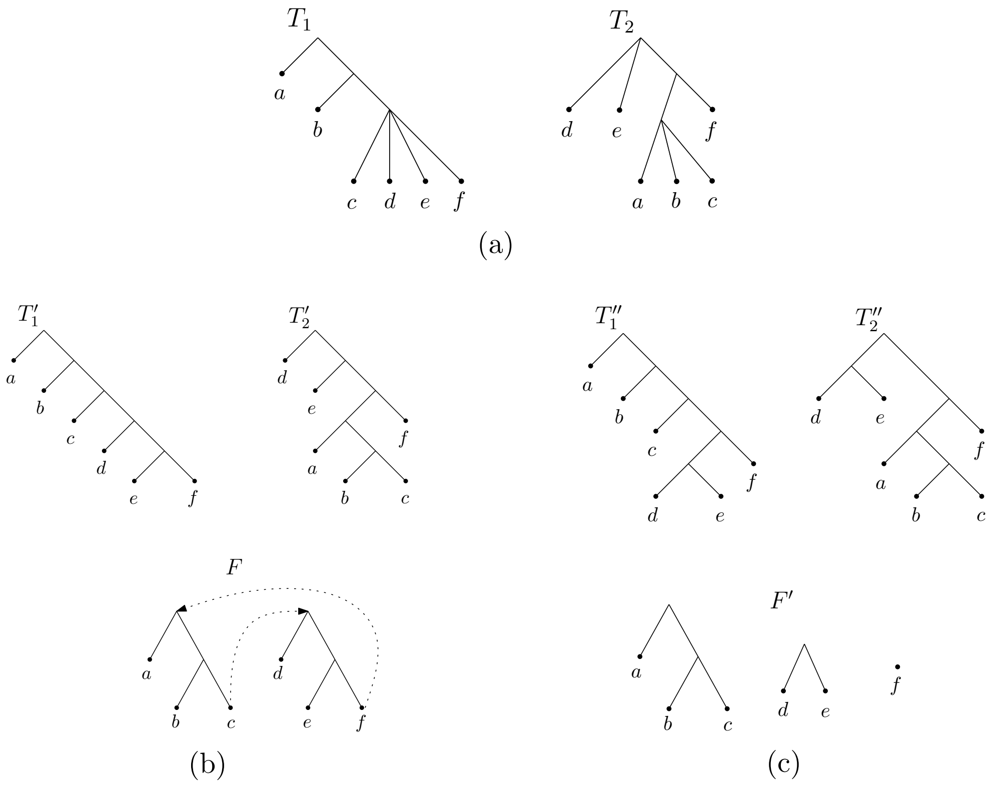


Figure 4.1: (a) Multifurcating rooted X-trees T_1 and T_2. (b) An AF F produced by the 4-way branching algorithm and the corresponding resolutions of T_1 and T_2. (c) An MAAF F' and the corresponding resolutions of T_1 and T_2.

An example for this is shown in Figure 4.1, where F' is an AF of T_1' and T_2', but it is not an AAF of T_1' and T_2'. The reason why this happens is that the algorithm resolves matching sibling pairs of the two trees without checking whether this introduces cycles. In fact, it is impossible to avoid creating these cycles because an invocation that performs a set of such resolutions has multiple descendant leaf invocations producing different forests and which set of resolutions creates cycles depends on the final agreement forest we consider. Thus, an intermediate step we employ before invoking the refinement step on any forest produced by the branching phase is to collapse all bifurcations in this forest that can be collapsed while maintaining that this forest is an agreement forest of $T_2 \div E'$. In this chapter, we discuss how we do this.

4.1 Restoring Multifurcations in an AF

In this section, we show how to construct a *multifurcating agreement forest* F_m from an agreement forest F that is "minimally resolved" in the sense that it retains all multifurcations compatible between T_1 and T_2. Thus, if E is an edge set such that $T_2 \div E$ is an MAAF of T_1 and T_2 and we cut only edges in E to produce F, then $T_2 \div E$ can be obtained by cutting additional edges in F_m.

The construction of F_m collapses edges in F as long as the resulting forest remains a forest of T_1 and T_2. F consists of a set of binary trees, which we denote as $F^1, F^2, ..., F^p$. Let $X^1, X^2, ..., X^p$ be the leaf sets of these trees. F_m has p components $F_m^1, F_m^2, ..., F_m^p$ with the same leaf sets $X^1, X^2, ..., X^p$. To construct F_m^i from F^i, for $1 \leq i \leq p$, we first extract $T_1|X^i$ and $T_2|X^i$ from T_1 and T_2. Since $T_1|X^i$ and $T_2|X^i$ are computed analogously, we discuss only how to compute $T_1|X^i$.

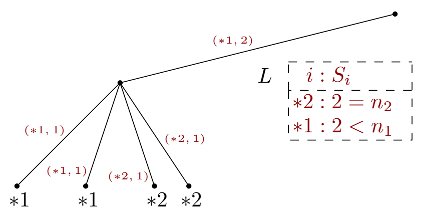
$T_1|X^i$ can be computed using one post-order traversal and one pre-order traversal of T_1. In fact, these two traversals can be used to compute all $T_1|X^1, T_1|X^2, ..., T_1|X^p$ at the same time. The idea is to mark each edge of T_1 to show which subtree $T_1(X^i)$ it belongs to. Since F is an AF of T_1 and T_2, every edge of T_1 belongs to at most one such subtree. $T_1|X^i$ can then be obtained from $T_1(X^i)$ by suppressing degree-2 nodes, for all $1 \leq i \leq p$. We associate a pair (i, d_i) with each edge of T_1, where i is the index of the subtree $T_1(X^i)$ it belongs to, and d_i is the number of leaves in X^i that are descendants of this edge. To distinguish the component index i and the counter d_i, we add a $*$ before the index i in Figures 4.2, 4.3, 4.4 and 4.5. To compute this labelling of the edges of T_1, we need a table (see Figure 4.2(b)) that stores for each index $1 \leq i \leq p$, the number of leaves n_i in X^i. We construct $T_1|X^i$ as follows.

1. (Edge labelling) This traversal marks each edge of T_1 with a pair (i, d_i). Initially, we mark the parent edge of each leaf in X^i with $(i, 1)$. Then, for every non-leaf edge e, the pairs of its child edges are computed before processing e. To compute the pair of e, we construct a list L of length equal to the number of different component indices below e. Each node of L stores an index i and a value S_i that is the sum of the d_i labels of all child edges of e whose component labels equal i. For every node in the list, we compare S_i with n_i, the number of leaves in X^i. There can be at most one value S_i that is smaller than n_i (otherwise, two components of F overlap in T_1, contradicting Observation 2). If there is such a value, e's label becomes (i, d_i); otherwise, e remains unlabelled. An example of this procedure is shown by Figure 4.2(a).

2. (Extraction of subtrees) Given the edge labelling computed in the previous step, the subtrees $T_1(X^1), T_1(X^2), ..., T_1(X^p)$ are easily constructed using a top-down traversal of T_1. For each i, $T_i|X^i$ can be constructed by traversing $T_i(X^i)$ and suppressing degree-2 internal nodes in $T_1(X^i)$. These steps are shown in Figures 4.4 and 4.5.

Lemma 5. *$T_1|X^1, T_1|X^2, ..., T_1|X^p$ can be computed in $O(n)$ time.*

Proof. According to our construction steps, $T_1(X^i)$ is obtained by connecting nodes in X^i with edges in T_1; $T_1|X^i$ is obtained by suppressing degree-2 internal nodes in



(a)

$i : n_i$
$*1 : 4$
$*2 : 2$
$*3 : 1$
$*4 : 5$
$*5 : 1$

(b)

Figure 4.2: (a) Computing a label pair for a non-leaf edge to construct $T_1|X^i$. (b) A lookup table showing the size of every partition X^i.

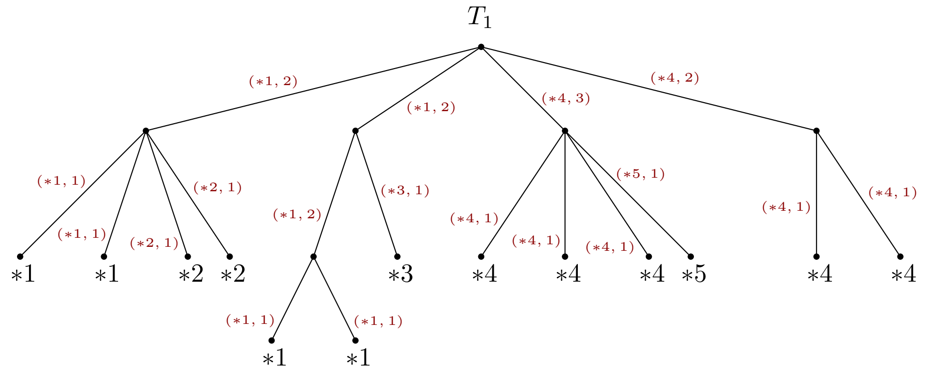


Figure 4.3: Labelling of the edges of T_1 to indicate their membership in trees $T_1(X^1)$, $T_1(X^2)$, ..., $T_1(X^p)$.

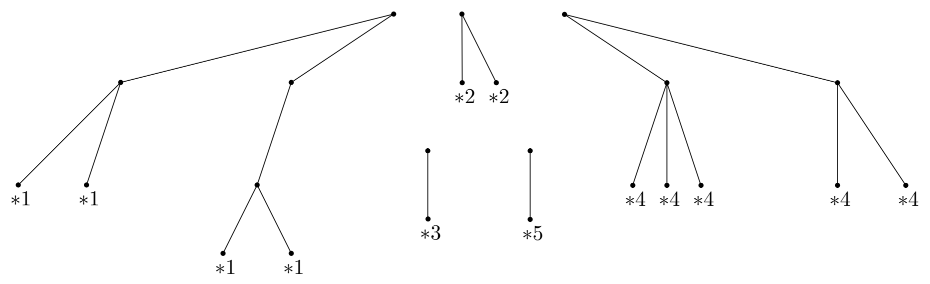


Figure 4.4: The trees $T_1(X^1), T_1(X^2), ..., T_1(X^p)$ extracted according to the labelling in Figure 4.3.

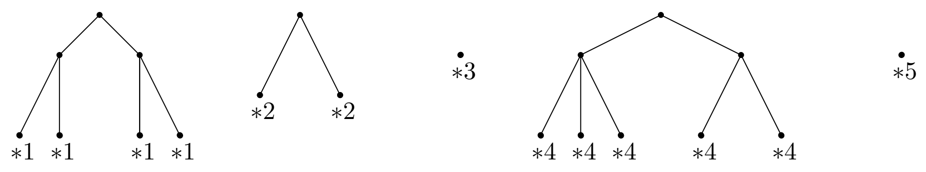


Figure 4.5: The trees $T_1|X^1, T_1|X^2, ..., T_1|X^p$ obtained from trees $T_1(X^1)$, $T_1(X^2)$, ..., $T_1(X^p)$ in Figure 4.4.

$T_1(X^i)$. The construction is correct, because it simply follows the definitions of $T_1(X^i)$ and $T_1|X^i$.

Next, we prove that the construction can be implemented in linear time. For the first traversal to take linear time, it suffices to prove that the parent edge of every leaf can be marked in constant time and any other edge e can be marked in $O(d)$ time if it has d child edges. The former is obvious. For the latter, the only challenge is the construction of the list L. We scan the child edges of e and decide for each such child edge with label (i, d_i) whether the list L already has an entry with index i. If so, we add d_i to the counter of this entry. Otherwise, we create a new entry. To do this in constant time per child edge, we allocate a table of size k whose ith entry points to the node in list L with index i if such an entry exists; otherwise the pointer is null. We allocate this table once and initialize all its pointers to null at the beginning of the traversal. Starting with a table of only null pointers, the construction of L is now easily implemented in constant time per child edge: if the ith pointer is null, we add a new node to the end of list L and change the ith pointer in the table to point to this node; otherwise, we update the value S_i of the node referenced by this pointer. In order to reuse the table for the next edge in the traversal, we have to reset the pointers to null. This can be done in time proportional to the length of the list L, which is at most d, by traversing L and for each node (i, S_i) in L, setting the ith pointer in the table to null. In the second traversal, each edge can be visited at most once as it can only belong to one component; each node can be visited at most once for contraction. Then this traversal also takes linear time. Thus, the two traversal takes time linear to size of T_1, i.e., $O(n)$ time. $\square$

In the remainder of this section, we show that F^i_m can be constructed from F^i, $T_1|X^i$ and $T_2|X^i$, in $O(|X^i|)$ time.

The construction starts with F^i and collapses unnecessary bipartitions bottom-up, based on the structure of $T_1|X^i$ and $T_2|X^i$. We associate a node set S_x with each node x in the construction procedure. The set may be open or closed. Each such set contains nodes that are siblings in both $T_1|X^i$ and $T_2|X^i$. An *open set* S_x is one whose members have siblings in both $T_1|X^i$ and $T_2|X^i$ that are not in S_x; a *closed set* is one whose members have no additional siblings in at least one of $T_1|X^i$ and $T_2|X^i$. Examples are shown in Figure 4.6. We construct these sets and use them to compute F^i_m as follows:

1. (Initial Step) Initially, we replace each leaf label of F^i with an open set that only includes the leaf itself. Throughout the iteration step, we maintain the property that each leaf of F^i is tagged with a node set and F^i is a binary resolution for both $T_1|X^i$ and $T_2|X^i$.

2. (Iteration Step) Choose the deepest leaf a in F^i. Its only sibling b must also be a leaf. Since F^i is a binary resolution of both $T_1|X^i$ and $T_2|X^i$, a and b must also be siblings in $T_1|X^i$ and $T_2|X^i$. Now, we remove a and b from F^i, and tag their parent node p_a as follows. There are three cases: (1) if both S_a and S_b are closed, we set $S_{p_a} = \{S_a, S_b\}$; (2) if both S_a and S_b are open, we set $S_{p_a} = S_a \bigcup S_b$; (3) if w.l.o.g. S_a is open and S_b is closed, we set $S_{p_a} = S_a \bigcup \{S_b\}$. To decide whether set S_{p_a} is open or closed, we check the parent nodes of a, b in $T_1|X^i$ and $T_2|X^i$. If both parent nodes have more than 2 children, S_{p_a} is *open.*

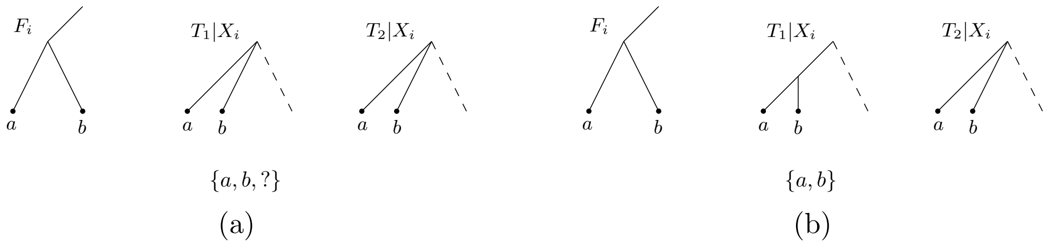


Figure 4.6: (a) An "open" set. (A question mark means the set is open.) (b) A "closed" set.

Otherwise, it is *closed*.

3. (End Condition) The procedure stops when F^i has only one node r. The nesting of sets in S_r represents the structure of F^i_m.

Figures 4.7 and 4.8 show an example of this procedure. Figure 4.9 shows the resulting F^i_m.

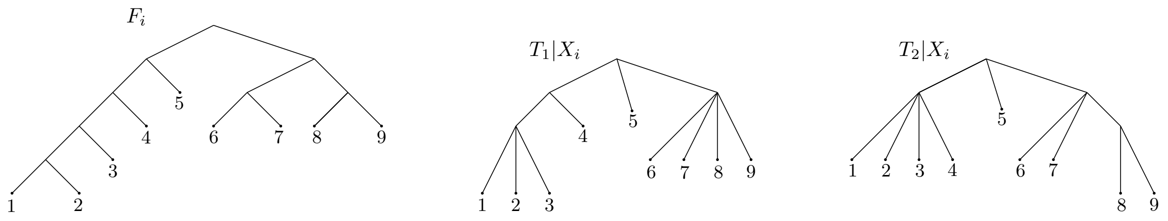


Figure 4.7: The inputs F_i, $T_1|X^i$ and $T_2|X^i$ for the construction of F^i_m in Figure 4.8.

The following observation follows immediately from the construction of F^i_m. Indeed, we create a closed set (and hence a node of F^i_m) only when we find a node of $T_1|X^i$ or $T_2|X^i$ whose descendant leaves are all in this set.

Observation 3. *For each internal node of $T_1|X^i$ or $T_2|X^i$, there is a corresponding internal node of F^i_m that has the same set of descendant leaves.*

Two important properties of F^i_m can be deduced from Observation 3.

Lemma 6. *(1) Every triplet $xy|z$ of $T_1|X^i$ or $T_2|X^i$ is also a triplet of F^i_m. (2) Every triplet $xy|z$ of F^i_m is also a triplet of at least one of $T_1|X^i$ and $T_2|X^i$.*

Proof. For Property (1), we prove that every triplet $xy|z$ of $T_1|X^i$ is also a triplet of F^i_m. (The proof for $T_2|X^i$ is analogous.) F^i is a binary resolution of $T_1|X^i$, so every triplet $xy|z$ of $T_1|X^i$ is also a triplet of F^i. As F^i_m is constructed by contracting edges in F^i, every triplet $xy|z$ of F^i becomes a triplet $xy|z$ or $x|y|z$ in F^i_m. Thus, every triplet $xy|z$ of $T_1|X^i$ should be of the form $xy|z$ or $x|y|z$ in F^i_m. Now assume for the sake of contradiction that some triplet $xy|z$ of $T_1|X^i$ becomes a triplet $x|y|z$

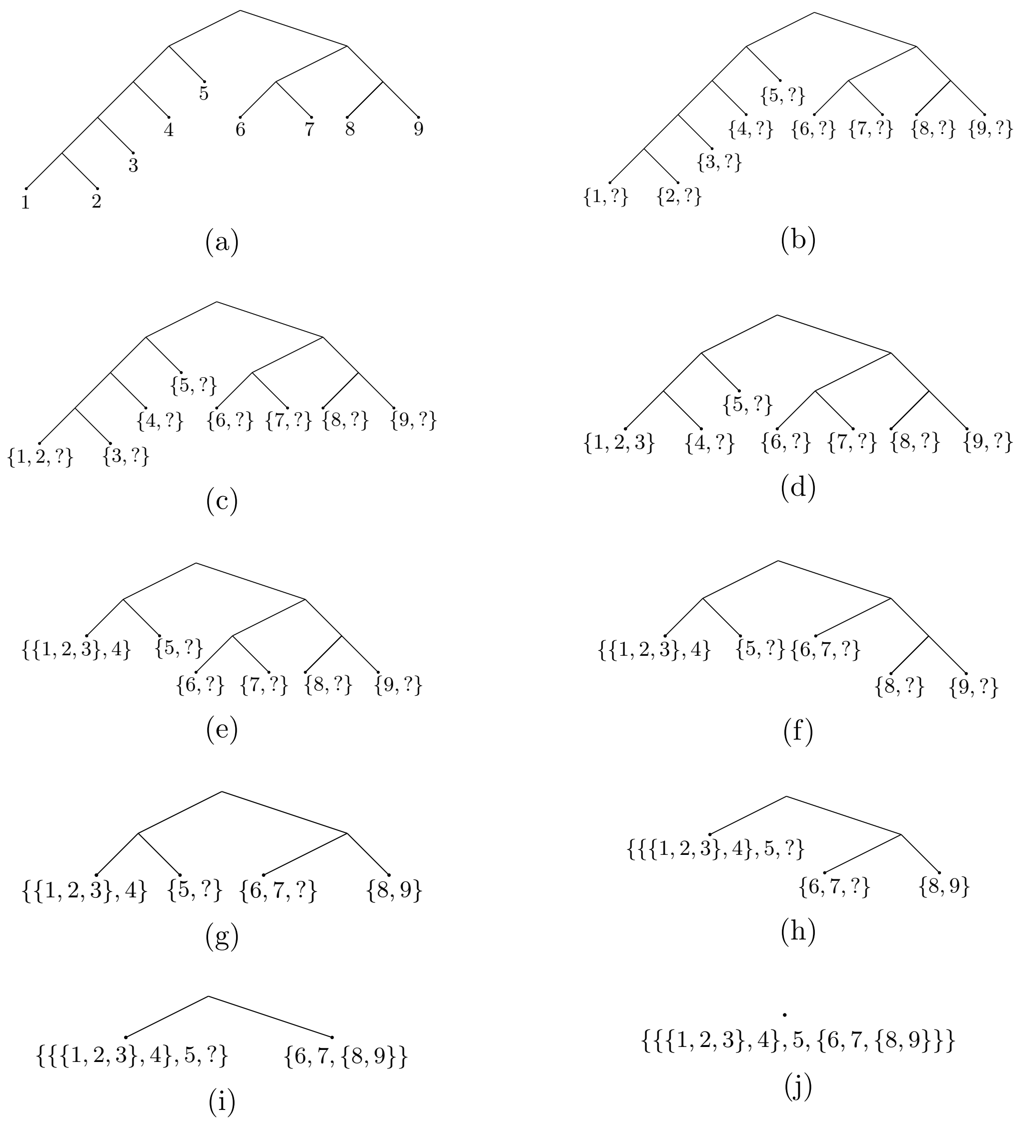


Figure 4.8: Detailed steps of collapsing F^i to construct F^i_m.

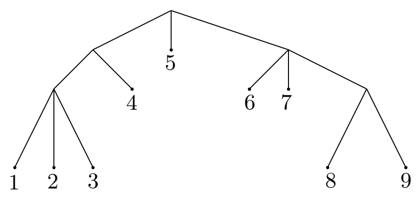


Figure 4.9: F^i_m.

in F^i_m. Let l_{xy} be the LCA of x and y in $T_1|X^i$. Note that z is not a descendant of l_{xy}. According to Observation 3, there is a corresponding internal node l'_{xy} of F^i_m with the same set of descendant leaves as l_{xy}. In particular, x and y are descendants of l'_{xy} and z is not. Thus, F^i_m contains the triplet $xy|z$, a contradiction.

Similarly, for Property (2), every triplet $xy|z$ of F^i_m is also a triplet of F^i, and every triplet $xy|z$ of F^i should be of the form $xy|z$ or $x|y|z$ in $T_1|X^i$ and $T_2|X^i$. Thus, a triplet $xy|z$ of F^i_m can only be of the form $xy|z$ or $x|y|z$ in $T_1|X^i$ and $T_2|X^i$. Now assume for the sake of contradiction that some triplet $xy|z$ of F^i_m is of the form $x|y|z$ in both $T_1|X^i$ and $T_2|X^i$. Then neither $T_1|X^i$ nor $T_2|X^i$ has an internal node that is an ancestor of x and y but not of z. Let l_{xy} be the LCA of x and y in F^i, and let l_{xyz} be the LCA of x, y and z in F^i. The bottom-up traversal of F^i processes l_{xy} before l_{xyz} and creates a set $S_{l_{xy}}$ containing x and y but not z. The children x' and y' of l_{xy} at this time are siblings also in $T_1|X^i$ and $T_2|X^i$. Let l_1 and l_2 be the LCAs of x, y and z in $T_1|X^i$ and $T_2|X^i$, respectively. Then x' and y' are children of l_1 and l_2, because $T_1|X^i$ and $T_2|X^i$ both have the triplet $x|y|z$. Since z is a descendant of neither x' nor y', l_1 and l_2 have additional siblings at the time we process l_{xy}, so the set $S_{l_{xy}}$ is open. The same argument applies to any ancestor u of l_{xy} that is a proper descendant of l_{xyz} because S_u contains x and y but not z. So l_1 and l_2 are also the parents of the children of u in $T_1|X^i$ and $T_2|X^i$ at the time we process u, and l_1 and l_2 must have at least one ancestor of z as an additional child. Therefore, S_u is open. This shows that the construction of F^i_m does not create a closed set containing x and y but not z, which contradicts that F^i_m contains the triplet $xy|z$. □

Lemma 7. *$F^1_m, F^2_m, ..., F^p_m$ can be computed in $O(n)$ time.*

Proof. To show the correctness of the procedure above, we need to prove the following three statements.

(i) F^i_m is a resolution of both $T_1|X^i$ and $T_2|X^i$.

(ii) F^i_m is minimally resolved, that is, there is no trees with fewer edges that satisfies (i).

(iii) There is only one minimal resolution for $T_1|X^i$ and $T_2|X^i$.

Statement (i) follows from Observation 3. For statement (ii), assume for the sake of contradiction that there is a less resolved tree $(F^i_m)'$ that satisfies (i). Then F^i_m is a resolution of $(F^i_m)'$ and there exists a triplet $xy|z$ of F^i_m whose leaves form the triplet $x|y|z$ in $(F^i_m)'$. According to Property (2) in Lemma 6, $xy|z$ is a triplet of at least one of $T_1|X^i$ and $T_2|X^i$. Thus, $(F^i_m)'$ cannot be a resolution of both $T_1|X^i$ and $T_2|X^i$, a contradiction. Statement (iii) can be proved analogously. Assume there are two different minimal resolutions $(F^i_m)_1$ and $(F^i_m)_2$ of $T_1|X^i$ and $T_2|X^i$. Since $(F^i_m)_1$ is a minimal resolution, every triplet of $(F^i_m)_1$ is either a triplet of $T_1|X^i$ or of $T_2|X^i$. As $(F^i_m)_1$ and $(F^i_m)_2$ are two different minimal resolutions, there exists such a triplet in $(F^i_m)_1$ that is not a triplet in $(F^i_m)_2$. Since this triplet is a triplet of at least one of $T_1|X^i$ and $T_2|X^i$, say $T_1|X^i$, $(F^i_m)_2$ cannot be a resolution of $T_1|X^i$, a contradiction.

Next we prove that the construction of each tree F^i_m can be implemented in $O(|X^i|)$ time, so the construction for all $F^1_m, F^2_m, ..., F^p_m$ takes linear time.

We perform at most $|X^i|$ merge operations (iteration steps), because each such operation decreases the number of leaves of F^i by one. If the merge operation's cost is constant, this proves that the cost of the whole procedure is $O(|X^i|)$. To achieve this, two data structures are necessary. (1) An array of linked lists to keep track of the deepest leaves. The length of the array is equal to the depth of F^i. The linked list at index i records the leaves at depth i. Thus, we can start from the last element of the array and keep track of the deepest leaves. To keep the time complexity constant, insertions are done at the head of each list. To achieve constant deletion in linked lists, a flag is used to mark if a node is deleted. In other words, we don't scan for a node in list to delete it immediately, the node will be deleted when we meet it. (2) A doubly linked list for each node set S_x. This allows us to merge these sets in constant time by concatenating their lists. Note that nested set can be represented by taking one linked list as another linked list's element. □

Combining Lemmas 5 and 7, we can construct F_m from an agreement forest F in $O(n)$ time.

4.2 Resolving Input Trees

A second preparation for the refinement step is to resolve T_1 and T_2 so that every internal node has the property that either all its children belong to the same tree $T_1(X^i)$ or $T_2(X^i)$, or they all belong to different such trees. Let $(T_1)_m$ be the resolution from T_1 and $(T_2)_m$ be the resolution from T_2. An example is shown in Figure 4.10. There is an internal node x of T_i that has a group of children $\{a_1, a_2, a_3, a_4, a_5\}$, whose membership in trees $T_1(X^i)$ and $T_2(X^i)$ is as shown. We resolve $\{a_1, a_2, a_3, a_4, a_5\}$ into two branches as shown in Figure 4.10(b). This kind of resolutions are in fact reducing the search space of refinement phase, since they exclude those resolutions of T_1 and T_2 which are inconsistent with F_m.

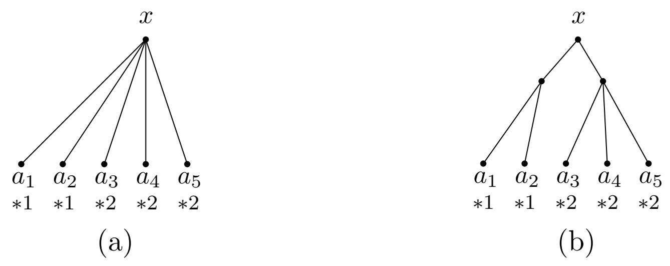


Figure 4.10: (a) Before the resolution from T_1 to $(T_1)_m$. (The number with a $*$ means the partition that the leaf above belongs to.) (b) After the resolution from T_1 to $(T_1)_m$.

$(T_1)_m$ can be computed from the labelled tree T_1 in Figure 4.3, using a post-order traversal on T_1. For each node x encountered, if it is a multifurcating node, we check the label pair (i, d_i) of all its child edges $e_{a_1}, e_{a_2}, ..., e_{a_m}$. First, we divide these child edges into groups according to their partition indices i. Correspondingly, x's children $a_1, a_2, ..., a_m$ are also divided into groups. Let $A_1, A_2, ..., A_p$ be these groups of children. For $1 \leq j \leq p$, if A_j only has one member, there is no need to expand it; otherwise, we resolve A_j. The result of resolving T_1 into $(T_1)_m$ is shown in Figure 4.11. The computation of $(T_2)_m$ is similar.

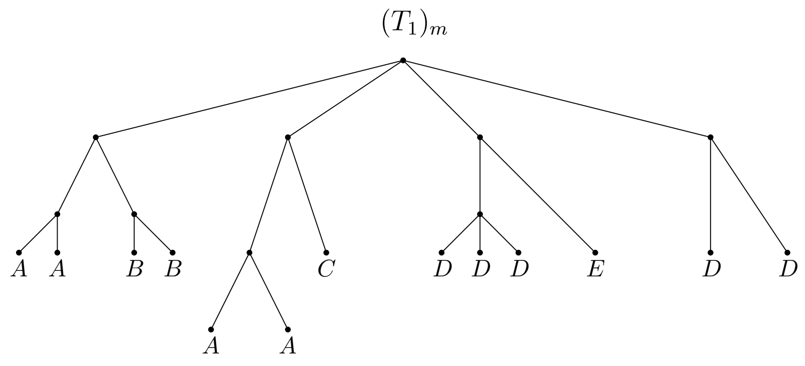


Figure 4.11: Resolve T_1 to $(T_1)_m$, according to partitions of F_m.

For each node we traverse on T_1, the cost can be paid by its children and each child pays for a constant cost. Note that the grouping requires some lookup table that is similar to the one for grouping child edges to compute the edge labels. In total, this post-order traversal takes $O(n)$ time. Hence, we have the following result.

Lemma 8. *$(T_1)_m$ and $(T_2)_m$ can be computed in $O(n)$ time.*

The following observation is an immediate consequence of the construction of $(T_1)_m$ and $(T_2)_m$.

Observation 4. *For every internal node in $(T_i)_m$, $i \in \{1, 2\}$, either all the children belong to the same tree $(T_i)_m(X^i)$, or each of them belongs to a different such tree.*

Chapter 5

A First Complete Hybridization Algorithm

In this chapter, we introduce the refinement algorithm and combine it with the 4-way branching algorithm from Chapter 3 to obtain a first complete FPT algorithm for computing the hybridization number of two multifurcating rooted X-trees. It will be obvious from the description of the algorithm that it also produces a corresponding MAAF. In the remainder of this chapter, we focus only on computing $hyb(T_1, T_2)$.

Similar with the 4-way branching algorithm in Section 3.2, we focus on the decision version of the problem: "Give two multifurcating rooted X-trees T_1 and T_2 and a parameter k_p, is $hyb(T_1, T_2) \leq k_p$?" To compute the hybridization number of two trees, we start with $k_p = 0$ and increase it until we receive an affirmative answer.

Every AAF of T_1 and T_2 can be computed by first computing an AF F of T_1 and T_2, and then cutting additional edges in F as necessary to break cycles in F's cycle graph. This suggests the following strategy to decide whether $hyb(T_1, T_2) \leq k_p$: First we use the branching algorithm to compute a set of AFs. As we proved in Chapter 3, this algorithm ensures that at least one AF it finds can be refined to an MAAF of T_1 and T_2. Second we try to refine each of the AFs to an AAF with at most $k_p + 1$ components by cutting additional edges, and return "Yes" if and only if this succeeds for at least one of the AFs.

Now let us call an invocation $\textsc{Maaf}(T_1,\ T_2,\ F_1,\ F_2,\ k)$ *viable* if there exists an MAAF F of T_1 and T_2 that is a forest of F_2. The top-level invocation $\textsc{Maaf}(T_1,\ T_2,\ T_1,\ T_2,\ k_p)$ is always viable and by Theorem 1, we have the following observation.

Observation 5. *Every viable invocation* $\textsc{Maaf}(T_1, T_2, F_1, F_2, k)$ *that is not a leaf* *invocation has a viable child invocation.*

In the remainder of this chapter, we introduce the refinement procedure Refine($\cdot$) that works on the multifurcating agreement forest F_m, and partially resolved trees $(T_1)_m$ and $(T_2)_m$ obtained from F_m, T_1, and T_2 using the procedure in Section 4.2. To make notation easier, we simply use F to refer to F_m, T_1 to refer to $(T_1)_m$, and T_2 to refer to $(T_2)_m$ in this chapter. The organization is as following: Section 5.1 introduces an expanded cycle graph G_F^* based on an AF F and correspondingly resolved T_1, T_2. Section 5.2 defines essential components for cycles in G_F^*, and their exit nodes. It shows that for some essential component C in a cycle O, cutting all edges from C's exit node in O to C's root makes progress towards an MAAF. We call cutting these edges *fixing* the exit node. Section 5.3 identifies potential exit nodes and fixes a subset of them to break cycles in G_F^*. Section 5.4 summarizes the refinement algorithm and includes the analysis and correctness proof of the algorithm. The refinement procedure, including the definition of the expanded cycle graph, is directly inspired by the refinement procedure in the hybridization algorithm for binary trees of [18]. However, multifurcations complicate the correctness proof substantially.

5.1 Expanded Cycle Graph

The *expanded cycle graph* G_F^* of a multifurcating agreement forest F of two multifurcating rooted trees T_1 and T_2 is a supergraph of F with the same vertex set as F, i.e. $G_F^* \supseteq F$; see Figure 5.2(c). In addition to the edges of F, G_F^* contains two *hybrid edges* per component (except the component with root ρ) in F, one induced by T_1 and one induced by T_2. To define the hybrid edges, we define mappings from nodes of F to nodes of T_1 and T_2 and vice versa. As in the binary case [18], we map each node x in F to nodes $\phi_1(x)$ in T_1 and $\phi_2(x)$ in T_2 such that $\phi_i(x)$ is the lowest common ancestor of all labelled leaves in T_i that are descendants of x in F, for $i \in \{1, 2\}$.

The difference to the binary case is that several nodes in F may be mapped to the same multifurcating node in T_1 (see Figure 5.1). For the reverse mapping, we define a function $\phi_i^{-1}(\cdot)$ mapping nodes in T_i to nodes in F. $\phi_i^{-1}(x)$ is defined if and only if x is labelled or belongs to the path between two labelled nodes a and b in T_i such that $a \sim_F b$. In this case, $\phi_i^{-1}(x)$ is the node in F that is the lowest common ancestor of all labelled leaves y in T_i^x such that $x \sim_F y$. Note that, different from the binary case, $\phi_i^{-1}(\phi_i(x))$ might be different from x, but $\phi_i^{-1}(\phi_i(x))$ is always defined. Observation 4 guarantees that the reverse mapping $\phi_i^{-1}(\cdot)$ is well defined.

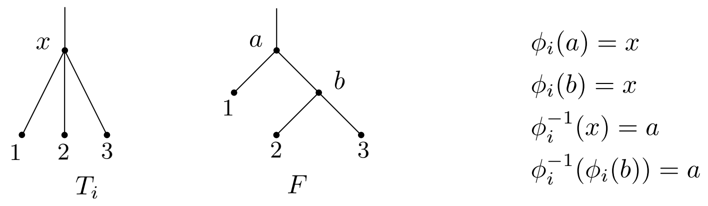


Figure 5.1: An example for mapping and reverse mapping based on the expanded cycle graph.

The hybrid edges in G_F^* are now defined as follows. There are two such edges per root node y of F, except ρ, one induced by T_1 and one induced by T_2. Let z_i be the lowest ancestor of $\phi_i(y)$ in T_i such that $\phi_i^{-1}(z_i)$ is defined. Then $(\phi_1^{-1}(z_1), y)$ is a T_1-*hybrid edge* and $(\phi_2^{-1}(z_2), y)$ is a T_2-*hybrid edge.* See Figure 5.2(c) for an illustration of these edges. Our first lemma shows that neither $\phi_1^{-1}(z_1)$ nor $\phi_2^{-1}(z_2)$ is a root of F.

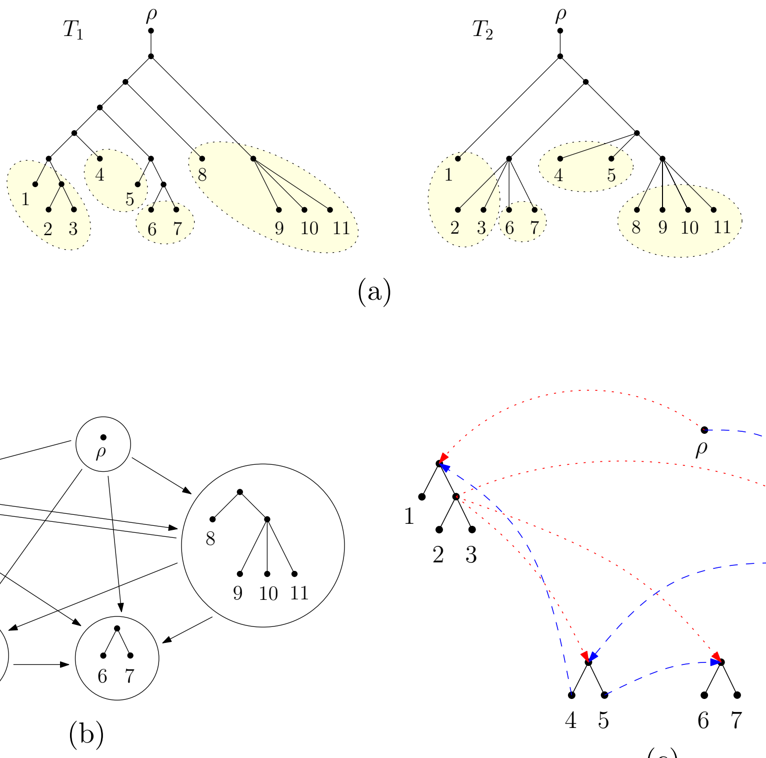
Lemma 9. *A root of F cannot be the tail of a hybrid edge.*

Proof. Assume there is a root $\phi_i^{-1}(z_i)$ in F that is the tail of a hybrid edge $(\phi_i^{-1}(z_i), y)$. Since z_i is the lowest ancestor of $\phi_i(y)$ in T_i such that $\phi_i^{-1}(z_i)$ is defined, let y' be a child of z_i in T_i such that y' is an ancestor of $\phi_i(y)$ in T_i. Note that either $\phi_i^{-1}(y')$ is not defined or $y' = \phi_i(y)$. Because $\phi_i^{-1}(z_i)$ is a root in F, it has at least two children a_1, a_2. These two children correspond to children a'_1 and a'_2 of z_i in T_i, that is, $\phi_i^{-1}(a'_1) = a_1$ and $\phi_i^{-1}(a'_2) = a_2$. Thus, in T_i, z_i has at least three children a'_1, a'_2 and y' because $\phi_i^{-1}(a'_1)$ and $\phi_i^{-1}(a'_2)$ are defined and $y \not\sim_F \phi_i^{-1}(z_i)$ but $a_1 \sim_F \phi_i^{-1}(z_i) \sim_F a_2$.

Thus, however, z_i has two children in the same tree $T_i(X^j)$ and one not in this tree. This contradicts Observation 4. $\square$

Our next lemma shows that the forest F is an AAF of T_1 and T_2 if and only if G^*_F contains no cycles, that is, we can use G^*_F in place of G_F to test whether F is acyclic. This lemma was shown by Whidden et al. [18] for binary trees. Their proof trivially extends to multifurcating trees.

Lemma 10. *G^*_F contains a cycle if and only if G_F does.*



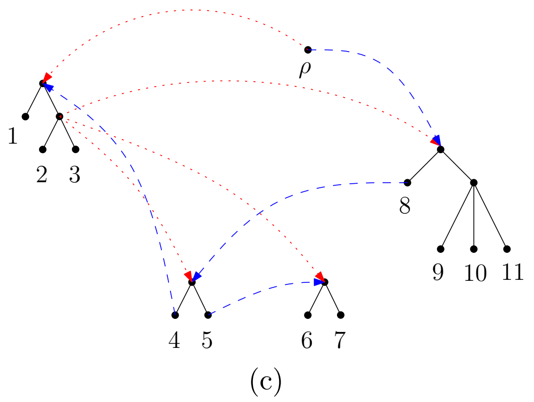


Figure 5.2: (a) Two trees T_1 and T_2. (b) A multifurcating agreement forest F of T_1 and T_2 and its cycle graph G_F. (c) The expanded cycle graph G^*_F. Dashed edges are T_1-hybrid edges, dotted ones are T_2-hybrid edges.

The final lemma of this section shows that G^*_F can be constructed from T_1, T_2, and F in linear time.

Lemma 11. *The expanded cycle graph G^*_F of a multifurcating agreement forest F of two multifurcating rooted trees T_1 and T_2 can be computed in $O(n)$ time.*

Proof. We construct G_F^* by adding hybrid edges to F. To add the hybrid edges induced by T_1, we perform a post-order traversal of T_1 to compute the mappings $\phi_1(\cdot)$ and $\phi_1^{-1}(\cdot)$, and the hybrid edges induced by T_1. Similarly, a post-order traversal of T_2 can be performed to compute $\phi_2(\cdot)$, $\phi_2^{-1}(\cdot)$, and the hybrid edges induced by T_2.

We assume that each labelled node of T_1 or T_2 stores a pointer to its counterpart in F and vice versa. Thus, for each leaf x, $\phi_1(x)$, $\phi_2(x)$, $\phi_1^{-1}(x)$, and $\phi_2^{-1}(x)$ are given. In addition, we associate a set L_x with each leaf x, where $L_x = \{x\}$ if x is a root of F, and $L_x = \emptyset$ otherwise. In general, after processing a node x of T_1, L_x stores the set of roots of F that map to descendants of x and have proper ancestors of x as the tails of their T_1-hybrid edges. All the roots in L_x share the same tail for their T_1-hybrid edges.

After initializing the information for the leaves of T_1, the post-order traversal computes the same information for the non-leaf nodes of T_1 and uses it to compute the T_1-hybrid edges in G_F^*. For a non-leaf node x with children $a_1, a_2, ..., a_m$ $(m \geq 2)$, the mapping $\phi_1^{-1}(a_i)$ and the root set L_{a_i} of a_i are computed before processing x, for all $1 \leq i \leq m$. Hence, we can use them to compute the mapping $\phi_1^{-1}(x)$ and the root set L_x. There are four cases.

(1) If each of $\phi_1^{-1}(a_1), \phi_1^{-1}(a_2), ..., \phi_1^{-1}(a_m)$ is undefined or a root of F, then $\phi_1^{-1}(x)$ is undefined (as x can belong to the path between two labelled nodes a and b in T_1 such that $a \sim_F b$ only if this is true for at least one of its children) and we set $L_x = L_{a_1} \cup L_{a_2} \cup ... \cup L_{a_m}$.

(2) If each of $\phi_1^{-1}(a_1), \phi_1^{-1}(a_2), ..., \phi_1^{-1}(a_m)$ is defined and not a root of F, $\phi_1^{-1}(a_1)$, $\phi_1^{-1}(a_2), ..., \phi_1^{-1}(a_m)$ must be in the same component of F (because of Observation 2) and their LCA p in F exists. We set $\phi_1^{-1}(x) = p$. If p is a root other than ρ, we set $L_x = \{p\}$; otherwise $L_x = \emptyset$.

(3) If only one of $\phi_1^{-1}(a_1), \phi_1^{-1}(a_2), ..., \phi_1^{-1}(a_m)$ is defined and not a root of F, let a_j be this child for some $1 \leq j \leq m$. We set $\phi_1^{-1}(x) = \phi_1^{-1}(a_j)$ and add a T_1-hybrid

edge $(\phi_1^{-1}(x), y)$ to G_F^* for every root y in L_{a_i}, for $1 \leq i \leq m$ and $i \neq j$. Then we set $L_x = \emptyset$.

(4) If more than one of $\phi_1^{-1}(a_1), \phi_1^{-1}(a_2), ..., \phi_1^{-1}(a_m)$ (but not all of them) is defined and not a root of F, assume $\phi_1^{-1}(a_1)$ and $\phi_1^{-1}(a_2)$ satisfy this condition. Then either $\phi_1^{-1}(a_1)$ and $\phi_1^{-1}(a_2)$ are in the same component of F, or $\phi_1^{-1}(a_1)$ and $\phi_1^{-1}(a_2)$ are in two different components of F. In the former case, we obtain a contradiction to Observation 4. In the latter case, the two components of F overlap in T_1, contradicting Observation 2. Therefore, this case cannot happen.

Correctness: The procedure correctly constructs G_F^* because it directly follows the definition of G_F^*. In particular, the correctness relies on the invariant that, after processing a node x, L_x stores the set of roots of F that map to descendants of x and have proper ancestors of x as the tails of their T_1-hybrid edges. Initially, for each leaf x, $L_x = \{x\}$ if x is a root of F; $L_x = \emptyset$ otherwise. So the invariant holds. Then for each non-leaf node x, the invariant is kept in all 3 possible cases. For Case (1), since $\phi_1^{-1}(x)$ is undefined, we keep looking for hybrid edges for the roots in $L_{a_1} \cup L_{a_2} \cup ... \cup L_{a_m}$ by setting $L_x = L_{a_1} \cup L_{a_2} \cup ... \cup L_{a_m}$. For Case (2), a possible new root p is introduced for which we have to find a hybrid edge, so we need to set $L_x = L_{a_1} \cup L_{a_2} \cup ... \cup L_{a_m} \cup \{p\}$. However, since $\phi_1^{-1}(a_1)$ is defined for all a_i, we have $L_{a_i} = \emptyset$ for all a_i. So we handle this case correctly. For Case (3), since $\phi_1^{-1}(x)$ is defined, we set $L_x = \emptyset$ because $L_{a_j} = \emptyset$ and every root y in L_{a_i}, for $1 \leq i \leq m$ and $i \neq j$, has the hybrid edge $(\phi_1^{-1}(x), y)$.

Running time: To compute the mapping $\phi_1^{-1}(x)$ from the children of x, we have to check the reverse mapping of each child. The cost is $O(1)$ per child. Since there are in total less than n children, the total cost is $O(n)$. Since the number of hybrid edges is twice the number of roots in F, and adding a hybrid edge takes constant time, the total cost of adding hybrid edges to F is no more than $O(n)$. When L_x is computed as the union of the root sets of x's children, the cost of concatenating these lists can also be paid by each child involved. As each node can be involved in the concatenation at most once, the cost is constant for each node, $O(n)$ in total. The running time of the traversal of T_2 is bounded by $O(n)$ using the same arguments. Hence, the entire algorithm takes linear time. □

5.2 Essential Components and Exit Nodes

In this section, we define the essential components of a cycle in G_F^* and their exit nodes. There two concepts are crucial for deciding which edges to cut in order to break cycles in G_F^*. We prove that, if F can be refined to an AAF of T_1 and T_2 with at most $k_p + 1$ components, this can be achieved by cutting only edges on the paths from exit nodes to the roots of their components in F.

Let H_1 be the set of T_1-hybrid edges in G_F^*, and let H_2 be the set of T_2-hybrid edges in G_F^*. Assume G_F^* contains a cycle O. Let $h_0, h_1, ..., h_{m-1}$ be the hybrid edges in O, and let $C_0, C_1, ..., C_{m-1}$ be the components of F connected by these hybrid edges. More precisely, using index arithmetic modulo m, we assume the tail and head of edge h_i belong to components C_i and C_{i+1}, respectively. The cycle O enters each component C_i at the head of edge h_{i-1} and leaves it at the tail of edge h_i. We say a component C_i is *essential to O* if $h_{i-1} \in H_1$ and $h_i \in H_2$ or vice versa. We say a component C is *essential* if it is essential to at least one cycle in G_F^*. A node x in a component C of F is an *exit node* of C, if C is essential to some cycle O in G_F^* and x is the tail of a hybrid edge in this cycle. Figure 5.3(c) illustrates these concepts.

Our first lemma in this section shows that there exits an exit node of an essential component such that cutting its parent edge in F reduces $\tilde{e}(T_1, T_2, F)$ by one, i.e., by cutting this edge, we make progress towards an MAAF of T_1 and T_2. This lemma was proved by Whidden et al. [18] for binary trees, and we verified that it also applies to multifurcating trees.

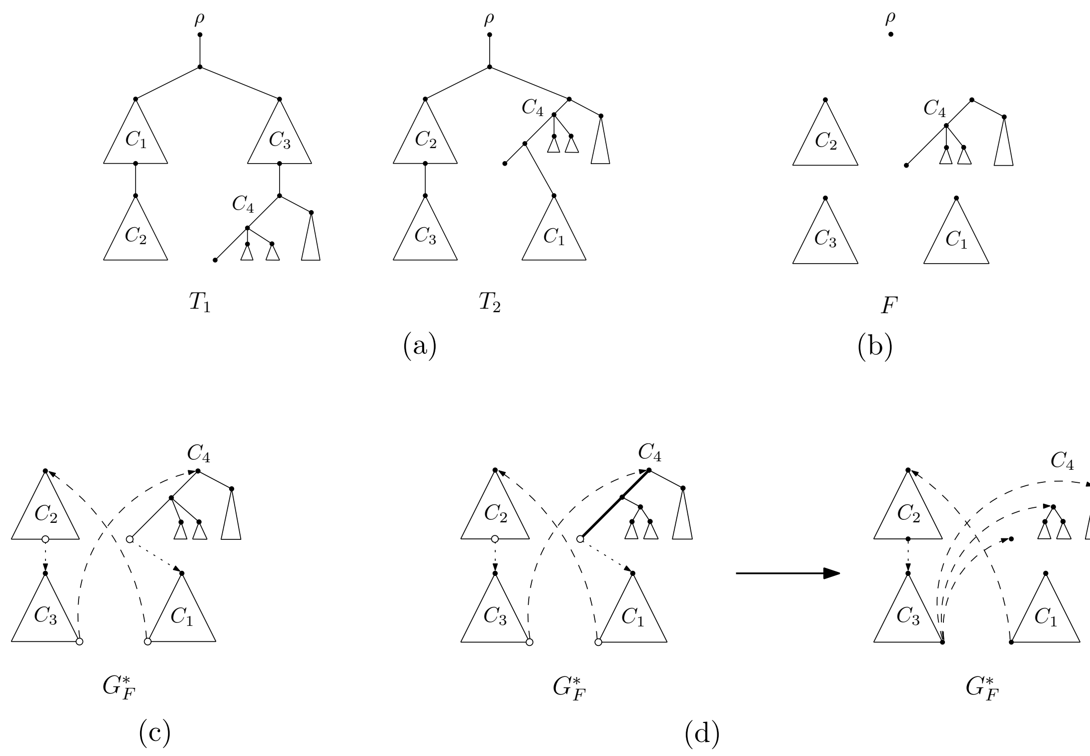


Figure 5.3: (a) Two trees T_1 and T_2. (b) A multifurcating agreement forest F of T_1 and T_2. (c) G_F^* (with ρ's component removed for clarity) contains a cycle of length 4. Dashed edges are T_1-hybrid edges, dotted ones are T_2-hybrid edges. White nodes indicate exit nodes. (d) Fixing the exit node of component C_4 (cutting the bold edges) destroys the cycle. Note that multifurcations on the path from an exit node to the root should be resolved. This doesn't change the results of our algorithms but it will help with the proof for Lemma 13.

Lemma 12. *Let O be a cycle in G_F^*, let $C_0, C_1, ..., C_{m-1}$ be its essential components, and let v_i be the exit node of component C_i in O, for all $0 \leq i \leq m-1$. Then $\tilde{e}(T_1, T_2, F \div \{e_{v_i}\}) = \tilde{e}(T_1, T_2, F) - 1$, for some $0 \leq i \leq m-1$.*

The following lemma is a more powerful version of Lemma 12. It shows that we can in fact make progress towards an MAAF by cutting all edges on the path from an appropriate exit node to the root of its component. And we resolve multifurcations on the path, which will be useful for Lemma 13's proof. We call this *fixing the exit node.* Breaking a cycle by fixing an exit node is illustrated in Figure 5.3(d).

Lemma 13. *Let O be a cycle in G_F^*, let $C_0, C_1, ..., C_{m-1}$ be its essential components,* *let v_i be the exit node of component C_i in O, let F^i be the forest obtained from F by fixing v_i, and let l_i be the length of the path in C_i from v_i to the root of C_i, for all $0 \leq i \leq m-1$. Then $\tilde{e}(T_1, T_2, F^i) = \tilde{e}(T_1, T_2, F) - l_i$, for some $0 \leq i \leq m-1$.*

Proof. This statement can be proved by induction on $\tilde{e}(T_1, T_2, F)$. By Lemma 12, there exists some exit node v_i such that $\tilde{e}(T_1, T_2, F') = \tilde{e}(T_1, T_2, F) - 1$, where $F' = F \div \{e_{v_i}\}$. Cutting e_{v_i} splits C_i into two components A_i and B_i that contain the leaves in $C_i^{v_i}$ and in $C_i \setminus C_i^{v_i}$, respectively. Before cutting e_{v_i}, we resolve v_i's siblings so that v_i's parent node p_{v_i} becomes binary. This won't cause any problem because of Lemma 2.

If $\tilde{e}(T_1, T_2, F) = 1$, then $\tilde{e}(T_1, T_2, F') = \tilde{e}(T_1, T_2, F) - 1 = 0$. This means F' is an AAF of T_1 and T_2, and hence the path from v_i to r_i only contains e_{v_i}. Thus, the statement holds for $\tilde{e}(T_1, T_2, F) = 1$.

If $\tilde{e}(T_1, T_2, F) > 1$, the lemma holds for F' with inductive hypothesis. If $l_i = 1$, then the path from v_i to r_i only contains e_{v_i}. Since $F' = F^i$ and $\tilde{e}(T_1, T_2, F') = \tilde{e}(T_1, T_2, F) - 1$, the lemma holds in this case. If $l_i > 1$, then $C'_0, C'_1, ..., C'_{i-1}, C'_i, C'_{i+1}, ..., C'_{m-1} = C_0, C_1, ..., C_{i-1}, B_i, C_{i+1}, ..., C_{m-1}$ is a cycle O' in $G^*_{F'}$. For $j \neq i$, the exit node v'_j of C'_j is v_j; the exit node v'_i of C'_i is v_i's sibling in C_i (note that p_{v_i} is resolved to binary right before cutting e_{v_i}). By the inductive hypothesis, $\tilde{e}(T_1, T_2, F'^j) = \tilde{e}(T_1, T_2, F') - l'_j$ for some j. We distinguish two cases. If $j \neq i$, then $l'_j = l_j$ and $F'^j = F^j \div \{e_{v_i}\}$. Then we have $\tilde{e}(T_1, T_2, F^j) - 1 \leq \tilde{e}(T_1, T_2, F^j \div \{e_{v_i}\}) = \tilde{e}(T_1, T_2, F'^j) = \tilde{e}(T_1, T_2, F') - l'_j = \tilde{e}(T_1, T_2, F) - l_j - 1$. Hence, $\tilde{e}(T_1, T_2, F^j) \leq \tilde{e}(T_1, T_2, F) - l_j$. Since F^j is obtained by cutting l_j edges in F, we also have $\tilde{e}(T_1, T_2, F^j) \geq \tilde{e}(T_1, T_2, F) - l_j$. Thus, the statement holds in this case. If $j = i$, then $l_j = l'_j + 1$ and $F'^j = F^j$. Thus, $\tilde{e}(T_1, T_2, F^j) = \tilde{e}(T_1, T_2, F'^j) = \tilde{e}(T_1, T_2, F') - l'_j = \tilde{e}(T_1, T_2, F) - 1 - l'_j = \tilde{e}(T_1, T_2, F) - l_j$. Therefore, the statement holds in this case as well. $\square$

5.3 Potential Exit Nodes

Exit nodes of F are defined as the tails of hybrid edges in G_F^* that belong to some cycle O in G_F^*. Hence, the set of exit nodes is a subset of tails of hybrid edges in G_F^*, and we call the tails of hybrid edges *potential exit nodes.* If F has $k'+1$ components, there are $2k'$ potential exit nodes because the component with root ρ doesn't have any incoming hybrid edges. If F is a forest produced by the branching phase of our algorithm, it has at most k_p+1 components, and thus at most $2k_p$ potential exit nodes. The following lemma shows that the set of potential exit nodes of the forest obtained by fixing a potential exit node in F is a subset of F's potential exit nodes. This property is very important because it guarantees that we are not creating new potential exit nodes when fixing potential exit nodes. In particular, this lemma will be used to prove that, if F can be refined to an AAF with at most k_p+1 components, then fixing an appropriate subset of potential exit nodes produces such a forest.

Lemma 14. *Let F be an agreement forest of two trees T_1 and T_2, let V be the set of potential exit nodes of F, and let v be an arbitrary node in V. Let F' be the forest obtained from F by fixing v, and let V' be the set of its potential exit nodes. Then $V' \subset V$.*

Proof. Let r be the root of the component of F containing v. Let (u_1, r) and (u_2, r) be hybrid edges in G_F^* induced by T_1 and T_2, respectively. In other words, u_1 and u_2 are the potential exit nodes for root r. By fixing v, the root r is removed while v becomes a new root in F', as do the pendent nodes r_1, r_2, ..., r_p of the path from v to r in F. Let V_1' be the set of potential exit nodes corresponding to roots that F' shares with F. Let V_2' be the set of potential exit nodes corresponding to new roots of F' (r_1, r_2, ..., r_p and v). Clearly, $V_1' \cup V_2' = V'$. For any root shared by F' and F, its two potential exit nodes will not change unless the potential exit nodes are on the path from v to r in F and these nodes are contracted when fixing v. In that case, the new potential exit nodes are u_1 and u_2 because of the definition of hybrid edges. Thus, $V_1' \subseteq V$. For any new root, its two potential exit nodes are u_1 and u_2, according to the definitions of hybrid edges and potential exit nodes. Thus, $V_2' \subseteq V$. This shows that $V' \subseteq V$. To finish the proof, it suffices to observe that $v \in V$ but $v \notin V'$ because v is a root of F' and thus, by Lemma 9, cannot be the tail of a hybrid edge. Thus, $V_2' \subseteq V$ and $V_2' \neq V$, that is $V' \subset V$. $\square$

5.4 A Simple Refinement Algorithm

By Lemma 13, if F can be refined to an AAF F' with at most $k_p + 1$ components, we can do so by fixing an appropriate exit node in $F_0 = F$, then fixing an appropriate exit node in the resulting forest F_1, and so on until we obtain F'. Let $F_0 = F, F_1, ..., F_q = F'$ be the sequence of forests produced in this fashion. For $0 \leq i \leq q$, the exit nodes of F_i are included in the set of F_i's potential exit nodes and, by Lemma 14, these potential exit nodes of F_i are included in the set of F_0's potential exit nodes. Thus, F' can be obtained from F by choosing an appropriate subset of F's potential exit nodes and fixing them. Moreover, fixing a subset of exit nodes one node at a time produces the same forest as simultaneously cutting all edges in the union of the paths from these exit nodes to the roots of their components in F. Thus, our strategy to decide whether F can be refined to an AAF with at most $k_p + 1$ components is to consider every subset of potential exit nodes and check whether cutting the edges on the paths to the roots of their components yields such an AAF. The resulting algorithm is shown as Algorithm 1.

If F cannot be refined to an AAF with at most $k_p + 1$ components, G_F^* cannot be made acyclic by fixing any subset of potential exit nodes, and Algorithm 1 returns "No" in this case. Otherwise, G_F^* can be made acyclic for at least one subset of potential exit nodes. Thus, this implementation of Refine(T_1, T_2, F, k_p) is correct.

Algorithm 1 Refine(T_1, T_2, F, k_p)

```
Build the expanded cycle graph $G_F^*$ from $F$, $T_1$ and $T_2$, mark all the potential exit
nodes in $G_F^*$, and use $V$ to denote the set of these nodes;
for each subset $V_{sub}$ of the marked node set $V$ do
    Fix every node $v$ in $V_{sub}$;
    if the number of components in the resulting forest $F'$ is no more than $k_p + 1$
    and $G_{F'}^*$ is acyclic then
        Return "Yes";
    end if
end for
Return "No";
```

If F has $k' + 1 \leq k_p + 1$ components, there are at most $2^{2k'} \leq 2^{2k_p} = 4^{k_p}$ subsets of potential exit nodes to test by Refine(T_1, T_2, F, k_p). Thus, the running time of Refine(T_1, T_2, F, k_p) is $O(4^{k_p}n)$. Furthermore, since we use Algorithm 1 as a subroutine of Maaf(T_1, T_2, F_1, F_2, k_p), we call Refine(T_1, T_2, F, k_p) for each of the up to 4^{k_p} leaves of the recursion tree of the branching phase. This results in a complete fixed parameter algorithm with a running time of $O(4^{k_p}(n + 4^{k_p}n)) = O(16^{k_p}n)$. Hence, we obtain the following result.

Theorem 2. *For two multifurcating rooted trees T_1, T_2 and a parameter k_p, it takes $O(16^{k_p}n)$ time to decide whether $\tilde{e}(T_1, T_2, T_2) \leq k_p$.*

Chapter 6

An Improved Refinement Algorithm

In the simple refinement algorithm, we test every subset of potential exit nodes. To reduce the running time of the refinement phase from $O(4^{k_p}n)$ to $O(2^{k_p}n)$, and hence the running time of the entire MAAF algorithm from $O(16^{k_p}n)$ to $O(8^{k_p}n)$, we show how to mark only half of the potential exit nodes chosen so that an MAAF, if it exists, can be found by fixing a subset of marked potential exit nodes. This is the same idea used in [18] for binary trees, but multifurcations in our case pose some new challenges.

Marking only part of the potential exit nodes may create a situation where the branching phase finds an AF F of T_1 and T_2 that can be refined to an AAF F' of T_1 and T_2 with at most $k_p + 1$ components but cannot be refined to such a forest F' by fixing a subset of the marked potential exit nodes. The intuition why this isn't a problem is that, whenever there is a potential exit node u that should be fixed but is not marked, there is another branch in the branching phase that cuts e_u. Although the two branches lead to different AFs, the second branch can end with the AAF that the first branch has missed. While this is the intuition, we in fact cannot guarantee that for any missed AAF, there always exists another branch that allows us to find it. What we prove is that, if $\tilde{e}(T_1, T_2, T_2) \leq k_p$, then there exists a "canonical" AF F_C produced by the branching phase of our algorithm such that it can be refined to an AAF F'' of T_1 and T_2 with at most $k_p + 1$ components by fixing a subset of the marked potential exit nodes in F_C.

The rules for marking half of the potential exit nodes are as follows. The branching phase assigns a tag "T_1" or "T_2" to each component root other than ρ of each AF F it produces. Then, after constructing the expanded cycle graph $G^*_{F_m}$, we mark a potential exit node u if there is a T_i-hybrid edge (u, w) in $G^*_{F_m}$ such that w is tagged with "T_i". Since each component root w is tagged with either "T_1" or "T_2", only one potential exit node is marked per component while there are two potential exit nodes per component. Thus, we mark only half of the potential exit nodes.

We implement the tagging of component roots by augmenting Step 7 of the 4-way branching algorithm to tag the bottom endpoints of the edges cut in F_2. When a tagged root x loses a child c by cutting its parent edge e_c, x's tag is unchanged if x still has at least two children; otherwise, x is contracted into the remaining child and that child inherits x's tag. This guarantees that exactly the roots in the current forest F_2 are tagged.

6.1 Improved MAAF Algorithm

The following is the pseudo-code of the MAAF algorithm, which shows how to tag the component roots produced in Step 7. In the description of the algorithm, k is used to denote the parameter passed to the current invocation, while k_p is used to denote the parameter of the top-level invocation $\textsc{Maaf}(T_1, T_2, k_p)$. Thus, $k_p + 1$ is the maximal number of components that the final AAF is allowed to have.

1. (Failure) If $k < 0$, there is no edge set E of at most k edges such that $F_2 \div E$ is an AF of T_1 and T_2. Return "No" in this case.

2. (Refinement) If $|R_t| = 0$, then F_0 is an AF of T_1 and T_2. Then transform F_0 to F_m and resolve T_1, T_2 into $(T_1)_m$, $(T_2)_m$, as described in Chapter 4. Invoke an algorithm $\textsc{Refine}((T_1)_m, (T_2)_m, F_m, k_p)$ to decide whether F_m can be refined to an AAF of T_1 and T_2 with at most $k_p + 1$ components. Return the answer of $\textsc{Refine}((T_1)_m, (T_2)_m, F_m, k_p)$.

3. (Prune maximal agreeing subtrees) If there is a node $r \in R_t$ that is a root of $\dot{F}_2$, remove r from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F_0; then cut the edge e_r in $\dot{T}_1$ and return to Step 2. If no such root r exists, proceed to Step 4.

4. (Choose sibling group) Choose a sibling group $\{a_1, a_2, ..., a_m\}$ in $\dot{T}_1$ such that $a_1, a_2, ..., a_m \in R_t$. If the sibling group chosen in the parent invocation still has two or more members in $\dot{T}_1$, choose that sibling group.

5. (Grow agreeing subtrees) While there exist indices $1 \leq i < j \leq m$ such that a_i and a_j are siblings in $\dot{F}_2$, merge a_i and a_j as follows: remove a_i and a_j from R_t; resolve a_i and a_j in $\dot{T}_1$ and $\dot{F}_2$; label their new parent in both forests with (a_i, a_j); and add it to R_t. The new node (a_i, a_j) becomes a member of the current sibling group and m decreases by 1. If $m = 1$ after resolving all such sibling pairs $\{a_i, a_j\}$, contract the parent of the only remaining member of the sibling group and return to Step 2; otherwise, proceed to Step 6.

6. (Choose minimal LCA) If $a_i \not\sim a_j$, for all $1 \leq i < j \leq m$, proceed to Step 7. Otherwise there exists a node l in $\dot{F}_2$ that is a minimal LCA of a group of nodes in the current sibling group. If the minimal LCA in the parent invocation still has two or more members of the current sibling group as descendants, choose l to be this node; otherwise choose l arbitrarily. Now rename the nodes in the sibling group $\{a_1, a_2, ..., a_m\}$ so that, for some $r \geq 2$, $a_1, a_2, ..., a_r$ are descendants of l; a_r is the only sibling that may be a child of l; and for $1 \leq i \leq r < j \leq m$, either the LCA of a_i and a_j is a proper ancestor of l or $a_i \not\sim_{F_2} a_j$.

7. (Cut edges) Make four recursive calls (shown in Figure 3.2):

   (i) $\textsc{Maaf}(F_1, F_2 \div \{e_{a_1}\}, k-1)$, and tag a_1 with "T_2" in $F_2 \div \{e_{a_1}\}$.

   (ii) $\textsc{Maaf}(F_1, F_2 \div \{e_{a_2}\}, k-1)$, and tag a_2 with "T_2" in $F_2 \div \{e_{a_2}\}$.

(iii) $\textsc{Maaf}(F_1, F_2 \div \{e_{B_1}\}, k-1)$, and tag B_1 with "T_1" in $F_2 \div \{e_{B_1}\}$.

(iv) $\textsc{Maaf}(F_1, F_2 \div \{e_{B_2}\}, k-1)$, and tag B_2 with "T_1" in $F_2 \div \{e_{B_2}\}$.

Return "Yes" if one of these recursive calls does; otherwise return "No".

6.2 Correctness Proof

We assume from here on that $hyb(T_1, T_2) \leq k_p$ because otherwise $\textsc{Refine}$ $((T_1)_m$, $(T_2)_m$, F_m, $k_p)$ returns "No" for any AF F found in the branching phase. It means, the algorithm gives the correct answer if $hyb(T_1, T_2) > k_p$. Thus, it suffices to prove that the algorithm returns "Yes" if $hyb(T_1, T_2) \leq k_p$ to finish the correctness proof.

The proof is divided into two stages. In the first stage, we prove that $\textsc{Maaf}(T_1'$, T_2, $k_p)$ returns "Yes" if $hyb(T_1', T_2) \leq k_p$, where T_1' is a binary resolution of T_1 that has a particular MAAF of T_1 and T_2 as a forest. In the second stage, we argue that, if the algorithm is correct for T_1' and T_2, it must also be correct for T_1 and T_2.

6.2.1 Stage One

To prove that $\textsc{Maaf}(T_1', T_2, k_p)$ returns "Yes" if $hyb(T_1', T_2) \leq k_p$, we introduce the concept of a canonical agreement forest and modify the algorithm slightly.

First, we discuss the modification of the algorithm. This modification does not change its behaviours significantly but helps us with reasoning about its correctness. In the original algorithm, when cutting an edge e_x, $x \in \{a_1, a_2\}$, in Step 7, x becomes a component root of F_2 that agrees with a subtree of F_1. Hence, the first thing Step 3 of the next recursive call does is to cut the parent edge of x in F_1. In the modified algorithm, we cut the parent edge of x in F_1 immediately after cutting the parent edge of x in F_2, as part of Step 7, instead of postponing the cutting of x's parent edge in F_1 to Step 3 of the next recursive call. Clearly, this modification does not change the set of AFs produced by the branching phase or the set of potential exit nodes that are marked in them.

Now, recall that not every AF found by the branching phase can be refined to an AAF of T_1' and T_2 by fixing marked potential exit nodes. We choose a *canonical AF* F_C' from among these AFs and prove that F_C' can be refined to an AAF of T_1' and T_2 with at most $k_p + 1$ components by fixing a subset of its marked potential exit nodes. We choose F_C' by specifying a sequence of recursive calls of procedure $\textsc{Maaf}(\cdot)$ that produce F_C' from T_2. This sequence of invocations form a path in the recursion tree of $\textsc{Maaf}(\cdot)$. Let F_1^i, F_2^i and k_i be the inputs to the ith invocation $\textsc{Maaf}(F_1^i, F_2^i, k_i)$ along this path. The first invocation is $\textsc{Maaf}(T_1', T_2, k_p)$, so $F_1^1 = T_1'$, $F_2^1 = T_2$, and $k_1 = k_p$. Assume we have constructed the path up to the ith invocation. The $(i+1)$st invocation is made in Step 7 of the ith invocation. Invocation $\textsc{Maaf}(T_1', T_2, k_p)$ is viable and by Observation 5, every viable invocation that is not a *leaf invocation* (an invocation that returns in Step 1 or 2) has at least one viable child invocation. If there is only one viable invocation made in Step 7 of the ith invocation $\textsc{Maaf}(F_1^i, F_2^i, k_i)$, then we choose this invocation as the $(i + 1)$st invocation $\textsc{Maaf}(F_1^{i+1}, F_2^{i+1}, k_{i+1})$. Otherwise we apply the following rules to choose $\textsc{Maaf}(F_1^{i+1}, F_2^{i+1}, k_{i+1})$ from among the viable child invocations of $\textsc{Maaf}(F_1^i, F_2^i, k_i)$.

Case 1: If both $\textsc{Maaf}(F_1^i \div \{e_{a_1}\}, F_2^i \div \{e_{a_1}\}, k_i - 1)$ and $\textsc{Maaf}(F_1^i \div \{e_{a_2}\}, F_2^i \div \{e_{a_2}\}, k_i - 1)$ are viable, we choose $\textsc{Maaf}(F_1^{i+1}, F_2^{i+1}, k_{i+1})$ from among them as follows: For $x \in \{a_1, a_2\}$, let F_x be the agreement forest found by tracing a path from $\textsc{Maaf}(F_1^i \div \{e_x\}, F_2^i \div \{e_x\}, k_i - 1)$ to a viable leaf invocation using recursive application of these rules, and let E_x be an edge set such that $F_x = F_1^i \div E_x$. Let $y = a_2$ if $x = a_1$, and $y = a_1$ if $x = a_2$. Now let $\phi_1(y)$ once again be the LCA in T_1' of all labelled leaves that are descendants of y in F_2^i, and let $\phi_x(y)$ be the LCA in F_x of all labelled leaves l that are descendants of y in F_1^i and such that the path from l to y does not contain an edge in E_x. In other words, $\phi_x(y)$ is the node of F_x that y is merged into by suppressing nodes during the sequence of recursive calls that produce F_x from F_2^i. Finally, let $d_1(y) = 0$ if $\phi_x(y)$ is a component root of F_x; otherwise, let $d_1(y) > 0$ be the distance from the root ρ of T_1 to $\phi_1(p(\phi_x(y)))$, where, $p(\phi_x(y))$ refers to the parent node of $\phi_x(y)$ in F_x. If $d_1(a_1) > d_1(a_2)$, we choose the invocation $\textsc{Maaf}(F_1^i \div \{e_{a_1}\}, F_2^i \div \{e_{a_1}\}, k_i - 1)$ as the $(i+1)$st invocation, i.e., $F'_C = F_{a_1}$; if $d_1(a_1) < d_1(a_2)$, we choose the invocation $\textsc{Maaf}(F_1^i \div \{e_{a_2}\}, F_2^i \div \{e_{a_2}\}, k_i - 1)$ as the $(i+1)$st invocation, i.e., $F'_C = F_{a_2}$; if $d_1(a_1) = d_1(a_2)$, we define $d_2(x)$ for $x \in \{a_1, a_2\}$, analogously to $d_1(x)$, using T_2 in place of T'_1. The only difference is that T_2 may be multifurcating while T'_1 is binary. When a_1 and a_2 share the same minimal LCA l, a_2 is a child of l, and l has at least 3 children, let C be the set of children of l that are not ancestors of a_1 or a_2. We resolve l so that a_2 and the nodes in C are siblings (shown in Figures 6.3(a) and 6.3(b)), and define $d_2(a_1)$ and $d_2(a_2)$ with respect to the resulting tree in this case. Note that this resolution is used only in the definition of $d_2(a_1)$ and $d_2(a_2)$, not in the algorithm. If $d_2(a_1) > d_2(a_2)$, we choose the invocation $\textsc{Maaf}(F_1^i \div \{e_{a_1}\}, F_2^i \div \{e_{a_1}\}, k_i - 1)$ as the (i+1)st invocation, i.e., $F'_C = F_{a_1}$; if $d_2(a_1) < d_2(a_2)$, we choose the invocation $\textsc{Maaf}(F_1^i \div \{e_{a_2}\}, F_2^i \div \{e_{a_2}\}, k_i - 1)$ as the $(i+1)$st invocation, i.e., $F'_C = F_{a_2}$; if $d_1(a_1) = d_1(a_2)$, choose an arbitrary one from among them as the $(i+1)$st invocation.

Case 2: If exactly one of $\textsc{Maaf}(F_1^i \div \{e_{a_1}\}, F_2^i \div \{e_{a_1}\}, k_i - 1)$ and $\textsc{Maaf}(F_1^i \div \{e_{a_2}\}, F_2^i \div \{e_{a_2}\}, k_i - 1)$ is viable, choose it.

Case 3: If neither $\textsc{Maaf}(F_1^i \div \{e_{a_1}\}, F_2^i \div \{e_{a_1}\}, k_i - 1)$ nor $\textsc{Maaf}(F_1^i \div \{e_{a_2}\}, F_2^i \div \{e_{a_2}\}, k_i - 1)$ is viable, both $\textsc{Maaf}(F_1^i \div \{e_{B_1}\}, F_2^i \div \{e_{B_1}\}, k_i - 1)$ and $\textsc{Maaf}(F_1^i \div \{e_{B_2}\}, F_2^i \div \{e_{B_2}\}, k_i - 1)$ must be viable. Then, choose one of these two invocations arbitrarily.

Lemma 15. *If $hyb(T'_1, T_2) \le k_p$, then F'_C can be refined to an AAF of T'_1 and T_2 with at most $k_p + 1$ components by fixing a subset of the marked potential exit nodes in F'_C.*

Proof. Let E' be an edge set such that $F' = F'_C \div E'$ is an AAF of T'_1 and T_2 with at most $k_p + 1$ components. By Lemma 13 and Lemma 14, we can assume E' is the union of paths from a subset of potential exit nodes to the roots of their respective components in F'_C. These potential exit nodes may or may not be marked. Note that since T'_1 is binary, F'_C, as its forest, is also binary. We say an edge is marked if it belongs to the path from a marked potential exit node to the root of its component, or it belongs to the path from the sibling of a marked potential exit node to the root of its component. Next, we prove that all edges in E' are marked. This implies that F' can be produced by fixing a subset of marked potential exit nodes in F'_C, so the lemma holds.

Assume for the sake of contradiction that there is an unmarked edge in E'. Since all ancestor edges of a marked edge are marked, there must exist a potential exit node u such that both u and its sibling u' are unmarked. The sequence of invocations that produce F'_C from T'_1 and T_2 gives rise to a sequence of edge cuts. We use F^i_1 and F^i_2 to refer to the forests obtained from T'_1 and T_2 after cutting the first i edges. (This is a change of notation, different from the definition of F'_C, where we used F^i_1 and F^i_2 to denote the input forests to the ith invocation.) Since F'_C is a refinement of F^i_1 and F^i_2, every node $x \in F'_C$ maps to the node in F^i_j which is the LCA of all descendant leaves of x in F'_C. This is analogous to the mappings $\phi_1(\cdot)$ and $\phi_2(\cdot)$ from F'_C to T'_1 and T_2. To avoid excessive notation, we refer to the nodes in F^i_1 and F^i_2 a node $x \in F'_C$ maps to simply as x.

With this notation, the common parent p_u of u and u' in F'_C is the LCA of both nodes in any forest F^i_j. Since u is a potential exit node of F'_C, there is at least one hybrid edge in $G^*_{F'_C}$ induced by cutting a pendant edge e_z of the path from u to p_u in some forest F^i_j. There may also be a hybrid edge induced by cutting a pendant edge $e_{z'}$ of the path from u' to p_u in some forest F^i_j. Either of the two types of edges are pendant to the path from u to u' in F^i_j. Let i be the highest index such that the ith edge we cut is pendant to the path from u to u' in F_1^{i-1} or F_2^{i-1}, and let e_y be this edge. Let $j \in \{1,2\}$ so that we cut e_y in F_j^{i-1}. The choice of index i implies that u and u' are siblings in F_1^i and F_2^i, and either u or u' is y's sibling in F_j^{i-1}. We use x to refer to this sibling, and x' to refer to x's sibling in F'_C (i.e. $x' = u'$ if $x = u$ and vice versa). We make two observations about x, x', and y:

(i) Since every exit node in F'_C has a binary parent node, fixing an exit node or its sibling produces the same forest. Hence F' can be obtained from F'_C by fixing a set of nodes that includes x or x'. In particular, $\tilde{e}(T'_1, T_2, F_j^i \div \{e_x\}) = \tilde{e}(T'_1, T_2, F_j^i \div \{e_{x'}\}) = \tilde{e}(T'_1, T_2, F_j^i) - 1$, for $j \in \{1,2\}$.

(ii) Since neither u nor u' is marked, x is not marked in F'_C, and hence, y is not tagged with "T_j" in F'_C.

Now we check each of the steps in the algorithm in which we cut e_y and prove that these observations lead to a contradiction. Thus, E' cannot contain an unmarked edge, and the lemma follows.

We consider a number of cases depending on whether we cut e_y in F_1^{i-1} or F_2^{i-1} and which step of the algorithm cuts this edge.

Scenario 1: If e_y is cut from F_1^{i-1}, y cannot be tagged with "T_1". Also note that F_1^{i-1} is binary, since T'_1 is binary.

1.1 If e_y is cut by an application of Step 3, y must be a root in F_2^{i-1}. It implies that there exists an $i' < i$ such that the i'th edge we cut is y's parent edge e'_y in $F_2^{i'-1}$ or an edge e_z such that z is an ancestor of y in $F_2^{i'-1}$. In the first case, e'_y must be cut as a B-type edge (e_{B_1} or e_{B_2}). Otherwise, e_y would be cut immediately after e'_y in Step 7 rather than in Step 3. Therefore, y is tagged with "T_1", a contradiction. In the second case, e_z must also be cut as a B-type edge and hence, z is tagged with "T_1". Otherwise, if $z \in \{a_1, a_2\}$, the subtree of z would

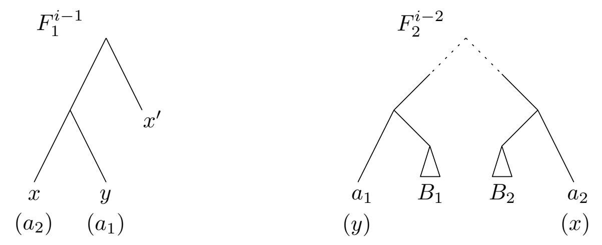


Figure 6.1: Scenario 1.2.1 in proof of Lemma 15.

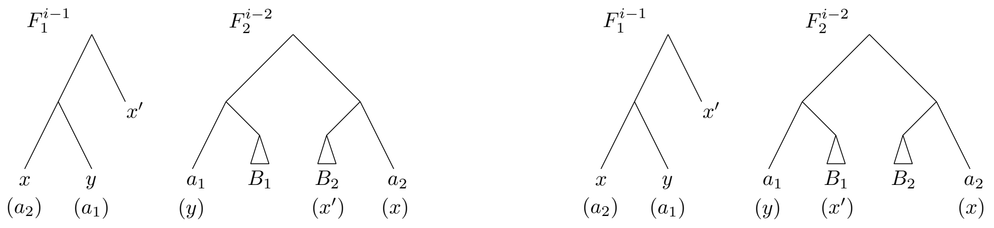
be a component of $F_2^{i'-1}$ that agrees with $F_1^{i'-1}$ and we would not cut edge e_y in F_1^{i-1}. We choose i' maximal such that we cut an ancestor edge e_z of y in $F_2^{i'-1}$. This implies that no edges on the path from z to y are cut but all pendant edges of this path are cut because y is a root of F_2^{i-1}. Thus, y inherits z's "T_1" tag, a contradiction.

1.2 If e_y is cut by an application of Step 7, e_y is the parent edge in F_1^{i-1} of a node $y \in \{a_1, a_2\}$ whose parent edge in F_2^{i-2} is the $(i-1)$st edge we cut.

1.2.1 Assume a_1, a_2 are in separate components in F_2^{i-2}. W.l.o.g., let $y = a_1$ and $x = a_2$. In the invocation $\textsc{Maaf}(F_1^{i-2}, F_2^{i-2}, k)$, cutting e_{a_1} is viable by the choice of F'_C and cutting e_{a_2} is also viable because $\tilde{e}(T'_1, T_2, F_2^{i-2} \div \{e_x\}) = \tilde{e}(T'_1, T_2, F_2^{i-2}) - 1$. Since x and x' are siblings in F'_C, and F'_C is a refinement of F_2^{i-2}, we have $y \nsim_{F_2^{i-2}} x \sim_{F_2^{i-2}} x'$. Since F_x is also a refinement of F_2^{i-2}, this implies that $y \nsim_{F_x} x'$. In particular, x' and y are not siblings in F_x. Because e_y is the only pendant edge of the path from x to x' in F_1^{i-1} and F_1^{i-1} is binary, this implies (see Figure 6.1) that either y is a root in F_x or the parent of y in F_x is a proper ancestor in F_1^{i-1} of the common parent of x and x' in $F_y = F'_C$. In both cases, $d_1(y) < d_1(x)$, which means we would have chosen invocation $\textsc{Maaf}(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1)$ instead of invocation $\textsc{Maaf}(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1)$ to be on the path to F'_C, a contradiction.

1.2.2 Assume a_1, a_2 are in the same component in F_2^{i-2}. If x' and y are not siblings in F_x, then the same arguments as in Scenario 1.2.1 also hold here. So assume

x' and y are siblings in F_x. Then, $d_1(y) = d_1(x)$. Since e_y is the last pendant edge of the path from x to x' in F_1^{i-1}, x and x' are siblings in F_2^{i-1}. There are two possibilities: either x and x' are siblings in F_2^{i-2} already or e_y is the last pendant edge of the path from x to x' in F_2^{i-2}.

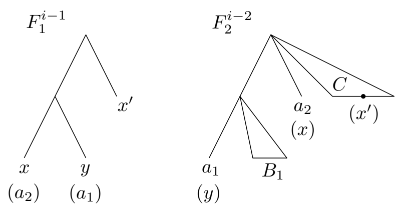


(a) x and x' are siblings in F_2^{i-2}. (b) y and x' are siblings in F_2^{i-2}.

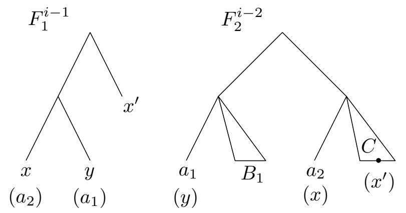
Figure 6.2: Scenario 1.2.2.1 in proof of Lemma 15.

1.2.2.1 Assume a_2 is not a child of the minimal LCA l of a_1 and a_2 in F_2^{i-2}. First consider the case that x and x' are siblings in F_2^{i-2} already, and assume w.l.o.g. that $y = a_1$ and $x = a_2$. This is shown in Figure 6.2(a). Since x and x' are siblings in F_2^{i-2}, x' has to be in B_2. Thus in F_2^{i-2}, the common parent in F_x of x' and y is a proper ancestor of the common parent in F_y of x and x'. Thus, $d_2(y) < d_2(x)$, which means we would have chosen invocation $\textsc{Maaf}(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1)$ instead of invocation $\textsc{Maaf}(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1)$ to be on the path to F'_C, a contradiction. Now consider the case that e_y is the last pendant edge of the path from x to x' in F_2^{i-2}. This implies that either y and x or y and x' are siblings in F_2^{i-2}. The first topology is impossible because there are at least two pendant edges e_{B_1} and e_{B_2} on the path from y to x. The second topology, shown in Figure 6.2(b), is also impossible. The reason is that, after cutting e_y, the path from x to x' would still have a pendant node in B_2, a contradiction because e_y is the last pendant edge of this path we cut and x to x' are siblings after cutting this edge.

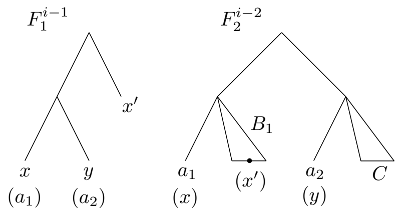
1.2.2.2 The last case is that a_2 is a child of the minimal LCA l of a_1 and a_2 in F_2^{i-2}. In this case, let a'_1 be the child of l that is an ancestor of a_1, and let C be the set



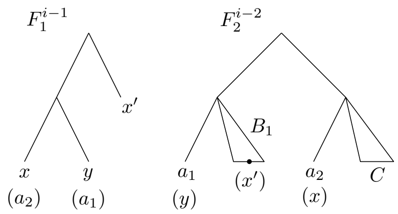
(a) x and x' are siblings in F_2^{i-2} when $x = a_2$.



(b) The resolution to compare $d_2(y)$ and $d_2(x)$ in Scenario 1.2.2.2.



(c) x and x' are siblings in F_2^{i-2} when $y = a_2$.



(d) e_y is the last pendant edge on path from x to x' in F_2^{i-2}.

Figure 6.3: Scenario 1.2.2.2 in proof of Lemma 15.

of children of l other than a_1' and a_2. First consider the case that x and x' are siblings in F_2^{i-2} already. If $y = a_1$, then $x = a_2$ (see Figure 6.3(a)). In this case, x' has to be in C. By the rules for choosing the path to F_C', we compare $d_2(y)$ and $d_2(x)$ in Figure 6.3(b). Since in F_2^{i-2}, the common parent in F_x of x' and y is a proper ancestor of the common parent in F_y of x and x', $d_2(y) < d_2(x)$. Thus, we would have chosen invocation $\text{MAAF}(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1)$ instead of invocation $\text{MAAF}(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1)$ to be on the path to F_C', a contradiction. If $y = a_2$ and $x = a_1$ (see Figure 6.3(c)), then x' has to be in B_1. Thus, in F_2^{i-2}, the common parent in F_x of x' and y is a proper ancestor of the common parent in F_y of x and x'. Thus, $d_2(y) < d_2(x)$, which means we would have chosen invocation $\text{MAAF}(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1)$ instead of invocation $\text{MAAF}(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1)$ to be on the path to F_C', a contradiction. Now consider the case that e_y is the last pendant edge of the path from x to x' in F_2^{i-2}. If $y = a_1$ and $x = a_2$ (see Figure 6.3(d)), then $B_1 = \{x'\}$ to make e_y the last pendant edge of the path from x to x' in F_2^{i-2}.

However, cutting y tags y with "T_2" in F_2 and therefore marks x', contradicting our assumption that both x and x' are unmarked. If $y = a_2$ and $x = a_1$, there is at least one pendant edge e_{B_1} on the path from x to y and e_y cannot be the last pendant edge of the path from x to x' in F_2^{i-2}. Therefore, this case is also impossible.

Scenario 2: If e_y is cut from F_2^{i-1}, y cannot be tagged with "T_2".

2.1 If $y \in \{a_1, a_2\}$, y is tagged with "T_2", a contradiction.

2.2 If $y \in \{B_1, B_2\}$, the sibling of y in F_2^{i-1} is a_1 or a_2. Cutting the sibling of y must be viable as it is x. However, according to the rules we use to find F'_C, we do not choose $\textsc{Maaf}(F_1^{i-1} \div \{e_{B_1}\}, F_2^{i-1} \div \{e_{B_1}\}, k-1)$ or $\textsc{Maaf}(F_1^{i-1} \div \{e_{B_2}\}, F_2^{i-1} \div \{e_{B_2}\}, k-1)$ to be on the path to F'_C unless neither $\textsc{Maaf}(F_1^{i-1} \div \{e_{a_1}\}, F_2^{i-1} \div \{e_{a_1}\}, k-1)$ nor $\textsc{Maaf}(F_1^{i-1} \div \{e_{a_2}\}, F_2^{i-1} \div \{e_{a_2}\}, k-1)$ is viable, a contradiction. □

6.2.2 Stage Two

In this stage, we prove that if our MAAF algorithm can find an MAAF of T'_1 and T'_2 by only fixing marked potential exit nodes, then our MAAF algorithm can also find an MAAF of T_1 and T_2 by only fixing marked potential exit nodes. In other words, we prove that $\textsc{Maaf}(T_1, T_2, k_p)$ returns "Yes" if $hyb(T_1, T_2) \leq k_p$.

As a reminder, T'_1 is defined as a binary resolution of T_1 and this resolution is based on an arbitrary MAAF F_{AA} of (T_1, T_2). Formally, let F_{AA} be an MAAF of (T_1, T_2), and let $(F_{AA})_b$ be a binary resolution of F_{AA}. Then T'_1 is a binary resolution of T_1 with the following property: For every maximal set S of at least two siblings in T_1 that belong to the same subtree $T_1(X^i)$, there exists a node in T'_1 whose set of descendant leaves is exactly the set of descendant leaves of this set of siblings. $(F_{AA})_b$ is an AF of (T'_1, T_2) and is acyclic because the construction of T'_1 from T_1 does not create any ancestry relationships of components of $(F_{AA})_b$ in T_1' that are not present between the corresponding components of F_{AA} in T_1. $(F_{AA})_b$ is also an MAAF of (T_1', T_2) because F_{AA} is an MAAF of (T_1, T_2).

Lemma 16. *Any MAAF of* (T_1', T_2) *is also an MAAF of* (T_1, T_2).

Proof. Consider an MAAF F_{AA}' of (T_1', T_2). Since T_1' is a resolution of T_1, F_{AA}' is an AF of (T_1, T_2). F_{AA}' must be an AAF of (T_1, T_2) because otherwise the cycle graph $G_{F_{AA}'}$ of F_{AA}' w.r.t. (T_1, T_2) has a cycle in it, which implies that the cycle graph $G'_{F_{AA}'}$ of F_{AA}' w.r.t. (T_1', T_2) also contains a cycle. F_{AA}' has the same number of components as $(F_{AA})_b$ because both of them are MAAFs of (T_1', T_2). Since $(F_{AA})_b$ is an MAAF of (T_1, T_2) and F_{AA}' has no more components than $(F_{AA})_b$, then F_{AA}' is also an MAAF of (T_1, T_2). $\square$

The roadmap of our proof that Maaf(T_1, T_2, k_p) returns "Yes" if $hyb(T_1, T_2) \leq k_p$ is shown in Figure 6.4. Let F_C' be the canonical AF found by the branching phase of Maaf(T_1', T_2, k_p). Clearly, F_C' is an AF of (T_1, T_2). Let F_C be the result of collapsing F_C' as much as possible as long as F_C is an AF of (T_1, T_2). We proved that F_C' can be refined to an MAAF of (T_1', T_2) by fixing marked potential exit nodes. We need to prove that the branching phase of Maaf(T_1, T_2, k_p) finds F_C and that F_C can be refined to an MAAF of (T_1, T_2) by fixing marked potential exit nodes.

$$\begin{tikzcd}
 & (T_1, T_2) \arrow[r, "T_1' \text{ is a binary resolution of } T_1", "T_2' \text{ is just } T_2"'] \arrow[d, dashed] & (T_1', T_2') \arrow[d] & \\
F'' & F_C \arrow[l, "Refine"'] & F_C' \arrow[l, "\text{Collapse as much as possible}"'] \arrow[r, "Refine"] & F'
\end{tikzcd}$$

Figure 6.4: Roadmap of the correctness proof.

Theorem 3. *If F'_C, the canonical AF found by* $\textsc{Maaf}$*(T'_1, T_2, k_p), can be refined to an MAAF F' of (T'_1, T_2) by fixing marked potential exit nodes, then one of the AFs found by* $\textsc{Maaf}$*(T_1, T_2, k_p) can also be refined to an MAAF F'' of (T_1, T_2) by fixing marked potential exit nodes.*

Proof. We divide this proof into three claims we prove separately.

Claim 1. *F_C is one of the AFs found by the branching phase of* $\textsc{Maaf}$*(T_1, T_2, k_p).*

Claim 2. *The roots of F_C and F'_C have the same tags.*

Claim 3. *Let V' be a subset of marked potential exit nodes in F'_C that gives us an MAAF F' of (T'_1, T_2). Then there exists a subset V of marked potential exit nodes in F_C that also gives us an AAF F'' of (T_1, T_2) and fixing it cuts no more edges than fixing V'.*

With the first claim, if F_C can be refined to an MAAF of (T_1, T_2) by fixing marked potential exit nodes, then the theorem holds. The second claim will be used to support the proof of the third claim. By the third claim, we know that F'' is an AAF of (T_1, T_2) and F'' has no more components than F'. Since F' is an MAAF of (T_1, T_2) (because of Lemma 16), F'' is an MAAF of (T_1, T_2). Therefore, this theorem holds. $\square$

Let's prove the three claims one by one.

Proof of Claim 1. According to the correctness proof of the branching phase (Theorem 1, Lemma 3, and Lemma 4), for any agreement forest F of the input two trees, we can always make progress to F by cutting one of $e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}$. Thus, $\textsc{Maaf}(T_1, T_2, k_p)$ finds all AFs of T_1 and T_2 with no more than $k_p + 1$ components. Since F' has no more than $k_p + 1$ components, so does F'_C because F' is a refined forest of F'_C. Since collapsing F'_C to F_C doesn't change the number of components, F_C also has no more than $k_p + 1$ components. Therefore, F_C is found by the branching phase of $\textsc{Maaf}(T_1, T_2, k_p)$. $\square$

Proof of Claim 2. To prove Claim 2, we modify the procedure $\textsc{Maaf}(T_1', T_2, k_p)$ slightly without changing its result. Each internal node v in T_1 gives rise to a sibling group G_v that is considered in procedure $\textsc{Maaf}(T_1, T_2, k_p)$. To "simulate" such a sibling group in $\textsc{Maaf}(T_1', T_2, k_p)$, let v' be the node in T_1' that has the same set of descendant leaves as v does in T_1. Then each sibling group G_v in T_1 corresponds to a subtree $T_1'^{v'}$ in T_1'. The modification to procedure $\textsc{Maaf}(T_1', T_2, k_p)$ is the following: each time we need to pick a sibling pair, we pick a pair in the previous subtree $T_1'^{v'}$ as long as v' is not a leaf. Originally, the order of picking sibling groups (for T_1', it's picking sibling pairs because T_1' is binary) is arbitrary. Thus, our modification has no impact on the result of procedure $\textsc{Maaf}(T_1', T_2, k_p)$.

Let $\mathcal{I}$ be the invocation path in the recursion tree of $\textsc{Maaf}(T_1, T_2, k_p)$ that leads to F_C. Let $\mathcal{I}'$ be the invocation path in the recursion tree of $\textsc{Maaf}(T_1', T_2, k_p)$ that leads to F_C'. We say that two edge cuts are the same if (1) the two cut edges have the same set of descendant leaves and (2) the two cut edges are either both cut as a-type edges (e_{a_1} or e_{a_2}) or they are both cut as B-type edges (e_{B_1} or e_{B_2}). We prove that the edge cuts performed along $\mathcal{I}$ and $\mathcal{I}'$ are the same though these cuts may be made in different orders.

Assume that until now, $\mathcal{I}$ and $\mathcal{I}'$ have made the same cuts (possibly in a different order). Let G_v be the sibling group considered by $\mathcal{I}$ at this moment. There is a corresponding binary subtree $T_1'^{v'}$ that can be chosen by $\mathcal{I}'$. Let E' be the edge set cut by $\mathcal{I}'$ while it processes the sibling pairs in $T_1'^{v'}$. We prove that, every invocation in $\mathcal{I}$ that considers the sibling group G_v always has the option to cut an edge in E' as one of its 4 choices. Thus, once we are done processing G_v, $\mathcal{I}$ and $\mathcal{I}'$ will once again reach a state where they've made the same edge cuts.

For any pair (a_i, a_j) in sibling group G_v considered by an invocation in $\mathcal{I}$, there are three difference cases:

(i) If $e_{a_i} \in E'$, we cut e_{a_i} as the next edge on $\mathcal{I}$.

(ii) If $e_{a_j} \in E'$, we cut e_{a_j} as the next edge on $\mathcal{I}$.

(iii) If $e_{a_i} \notin E'$ and $e_{a_j} \notin E'$, we cut e_{B_i} unless a_i is a child of the LCA of a_i and a_j. In this case, we cut e_{B_j}.

In the first two cases, we clearly cut an edge in E'. In the third case, since $e_{a_i} \notin E'$ and $e_{a_j} \notin E'$, a_i and a_j must become siblings in T_2 while processing subtree $T_1'^{v'}$ by $\mathcal{I}$. To merge a_i and a_j, all pendant edges on the path from a_i to a_j must be cut. In other words, E' includes both e_{B_i} and e_{B_j}, and cutting one of them thus cuts an edge in E'. Therefore, after processing sibling group G_v, $\mathcal{I}$ and $\mathcal{I}'$ have made the same cuts.

Since this invariant holds trivially at the beginning of $\mathcal{I}$ and $\mathcal{I}'$, because neither has made any cuts. Thus, the inductive step above shows that when $\mathcal{I}$ and $\mathcal{I}'$ reach F_C and F'_C respectively, they have made exactly the same cuts. This implies that F_C and F'_C have the same roots and each of them has the same tag in F_C and F'_C. □

Proof of Claim 3. Since F_C is the result of collapsing F'_C, there is a one-to-one mapping between roots of F_C and F'_C. Any two roots that are mapped to each other have the same set of leaves. Let r_1 be an arbitrary root in F_C, and r'_1 be the counterpart in F'_C. r_1 and r'_1 have the same tag, according to the second claim. Let the tag be "T_i", $i \in \{1, 2\}$. r_1 marks a potential exit node u in F_C, and r'_1 marks a potential exit node u' in F'_C. We call u the surrogate of u' in F_C. Let r_2 and r'_2 be the roots of the components containing u and u' in F_C and F'_C, respectively. These definitions are illustrated in Figure 6.5.

Let u'' be the node in F'_C that is the LCA of all descendant leaves of u in F_C. Then we make the following three claims, which we prove separately below.

Claim 4. *u'' and u' are in the same component of F'_C.*

Claim 5. *u'' is the same node as u' or u'' is an ancestor of u'.*

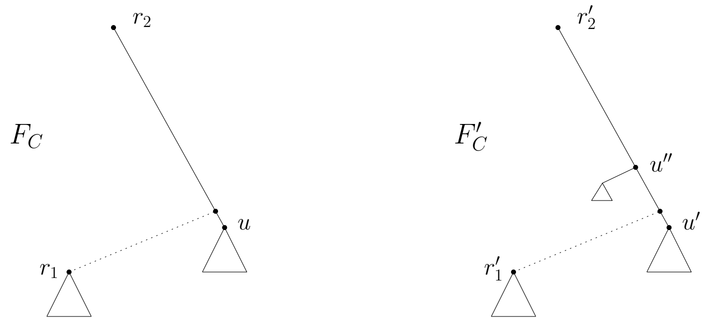


Figure 6.5: Illustration for the paths to cut when fixing u and u'.

Claim 6. *There are no more edges on the path from u to r_2 than on the path from u'' to r_2'.*

Claim 4 will support the proof of Claim 5. Claim 5 and 6 imply that fixing u cuts no more edges than fixing u'. Next, we construct a subset V of marked potential exit nodes in F_C that can be fixed to produce an AAF F'' of (T_1, T_2). Let V' be a subset of marked potential exit nodes in F_C' that can be fixed to produce an MAAF F' of (T_1', T_2'). We choose V to be the set of surrogates of nodes in V'. V contains no more nodes than V'. Therefore, by Claim 5 and 6, fixing the nodes in V cuts no more edges than fixing the nodes in V'. To finish the proof, we need to show that fixing the nodes in V gives an AAF of T_1 and T_2.

Let C_i and C_i', for $1 \leq i \leq m$, be the components of F_C and F_C', respectively. C_i and C_i' have the same set of leaves. We also use C_i and C_i' to denote the nodes in G_{F_C} and $G_{F_C'}$ corresponding to C_i and C_i'. Let $G_{F''}$ be the cycle graph of $F'' = F_C \div V$ (as $F_C \div E$ is defined as the resulting forest after cutting edge set E, similarly $F_C \div V$ is defined as the resulting forest after fixing V in F_C) and let $\bar{C}_i$, for $1 \leq i \leq m''$, be the components of F''. To prove that fixing V gives us an AAF F'' of (T_1, T_2), the following three claims are necessary. The flow of proof is shown in Figure 6.6.

Claim 7. *Let $\bar{C}_i$ and $\bar{C}_j$ be two components in F''. If $\bar{C}_i$ is an ancestor of $\bar{C}_j$ in T_1 or T_2, i.e., $G_{F''}$ contains the edge $(\bar{C}_i, \bar{C}_j)$, then there exist components $C_i \supseteq \bar{C}_i$ and $C_j \supseteq \bar{C}_j$ such that (1) C_i is an ancestor of C_j in T_1 or T_2, or (2) $C_i = C_j$ (i.e., $\bar{C}_i$*

and $\bar{C}_j$ are produced by breaking up a component of F_C).

Claim 8. *Let F_A be an AF of (T_1, T_2), and let F'_A be an AF of (T'_1, T'_2) such that F'_A is a binary resolution of F_A. Then F_A's cycle graph G_{F_A} is a subgraph of F'_A's cycle graph $G_{F'_A}$ (considering the node $C_i \in G_{F_A}$ and its corresponding node $C'_i \in G'_{F_A}$ to be the same node).*

Claim 9. *Let C_i and C_j be components of F_C such that C_i is an ancestor of C_j in T_1 or T_2, C'_i is an ancestor of C'_j in T'_1 or T'_2, and C'_i has a node $u' \in V'$ such that fixing u' will cut all edges in C'_i that are ancestors of C'_j. Let u be u''s surrogate in C_i. Any edge e in C_i that is an ancestor of C_j in T_1 or T_2 must be an ancestor of u, that is it is cut by fixing u.*

$$\begin{tikzcd}
\text{A cycle } \bar{O} \text{ in } G_{F''} \arrow[r] & \text{A cycle } O \text{ in } G_{F_C} \arrow[r] & \text{A cycle } O' \text{ in } G_{F'_C} \arrow[d] \\
\begin{array}{c}\text{Cycle } \bar{O} \text{ cannot exist} \\ \text{in } G_{F''}\end{array} & \begin{array}{c}\text{Cycle } O \text{ gets broken} \\ \text{after fixing } V\end{array} \arrow[l] & \begin{array}{c}\text{Cycle } O' \text{ gets broken} \\ \text{after fixing } V'\end{array} \arrow[l]
\end{tikzcd}$$

Figure 6.6: Roadmap of proving that F'' is acyclic.

Now assume for the sake of contradiction that there exists a cycle $\bar{O}$ in the cycle graph $G_{F''}$. Let $\bar{C}_1, \bar{C}_2, ..., \bar{C}_p$ be the components that compose $\bar{O}$. By Claim 7, there is a cycle O in G_{F_C} that consists of components $C_1, C_2, ..., C_p$, where some pairs of consecutive components may be the same. By Claim 8, for every edge (C_1, C_2) in G_{F_C}, there is an edge (C'_1, C'_2) in $G_{F'_C}$. Thus, there is a cycle O' in $G_{F'_C}$ that corresponds to the cycle O in G_{F_C}. Since $F' = F'_C \div V'$ is acyclic, fixing V' breaks O'. Then among the components composing cycle O', there exists at least one pair of components C'_i and C'_{i+1} such that (1) C'_i is an ancestor of C'_{i+1} in, say T_1, and (2) C'_i has a node $u' \in V'$ such that fixing u' cuts all edges in C'_i that are ancestors of C'_{i+1}. By Claim 9, fixing u''s surrogate u in C_i also cuts all edges in C_i that are ancestors of C_{i+1}. This implies that no component $\bar{C}_i \subseteq C_i$ of F'' can be an ancestor of C_{i+1} after fixing u. Thus, we cannot find $\bar{C}_i \subseteq C_i$ and $\bar{C}_{i+1} \subseteq C_{i+1}$ in F'' such that $\bar{C}_i$ is an ancestor of $\bar{C}_{i+1}$. This means that the edge $(\bar{C}_i, \bar{C}_{i+1})$ doesn't exist, which contradicts our initial assumption that $\bar{C}_1, \bar{C}_2, ..., \bar{C}_p$ forms a cycle. Therefore, the cycle graph $G_{F''}$ is acyclic and F'' is an AAF of (T_1, T_2). $\square$

Proof of Claim 4. Assume for the sake of contradiction that u'' and u' are in different components of F'_C. Let C_1 and C'_1 be the components with roots r_1 and r'_1, respectively, let C'_2 be the component of F'_C that contains u', and let C_3 be the component of F_C that contains u. Let C'_3 be the counterpart of C_3 in F'_C and let C_2 be the counterpart of C'_2 in F_C. We denote the roots of C_2, C_3, C'_2, and C'_3 by r_2, r_3, r'_2, and r'_3, respectively. Then $C'_3 \neq C'_2$. Since r_1 marks u, C_3 is an ancestor component of C_1, which implies that C'_3 is also an ancestor component of C'_1. Thus, since $C'_2 \neq C'_3$, C'_2 contains an edge on the path P' from C'_1 to C'_3 (see Figure 6.7(a)). Since $u \in C_3$, C_2 does not contain an edge on the path P from C_1 to C_3. Now, if C_2 does not even have a node on P, then no matter how we resolve C_2 to obtain C'_2, C'_2 cannot have an edge on P', a contradiction. If C_2 has a node but no edge on P, the node must be the root r_2 of C_2. Let e be the edge on P that has r_2 as its top endpoint and thus does not belong to C_2. Since r_2 is the root of C_2, r_2 has at least two child edges e_1 and e_2 that do belong to C_2. Thus, by the definition of T'_1, the set of descendant leaves of r'_2 in T'_1 is the set of descendant leaves of all child edges of r_2 that belong to C_2. In particular, no descendant leaf of edge e is a descendant leaf of r'_2 and, thus, no descendant leaf of r'_1 is a descendant leaf of r'_2 (see Figure 6.7(b)). This contradicts the assumption that C'_2 contains an ancestor edge of r'_1. $\square$

Proof of Claim 5. By Claim 4, u'' and u' both belong to the component C'_2 with root r'_2. Assume for the sake of contradiction that u'' is not an ancestor of u'. Then there exists at least one leaf $x \in C'_2$ that is a descendant of u' but not a descendant

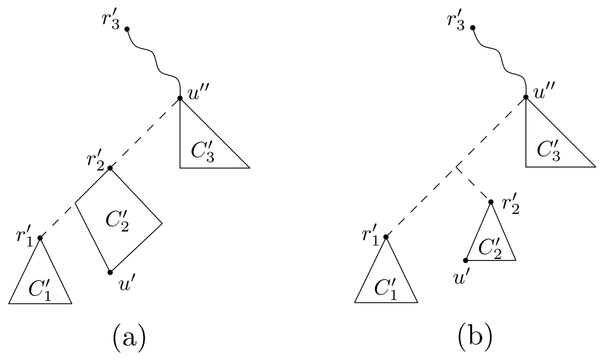


Figure 6.7: (a) The structure of F'_C if u'' and u' are in different components of F'_C. (b) The structure of F'_C based on the construction of T'_1.

of u'' (see Figure 6.8). Since u and u'' have the same set of descendant leaves, x is not a descendant of u either. Let l be the lowest ancestor of r_1 in T_1 that belongs to C_2, and let l' be the lowest ancestor of r'_1 in T'_1 that belongs to C'_2. Since l' is an ancestor of u', x is a descendant leaf of l'. In C_2, x is a descendant leaf of l because C'_2 is a binary resolution of C_2 and T'_1 is a binary resolution of T_1. This implies that there exists a proper ancestor of u that is a descendant of l and has two children in C_2, one an ancestor of x, the other an ancestor of u. This, however, is a contradiction because r_1 would mark this ancestor instead of u in C_2. □

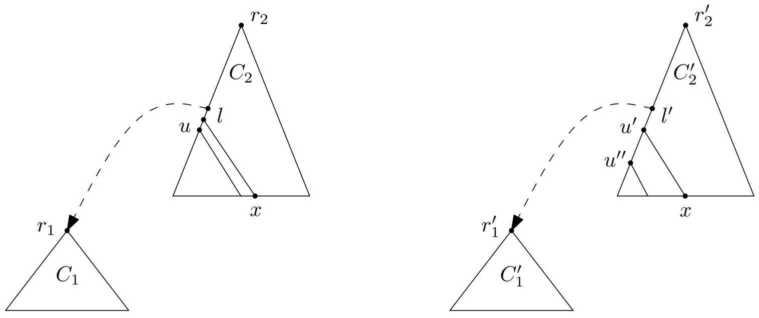


Figure 6.8: The structure of F_C and F'_C if u'' is a proper descendant of u' in component C'_2.

Proof of Claim 6. This is trivially true because the component rooted at r'_2 is a binary resolution of the component rooted at r_2. More precisely, the path from u''

to r'_2 contains all edges on the path from u to r_2, but may contains additional edges introduced by resolving mutlfurcations on this path. □

Proof of Claim 7. Since F'' is a refined forest of F_C, F'' is a subgraph of F_C. Thus, we can always find components $C_i \supseteq \bar{C}_i$ and $C_j \supseteq \bar{C}_j$. If $C_i = C_j$, the claim holds. If $C_i \neq C_j$ and w.l.o.g. $\bar{C}_i$ is an ancestor of $\bar{C}_j$ in T_1, we prove that C_i is an ancestor of C_j in T_1. Since $\bar{C}_i$ is an ancestor of $\bar{C}_j$, there exists an edge e in $\bar{C}_i$ that is an ancestor of $\bar{C}_j$. C_i, as a supergraph of $\bar{C}_i$, must also contain this edge e. Moreover, the edge e must be an ancestor of C_j, otherwise e is part of C_j, which is impossible because $Ci \neq C_j$ and two components of F_C do not overlap. In summary, C_i contains an edge e that is an ancestor of C_j. Therefore, C_i is an ancestor of C_j. □

Proof of Claim 8. Let C_i and C'_i, for $1 \leq i \leq m$, be the components in F_A and F'_A, respectively. Since F'_A is a binary resolution of F_A, they have the same set of components if we consider C_i and C'_i to be the same component for all $1 \leq i \leq m$. Thus, G_{F_A} and $G_{F'_A}$ have the same vertex set. To prove that G_{F_A} is a subgraph of $G_{F'_A}$, it is sufficient to prove that for any edge (C_i, C_j) in G_{F_A}, there exists an edge (C'_i, C'_j) in $G_{F'_A}$. If (C_i, C_j) is a T_1-hybrid edge, then there exists an edge e in C_i such that e is an ancestor edge of C_j in T_1. As a binary resolution of T_1, T'_1 has all the edges that T_1 has. Thus, e also belongs to C'_i and is an ancestor edge of C'_j. This implies that $G_{F'_A}$ contains the edge (C'_i, C'_j). If (C_i, C_j) is a T_2-hybrid edge, the claim trivially holds because G_{F_A} and $G_{F'_A}$ are based on the same T_2. Therefore, the claim is true. □

Proof of Claim 9. Assume w.l.o.g. that C_i is an ancestor of C_j in T_1 and, thus, that C'_i is an ancestor of C'_j in T'_1. Since $u' \in V'$, there exists a component C'_k whose root r_k has a tag that causes u' to be marked. Let the tag of r_k be "T_h" and let l'_k be the lowest ancestor of r_k in T'_h that belongs to C'_i. Then u' is a descendant of l'_k. Similarly, let l'_j be the lowest ancestor of r_j in T'_1 that belongs to C'_i. Since fixing u'

cuts all ancestor edges of C'_j in C'_i, l'_j is an ancestor of u'. Now let l_j and l_k be the nodes in C_i with the same set of descendant leaves as l'_j and l'_k. Then l_k is the lowest ancestor of r_k in T_h that belongs to C_i and l_j is the lowest ancestor of r_j in T_1 that belongs to C_i. The surrogate u of u' in C_i is the node in C_i marked by r_k and thus is a descendant of l_k. If l_j is an ancestor of u, then fixing u cuts all ancestor edges of C_j in C_i, so the claim holds. Assume therefore that l_j is not an ancestor of u, which is possible only if l_j is a proper descendant of l_k. Since u is a descendant of l_k, we distinguish two cases:

If u is a proper ancestor of l_j, then let u'' be the node in C'_i that has the same set of descendant leaves as u and let U be the subtree of C'_i that has u'' as its root and contains all nodes introduced by resolving the multifurcation at u into bifurcations. (If u is bifurcating or a leaf, then U has a single node, namely u''.) Since C'_i is a binary resolution of C_i, u'' and U exist. For the same reason, since u is a proper ancestor of l_j and a descendant of l_k, u'' is a proper ancestor of l'_j and a proper descendant of l'_k. Since u is marked by r_k, u has at least two children with descendant leaves in C_i. This implies that there exists a node u''' in U that has two children with descendant leaves in C'_i. Since u is a proper ancestor of l_j, the highest such node u''' is a proper ancestor of l'_j and thus of u' because u' is a descendant of l'_j. This contradicts the assumption that r_k marks u' in C'_i.

If u is neither a descendant nor an ancestor of l_j, then because u and l_j are descendants of l_k, there exists a descendant x of l_k that is the LCA of u and l_j in C_i. Since u and l_j both have descendant leaves in C_i and belong to different subtrees of x, x has two children with descendant leaves in C_i. This contradicts the assumption that u is the node in C_i marked by r_k because x is a proper ancestor of u and every proper ancestor of u that belongs to the path from l_k to u can have only one child with descendant leaves in C_i for r_k to mark u. $\square$

By Theorem 3, the MAAF algorithm returns "Yes" if $hyb(T_1, T_2) \leq k_p$, and it returns "No" if $hyb(T_1, T_2) > k_p$. Thus, our algorithm is correct.

6.3 Complexity Analysis

In the branching phase, each invocation that is not a leaf invocation has 4 child invocations and the height of the recursion tree is no more than k_p. Thus, there are $O(4^{k_p})$ invocations in the branching phase, with total cost $O(4^{k_p}n)$. In the refinement phase, we mark only one potential exit node per root (other than ρ). Thus, there are at most k_p marked potential exit nodes, and $\textsc{Refine}((T_1)_m, (T_2)_m, F_m, k_p)$ takes $O(2^{k_p}n)$ time to test whether fixing any subset of these marked potential exit nodes yields an AAF of T_1 and T_2 with at most $k_p + 1$ components. Thus, the total running time of the algorithm is $O(4^{k_p}(n + 2^{k_p}n)) = O(8^{k_p}n)$.

Theorem 4. *For two multifurcating rooted trees T_1 and T_2 and a parameter k_p, it takes $O(8^{k_p}n)$ time to decide whether $\tilde{e}(T_1, T_2, T_2) \leq k_p$.*

6.4 Tightening the Complexity Analysis

Our improved MAAF algorithm consists of two phases. The first phase uses the 4-way branching to find a set of agreement forests with marked potential exit nodes such that at least one of these AFs F can be refined to an MAAF F' by fixing a subset of the marked potential exit nodes in F. The second phase then fixes every subset of the marked potential exit nodes for each AF F found in the first phase. Let k' be the number of edges we cut in the first phase. Then there are k' marked potential exit nodes and $2^{k'}$ subsets of marked potential exit nodes to check. If k' is small, the refinement phase's cost of $O(2^{k'}n)$ is substantially smaller than the naive bound of $O(2^{k_p}n)$ that we used so far. If k' is large, then the number k'' of edges we can still cut in the second phase is small. Indeed, cutting k'' edges in the refinement phase increases the number of components by k'', so $k'' = k_p - k'$. Since fixing one potential exit node cuts at least one edge, the number of potential exit nodes we can fix in the refinement phase is no more than $k'' = k_p - k'$. Since there are k' marked potential exit nodes to choose from, this reduces the running time of the refinement phase on F to $O(\Sigma_{j=0}^{k''}\binom{k'}{j}n)$. Since k'' is substantially less than k_p, this sum is significantly less than $O(2^{k'}n) = O(2^{k_p}n)$. Thus, no matter whether k' is small or large, it should be possible to achieve a running time better than $O(2^{k_p}n)$ for the refinement phase.

To achieve this improvement, the only change to our MAAF algorithm in Section 6.1 is to inspect all subsets of at most k'' marked potential exit nodes in the refinement phase, where $k'' = min(k', k_p - k')$. The new refinement subroutine is shown in Algorithm 2.

Algorithm 2 REFINE(T_1, T_2, F, k_p)

```
Build the expanded cycle graph $G^*_F$ from $F$, $T_1$ and $T_2$, mark half of the potential
exit nodes in $G^*_F$ according to the tags of component roots, and use $V$ to denote
the set of marked potential exit nodes;
for each subset $V_{sub}$ of $V$ of size at most $k''$ do
    Fix every node $v$ in $V_{sub}$;
    if the number of components in the resulting forest $F'$ is no more than $k_p + 1$
    and $G^*_{F'}$ is acyclic then
        Return "Yes";
    end if
end for
Return "No";
```

To analyze the running time of the resulting MAAF algorithm, we split each refinement invocation into several refinement steps. A refinement invocation that inspects all subsets of at most k'' marked potential exit nodes is divided into $k'' + 1$ refinement steps: for $0 \leq j \leq k''$, the jth refinement step inspects all subsets of exactly j marked potential exit nodes. Its running time is therefore $O(\binom{k'}{j}n)$. Now we partition all the refinement steps invoked for the different AFs found during the branching phase into $k_p + 1$ groups. For $0 \leq h \leq k_p$, the hth group contains a refinement step applied to an AF F if the number k' of edges cut to obtain F and the number j of marked potential exit nodes the refinement step inspects satisfy $k' + j = h$. We prove that the running time of all refinement steps in the hth group is $O(4.83^h n)$. Hence, the total running time of all refinement steps in all groups is $O(\Sigma_{h=0}^{k_p} 4.83^h n) = O(4.83^{k_p} n)$, which dominates the $O(4^{k_p} n)$ time bound of the branching phase. Therefore, the running time of the entire MAAF algorithm is $O(4.83^{k_p} n)$.

To obtain the time bound of all refinement steps in the hth group, we first consider the tree of recursive calls made in the branching phase. Let $\textsc{Maaf}(F_1, F_2, k'')$ be an invocation in the recursion tree, and let k' be the number of edges cut before this invocation. Hence, $k' + k'' = k_p$. Since every refinement step in the hth group satisfies $k' + j = h$ and thus $k' \leq h$, refinement steps in the hth group can be invoked only for agreement forests that can be produced by cutting at most h edges in T_2. Thus, we can restrict our attention to the subtree of $\textsc{Maaf}(F_1, F_2, k'')$ such that F_2 can be obtained from T_2 by cutting at most h edges, which means $k'' = k_p - k' \geq k_p - h = d$. Now we bound the running time of the refinement steps in the hth group in two steps.

First, we construct a maximized recursion tree without refinement steps. For each invocation $\textsc{Maaf}(F_1, F_2, k'')$, according to the branching algorithm in Section 3.2, it has a subtree of $\Theta(4^{k''-d})$ recursive calls below it. The size of the entire recursion tree is thus $O(4^{k_p-d}) = O(4^h)$ because the top invocation is $\textsc{Maaf}(T_1, T_2, k_p)$. Second, we choose a subset of recursive calls in this tree which are not made because the refinement step is invoked instead. We charge the cost of the refinement step equally to the nodes in the recursion subtree it replaces. To be specific, for each invocation with parameter k'', we replace its subtree of $\Theta(4^{k''-d})$ recursive calls with a single refinement step of cost $O(\binom{k'}{j} n)$, where $k'' - d = k'' - (k' + k'' - h) = h - k' = j$. By charging the cost of this refinement step equally to the nodes in the replaced subtree, each node pays a cost of $O(\binom{k'}{j} n/(4^{k''-d})) = O(\binom{k'}{j} n/(4^j))$. Hence, the total running time of all refinement steps in the hth group is bounded by the sum of the charges of all nodes replaced in the size $O(4^h)$ recursion tree. Since at most $O(4^h)$ nodes can be replaced, the total cost is therefore

$$O\left(4^h \cdot \frac{\binom{k'}{j}n}{4^j}\right) = O\left(4^{k'} \cdot \tbinom{k'}{j}n\right) \tag{6.1}$$

To bound Expression (6.1), we consider three cases: $k' \leq h/2$, $h/2 < k' < h$, and $k' = h$. In the first case, $\binom{k'}{j}$ is bounded by $2^{k'}$, so $4^{k'} \cdot \binom{k'}{j}$ is bounded by $8^{k'} \leq 8^{h/2} \leq 2.83^h$. This implies that Expression (6.1) is bounded by $O(2.83^h n)$. In the third case, $k' = h$ implies that $j = 0$. Thus $4^{k'} \cdot \binom{k'}{j} = 4^h \cdot \binom{h}{0} = 4^h$, that is, Expression (6.1) is bounded by $O(4^h n)$. For the case when $h/2 < k' < h$, we make use of the following observation, which is taken from [18].

Observation 6. $\binom{x}{y} = O\left(\left(\frac{x}{y}\right)^y \left(\frac{x}{x-y}\right)^{x-y}\right)$.

Observation 6 allows us to bound Expression (6.1) by

$$O\left(4^{k'} \cdot \left(\frac{k'}{j}\right)^j \left(\frac{k'}{k'-j}\right)^{k'-j} n\right) = O\left(\left(4^\alpha \cdot \left(\frac{\alpha}{1-\alpha}\right)^{1-\alpha} \left(\frac{\alpha}{2\alpha-1}\right)^{2\alpha-1}\right)^h n\right), \tag{6.2}$$

where $\alpha = k'/h$ and hence, $k' = \alpha h$ and $j = (1-\alpha)h$. It remains to find the maximal value of the following function when $1/2 < \alpha < 1$.

$$f(\alpha) = 4^\alpha \cdot \left(\frac{\alpha}{1-\alpha}\right)^{1-\alpha} \left(\frac{\alpha}{2\alpha-1}\right)^{2\alpha-1} \tag{6.3}$$

Taking the derivative and setting it to zero, we obtain that $f(\alpha)$ is maximized when $\alpha = \frac{1}{2} + \frac{1}{2\sqrt{2}} \approx 0.854$, which implies that $f(\alpha) \leq 2(1+\sqrt{2}) < 4.829$. This finishes the proof that the running time of all refinement steps in the hth group is $O(4.83^h n)$. As we argued already, it implies that the running time of the entire MAAF algorithm is $O(4.83^{k_p}n)$. Thus, we have the following theorem.

Theorem 5. *For two multifurcating rooted trees T_1 and T_2 and a parameter k_p, it takes $O(4.83^{k_p}n)$ time to decide whether $\tilde{e}(T_1, T_2, T_2) \leq k_p$.*

Chapter 7

Experimental Evaluation

In this chapter, we demonstrate the practical efficiency of our fixed-paramenter algorithm for computing the hybridization number of two multifurcating trees. The $O(4.83^k n)$ time algorithm described in Chapter 6 was implemented in C++, and benchmarked on a 2.4GHz AMD Opteron processor with 16GB of RAM running Debian GNU/Linux 7. The source code is available upon request. Before the experimental results, Section 7.1 introduces a very important and useful optimization in practice, *cluster reduction*. It often allows an input instance to be divided into smaller instances that can be solved independently. Since these smaller instances often also have much smaller hybridization numbers, it has the potential to speed up the algorithm exponentially. This technique was introduced by Linz and Semple [49]. Section 7.2 discusses several important details of how to implement the $O(4.83^k n)$ MAAF algorithm and proposes another practical optimization that decreases the number of potential exit nodes to fix. Finally, Sections 7.3 and 7.4 provide the experimental evaluation of our algorithm implementation, and discuss the data sets and system details used for this evaluation. These experiments shows the correctness of our implementation and the practical efficiency of our algorithm based on real biological data.

7.1 Cluster Reduction

Cluster reduction partitions two input trees into pairs of subtrees, or clusters, which can be solved independently and reassembled into a full solution for the original input

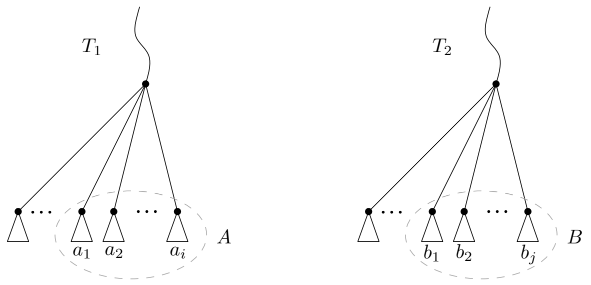


Figure 7.1: A cluster for two multifurcating trees such that group $A = \{a_1, a_2, ..., a_i\}$ and group $B = \{b_1, b_2, ..., b_j\}$ have the same set of labelled descendant leaves.

trees. Cluster reduction has been used to compute weighted MAFs [49], ordinary MAFs [12], and binary hybridization number [43]. Here, we apply it to computing multifurcating hybridization number. In the binary case, a *cluster* of two trees T_1 and T_2 [49] is defined as a pair of subtrees T_1^u and T_2^v, for appropriate nodes u in T_1 and v in T_2 such that both subtrees have the same set of labelled descendant leaves. In the multifurcating case, we define a *cluster* of two trees T_1 and T_2 as a pair of sibling sets $\{a_1, a_2, ..., a_i\}$ and $\{b_1, b_2, ..., b_j\}$, such that both sibling sets have the same set of labelled descendant leaves (shown in Figure 7.1). Since all multifurcations are assumed to be soft, this definition corresponds to finding clusters in appropriate binary resolutions of the input trees. Note that a cluster has at least two leaves. A cluster can be used to speed up the computation of hybridization number because of the following property (one of the reduction rules in [43]).

Lemma 17. *Let A be a cluster of two multifurcating trees T_1 and T_2 and with at least two leaves and define two pairs of new trees as follows:*

(i) *T_1' and T_2' are obtained from T_1 and T_2 by replacing the cluster with a new leaf labelled A.*

(ii) *T_1^A and T_2^A are the subtrees of T_1 and T_2 with leaf set A.*

Then $hyb(T_1, T_2) = hyb(T_1', T_2') + hyb(T_1^A, T_2^A)$.

By Lemma 17, we can compute the hybridization number of T_1 and T_2 by computing the hybridization number of T_1' and T_2', and of T_1^A and T_2^A. Moreover, assuming T_1^A and T_2^A is the minimal cluster such that no other cluster A' has a smaller leaf set than A does, we can take T_1' and T_2' as the new input and apply Lemma 17 to T_1' and T_2' recursively until no further clusters are found. This produces a partition of (T_1, T_2) into a *cluster sequence* of tree pairs $(T_1^{A_1}, T_2^{A_1})$, $(T_1^{A_2}, T_2^{A_2})$, ..., $(T_1^{A_p}, T_2^{A_p})$ such that $hyb(T_1, T_2) = hyb(T_1^{A_1}, T_2^{A_1}) + hyb(T_1^{A_2}, T_2^{A_2}) + ... + hyb(T_1^{A_p}, T_2^{A_p})$. Since our hybridization algorithm's complexity is $O(4.83^k n)$, where k is the hybridization number, cluster reduction may decrease the cost of computing the hybridization number of T_1 and T_2 exponentially if we find a sufficient number of non-trivial clusters. Next, we discuss how to construct such a cluster sequence efficiently.

To maximize the benefit of cluster reduction, T_1 and T_2 should be partitioned into as many clusters as possible. Thus, a cluster sequence of minimal clusters should be constructed. The problem is: "Given two trees T_1' and T_2', how can we find a minimal cluster?" A naive approach is the following: For every node u in T_1' and every node v in T_2', we compare every subset of children of u with every subset of children of v to see whether they have the same set of leaves. Let n be the leaf set size of T_1' and T_2', and d be the maximal degree of all nodes in T_1' and T_2'. Since T_1' and T_2' both have $O(n)$ nodes, each node has $O(2^d)$ subsets of children, and comparing two subsets' leaf sets takes $O(n)$ time, this approach takes $O(n^3 \cdot 2^d \cdot 2^d)$ time. In theory, this is inefficient, especially when d is large. In practice, there are many possible strategies to bound the cost of cluster reduction. The price is that the algorithm may fail to recognize some clusters, but our experiments confirm that the clusters found by our cluster reduction algorithm still yield substantial improvements of the running time.

The first type of cluster our algorithm is able to recognize is a pair of subtrees with the same set of labelled leaves in T_1 and T_2. This is similar to the binary case. A

Algorithm 3 ClusterReduction(T_1, T_2, $Q_{clusters}$)

```
Preprocess $T_1$ and $T_2$ for constant-time LCA queries;
Preprocess $T_1$ and $T_2$ such that every node $u$ has a counter $u.size$ that records the number of descendant leaves under $u$;
Map every node $u$ in $T_1$ and $T_2$ to its counterpart in the other tree, which is the LCA of its descendant leaves;
for each node $u$ in $T_1$ (in reverse level order) do
  for each subset of children of $u$ of size at least 2 do
    Compute the LCA $v$ of these children's mappings in $T_2$;
    if $v.size ==$ sum of the sizes of these children then
      Add $v$ and this set of $u$'s children as a cluster into $Q_{clusters}$;
    end if
  end for
end for
Do the same check to nodes of $T_2$;
```

sequence of such clusters can be constructed in $O(n)$ time with the help of constant-time LCA queries that are borrowed from the binary case [12,50] is to build a mapping between nodes in T_1 and T_2 first. For a node u in T_1, its mapping in T_2 is the LCA of all of u's descendant leaves; the mapping from T_2 to T_1 is defined analogously. If two nodes are mapped to each other, they form a cluster.

The second type of cluster we are able to recognize is a node u in T_1 (T_2) and a subset of children of v in T_2 (T_1) such that the descendant leaves of these children of v are exactly the descendant leaves of u. A sequence of such clusters can be constructed in $O(2^d \cdot n)$ time also with the help of constant-time LCA queries and linear-time preprocessing.

The details of our cluster reduction algorithm are given in Algorithm 3. The procedure ClusterReduction(T_1, T_2, $Q_{clusters}$) finds all the clusters using a linear scan of both T_1 and T_2. The resulting cluster sequence is stored in $Q_{clusters}$. The hybridization number of T_1 and T_2 can be computed by summing the hybridization numbers of all tree pairs in $Q_{clusters}$.

7.2 Efficient Implementation of the MAAF Algorithm

This section reviews the main steps of our MAAF algorithm described in Chapter 6 (see Figure 7.2), and explains some of the implementation details. A simple optimization is proposed at the end of this section. The implementation should consist of $k+1$ iterations, where k is the hybridization number of the two given multifurcating rooted X-trees T_1 and T_2. Each iteration is given a parameter k_p, starting from $k_p = 0$, and answers the question: "Is $hyb(T_1, T_2) \leq k_p$?"

Let procedure $\textsc{Maaf}(T_1, T_2, k_p)$ be the implementation of an iteration with parameter k_p. It consists of two main phases and one intermediate phase.

- The first phase is the 4-way branching phase, which is described in detail in Chapter 3. It cuts edges from T_2 and makes 4 recursive invocations unless the given forest F_2 is an AF of T_1 and T_2. It guarantees that at least one of the AFs found can be refined to an MAAF of T_1 and T_2.

- For each AF F found in the first phase, it enters the intermediate phase in preparation for the second phase. The intermediate phase is described in Chapter 4. F is collapsed to F_m to restore useful multifurcations. T_1 and T_2 are resolved to $(T_1)_m$ and $(T_2)_m$ to match the components of F_m.

- The second phase is the refinement phase, which is described in Chapter 5 and improved in Section 6.4. It traverses the expanded cycle graph G_F^* to collect all marked potential exit nodes as set V according to the tagging from the first phase. Then it iterates through the subsets V_{sub} of V of size at most k''. In each iteration, we fix the nodes in V_{sub} and check whether the resulting forest is acyclic and has at most $k_p + 1$ components.

The 4-way branching phase's implementation is described in detail in Section 3.2 and Section 6.1. The implementation of the intermediate phase is discussed step by

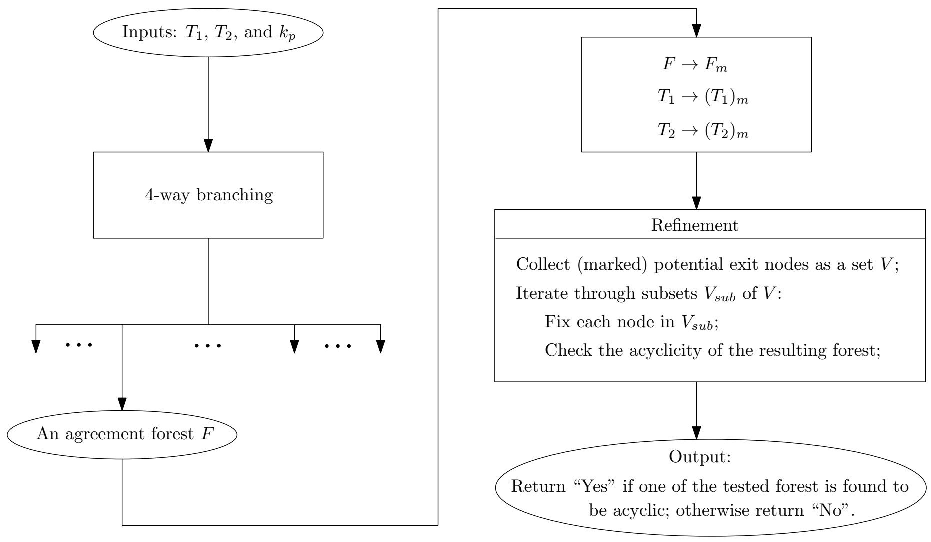

Inputs: T_1, T_2, and k_p
4-way branching
An agreement forest F
$F \to F_m$
$T_1 \to (T_1)_m$
$T_2 \to (T_2)_m$
Refinement
Collect (marked) potential exit nodes as a set V;
Iterate through subsets V_{sub} of V:
Fix each node in V_{sub};
Check the acyclicity of the resulting forest;
Output:
Return "Yes" if one of the tested forest is found to be acyclic; otherwise return "No".

Figure 7.2: The main steps of our algorithm from Chapter 6.

step in Chapter 4. Now, we discuss how to fix a subset of potential exit nodes, how to check the acyclicity of a forest within the claimed time bound for the refinement phase, and how to support constant-time lowest common ancestor (LCA) queries, which is the most important subroutine for cluster reduction.

7.2.1 Fix Potential Exit Nodes

To fix a given set of potential exit nodes, we fix them all together: first we mark all the edges to cut, then we remove these edges, and finally we contract nodes with only one child and discard empty components (components without any labelled leaves). Since it is not easy and not necessary to maintain the expanded cycle graph when fixing potential exit nodes, we compute the expanded cycle graph of the forest obtained by fixing the chosen set of potential exit nodes from scratch.

7.2.2 Check Acyclicity

Instead of traversing the whole expanded cycle graph, which costs $O(n)$, we implement a sub-linear solution by building a graph G with each component as a super node and hybrid edges as the edge set. Since each component cannot contain any cycles, the G is acyclic if and only if G^*_F is. The construction of G takes time linear to the number of components, because each component contributes at most two edges to G corresponding to the hybrid edges of this component. The number of components in G^*_F is less than $O(n)$. We check whether G is acyclic by trying to topological sort [51] it.

7.2.3 Constant-Time LCA Queries

The constant-time lowest common ancestor (LCA) query of two nodes with linear time preprocessing used to construct clusters comes from Michael A.Bender and Martin Farach-Colton [50].

7.2.4 Only Fix nodes in SCC

What the refinement phase does is to break cycles in the cycle graph G_F. Clearly, these cycles form strongly connnected components (SCC) in G_F. For components of the agreement forest F not involved in any SCCs of G_F, there is no need to fix the potential exit nodes in these components. This has the potential to decrease the number of potential exit nodes we add into set V. Since the running time of the refinement phase is exponential in the size of V, this change is a useful optimization in practice.

7.3 Data Sets

To demonstrate our algorithm's performance for true biological data, we tested our implementation on two biological datasets:

Aquificae dataset [52]. It has been shown that LGT plays an important role in the evolution of microbial communities [53, 54], especially for the phylum Aquificae [55]. Researchers believe that Aquificae have high rates of reticulation events to facilitate their adaption to new habitats. Thus, a data set consisting of phylogenies covering taxa in this phylum provides an ideal test bed for our algorithm. Robert Beiko [52] provided phylogenetic trees generated from 1173 sequenced bacterial and archaeal genomes. Chris Whidden [12] used 40463 trees among them with 1251 taxa to construct an MRP supertree and an SPR supertree. Multifurcations were introduced in these trees by collapsing bipartitions with less than 0.8 support. Then these trees were rooted to match the MRP supertree and the SPR supertree, respectively. We used each of these sets of trees as one data set whose trees we compared pairwise. We refer to the two data sets as MRP-Aquificae and SPR-Aquificae. The number of taxa of these trees ranges from 4 to 74.

Poaceae dataset. This dataset is provided by the Grass Phylogeny Working Group [56]. The dataset contains sequences for six loci: internal transcribed spacer of ribosomal DNA (ITS); NADH dehydrogenase, subunit F (ndhF); phytochrome B (phyB); ribulose $1,5-$biphosphate carboxylase/oxygenase, large subunit (rbcL); RNA polymerase II, subunit β'' (rpoC2); and granule bound starch synthase I (GBSSI or waxy). The six trees were previously analyzed by Heiko Schmidt [57], who generated the rooted binary trees for these loci. Bordewich et al. [43], Collins et al. [36] and Wu et al. [45] computed the hybridization number for each of the 15 pairs of binary trees. To compare our multifurcating solution with the binary solution, we ran our multifurcating algorithm for the same 15 pairs of trees.

7.4 Results

Our hybridization algorithm solved all instances in the given two data sets, which in total have about 800 thousand pairs of phylogenies. The longest running time for a single instance was 3 hours. The memory consumption was below 20 MB for every instance.

7.4.1 Correctness Evaluation

Until now, we haven't found any implementation for hybridization number of two multifurcating trees in the phylogenetics literature. Thus, it is impossible to verify the correctness of our implementation by comparing our results with other implementation's results. However, we found Dendroscope 3 [58] is able to compute the hybridization number between two multifurcating trees. We picked 20 pairs of trees with different numbers of taxa from the Aquificae dataset and compared our algorithm's results with the ones from Dendroscope 3. They all returned the same hybridization number. Since implementations of algorithms for computing binary hybridization number are available, we also ran our algorithm on the Poaceae dataset, which consists of binary trees, to compare our results with the ones obtained using Wu's implementation of an algorithm for binary hybridization number [45]. Our implementation produced the same answers as Wu's binary implementation. Thus, in both the multifurcating and the binary case, we have gained some confidence in the correctness of our implementation.

7.4.2 Performance Evaluation

Although there is no other implementation for hybridization number of two multifurcating trees to compare with, we can still validate its theoretical running time by regression and compare its running time to that of existing implementations of algorithms for multifurcating SPR distance with similar running times. To show our algorithm's running time is exponential in the parameter k (hybridization number), but linear in the input size n (number of taxa), we created Figure 7.3. The Aquificae dataset was used here. As most of the tree pairs in the Aquificae dataset have very small hybridization number (less than 5), most problem instances could be solved in several milliseconds. Therefore, we only used the 100 problem instances with the

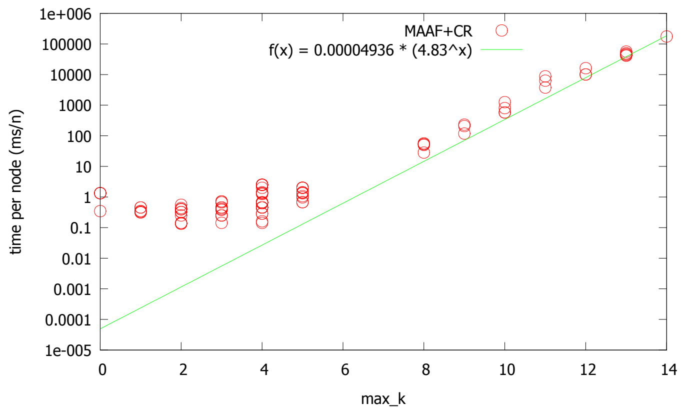


Figure 7.3: Performance of our MAAF algorithm with cluster reduction.

longest running time. Their number of taxa ranges from 8 to 74. Since we used cluster reduction to divide the original problem into smaller subproblems, theoretically, the algorithm's running time would be dominated by the cluster with the maximal hybridization number max_k. Thus, the running time is expected to be exponential in max_k among these subproblems. Therefore, in Figure 7.3, we chose $\frac{t}{n}$ as y-axis and max_k as x-axis, where t is the running time in milliseconds. We performed a regression analysis on these data points with function $f(x) = a*(4.83^x)$. From Figure 7.3, the regression line matches our data points well, which shows a growth exponentially in max_k, the maximal hybridization number among all subproblems. This matches the theoretical complexity of $O(4.83^k n)$. The deviation from the regression line for small values of max_k is likely due to the cost of cluster reduction and the very low cost of computing the hybridization number for very small values of k.

The same result from the Aquificae dataset was used again to create Figure 7.4, which shows the running time as a function of the hybridization number of the entire input. Running time in milliseconds was chosen as y-axis and the hybridization k was chosen as x-axis. The large problem instances solved by our MAAF algorithm have a hybridization number of 17 of two trees with 74 taxa, which lasted for from 11 minutes to 1 hour. The longest run was from an instance with a hybridization number of 16 of

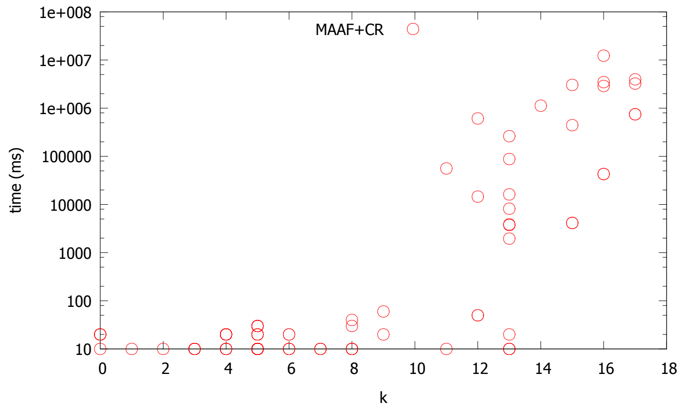


Figure 7.4: Actual running time of our MAAF algorithm with cluster reduction by the hybridization number.

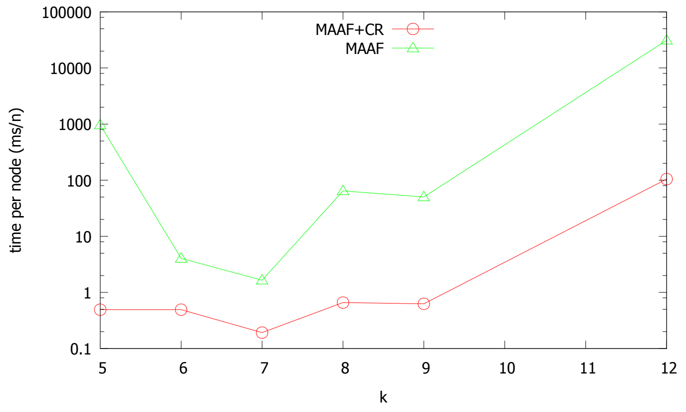


Figure 7.5: Performance comparison of our MAAF algorithm with and without cluster reduction.

two trees with 70 taxa, which lasted for about 3 hours. The reason why this happened is because some problem with bigger input size and bigger parameter may have better clusters, that is, the taxa and parameter divided to each cluster is significantly less than the original input.

To show the acceleration of cluster reduction on our MAAF algorithm, a subset of the 100 problem instances used in Figures 7.3 and 7.4 were solved using our MAAF algorithm without cluster reduction. If cluster reduction is not applied, by the theoretical complexity of $O(4.83^k n)$ and the fact that an instance with a hybridization number 11 of trees of 20 taxa ran for 10 minutes, an instance with a hybridization number 17 can run as long as 88 days. To finish this experiment in a reasonable time, we picked problem instances with hybridization numbers from 5 to 12. Let t again be the running time in milliseconds. In Figure 7.5, $\frac{t}{n}$ was chosen as y-axis and k was chosen as x-axis but note that $\frac{t}{n}$ was taken as the average of multiple instances with the same hybridization number. By Figure 7.5, the combination of our MAAF algorithm was always about 2 orders of magnitude faster than our MAAF algorithm only. This verified our speculation that cluster reduction can accelerate our MAAF algorithm in an exponential way.

Although there is no other implementation of multifurcating hybridization number to compare with, we compared our implementation with van Iersel's multifurcating MAF algorithm [47], which was implemented in Java with source code available from `http://homepages.cwi.nl/~iersel/MAF/`. This comparison is meaningful because our MAAF algorithm and van Iersel's MAF algorithm used the same 4-way branching and its complexity of $O(4^k n)$ is close to our algorithm's complexity of $O(4.83^k n)$. Moreover, van Iersel's implementation also applied cluster reduction. Again, we ran our MAAF algorithm and van Iersel's MAF algorithm on Aquificae dataset. We chose the 100 problem instances with the longest running time to generate Figure 7.6. Again, $\frac{t}{n}$ was chosen as y-axis and k was chosen as x-axis, where t is the running time in milliseconds, and k is the hybridization number for the MAAF algorithm and the SPR distance for the MAF algorithm.

The result in Figure 7.6 shows that the performance of our MAAF algorithm is almost the same as the one of van Iersel's MAF algorithm, while the MAAF algorithm has a theoretical complexity of $O(4.83^k n)$ and the MAF algorithm has a theoretical complexity of $O(4^k n)$. It may be caused by the fact that our MAAF algorithm was implemented in C++ but van Iersel's MAF algorithm was implemented in Java, or our MAAF algorithm may be better implemented resulting in smaller constant factors.

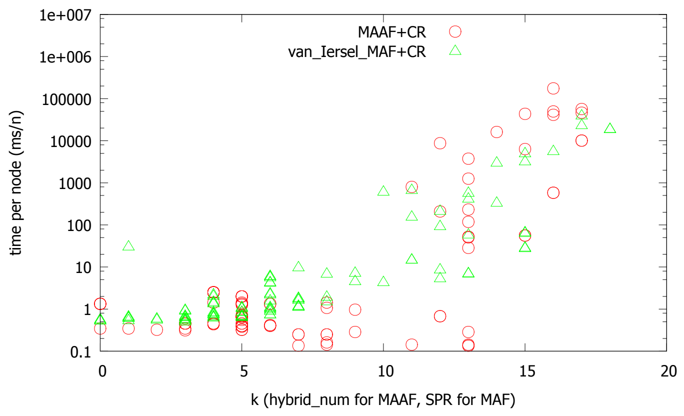


Figure 7.6: Comparison of running times of our MAAF algorithm with van Iersel's MAF algorithm.

Chapter 8

Conclusions and Future Work

In this thesis, we presented a fixed-parameter algorithm for hybridization number of two multifurcating phylogenetic trees, which also computes an MAAF of the two trees. Theoretically, our MAAF algorithm runs in $O(4.83^k n)$ time, where k is the soft hybridization number of the two trees. It is the first algorithm with a running time bounded by $O(c^k n)$, where c is a small constant. Practically, our implementation demonstrates that the FPT algorithm can efficiently compute moderate hybridization numbers. For example, our implementation took less than 1 hour to compute a hybridization number of 17 between trees with 74 taxa. Unfortunately, there is no other implementation to compare with in the phylogenetics literature until now.

A natural extension of our algorithm is to compare more than two phylogenies at the same time, either by computing an MAAF of these trees or by computing a hybridization network. Several results [37–39] are available for multiple binary trees, but there exists only one result [42] for multiple non-binary trees, which proposed two non-polynomial kernels. It is still unknown whether hybridization number for multiple non-binary trees is fixed-parameter tractable, and even for multiple binary trees, hybridization number poses numerous challenges due to the more tenuous relationship to MAAFs in the case of more than two trees. Several other directions for future work are the following:

- In Section 3.1, we discussed the structural results of multifurcating agreement forests. Lemmas 3 and 4 show that a 4-way branching is not always necessary. A 3-way branching is sufficient when one sibling is a child of the minimal LCA, and a 2-way branching is sufficient when no minimal LCA exists. Reducing the 4-way branching to 3-way or 2-way will decrease the size of the recursion tree of the branching phase and thus improve the efficiency of the whole MAAF algorithm. However, it is unclear whether any provable improvements can be obtained in this way and whether can be combined with the improved refinement step that considers only a subset of the potential exit nodes.

- In Section 7.1, we mentioned that the effect of cluster reduction is maximized when two trees are partitioned into the longest cluster sequence consisting of minimal clusters, but we failed to develop an efficient algorithm to construct such an optimal cluster sequence. The problem is how to find a pair of sibling sets $\{a_1, a_2, ..., a_i\}$ in T_1 and $\{b_1, b_2, ..., b_j\}$ in T_2 such that both sibling sets have the same set of labelled leaves and this shared set is minimized. We believe this will further improve the algorithm's performance but also believe that this problem is hard.

- As discussed in Chapter 7, we didn't find any implementation of multifurcating hybridization number in the phylogenetics literature, but recently, we found one implementation for multifurcating hybridization number in Dendroscope 3 [58], using Autumn algorithm [59]. However, the source code is not available and [59] is still under review. It would be very valuable to perform a comprehensive comparison between our MAAF algorithm and the Autumn algorithm once [59] is published.

- Another possible improvement for our MAAF algorithm is to extend Bordewich and Semple's kernelization rules [14] for binary hybridization to our multifurcating case. Ideally, it will decrease our algorithm's running time from $O(4.83^k n)$ to $O(4.83^k k + n^3)$. However, it may be challenging to apply the maximal n-chain replacement to two multifurcating trees, or this extension may lead to a kernel different from the binary case.

References

[1] C. R. Woese, O. Kandler, and M. L. Wheelis, "Towards a natural system of organisms: proposal for the domains archaea, bacteria, and eucarya." *Proceedings of the National Academy of Sciences*, vol. 87, no. 12, pp. 4576–4579, 1990.

[2] O. Gascuel, *Mathematics of evolution and phylogeny.* Oxford University Press, 2005.

[3] O. Gascuel, M. Steel *et al.*, "Reconstructing evolution. new mathematical and computational advances," *AMC*, vol. 10, p. 12, 2007.

[4] C. Semple and M. A. Steel, *Phylogenetics.* Oxford University Press, 2003, vol. 24.

[5] W. M. Fitch, "Toward defining the course of evolution: minimum change for a specific tree topology," *Systematic Biology*, vol. 20, no. 4, pp. 406–416, 1971.

[6] Z. Yang, "Paml: a program package for phylogenetic analysis by maximum likelihood," *Computer applications in the biosciences: CABIOS*, vol. 13, no. 5, pp. 555–556, 1997.

[7] B. Larget and D. L. Simon, "Markov chain monte carlo algorithms for the bayesian analysis of phylogenetic trees," *Molecular Biology and Evolution*, vol. 16, pp. 750–759, 1999.

[8] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic networks: concepts, algorithms and applications.* Cambridge University Press, 2010.

[9] L. Nakhleh, "Evolutionary phylogenetic networks: models and issues," in *Problem Solving Handbook in Computational Biology and Bioinformatics.* Springer, 2011, pp. 125–158.

[10] D. M. Hillis, C. Moritz, B. K. Mable, and R. G. Olmstead, *Molecular systematics.* Sinauer Associates Sunderland, MA, 1996, vol. 23.

[11] M. Baroni, S. Grünewald, V. Moulton, and C. Semple, "Bounding the number of hybridisation events for a consistent evolutionary history," *Journal of mathematical biology*, vol. 51, no. 2, pp. 171–182, 2005.

[12] C. Whidden, N. Zeh, and R. G. Beiko, "Supertrees based on the subtree prune-and-regraft distance," PeerJ PrePrints, Tech. Rep., 2013.

[13] C. Whidden, "Efficient computation of maximum agreement forests and their applications," Ph.D. dissertation, Dalhousie University, 2013.

[14] M. Bordewich and C. Semple, "Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 4, no. 3, pp. 458–466, 2007.

[15] B. L. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.

[16] M. Bordewich and C. Semple, "On the computational complexity of the rooted subtree prune and regraft distance," *Annals of Combinatorics*, vol. 8, no. 4, pp. 409–423, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00026-004-0229-z

[17] W. Maddison, "Reconstructing character evolution on polytomous cladograms," *Cladistics*, vol. 5, no. 4, pp. 365–377, 1989. [Online]. Available: http://dx.doi.org/10.1111/j.1096-0031.1989.tb00569.x

[18] C. Whidden, R. G. Beiko, and N. Zeh, "Fixed-parameter algorithms for maximum agreement forests," *SIAM Journal on Computing*, vol. 42, no. 4, pp. 1431–1466, 2013.

[19] L. van Iersel, S. Kelk, N. Lekic, and L. Stougie, "A short note on exponential-time algorithms for hybridization number," *arXiv preprint arXiv:1312.1255*, 2013.

[20] T. Piovesan and S. Kelk, "A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 10, no. 1, pp. 18–25, 2013.

[21] C. Whidden, R. G. Beiko, and N. Zeh, "Fixed-parameter and approximation algorithms for maximum agreement forests of multifurcating trees," *CoRR*, vol. abs/1305.0512, 2013.

[22] V. V. Vazirani, *Approximation algorithms.* springer, 2001.

[23] J. Hein, T. Jiang, L. Wang, and K. Zhang, "On the complexity of comparing evolutionary trees," *Discrete Applied Mathematics*, vol. 71, no. 1, pp. 153–169, 1996.

[24] E. M. Rodrigues, M.-F. Sagot, and Y. Wakabayashi, "The maximum agreement forest problem: Approximation algorithms and computational experiments," *Theoretical Computer Science*, vol. 374, no. 1, pp. 91–110, 2007.

[25] M. Bordewich, C. McCartin, and C. Semple, "A 3-approximation algorithm for the subtree distance between phylogenies," *Journal of Discrete Algorithms*, vol. 6, no. 3, pp. 458 – 471, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866707000627

[26] C. Whidden and N. Zeh, *A unifying view on approximation and FPT of agreement forests*. Springer, 2009.

[27] L. van Iersel, S. Kelk, N. Lekić, and L. Stougie, “Approximation algorithms for nonbinary agreement forests,” *ArXiv e-prints*, Oct. 2012.

[28] S. Kelk, L. van Iersel, N. Lekic, S. Linz, C. Scornavacca, and L. Stougie, “Cycle killer... qu’est-ce que c’est? on the comparative approximability of hybridization number and directed feedback vertex set,” *SIAM Journal on Discrete Mathematics*, vol. 26, no. 4, pp. 1635–1656, 2012.

[29] L. van Iersel, S. Kelk, N. Lekić, and C. Scornavacca, “A practical approximation algorithm for solving massive instances of hybridization number,” in *Algorithms in Bioinformatics*. Springer, 2012, pp. 430–440.

[30] L. van Iersel, S. Kelk, N. Lekic, and L. Stougie, “Approximation algorithms for nonbinary agreement forests,” *SIAM Journal on Discrete Mathematics*, vol. 28, no. 1, pp. 49–66, 2014.

[31] R. Niedermeier, “Invitation to fixed-parameter algorithms,” 2006.

[32] F. Shi, J. Wang, J. Chen, Q. Feng, and J. Guo, “Algorithms for parameterized maximum agreement forest problem on multiple trees,” *Theoretical Computer Science*, 2014.

[33] Z.-Z. Chen and L. Wang, “Hybridnet: a tool for constructing hybridization networks,” *Bioinformatics*, vol. 26, no. 22, pp. 2912–2913, 2010.

[34] C. Scornavacca, S. Linz, and B. Albrecht, “A first step toward computing all hybridization networks for two rooted binary phylogenetic trees,” *Journal of Computational Biology*, vol. 19, no. 11, pp. 1227–1242, 2012.

[35] B. Albrecht, C. Scornavacca, A. Cenci, and D. H. Huson, “Fast computation of minimum hybridization networks,” *Bioinformatics*, vol. 28, no. 2, pp. 191–197, 2012.

[36] J. Collins, S. Linz, and C. Semple, “Quantifying hybridization in realistic time,” *Journal of Computational Biology*, vol. 18, no. 10, pp. 1305–1318, 2011.

[37] L. van Iersel, S. Kelk, N. Lekić, C. Whidden, and N. Zeh, “Hybridization number on three trees,” *arXiv preprint arXiv:1402.2136*, 2014.

[38] Z.-Z. Chen and L. Wang, “Algorithms for reticulate networks of multiple phylogenetic trees,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 9, no. 2, pp. 372–384, 2012.

[39] L. Van Iersel and S. Linz, “A quadratic kernel for computing the hybridization number of multiple trees,” *Information Processing Letters*, vol. 113, no. 9, pp. 318–323, 2013.

[40] S. Linz and C. Semple, "Hybridization in nonbinary trees," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. 6, no. 1, pp. 30–45, 2009.

[41] S. Kelk and C. Scornavacca, "Towards the fixed parameter tractability of constructing minimal phylogenetic networks from arbitrary sets of nonbinary trees," *arXiv preprint arXiv:1207.7034*, 2012.

[42] L. van Iersel and S. Kelk, "Kernelizations for the hybridization number problem on multiple nonbinary trees," *arXiv preprint arXiv:1311.4045*, 2013.

[43] M. Bordewich, S. Linz, K. S. John, and C. Semple, "A reduction algorithm for computing the hybridization number of two trees," *Evolutionary bioinformatics online*, vol. 3, p. 86, 2007.

[44] Y. Wu, "A practical method for exact computation of subtree prune and regraft distance," *Bioinformatics*, vol. 25, no. 2, pp. 190–196, 2009.

[45] Y. Wu and J. Wang, "Fast computation of the exact hybridization number of two phylogenetic trees," in *Bioinformatics Research and Applications.* Springer, 2010, pp. 203–214.

[46] M. L. Bonet and K. S. John, "Efficiently calculating evolutionary tree measures using sat," in *Theory and Applications of Satisfiability Testing-SAT 2009.* Springer, 2009, pp. 4–17.

[47] L. van Iersel, "Maf: Maximum agreement forests for nonbinary trees," http://homepages.cwi.nl/~iersel/MAF/, 2012.

[48] C. Whidden, R. G. Beiko, and N. Zeh, "Fast fpt algorithms for computing rooted agreement forests: Theory and experiments," in *Experimental Algorithms.* Springer, 2010, pp. 141–153.

[49] S. Linz and C. Semple, "A cluster reduction for computing the subtree distance between phylogenies," *Annals of Combinatorics*, vol. 15, no. 3, pp. 465–484, 2011.

[50] M. A. Bender and M. Farach-Colton, "The lca problem revisited," in *LATIN 2000: Theoretical Informatics.* Springer, 2000, pp. 88–94.

[51] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms.* MIT press Cambridge, 2001, vol. 2.

[52] R. G. Beiko, "Telling the whole story in a 10,000-genome world," *Biol Direct*, vol. 6, p. 34, 2011.

[53] R. G. Beiko, T. J. Harlow, and M. A. Ragan, "Highways of gene sharing in prokaryotes," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 40, pp. 14 332–14 337, 2005.

[54] B. Boussau, L. Guéguen, and M. Gouy, "Accounting for horizontal gene transfers explains conflicting hypotheses regarding the position of aquificales in the phylogeny of bacteria," *BMC evolutionary biology*, vol. 8, no. 1, p. 272, 2008.

[55] R. J. Eveleigh, C. J. Meehan, J. M. Archibald, and R. G. Beiko, "Being aquifex aeolicus: Untangling a hyperthermophiles checkered past," *Genome biology and evolution*, vol. 5, no. 12, pp. 2478–2497, 2013.

[56] G. P. W. Group, N. P. Barker, L. G. Clark, J. I. Davis, M. R. Duvall, G. F. Guala, C. Hsiao, E. A. Kellogg, H. P. Linder *et al.*, "Phylogeny and subfamilial classification of the grasses (poaceae)," *Annals of the Missouri Botanical Garden*, pp. 373–457, 2001.

[57] H. A. Schmidt, "Phylogenetic trees from large datasets," 2003.

[58] D. H. Huson and C. Scornavacca, "Dendroscope 3: an interactive tool for rooted phylogenetic trees and networks," *Systematic biology*, p. sys062, 2012.

[59] D. H. Huson and S. Linz, "Computing minimum hybridization networks from real phylogenetic trees," *Under Review*, 2012.