EFFICIENT COMPUTATION AND APPLICATION OF MAXIMUM AGREEMENT FORESTS

by

Chris Whidden

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
July 2013

Table of Contents

List of Tables

List of Figures

Abstract

Rampant lateral gene transfer (LGT) among prokaryotes, hybridization in plants and other reticulate evolutionary processes invalidate typical phylogenetic tree models by violating the assumption that organisms only inherit genetic information from a single parent species. Comparing the different evolutionary histories of multiple genes is necessary to identify and assess these processes. In this work I develop efficient approximation and fixed-parameter algorithms for computing rooted maximum agreement forests (MAFs) and maximum acyclic agreement forests (MAAFs) of pairs of phylogenetic trees. Their sizes correspond to the subtree-prune-and-regraft (SPR) distance and the hybridization number of these pairs of trees, which are important measures of the dissimilarity of phylogenies used in studying reticulate evolution.

Although these MAFs and MAAFs are NP-hard to compute, my fixed-parameter algorithms are practical because they scale exponentially with the computed distance rather than the size of the trees. I contribute efficient fixed-parameter algorithms for computing MAFs and MAAFs of two binary rooted trees and give the first efficient fixed-parameter and approximation algorithms for computing MAFs of two multifurcating rooted trees. My open-source implementation of the MAF algorithms is orders of magnitude faster than previous approaches, reducing the time required to compute SPR distances of 46 between trees of 144 species to fractions of a second whereas previous approaches required hours to compute SPR distances of 25.

These fast MAF–based distance metrics enable the construction of supertrees to reconcile a collection of gene trees and rapid inference of LGT. Simulations demonstrate that supertrees minimizing the SPR distance are more accurate than other supertree methods under plausible rates of LGT. I constructed an SPR supertree from a phylogenomic dataset of 40,631 gene trees covering 244 genomes from several major bacterial phyla and inferred "highways" of gene transfer between these bacterial classes and genera; a small number of these highways connect distantly related genera and can highlight specific genes implicated in long-distance LGT. These fast MAF algorithms are thus practical and enable new analyses of reticulate evolution.

List of Abbreviations and Symbols Used

Abbreviations

AF	Agreement Forest
AAF	Acyclic Agreement Forest
DNA	Deoxyribonucleic acid
FPT	Fixed-Parameter Tractable
LCA	Least common ancestor
LGT	Lateral Gene Transfer
MAF	Maximum Agreement Forest
MAAF	Maximum Acyclic Agreement Forest
MRP	Matrix Representation with Parsimony
RF	Robinson-Foulds
RNA	Ribonucleic acid
SPR	Subtree Prune-and-Regraft

Symbols

$\boldsymbol{a \sim_F b}$	There exists a path from $\boldsymbol{a}$ to $\boldsymbol{b}$ in $\boldsymbol{F}$
$\boldsymbol{ab\|c}$	A triple $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, where the LCA of $\boldsymbol{a}$ and $\boldsymbol{b}$ is deeper than the LCA of $\boldsymbol{a}$,$\boldsymbol{b}$, and $\boldsymbol{c}$
$\boldsymbol{(a, c)}$	A sibling pair of a tree. The siblings of $\boldsymbol{a}$ and $\boldsymbol{c}$ in another tree are generally denoted $\boldsymbol{b}$ and $\boldsymbol{d}$, respectively

$\{\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_m}\}$ A sibling group of a tree. The leaves of the sibling subtree of $\boldsymbol{a_i}$ in another tree are generally denoted $\boldsymbol{B_i}$

$\boldsymbol{a_0}$ A member of a sibling group whose parent edge should not be cut

$\boldsymbol{\mathcal{C}_T(v)}$ The cluster of node $\boldsymbol{v}$ in $\boldsymbol{T}$

$\boldsymbol{\mathcal{C}_T}$ All of the clusters in $\boldsymbol{T}$

$\boldsymbol{d_{SPR}(T_1, T_2)}$ The SPR distance between $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{d_{RF}(T_1, T_2)}$ The RF distance between $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{e(T_1, T_2, F)}$ The number of edges that must be cut in $\boldsymbol{F}$ to reconcile it with $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{e_x}$ The parent edge of node $\boldsymbol{x}$

$\boldsymbol{F}$ A forest

$\boldsymbol{F_1}$, $\boldsymbol{F_2}$ Intermediate forests of $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{F^x}$ The subtree rooted at node $\boldsymbol{x}$ in $\boldsymbol{F}$

$\boldsymbol{F \setminus E}$ Remove the edges in $\boldsymbol{E}$ from forest $\boldsymbol{F}$

$\boldsymbol{F \div E}$ Remove the edges in $\boldsymbol{E}$ from $\boldsymbol{F}$ and suppress degree-two nodes

$\boldsymbol{F \diamond E}$ Protect the edges of forest $\boldsymbol{F}$ in set $\boldsymbol{E}$

$\boldsymbol{G_F}$ The cycle graph of $\boldsymbol{F}$

$\boldsymbol{G_F^*}$ The expanded cycle graph of $\boldsymbol{F}$

$\boldsymbol{hyb(T_1, T_2)}$ The hybridization number of $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{k}$ The distance measure under consideration

$\boldsymbol{l}$ The minimal LCA of a subset of a sibling group

$\boldsymbol{n}$ The number of leaves in a tree

$\boldsymbol{p_x}$ The parent of a node $\boldsymbol{x}$

$\boldsymbol{\phi(x)}$ A mapping of nodes from a tree to an AF

$\boldsymbol{\phi^{-1}(x)}$ A mapping of nodes from an AF to a tree

$\boldsymbol{\rho}$ The labelled root of a tree

$\boldsymbol{r}$ The number of nodes of a subset of a sibling group that share a given minimal LCA

$\boldsymbol{R_d}$ A set of leaves that have been reconciled

$\boldsymbol{R_t}$ A set of leaves that must be reconciled

$\boldsymbol{s_i}$ The number of nodes on the path from $\boldsymbol{a_i}$ to $\boldsymbol{l}$

$\boldsymbol{T}$ A tree

$\boldsymbol{T(V)}$ the smallest subtree of $\boldsymbol{T}$ that connects all nodes in $\boldsymbol{V}$

$\boldsymbol{T|V}$ Suppress degree-two nodes in $\boldsymbol{T(V)}$

$\boldsymbol{T_1}$, $\boldsymbol{T_2}$ Two trees being compared

$\boldsymbol{\dot{T}_1}$, $\boldsymbol{\dot{F}_2}$ The remaining portion of $\boldsymbol{T_1}$ and $\boldsymbol{F_2}$ that must be reconciled

$\boldsymbol{T^x}$ The subtree rooted at node $\boldsymbol{x}$ in $\boldsymbol{T}$

$\boldsymbol{\tilde{e}\,(T_1, T_2, F)}$ The number of edges that must be cut in $\boldsymbol{F}$ to reconcile it acyclicly with $\boldsymbol{T_1}$ and $\boldsymbol{T_2}$

$\boldsymbol{X}$-tree A tree with label set $\boldsymbol{X}$

Acknowledgements

This thesis is dedicated to my wife, Hallie, and beautiful daughters Emily and Leia. Without your love and support this work would never have been completed.

I wish to thank my supervisors, Robert Beiko and Norbert Zeh, for their feedback, brainstorming, proof-breaking, and the myriad other ways they have contributed to my research. I also thank the other members of my committee, Christian Blouin and Meng He, for their many helpful comments.

The Beiko lab has been instrumental to my success by listening to my work in progress and offering suggestions and advice. Thanks go to Sylvia Churcher, Kathryn Duffy, Kathryn Dunphy, Rob Eveleigh, Catherine Holloway, Morgan Langille, Norm MacDonald, Timothy Mankowski, Conor Meehan, Brett O'Donnel, Donovan Parks, Scott Perry, and Dennis Wong. I particularly wish to thank Joel Navarrete for his work implementing and testing some of my supertree ideas.

Finally, I wish to thank the funding agencies that made this work possible. My research was funded by generous support from the Tula Foundation, a Natural Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship, and the Killam Trusts Predoctoral Scholarship.

Chapter 1

Introduction

Phylogenetic trees are a standard model to represent the evolutionary relationships among a set of species and are an indispensable tool in evolutionary biology [54]. Early methods of building phylogenetic trees used morphology, or structural characteristics of species, to determine their relatedness. However, advances in molecular biology have allowed the widespread use of DNA and protein sequence data to build phylogenies based on homologous genes, that is, genes from separate organisms that have a shared ancestry. Figure 1.1 shows, for example, the seminal tree constructed by Woese et al. [108] that separated life into the three domains Bacteria, Archaea and Eucarya. Molecular phylogenetics is particularly useful in the study of microscopic organisms due to their high rates of evolution and subtle differences in appearance. However, even good phylogenetic inference methods cannot guarantee that a constructed tree correctly represents evolutionary relationships—and there may not even exist such a tree—because not all groups of species follow a simple tree-like evolutionary pattern. Collectively known as reticulation events, non-tree-like evolutionary processes, such as hybridization, lateral gene transfer (LGT), and recombination, result in species being composites of genes derived from different ancestors. For example, evidence suggesting LGT between Archaea and Bacteria [75] challenges the tree in Figure 1.1, which was constructed from a single "marker" gene. These reticulate processes allow species to rapidly acquire useful traits and adapt to new environments. This includes harmful traits of pathogenic bacteria, such as antibiotic resistance, and LGT appears to have contributed to the emergence of pathogens such as *Mycobacterium tuberculosis* [84].

Due to reticulation events, phylogenetic trees representing the evolutionary history of different genes found in the same set of species may differ. To reconcile these differing evolutionary histories, one can use phylogenetic distance metrics that determine how well the evolutionary hypotheses of two or more phylogenetic trees agree

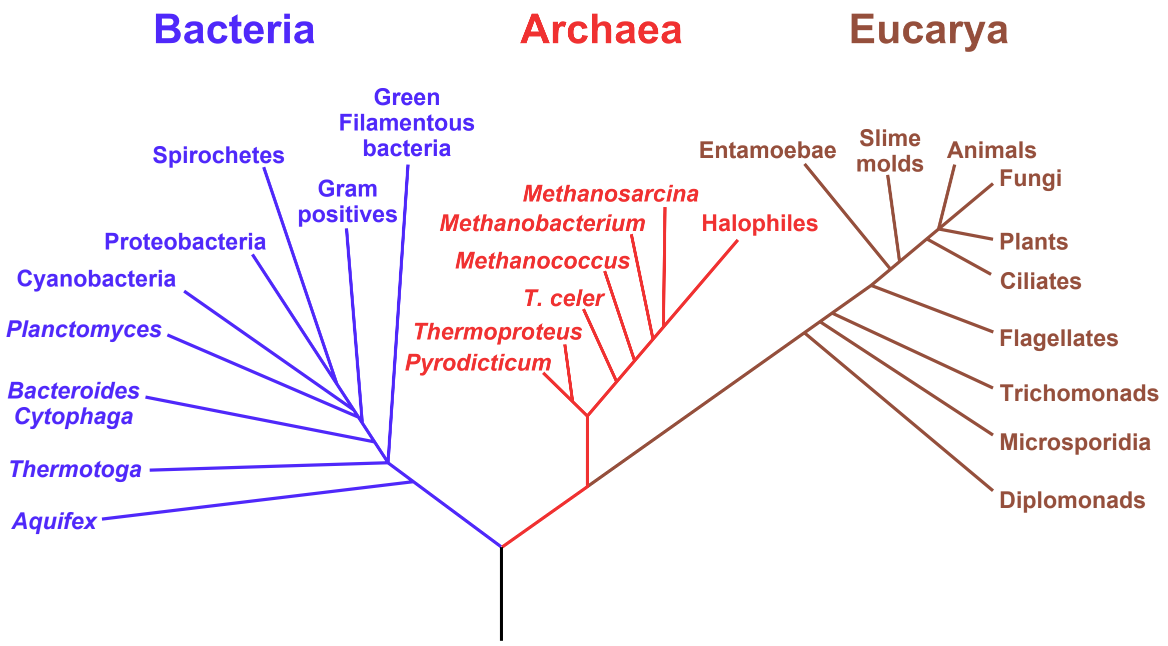


Figure 1.1: A phylogenetic tree based on small subunit ribosomal RNA genes as developed by Woese et al. [108], proposing the three domains Bacteria, Archaea, and Eucarya. Figure adapted from [71]. Reticulation events challenge the assumptions behind such trees.

and often allow us to discover reticulation events that explain the differences between the trees. To simultaneously represent these discordant topologies, one can use a phylogenetic network, which is a generalization of a phylogenetic tree that allows species to inherit from more than one parent.

Several metrics are commonly used to define the distance between phylogenies. The Robinson-Foulds distance [82] is popular, as it can be calculated in linear time [39]. Other metrics, such as the *subtree prune-and-regraft* (SPR) distances [54] and the *hybridization number* [7], are more biologically meaningful but also NP-hard to compute [3, 23, 25, 53]. The SPR distance is equivalent to the minimum number of lateral gene transfers required to transform one tree into the other [7, 13] and thus provides a lower bound on the number of reticulation events needed to reconcile the two phylogenies. The hybridization number is the minimum number of edges that must be added to one tree to transform it into a hybridization network of both trees and thus provides a lower bound on the number of hybridization events in an evolutionary history consistent with both trees [7].

Multifurcations (alternatively, *polytomies*) are vertices of a tree with three or more children. A multifurcation is *hard* if it indeed represents an inferred common ancestor which produced three or more species as direct descendants, and is *soft* if it simply represents ambiguous or uncertain evolutionary relationships [66]. Simultaneous speciation events are assumed to be rare, so the usual assumption is that all multifurcations are soft. If we force the resolution of uncertain relationships into binary trees, then we infer evolutionary relationships that are not supported by the original data and may infer meaningless reticulation events. Thus, it is crucial to develop efficient algorithms to compute these distances (and their associated series of permutations) for multifurcating trees.

The minimum number of reticulation events required to reconcile two trees provides the simplest explanation for the difference between the trees. For this reason, these metrics have been used regularly to model reticulate evolution [67, 74], and the development of efficient algorithms to compute the distance between two trees under these metrics has been the focus of much research (see Section 1.1.1). The close relationship between SPR operations and reticulation events has also led to advances in network models of evolution [7, 25, 74].

In addition, these distance metrics can be used as an optimality criterion to construct supertrees. Supertree methods aim to reconcile a multitude of gene trees into a single tree, which may serve as a hypothesis of organismal descent or relatedness, by optimizing a similarity criterion. The Robinson-Foulds distance has been used in this manner but, due to the lack of efficient algorithms for computing SPR distances and hybridization numbers, these distance metrics have not. Section 1.1.2 discusses related work on supertrees.

Each of these distance metrics can be modeled using appropriately defined agreement forests (AFs); rooted maximum agreement forests (MAFs) for SPR, such as the MAF of Figure 1.2, and rooted maximum acyclic agreement forests (MAAFs) for the hybridization number. An agreement forest of two phylogenies has the property that it can be obtained from either tree by cutting an appropriate set of edges. Given an agreement forest obtained by removing k edges from each tree, a set of k SPR operations that transform one tree into the other can be recovered easily. As such, it captures the evolutionary relationships that are consistent between both trees. A

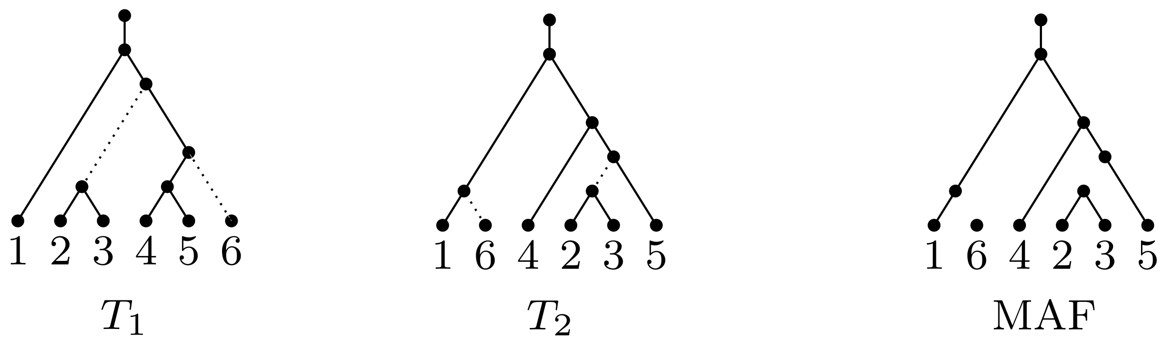


Figure 1.2: An MAF of two phylogenetic trees. The MAF can be obtained by cutting the dotted edges in both trees. Any such AF requires cutting at least two edges from each tree, which is also the SPR distance between these trees.

maximum agreement forest is an agreement forest obtained by removing the minimum possible number of edges. The corresponding set of operations represents a possible minimum set of reticulation events that reconcile the two trees. Similarly, given an *acyclic* agreement forest of two trees (a restriction of an agreement forest that disallows the donation of genetic information from descendant nodes to ancestor nodes), a hybrid network with that many hybridization events can be constructed quickly [25]. Thus, developing efficient algorithms for computing these MAFs and MAAFs is the best way to enable the effective study of reticulation events.

1.1 Related Work

1.1.1 MAFs and MAAFs

While SPR distance and the hybridization number (and potentially TBR distance) capture biologically meaningful notions of similarity between phylogenies, their practical use has been limited by the fact that they are NP-hard to compute [3,23,25,53]. There are several standard approaches for dealing with NP-hard optimization problems that have been employed to compare phylogenies using these distance measures.

Approximation algorithms. Hein et al. [51] claimed a 3-approximation algorithm for computing SPR distances and introduced the notion of a maximum agreement forest (MAF) as the main tool underlying both the approximation algorithm and a proposed NP-hardness proof for computing SPR distances. The central claim was that the number of components in an MAF of two phylogenies is one more than the minimum number of SPR operations needed to transform one into the other. Unfortunately, there were subtle mistakes in the proofs. Allen and Steel [3] proved that the number of components in an MAF is in fact one more than the closely related tree bisection and reconnection (TBR) distance between the two trees. Rodrigues et al. [83] provided instances where the algorithm of [51] provides an approximation guarantee no better than 4 for the size of an MAF, thereby disproving the 3-approximation claim of [51]. They also proposed a modification to the algorithm, which they claimed to produce a 3-approximation for the TBR distance. A counterexample to this claim was provided by Bonet et al. [20], who showed, however, that both the algorithms of [51] and [83] compute 5-approximations of the SPR distance between two *rooted* phylogenies, and that the algorithms can be implemented to run in linear time. The approximation ratio was improved to 3 by Bordewich et al. [22], but at the expense of an increased running time of $\mathrm{O}(n^5)$.[1] A second 3-approximation algorithm presented in [83] achieves a running time of $\mathrm{O}(n^2)$. In [103] we improved the time required by this algorithm to linear. Using entirely different ideas, Chataigner [28] obtained an 8-approximation algorithm for TBR distances of two or more trees. There is currently no constant factor approximation algorithm for the hybridization number of two rooted phylogenies, and achieving such an algorithm is unlikely—Bordewich and Semple [25] showed that this problem is APX-hard and Kelk et. al [56] showed that it has a constant factor approximation algorithm if, and only if, the directed feedback vertex set problem [56] has such an algorithm. Further, there has been no previous work on developing approximation algorithms for these metrics on multifurcating trees other than a recently proposed polynomial time 4-approximation for the multifurcating SPR distance [97].

Fixed-parameter algorithms. Fixed-parameter algorithms have a running time that is exponential in some parameter that is specific to the problem but independent of the input size. These algorithms are more attractive for determining reticulation events than approximate methods, as they provide exact solutions. Given that the identification of meaningful putative reticulation events from two phylogenetic trees is possible only if the trees carry a strong vertical signal, that is, if the number of

[1]Using non-trivial but standard data structures, the running time can be reduced to $\mathrm{O}(n^4)$.

reticulation events is small compared to the size of the trees, the distance under a given metric is a natural parameter for fixed-parameter algorithms that compute distances between phylogenies.

The previously best fixed-parameter algorithm for the rooted SPR distance of binary phylogenies is due to Whidden and Zeh [103] and runs in $\mathrm{O}\left(3^k n\right)$ time, where k is the distance between the two trees. Bordewich et al. [22] gave a previous algorithm that runs in $\mathrm{O}\left(4^k \cdot k^4 + n^3\right)$ time. A preprint by van Iersel et al. [97] extends the algorithm of Whidden and Zeh [103] to nonbinary phylogenies and requires $\mathrm{O}\left(4^k \cdot p(n)\right)$ time, where $p(\cdot)$ is a polynomial function. For TBR distance, the previous best result is also due to Whidden and Zeh [103], who provided an algorithm with running time $\mathrm{O}\left(4^k n\right)$. Earlier algorithms for this problem by Hallett and McCartin [49] and Allen and Steel [3] had running times $\mathrm{O}\left(4^k \cdot k^5 + p(n)\right)$ and $\mathrm{O}\left(k^{3k} + p(n)\right)$, respectively. For unrooted SPR, Hickey et al. [52] first claimed a fixed-parameter algorithm, but the correctness proof was flawed. Recently, St. John [89] proposed a correction of the central technical lemma in Hickey et al.'s result. In [24], Bordewich and Semple provided a fixed-parameter algorithm for the hybridization number of two rooted phylogenies with running time $\mathrm{O}\left((28k)^k + n^3\right)$. Linz and Semple [62] extended these results to non-binary rooted phylogenies. Kelk et al. [56] provided an improved analysis of the kernel size for hybridization number, which reduces the running time of the algorithm by Bordewich and Semple to $\mathrm{O}\left((18k)^k + n^3\right)$. Using a different approach of 'terminals', Piovesan et al [77] gave an algorithm for the hybridization number of non-binary phylogenies with running time $\mathrm{O}\left(6^k k! \cdot p(n)\right)$. Chen and Wang [31] recently proposed an algorithm for computing all MAAFs of two or more binary phylogenies. Their algorithm combines the $\mathrm{O}\left(3^k n\right)$ search for agreement forests of Whidden and Zeh [103] with an exhaustive search based on an observation in the same paper that a superforest of an MAAF can be refined to an MAAF by cutting appropriate edges incident to the roots in the current forest. There have also been several recent fixed-parameter algorithms for hybridization [2,33] that are extensions of the rooted MAF algorithm from [100,102] (Chapter 3); however, their running times are $\mathrm{O}\left(3^{14k} + n^3\right)$ and unbounded, respectively. These times are exponential in their reduced input size, bounded by a function on k, unlike a bounded search tree algorithm with a running time of $\mathrm{O}\left(b^k \cdot p(n)\right)$ where b is a small constant and $p(n)$ is a polynomial function of n.

Heuristics. We distinguish two types of heuristic approach to solving NP-hard problems. The first type of heuristic algorithms are similar to approximation algorithms in that they provide approximate solutions efficiently, but they do not provide a guaranteed approximation ratio. The second type of heuristic algorithms provide exact solutions with no guaranteed running time bound.

LatTrans by Hallet and Lagergen [48] models lateral gene transfer events by a restricted version of rooted SPR operations. It computes the exact distance under this simpler metric in $\mathrm{O}\left(2^k n^2\right)$ time. HorizStory by Macleod et al. [65] supports multifurcating trees but does not consider SPR operations where the pruned subtree contains more than one leaf. EEEP by Beiko and Hamilton [13] performs a breadth-first SPR search on a rooted start tree but performs unrooted comparisons between the explored trees and an unrooted reference tree. The distance returned is not guaranteed to be exact due to optimizations and heuristics that limit the scope of the search, although EEEP provides options to compute the exact unrooted SPR distance with no non-trivial bound on the running time. More recently, RiataHGT by Nakhleh et al. [73] and TNT by Goloboff [44] quickly calculate an (unbounded) approximation of the SPR distance between rooted multifurcating trees.

Reductions. Two algorithms for computing rooted SPR distances, Sprdist [110] and TreeSAT [19], express the problem of computing maximum agreement forests as an integer linear program (ILP) and a satisfiability problem (SAT), respectively, and employ efficient ILP and SAT solvers to obtain a solution. Sprdist has been shown to outperform EEEP and Lattrans [110]. Although such algorithms draw on the close scrutiny that has been applied to these problems, the conversion process may throw away information that can be exploited, such as a fixed parameter.

1.1.2 Supertrees

Genomes contain a great deal of hereditary information in the form of homologous genes, but histories of these genes can differ due to artifacts of phylogenetic inference

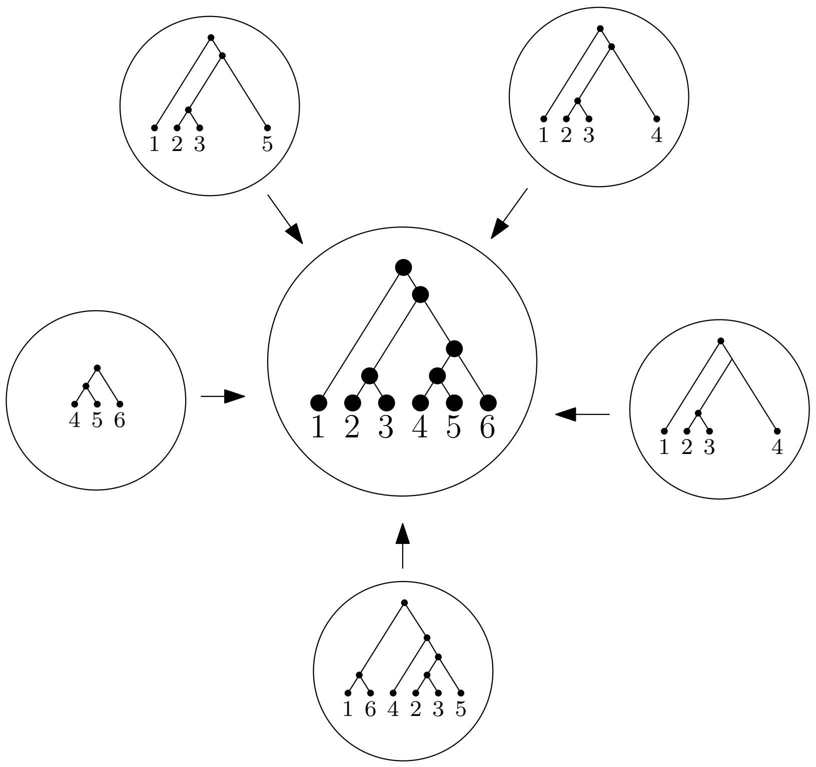


Figure 1.3: An example of supertree inference.

and phenomena such as lateral gene transfer (LGT). Supertree methods aim to reconcile a multitude of gene trees into a single tree, which may serve as a hypothesis of organismal descent or relatedness, by optimizing a similarity criterion. Figure 1.3 shows an example of reconciling trees into a single supertree. Supertree approaches have been used to build large-scale phylogenies including the first phylogeny of nearly all extant mammals [17], the first family-level phylogeny of flowering plants [38], and the first species-level phylogeny of non-avian dinosaurs [64]. They have also been used to study the extent of LGT in prokaryotes [14] and to disentangle the origin of eukaryotic genomes [78]. One key advantage of supertree methods is that they can take as input sets of gene trees sampled from overlapping but non-identical sets of taxa, in contrast with consensus tree approaches, which require that all input trees contain exactly the same set of leaves [1]. Simulations have shown that supertrees are more reliable in the presence of a moderate amount of misleading LGT than the supermatrix approach which requires concatenated alignments of many gene sequences [59].

Many optimality criteria have been proposed for supertree construction. Matrix representation with parsimony (MRP) [8, 80] was among the earliest methods proposed and remains the most commonly used, but detailed work with MRP has raised several concerns with the approach. MRP converts input trees into a binary character matrix and scores candidate supertrees with the parsimony problem on this matrix. In the parsimony problem, each row of the matrix represents one species, each column is mapped onto the supertree leaves, and the parsimony score of a supertree is the minimum number of character changes required in a mapping of characters to the supertree interior nodes. Although it is NP-hard to find a supertree with the minimum parsimony score, fast hill-climbing heuristics in PAUP* allow MRP to be applied to large datasets [42, 85, 94]. MRP is very effective in practice, quickly constructing supertrees of competitive quality in every tested metric [18,30,40]. However, it is not clear why the MRP approach performs so well and it may generate relationships that do not belong to any of the source trees [79], has problems resulting from unequal representation of taxa [16], and may include relationships contradicted by the majority of source trees [43]. Other developed supertree criteria include consensus supertrees [46], majority-rule supertrees [35], Quartet supertrees [76] and Triplet supertrees [61]. However, like MRP, other supertree building methods that are not based on symmetric tree-to-tree similarity measures may be unduly influenced by the shapes of the input trees [107].

Bansal et al. [6] recently proposed Robinson-Foulds (RF) supertrees, which aim to minimize the total RF distance [82] between the supertree and the set of input trees. The RF measure captures the number of clusters (clades, in the binary case) that differ between two trees, so the RF supertree approach aims to maintain as much phylogenetic information from the input trees as possible. RF supertrees can be feasibly computed from binary input trees with the fast hill-climbing heuristics of Bansal et al. [6]; others have begun to extend the approach to unrooted trees with local search heuristics [29]. While RF appears to be a good criterion for supertrees, it may not be suitable for datasets with substantial amounts of LGT: a single "long-distance" LGT event between distant taxonomic relatives will result in many discordant bipartitions and a high RF distance. If many organisms participate in long-distance LGT, then "phylogenetic compromise" trees [12] may emerge which reflect neither the correct

Table 1.1: Running time improvement for computing the rooted SPR distance and hybridization number.

	Previous	**New**
Binary SPR distance	$O\left(3^k n\right)$ time [103]	$O\left(2^k n\right)$ time
Multifurcating SPR distance	$O\left(4^k \cdot \text{poly}(n)\right)$ time [97]	$O\left(2.42^k n\right)$ time
Hybridization number	$O\left((18k)^k + n^3\right)$ time [24, 56]	$O\left(3.18^k n\right)$ time

species relationships, nor the dominant pathways of gene sharing. The requirement that all input trees be binary is also potentially limiting, as many relationships in trees inferred from sequence data are unsupported by statistics such as the bootstrap, and should be collapsed into multifurcations.

1.2 Contribution

My contribution is to develop substantially more efficient algorithms for computing MAFs and MAAFs (and, thus, the SPR distance and hybridization number). Using a "shifting lemma" central to Bordewich et al.'s 3-approximation algorithm [22], one can obtain a depth-bounded search algorithm for computing MAFs of binary trees with running time $O\left(3^k n\right)$ and a linear-time 3-approximation algorithm [103]. Building on my prior work, I developed efficient algorithms for computing rooted MAFs of two binary trees, rooted MAAFs of two binary trees, and rooted MAFs of two multifurcating trees. Table 1.1 shows these results in comparison with previous work. The MAF algorithms were implemented and shown to be orders of magnitude faster than previous approaches. The resulting software, RSPR, is available under the open source GPL license [105]. Finally, these algorithms were applied to construct supertrees based on the SPR distance and infer LGT between bacteria. The SPR Supertrees software is also available under the GPL [106].

Chapter 2 introduces the technical notation and definitions used throughout this thesis.

Chapters 3–6 present new theoretical results. In Chapter 3 I present an efficient fixed-parameter algorithm for computing rooted MAFs of two binary trees that runs in $O\left(2.42^k n\right)$ time. By focusing on search paths that are guaranteed to find a solution, this algorithm avoids searching the full space of solutions. A preliminary version of this chapter (without proofs) was presented at the Symposium for Experimental

Algorithms (SEA) [100] and the full version is the first half of an article that will soon appear in the SIAM Journal on Computing [102]. The focus of Chapter 4 is on extending the MAF algorithm to compute MAAFs in $\mathrm{O}\left(3.18^k n\right)$ time. Developing the first bounded search tree algorithm for this problem required significant new insights including an efficient new "refined cycle graph" data structure, a two-phase search procedure with the novel concept of *marking* edges in the first phase to inform the second phase, and a combined running time analyis of these two phases. This chapter is the second half of the SIAM Journal on Computing article [102]. Chapter 5 extends the MAF algorithm to compute rooted MAFs of two multifurcating trees in $\mathrm{O}\left(2.42^k n\right)$ time, matching the time for the binary case, and introduces an $\mathrm{O}(n \log n)$-time 3-approximation for this problem. Again, maintaining extra information between separate branches of the search procedure is the key in this algorithm, allowing it to avoid duplicate subproblems. In Chapter 6 these ideas inspire the new concept of "protecting" edges in some parts of the search and improve the time required to compute binary MAFs to $\mathrm{O}\left(2^k n\right)$.

Chapter 7 presents an efficient implementation of my MAF algorithms, RSPR [105], which is available under the GNU GPL open source license. I demonstrate that my MAF algorithms are orders of magnitude faster than previous approaches and thus applicable to practical datasets. Some of these results were presented at the Symposium for Experimental Algorithms (SEA) [100] and are reproduced here with the kind permission of Springer Science+Business Media.

Chapter 8 demonstrates the applicability of my efficient MAF algorithms to large-scale phylogenetic inference. My SPR Supertrees software [106], also available under the GPL, constructs supertrees by minimizing their SPR distance to a collection of gene trees. Experiments using simulated datasets with LGT show that the SPR approach is more accurate than RF and, for some realistic rates and regimes of LGT, MRP as well. To demonstrate the application of the SPR supertree approach on a dataset in which considerable LGT is expected, a phylogenomic data set of 244 bacteria covering 393,876 genes in 40,631 orthologous sets was used to analyze preferential transfer of genes between bacterial lineages. A highly plausible supertree was reconstructed and the SPR approach identified putative highways of gene sharing.

In Chapter 9, I present concluding remarks and suggest future work.

Chapter 2

Preliminaries

In this chapter, we introduce the definitions and notation used throughout this thesis. Most of these are standard in the literature from [3, 20, 22, 23, 83]. Additional definitions and notation are introduced in following chapters as necessary.

A *(rooted binary phylogenetic) X-tree* is a rooted tree T whose nodes each have zero or two children. The leaves are bijectively labelled with the members of a label set X. As in [20, 22, 23, 83], we augment the tree with a labelled root node whose label is distinct from the labels of all leaves and whose only child is the original root of T; see Figure 2.1(a). In the remainder of this thesis, we consider ρ to be part of X. For a subset V of X, $T(V)$ is the smallest subtree of T that connects all nodes in V; see Figures 2.1(b). The *V-tree induced by* T is the smallest tree $T|V$ that can be obtained from $T(V)$ by suppressing unlabelled nodes with fewer than two children; see Figure 2.1(c). *Suppressing* a node v deletes v and its incident edges; if v is of degree 2 with parent u and child w, u and w are reconnected using a new edge (u, w).

Each non-root internal node v of an X-tree T has an associated *cluster* $\mathcal{C}_T(v)$, defined to be the set of leaf descendants of v. The set of clusters of T is denoted $\mathcal{C}_T$. The Robinson-Foulds (RF) distance metric between two rooted trees is the normalized count of the symmmetric difference between the sets of clusters of the two trees, that is, for two trees T_1 and T_2, the RF distance $d_{RF}(T_1, T_2)$ is

$$d_{RF}(T_1, T_2) = \frac{|\,(\mathcal{C}_{T_1} \setminus \mathcal{C}_{T_2}) \cup (\mathcal{C}_{T_2} \setminus \mathcal{C}_{T_1})\,|}{2}.$$

A *subtree prune-and-regraft* (SPR) operation on an X-tree T cuts an edge $e_x := (x, p_x)$, where p_x denotes the parent of x. This divides T into subtrees T_x and T_{p_x} containing x and p_x, respectively. Then it introduces a new node p'_x into T_{p_x} by subdividing an edge of T_{p_x} and adds an edge (x, p'_x), thereby making x a child of p'_x. Finally, p_x is suppressed. See Figure 2.1(d).

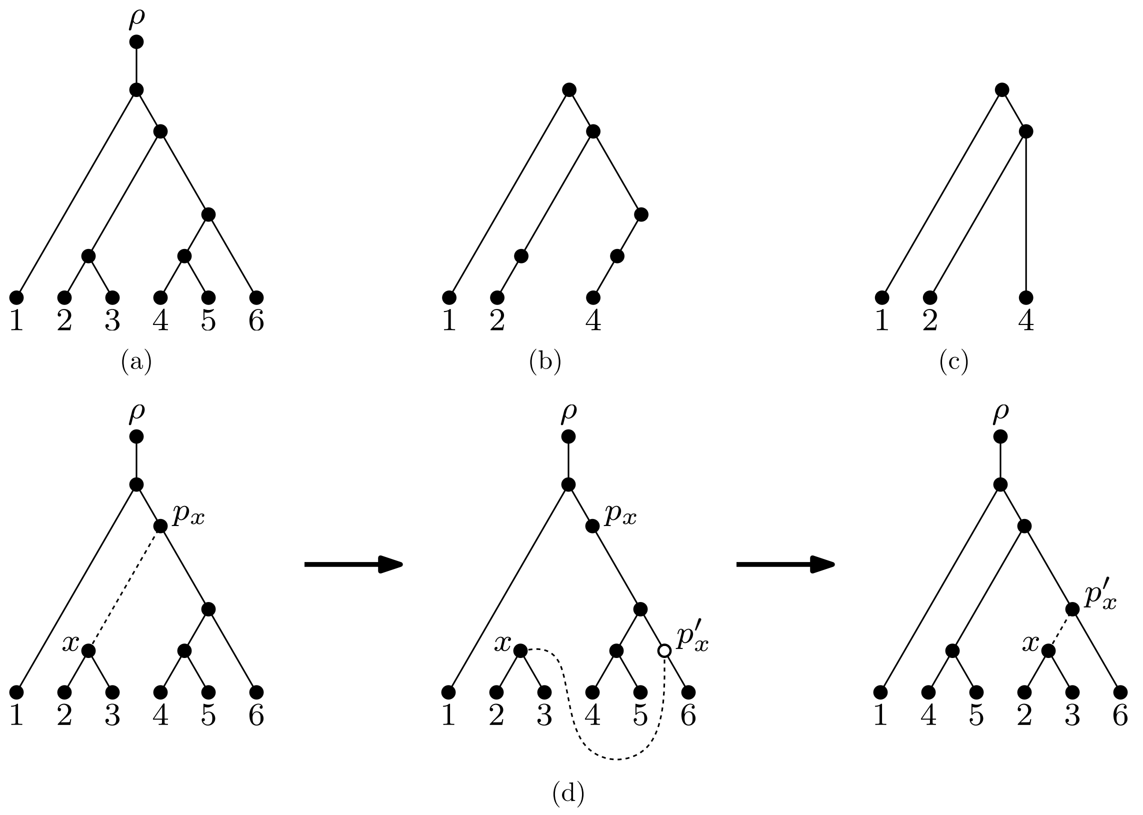


Figure 2.1: (a) An X-tree T. (b) The subtree $T(V)$ for $V = \{1, 2, 4\}$. (c) $T|V$. (d) An SPR operation.

SPR operations give rise to a distance measure $d_{SPR}(\cdot, \cdot)$ between X-trees, defined as the minimum number of SPR operations required to transform one tree into the other. The trees in Figure 2.2(a), for example, have SPR distance $d_{SPR}(T_1, T_2) = 3$.

A related distance measure for X-trees is their *hybridization number*, $hyb(T_1, T_2)$, which is defined in terms of hybrid networks of the two trees. A *hybrid network* of two X-trees T_1 and T_2 is a directed acyclic graph H with a single source ρ, whose sinks are labelled bijectively with the labels in $X \setminus \{\rho\}$, and such that both T_1 and T_2, with their edges directed away from the root, can be obtained from H by deleting edges and suppressing nodes. For a vertex $x \in H$, let $\deg_{\text{in}}(x)$ be its in-degree. Then the hybridization number of T_1 and T_2 is $\min_H \sum_{x \in H, x \neq \rho} (\deg_{\text{in}}(x) - 1)$, where the minimum is taken over all hybrid networks H of T_1 and T_2. This is illustrated in Figure 2.2(c).

These distance measures are related to the sizes of appropriately defined agreement forests. To define these, we first introduce some terminology. For a forest F whose

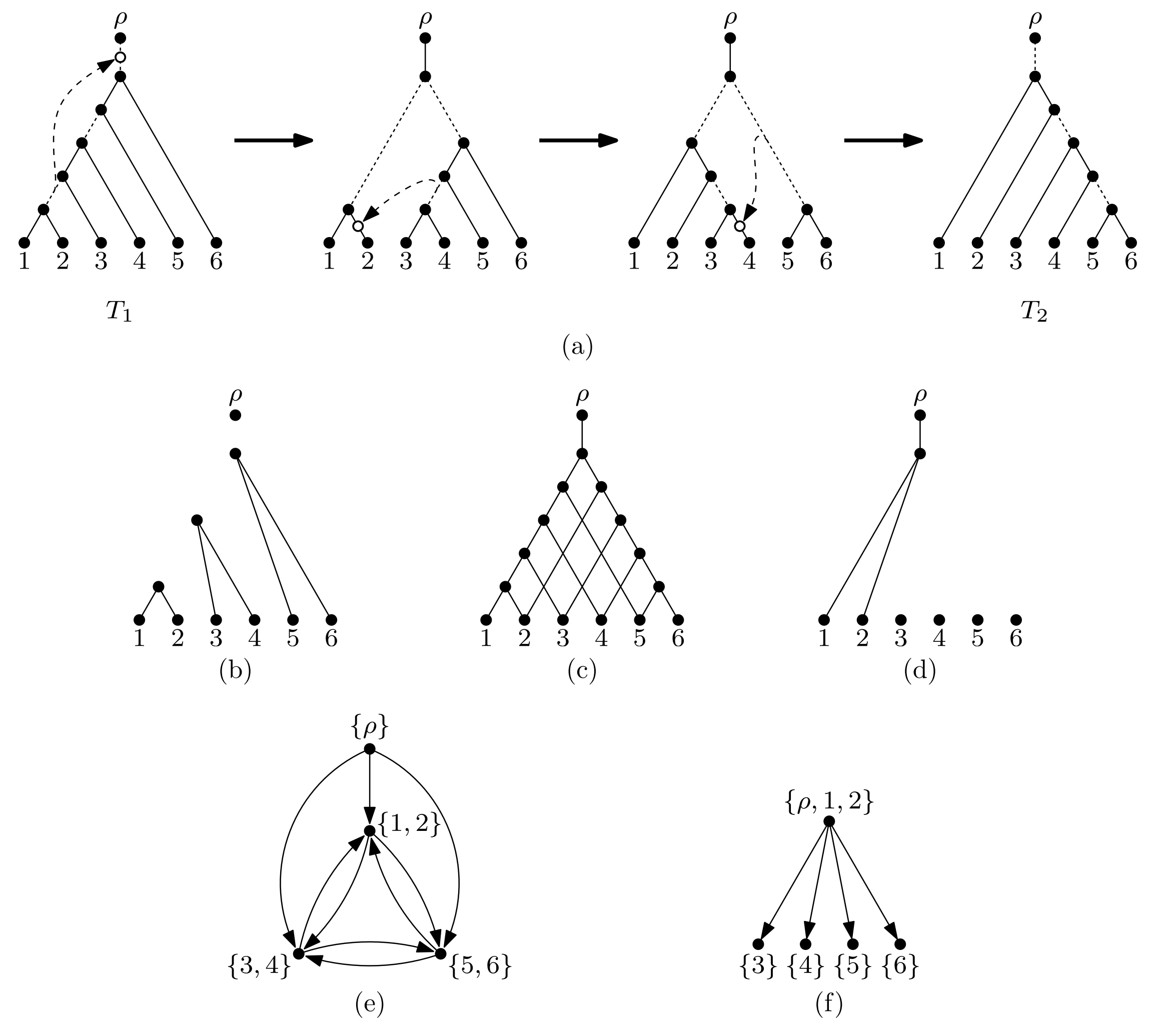


Figure 2.2: (a) SPR operations transforming T_1 into T_2. Each operation changes the top endpoint of one of the dotted edges. (b) The corresponding agreement forest, which can be obtained by cutting the dotted edges in both trees. This is an MAF with 4 components, so $m\left(T_1, T_2\right) = 4$ and $e\left(T_1, T_2, T_2\right) = d_{SPR}\left(T_1, T_2\right) = 3$. Note that this is not an MAAF, as its cycle graph, shown in (e), contains a cycle. (c) A hybrid network of T_1 and T_2. This network has 4 nodes with an extra parent, so the hybridization number is 4. (d) An MAAF of T_1 and T_2. $\tilde{e}\left(T_1, T_2, T_2\right) = hyb\left(T_1, T_2\right) = 4$. Note that this is not an MAF of T_1 and T_2, as it has one more component than the MAF in (b). (e) The cycle graph of the agreement forest in (b), which contains a cycle. (f) The cycle graph of the agreement forest in (d), which does not contain a cycle.

components are rooted phylogenetic trees $T_1, T_2, \ldots, T_k$ with label sets $X_1, X_2, \ldots, X_k$, we say F *yields* the forest with components $T_1|X_1, T_2|X_2, \ldots, T_k|X_k$; if $X_i = \varnothing$, then $T_i(X_i) = \varnothing$ and, hence, $T_i|X_i = \varnothing$. In other words, the forest yielded by F is the smallest forest that can be obtained from F by suppressing unlabelled nodes with less than two children. For a subset E of edges of F, we use $F - E$ to denote the forest obtained by deleting the edges in E from F, and $F \div E$ to denote the forest yielded by $F - E$. We say $F \div E$ is a *forest of* F.

Given X-trees T_1 and T_2 and forests F_1 of T_1 and F_2 of T_2, a forest F is an *agreement forest* (AF) of F_1 and F_2 if it is a forest of both F_1 and F_2. F is a *maximum agreement forest* (MAF) of F_1 and F_2 if there is no AF of F_1 and F_2 with fewer components. We denote the number of components in an MAF of F_1 and F_2 by $m(F_1, F_2)$. For a forest F of F_1 or 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. Bordewich and Semple [23] showed that, for two X-trees T_1 and T_2, $d_{SPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$. An MAF of the trees in Figure 2.2(a) is shown in Figure 2.2(b).

The hybridization number of two X-trees T_1 and T_2 corresponds to an MAF of T_1 and T_2 with an additional constraint. For two forests F_1 and F_2 of T_1 and T_2 and an AF $F = \{C_\rho, C_1, C_2, \ldots, C_k\}$ of F_1 and F_2, we define a *cycle graph* G_F of F. Each node of G_F represents a component of F, and there is an edge from node C_i to node C_j if C_i is an ancestor of C_j in one of the trees. Formally, we map every node $x \in F$ to two nodes $\phi_1(x) \in T_1$ and $\phi_2(x) \in T_2$ by defining $\phi_i(x)$ to be the lowest common ancestor in T_i of all labelled leaves that are descendants of x in F. We refer to $\phi_1(x)$ and $\phi_2(x)$ simply as x in this thesis, except when this creates confusion. For two components C_i and C_j of F with roots r_i and r_j, G_F contains the 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 $\phi_2(r_j)$. We say F is *cyclic* if G_F contains a directed cycle. Otherwise F is an *acyclic* agreement forest (AAF) of F_1 and F_2. A *maximum acyclic agreement forest* (MAAF) of F_1 and F_2 is an AAF with the minimum number of components. We denote its size by $\tilde{m}(F_1, F_2)$ and the number of edges in a forest F of F_1 or F_2 that must be cut to obtain an AAF of F_1 and F_2 by $\tilde{e}(F_1, F_2, F)$. Baroni et al. [7] showed that $hyb(T_1, T_2) = \tilde{e}(T_1, T_2, T_2) = \tilde{m}(T_1, T_2) - 1$. An MAAF of the trees in Figure 2.2(a) is shown in Figure 2.2(d). The cycle graphs for the MAF and MAAF of these trees

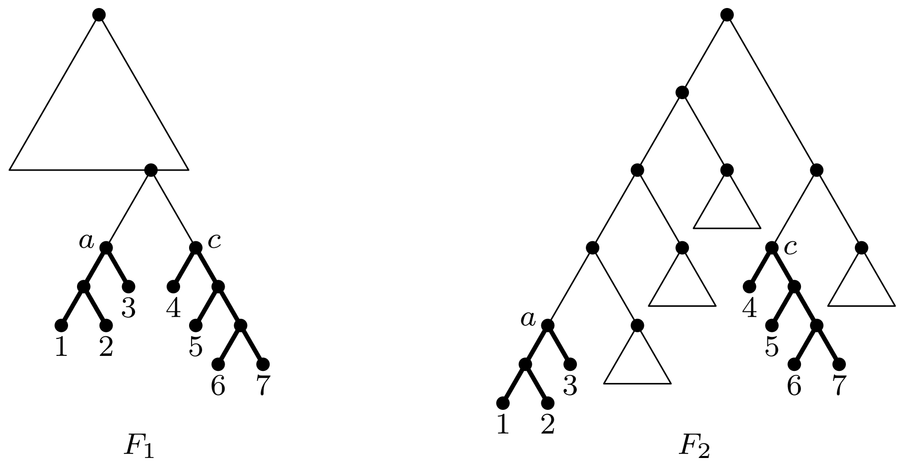


Figure 2.3: A sibling pair (a, c) of two forests F_1 and F_2: a and c have a common parent in F_1, and both a and c exist also in F_2.

shown in Figures 2.2(b) and 2.2(d) are shown in Figures 2.2(e) and 2.2(f).

For two nodes a and b of a forest F, we write $a \sim_F b$ if there exists a path between a and b in F. An *internal node* of a path P in F is a node of P that is not an endpoint of P; a *pendant node* of P is a node not in P and whose parent is an internal node of P. For a node x of a rooted forest F, F^x denotes the subtree of F induced by all descendants of x, including x. 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 is 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, we say (a, c) is a *sibling pair* of F_1 if a and c exist in F_2. Figure 2.3 shows such a sibling pair.

The correctness proofs of our algorithms make use of the following two lemmas. Lemma 2.1 was shown by Bordewich et al. [22] and is illustrated in Figure 2.4. Suppose we cut a set of edges E from a forest F to obtain $F \div E$, and there is an edge e of F such that $F - (E \cup \{e\})$ has a component without labelled nodes. This lemma shows that the forest $F \div (E \setminus \{f\} \cup \{e\})$ obtained by replacing any edge $f \in E$ on the boundary of this "empty" component with e is the same as $F \div E$.

Lemma 2.1 (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 closer to e, and v_e an end vertex of e. If $v_f \sim_{F-E} v_e$ and $x \nsim_{F-(E\cup\{e\})} v_f$, for all $x \in X$, then $F \div E = F \div (E \setminus \{f\} \cup \{e\})$.*

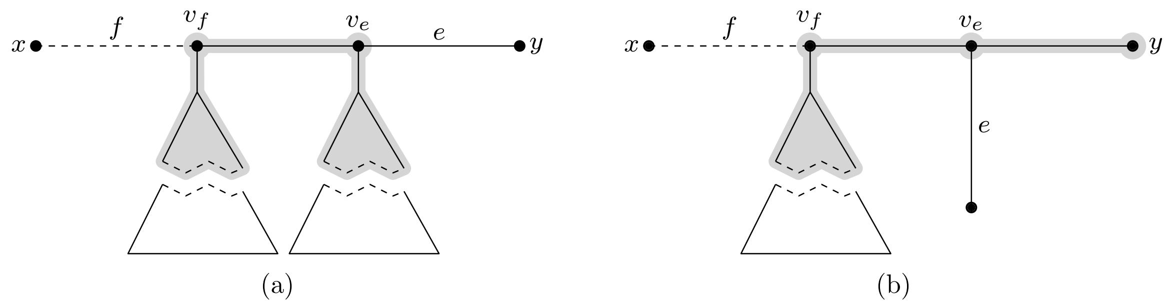


Figure 2.4: Illustration of the Shifting Lemma. (a) The lemma applies because e and f are on the boundary of an "empty" component of $F - (E \cup \{e\})$, shown in grey. (b) The lemma does not apply because the component with e and f on its boundary contains a labelled leaf y: $v_f \sim_{F-(E\cup\{e\})} y$.

Let F_1 and F_2 be forests of X-trees T_1 and T_2, respectively. Any agreement forest of F_1 and F_2 is an agreement forest of T_1 and T_2. Conversely, an agreement forest of T_1 and T_2 is an agreement forest of F_1 and F_2 if it is a forest of F_2 and there are no two leaves a and b such that $a \sim_{F_2} b$ but $a \nsim_{F_1} b$. This is formalized in the following lemma. Our algorithms ensure that any intermediate forests F_1 and F_2 they produce have this latter property. Thus, we can reason about agreement forests of F_1 and F_2 and of T_1 and T_2 interchangeably.

Lemma 2.2. *Let F_1 and F_2 be forests of X-trees T_1 and T_2, respectively. Let F_1 be the union of trees $\dot{T}_1, \dot{T}_2, \dots, \dot{T}_k$ and F_2 be the union of forests $\dot{F}_1, \dot{F}_2, \dots, \dot{F}_k$ such that $\dot{T}_i$ and $\dot{F}_i$ have the same label set, for all $1 \leqslant i \leqslant k$. A forest of F_2 is an AF of T_1 and T_2 if and only if it is an AF of F_1 and F_2.*

A *triple* $ab|c$ of a rooted forest F is defined by a set $\{a, b, c\}$ of three leaves 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. A triple of a forest F_1 is *compatible* with a forest F_2 if it is also a triple of F_2; otherwise it is *incompatible* with F_2. An agreement forest of two forests F_1 and F_2 cannot contain a triple incompatible with either of the two forests. Thus, we have the following observation.

Observation 2.3. *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 triple 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_1 are said to *overlap* in F_2 if there exist leaves $a, b \in C_1$ and $c, d \in C_2$ such that the paths from a to b and from c to d in F_2 exist and are nondisjoint. When considering binary trees, this means the two paths share an edge. The following lemma is an easy extension of a lemma of [22], which states the same result for a tree T_2 instead of a forest F_2.

Lemma 2.4. *Let F_1 and F_2 be forests of two X-trees T_1 and T_2, and denote the label sets of the components of F_1 by $X_1, X_2, \ldots, X_k$ and the label sets of the components of F_2 by $Y_1, Y_2, \ldots, Y_l$. 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 triple of F_2 is incompatible with F_1.*

Chapter 3

Bounding the Search: Computing MAFs of Binary Trees in $\mathrm{O}\left(2.42^k n\right)$-time

In this chapter, we present an algorithm for computing MAFs of two binary rooted X-trees (and, hence, their SPR distance) that runs in $\mathrm{O}\left(2.42^k n\right)$ time. Chapter 6 enhances the algorithm and reduces the running time to $\mathrm{O}\left(2^k n\right)$. A preliminary version of this chapter (without proofs) was presented at the Symposium on Experimental Algorithms (SEA) [100] and the full version is the first half of an article that will soon appear in the SIAM Journal on Computing [102]. It will be obvious from the description of the algorithm that it also produces a corresponding MAF. We do not discuss this further in the remainder of this chapter and focus only on computing $d_{SPR}(T_1, T_2)$.

Using Lemma 2.1, one can obtain a depth-bounded search algorithm for computing the SPR distance with running time $\mathrm{O}\left(3^k n\right)$ [103]. We analyze the structure of rooted agreement forests further and identify three distinct subcases that allow us to improve the algorithm's running time to $\mathrm{O}\left(2.42^k n\right)$. By combinining this result with kernelization rules by Bordewich and Semple [23], we obtain an algorithm with running time $\mathrm{O}\left(2.42^k k + n^3\right)$. We note here that the approach discussed in this chapter also leads to linear-time 3-approximation algorithms for rooted SPR distance and unrooted TBR distance, as well as to an $\mathrm{O}\left(4^k k + n^3\right)$-time algorithm for unrooted TBR distance. Details can be found in [103, 104].

As is customary for FPT algorithms, we focus on the decision version of the problem: "Given two X-trees T_1 and T_2 and a parameter k, is $d_{SPR}(T_1, T_2) \leqslant k$?" To compute the distance between two trees, we start with $k = 0$ 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, as the running time depends exponentially on k and will be determined by the largest examined k. The following theorem states the main result of this chapter.

Theorem 3.1. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left(\left(1+\sqrt{2}\right)^k n\right) = \mathrm{O}\left(2.42^k n\right)$ time to decide whether $e\left(T_1, T_2, T_2\right) \leqslant k$.*

Using reduction rules by Bordewich et al. [23], we can improve the running time in Theorem 3.1 for values of k such that $k \geqslant 2\log_{2.42} n$ and $k = \mathrm{o}\left(n\right)$. Given two trees T_1 and T_2, these reduction rules take $\mathrm{O}\left(n^3\right)$ time to produce two trees T_1' and T_2' of size at most $c \cdot e\left(T_1, T_2, T_2\right)$ each, for some constant $c > 0$ (determined by Bordewich et al.), and such that $e\left(T_1', T_2', T_2'\right) = e\left(T_1, T_2, T_2\right)$. If one of the trees has size greater than ck, then $e\left(T_1, T_2, T_2\right) > k$, and we can answer "no" without any further processing. If both trees have size at most ck, we can apply Theorem 3.1 to T_1' and T_2' to decide in $\mathrm{O}\left(2.42^k k\right)$ time whether $e\left(T_1', T_2', T_2'\right) \leqslant k$. Thus, we obtain the following corollary.

Corollary 3.2. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left(2.42^k k + n^3\right)$ time to decide whether $e\left(T_1, T_2, T_2\right) \leqslant k$.*

In the remainder of this section, we prove Theorem 3.1. Our 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, and decides whether $e\left(T_1, T_2, F_2\right) \leqslant k$. We denote such an invocation by $\textsc{Maf}\left(F_1, F_2, k\right)$. The forest F_1 is the union of a tree $\dot{T}_1$ and a forest F disjoint from $\dot{T}_1$, while F_2 is the union of the same forest F and another forest $\dot{F}_2$ with the same label set as $\dot{T}_1$. We maintain two sets of labelled nodes: R_d (roots-done) contains the roots of F, 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, $\textsc{Maf}\left(T_1, T_2, k\right)$, $F_1 = \dot{T}_1 = T_1$, $F_2 = \dot{F}_2 = T_2$, and $F = \varnothing$; R_d is empty, and R_t contains all leaves of T_1.

$\textsc{Maf}\left(F_1, F_2, k\right)$ identifies a small collection $\{E_1, E_2, \ldots, E_q\}$ of subsets of edges of $\dot{F}_2$ such that $e\left(T_1, T_2, F_2\right) \leqslant k$ if and only if $e\left(T_1, T_2, F_2 \div E_i\right) \leqslant k - |E_i|$, for at least one $1 \leqslant i \leqslant q$. It makes a recursive call $\textsc{Maf}\left(F_1, F_2 \div E_i, k - |E_i|\right)$, for each subset E_i, and returns "yes" if and only if one of these calls does. The steps of this procedure are as follows.

1. (Failure) If $k < 0$, there is no subset E of at most k edges of F_2 such that $F_2 - E$ yields an AF of T_1 and T_2: $e\left(T_1, T_2, F_2\right) \geqslant 0 > k$. Return "no" in this case.

2. (Success) If $|R_t| \leqslant 2$, then $\dot{F}_2 \subseteq \dot{T}_1$. Hence, $F_2 = \dot{F}_2 \cup F$ is an AF of F_1 and F_2 and, by Lemma 2.2, also of T_1 and T_2. Thus, $e\left(T_1, T_2, F_2\right) = 0 \leqslant k$. Return "yes" in this case.

3. (Prune maximal agreeing subtrees) If there is a node $r \in R_t$ that is a root in $\dot{F}_2$, remove r from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F; cut the edge e_r in $\dot{T}_1$ and suppress r's parent in $\dot{T}_1$; return to Step 2. This does not alter F_2 and, thus, neither $e\left(T_1, T_2, F_2\right)$. If no such root r exists, proceed to Step 4.

4. Choose a sibling pair (a, c) in $\dot{T}_1$ such that $a, c \in R_t$.

5. (Grow agreeing subtrees) If (a, c) is a sibling pair of $\dot{F}_2$, remove a and c from R_t; label their parent in both forests with (a, c) and add it to R_t; return to Step 2. If (a, c) is not a sibling pair of $\dot{F}_2$, proceed to Step 6.

6. (Cut edges) Distinguish three cases (see Figure 3.1):

   6.1. If $a \nsim_{F_2} c$, call $\textsc{Maf}\left(F_1, F_2 \div \{e_a\}, k-1\right)$ and $\textsc{Maf}\left(F_1, F_2 \div \{e_c\}, k-1\right)$ recursively.

   6.2. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has only one pendant node b, call $\textsc{Maf}\left(F_1, F_2 \div \{e_b\}, k-1\right)$ recursively.

   6.3. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$, call $\textsc{Maf}\left(F_1, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\right)$, $\textsc{Maf}\left(F_1, F_2 \div \{e_a\}, k-1\right)$, and $\textsc{Maf}\left(F_1, F_2 \div \{e_c\}, k-1\right)$ recursively.

   Return "yes" if one of the recursive calls does; otherwise return "no".

To prove that the algorithm achieves the running time stated in Theorem 3.1, we show that each invocation takes linear time (Lemma 3.3) and that the algorithm makes $\mathrm{O}\left(\left(1+\sqrt{2}\right)^k\right)$ recursive calls (Lemma 3.4). The correctness of the algorithm will be shown in Lemmas 3.6, 3.7 and 3.8.

Lemma 3.3. *Each invocation* $\textsc{Maf}\left(F_1, F_2, k\right)$*, excluding recursive calls it makes, takes linear time.*

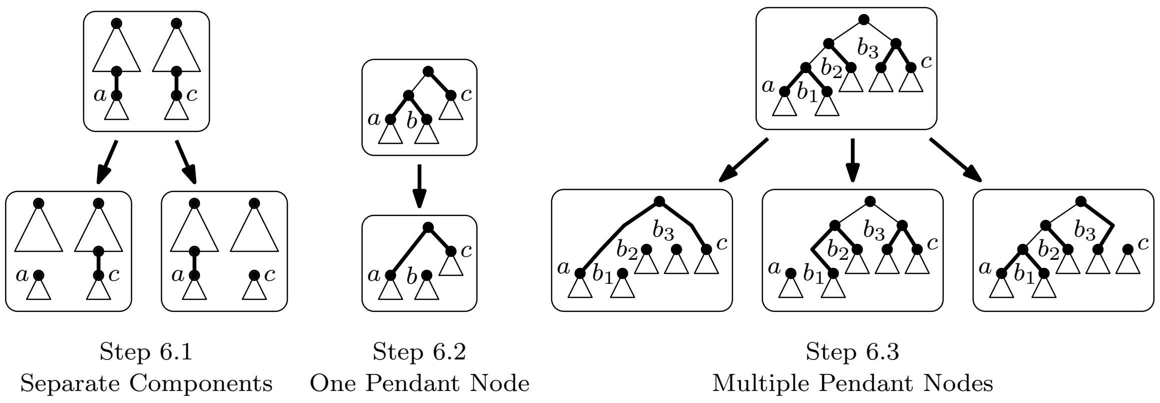


Figure 3.1: The cases in Step 6 of the MAF algorithm. Only $\dot{F}_2$ is shown. Each box represents a recursive call.

Proof. We represent each forest as a collection of nodes, each of which points to its parent, left child, and right child. In addition, every labelled node (i.e., each node in R_t or R_d) stores a pointer to its counterpart in the other forest. For $\dot{T}_1$, we maintain a list of sibling pairs of labelled nodes. Every labelled node of $\dot{T}_1$ stores a pointer to the pair it belongs to, if any. For $\dot{F}_2$, we maintain a list $R'_d \subseteq R_t$ of nodes that are roots of $\dot{F}_2$. This list is used to move these roots from R_t to R_d in Step 3.

It is easily verified that, using this representation of F_1 and F_2, each execution of Steps 1–5 takes constant time and that Step 6, excluding recursive calls it spawns, takes linear time. Steps 1 and 6 are executed only once per invocation. Steps 2–5 form a loop, and each iteration, except the first one, is the result of finding a root of $\dot{F}_2$ in Step 3 or merging a sibling pair in Step 5. In the former case, Step 3 cuts an edge in F_1, which can happen only $\mathrm{O}(n)$ times because F_1 has $\mathrm{O}(n)$ edges. In the latter case, the number of nodes in R_t decreases by one, which cannot happen more than n times because the algorithm starts with the n leaves of T_1 in R_t and the number of nodes in R_t never increases. Thus, Steps 2–5 are executed $\mathrm{O}(n)$ times, and the cost of the entire invocation is linear. $\square$

Lemma 3.4. *An invocation* $\textsc{Maf}(F_1, F_2, k)$ *spawns* $\mathrm{O}\left(\left(1+\sqrt{2}\right)^k\right)$ *recursive calls.*

Proof. Let $I(k)$ be the number of recursive calls spawned by an invocation with parameter k. By inspecting the different cases of Step 6, we obtain

$$\begin{aligned} I(k) &= \begin{cases} 1 & \text{if Step 6 is not executed} \\ 1 + 2I(k-1) & \text{Case 6.1} \\ 1 + I(k-1) & \text{Case 6.2} \\ 1 + 2I(k-1) + I(k-q) & \text{Case 6.3} \end{cases} \\ &\leqslant 1 + 2I(k-1) + I(k-2) \end{aligned}$$

because Case 6.3 dominates the other two cases and $q \geqslant 2$ in this case. Simple substitution shows that this recurrence solves to $I(k) = \mathrm{O}\left(\left(1+\sqrt{2}\right)^k\right)$. □

It remains to prove the correctness of the algorithm, which we do by induction on k. An invocation $\textsc{Maf}\,(F_1, F_2, k)$ with $k < 0$ correctly returns "no" in Step 1, so assume $k \geqslant 0$. In this case, the invocation produces its answer in Step 2 or 6. If it produces its answer ("yes") in Step 2, this is correct because F_2 is an MAF of T_1 and T_2. If it produces its answer in Step 6, it suffices to prove that $e\,(T_1, T_2, F_2) \leqslant k$ if and only if $e\,(T_1, T_2, F_2 \div E_i) \leqslant k - |E_i|$, for at least one of the recursive calls $\textsc{Maf}\,(F_1, F_2 \div E_i, k - |E_i|)$ the invocation makes in Step 6. This in turn follows if $e\,(T_1, T_2, F_2 \div E_i) \geqslant e\,(T_1, T_2, F_2) - |E_i|$, for all recursive calls $\textsc{Maf}\,(F_1, F_2 \div E_i, k - |E_i|)$, which is trivial, and $e\,(T_1, T_2, F_2 \div E_i) = e\,(T_1, T_2, F_2) - |E_i|$, for at least one recursive call $\textsc{Maf}\,(F_1, F_2 \div E_i, k - |E_i|)$. Lemmas 3.6, 3.7, and 3.8 below prove the latter for each case of Step 6. For Cases 6.1 and 6.3, we prove also that $\tilde{e}\,(T_1, T_2, F_2 \div E_i) = \tilde{e}\,(T_1, T_2, F_2) - |E_i|$, for at least one recursive call $\textsc{Maf}\,(F_1, F_2 \div E_i, k - |E_i|)$. This will be used in the correctness proof of the MAAF algorithm in Chapter 4.

In Step 6, (a, c) is a sibling pair of $\dot{T}_1$ but not of F_2—otherwise Step 5 would have replaced a and c with their parent in R_t—and neither F_2^a nor F_2^c is a component of F_2—otherwise Step 3 would have removed a or c from R_t. Note that a and c belong to $\dot{F}_2$ because $\dot{T}_1$ and $\dot{F}_2$ have the same label set. Let b be a's sibling in F_2. If a and c belong to the same component of F_2, we assume w.l.o.g. that a's distance from the root of this component is no less than c's. Since a and c are not siblings in F_2, this implies that $c \notin F_2^b$. If $a \nsim_{F_2} c$, we also have $c \notin F_2^b$ because $a \sim_{F_2} b$.

Our first lemma shows that we can always cut one of e_a, e_b, and e_c to make progress towards an MAF or MAAF of T_1 and T_2 in Step 6. In [103], we used this as a basis for a simple $\mathrm{O}\left(3^k n\right)$-time MAF algorithm. Here, we need this lemma as a basis for the proofs of Lemmas 3.6, 3.7, 3.8, and 4.1.

Lemma 3.5. *If (a, c) is a sibling pair of F_1 and (i) $a \nsim_{F_2} c$ and neither F_2^a nor F_2^c is a component of F_2 or (ii) $a \sim_{F_2} c$ but a and c are not siblings in F_2, then there exists an edge set E of size $e(T_1, T_2, F_2)$ (resp. $\tilde{e}(T_1, T_2, F_2)$) and such that $F_2 \div E$ is an AF (resp. AAF) of T_1 and T_2 and $E \cap \{e_a, e_b, e_c\} \neq \varnothing$.*

Proof. Consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AF of F_1 and F_2, and assume E contains the maximum number of edges from $\{e_a, e_b, e_c\}$ among all edge sets satisfying these conditions. Assume for the sake of contradiction that $E \cap \{e_a, e_b, e_c\} = \varnothing$.

If $a' \nsim_{F_2 - E} a$, for all leaves $a' \in F_2^a$, then we choose an arbitrary such leaf $a' \in F_2^a$ and the first edge f on the path from a to a'. Lemma 2.1 now implies that $F_2 - E$ and $F_2 - (E \setminus \{f\} \cup \{e_a\})$ yield the same forest, which contradicts our choice of E. The same argument leads to a contradiction if $b' \nsim_{F_2 - E} b$, for all leaves $b' \in F_2^b$, or $c' \nsim_{F_2 - E} c$, for all leaves $c' \in F_2^c$. Thus, there exist leaves $a' \in F_2^a$, $b' \in F_2^b$, and $c' \in F_2^c$ such that $a' \sim_{F_2 - E} a$, $b' \sim_{F_2 - E} b$, and $c' \sim_{F_2 - E} c$.

Since (a, c) is a sibling pair of $\dot{T}_1$, $a'c'|b'$ is a triple of F_1, while $c \notin F_2^b$ implies that either $a'b'|c'$ is a triple of F_2 or $a' \nsim_{F_2} c'$. In either case, the triple $a'c'|b'$ is incompatible with F_2 and, by Observation 2.3 and because $a' \sim_{F_2 - E} b'$, we have $a' \nsim_{F_2 - E} c'$ and, hence, $a'' \nsim_{F_2 - E} c'$, for every leaf $a'' \in F_2^a$. Now, if there existed a leaf $x \notin F_2^c$ such that $c' \sim_{F_2 - E} x$, then the components of $F_2 \div E$ containing a' and c' would overlap in F_1: they would both include e_{p_a} because $b', x \notin F_1^{p_a}$. By Lemma 2.4, this would contradict that $F_2 \div E$ is an AF of F_1 and F_2. Thus, no such leaf x exists. On the other hand, since F_2^c is not a component of F_2, there exists a leaf $x \notin F_2^c$ such that $c' \sim_{F_2} x$. Since $x \nsim_{F_2 - E} c'$, at least one edge on the path from c' to x belongs to E. Let f be the first such edge. Since $c \sim_{F_2 - E} c'$, f does not belong to F_2^c. Hence, edges e_c and f satisfy the conditions of Lemma 2.1, and $F_2 - E$ and $F_2 - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest, contradicting the choice of E.

The second claim of the lemma follows using the same arguments after choosing E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E$ is an AAF of T_1 and T_2. $\square$

The last three lemmas of this chapter now establish the correctness of each case in Step 6 of the algorithm and conclude the proof of Theorem 3.1.

Lemma 3.6 (Case 6.1—Separate Components). *If (a, c) is a sibling pair of F_1, $a \nsim_{F_2} c$, and neither F_2^a nor F_2^c is a component of F_2, then there exists an edge set E of size $e(T_1, T_2, F_2)$ (resp. $\tilde{e}(T_1, T_2, F_2)$) and such that $F_2 \div E$ is an AF (resp. AAF) of T_1 and T_2 and $E \cap \{e_a, e_c\} \neq \varnothing$.*

Proof. Consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AF of F_1 and F_2, and assume E contains the maximum number of edges from $\{e_a, e_c\}$ among all edge sets satisfying these conditions. Assume for the sake of contradiction that $E \cap \{e_a, e_c\} = \varnothing$.

By the arguments in the proof of Lemma 3.5, there exist leaves $a' \in F_2^a$ and $c' \in F_2^c$ such that $a' \sim_{F_2 - E} a$ and $c' \sim_{F_2 - E} c$. Since (a, c) is a sibling pair of F_1 but $a \nsim_{F_2} c$ and, hence, $a' \nsim_{F_2 - E} c'$, we must have $a' \nsim_{F_2 - E} x$, for every leaf $x \notin F_2^a$, or $c' \nsim_{F_2 - E} x$, for every leaf $x \notin F_2^c$. W.l.o.g. assume the latter. As shown in the proof of Lemma 3.5, this implies that $F_2 - E$ and $F_2 - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest, where f is the first edge on the path from c' to a leaf $x \notin F_2^c$ and such that $c' \sim_{F_2} x$. This contradicts the choice of E.

The second claim of the lemma follows using the same arguments after choosing E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E$ is an AAF of T_1 and T_2. $\square$

Lemma 3.7 (Case 6.2—One Pendant Node—MAF). *If (a, c) is a sibling pair of F_1, $a \sim_{F_2} c$, and the path from a to c in F_2 has only one pendant node b, then there exists an edge set E of size $e(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AF of T_1 and T_2 and $e_b \in E$.*

Proof. Again, consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AF of F_1 and F_2, and assume E contains the maximum number of edges from $\{e_a, e_b, e_c\}$ among all edge sets satisfying these conditions. By Lemma 3.5, $E \cap \{e_a, e_b, e_c\} \neq \varnothing$. If $e_b \in E$, there is nothing to prove, so assume $e_b \notin E$. Let $v = p_a = p_b$, and $u = p_v = p_c$.

If $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ is an AF of F_1 and F_2, we are done because $E \cap \{e_a, e_c\} \neq \varnothing$ and, hence, $|E \setminus \{e_a, e_c, e_v\} \cup \{e_b\}| \leqslant |E| = e(T_1, T_2, F_2)$. So assume $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ is not an AF of F_1 and F_2. We prove that $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ is an AF of F_1 and F_2 and that $|E \cap \{e_a, e_c, e_v\}| \geqslant 2$ in this case. The latter implies that $|E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\}| \leqslant |E|$, that is, $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ is an MAF of F_1 and F_2.

If $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ is not an AF of F_1 and F_2, then either two of its components overlap in F_1 or it contains a triple incompatible with F_1. First consider the case of overlapping components. Observe that $F_2 \div (E \cup \{e_b\})$ is an AF of F_1 and F_2 because it is a refinement of $F_2 \div E$. The only component of $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ that is not a component of $F_2 \div (E \cup \{e_b\})$ is the one containing a and c. Call this component C. Thus, if two components of $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ overlap in F_1, one of them must be C. Call the other component C'. For any two leaves x and y in C such that $x, y \notin F_1^{p_a}$, the path P between x and y also exists in $F_2 \div (E \cup \{e_b\})$ and, thus, cannot overlap C'. Thus, w.l.o.g. $x \in F_1^{p_a}$. Now, if the edge e shared by P and C' belonged to $F_1^{p_a}$, P and C' would also overlap in F_2 because $F_1^{p_a}$ is the same as the subtree of $F_2 \div \{e_b\}$ with root u. This, however, is impossible because $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ is a forest of F_2. Thus, the edge e shared by P and C' cannot belong to $F_1^{p_a}$, and we have $y \notin F_1^{p_a}$. This implies that the path from x' to y, for any leaf $x' \in F_2^a \cup F_2^c$, includes e. Therefore, since $F_2 \div (E \cup \{e_b\})$ is an AF of F_1 and F_2, we have $x' \nsim_{F_2 - (E \cup \{e_b\})} y$, for every leaf $x' \in F_2^a \cup F_2^c$. Since $x \sim_{F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})} y$, the path from u to y in F_2 contains no edge in E. Thus, since $x' \nsim_{F_2 - (E \cup \{e_b\})} y$, for all leaves $x' \in F_2^a \cup F_2^c$, the choice of E and Lemma 2.1 imply that E must include e_c and at least one of e_a or e_v, that is, $|E \cap \{e_a, e_c, e_v\}| \geqslant 2$.

In $F_2 - (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$, C is split into two components $C_1 = C \cap F_2^u$ and $C_2 = C \setminus F_2^u$. All other components are the same as in $F_2 - (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$. Since $x, y \in F_1^{p_a}$, for all leaves $x, y \in C_1$, and $x, y \notin F_1^{p_a}$, for all leaves $x, y \in C_2$, the same argument as in the previous paragraph shows that neither C_1 nor C_2 overlaps a component $C' \notin \{C_1, C_2\}$. C_1 and C_2 do not overlap either because $C_1 \subseteq F_1^{p_a}$ and $C_2 \cap F_1^{p_a} = \varnothing$. Thus, no two components of $F_2 - (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ overlap in F_1.

Now assume $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ contains a triple incompatible with F_1. Then, once again, this triple has to be part of C and must involve a leaf in $F_2^a \cup F_2^c$ and a leaf not in $F_2^a \cup F_2^c$ because any other triple is either a triple of $F_2 \div (E \cup \{e_b\})$ or a triple of $F_1^{p_a}$; in either case, it is a triple of F_1. $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ cannot contain a triple with one leaf in $F_2^a \cup F_2^c$ and one leaf not in $F_2^a \cup F_2^c$ because the path between any two such leaves includes e_u. Thus, $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ contains no triples incompatible with F_1. Since we have just shown that no two components of $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ overlap in F_1, $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\})$ is an AF of F_1 and F_2.

It remains to prove that $|E \cap \{e_a, e_c, e_v\}| \geqslant 2$ if $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$ contains a triple $xy|z$ incompatible with F_1. Since this triple needs to involve a leaf in $F_2^a \cup F_2^c$ and one not in $F_2^a \cup F_2^c$, we have (i) $x, y \in F_2^a \cup F_2^c$ and $z \notin F_2^a \cup F_2^c$, (ii) $x \in F_2^a \cup F_2^c$ and $y, z \notin F_2^a \cup F_2^c$ or (iii) $x, y \notin F_2^a \cup F_2^c$ and $z \in F_2^a \cup F_2^c$. The first case cannot arise because $x, y \in F_1^{p_a}$ and $z \notin F_1^{p_a}$ in this case, that is, $xy|z$ is also a triple of F_1.

In the second case, assume for the sake of contradiction that $|E \cap \{e_a, e_c, e_v\}| = 1$, and assume w.l.o.g. that $x \in F_2^a$. Since every triple $x'y|z$ with $x' \in F_2^a$ would also be incompatible with F_1, $F_2 \div (E \setminus \{e_b\})$ cannot contain such a triple. Hence, the choice of E and Lemma 2.1 imply that $E \cap \{e_a, e_v\} \neq \varnothing$ and, therefore, $e_c \notin E$. As in the proof of Lemma 3.5, this implies that there exists a leaf $c' \in F_2^c$ such that $c' \sim_{F_2 - E} c \sim_{F_2 - E} u$, by the choice of E and Lemma 2.1. Since $xy|z$ is a triple of $F_2 \div (E \setminus \{e_a, e_c, e_v\} \cup \{e_b\})$, we have $y \sim_{F_2 - E} u \sim_{F_2 - E} z$. Hence, $c'y|z$ is a triple of $F_2 \div E$ and this triple is incompatible with F_1 because $xy|z$ is, $x, c' \in F_1^{p_a}$, and $y, z \notin F_1^{p_a}$. This is a contradiction, that is, $|E \cap \{e_a, e_c, e_v\}| \geqslant 2$.

In the last case, if we assume w.l.o.g. that $z \in F_2^a$, an analogous argument as for the second case shows that, if $|E \cap \{e_a, e_c, e_v\}| = 1$, then $F_2 \div E$ contains a triple $xy|c'$ with $c' \in F_2^c$. This triple is incompatible with F_1, which is again a contradiction. $\square$

Lemma 3.8 (Case 6.3—Multiple Pendant Nodes). *If (a, c) is a sibling pair of F_1, $a \sim_{F_2} c$, and the path from a to c in F_2 has $q \geqslant 2$ pendant nodes $b_1, b_2, \dots, b_q$, then there exists an edge set E of size $e(T_1, T_2, F_2)$ (resp. $\tilde{e}(T_1, T_2, F_2)$) and such that $F_2 \div E$ is an AF (resp. AAF) of T_1 and T_2 and either $E \cap \{e_a, e_c\} \neq \varnothing$ or $\{e_{b_1}, e_{b_2}, \dots, e_{b_q}\} \subseteq E$.*

Proof. We prove the lemma by induction on q. For $q = 1$, the claim holds by Lemma 3.5, so assume $q > 1$ and the claim holds for $q - 1$. Assume further that b_1 is the sibling of a. By Lemma 3.5, there exists a set E'' of size $e(T_1, T_2, F_2)$ and such that $F_2 \div E''$ is an AF of F_1 and F_2 and $E'' \cap \{e_a, e_{b_1}, e_c\} \neq \varnothing$. If $E'' \cap \{e_a, e_c\} \neq \varnothing$, we are done. Otherwise $e_{b_1} \in E''$ and $e(T_1, T_2, F_2') = e(T_1, T_2, F_2) - 1$, where $F_2' := F_2 \div \{e_{b_1}\}$. In F_2', the path from a to c has $q-1$ pendant nodes, namely $b_2, b_3, \ldots, b_q$. Thus, by the induction hypothesis, there exists an edge set E' of size $e(T_1, T_2, F_2')$ and such that $F_2' \div E'$ is an AF of F_1 and F_2' and $E' \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_2}, e_{b_3}, \ldots, e_{b_q}\} \subseteq E'$. The set $E := E' \cup \{e_{b_1}\}$ has size $|E'| + 1 = e(T_1, T_2, F_2)$, $F_2 \div E = F_2' \div E'$ is an AF of F_1 and F_2, and either $E \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\} \subseteq E$.

The second claim of the lemma follows using the same arguments, since Lemma 3.5 holds for both AF and AAF. $\square$

Chapter 4

Analyzing and Avoiding Cycles: Computing MAAFs of Binary Trees in $\mathrm{O}\left(3.18^k n\right)$-time

In this chapter, we present our algorithm for computing the hybridization number of two X-trees. This is the second half of an article that will soon appear in the SIAM Journal on Computing [102]. As in Chapter 3, we focus on deciding whether $hyb\,(T_1, T_2) \leqslant k$, as $hyb\,(T_1, T_2)$ can be computed by trying increasing values of k and this does not increase the running time by more than a constant factor. Also as in Chapter 3, it will be obvious from the description of our algorithm that it produces a corresponding AAF when it answers "yes".

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 G_F. This suggests the following strategy to decide whether $hyb\,(T_1, T_2) \leqslant k$: We modify the MAF algorithm from Chapter 3 called with parameter k. Note that this algorithm may find AFs that are not maximum when $k > d_{SPR}\,(T_1, T_2)$, so we do not restrict our search to refinements of MAFs. For every invocation $\textsc{Maf}\,(F_1, F_2, k'')$ of the algorithm that would return "yes" in Step 2, F_2 is an AF of T_1 and T_2 obtained by cutting $k' := k - k''$ edges. F_2 may not be an AAF of T_1 and T_2, but it may be possible to break all cycles in G_{F_2} by cutting at most k'' additional edges, in which case $hyb\,(T_1, T_2) \leqslant k' + k'' = k$. Thus, instead of unconditionally returning "yes" in Step 2, we invoke a second algorithm $\textsc{Refine}\,(F_2, k)$, which decides whether F_2 can be refined to an AAF of T_1 and T_2 with at most $k + 1$ components, and return its answer. We use $\textsc{Maaf}\,(F_1, F_2, k'')$ to denote an invocation of this modified MAF algorithm. We refer to the part of the algorithm consisting of these invocations $\textsc{Maaf}\,(F_1, F_2, k'')$ as the *branching phase* of the algorithm and to the part that consists of the invocations $\textsc{Refine}\,(F_2, k)$ as the *refinement phase*. We also refer to a single invocation $\textsc{Refine}\,(F_2, k)$ as a *refinement step*. Note that this is not a linear process—our algorithm performs a refinement step for each agreement forest it finds and thus cycles between the branching phase and refinement phase.

Now let us call an invocation $\textsc{Maaf}\,(F_1, F_2, k'')$ *viable* if there exists an MAAF F of T_1 and T_2 that is a forest of F_2. Below we show how to ensure that there exists a viable invocation $\textsc{Maaf}\,(F_1, F_2, k'')$ such that F_2 is an (not necessarily maximum) AF of T_1 and T_2 if $hyb\,(T_1, T_2) \leqslant k$. The invocation $\textsc{Refine}\,(F_2, k)$ made by $\textsc{Maaf}\,(F_1, F_2, k'')$ returns "yes", so the whole algorithm returns "yes" in this case. If on the other hand $hyb\,(T_1, T_2) > k$, the algorithm either fails to find an AF of T_1 and T_2 with at most $k + 1$ components or none of the AFs it finds can be refined to an AAF with at most $k + 1$ components. Thus, it returns "no" in this case. In either case, the algorithm produces the correct answer.

So assume $hyb\,(T_1, T_2) \leqslant k$. We prove that every viable invocation $\textsc{Maaf}\,(F_1, F_2, k'')$ such that F_2 is not an AF of T_1 and T_2 has a viable child invocation. This immediately implies that there exists a viable invocation $\textsc{Maaf}\,(F_1, F_2, k'')$ such that F_2 is an AF of T_1 and T_2 because the top-level invocation $\textsc{Maaf}\,(T_1, T_2, k)$ is trivially viable and the number of invocations the algorithm makes is finite. If F_2 is not an AF of T_1 and T_2 in a viable invocation $\textsc{Maaf}\,(F_1, F_2, k'')$, this invocation applies one of Cases 6.1–6.3. If it applies Case 6.1 or 6.3, Lemmas 3.6 and 3.8 show that one of its child invocations is viable. In Case 6.2, on the other hand, the child invocation $\textsc{Maaf}\,(F_1, F_2 \div \{e_b\}, k'' - 1)$ is not guaranteed to be viable. The next lemma shows that either $\textsc{Maaf}\,(F_1, F_2 \div \{e_b\}, k'' - 1)$ or $\textsc{Maaf}\,(F_1, F_2 \div \{e_c\}, k'' - 1)$ is a viable invocation in this case. Thus, we modify the algorithm to make two invocations $\textsc{Maaf}\,(F_1, F_2 \div \{e_b\}, k'' - 1)$ and $\textsc{Maaf}\,(F_1, F_2 \div \{e_c\}, k'' - 1)$ in Case 6.2. Even with two recursive calls made in Case 6.2, the recurrence bounding the number of recursive calls made by the algorithm in the proof of Lemma 3.4 remains dominated by Case 6.3. Thus, the algorithm continues to make $\mathrm{O}\left(2.42^k\right)$ recursive calls.

Lemma 4.1 (Case 6.2—One Pendant Node—MAAF). *If (a, c) is a sibling pair of F_1, $a \sim_{F_2} c$, and the path from a to c in F_2 has only one pendant node b, then there exists an edge set E of size $\tilde{e}\,(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AAF of T_1 and T_2 and $E \cap \{e_b, e_c\} \neq \varnothing$.*

Proof. Let E' be an edge set of size $\tilde{e}\,(T_1, T_2, F_2)$ and such that $F_2 \div E'$ is an AAF of T_1 and T_2. Assume further that there is no such set containing more edges from $\{e_a, e_b, e_c\}$ than E' and that b is a's sibling in F_2. By Lemma 3.5, $E' \cap \{e_a, e_b, e_c\} \neq \varnothing$.

If $E' \cap \{e_b, e_c\} \neq \emptyset$, we are done. So assume $E' \cap \{e_b, e_c\} = \emptyset$ and, hence, $e_a \in E'$. As in the proof of Lemma 3.7, let $v = p_a = p_b$ and $u = p_c = p_v$. If $\{e_a, e_v\} \subseteq E'$, Lemma 2.1 implies that we can replace e_v with e_b in E' without changing $F_2 \div E'$. This contradicts the choice of E', so $e_v \notin E'$. As in the proof of Lemma 3.5, the choice of E' and Lemma 2.1 imply that there exist leaves $b' \in F_2^b$ and $c' \in F_2^c$ such that $b' \sim_{F_2 - E'} b$ and $c' \sim_{F_2 - E'} c$ because $E' \cap \{e_b, e_c\} = \emptyset$. Now let $E := E' \setminus \{e_a\} \cup \{e_b\}$. We have $|E| = |E'| = \tilde{e}(T_1, T_2, F_2)$ and $e_b \in E$. Moreover, since $E' \cap \{e_a, e_c, e_v\} = \{e_a\}$, the proof of Lemma 3.7 shows that $F_2 \div E$ is an AF of T_1 and T_2. Next we show that $F_2 \div E$ is acyclic.

Since $F_2 \div E$ and $F_2 \div E'$ are agreement forests of T_1 and T_2, the mapping $\phi_1(\cdot)$ maps each node of these two forests to a corresponding node in T_1. However, a node $x \in F_2$ that belongs to both $F_2 \div E$ and $F_2 \div E'$ may map to different nodes in T_1 if it has different sets of labelled descendant leaves in $F_2 \div E$ and $F_2 \div E'$. For the remainder of this proof, we use $\phi_1(x)$ to denote the node in T_1 a node $x \in F_2$ maps to based on its labelled descendant leaves in $F_2 - E$, and $\phi_1'(x)$ to denote the node it maps to based on its labelled descendant leaves in $F_2 - E'$.

Now assume for the sake of contradiction that $F_2 \div E$ is not acyclic, and let O be a cycle of $G_{F_2 \div E}$. We assume O is as short as possible, which implies in particular that O contains every component of $F_2 \div E$ at most once and that for any three consecutive components C_i, C_{i+1}, and C_{i+2} in O either C_i is an ancestor of C_{i+1} in T_1 and C_{i+1} is an ancestor of C_{i+2} in T_2 or vice versa. Since $F_2 \div E'$ is acyclic, the root r of at least one component in O either is not a root in $F_2 \div E'$ or satisfies $\phi_1(r) \neq \phi_1'(r)$. The only root in $F_2 \div E$ that does not exist in $F_2 \div E'$ is a result of cutting edge e_b and is a descendant z of b in F_2. Let C_z be the component of $F_2 \div E$ with root z. The only root in $F_2 \div E'$ that has a different set of labelled descendant leaves in $F_2 \div E$ is the root u' of the component C_u that contains u, and $\phi_1(u') \neq \phi_1'(u')$ only if $u' = u$. For any other component root x, we have $\phi_1(x) = \phi_1'(x)$. Thus, any cycle O in $G_{F_2 \div E}$ contains at least one of C_u and C_z. Next we prove that no such cycle exists in $G_{F_2 \div E}$, by using the following five observations.

(i) Since $u \sim_{F_2 \div E'} z$ and z is the only root of $F_2 \div E$ that does not exist in $F_2 \div E'$, there is no root $x \notin \{u, z\}$ of $F_2 \div E$ on the path from u to z in T_2.

(ii) Since $u \sim_{F_2 \div E'} z$ and $z \in F_2^u$, we have $\phi_1'(z) \in T_1^{\phi_1'(u)}$. Any component C_x with root x such that $x \notin \{u, z\}$ satisfies $\phi_1'(x) = \phi_1(x)$. If $\phi_1'(x)$ belonged to the path from $\phi_1'(u)$ to $\phi_1'(z)$, then C_x would overlap the component of $F_2 \div E'$ containing u in T_1. Since $F_2 \div E'$ is a forest of T_1, no such component C_x can exist.

(iii) Since $u \sim_{F_2 \div E'} c'$, we have $c' \in T_1^{\phi_1'(u)}$ and, by the same arguments as in (ii), there is no root $x \notin \{u, z\}$ such that $\phi_1'(x) = \phi_1(x)$ belongs to the path from c' to $\phi_1'(u)$ in T_1.

(iv) Since all labelled descendants of u in $F_2 \div E$ belong to $F_2^a \cup F_2^c$, with at least one descendant in each of F_2^a and F_2^c, we have $\phi_1(u) = p_a = p_c$. In particular, $c' \in T_1^{\phi_1(u)}$. Since u has c' and at least one labelled leaf in F_2^b as descendants in $F_2 \div E'$, $\phi_1'(u)$ is a proper ancestor of $\phi_1(u)$.

(v) $\phi_1'(z) = \phi_1(z)$ is neither an ancestor nor a descendant of $\phi_1(u)$. The latter follows because z has a labelled descendant leaf in $F_2 \div E$ that belongs to F_2^b, while all labelled descendant leaves of $\phi_1(u)$ belong to $F_2^a \cup F_2^c$. To see the former, observe that this would imply that $\phi_1'(z)$ is not a leaf and, hence, that there are two labelled descendant leaves b_1 and b_2 of z in $F_2 \div E'$ such that $b_1, b_2 \in F_2^b$ and the path from b_1 to b_2 in T_1 includes $\phi_1'(z)$. Since $u \sim_{F_2 \div E'} c'$ and $u \sim_{F_2 \div E'} z$, this would imply that $F_2 \div E'$ contains the triple $b_1 b_2 | c'$, while these leaves would form the triple $b_1 c' | b_2$ or $b_2 c' | b_1$ in T_1. This is a contradiction because $F_2 \div E'$ is a forest of T_1.

We now consider the different possible shapes of O. We use C_{x_1} and C_{x_2} to denote C_u's predecessor and successor in O, respectively, and C_{y_1} and C_{y_2} to denote C_z's predecessor and successor in O, respectively. First observe that $y_2 \neq u$ and, hence, $x_1 \neq z$. Indeed, $z \in F_2^u$, which implies that $y_2 = u$ only if $\phi_1(z)$ is an ancestor of $\phi_1(u)$. By (v), this is impossible.

If $y_1 = u$ (and $y_2 \neq u$), then $\phi_1(z) = \phi_1'(z)$ is an ancestor of $\phi_1(y_2) = \phi_1'(y_2)$ because, by (v), $\phi_1(u)$ is not an ancestor of $\phi_1(z)$ and the edges in O alternate between T_1 and T_2. By (ii), this implies that $\phi_1'(u)$ is an ancestor of $\phi_1'(y_2)$ in T_1. Also, for the predecessor C_{x_1} of C_u in O, $\phi_1'(x_1) = \phi_1(x_1)$ is an ancestor of $\phi_1(u)$ and, hence, by (iii) and (iv), an ancestor of $\phi_1'(u)$. This implies that we would obtain a cycle in $G_{F_2 \div E'}$ by removing C_z from O, which contradicts the assumption that $F_2 \div E'$ is acyclic. This shows that $y_1 \neq u$.

It remains to consider the case when C_u and C_z are not adjacent in O. In this case, all edges of O except those incident to C_u or C_z exist also in $G_{F_2 \div E'}$ because $\phi'_1(x) = \phi_1(x)$, for every root $x \notin \{u, z\}$. Next we show that, if $C_u \in O$, then the edges (C_{x_1}, C_u) and (C_u, C_{x_2}) also exist in $G_{F_2 \div E'}$, and if $C_z \in O$, then the edges (C_{y_1}, C_u) and (C_u, C_{y_2}) exist in $G_{F_2 \div E'}$. Thus, by replacing C_z with C_u in O (if $C_z \in O$), we obtain a cycle in $G_{F_2 \div E'}$, a contradiction because $F_2 \div E'$ is acyclic.

If $C_u \in O$, then either $\phi_1(x_1) = \phi'_1(x_1)$ is an ancestor of $\phi_1(u)$ and x_2 is a descendant of u, or x_1 is an ancestor of u and $\phi_1(x_2) = \phi'_1(x_2)$ is a descendant of $\phi_1(u)$. In the former case, (iii) and (iv) imply that $\phi'_1(x_1)$ is an ancestor of $\phi'_1(u)$. In the latter case, (iv) implies that $\phi'_1(x_2)$ is also a descendant of $\phi'_1(u)$. In both cases, the edges (C_{x_1}, C_u) and (C_u, C_{x_2}) exist in $G_{F_2 \div E'}$.

If $C_z \in O$, then either $\phi_1(y_1) = \phi'_1(y_1)$ is an ancestor of $\phi_1(z) = \phi'_1(z)$ and y_2 is a descendant of z, or y_1 is an ancestor of z and $\phi_1(y_2) = \phi'_1(y_2)$ is a descendant of $\phi_1(z) = \phi'_1(z)$. In the former case, (ii) implies that $\phi'_1(y_1)$ is an ancestor of $\phi'_1(u)$ and y_2 is a descendant of u. In the latter case, (i) and (ii) imply that y_1 is an ancestor of u and $\phi'_1(y_2)$ is a descendant of $\phi'_1(u)$. In both cases, the edges (C_{y_1}, C_u) and (C_u, C_{y_2}) exist in $G_{F_2 \div E'}$.

We have shown how to construct a corresponding cycle in $G_{F_2 \div E'}$ for every cycle $O \in G_{F_2 \div E}$. Since $F_2 \div E'$ is acyclic, this shows that $F_2 \div E$ is acyclic. □

We have thus shown that the branching phase of our algorithm will find at least one (not necessarily maximal) AF, F, that can be refined to an MAAF.

In the remainder of this section, we develop an efficient implementation of Refine (F, k). To do so, we need several new ideas. Each of the following sections discusses one of them. The tools introduced in Sections 4.1–4.3 suffice to obtain a fairly simple implementation of Refine (F, k) that leads to an MAAF algorithm with running time $\mathrm{O}\left(9.68^k n\right)$. Sections 4.4 and 4.5 then introduce two refinements that improve the algorithm's running time first to $\mathrm{O}\left(4.84^k n\right)$ and then to $\mathrm{O}\left(3.18^k n\right)$.

In Section 4.1, we introduce an expanded cycle graph G^*_F. In G^*_F, every node of G_F is replaced with the component of F it represents. This allows us to identify exactly which edges in a component C need to be cut if we want to break a cycle in G_F by removing C from this cycle. Moreover, if F has $k' + 1$ components, G^*_F

contains only $2k'$ of the edges of G_F. This ensures that G_F^* has size $\mathrm{O}(n)$, which is the key to keeping the MAAF algorithm's dependence on n linear.

In Section 4.2, we identify components of F that are *essential* for the cycles in G_F^* in the sense that at least one essential component of each cycle O in G_F^* has to be eliminated to break O (as opposed to replacing it with a shorter cycle). For every essential component C in such a cycle O, we identify one node in C, called an *exit node*, and show that there exists a component C in O such that cutting all edges on the path from C's exit node to C's root reduces $\tilde{e}(T_1, T_2, F)$ by the number of edges cut. We call the process of cutting these edges *fixing* the exit node.

In Section 4.3, we show how to mark a subset of at most $2k$ nodes in F such that, if F can be refined to an AAF of T_1 and T_2 with at most $k + 1$ components, then fixing an appropriate subset of these marked nodes produces such an AAF. We call these marked nodes *potential exit nodes* because they include the exit nodes of all essential components of all cycles in G_F^*. We obtain a first simple implementation of $\textsc{Refine}(F, k)$ by testing for each subset of potential exit nodes whether fixing it produces an AAF with at most $k + 1$ components. Since this test can be carried out in linear time for each subset and there are $2^{2k} = 4^k$ subsets to test, the running time of this implementation of $\textsc{Refine}(F, k)$ is $\mathrm{O}(4^k n)$. Since we make at most one invocation $\textsc{Refine}(F_2, k)$ per invocation $\textsc{Maaf}(F_1, F_2, k'')$ of the MAAF algorithm and the MAAF algorithm makes $\mathrm{O}(2.42^k)$ invocations $\textsc{Maaf}(F_1, F_2, k'')$, the resulting MAAF algorithm has running time $\mathrm{O}(2.42^k(n + 4^k n)) = \mathrm{O}(9.68^k n)$.

The bound of $2k$ on the number of potential exit nodes is obtained quite naturally: We can obtain F from both T_1 and T_2 by cutting the edges connecting the roots of the components of F to their parents in these trees. There are at most k component roots of F that are not roots in T_2. Each such component has two corresponding parent edges, one in T_1 and one in T_2. The potential exit nodes are essentially the top endpoints of these at most $2k$ parent edges, and the top endpoints of the two parent edges of each component root form a pair of potential exit nodes. In Section 4.4, we augment the search for agreement forests to annotate the component roots of each found agreement forest F with information about how F was obtained from T_2. Using this information, we mark one potential exit node in each pair of potential exit nodes and show that it suffices to test for each subset of marked potential exit nodes whether fixing it produces an AAF with at most $k+1$ components. Since at most k potential exit nodes get marked, this reduces the cost of $\textsc{Refine}(F,k)$ to $\mathrm{O}\left(2^k n\right)$ and, hence, the running time of the MAAF algorithm to $\mathrm{O}\left(4.84^k n\right)$.

In Section 4.5, we tighten the analysis of our algorithm. So far, we allowed both phases of the algorithm to cut k edges. However, k is the *total* number of edges we are allowed to cut. Thus, if the number k' of edges we cut to obtain an AF is large, there are only $k'' := k - k'$ edges left to cut in the refinement step, allowing us to restrict our attention to small subsets of marked potential exit nodes and thereby reducing the cost of the refinement step substantially. If, on the other hand, k' is small, then there are only few marked potential exit nodes and even trying all possible subsets of these nodes is not too costly. By analyzing this trade-off between the number of edges cut in each phase of the algorithm, we obtain the claimed running time of $\mathrm{O}\left(3.18^k n\right)$.

4.1 An Expanded Cycle Graph

The expanded cycle graph G_F^* of an agreement forest F of two rooted phylogenies T_1 and T_2 is a supergraph $G_F^* \supset F$ with the same vertex set as F; see Figure 4.1(c). Let E_1 and E_2 be minimal subsets of edges of T_1 and T_2 such that $F = T_1 \div E_1 = T_2 \div E_2$. In addition to the edges of F, G_F^* contains one *hybrid edge* per edge in $E_1 \cup E_2$. To define these edges, we define mappings from nodes of F to nodes of T_1 and T_2 and vice versa. As in the definition of the original cycle graph G_F in Chapter 2, 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 the reverse direction, 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 the path between x and y does not contain any edges in E_i. These mappings are well defined in the sense that $\phi_i^{-1}(\phi_i(x)) = x$, for all $x \in F$ and $i \in \{1,2\}$.

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 $\left(\phi_1^{-1}(z_1), y\right)$ is a T_1-*hybrid edge* and $\left(\phi_2^{-1}(z_2), y\right)$ is a T_2-*hybrid edge*. See Figure 4.1(c) for an illustration

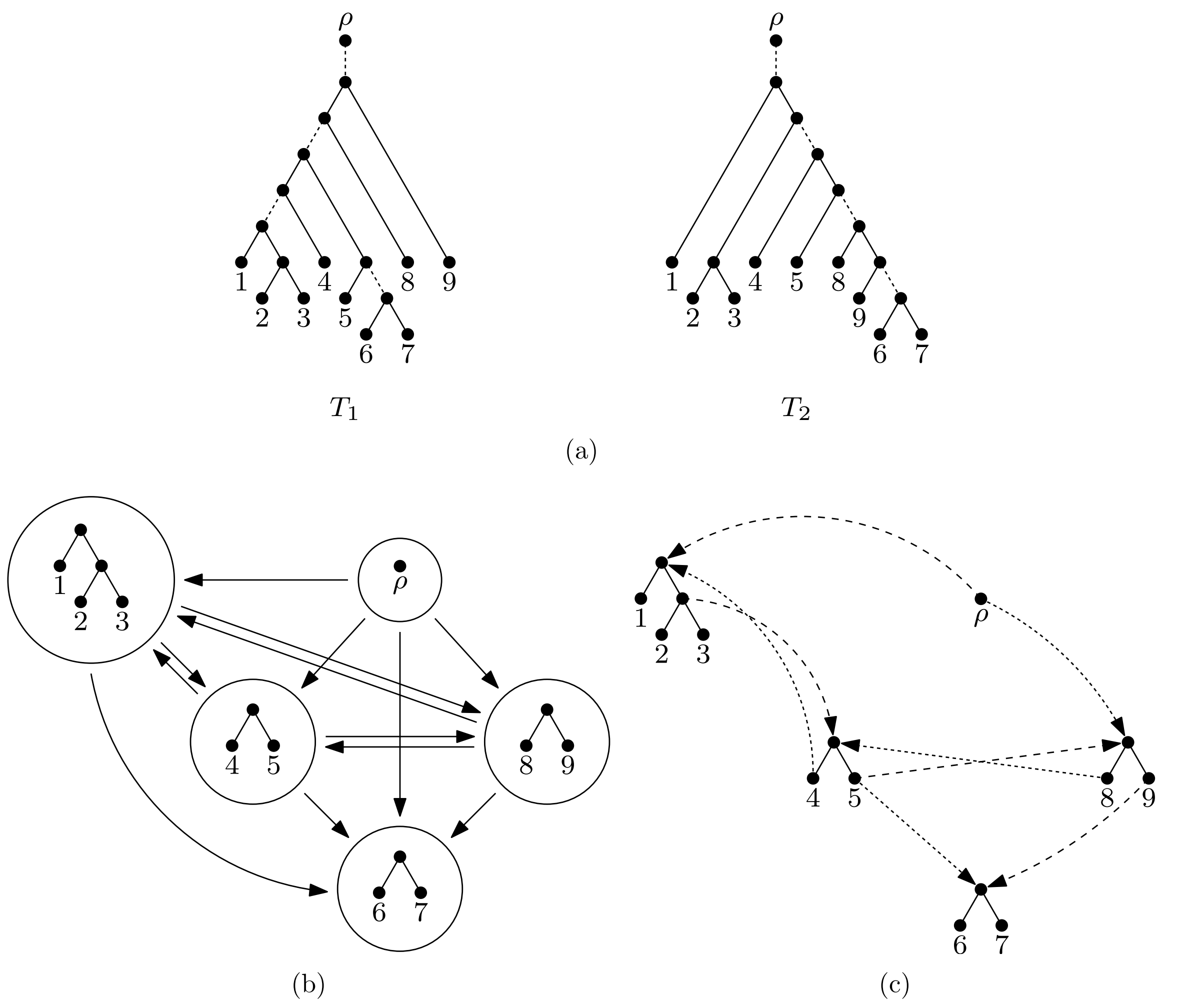


Figure 4.1: (a) Two trees T_1 and T_2. (b) An agreement forest F of T_1 and T_2 obtained by cutting the dotted edges in T_1 and T_2, and its cycle graph G_F. The component of F represented by each node of G_F is drawn inside the node. (c) The expanded cycle graph G_F^*. Dotted edges are T_1-hybrid edges, dashed ones are T_2-hybrid edges.

of these edges. Note that neither $\phi_1^{-1}(z_1)$ nor $\phi_2^{-1}(z_2)$ is a root of F. Our first 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.

Lemma 4.2. *G_F^* contains a cycle if and only if G_F does.*

Proof. First observe that G_F^* can be obtained from G_F by choosing a subset of the edges of G_F and then replacing each vertex of G_F with a component of F. Since the components of F do not contain cycles, this shows that G_F^* is acyclic if G_F is.

Conversely, for two nodes u and v of F, G_F^* contains a path from u to v if $\phi_1(u)$ is an ancestor of $\phi_1(v)$ or $\phi_2(u)$ is an ancestor of $\phi_2(v)$. Along with the fact that tree edges are directed away from the root of their component, this implies that every edge in G_F can be replaced by a directed path in G_F^*, so that G_F^* contains a cycle if G_F does. □

In the remainder of this section, we show that G_F^* can be constructed in linear time from T_1, T_2, and F, a fact we use in our algorithms in Sections 4.3, 4.4, and 4.5.

Lemma 4.3. *The expanded cycle graph G_F^* of an agreement forest F of two rooted phylogenies T_1 and T_2 can be computed in linear time.*

Proof. Our construction of G_F^* starts with F and then adds the hybrid edges. To add the hybrid edges induced by T_1, we perform a postorder traversal of T_1 that computes the mappings $\phi_1(\cdot)$ and $\phi_1^{-1}(\cdot)$, and the hybrid edges induced by T_1. A similar postorder traversal of T_2 then computes $\phi_2(\cdot)$, $\phi_2^{-1}(\cdot)$, and the hybrid edges induced by T_2.

We can assume 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 list L_x with each leaf x, where $L_x := \{x\}$ if x is a root of F, and $L_x = \varnothing$ otherwise. In general, 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. (It is not hard to see that this is the same ancestor of x, for every root in L_x.)

After setting up this information for the leaves of T_1, the postorder traversal computes the same information for the nonleaf nodes of T_1 and uses it to compute the T_1-hybrid edges in G_F^*. For a nonleaf node x with children l and r, the mappings $\phi_1^{-1}(l)$ and $\phi_1^{-1}(r)$ and the root lists L_l and L_r of l and r are computed before processing x. Hence, we can use them to compute the mapping $\phi_1^{-1}(x)$ and the root list L_x. We distinguish four cases.

If neither $\phi_1^{-1}(l)$ nor $\phi_1^{-1}(r)$ is undefined or a root of F, then they must have a common parent p in F (because l and r are siblings in T_1 and F is a forest of T_1). In this case, we set $\phi_1^{-1}(x) = p$ and $\phi_1(p) = x$. If p is a root other than ρ, we set $L_x = \{p\}$; otherwise $L_x = \varnothing$.

If both $\phi_1^{-1}(l)$ and $\phi_1^{-1}(r)$ are undefined or a root of F, then $\phi^{-1}(x)$ is undefined (as x can belong to a path between two labelled nodes a and b such that $a \sim_F b$ only if this is true for at least one of its children) and we set $L_x = L_l \cup L_r$.

If only $\phi_1^{-1}(l)$ is undefined or a root of F, we set $\phi_1^{-1}(x) := \phi_1^{-1}(r)$ and add a T_1-hybrid edge $\left(\phi_1^{-1}(x), y\right)$ to G_F^*, for every root y in L_l. Then we set $L_x = \varnothing$ (x cannot be the image $\phi_1(x')$ of a root x' of F and $L_r = \varnothing$ in this case).

The final case in which only $\phi_1^{-1}(r)$ is undefined or a root of F is symmetric to the previous case.

It is easy to see that this procedure correctly constructs G_F^* because it directly follows the definition of G_F^*. The running time of the algorithm is also easily seen to be linear. Indeed, computing the mappings $\phi_1^{-1}(x)$ and possibly $\phi_1(p)$ from $\phi_1^{-1}(l)$ and $\phi_1^{-1}(r)$ takes constant time per visited node x, linear time in total. In the case when L_x is computed as the union of L_l and L_r, L_l and L_r can be concatenated in constant time. In the case when we add a hybrid edge to G_F^*, for every node in L_l or L_r, this takes constant time per node, and we then pass an empty list L_x to x's parent. The latter implies that every root added to a list L_x leads to the addition of exactly one hybrid edge to G_F^*. Since every node adds at most one root to L_x that is not already present in L_l or L_r, this shows that the addition of hybrid edges to G_F^* also takes linear time in total for all nodes of T_1. The running time of the traversal of T_2 is bounded by $\mathrm{O}(n)$ using the same arguments. Hence, the entire algorithm takes linear time. $\square$

One thing to note about the algorithm for constructing G_F^* is that it does not require knowledge of the edge sets E_1 and E_2, even though we used these sets to define G_F^*. This implies in particular that, even though there may be different edge sets E_1 and E_2 such that $T_1 \div E_1 = T_2 \div E_2 = F$, all of them lead to the same cycle graph—G_F^* is completely determined by F alone.

4.2 Essential Components and Exit Nodes

In this section, we define the essential components of a cycle in G_F^* and their exit nodes. Our goal is to prove that, if F can be refined to an AAF of T_1 and T_2 with at most $k+1$ components, this is possible exclusively by cutting the 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 H_2 the set of T_2-hybrid edges in G^*_F, and assume G^*_F contains a cycle O. Let $h_0, h_1, \ldots, h_{m-1}$ be the hybrid edges in O, and consider the components $C_0, C_1, \ldots, C_{m-1}$ of F connected by these 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 its root and leaves it at the tail of the edge h_i. We say a component C_i is *essential for* O if $h_{i-1} \in H_1$ and $h_i \in H_2$ or vice versa. We say a component C of F is *essential* if it is essential for at least one cycle in G^*_F. A node x of a component C of F is an *exit node* of C if C is an essential component C_i for some cycle O in G^*_F and x is the tail of edge h_i in this cycle. Figure 4.2(c) illustrates these concepts. Our first result in this section shows that there exists 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, that is, by cutting this edge, we make progress towards an MAAF of T_1 and T_2.

Lemma 4.4. *Let O be a cycle in G^*_F, let $C_0, C_1, \ldots, C_{m-1}$ be its essential components, and let v_i be the exit node of component C_i in O, for all $0 \leqslant i \leqslant m-1$. Then $\tilde{e}(T_1, T_2, F \div \{e_{v_i}\}) = \tilde{e}(T_1, T_2, F) - 1$, for some $0 \leqslant i < m$.*

Proof. Let E be an arbitrary edge set of size $\tilde{e}(T_1, T_2, F)$ and such that $F' := F \div E$ is an AAF of T_1 and T_2. If $E \cap \{e_{v_0}, e_{v_1}, \ldots, e_{v_{m-1}}\} \neq \varnothing$, the lemma holds. If $E \cap \{e_{v_0}, e_{v_1}, \ldots, e_{v_{m-1}}\} = \varnothing$, we show that there exists an edge $f \in E$ such that $F' = F \div (E \setminus \{f\} \cup \{e_{v_i}\})$, for some $0 \leqslant i < m$, which again proves the lemma. Let r_i be the root of component C_i, for all $0 \leqslant i < m$. To avoid excessive use of modulo notation in indices, we define T_i, $\phi_i(\cdot)$, etc. to be the same as $T_{2-(i \bmod 2)}$, $\phi_{2-(i \bmod 2)}(\cdot)$, etc. in the remainder of this proof.

First suppose there exist leaves $a_i \in C_i^{v_i}$ and $c_i \in C_i \setminus C_i^{v_i}$ such that $a_i \sim_{F'} c_i$, for all $0 \leqslant i < m$, and let l_i be the LCA of a_i and c_i in F'. Further, for every node $x \in F'$ and for $i \in \{1, 2\}$, let $\phi_i(x)$ and $\phi'_i(x)$ be the nodes in T_i x maps to based on its descendants in F and F', respectively. Since $C_0, C_1, \ldots, C_{m-1}$ are the essential components of O, m is even and, w.l.o.g., the hybrid edge with head r_i is T_{i-1}-hybrid and the hybrid edge with tail v_i is T_i-hybrid. This implies that the lowest ancestor x_i of $\phi_i(r_{i+1})$ such that $\phi_i^{-1}(x_i)$ is defined and belongs to C_i satisfies $\phi_i^{-1}(x_i) = v_i$.

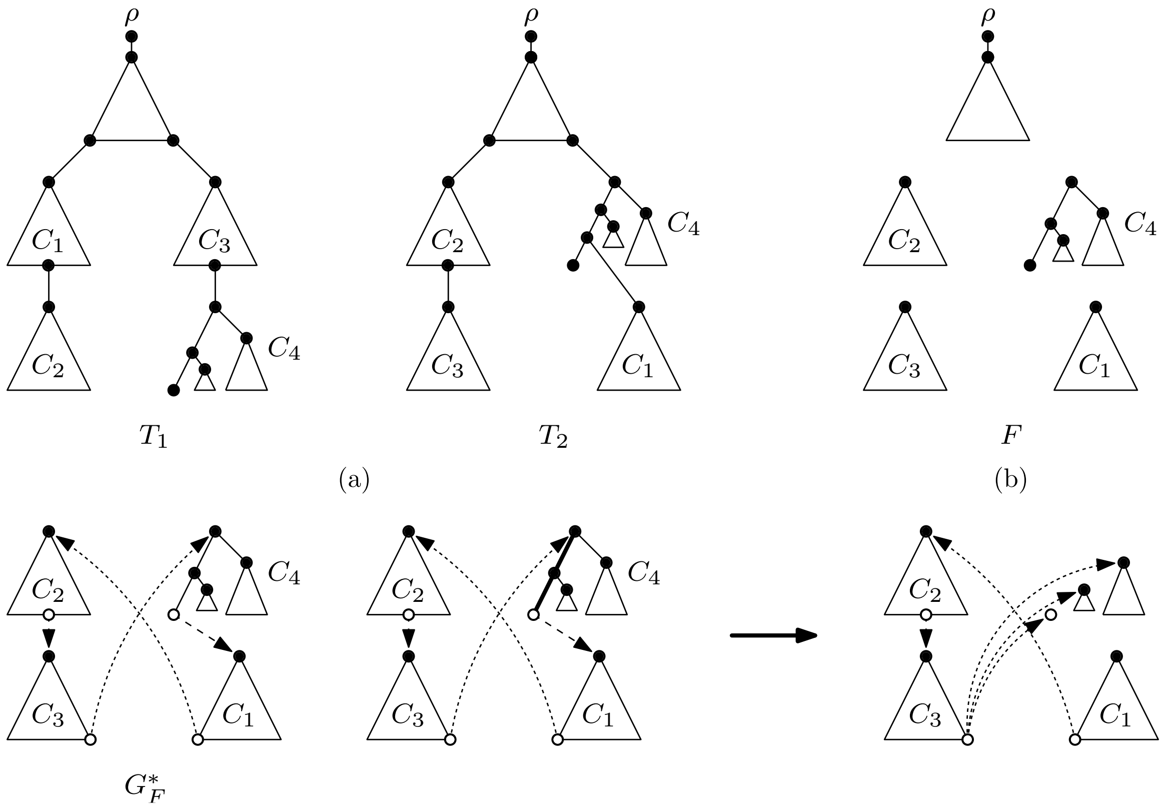


Figure 4.2: (a) Two trees T_1 and T_2. (b) An agreement forest F of T_1 and T_2. (c) G_F^* (with ρ's component removed for clarity) contains a cycle of length 4. White nodes indicate exit nodes. (d) Fixing the exit node of component C_4 (cutting the bold edges) removes the cycle because none of the resulting subcomponents of C_4 is an ancestor of C_1 in T_2.

Now observe that $\phi'_i(l_i)$ is a descendant of $\phi_i(r_i)$ and an ancestor of x_i in T_i. The former follows because (i) the set of l_i's descendants in F' is a subset of l_i's descendants in F and, thus, $\phi'_i(l_i)$ is a descendant of $\phi_i(l_i)$, and (ii) l_i is a descendant of r_i in F and, hence, $\phi_i(l_i)$ is a descendant of $\phi_i(r_i)$. The latter follows because $\phi'_i(a_i) = \phi_i(a_i)$ is a descendant of x_i, while $\phi'_i(c_i) = \phi_i(c_i)$ is not. Since x_i is an ancestor of $\phi_i(r_{i+1})$, for all i, this implies that $\phi'_i(l_i)$ is an ancestor of $\phi'_i(l_{i+1})$, for all i, which shows that the components of F' containing these nodes form a cycle in $G_{F'}$, contradicting that F' is acyclic.

Thus, there exists a component C_i such that $a \nsim_{F'} c$, for all labelled leaves $a \in C_i^{v_i}$ and $c \in C_i \setminus C_i^{v_i}$. This in turn implies that either $a \nsim_{F'} v_i$, for all labelled leaves $a \in C_i^{v_i}$, or $c \nsim_{F'} v_i$, for all labelled leaves $c \in C_i \setminus C_i^{v_i}$. W.l.o.g., assume the former.

We choose an arbitrary labelled leaf $a' \in C_i^{v_i}$ and let f be the first edge in E on the path from v_i to a'. Since $a \not\sim_{F'} v_i$, for all $a \in C_i^{v_i}$, this edge f and the edge $e = e_{v_i}$ satisfy the conditions of Lemma 2.1 and, hence, $F \div E = F \div (E \setminus \{f\} \cup \{e_{v_i}\})$ is an AAF of T_1 and T_2. □

The following corollary of Lemma 4.4 shows that we can in fact make progress towards an AAF by cutting *all* edges on the path from an appropriate exit node to the root of its component. We call this *fixing* the exit node. Removing a cycle by fixing an exit node is illustrated in Figure 4.2(d).

Corollary 4.5. *Let O be a cycle in G_F^*, let $C_0, C_1, \ldots, C_{m-1}$ 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 $0 \leqslant i \leqslant m-1$. Then $\tilde{e}(T_1, T_2, F_i) = \tilde{e}(T_1, T_2, F) - \ell_i$, for some $0 \leqslant i \leqslant m-1$.*

Proof. The proof is by induction on $\tilde{e}(T_1, T_2, F)$. By Lemma 4.4, 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 containing the leaves in $C_i^{v_i}$ and in $C_i \setminus C_i^{v_i}$, respectively.

If $\tilde{e}(T_1, T_2, F) = 1$, then $\tilde{e}(T_1, T_2, F') = 0$. This implies that the path from v_i to r_i in C_i cannot contain any edges apart from e_{v_i} because otherwise $C_0, C_1, \ldots, C_{i-1}, B_i, C_{i+1}, \ldots, C_{m-1}$ would form a cycle in $G_{F'}$, that is, $\tilde{e}(T_1, T_2, F') > 0$. Thus, the corollary holds for $\tilde{e}(T_1, T_2, F) = 1$.

If $\tilde{e}(T_1, T_2, F) > 1$, we can assume by induction that the corollary holds for F'. If $\ell_i = 1$, then the path from v_i to r_i consists of only e_{v_i}, and the corollary holds for F. Otherwise $C'_0, C'_1, \ldots, C'_{i-1}, C'_i, C'_{i+1}, \ldots, C'_{m-1} := C_0, C_1, \ldots, C_{i-1}, B_i, C_{i+1}, \ldots, C_{m-1}$ is a cycle O' in $G_{F'}$. Note that, for $j \neq i$, the exit node v'_j of C'_j in O' is v_j; the exit node v'_i of C'_i is v_i's sibling in C_i. By the inductive hypothesis, there exists some C'_j, $0 \leqslant j < m$, such that $\tilde{e}\left(T_1, T_2, F'_j\right) = \tilde{e}(T_1, T_2, F') - \ell'_j$, where F'_j is obtained from F' by fixing v'_j and ℓ'_j is the length of the path from v'_j to the root of C'_j. In particular, $\ell'_j = \ell_j$, for $j \neq i$; and $\ell'_i = \ell_i - 1$. If $j \neq i$, we have $F'_j = F_j \div \{e_{v_i}\}$ and $\tilde{e}(T_1, T_2, F_j \div \{e_{v_i}\}) = \tilde{e}\left(T_1, T_2, F'_j\right) = \tilde{e}(T_1, T_2, F') - \ell'_j = \tilde{e}(T_1, T_2, F) - \ell_j - 1$. Hence, $\tilde{e}(T_1, T_2, F_j) \leqslant \tilde{e}(T_1, T_2, F) - \ell_j$. Since F_j is obtained from F by cutting ℓ_j edges, we also have $\tilde{e}(T_1, T_2, F_j) \geqslant \tilde{e}(T_1, T_2, F) - \ell_j$. Thus, the corollary holds in this case. If $j = i$, we have $F'_j = F_j$ and $\tilde{e}(T_1, T_2, F_j) = \tilde{e}\left(T_1, T_2, F'_j\right) = \tilde{e}(T_1, T_2, F') - \ell'_j = \tilde{e}(T_1, T_2, F) - \ell_j$. Thus, the corollary holds in this case as well. $\square$

4.3 Potential Exit Nodes and a Simple Refinement Algorithm

In this section, we introduce the concept of potential exit nodes and show that a first simple refinement algorithm can be obtained by testing for each subset of potential exit nodes whether fixing these nodes produces an AAF with at most $k + 1$ components.

Given an agreement forest F of T_1 and T_2, we mark all those nodes in F that are the tails of hybrid edges in G^*_F. Since this includes all exit nodes of F, we call these nodes *potential exit nodes.* If F has k' components, there are $2(k' - 1)$ potential exit nodes. If F is a forest produced by the branching phase of our algorithm, it has at most $k + 1$ components and, thus, at most $2k$ potential exit nodes. The main result in this section is Lemma 4.6, which 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. We use this lemma to prove that, if F can be refined to an AAF with at most $k + 1$ components, then fixing an appropriate subset of potential exit nodes produces such a forest.

Lemma 4.6. *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. Since fixing v removes v's parent edge, v is a root of F', which implies that $v \notin V'$ because potential exit nodes are not component roots. Thus, $V' \neq V$, and it suffices to prove that $V' \subseteq V$. So let $u \in V'$, and let (u, w) be a hybrid edge in $G^*_{F'}$ with tail u. Assume w.l.o.g. that (u, w) is a T_1-hybrid edge, let $\phi_1(\cdot)$ and $\phi_1^{-1}(\cdot)$ be defined as before with respect to F, and let $\phi'_1(\cdot)$ and $\phi'^{-1}_1(\cdot)$ be the same mappings defined with respect to F'. By the definition of a hybrid edge, w is the root of a component of F' and $u = \phi'^{-1}_1(x)$, for the lowest proper ancestor x of $\phi'_1(w)$ such that $\phi'^{-1}_1(x)$ is defined.

Now let $E_1 \subset E_1'$ be edge sets such that $F = T_1 \div E_1$ and $F' = T_1 \div E_1'$, and let E be the set of edges cut in F to fix v, that is, $F' = F \div E$. We prove that $a \sim_{T_1 - E_1} x$ if and only if $a \sim_{T_1 - E_1'} x$, for every labelled leaf $a \in T_1^x$. This implies in particular that $\phi_1'^{-1}(y) = \phi_1^{-1}(y)$, for all nodes $y \in T_1^x$ such that $x \sim_{T_1 - E_1} y$.

Clearly, if $a \sim_{T_1 - E_1'} x$, then $a \sim_{T_1 - E_1} x$ because $E_1 \subset E_1'$. So assume $a \sim_{T_1 - E_1} x$ but $a \nsim_{T_1 - E_1'} x$, for some labelled leaf $a \in T_1^x$. Since $\phi_1'^{-1}(x)$ is defined, there exist labelled nodes b and c such that $b \sim_{F'} c$ and x is on the path from b to c in $T_1 - E_1'$. This implies that $b \sim_{T_1 - E_1'} x \sim_{T_1 - E_1'} c$ and, hence, $b \sim_{T_1 - E_1} x \sim_{T_1 - E_1} c$. Together with $a \sim_{T_1 - E_1} x$, this implies that a, b, and c belong to the same connected component of $T_1 - E_1$ and, hence, to the same connected component of F, while a belongs to a different connected component of F' than b and c. Now observe that, since x is an ancestor of a and is on the path from b to c, the lowest common ancestor of b and c in T_1 is an ancestor of a. Since F is a forest of T_1, this implies that the lowest common ancestor l of b and c in F also is an ancestor of a. Since $b \sim_{F'} l \sim_{F'} c$ and $a \nsim_{F'} c$, the path from a to l must contain at least one edge in E. By the choice of E, this implies that one of the child edges of l also belongs to E and, hence, that $b \nsim_{F-E} c$, a contradiction because $F' = F \div E$ and $b \sim_{F'} c$.

To finish the proof, let y be the first node after x on the path from x to $\phi_1'(w)$ and such that $\phi_1^{-1}(y)$ is defined. Since $\phi_1'^{-1}(\phi_1'(w)) = w$, $\phi_1'^{-1}(\phi_1'(w))$ and, hence, $\phi_1^{-1}(\phi_1'(w))$ is defined, that is, such a node y exists. If $x \nsim_{T_1 - E_1} y$, then $\phi_1^{-1}(y)$ is a root of F and $(\phi_1^{-1}(x), \phi_1^{-1}(y))$ is a hybrid edge in G_F^*. Since $\phi_1^{-1}(x) = \phi_1'^{-1}(x) = u$, this proves that u is also a potential exit node of F. If $x \sim_{T_1 - E_1} y$, then $\phi_1'^{-1}(y) = \phi_1^{-1}(y)$, that is, $\phi_1'^{-1}(y)$ is defined. By the choice of x, this implies that $y = \phi_1'(w)$. Since $\phi_1'^{-1}(\phi_1'(w))$ is defined, there exists a leaf $a \in T_1^{\phi_1'(w)}$ such that $a \sim_{T_1 - E_1'} \phi_1'(w)$ and, hence, $a \sim_{F'} w$ and $a \sim_{T_1 - E_1} \phi_1'(w)$. Together with $\phi_1'(w) \sim_{T_1 - E_1} x$, the latter implies that $a \sim_{T_1 - E_1} x$, while (u, w) being a hybrid edge implies that $u \nsim_{F'} w$ and, hence, $a \nsim_{F'} u$ and $a \nsim_{T_1 - E_1'} x$. This is a contradiction, that is, the case $x \sim_{T_1 - E_1} y$ cannot occur. □

By Corollary 4.5, if F can be refined to an AAF F' with at most $k + 1$ 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 $F = F_0, F_1, \ldots, F_{k+1} = F'$ be the sequence of forests produced in this fashion. For

$0 \leqslant i \leqslant k$, the exit nodes of F_i are included in the set of F_i's potential exit nodes and, by Lemma 4.6, these potential exit nodes are included in the set of F's potential exit node. Thus, F' can be obtained from F by choosing an appropriate subset of F's potential exit nodes and fixing them. Now it suffices to observe that fixing a subset of exit nodes one node at a time produces the same forest as simultaneous cutting all edges in the union of the paths from these exit nodes to the roots of their components in F.

This leads to the following simple refinement algorithm: We mark the potential exit nodes in F, which is easily done in linear time as part of constructing G^*_F. Then we consider every subset of potential exit nodes. For each such subset, we can in linear time identify the edges on the paths from these potential exit nodes to the roots of their components, cut these edges and suppress nodes with only one child, construct the expanded cycle graph $G^*_{F'}$ of the resulting forest F', and test whether F' has at most $k+1$ components and $G^*_{F'}$ is acyclic. We return "yes" as soon as we find a subset of potential exit nodes for which this test succeeds. If it fails for all subsets of potential exit nodes, we return "no". If F cannot be refined to an AAF with at most $k+1$ components, this test fails for every subset of potential exit nodes. Otherwise, as we have argued above, it will succeed for at least one subset of potential exit nodes. Thus, this implementation of $\textsc{Refine}\,(F, k)$ is correct.

If F has $k' \leqslant k+1$ components, there are at most $2^{2(k'-1)} \leqslant 2^{2k} = 4^k$ subsets of potential exit nodes to test by $\textsc{Refine}\,(F, k)$. Thus, the running time of $\textsc{Refine}\,(F, k)$ is $\mathrm{O}\left(4^k n\right)$. As we argued at the beginning of this section, using this implementation of $\textsc{Refine}\,(F, k)$ for the refinement phase of our MAAF algorithm results in a running time of $\mathrm{O}\left(2.42^k\left(n + 4^k n\right)\right) = \mathrm{O}\left(9.68^k n\right)$, and we obtain the following result.

Theorem 4.7. *For two rooted trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left(9.68^k n\right)$ time to decide whether $\tilde{e}\,(T_1, T_2, T_2) \leqslant k$.*

4.4 Halving the Number of Potential Exit Nodes

In this section, we show how to mark half of the at most $2k$ potential exit nodes defined in Section 4.3 and show that it suffices to test for every subset of *marked* potential exit nodes whether fixing it produces an AAF of T_1 and T_2 with at most

$k+1$ components. Since this reduces the number of subsets to be tested from 4^k to 2^k, the running time of the refinement step is reduced to $\mathrm{O}\left(2^k n\right)$, and the running time of the entire MAAF algorithm is reduced to $\mathrm{O}\left(4.84^k n\right)$.

In general, the result of marking only a subset of potential exit nodes is that we may obtain an AF F of T_1 and T_2 that *can* be refined to an AAF of T_1 and T_2 with at most $k+1$ components but *cannot* be refined to such an AAF by fixing any subset of the potential exit nodes marked in F. Intuitively, the reason why this is not a problem is that, whenever we reach such an AF F where a potential exit node u should be fixed but is not marked, there exists a branch in the branching phase's search for AFs that cuts a subset of the edges cut to produce F and then cuts e_u. Thus, if it is necessary to fix u in F to obtain an AAF F' of T_1 and T_2 with at most $k+1$ components, there exists an alternate route to obtain the same AAF F' by first producing a different AF F'' and then refining it. While this is the intuition, it is in fact possible that our algorithm is not able to produce F' from F'' either. What we do prove is that, if $\tilde{e}\left(T_1, T_2, T_2\right) \leqslant k$, then there exists a "canonical" AF F_C produced by the branching phase of our algorithm and which can be refined to an AAF F'_C of T_1 and T_2 with at most $k+1$ components by fixing a subset of the marked potential exit nodes in F_C.

We accomplish the marking of potential exit nodes 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. After constructing G_F^*, the refinement step marks a potential exit node u if there exists a T_i-hybrid edge (u, w) in G_F^* such that w's tag is "T_i". Finally, the refinement step checks whether an AAF of T_1 and T_2 with at most $k+1$ components can be obtained from F by fixing a subset of the marked potential exit nodes.

To tag component roots during the branching phase of the algorithm, we augment the three cases of Step 6 to tag the bottom endpoints of the edges they cut in F_2. When a tagged node x loses a child l by cutting its parent edge e_l, x is contracted into its other child r; in this case, r inherits x's tag. This ensures that at any time exactly the roots in the current forest F_2 are tagged. The following is the pseudocode of the MAAF algorithm, which shows the tags assigned to the component roots produced in Step 6. Note that Case 6.2 has an additional branch that cuts both e_a and e_c. This is necessary to ensure we find an AAF of T_1 and T_2 with at most $k+1$ components in spite of considering only subsets of marked potential exit nodes in the refinement phase (if such an AAF exists). In the description of the algorithm, we use k to denote the parameter passed to the current invocation (as in the MAF algorithm), and k_0 to denote the parameter of the top-level invocation $\textsc{Maaf}\,(T_1, T_2, k_0)$. Thus, $k_0 + 1$ is the number of connected components we allow the final AAF to have.

1. (Failure) If $k < 0$, there is no subset E of at most k edges of F_2 such that $F_2 - E$ yields an AF of T_1 and T_2: $\tilde{e}\,(T_1, T_2, F_2) \geqslant 0 > k$. Return "no" in this case.

2. (Refinement) If $|R_t| \leqslant 2$, then $F_2 = \dot{F}_2 \cup F$ is an AF of T_1 and T_2. Invoke an algorithm $\textsc{Refine}\,(F_2, k_0)$ that decides whether F_2 can be refined to an AAF of T_1 and T_2 with at most $k_0 + 1$ components. Return the answer returned by $\textsc{Refine}\,(F_2, k_0)$.

3. (Prune maximal agreeing subtrees) If there is a node $r \in R_t$ that is a root in $\dot{F}_2$, remove r from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F; cut the edge e_r in $\dot{T}_1$ and suppress r's parent from $\dot{T}_1$; return to Step 2. This does not alter F_2 and, thus, neither $\tilde{e}\,(T_1, T_2, F_2)$. If no such root r exists, proceed to Step 4.

4. Choose a sibling pair (a, c) in $\dot{T}_1$ such that $a, c \in R_t$.

5. (Grow agreeing subtrees) If (a, c) is a sibling pair of $\dot{F}_2$, remove a and c from R_t; label their parent in both trees with (a, c) and add it to R_t; return to Step 2. If (a, c) is not a sibling pair of $\dot{F}_2$, proceed to Step 6.

6. (Cut edges) Distinguish three cases:

   6.1. If $a \nsim_{F_2} c$, make two recursive calls:

      (i) $\textsc{Maaf}\,(F_1, F_2 \div \{e_a\}, k - 1)$ with a tagged with "T_2" in $F_2 \div \{e_a\}$, and

      (ii) $\textsc{Maaf}\,(F_1, F_2 \div \{e_c\}, k - 1)$ with c tagged with "T_2" in $F_2 \div \{e_c\}$.

   6.2. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has one pendant node b, swap the names of a and c if necessary to ensure that b is a's sibling. Then make three recursive calls (see Figure 4.3):

      (i) $\textsc{Maaf}\,(F_1, F_2 \div \{e_b\}, k - 1)$ with b tagged with "T_1" in $F_2 \div \{e_b\}$,

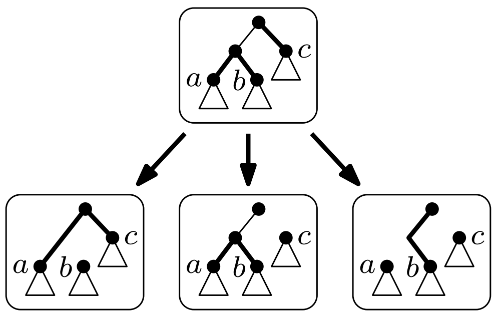


Figure 4.3: Case 6.2 of Step 6 of the rooted MAAF algorithm. (Cases 6.1 and 6.3 are as in the rooted MAF algorithm and are illustrated in Figure 3.1). Only $\dot{F}_2$ is shown. Each box represents a recursive call.

(ii) MAAF $(F_1, F_2 \div \{e_c\}, k-1)$ with c tagged with "T_2" in $F_2 \div \{e_c\}$, and

(iii) MAAF $(F_1, F_2 \div \{e_a, e_c\}, k-2)$ with c tagged with "T_1" and a tagged with "T_2" in $F_2 \div \{e_a, e_c\}$.

6.3. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$, make three recursive calls:

(i) MAAF $\big(F_1, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\big)$ with each node b_i, $1 \leqslant i \leqslant q$, tagged with "$T_1$" in $F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}$,

(ii) MAAF $(F_1, F_2 \div \{e_a\}, k-1)$ with a tagged with "T_2" in $F_2 \div \{e_a\}$, and

(iii) MAAF $(F_1, F_2 \div \{e_c\}, k-1)$ with c tagged with "T_2" in $F_2 \div \{e_c\}$.

Return "yes" if one of the recursive calls does; otherwise return "no".

To give some intuition behind the choice of tags in Step 6, and as a basis for the correctness proof of the algorithm, we consider a slightly modified algorithm that produces the same set of forests: When cutting an edge e_x, $x \in \{a, c\}$, in Step 6, x becomes the root of a component 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 edges of x in *both* F_1 and F_2 in Step 6 instead of postponing the cutting of x's parent edge in F_1 to Step 3 of the next recursive call.

Now consider a labelled node x of F_2, and let y_1 and y_2 be x's siblings in F_1 and F_2, respectively. If a case of Step 6 cuts the edge e_x, x becomes a root and, in the absence of further changes that eliminate x, y_1 or y_2 from the forest, x is the head of a T_1-hybrid edge (y_1, x) and of a T_2-hybrid edge (y_2, x), making y_1 and y_2 potential exit nodes that may need to be fixed to obtain a certain AAF of T_1 and T_2. The first step in fixing a potential exit node is to cut its parent edge, and an alternate sequence of edge cuts that produce the same AAF starts by cutting this parent edge instead of e_x. Thus, if apart from cutting e_x, the current case includes a branch that cuts the parent edge of y_1 or y_2, we do not have to worry about fixing this exit node in the branch that cuts e_x—there exits another branch we explore that has the potential of leading to the same AAF. To illustrate this idea, consider Case 6.1. Here, when we cut e_a, c becomes the tail of the T_1-hybrid edge (c, a) because a and c are siblings in F_1. Since the other branch of this case cuts e_c, we do not have to worry about fixing c in the branch that cuts e_a. On the other hand, neither of the two cases considers cutting the parent edge of a's sibling b in F_2, which is the tail of the T_2-hybrid edge with head a. Thus, we need to give the refinement step the opportunity to fix b. We do this by tagging a with "T_2", which causes the refinement step to mark b. The same reasoning justifies the tagging of c with "T_2" in the other branch of this case and the tagging of a and c with "T_2" in Case 6.3. The tagging of every node b_i with "T_1" in Case 6.3 is equally easy to justify: Cutting an edge e_{b_i} makes b_i a root in F_2. Thus, either b_i is itself a root of the final AF we obtain or it is contracted into such a root z after cutting additional edges; this root z inherits b_i's tag. At the time we cut edge e_{b_i}, we do not know which descendant of b_i will become this root z, nor whether any branch of our algorithm considers cutting the parent edge of the tail of z's T_1-hybrid edge. On the other hand, the tail of z's T_2-hybrid edge is either a or c, and we cut their parent edges in the other two branches of Case 6.3. Unfortunately, the tagging in Case 6.2 does not follow the same intuition and is in fact difficult to justify intuitively. The proof of Theorem 4.9 below shows that the chosen tagging rules lead to a correct algorithm.

We assume from here on that $hyb\,(T_1, T_2) \leqslant k_0$ because otherwise Refine (F, k_0) returns "no" for any AF F the branching phase may find, that is, the algorithm gives the correct answer when $hyb\,(T_1, T_2) > k_0$. Among the AFs of T_1 and T_2 produced by the branching phase, there may be several that can be refined to an AAF of T_1 and T_2 with at most $k + 1$ components. We choose a *canonical AF F_C* from among these AFs. The proof of Theorem 4.9 below shows that the potential exit nodes in F_C that need to be fixed to obtain such an AAF are marked. Since F_C is produced by a sequence of recursive calls of procedure $\textsc{Maaf}(\cdot,\cdot,\cdot)$, we can define F_C by specifying the path to take from the top-level invocation $\textsc{Maaf}(T_1, T_2, k_0)$ to the invocation $\textsc{Maaf}(F_1, F_2, k)$ with $F_2 = F_C$. We use F_1^i and F_2^i to denote the inputs to the ith invocation $\textsc{Maaf}(F_1^i, F_2^i, k_i)$ along this path. We also compute an arbitrary numbering of the nodes of T_1 and denote the number of $x \in T_1$ by $\nu(x)$. This number is used as a tie breaker when choosing the next invocation along the path of invocations that produce F_C. The first invocation is of course $\textsc{Maaf}(T_1, T_2, k_0)$, that is, $F_1^0 = T_1$ and $F_2^0 = T_2$. So assume we have constructed the path up to the ith invocation with inputs F_1^i and F_2^i. The $(i+1)$st invocation is made in Step 6 of the ith invocation. We say an invocation $\textsc{Maaf}(F_1, F_2, k)$ is a *leaf invocation* if F_2 is an AF of T_1 and T_2. Recall the definition of a viable invocation from the beginning of this section and recall that $\textsc{Maaf}(T_1, T_2, k_0)$ is viable and that every viable invocation that is not a leaf invocation has a viable child invocation. If there is only one viable invocation made in Step 6 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 $\textsc{Maaf}\left(F_1^{i+1}, F_2^{i+1}, k_{i+1}\right)$. Otherwise we apply the following rules to choose $\textsc{Maaf}\left(F_1^{i+1}, F_2^{i+1}, k_{i+1}\right)$ from among the viable invocations made in Step 6 of invocation $\textsc{Maaf}(F_1^i, F_2^i, k_i)$. We distinguish the three cases of Step 6.

Case 6.1. In this case, $\textsc{Maaf}(F_1^i \div \{e_a\}, F_2^i \div \{e_a\}, k_i - 1)$ and $\textsc{Maaf}(F_1^i \div \{e_c\}, F_2^i \div \{e_c\}, k_i - 1)$ are both viable invocations. For $x \in \{a, c\}$, 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 $F_x = T_1 \div E_x$. Let y be the sibling of x in F_1^i (i.e., $y = c$ if $x = a$ and vice versa). 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 $\phi_1(y)$ in T_1 and such that the path from l to $\phi_1(y)$ in T_1 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 produces F_x from F_2^i. Finally, if $\phi_x(y)$ is the root of a component of F_x, let $\lambda_1(y) := \phi_1(y)$; otherwise let $\lambda_1(y)$ be the LCA in T_1 of all labelled leaves that are descendants of the parent of $\phi_x(y)$ in F_x. In other words, if

$\phi_x(y)$ is not a root in F_x, then $\lambda_1(y)$ is the node in T_1 where $\phi_x(y)$ and its sibling in F_x are joined by an application of Step 5 in some recursive call on the path to F_x. Now let $d_1(y) > 0$ be the distance from the root ρ of T_1 to $\lambda_1(y)$ if $\lambda_1(y) \neq \phi_1(y)$, and $d_1(y) = 0$ otherwise. If $d_1(a) > d_1(c)$ or $d_1(a) = d_1(c)$ and $\nu(a) < \nu(c)$, we choose the invocation $\textsc{Maaf}\,(F_1^i \div \{e_a\}, F_2^i \div \{e_a\}, k_i - 1)$ as the $(i+1)$st invocation, that is, $F_C = F_a$. If $d_1(a) < d_1(c)$ or $d_1(a) = d_1(c)$ and $\nu(a) > \nu(c)$, we choose the invocation $\textsc{Maaf}\,(F_1^i \div \{e_c\}, F_2^i \div \{e_c\}, k_i - 1)$ as the $(i+1)$st invocation, that is, $F_C = F_c$. This is illustrated in Figure 4.4.

Case 6.2. In this case, if $\textsc{Maaf}\,(F_1^i \div \{e_a, e_c\}, F_2^i \div \{e_a, e_c\}, k_i - 2)$ is viable, we choose it as the $(i+1)$st invocation. If the invocation $\textsc{Maaf}\,(F_1^i \div \{e_a, e_c\}, F_2^i \div \{e_a, e_c\}, k_i - 2)$ is not viable, then the invocations $\textsc{Maaf}\,(F_1^i, F_2^i \div \{e_b\}, k_i - 1)$ and $\textsc{Maaf}\,(F_1^i \div \{e_c\}, F_2^i \div \{e_c\}, k_i - 1)$ are both viable. In this case, we choose the latter as the $(i+1)$st invocation.

Case 6.3. Since there is more than one viable invocation in this case, at least one of the invocations $\textsc{Maaf}\,(F_1^i \div \{e_a\}, F_2^i \div \{e_a\}, k_i - 1)$ and $\textsc{Maaf}\,(F_1^i \div \{e_c\}, F_2^i \div \{e_c\}, k_i - 1)$ is viable. If exactly one of them is viable, we choose it to be the $(i+1)$st invocation. If both are viable, we define $\lambda_1(a)$ and $\lambda_1(c)$ as in Case 6.1. If $\lambda_1(a) \neq \lambda_1(c)$, we choose the $(i+1)$st invocation as in Case 6.1. If $\lambda_1(a) = \lambda_1(c)$, we define $\lambda_2(x)$ and $d_2(x)$, for $x \in \{a, c\}$, analogously to $\lambda_1(x)$ and $d_1(x)$ but using $\phi_2(\cdot)$ and T_2 in place of $\phi_1(\cdot)$ and T_1. Now we choose the $(i+1)$st invocation as in Case 6.1 but using $d_2(\cdot)$ instead of $d_1(\cdot)$.

Lemma 4.8. *If $hyb\,(T_1, T_2) \leqslant k_0$, then F_C can be refined to an AAF of T_1 and T_2 with at most $k_0 + 1$ 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 $k_0 + 1$ components. By Corollary 4.5, 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. Now let M be the set of nodes $m \in F_C$ such that every edge on the path from m to the root of its component in F_C is in E and m or its sibling in F_C is marked. We say an edge is *marked* if it belongs to the path from a node $m \in M$ to the root of its component, that is, if it is removed by fixing this node m. Next we prove that all edges in E are

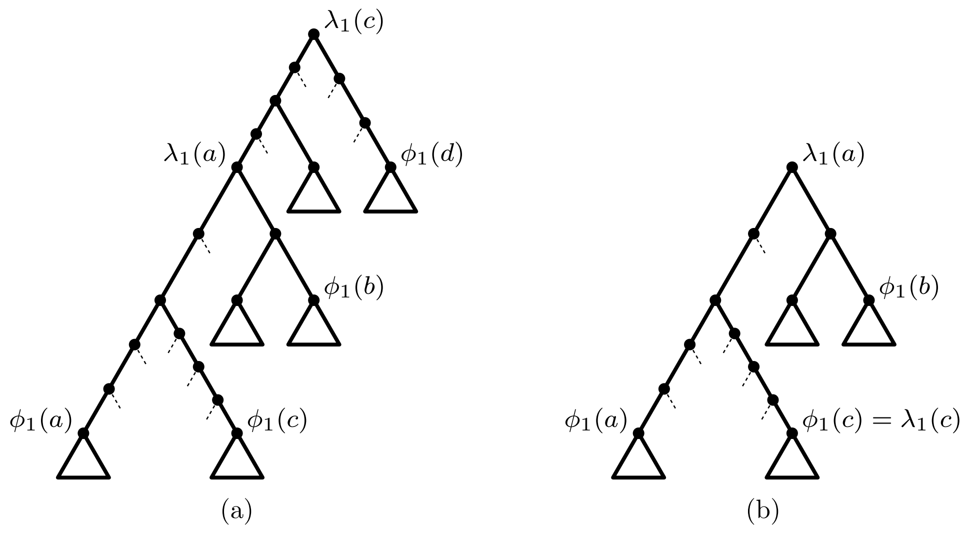


Figure 4.4: Two applications of Case 6.1 where we choose the invocation MAAF $(F_1^i \div \{e_a\}, F_2^i \div \{e_a\}, k_i - 1)$ on the path to F_C. Both figures show the relevant portion of T_1. Dotted edges have been removed to obtain F_1^i, making a and c siblings in F_1^i. The bold portion of T_1 yields F_1^i. Node b is a's sibling in F_c. In Figure (a), d is c's sibling in F_a, and the highest node $\lambda_1(c)$ on the path from $\phi_1(c)$ to $\phi_1(d)$ is an ancestor of the highest node $\lambda_1(a)$ on the path from $\phi_1(a)$ to $\phi_1(b)$. Hence, $d_1(a) > d_1(c)$. In Figure (b), $\phi_a(c)$ is assumed to be a root of F_a. Hence $\phi_1(c) = \lambda_1(c)$ and $d_1(a) > 0 = d_1(c)$.

marked. Since fixing a node or its sibling in F_C results in the same forest and every node in m is itself marked or has a marked sibling, this implies that there exists a subset of *marked* potential exit nodes such that fixing them produces F', that is, the refinement step applied to F_C finds F'.

Assume for the sake of contradiction that there is an unmarked edge in E. Since all ancestor edges of a marked edge are themselves marked, this implies that there exists a potential exit node $u \in G^*_{F_C}$ whose parent edge e_u belongs to E but is not marked, which in turn implies that neither u nor its sibling u' in F_C is marked. The sequence of invocations that produce F_C from T_1 and T_2 gives rise to a sequence of edges the algorithm cuts to produce F_C. For a step that cuts more than one edge, we cut these edges one by one. For Step 3 and branch (i) of Case 6.3, this ordering is chosen arbitrarily. For every branch of Step 6 that cuts an edge e_x with $x \in \{a, c\}$, we choose the ordering so that the parent edge of x in F_1 is cut immediately after cutting e_x in F_2. Finally, in branch (iii) of Case 6.2, we cut e_c after e_a. In the remainder of this proof, we use F_1^i and F_2^i to refer to the forests obtained from T_1 and T_2 after cutting the first i edges. (This is a slight change of notation from the definition of F_C, where we used F_1^i and F_2^i to denote the forests passed as arguments to the ith invocation.) Since F_C is a refinement of F_1^i and F_2^i, every node $x \in F_C$ maps to the lowest node y in F_j^i such that the labelled descendant leaves of x in F_C are descendants of y in F_j^i. 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_1^i and F_2^i 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 lowest common ancestor of both nodes in any forest F_j^i. 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 of the path from u to p_u in some forest F_j^i. There may also be a hybrid edge induced by cutting a pendant edge of the path from u' to p_u in some forest F_j^i. Either of these two types of edges are pendant to the path from u to u' in F_j^i. 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}. Since u and u' are siblings in F_C, the choice of index i implies that u and u' are siblings in F_1^i and F_2^i. In particular, y is the only pendant edge of the path from u to u' in F_j^{i-1} 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 (that is, $x' = u'$ if $x = u$ and vice versa). We make two observations about x, x', and y:

(i) Since fixing a node in F_C or its sibling produces the same forest, F' can be obtained from F_C by fixing a subset of nodes that includes x or x'. In particular, $\tilde{e}\left(T_1, T_2, F_j^i \div \{e_x\}\right) = \tilde{e}\left(T_1, T_2, F_j^i \div \{e_{x'}\}\right) = \tilde{e}\left(T_1, T_2, F_j^i\right) - 1$, for $j \in \{1, 2\}$.

(ii) Since edge e_u is not marked, neither u nor u' is marked, that is, x is not marked in F_C and, hence, y is not tagged with "T_j" in F_C.

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

First assume e_y belongs to F_1^{i-1}. Then e_y is cut by an application of Step 3 or e_y is the parent edge in F_1^{i-1} of a node $y \in \{a, c\}$ whose parent edge in F_2^{i-2} is the $(i-1)$st edge we cut. First assume the former. Then y is a root in F_2^{i-1}, which implies that there exists an $i' < i$ such that the i'th edge we cut is an edge e_z in $F_2^{i'-1}$ such that z is an ancestor of y in $F_2^{i'-1}$ and z is a node b or b_i in this application of Step 6. We choose the maximal such i'. This implies that no edge on the path from y to z is cut by any subsequent step. Indeed, if we cut such an edge in a forest $F_2^{i''-1}$, for $i' < i'' < i$, it would have to be an edge $e_{z'}$ with $z' \in \{a, c\}$, by the choice of i'. If $z' = y$, then e_y would be cut in Step 6; if $z' \neq y$, then e_y would belong to a subtree of $F_1^{i''-1}$ whose root is the member of a sibling pair, and e_y would never be cut. In either case, we obtain a contradiction. Now observe that any case of Step 6 that cuts an edge e_b or e_{b_i} tags b or b_i with "T_1", that is, z is tagged with "T_1" immediately after cutting e_z. Since we have just argued that no edges are cut on the path from z to y and y is a root in F_2^{i-1}, our rules for maintaining tags when suppressing nodes imply that y inherits z's "T_1" tag, a contradiction.

Now suppose e_y belongs to F_1^{i-1} and is cut in Step 6, that is, $y \in \{a, c\}$. Since Step 6 cuts an edge in F_1 immediately after cutting the corresponding edge in F_2, the $(i-1)$st edge we cut is y's parent edge in F_2^{i-2}. If e_y is cut by an application of Case 6.1, assume w.l.o.g. that $y = c$ and, hence, $x = a$. Since the invocation MAAF $\left(F_1^{i-2}, F_2^{i-2}, k\right)$ that cuts e_y is viable and $\tilde{e}\left(T_1, T_2, F_2^{i-2} \div \{e_x\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-2}\right) - 1$, the invocation MAAF $\left(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1\right)$ is also viable. Since we apply Case 6.1, x and x' are siblings in F_C, and F_C is a refinement of F_2^{i-2}, we have $x' \sim_{F_2^{i-2}} x \nsim_{F_2^{i-2}} y$. Since F_x is also a refinement of F_2^{i-2}, this implies that $x' \nsim_{F_x} y$. In particular, x' and y are not siblings in F_x. Since e_y is the only pendant edge of the path from x to x' in F_1^{i-2}, this implies that either y is a root in F_x or its parent in F_x is a proper ancestor in F_1^{i-2} of the common parent of x and x' in $F_y = F_C$. In both cases, $d_1(y) < d_1(x)$, contradicting that we chose the invocation MAAF $\left(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1\right)$ instead of the invocation MAAF $\left(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div \{e_x\}, k-1\right)$ on the path to F_C.

If e_y is cut by an application of Case 6.2, y is tagged with "T_1" unless $y = a$ and we apply the third branch of this case, or $y = c$ and we apply the second branch of this case. If $y = a$ and we apply the third branch, then $x = c$ and the $(i+2)$nd edge we cut is edge e_c in F_1^{i+1}, which contradicts the fact that $x = c$ has a sibling in F_C. If $y = c$ and we apply the second branch of this case, then $x = a$. However, since $\tilde{e}\left(T_1, T_2, F_C \div \{e_x\}\right) = \tilde{e}\left(T_1, T_2, F_C\right) - 1$ and we cut edge e_y to obtain F_C from F_2^{i-2}, we have in fact $\tilde{e}\left(T_1, T_2, F_2^{i-2} \div \{e_a, e_c\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-2}\right) - 2$, that is, the invocation Maaf $\left(F_1^{i-2} \div \{e_a, e_c\}, F_2^{i-2} \div \{e_a, e_c\}, k-2\right)$ is viable. This contradicts that we chose the invocation Maaf $\left(F_1^{i-2} \div \{e_c\}, F_2^{i-2} \div \{e_c\}, k-1\right)$ as the next invocation on the path to F_C.

Finally, suppose e_y is cut by an application of Case 6.3. If x' and y are not siblings in F_x, then the same argument as for Case 6.1 leads to a contradiction to the choice of F_C. So assume that x' and y are siblings in F_x, that is, that $d_1(x) = d_1(y)$. Since e_y is the last pendant edge of the path from x to x' in either of the two forests F_1 and F_2, x and x' are siblings in F_2^{i-1}. This implies that either x and x' are siblings also in F_2^{i-2} or e_y is the only pendant edge of the path from x to x' in F_2^{i-2}. In the first case, we have $d_2(y) < d_2(x)$, contradicting that we chose the invocation Maaf $\left(F_1^{i-2} \div \{e_y\}, F_2^{i-2} \div \{e_y\}, k-1\right)$ on the path to F_C, even though the invocation Maaf $\left(F_1^{i-2} \div \{e_x\}, F_2^{i-2} \div e_x, k-1\right)$ is viable. In the second case, cutting e_y in F_2^{i-2} tags y with "T_2". Since y is the sibling of x or x' in F_2^{i-2}, this implies that x or x' is marked in F_C, again a contradiction.

Finally, assume e_y belongs to F_2^{i-1}. Then e_y is cut by an application of Case 6.2 or Case 6.3 because Case 6.1 tags the bottom endpoint of each edge it cuts with "T_2", contradicting that y is not tagged with "T_2".

In Case 6.2, e_y is either e_b or e_c because, when edge e_a is cut, a is tagged with "T_2". First suppose $e_y = e_b$. Since e_y is the last pendant edge of the path from x to x' we cut in either of the two forests F_1 and F_2, we have $x = a$ and $x' = c$. However, since the current invocation Maaf $\left(F_1^{i-1}, F_2^{i-1}, k\right)$ is viable and $\tilde{e}\left(T_1, T_2, F_2^{i-1} \div \{e_c\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-1} \div \{e_{x'}\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-1}\right) - 1$, the invocation Maaf $\left(F_1^{i-1} \div \{e_c\}, F_2^{i-1} \div \{e_c\}, k-1\right)$ is also viable, which contradicts that we chose the invocation Maaf $\left(F_1^{i-1}, F_2^{i-1} \div \{e_b\}, k-1\right)$ as the next invocation on the path to F_C.

If $e_y = e_c$, it must be an application of branch (iii) of Case 6.2 that cuts e_y because branch (ii) tags c with "T_2". In this case, $x = b$ because we cut e_a before e_c. Then, however, b is a's sibling in F_2^{i-3} and the tail of a's T_2-hybrid edge. Since a is tagged with "T_2" in this case, this implies that $x = b$ is marked in F_C, a contradiction.

In Case 6.3 we tag a or c with "T_2". So e_y must be e_{b_h}, for some pendant edge e_{b_h} of the path from a to c in F_2^{i-q}. Along with the fact that e_y is the last pendant edge of the path from x to x' we cut, this implies that $x = c$ or $x' = c$.

Since the invocation $\textsc{Maaf}\left(F_1^{i-q}, F_2^{i-q}, k\right)$ that cuts edges $b_1, b_2, \ldots, b_q$ is viable and $\tilde{e}\left(T_1, T_2, F_2^{i-q} \div \{e_x\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-q} \div \{e_{x'}\}\right) = \tilde{e}\left(T_1, T_2, F_2^{i-q}\right) - 1$, the invocation $\textsc{Maaf}\left(F_1^{i-q} \div \{e_c\}, F_2^{i-q} \div \{e_c\}, k-1\right)$ is also viable, contradicting that we chose the invocation $\textsc{Maaf}\left(F_1^{i-q}, F_2^{i-q} \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\right)$ as the next invocation on the path to F_C. $\square$

By Lemma 4.8, the algorithm returns "yes" if $hyb\,(T_1, T_2) \leqslant k_0$, and it cannot return "yes" if $hyb\,(T_1, T_2) > k_0$. Thus, our MAAF algorithm is correct. Case 6.2 makes an additional recursive call compared to the algorithm from Section 4.3, but the number of recursive calls in this case is still given by the recurrence $I(k) = 2I(k-1) + I(k-2)$, which is also the worst case of Case 6.3 in the MAF algorithm (see Lemma 3.4). Thus, the number of recursive calls made during the branching phase of the algorithm remains $\mathrm{O}\left(2.42^{k_0}\right)$. Since at most k_0 of the potential exit nodes of an AF F found during the branching phase are marked (one per root of F other than ρ), $\textsc{Refine}\,(F, k_0)$ takes $\mathrm{O}\left(2^{k_0} n\right)$ 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 $k_0 + 1$ components. Thus, the total running time of the algorithm is $\mathrm{O}\left(2.42^{k_0}\left(n + 2^{k_0} n\right)\right) = \mathrm{O}\left(4.84^{k_0} n\right)$, and we obtain the following theorem.

Theorem 4.9. *For two rooted X-trees T_1 and T_2 and a parameter k_0, it takes $\mathrm{O}\left(4.84^{k_0} n\right)$ time to decide whether $\tilde{e}\,(T_1, T_2, T_2) \leqslant k_0$.*

4.5 Improved Refinement and Analysis

The algorithm we have developed so far finds 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 exit nodes in F. The algorithm then fixes every subset of these marked potential exit nodes for each agreement forest it finds. If k' is the number of edges we cut to obtain F, there are k' marked potential exit nodes and $2^{k'}$ subsets of marked potential exit nodes to check. When k' is small, the resulting time bound of $\mathrm{O}\left(2^{k'} n\right)$ for the refinement step is substantially better than the bound of $\mathrm{O}\left(2^{k} n\right)$ obtained using the naive upper bound of $k' \leqslant k$ we used so far. For large values of k', we observe that F has $k' + 1$ components because we always cut edges in a fully contracted forest (i.e., a forest without degree-2 vertices other than its component roots). When fixing a set of k'' potential exit nodes in the refinement step, we cut at least k'' edges, and this increases the number of connected components by at least k'', again because we cut edges along paths in fully contracted forests. Thus, if $k' + k'' > k$, we cannot possibly obtain an AAF with at most $k + 1$ components: the refinement step applied to F needs to consider only subsets of at most $k'' := k - k'$ potential exit nodes. Since there are k' marked potential exit nodes to choose from, this reduces the running time of the refinement step applied to such a forest F to $\mathrm{O}\left(\sum_{j=0}^{k''} \binom{k'}{j} n\right)$. For large values of k', k'' is small and the sum is significantly less than $\mathrm{O}\left(2^{k'} n\right) = \mathrm{O}\left(2^k n\right)$. Thus, we obtain a substantial improvement of the running time of the refinement step also in this case, without affecting its correctness. In summary, the only change to the MAAF algorithm from Section 4.4 we make in this section is to inspect all subsets of at most k'' marked potential exit nodes in the refinement step, where $k'' := \min(k', k - k')$.

To analyze the running time of our algorithm using this improved refinement step, we split each refinement step into several refinement steps. A refinement step that tries all subsets of between 0 and k'' marked potential exit nodes is replaced with $k'' + 1$ refinement steps: for $0 \leqslant j \leqslant k''$, the jth such refinement step tries all subsets of exactly j marked potential exit nodes. Its running time is therefore $\mathrm{O}\left(\binom{k'}{j} n\right)$, and the total cost of all refinement steps remains unchanged. Now we partition the refinement steps invoked for the different AFs found during the branching phase into $k + 1$ groups. For $0 \leqslant h \leqslant k$, the hth group contains a refinement step applied to an agreement forest F if the number k' of edges cut to obtain F and the size j of the subsets of marked potential exit nodes the refinement step tries satisfy $k' + j = h$. We prove that the total running time of all refinement steps in the hth group is $\mathrm{O}\left(3.18^h n\right)$. Hence, the total running time of all refinement steps is $\mathrm{O}\left(\sum_{h=0}^{k} 3.18^h n\right) = \mathrm{O}\left(3.18^k n\right)$, which dominates the $\mathrm{O}\left(2.42^k n\right)$ time bound of the branching phase, that is, the running time of the entire MAAF algorithm is $\mathrm{O}\left(3.18^k n\right)$.

Now consider the tree of recursive calls made in the branching phase. Since a given invocation $\textsc{Maaf}\left(F_1, F_2, k''\right)$ spawns further recursive calls only if F_2 is not an AF of T_1 and T_2, and we invoke the refinement step on F_2 only if F_2 is an AF of T_1 and T_2, refinement steps are invoked only from the leaves of this recursion tree. Moreover, since every refinement step in the hth group satisfies $k' + j = h$ and, hence,

$k' \leqslant h$, 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, to bound the running time of the refinement steps in the hth group, we can restrict our attention to the subtree of the recursion tree containing all recursive calls $\textsc{Maaf}\,(F_1, F_2, k'')$ such that F_2 can be obtained from T_2 by cutting at most h edges, that is, $k'' \geqslant d := k - h$. Since we want to obtain an upper bound on the cost of the refinement steps in the hth group, we can assume that the shape of this subtree and the set of refinement steps invoked from its leaves are such that the total cost of the refinement steps is maximized. We construct such a worst-case recursion tree for the refinement steps in the hth group in two steps.

First we construct a recursion tree without refinement steps and such that, for each $d \leqslant k'' \leqslant k$, the number of invocations with parameter k'' in this tree is maximized. As in the proof of Lemma 3.4, this is the case if each recursive call with parameter $k'' \geqslant d + 2$ makes three recursive calls, two with parameter $k'' - 1$ and one with parameter $k'' - 2$, and each recursive call with parameter $k'' = d + 1$ makes two recursive calls with parameter $k'' - 1$. As in the proof of Lemma 3.4, this implies that every recursive call with parameter k'' has a tree of $\Theta\left(\left(1+\sqrt{2}\right)^{k''-d}\right)$ recursive calls below it, and the size of the entire tree is $\mathrm{O}\left(\left(1+\sqrt{2}\right)^{k-d}\right) = \mathrm{O}\left(\left(1+\sqrt{2}\right)^{h}\right)$. The second step is to choose a subset of recursive calls in this tree for which we invoke the refinement step *instead of* spawning further recursive calls, thereby turning them into leaves. In effect, for each such node with parameter k'', we replace its subtree of $\Theta\left(\left(1+\sqrt{2}\right)^{k''-d}\right)$ recursive calls with a single refinement step of cost $\mathrm{O}\left(\binom{k'}{j} n\right)$, where $k' := k - k'' = h + d - k''$ and $j := h - k' = k'' - d$. By charging the cost of this refinement step equally to the nodes in the removed subtree, each node in this subtree is charged a cost of $\Theta\left(\binom{k'}{j} n / \left(1+\sqrt{2}\right)^{k''-d}\right) = \Theta\left(\binom{k'}{j} n / \left(1+\sqrt{2}\right)^{j}\right)$. The total running time of all refinement steps in the hth group is the sum of the charges of all nodes removed from the recursion tree. Since we can remove at most $\mathrm{O}\left(\left(1+\sqrt{2}\right)^{h}\right)$ nodes from the tree, the cost of all refinement steps in the hth group is therefore

$$\mathrm{O}\left(\left(1+\sqrt{2}\right)^{h} \frac{\binom{k'}{j} n}{\left(1+\sqrt{2}\right)^{j}}\right) = \mathrm{O}\left(\left(1+\sqrt{2}\right)^{k'} \binom{k'}{j} n\right), \tag{4.1}$$

where k' and j are chosen so that $\binom{k'}{j} / \left(1+\sqrt{2}\right)^j$ is maximized subject to the constraints $0 \leqslant j \leqslant k'$ and $k'+j = h$. It remains to bound this expression by $\mathrm{O}\left(3.18^h n\right)$. First assume that $k' \leqslant 2h/3$. Then we can bound $\binom{k'}{j}$ by $2^{k'}$, and $\left(1+\sqrt{2}\right)^{k'} \cdot \binom{k'}{j}$ by $\left(2+2\sqrt{2}\right)^{k'} \leqslant 4.84^{2h/3} \leqslant 2.87^h$, that is, (4.1) is bounded by $\mathrm{O}\left(2.87^h n\right)$. For $k' = h$, we have $j = 0$ and, hence, (4.1) is bounded by $\mathrm{O}\left(2.42^h n\right)$ in this case. To bound (4.1) for $2h/3 < k' < h$, we make use of the following observation.

Observation 4.10. $\displaystyle\binom{x}{y} = \mathrm{O}\left(\left(\frac{x}{y}\right)^y \left(\frac{x}{x-y}\right)^{x-y}\right).$

Observation 4.10 allows us to bound (4.1) by

$$\mathrm{O}\left(\left(1+\sqrt{2}\right)^{k'} \left(\frac{k'}{j}\right)^j \left(\frac{k'}{k'-j}\right)^{k'-j} n\right) = \mathrm{O}\left(\left(\left(1+\sqrt{2}\right)^{\alpha} \left(\frac{\alpha}{1-\alpha}\right)^{1-\alpha} \left(\frac{\alpha}{2\alpha-1}\right)^{2\alpha-1}\right)^h n\right),$$

where $\alpha := k'/h$ and, hence, $k' = \alpha h$ and $j = (1-\alpha)h$. It remains to determine the value of α such that $2/3 < \alpha < 1$ and the function

$$b(\alpha) = \left(1+\sqrt{2}\right)^{\alpha} \left(\frac{\alpha}{1-\alpha}\right)^{1-\alpha} \left(\frac{\alpha}{2\alpha-1}\right)^{2\alpha-1}$$

is maximized. Taking the derivative and setting to zero, we obtain that $b(\alpha)$ is maximized for $\alpha = \frac{1}{2} + \frac{\sqrt{7+6\sqrt{2}}}{10+2\sqrt{2}}$, which gives $b(\alpha) \leqslant 3.18$. This finishes the proof that the total cost of the refinement steps in the hth group is $\mathrm{O}\left(3.18^h n\right)$, which, as we argued already, implies that the running time of the entire algorithm is $\mathrm{O}\left(3.18^k n\right)$. Thus, we have the following theorem.

Theorem 4.11. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left(3.18^k n\right)$ time to decide whether $\tilde{e}\left(T_1, T_1, T_2\right) \leqslant k$.*

As with the MAF algorithms, we can use known kernelization rules [24] to transform the trees T_1 and T_2 into two trees T_1' and T_2' of size $\mathrm{O}\left(e\left(T_1, T_2, T_2\right)\right)$. However, unlike the kernelization rules used for SPR distance, these kernelization rules produce trees that do *not* have the same hybridization number as T_1 and T_2. One of these rules, the *Chain Reduction*, replaces a chain of leaves $a_1, a_2, \ldots$ with a pair of leaves a, b. Bordewich and Semple [24] showed that in an MAAF of the resulting two trees, either a and b are both isolated or neither is. A corresponding MAAF of T_1 and T_2 can be obtained by cutting the parent edges of $a_1, a_2, \ldots$ in the first case or replacing a and b with the sequence of leaves $a_1, a_2, \ldots$ in the second case. The difference in size between these two MAAFs is captured by assigning the number of leaves removed by the reduction as a weight to the pair (a, b). The weight of an AAF of the two reduced trees T_1' and T_2' then is the number of components of the AAF plus the weights of all such pairs (a, b) such that a and b are isolated in the AAF. This weight equals the size of the corresponding AAF of T_1 and T_2.

It is not difficult to incorporate these weights into our MAAF algorithm. Whenever the refinement algorithm would return "yes", we first add the sum of the weights of isolated pairs to the number of components in the found AAF. If, and only if, this total is less than or equal to k_0, we return "yes". Any AF F of T_1' and T_2' with weight $w(F) = \tilde{e}\,(T_1, T_2, T_2)$ has at most $w(F)$ components and thus will be examined by this strategy. Similarly, the depth of the recursion is bounded by the number of components, and thus by k_0. Thus, we obtain the following corollary.

Corollary 4.12. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left(3.18^k k + n^3\right)$ time to decide whether $\tilde{e}\,(T_1, T_2, T_2) \leqslant k$.*

Chapter 5

Incorporating Uncertainty: Computing MAFs of Multifurcating Trees in $\mathrm{O}\left(2.42^k n\right)$-time

Recall that multifurcations (or *polytomies*) are vertices of a tree with two or more children. A multifurcation is *hard* if it indeed represents an inferred common ancestor which produced three or more species as direct descendants; it is *soft* if it simply represents ambiguous evolutionary relationships [66]. Simultaneous speciation events are assumed to be rare, so a common assumption is that all multifurcations are soft. If we force the resolution of multifurcating trees into binary trees, then we infer evolutionary relationships that are not supported by the original data and may infer meaningless reticulation events. Thus, it is crucial to develop efficient algorithms to compare multifurcating trees directly.

In this chapter, we extend the approximation and fixed-parameter algorithms for computing MAFs of binary rooted trees from Chapter 3 to multifurcating trees (thus showing that computing MAFs for multifurcating trees is fixed-parameter tractable). The size of an MAF of two binary trees is equivalent to their SPR distance. In keeping with the assumption that multifurcations are soft, we define an MAF of two multifurcating trees so that its size is equivalent to what we call the *soft* SPR distance: the minimum number of SPR operations required to transform a binary resolution of one tree into a binary resolution of the other. This distinction avoids the inference of meaningless differences between the trees that arises, for example, when one tree has a set of resolved bifurcations that are part of a multifurcation in the other tree. This is similar to the extension of the hybridization number to multifurcating trees by Linz and Semple [62]. At the time of writing this chapter has been submitted for publication but has not yet been published.

Our fixed-parameter algorithm achieves the same running time as in the binary case. Our approximation algorithm achieves the same approximation factor as in the binary case at the cost of increasing the running time from linear to $\mathrm{O}(n \lg n)$.

These results are not trivial extensions of the algorithm for binary trees. They require new structural insights and a novel method for terminating search branches of the depth-bounded search tree, coupled with a careful analysis of the resulting recurrence relation.

The rest of this chapter is organized as follows. Section 5.1 introduces the necessary terminology and notation for agreement forests of multifurcating trees. Section 5.2 presents the key structural results for multifurcating agreement forests. Sections 5.3 and 5.4 present our fixed-parameter and approximation algorithms, respectively, based on these results. Finally, in Section 5.5, we present closing remarks and discuss open problems and possible extensions of this work.

5.1 Preliminaries

To represent and analyze multifurcating trees we require several new definitions and notation. In this section we reintroduce several concepts from Chapter 2 for context and extend these concepts to multifurcating trees.

Recall that 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; see Figure 5.1(a). T is *binary* (or *bifurcating*) if all internal nodes have exactly two children each; otherwise it is *multifurcating*. For a subset V of X, $T(V)$ and $T|V$ generalize to nonbinary trees as in Figures 5.1(b)and 5.1(c). An *expansion* is the opposite of a contraction: It splits a node v into two nodes v_1 and v_2 such that v_1 is v_2's parent and divides the children of v 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.

Let T_1 and T_2 be two X-trees. We say that T_2 *resolves* T_1 or, equivalently, T_2 is a *resolution* of T_1 if T_1 can be obtained from T_2 by contracting internal edges. T_2 is a *binary resolution* of T_1 if T_2 is binary. See Figure 5.1(d).

A *subtree prune-and-regraft* (SPR) operation on a binary rooted X-tree was defined in Chapter 2. It cuts an edge xp_x, dividing T into subtrees T_x and T_{p_x}. Then it introduces a node p'_x into T_{p_x} by subdividing an edge of T_{p_x} and adds an edge xp'_x, thereby making x a child of p'_x, and, finally, p_x is removed using a contraction. On

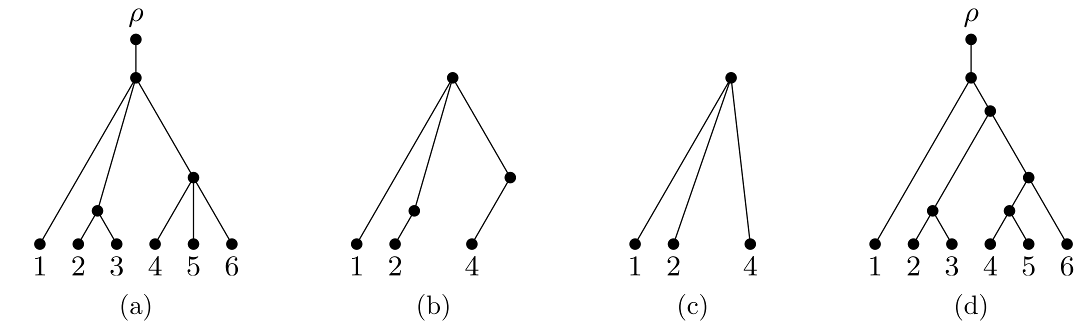


Figure 5.1: (a) An X-tree T with a multifurcation containing leaves 4, 5 and 6. (b) The subtree $T(V)$ for $V = \{1, 2, 4\}$. (c) $T|V$. (d) A binary resolution of T.

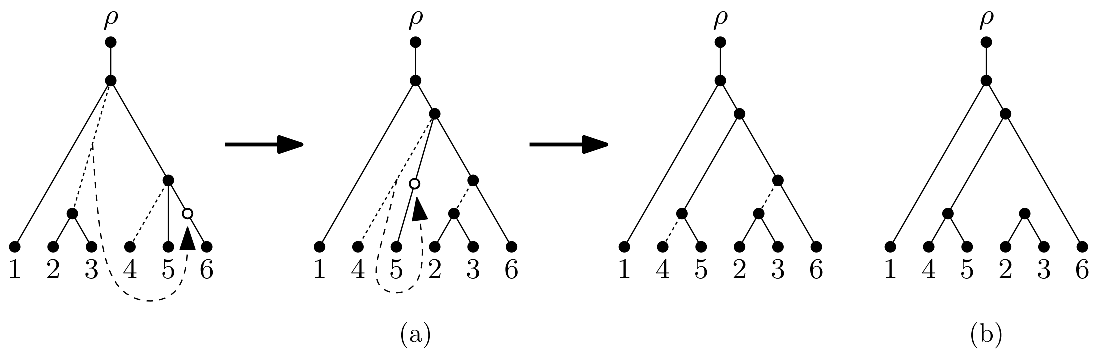


Figure 5.2: (a) SPR operations transforming the tree T from Figure 5.1(a) into another tree. Each operation changes the top endpoint of one of the dotted edges. The hard SPR distance between the two trees is 2. (b) The MAF representing only the first transfer of (a). The second transfer is unnecessary if the multifurcation represents ambiguous data rather than simultaneous speciation. Thus, the soft SPR distance is 1.

a multifurcating tree, an SPR operation may also use any existing node of T_{p_x} as p'_x and contracts p_x only if it has only one child besides x.

For binary trees, SPR operations gave rise to the distance measure $d_{SPR}(\cdot, \cdot)$, defined as the minimum number of such operations required to transform one tree into the other. An analogous distance measure, which we call the *hard SPR distance*, could be defined for multifurcating X-trees; however, under the assumption that most multifurcations are soft, this would capture differences between the trees that are meaningless. Instead, we define the *soft SPR distance* $d_{SPR}(T_1, T_2)$ between two multifurcating trees T_1 and T_2 to be the minimum SPR distance of all pairs of binary resolutions of T_1 and T_2.[1] For simplicity, we simply refer to this as the SPR distance

[1]This is similar to the generalization of the hybridization number used by Linz and Semple [62]

in the remainder of this chapter. These two distance measures are illustrated in Figure 5.2. Note that the soft SPR distance is not a metric but captures the minimum number of SPR operations needed to explain the difference between the two trees.

Given X-trees T_1 and T_2 and forests F_1 of T_1 and F_2 of T_2, a forest F is an agreement forest of F_1 and F_2 if it is a forest of a binary resolution of F_1 and of a binary resolution of F_2. $F \setminus E$ and $F \div E$ generalize from binary forests. As with binary trees, F is an MAF of F_1 and F_2 if there is no AF of F_1 and F_2 with fewer components. An MAF of two trees is shown in Figure 5.2(b). We continue to denote the number of components in an MAF of F_1 and F_2 by $m(F_1, F_2)$, and now $e(F_1, F_2, F)$ is the size of the smallest edge set E such that $F' \div E$ is an AF of F_1 and F_2, where F is a forest of F_2 and F' is a binary resolution of F. Recall that Bordewich and Semple [23] showed that, for two *binary* rooted X-trees T_1 and T_2, $d_{SPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$. This implies that $d_{SPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$ for two arbitrary rooted X-trees 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. Thus, to determine the SPR distance between two rooted X-trees, we need to compute a binary MAF of the two trees.

Much of our previous notation also generalizes to multifurcating trees, such as $a \sim_F b$, denoting that there exists a path from a to b in F, and F^x, the subtree of F induced by all descendants of x, inclusive. The definition of a sibling pair (a, c) remains unchanged, it is a pair of siblings of a forest F_1 such that a and c exist in another forest F_2. Figure 5.3 shows such a sibling pair. However, we denote such a sibling pair by $\{a, c\}$ in this chapter, for consistency with its generalization. We say $\{a_1, a_2, \ldots, a_m\}$ is a *sibling group* if $\{a_i, a_j\}$ is a sibling pair of F_1, for all $1 \leqslant i < j \leqslant m$, and a_1 has no sibling not in the group.

The correctness proofs of our algorithms in the next sections make use of the following three lemmas. Recall that Lemma 2.1 was shown by Bordewich et al. [22] for binary trees. The proof trivially extends to multifurcating trees. We restate the lemma here:

Lemma 2.1 (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 closer to e, and v_e an end vertex of e. If (1) $v_f \sim_{F-E} v_e$ and (2) $x \nsim_{F-(E \cup \{e\})} v_f$, for all $x \in X$, then $F \div E = F \div (E \setminus \{f\} \cup \{e\})$.*

Let F_1 and F_2 be forests of X-trees T_1 and T_2, respectively. Any agreement forest of F_1 and F_2 is clearly also an agreement forest of T_1 and T_2. Conversely, an agreement forest of T_1 and T_2 is an agreement forest of F_1 and F_2 if it is a forest of F_2 and there are no two leaves a and b such that $a \sim_{F_2} b$ but $a \nsim_{F_1} b$. This is formalized in the following lemma which extends Lemma 2.2. Our algorithms ensure that any intermediate forests F_1 and F_2 they produce have this latter property. Thus, this lemma allows us to reason about agreement forests of F_1 and F_2 and of T_1 and T_2 interchangeably, as long as they are forests of F_2.

Lemma 5.1. *Let F_1 and F_2 be forests of X-trees T_1 and T_2, respectively. Let F_1 be the union of trees $\dot{T}_1, \dot{T}_2, \ldots, \dot{T}_k$ and F_2 be the union of forests $\dot{F}_1, \dot{F}_2, \ldots, \dot{F}_k$ such that $\dot{T}_i$ and $\dot{F}_i$ have the same label set, for all $1 \leqslant i \leqslant k$. Let F_2' be a resolution of F_2. $F_2' \div E$ is an AF of T_1 and T_2 if and only if it is an AF of F_1 and F_2.*

To use Lemma 2.1 to prove structural properties of agreement forests, which are defined in terms of resolutions of forests, we also need the following lemma, which specifies when an expansion does not change the SPR distance.

Lemma 5.2. *Let F_1 and F_2 be resolutions of forests of rooted X-trees T_1 and T_2, and let $F \div E$ be a maximum agreement forest of F_1 and F_2, where F is a binary resolution of F_2. Let $a_1, a_2, \ldots, a_p, a_{p+1}, \ldots, a_m$ be the children of a node in F_2 and let F_2' be the result of expanding $\{a_{p+1}, a_{p+2}, \ldots, a_m\}$ in F_2. If $a_i' \nsim_{F \div E} a_j'$, for all $1 \leqslant i \leqslant p$, $p+1 \leqslant j \leqslant m$, and all leaves $a_i' \in F_2^{a_i}$ and $a_j' \in F_2^{a_j}$, then $e(F_1, F_2, F_2) = e(F_1, F_2, F_2')$.*

Proof. The only difference between F_2 and F_2' is the expansion of $\{a_{p+1}, a_{p+2}, \ldots, a_m\}$ in F_2', so $e(F_1, F_2, F_2) \leqslant e(F_1, F_2, F_2')$. Since $F \div E$ is an MAF of F_1 and F_2, it suffices to show that $F \div E$ is an AF of F_1 and F_2' to prove that $e(F_1, F_2, F_2') \leqslant e(F_1, F_2, F_2)$. Assume the contrary. Then, since $F \div E$ is a forest of F_1, it cannot be a forest of F_2'. Since the only difference between F_2 and F_2' is the expansion of $\{a_{p+1}, a_{p+2}, \ldots, a_m\}$, this implies that some component of $F \div E$ contains leaves $a_i' \in F_2^{a_i}$ and $a_j' \in F_2^{a_j}$, for some $1 \leqslant i \leqslant p$ and $p+1 \leqslant j \leqslant m$, contradicting that $a_i' \nsim_{F \div E} a_j'$ for all such leaves. □

A *triple* $ab|c$ of a rooted forest F is again defined by a set $\{a, b, c\}$ of three leaves in the same component of F and such that the path from a to b in F is disjoint from the

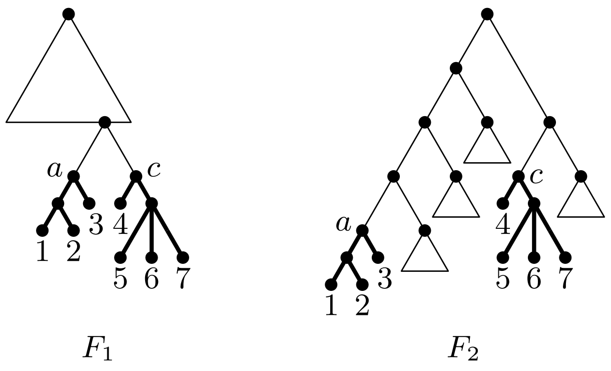


Figure 5.3: A sibling pair $\{a, c\}$ of two forests F_1 and F_2: a and c have a common parent in F_1, and both subtrees F_1^a and F_1^c exist also in F_2.

path from c to the root of the component. Multifurcating trees also allow for triples $a|b|c$ where a, b, and c share the same lowest common ancestor (LCA). A triple $ab|c$ of a forest F_1 is *compatible* with a forest F_2 if it is also a triple of F_2 or F_2 contains the triple $a|b|c$; otherwise it is *incompatible* with F_2.

An agreement forest of two forests F_1 and F_2 cannot contain a triple incompatible with either of the two forests. Thus, Observation 2.3 extends to multifurcating trees:

Observation 2.3. *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 triple of F_1 incompatible with F_2, then $a \nsim_F b$ or $a \nsim_F c$.*

Recall that, For two forests F_1 and F_2 with the same label set, two components C_1 and C_2 of F_2 are said to *overlap* in F_1 if there exist leaves $a, b \in C_1$ and $c, d \in C_2$ such that the paths from a to b and from c to d in F_1 exist and are not edge-disjoint. Lemma 2.4 thus extends to multifurcating trees:

Lemma 2.4. *Let F_1 and F_2 be binary resolutions of forests of two X-trees T_1 and T_2, and denote the label sets of the components of F_1 by $X_1, X_2, \dots, X_k$ and the label sets of the components of F_2 by $Y_1, Y_2, \dots, Y_l$. 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 triple of F_2 is incompatible with F_1.*

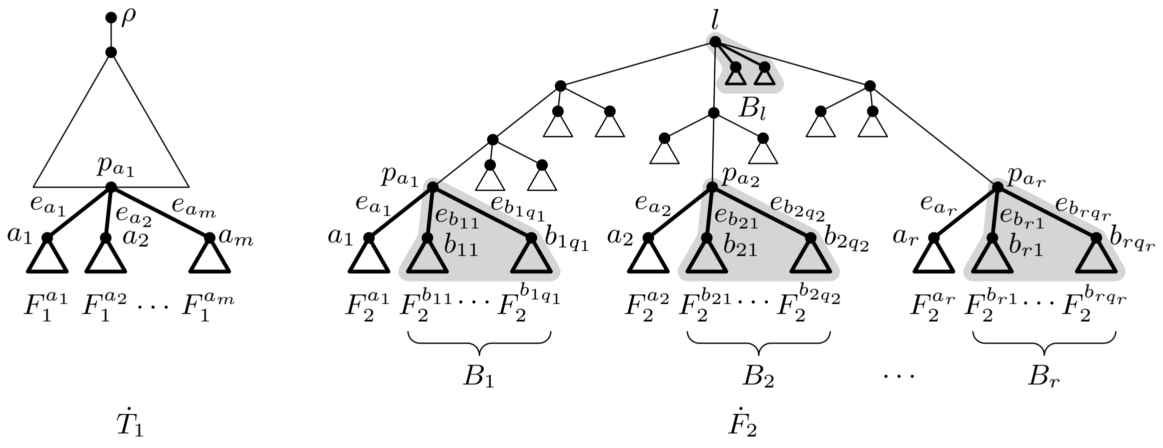

ρ
p_{a_1}
e_{a_1} e_{a_2} e_{a_m}
a_1 a_2 a_m
$F_1^{a_1}$ $F_1^{a_2} \cdots F_1^{a_m}$
$\dot{T}_1$
l
B_l
p_{a_1} e_{a_1} $e_{b_{11}}$ $e_{b_{1q_1}}$ a_1 b_{11} b_{1q_1}
$F_2^{a_1}$ $F_2^{b_{11}} \cdots F_2^{b_{1q_1}}$
B_1
p_{a_2} e_{a_2} $e_{b_{21}}$ $e_{b_{2q_2}}$ a_2 b_{21} b_{2q_2}
$F_2^{a_2}$ $F_2^{b_{21}}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

Figure 5.4: Tree labels for a sibling group $\{a_1, a_2, \ldots, a_m\}$ such that $a_1, a_2, \ldots, a_r$ share a minimal LCA l.

5.2 The Structure of Multifurcating Agreement Forests

This section presents the structural results that provide the intuition and formal basis for the algorithms presented in Section 5.3 and 5.4. All these algorithms start with a pair of trees (T_1, T_2) and then cut edges, expand sets of nodes, remove agreeing components from consideration, and merge sibling pairs until the resulting forests are identical. The intermediate state is that T_1 and T_2 have been resolved and reduced to forests F_1 and F_2, respectively. F_1 consists of a tree $\dot{T}_1$ and a set, F_0, of components that exist in F_2. F_2 has two sets of components. One is F_0. The other, $\dot{F}_2$, has the same label set as $\dot{T}_1$ but may not agree with $\dot{T}_1$. 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 MAF of T_1 and T_2. The results in this section identify small edge sets in $\dot{F}_2$ such that at least one edge in each of these sets has the property that cutting it reduces $e\left(T_1, T_2, F_2\right)$ by one. Some of these edges are introduced by expanding nodes. The approximation algorithm cuts all edges in the identified set, and the size of the set gives the approximation ratio of the algorithm. The FPT algorithm tries each edge in the set in turn, so that the size of the set gives the branching factor for a depth-bounded search algorithm.

Let $\{a_1, a_2, \ldots, a_m\}$ be a sibling group of $\dot{T}_1$. If there exist indices $i \neq j$ such that a_i and a_j are also siblings in F_2, we can expand this sibling pair $\{a_i, a_j\}$ and replace a_i and a_j with their 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, then we can cut a_i's parent edge in F_1, thereby removing a_i from the sibling group. Thus, we can assume a_i and a_j are not siblings in F_2, for all $1 \leqslant i < j \leqslant m$, and $F_2^{a_i}$ is not a component of F_2, for all $1 \leqslant i \leqslant m$. We have $a_i \in \dot{F}_2$, for all $1 \leqslant i \leqslant m$, because $\dot{T}_1$ and $\dot{F}_2$ have the same label set. Let $B_i = \{b_{i1}, b_{i2}, \dots b_{iq_i}\}$ be the siblings of a_i in F_2, for $1 \leqslant i \leqslant m$. We use e_x to denote the edge connecting a node x to its parent p_x, e_{B_i} to denote the edge introduced by expanding B_i, and p_{B_i} to denote the common parent of the nodes in B_i. $F_2 - \{e_{B_i}\}$ denotes the forest obtained from F_2 by expanding B_i and then cutting e_{B_i}, and we use $F_2^{B_i}$ to denote the subforest of F_2 comprised of the subtrees $F_2^{b_{i1}}, F_2^{b_{i2}}, \dots, F_2^{b_{iq_i}}$.

Consider a subset $\{a_{i_1}, a_{i_2}, \dots, a_{i_r}\}$ of a sibling group $\{a_1, a_2, \dots, a_m\}$. We say $a_{i_1}, a_{i_2}, \dots, a_{i_r}$ *share their LCA l* if $l = LCA_{F_2}(a_i, a_j)$, for all $i, j \in \{i_1, i_2, \dots, i_r\}$, $i \neq j$. If, in addition, $LCA_{F_2}(a_i, a_j)$ is not a proper descendant of l, for all $1 \leqslant i < j \leqslant m$, we say that $a_{i_1}, a_{i_2}, \dots, a_{i_r}$ *share a minimal LCA l*. For simplicity, we always order the elements of the group so that $\{a_{i_1}, a_{i_2}, \dots, a_{i_r}\} = \{a_1, a_2, \dots, a_r\}$ and assume the subset that shares l is maximal, that is, a_i is not a descendant of l, for all $r < i \leqslant m$. We use B_l to denote the set of children of l that do not have any member a_i of the sibling group as a descendant. Note that $B_l \subseteq B_i$ when a_i is a child of l. These labels are illustrated in Figure 5.4.

Our first result shows that at least one of the edges e_{a_1}, e_{a_2}, e_{B_1}, and e_{B_2} has the property that cutting it reduces $e\,(T_1, T_2, F_2)$ by one. This implies that cutting e_{a_1}, e_{a_2}, $e_{p_{a_1}}$, and $e_{p_{a_2}}$ reduces $e\,(T_1, T_2, F_2)$ by at least 1.

Theorem 5.3. *Let F_1 and F_2 be forests of rooted X-trees T_1 and T_2, respectively, and assume F_1 consists of a tree $\dot{T}_1$ and a set of components that exist in F_2. Let $\{a_1, a_2, \dots, a_m\}$ be a sibling group of $\dot{T}_1$ such that either $a_1, a_2, \dots, a_r$ share a minimal LCA l in F_2 or $a_2, a_3, \dots, a_r$ share a minimal LCA l in F_2 and $a_1 \nsim_{F_2} a_i$, for all $2 \leqslant i \leqslant m$; a_1 is not a child of l; a_2 is not a child of l unless $r = 2$; a_i and a_j are not siblings in F_2, for all $1 \leqslant i < j \leqslant m$; and $F_2^{a_i}$ is not a component of F_2, for all $1 \leqslant i \leqslant m$. Then*

(i) $e\,(T_1, T_2, F_2 - \{e_x\}) = e\,(T_1, T_2, F_2) - 1$, *for some* $x \in \{a_1, a_2, B_1, B_2\}$.

(ii) $e\left(T_1, T_2, F_2 - \{e_{a_1}, e_{a_2}, e_{p_{a_1}}, e_{p_{a_2}}\}\right) \leqslant e\,(T_1, T_2, F_2) - 1$.

Proof. (ii) follows immediately from (i) because cutting $\{e_{a_1}, e_{a_2}, e_{p_{a_1}}, e_{p_{a_2}}\}$ is equivalent to cutting $\{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}$. For (i), it suffices to prove that there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an MAF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \emptyset$.

So assume $F \div E$ is an MAF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} = \emptyset$. By Lemma 5.1, $F \div E$ is also an MAF of F_1 and F_2. We prove that we can replace some edge $f \in E$ with an edge in $\{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}$ without changing $F \div E$.

First assume $b_1' \nsim_{F \div E} x$, for all leaves $b_1' \in F_2^{B_1}$ and $x \notin F_2^{B_1}$. By Lemma 5.2, expanding B_1 does not change $e(F_1, F_2, F_2)$, so we can assume F contains this expansion.[2] Now we choose an arbitrary leaf $b_1' \in B_1$ and the first edge $f \in E$ on the path from p_{B_1} to b_1'. By Lemma 2.1, $F \div E = F \div (E \setminus \{f\} \cup \{e_{B_1}\})$. If $b_2' \nsim_{F \div E} x$, for all leaves $b_2' \in F_2^{B_2}$ and $x \notin F_2^{B_2}$, $a_1' \nsim_{F \div E} p_{a_1}$, for all leaves $a_1' \in F_2^{a_1}$, or $a_2' \nsim_{F \div E} p_{a_2}$, for all leaves $a_2' \in F_2^{a_2}$, then the same argument shows that $F \div E = F \div (E \setminus \{f\} \cup \{e_x\})$, for $x = B_2$, $x = a_1$, and $x = a_2$, respectively. Thus, we can assume there exist leaves $a_1' \in F_2^{a_1}$, $a_2' \in F_2^{a_2}$, $b_1' \in F_2^{B_1}$, and $b_2' \in F_2^{B_2}$ such that $a_1' \sim_{F \div E} p_{a_1} \sim_{F \div E} b_1'$ and $a_2' \sim_{F \div E} p_{a_2} \sim_{F \div E} b_2'$.

Now recall that either $a_1, a_2, \ldots, a_r$ share the minimal LCA l and a_1 is not a child of l or $a_2, a_3, \ldots, a_r$ share the minimal LCA l and $a_1 \nsim_{F_2} a_i$, for all $2 \leqslant i \leqslant m$. In either case, $a_i \notin F_2^{B_1}$, for all $1 \leqslant i \leqslant m$. Since a_i also is not an ancestor of p_{B_1}, this shows that $b_1' \notin F_2^{a_i}$, for all $1 \leqslant i \leqslant m$. Thus, F_1 contains the triple $a_1'a_2'|b_1'$, while F_2 contains the triple $a_1'b_1'|a_2'$ or $a_1' \nsim_{F_2} a_2'$. By Observation 2.3, this implies that $a_1' \sim_{F \div E} p_{a_1} \sim_{F \div E} b_1' \nsim_{F \div E} a_2' \sim_{F \div E} p_{a_2} \sim_{F \div E} b_2'$ and, hence, $b_2' \notin F_2^{a_1}$. Since a_2 is not a child of l unless $r = 2$, we also have $b_2' \notin F_2^{a_i}$, for all $2 \leqslant i \leqslant m$. Thus, F_1 also contains the triple $a_1'a_2'|b_2'$, which implies that the components of $F \div E$ containing a_1', b_1' and a_2', b_2' overlap in F_1, a contradiction. □

Theorem 5.3 covers every case where some minimal LCA l exists. If there is no such minimal LCA, then each a_i must be in a separate component of F_2. In the following lemma we show the stronger result that cutting e_{a_1} or e_{a_2} reduces $e(T_1, T_2, F_2)$ by one in this case (which immediately implies that claims (i) and (ii) of Theorem 5.3 also hold in this case).

[2]In fact, using the same ideas as in the proof of Lemma 5.2, it is not difficult to see that this expansion never precludes obtaining the same forest $F \div E$ by cutting a different set of $|E|$ edges. We discuss the importance of this to hybridization and reticulate analysis in Section 5.5.

Lemma 5.4 (Isolated Siblings). *If $a_1 \nsim_{F_2} a_i$, for all $i \neq 1$, $a_2 \nsim_{F_2} a_j$, for all $j \neq 2$, and $F_2^{a_i}$ is not a component of F_2, for all $1 \leqslant i \leqslant m$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$.*

Proof. Consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F \div E$ is an AF of F_1 and F_2, and assume E is chosen so that $|E \cap \{e_{a_1}, e_{a_2}\}|$ is maximized. Assume for the sake of contradiction that $E \cap \{e_{a_1}, e_{a_2}\} = \varnothing$. Then, by the same arguments as in the proof of Theorem 5.3, 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 $\{a_1, a_2, \ldots, a_m\}$ 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 $a'_1 \nsim_{F-E} x$, for all leaves $x \notin F_2^{a_1}$, or $a'_2 \nsim_{F-E} x$, for all leaves $x \notin F_2^{a_2}$. W.l.o.g. assume the former. Since $F_2^{a_1}$ is not a component of F_2, there exists a leaf $y \notin F_2^{a_1}$ such that $a_1 \sim_{F_2} y$ and, hence, $a'_1 \sim_{F_2} y$. For each such leaf y, the path from a'_1 to y in F contains an edge in E because $a'_1 \nsim_{F \div E} y$, 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 y, and let f be the first edge in E on the path from a'_1 to y. The edges e_{a_1} and f satisfy the conditions of Lemma 2.1, that is, $F \div E = F \div (E \setminus \{f\} \cup \{e_{a_1}\})$. This contradicts the choice of E. □

Theorem 5.3 and Lemma 5.4 are all that is needed to obtain a linear-time 4-approximation algorithm and an FPT algorithm with running time $\mathrm{O}\left(4^k n\right)$ for computing rooted MAFs, an observation made independently in [97]. To improve on this in our algorithms in Section 5.3, we exploit a useful observation from the proof of Theorem 5.3: if there exists an MAF $F \div E$ and leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_1}$ such that $a'_1 \sim_{F \div E} b'_1$, then $a'_2 \nsim_{F \div E} b'_2$, for all $a'_2 \in F_2^{a_2}$ and $b'_2 \in F_2^{B_2}$. This implies that, if we choose to cut e_{a_2} or e_{B_2} and keep both e_{a_1} and e_{B_1} in a branch of our FPT algorithm, then we need only decide which edge, e_{a_j} or e_{B_j}, to cut in each pair $\{e_{a_j}, e_{B_j}\}$, for $3 \leqslant j \leqslant r$, in subsequent steps of this branch. This allows us to follow each 4-way branch in the algorithm (where we decide whether to cut e_{a_1}, e_{B_1}, e_{a_2} or e_{B_2}) by a series of 2-way branches. We cannot use this idea when a sibling group consists of only two nodes. The following lemma addresses this case.

Lemma 5.5. *Let T_1 and T_2 be rooted X-trees, and let F_1 be a forest of T_1 and F_2 a forest of T_2. Suppose F_1 consists of a tree $\dot{T}_1$ and a set of components that exist in F_2.*

Let $\{a_1, a_2\}$ be a sibling group of $\dot{T}_1$ such that neither $F_2^{a_1}$ nor $F_2^{a_2}$ is a component of F_2 and, if a_1 and a_2 share a minimal LCA l, then a_1 is not a child of l. In particular, a_1 and a_2 are not siblings in F_2. Then

(i) $e(T_1, T_2, F_2 - \{e_x\}) = e(T_1, T_2, F_2) - 1$, *for some* $x \in \{a_1, a_2, B_1\}$.

(ii) $e\left(T_1, T_2, F_2 - \{e_{a_1}, e_{a_2}, e_{p_{a_1}}\}\right) \leqslant e(T_1, T_2, F_2) - 1$.

Proof. As in the proof of Theorem 5.3, (ii) follows immediately from (i), so it suffices to prove that there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an MAF of T_1 and T_2 (and, hence, of F_1 and F_2) and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\} \neq \varnothing$. Once again, we show that, if $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\} = \varnothing$, we can replace an edge $f \in E$ with an edge in $\{e_{a_1}, e_{a_2}, e_{B_1}\}$ without changing $F \div E$.

This follows from the same arguments as in the proof of Theorem 5.3 unless 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 $a_2' \sim_{F \div E} p_{a_2}$. In this case, since a_1 is not a child of l, we have $a_2 \notin F_2^{B_1}$. Thus, since $\{a_1, a_2\}$ is a sibling pair in F_1, F_1 contains the triple $a_1'a_2'|b_1'$. Since F_2 contains the triple $a_1'b_1'|a_2'$ and $a_1' \sim_{F \div E} b_1'$, this implies that $a_1' \nsim_{F \div E} a_2'$. Thus, we also have $a_2' \nsim_{F \div E} x$, for all $x \in F_2^l \setminus F_2^{a_2}$, as otherwise the components of $F \div E$ containing a_1', b_1' and a_2', x would overlap in F_1. We choose an arbitrary leaf $b_2' \in B_2$ and the first edge $f \in E$ on the path from p_{a_2} to b_2'. Lemma 2.1 implies that $F \div E = F \div (E \setminus \{f\} \cup \{e_{a_2}\})$. □

Next we examine the structure of a sibling group more closely as a basis for a refined analysis that leads to our final FPT algorithm with running time $\mathrm{O}\left(2.42^k n\right)$. First we require the notion of *pendant subtrees* that we will be able to cut in unison. Let $a_1, a_2, \ldots, a_r$ be the members of a sibling group $\{a_1, a_2, \ldots, a_m\}$ that share a minimal LCA l in F_2, and consider the path from a_i to l, for any $1 \leqslant i \leqslant r$. Let $x_1, x_2, \ldots, x_{s_i}$ be the nodes on this path, excluding a_i and l. For each x_j, let B_{ij} be the set of children of x_j, excluding the child that is an ancestor of a_i, and let $F_2^{B_{ij}}$ be the subforest of F_2 consisting of all subtrees F_2^b, $b \in B_{ij}$. Note that $B_{i1} = B_i$, and $F_2^{B_{i1}} = F_2^{B_i}$, if $s_i > 0$. Analogously to the definition of e_{B_i}, we use $e_{B_{ij}}$, for $1 \leqslant j \leqslant s_i$, to denote the edge introduced by expanding the nodes in B_{ij} in F_2 and $F_2 - \{e_{B_{ij}}\}$ to denote the forest obtained by expanding B_{ij} and cutting edge $e_{B_{ij}}$. Note that expanding B_{ij} turns the forest $F_2^{B_{ij}}$ into a single *pendant subtree* attached to x_j. We distinguish five cases for the structure of the subtree of F_2 induced by the paths between $a_1, a_2, \ldots, a_r$ and l:

Isolated Siblings: $a_1 \nsim_{F_2} a_i$, for all $i \neq 1$, and $a_2 \nsim_{F_2} a_j$, for all $j \neq 2$.

At Most One Pendant Subtree: $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2, $s_i = 1$, for all $1 \leqslant i < r$, and a_r is a child of l.

One Pendant Subtree: $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2 and $s_i = 1$, for all $1 \leqslant i \leqslant r$.

Multiple Pendant Subtrees, $m = 2$: a_1 and a_2 share a minimal LCA l in F_2 and $s_1 + s_2 \geqslant 2$.

Multiple Pendant Subtrees, $m > 2$: $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2 and $s_1 \geqslant 2$.

Since we assume $F_2^{a_i}$ is not a component of F_2, for all $1 \leqslant i \leqslant m$, and no two nodes a_i and a_j in the sibling group are siblings in F_2, we need only consider cases where at most one a_i is a child of some minimal LCA l, and we always label it a_r. Hence, $s_i > 0$, for all $i < r$ such that $a_i \sim_{F_2} l$. Thus, the five cases above cover every possible configuration of a sibling group where we must cut an edge of F_2.

The following four lemmas provide stronger statements than Theorem 5.3 about subsets of edges of a resolution F of F_2 that need to be cut in each of the last four cases above in order to make progress towards an AF of T_1 and T_2. All four lemmas consider a sibling group $\{a_1, a_2, \ldots, a_m\}$ of $\dot{T}_1$ as in Theorem 5.3 and assume $F_2^{a_i}$ is not a component of F_2, for all $1 \leqslant i \leqslant m$. Lemma 5.4 above covers the first of the five cases.

Lemma 5.6 (At Most One Pendant Subtree)**.** *If $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2, $s_i = 1$, for $1 \leqslant i < r$, and a_r is a child of l, then there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $\{e_{B_1}, e_{B_2}, \ldots, e_{B_{r-1}}\} \subseteq E$ or $\{e_{B_1}, e_{B_2}, \ldots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \ldots, e_{B_{r-1}}\} \subseteq E$ and $a'_i \sim_{F \div E} b'_i$, for some $1 \leqslant i \leqslant r-1$ and two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$.*

Proof. Let F be a binary resolution of F_2, and E an edge set of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2, and assume F and E are chosen so that

$|E \cap \{e_{a_1}, e_{a_2}, \ldots, e_{a_r}, e_{B_1}, e_{B_2}, \ldots, e_{B_{r-1}}\}|$ is maximized. Let $I := \{i \mid 1 \leqslant i \leqslant r - 1$ and $E \cap \{e_{a_i}, e_{B_i}\} \neq \emptyset\}$ and $I' := \{i \mid 1 \leqslant i \leqslant r-1$ and $e_{a_i} \in E\}$.

First observe that $|I| \geqslant r - 2$. Otherwise there would exist two indices $1 \leqslant i < j \leqslant r - 1$ such that $E \cap \{e_{a_i}, e_{B_i}, e_{a_j}, e_{B_j}\} = \emptyset$. By the choice of E and Lemma 2.1, this would imply that there exist leaves $a'_i \in F_2^{a_i}$, $b'_i \in F_2^{B_i}$, $a'_j \in F_2^{a_j}$, and $b'_j \in F_2^{B_j}$ such that $a'_i \sim_{F \div E} b'_i$ and $a'_j \sim_{F \div E} b'_j$. If $a'_i \sim_{F \div E} a'_j$, then $a'_i b'_i | a'_j$ would be a triple of $F \div E$ incompatible with F_1. If $a'_i \nsim_{F \div E} a'_j$, the components of $F \div E$ containing a'_i and a'_j would overlap in F_1. In both cases, $F \div E$ would not be an AF of F_1 and F_2, a contradiction.

Since $|I| \geqslant r - 2$ and $i \notin I$ implies that there are two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F \div E} b'_i$, the lemma follows if we can show that there exist a resolution F' of F_2 and an edge set E' of size $|E'| \leqslant |E|$ such that (i) $F' \div E'$ is an AF of F_1 and F_2, (ii) $\{e_{B_i} \mid i \in I\} \subseteq E'$, and (iii) for any $1 \leqslant i \leqslant r - 1$, there exist leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F' \div E'} b'_i$ if and only if there exist leaves $a''_i \in F_2^{a_i}$ and $b''_i \in F_2^{B_i}$ such that $a''_i \sim_{F \div E} b''_i$.

Let Y be the set of leaves in all trees $F_2^{a_i}$, $i \in I' \cup \{r\}$, and $F_2^{B_i}$, $i \in I'$, let $Z := X \setminus Y \cup \{a'_r\}$, for an arbitrary leaf $a'_r \in F_2^{a_r}$, let l' be the LCA in F of all nodes in Y, and let E'' be the set of edges in E that belong to the paths between l' and the nodes in $\{a_i \mid i \in I' \cup \{r\}\}$. Since $e_{a_i} \in E$, for all $i \in I'$, we have $|E''| \geqslant |I'|$. We construct F' from F_2 by resolving every node set B_i where $i \in I'$, resolving the set $\{a_r\} \cup \{p_{a_i} \mid i \in I'\}$, and resolving all remaining multifurcations so that $F|Y = F'|Y$ and $F|Z = F'|Z$. We define the set E' to be $E' := E \setminus E'' \cup \{e_{B_i} \mid i \in I'\}$ if $|E''| = |I'|$; otherwise $E' := E \setminus E'' \cup \{e_{B_i} \mid i \in I'\} \cup \{e_{l''}\}$, where l'' is the LCA in F' of all nodes in Y. It is easily verified that F' and E' satisfy properties (ii) and (iii) and that $|E'| \leqslant |E|$. Thus, it remains to prove that $F' \div E'$ is an AF of F_1 and F_2.

Any triple of $F' \div E'$ incompatible with F_1 has to involve exactly one leaf $a'_i \in F_2^{a_i}$, for some $i \in I'$, because any other triple exists either in $F \div E$ or in F_1 and, thus, is compatible with F_1. Thus, any triple $a'_i|xy$ or $a'_i x|y$ of $F' \div E'$ incompatible with F_1 must satisfy $x, y \notin (F')^{l''}$ because $e_{B_i} \in E'$, for all $i \in I'$. If $|E''| > |I'|$, no such triple exists because $e_{l''} \in E'$. If $|E''| = |I'|$, observe that $x, y \notin (F')^{l''}$ implies that $a'_r|xy$ or $a'_r x|y$ is also a triple of $F' \div E'$ incompatible with F_1. By the construction of F', this triple is also a triple of F, and since $E \setminus E' = \{e_{a_i} \mid i \in I'\}$ and $x, y \notin F_2^{a_i}$, for all

$i \in I'$, it is also a triple of $F \div E$ incompatible with F_1, a contradiction.

If two components of $F' \div E'$ overlap in F_1, let u, v, x, y be four leaves such that $u \sim_{F' \div E'} v \nsim_{F' \div E'} x \sim_{F' \div E'} y$ and the two paths P_{uv} and P_{xy} between u and v and between x and y, respectively, share an edge. Let Y' be the set of leaves in the subtrees $F_2^{a_i}$, $i \in I' \cup \{r\}$, and assume P_{uv} has no fewer endpoints in Y' than P_{xy}.

If both endpoints of P_{uv} are in Y', the two paths cannot overlap because $a_1, a_2, \dots, a_r$ are siblings in F_1 and our choices of E and E' ensure that all leaves in the same subtree $F_2^{a_i}$ belong to the same component of $F' \div E'$.

If both P_{uv} and P_{xy} have one leaf in Y', their corresponding paths in $F' \div E'$ include l''. Thus, $u \sim_{F' \div E'} x$.

If neither P_{uv} nor P_{xy} has an endpoint in Y', then $u \sim_{F \div E} v$ and $x \sim_{F \div E} y$. If P_{uv} has one endpoint in Y', say $u \in Y'$, and P_{xy} does not have an endpoint in Y', then $x \sim_{F \div E} y$ and, by the same arguments we used to show that no triple of $F' \div E'$ is incompatible with F_1, there exists a leaf $a'_r \in F_2^{a_r}$ such that $a'_r \sim_{F \div E} v$ and the path from a'_r to v in F_1 overlaps P_{xy}. Thus, in both cases we have two paths $P_{u'v}$, $u' \in \{u, a'_r\}$, and P_{xy} in F_1 such that $u' \sim_{F \div E} v$ and $x \sim_{F \div E} y$ and the two paths overlap. Since no two components of $F \div E$ overlap in F_1, these two paths belong to the same component of $F \div E$ and w.l.o.g. form the quartet $u'x|vy$, while $u' \sim_{F' \div E'} v \nsim_{F' \div E'} x \sim_{F' \div E'} y$. Since $E' \setminus E \subseteq \{e_{B_i} \mid i \in I'\} \cup \{e_{l''}\}$, we either have $x, y \in F_2^{B_i}$ and $u', v \notin F_2^{a_i} \cup F_2^{B_i}$, for some $i \in I'$, or $u', v \in (F')^{l''}$, $u', v \notin F_2^{B_i}$, for all $i \in I'$, and $x, y \notin (F')^{l''}$. In the former case, these four leaves form the quartet $u'v|xy$ in $F \div E$, a contradiction. In the latter case, we have $u', v \in Y'$, and we already argued that this case is impossible. $\square$

Lemma 5.7 (One Pendant Subtree)**.** *If $a_1, a_2, \dots, a_r$ share a minimal LCA l in F_2, $m > 2$, and $s_i = 1$, for all $1 \leqslant i \leqslant r$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $\{e_{a_1}, e_{a_2}, \dots, e_{a_r}\} \subseteq E$, $\{e_{B_1}, e_{B_2}, \dots, e_{B_r}\} \subseteq E$ or there exists an index $1 \leqslant i \leqslant r$ and two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F \div E} b'_i$ and either $\{e_{a_1}, e_{a_2}, \dots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \dots, e_{a_r}\} \subseteq E$ or $\{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_r}\} \subseteq E$.*

Proof. Let F be a resolution of F_2, and E an edge set of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2, and assume F and E are chosen so that $|E \cap$

$\{e_{a_1}, e_{a_2}, \ldots, e_{a_r}, e_{B_1}, e_{B_2}, \ldots, e_{B_r}\}|$ is maximized.

If there exists an index $1 \leqslant j \leqslant r$ such that $e_{B_j} \in E$, then assume w.l.o.g. that $j = r$. The forests F_1 and $F_2' := F_2 \div \{e_{B_r}\}$ satisfy the conditions of Lemma 5.6. Hence, there exist a resolution F' of F_2' and an edge set E' of size $e\,(T_1, T_2, F_2') = e\,(T_1, T_2, F_2) - 1$ such that $F' \div E'$ is an AF of F_1 and F_2' and, thus, of F_1 and F_2 and either $\{e_{B_1}, e_{B_2}, \ldots, e_{B_{r-1}}\} \subseteq E'$ or $a_i' \sim_{F' \div E'} b_i'$ and $\{e_{B_1}, e_{B_2}, \ldots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \ldots, e_{B_{r-1}}\} \subseteq E'$, for some index $1 \leqslant i \leqslant r-1$ and leaves $a_i' \in F_2^{a_i}$ and $b_i' \in F_2^{B_i}$. Thus, the resolution F'' of F_2 such that $F'' \div \{e_{B_r}\} = F'$ and the edge set $E' \cup \{e_{B_1}\}$ satisfy the lemma.

If $E \cap \{e_{B_1}, e_{B_2}, \ldots, e_{B_r}\} = \varnothing$, then by the same arguments as in Lemma 5.6, we can have at most one index $1 \leqslant i \leqslant r$ such that $a_i' \sim_{F \div E} b_i'$, for two leaves $a_i' \in F_2^{a_i}$ and $b_i' \in F_2^{B_i}$. If such an index i exists, then $\{e_{a_1}, e_{a_2}, \ldots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \ldots, e_{a_r}\} \subseteq E$. If no such index exists, then $\{e_{a_1}, e_{a_2}, \ldots, e_{a_r}\} \subseteq E$. $\square$

Lemma 5.8 (Multiple Pendant Subtrees, $m = 2$). *If $\{a_1, a_2\}$ is a sibling group such that a_1 and a_2 share a minimal LCA l in F_2 and $s_1 + s_2 \geqslant 2$, then there exist a resolution F of F_2 and an edge set E of size $e\,(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$ or $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \subseteq E$.*

Proof. We prove this by induction on $s = s_1 + s_2$. The base case is $s = 1$[3] and the claim follows from Lemma 5.5.

Having established the base case, we can assume $s > 1$ and the lemma holds for all $1 \leqslant s' < s$. By Theorem 5.3, there exist a resolution F of F_2 and an edge set E of size $e\,(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \varnothing$. Assume F and E are chosen so that $|E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}|$ is maximized. If $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$, the lemma holds, so assume the contrary. If $s_2 = 0$ and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} = \{e_{B_2}\}$, the choice of F and E and Lemma 2.1 imply that there exist leaves $a_1' \in F_2^{a_1}$, $b_1' \in F_2^{B_1}$, $a_2' \in F_2^{a_2}$, and $x \notin F_2^{a_2}$ such that $a_1' \sim_{F_2 \div \{e_{B_2}\}} b_1'$ and $a_2' \sim_{F_2 \div \{e_{B_2}\}} x$. Thus, a_1 and a_2 satisfy the conditions of Lemma 5.4 after cutting edge e_{B_2}, which implies that we can choose F and E so that $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$ in addition to $e_{B_2} \in E$, a contradiction. Now assume $s_2 \neq 0$ or $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq$

[3]We excluded this case from the statement of the lemma, in order to keep the cases covered by the different lemmas disjoint, but the lemma also holds for $s = 1$. A similar comment applies to Lemma 5.9.

$\{e_{B_2}\}$. In this case, $E \cap \{e_{B_{11}}, e_{B_{21}}\} \neq \varnothing$. Assume w.l.o.g. that $e_{B_{11}} \in E$. Then the inductive hypothesis shows that there exist a resolution F' of F_2 and an edge set E' of size $e(T_1, T_2, F_2)$ such that $F' \div E'$ is an AF of F_1 and $F_2 \div \{e_{B_{11}}\}$ (and, hence, of F_1 and F_2) such that either $E' \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$ or $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \subseteq E'$. Thus, the lemma also holds in this case. $\square$

The proofs of the following two lemmas are similar to that of Lemma 9.

Lemma 5.9 (Multiple Pendant Subtrees, $m > 2$ and $r > 2$)**.** *If $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2, $m > 2$, $r > 2$, and $s_1 \geqslant 2$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$, $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \subseteq E$.*

Proof. As in the proof of Lemma 5.8, we prove this using induction on $s = s_1 + s_2$. The base case is $s = 2$ and, hence, $s_1 = s_2 = 1$ because $r > 2$ implies that $s_1 > 0$ and $s_2 > 0$. In this case, Theorem 5.3 proves the lemma. So assume $s > 2$ and the claim holds for all $1 \leqslant s' < s$. By Theorem 5.3, there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_{11}}, e_{B_{21}}\} \neq \varnothing$. Using an inductive argument as in the proof of Lemma 5.8, it follows that there exist a resolution F' of F_2 and an edge set E' satisfying the lemma. $\square$

Lemma 5.10 (Multiple Pendant Subtrees, $m > 2$ and $r = 2$)**.** *If $a_1, a_2, \ldots, a_r$ share a minimal LCA l in F_2, $m > 2$, $r = 2$ and $s_1 \geqslant 2$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \varnothing$, $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}, e_{B'_2}\} \subseteq E$, where B'_2 is the set of siblings of a_i after cutting $e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}$.*

Proof. First consider the case when $s_2 = 0$, which is possible because $r = 2$. Then $B'_2 = B_2$ and, by Theorem 5.3, there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_{11}}, e_{B_2}\} \neq \varnothing$. If $E \cap \{e_{a_1}, e_{a_2}, e_{B_2}\} \neq \varnothing$, the lemma holds. Otherwise $e_{B_{11}} \in E$ and an inductive argument similar to the one in the proof of Lemma 5.8 proves the lemma.

If $s_2 > 0$, we observe that the proof of Lemma 5.9 did not use the assumption that $r > 2$ but only that it implies $s_2 > 0$. Hence, this proof shows that there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}, e_{B_{12}}, \ldots, e_{B_{1s_1}}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \subseteq E$. Among all such resolutions and edge sets, choose F and E so that $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$ or $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \subseteq E$ if possible. If we can find such a pair (F, E), the lemma holds. Otherwise $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \subseteq E$. Let F' be the forest obtained from F_2 by resolving $B_{21}, B_{22}, \ldots, B_{2s_2}$ and cutting edges $e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}$. In F', we have $r = 2$ and $s_2 = 0$. Hence, by the argument in the previous paragraph, there exist a resolution F'' of F' and an edge set E'' of size $e(T_1, T_2, F')$ such that $F'' \div E''$ is an AF of T_1 and T_2 and $E'' \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \subseteq E''$ or $e_{B'_2} \in E''$. In the first two cases, we obtain a contradiction to the choice of F and E. In the latter case, the set $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \cup E''$ has size $s_2 + e(T_1, T_2, F'') = e(T_1, T_2, F_2)$, $F_2 \div (\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \cup E'')$ is an AF of T_1 and T_2, and $\{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}, e_{B'_2}\} \subseteq \{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\} \cup E''$. Thus, the lemma holds in this case as well. □

5.3 MAF Algorithms

In this section, we present an FPT algorithm for computing MAFs of multifurcating rooted trees. This algorithm also forms the basis for a 3-approximation algorithm with running time $\mathrm{O}(n \log n)$, which is presented in Section 5.4.

As is customary for FPT algorithms, we focus on the decision version of the problem: "Given two rooted X-trees T_1 and T_2 and a parameter k, is $d_{SPR}(T_1, T_2) \leqslant k$?" To compute the distance between two trees, we start with $k = 0$ 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, as the running time depends exponentially on k.

Our FPT algorithm is recursive. Each invocation $\textsc{Maf}(F_1, F_2, k, a_0)$ takes two (partially resolved) forests F_1 and F_2 of T_1 and T_2, a parameter k, and (optionally) a node a_0 that exists in F_1 and F_2 as inputs. F_1 is the union of a tree $\dot{T}_1$ and a forest F_0 disjoint from $\dot{T}_1$, while F_2 is the union of the same forest F_0 and another forest $\dot{F}_2$ with the same label set as $\dot{T}_1$. The output of the invocation $\textsc{Maf}(F_1, F_2, k, a_0)$ satisfies two conditions: (i) If $e(T_1, T_2, F_2) > k$, the output is "no". (ii) If $e(T_1, T_2, F_2) \leqslant k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0 is not a root of F and $a_0 \nsim_F a_i$, for every sibling a_i of a_0 in F_1, the output is "yes". Since the top-level invocation is $\textsc{Maf}\,(T_1, T_2, k, \text{nil})$, these two conditions ensure that this invocation decides whether $e\,(T_1, T_2, T_2) \leqslant k$.

The representation of the input to each recursive call includes two sets of labelled nodes: R_d (roots-done) contains the roots of F_0, R_t (roots-todo) contains the 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 $F_0 = \varnothing$; R_d is empty and R_t contains all leaves of T_1; $a_0 = \text{nil}$.

$\textsc{Maf}\,(F_1, F_2, k, a_0)$ uses the results from Section 5.2 to identify a small collection $\{E_1, E_2, \ldots, E_q\}$ of subsets of edges of $\dot{F}_2$ such that $e\,(T_1, T_2, F_2) \leqslant k$ only if $e\,(T_1, T_2, F_2 \div E_i) \leqslant k - |E_i|$, for at least one $1 \leqslant i \leqslant q$. It calls $\textsc{Maf}\,(F_1, F_2 \div E_i, k - |E_i|, a'_i)$ recursively, for each subset E_i and an appropriate parameter a'_i, and returns "yes" if and only if one of these recursive calls does.

A naïve use of the structural results from Section 5.2 would explore many overlapping edge subsets. For example, one branch of the algorithm may cut an edge e_{a_i} and then an edge e_{a_j}, while a sibling branch may cut e_{a_j} and then e_{a_i}. As we hinted at in Section 5.2, if we cut edge e_{a_i} or its sibling edge e_{B_i} in two sibling invocations, then there is no need to consider cutting either of these two edges in their sibling invocations or their descendants. Using the results of Lemmas 5.6–5.10, we obtain more generally: if we cut e_{a_i} or its set of progressive siblings $\{e_{B_{i1}}, e_{B_{i2}}, \ldots, e_{B_{is_i}}\}$ in two sibling invocations, then we need not consider these edge sets in their sibling invocations or their descendants. Thus, we set $a_0 = a_i$ in these sibling invocations and thereby instruct the algorithm to ignore these edges as candidates for cutting. An invocation $\textsc{Maf}\,(F_1, F_2, k, a_0)$ with $a_0 \neq \text{nil}$ (Step 7 below) makes only two recursive calls when it would make significantly more recursive calls if $a_0 = \text{nil}$ (Step 8 below). This is not a trivial change, as it is required to obtain the running time claimed in Theorem 5.12. The steps of our procedure are as follows.

1. (Failure) If $k < 0$, then $e\,(T_1, T_2, F_2) \geqslant 0 > k$. Return "no" in this case.

2. (Success) If $|R_t| = 1$, then $F_1 = F_2$. Hence, F_2 is an AF of T_1 and T_2, that is, $e\,(T_1, T_2, F_2) = 0 \leqslant k$. Return "yes" in this case.

3. (Prune maximal agreeing subtrees) If there is no node $r \in R_t$ that is a root in F_2, proceed to Step 4. Otherwise choose such a node $r \in R_t$; remove it from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F_0; and cut the edge e_r in F_1. If r's parent p_r in F_1 now has only one child, contract p_r. If $a_0 \neq$ nil and p_r's only child before the contraction was a_0, set $a_0 =$ nil. Note that these changes affect only F_1. Thus, $e(T_1, T_2, F_2)$ remains unchanged. Return to Step 2.

4. Choose a sibling group $\{a_1, a_2, \ldots, a_m\}$ in $\dot{T}_1$ such that $a_1, a_2, \ldots, a_m \in R_t$. If two or more members of the sibling group chosen in this invocation's parent invocation remain in $\dot{T}_1$, choose that sibling group.

5. (Grow agreeing subtrees) While there exist indices $1 \leqslant i < j \leqslant m$ such that a_i and a_j are siblings in $\dot{F}_2$, do the following: 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. If $a_i \nsim_{F_2} a_j$, for all $1 \leqslant i < j \leqslant m$, proceed to Step 7. Otherwise there exists a node l that is a minimal LCA of a group of nodes in the current sibling group. If the most recent minimal LCA chosen in an ancestor invocation is a minimal LCA of a subset of nodes in the current sibling group, choose l to be this node; otherwise choose l arbitrarily. Now order the nodes in the sibling group $\{a_1, a_2, \ldots, a_m\}$ so that, for some $r \geqslant 2$, $a_1, a_2, \ldots, a_r$ are descendants of l, while, for all $1 \leqslant i \leqslant r < j \leqslant m$, either the LCA of a_i and a_j is a proper ancestor of l or $a_i \nsim_{F_2} a_j$. Order $a_1, a_2, \ldots, a_r$ so that $s_1 \geqslant s_2 \geqslant \cdots \geqslant s_r$. (Recall that s_i is the number of nodes on the path from a_i to l, excluding a_i and l.) The order of $a_{r+1}, a_{r+2}, \ldots, a_m$ is arbitrary.

7. (Two-way branching) If $a_0 =$ nil, proceed to Step 8. Otherwise distinguish four cases, where $x = 1$ if $a_1 \neq a_0$, and $x = 2$ otherwise (see Figure 5.5).

   7.1. If $a_1 \nsim_{F_2} a_i$, for all $i \neq 1$, and $a_2 \nsim_{F_2} a_j$, for all $j \neq 2$, call Maf $(F_1, F_2 \div \{e_{a_x}\}, k - 1, a_0)$.

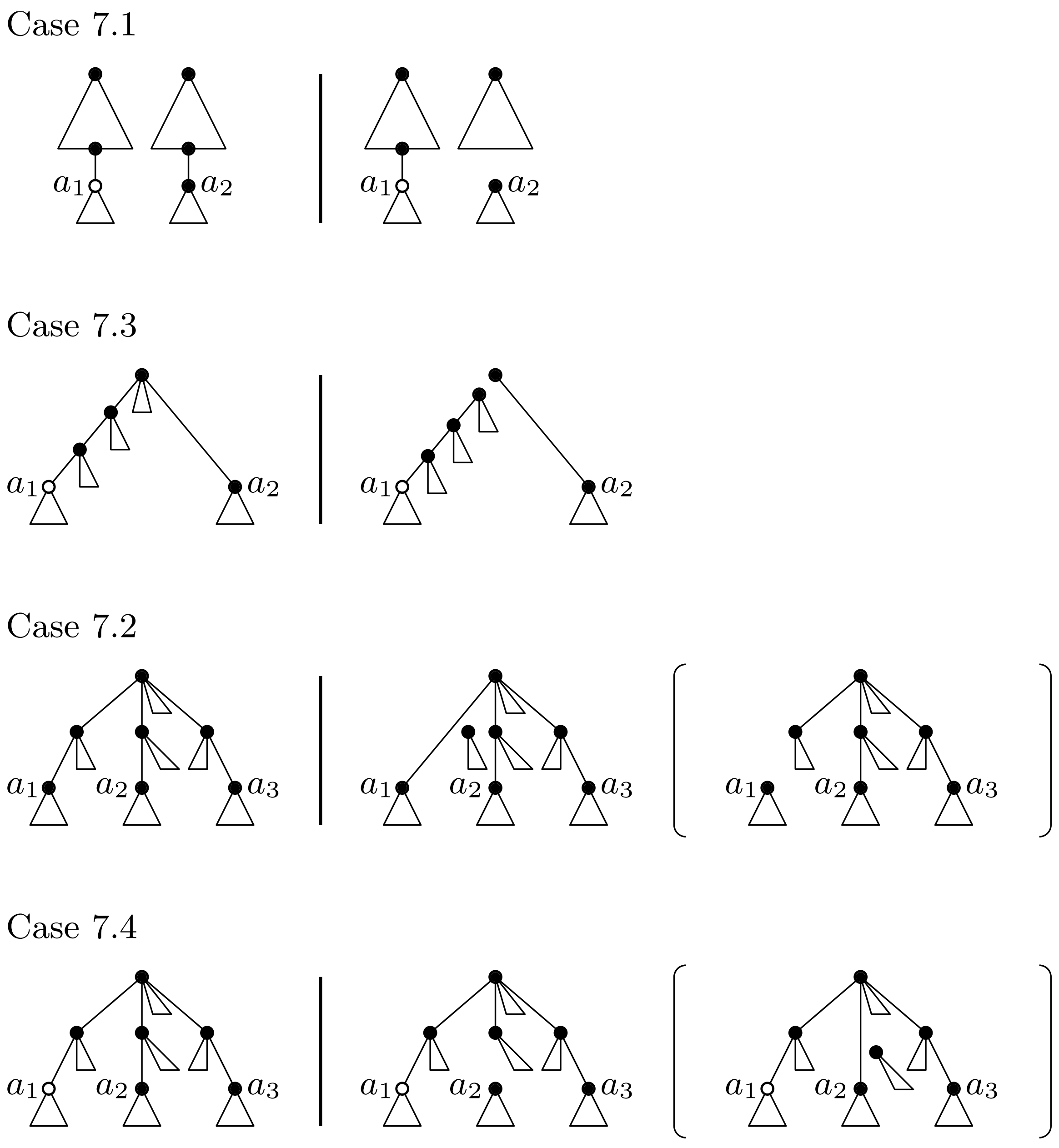


Figure 5.5: The different cases of Step 7. Only the subtree of $\dot{F}_2$ rooted in l is shown. The left side shows a possible input for each case, the right side visualizes the cuts made in each recursive call. The node a_0 is shown as a hollow circle (if it is a descendant of l).

7.2. If $a_1, a_2, \ldots, a_r$ share the minimal LCA l in F_2 and a_0 is not a descendant of l, call $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\}, k - s_1, a_0\big)$. If $s_1 > 1$ or $s_r > 0$, also call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_0)$.

7.3. If $r = 2$, $s_x = 0$, a_0 is a descendant of l, and either l is a root of F_2 or its parent has a member a_i of the current sibling group as a child, call $\textsc{Maf}\,(F_1, F_2 - \{e_{B_x}\}, k-1, a_0)$.

7.4. If $a_1, a_2, \ldots, a_r$ share the minimal LCA l in F_2, a_0 is a descendant of l, and Case 7.3 does not apply, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$. If $m > 2$ and $r > 2$, make another recursive call $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}\}, k - s_x, a_0\big)$. If $m > 2$ but $r = 2$, the second recursive call is $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}, e_{B'_x}\}, k - (s_x + 1), a_0\big)$, where B'_x is the set of siblings of x after cutting $e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}$.

Return "yes" if one of the recursive calls does; otherwise return "no".

8. (Unconstrained branching) Distinguish seven cases and choose the first case that applies (see Figure 5.6):

8.1. If $a_1 \nsim_{F_2} a_i$, for all $i \neq 1$, and $a_2 \nsim_{F_2} a_j$, for all $j \neq 2$, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, \mathrm{nil})$ and $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, \mathrm{nil})$.

8.2. If $s_i = 1$, for $1 \leqslant i < r$, and a_r is a child of l, call $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \ldots, e_{B_{r-1}}\}, k - (r-1), \mathrm{nil}\big)$ and $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \ldots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \ldots, e_{B_{r-1}}\}, k - (r-2), a_i\big)$, for all $1 \leqslant i \leqslant r-1$.

8.3. If $m = 2$, and $s_1 + s_2 \geqslant 2$, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, \mathrm{nil})$, $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, \mathrm{nil})$, and $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \ldots, e_{B_{2s_2}}\}, k - (s_1 + s_2), \mathrm{nil}\big)$.

8.4. If $m > 2$, $r = 2$, and $s_1 = s_2 = 1$, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}, e_{a_2}\}, k-2, \mathrm{nil})$, $\textsc{Maf}\,(F_1, F_2 \div \{e_{B_1}, e_{B_2}\}, k-2, \mathrm{nil})$, $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_1}, e_{B'_1}\}, k-2, a_2\big)$, and $\textsc{Maf}\big(F_1, F_2 \div \{e_{B_2}, e_{B'_2}\}, k-2, a_1\big)$, where B'_i is the set of siblings of a_i after cutting edge B_i. If l is a root, l's parent p_l has at least one child that is neither l nor a member a_j of the current sibling group or l's grandparent has at least one child that is neither p_l nor a member a_h of the current sibling group, make two additional calls $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ and $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$.

8.5. If $m > 2$, $r > 2$, and $s_i = 1$, for all $1 \leqslant i \leqslant r$, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}, e_{a_2}, \dots, e_{a_r}\}, k-r, \text{nil})$, $\textsc{Maf}\,(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_r}\}, k-r, \text{nil})$, $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{a_1}, e_{a_2}, \dots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \dots, e_{a_r}\}, k-(r-1), a_i\big)$, for all $1 \leqslant i \leqslant r$, and $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_r}\}, k-(r-1), a_i\big)$, for all $1 \leqslant i \leqslant r$.

8.6. If $m > 2$, $r = 2$, $s_1 \geqslant 2$, $s_2 = 0$ and either l is a root of F_2 or its parent has a member a_i of the current sibling group as a child, call $\textsc{Maf}\,(F_1, F_2 - \{e_{a_1}\}, k-1, \text{nil})$, $\textsc{Maf}\,\big(F_1, F_2 - \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k-s_1, \text{nil}\big)$, and $\textsc{Maf}\,(F_1, F_2 - \{e_{B_2}\}, k-1, \text{nil})$.

8.7. If $m > 2$, $s_1 \geqslant 2$, and Case 8.6 does not apply, call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, \text{nil})$, $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$, and $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k-s_1, \text{nil}\big)$. If $r > 2$, call $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\}, k-s_2, a_1\big)$. If $r = 2$, call $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}, e_{B'_2}\}, k-(s_2+1), a_1\big)$, where B'_2 is defined as in Lemma 5.10.

Return "yes" if one of the recursive calls does; otherwise return "no".

Lemma 5.11. *Each invocation* $\textsc{Maf}\,(F_1, F_2, k', a_o)$*, excluding recursive calls it makes, takes* $\mathrm{O}\,(n)$ *time*

Proof. We represent each forest as a collection of nodes, each of which points to its parent, to its leftmost child, and to its left and right siblings. This allows us to cut an edge in constant time, given the parent and child connected by this edge. Every labelled node (i.e., every node in R_t or R_d) stores a pointer to its counterpart in the other forest. For $\dot{T}_1$, we maintain a list of sibling groups of labelled nodes. For each such group, the list stores a pointer to the parent of the sibling group, which allows us to access the members of the sibling group by traversing the list of the parent's children. To detect the creation of such a sibling group, and add it to the list, each internal node of $\dot{T}_1$ stores the number of its unlabelled children. When labelling a non-root node, we decrease its parent's unlabelled children count by one. If this count is now 0, the children of this parent node form a new sibling group, and we add a pointer to the parent to the list of sibling groups. For $\dot{F}_2$, we maintain a list $R_d' \subseteq R_t$ of labelled nodes that are roots of $\dot{F}_2$. This list is used to move these roots from R_t to R_d.

Steps 1–4 are implemented similarly to the algorithm for binary trees [100]. Step 1 clearly takes constant time. In Step 2, we can test in constant time whether $|R_t| \leqslant 1$ by inspecting at most two nodes in the first two sibling groups. Step 3 takes constant time to test whether the root list R_d' is empty and, if it is not, cut the appropriate edge in $\dot{T}_1$ and update a constant number of lists and pointers. Step 4 takes contant time using the list of sibling groups. We always choose the next sibling group from the beginning of this list and append new sibling groups to the end. This automatically gives preference to the most recently chosen sibling group as required in Step 4.

Step 5 requires some care to implement efficiently. We iterate over the members $a_1, a_2, \ldots, a_m$ of the current sibling group and mark their parents in $\dot{F}_2$. Initially, all nodes in $\dot{F}_2$ are unmarked. When inspecting a node a_i whose parent p_{a_i} in $\dot{F}_2$ is unmarked, we mark p_{a_i} with a_i. If p_{a_i} is already marked with a node a_j $(j < i)$, then a_i and a_j are siblings in $\dot{T}_1$ and $\dot{F}_2$. We resolve them in constant time and mark p_{a_i} (which is now the grandparent of a_i) with the new parent (a_i, a_j) of a_i and a_j. Since we spend constant time per member a_i of the sibling group, this procedure takes $\mathrm{O}(m)$ time. Once it finishes, the remaining members of the sibling group are not siblings in $\dot{F}_2$. Performing a contraction if the remaining sibling group has only one member takes constant time.

In Step 6, we perform a linear-time traversal of $\dot{F}_2$ to label every node x with the number r_x of members of the current sibling group among its descendants. Then, if the previously chosen minimal LCA l still exists in $\dot{F}_2$ and has at least two descendants in the current sibling group, we keep this choice of l. Otherwise a node x is a minimal LCA of a subset of the current sibling group if and only if $r_x \geqslant 2$ and $r_y \leqslant 1$, for each child y of x. If there is no such node x, we proceed to Step 7 without choosing l because $a_i \nsim_{F_2} a_j$, for all $1 \leqslant i < j \leqslant m$. Otherwise we pick any node x satisfying this condition as the new minimal LCA l. No matter whether l is the previously chosen minimal LCA or a new node, we set $r = r_l$ and traverse the paths from l to its descendant members of the sibling group, $a_1, a_2, \ldots, a_r$. We do this by visiting all descendants y of l such that $r_y = 1$. For all $1 \leqslant i \leqslant r$, the length of the path from l to a_i, excluding l and a_i, is s_i. We sort $a_1, a_2, \dots, a_r$ by their path lengths $s_1, s_2, \dots, s_r$ using Counting Sort [34]. Since $s_i \leqslant n$, for all $1 \leqslant i \leqslant r$, this takes linear time.

To distinguish between Steps 7 and 8, it suffices to examine a_0. We distinguish between the cases in Steps 7 and 8 using the values of r, m, and $s_1, s_2, \dots, s_r$ and, in Step 7, by testing whether a_0 is among the descendants of l. In each case, we can easily copy the forests, cut the appropriate edges, and update our lists and pointers in linear time for each of the recursive calls.

To summarize: Each execution of Steps 1–4 takes constant time. Step 1 is executed once per invocation. Steps 2–4 are executed at most a linear number of times per invocation because each execution, except the first one, is the result of finding a root of $\dot{F}_2$ in Step 3 or resolving sibling pairs in Step 5, both of which can happen only $\mathrm{O}(n)$ times. Each execution of Step 5 takes $\mathrm{O}(m)$ time. In a given invocation, Step 5 is executed at most once per sibling group (because we either proceed to Step 6 or return to Step 2 after completely resolving the sibling group). Thus, since the total size of all sibling groups is bounded by $|\dot{T}_1|$, the total cost of all executions of Step 5 per invocation is $\mathrm{O}(n)$. Steps 6–8 are executed at most once per invocation and take linear time. Thus, each invocation of the algorithm takes linear time. $\square$

Theorem 5.12. *Given two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}\left((1+\sqrt{2})^k n\right) = \mathrm{O}\left(2.42^k n\right)$ time to decide whether $e(T_1, T_2, T_2) \leqslant k$.*

Proof. We use the algorithm in this section, invoking it as $\textsc{Maf}(T_1, T_2, k, \text{nil})$. We leave its correctness proof to a separate lemma (Lemma 5.13 below) and focus on bounding its running time here. As we showed in Lemma 5.11, each invocation $\textsc{Maf}(F_1, F_2, k', a_0)$ takes $\mathrm{O}(n)$ time. Thus, it suffices to bound the number of invocations by $\mathrm{O}\left((1+\sqrt{2})^k\right)$. Let $I(k,t)$ be the number of invocations that are descendants of an invocation $\textsc{Maf}(F_1, F_2, k, a_0)$ in the recursion tree, where $t = 1$ if the invocation executes Step 7 but not Step 8; otherwise $t = 0$. We develop a recurrence relation for $I(k,t)$ and use it to show that $I(k,t) \leqslant (1+\sqrt{2})^{2+\max(0,k-t+3)} + 2(t-1)$, which proves our claim.[4]

An invocation with $t = 0$ by definition either executes neither Step 7 nor Step 8, or it executes Step 8. By considering the different cases of Step 8, we obtain the

[4]This is a fairly loose bound on $I(k,t)$, but it is easy to manipulate.

following recurrence for the case when $t = 0$:

$$I(k,0) \leqslant \begin{cases} 1 & \text{no recursion} \\ 1 + 2I(k-1,0) & \text{Case 8.1} \\ 1 + I(k-1,0) + I(k,1) & \text{Case 8.2} \\ 1 + 2I(k-1,0) + I(k-2,0) & \text{Cases 8.3, 8.6} \\ 1 + 2I(k-2,0) + 2I(k-1,1) & \\ \quad + 2I(k-2,1) & \text{Case 8.4} \\ 1 + 2I(k-3,0) + 3I(k-2,0) & \\ \quad + 3I(k-2,1) & \text{Case 8.5} \\ 1 + I(k-1,0) + I(k-2,0) & \\ \quad + 2I(k-1,1) & \text{Case 8.7} \end{cases}$$

The values of the first arguments of all $I(\cdot,\cdot)$ terms are easily verified for Cases 8.1, 8.3, 8.4, and 8.6. For Case 8.2, we observe that $r \geqslant 2$ and the worst case arises when $r = 2$, giving the claimed recurrence. For Case 8.7, we observe again that $r \geqslant 2$. If $r > 2$, then $s_2 > 0$, giving the recurrence for this case. If $r = 2$, then s_2 may be 0, but the fourth recursive call cuts $s_2 + 1$ edges, thereby giving the same recurrence as when $r > 2$. For Case 8.5, finally, we have $r \geqslant 3$ and, once again, the minimum value, $r = 3$, is the worst case, which gives the recurrence.

Next we argue about the correctness of all second arguments that are 1 in these recurrences. Each such term $I(\cdot, 1)$ corresponds to a recursive call with $a_0 \neq$ nil. Thus, in order to justify setting $t = 1$, we need to show that each such invocation executes Step 7 but not Step 8, which follows if in this child invocation, the current invocation's sibling group exists and at least one additional edge cut in F_2 is required to makes this sibling group agree between F_1 and F_2—we say that the sibling group agrees between F_1 and F_2 if F_2 does not contain a triple incompatible with F_1 and involving descendants of at least two members of the sibling group, and there are no two paths between leaves in F_2 that (i) belong to different components of F_2, (ii) overlap in F_1, and (iii) have at least one endpoint each that is a descendant of a member of the current sibling group. The only cases that make recursive calls with

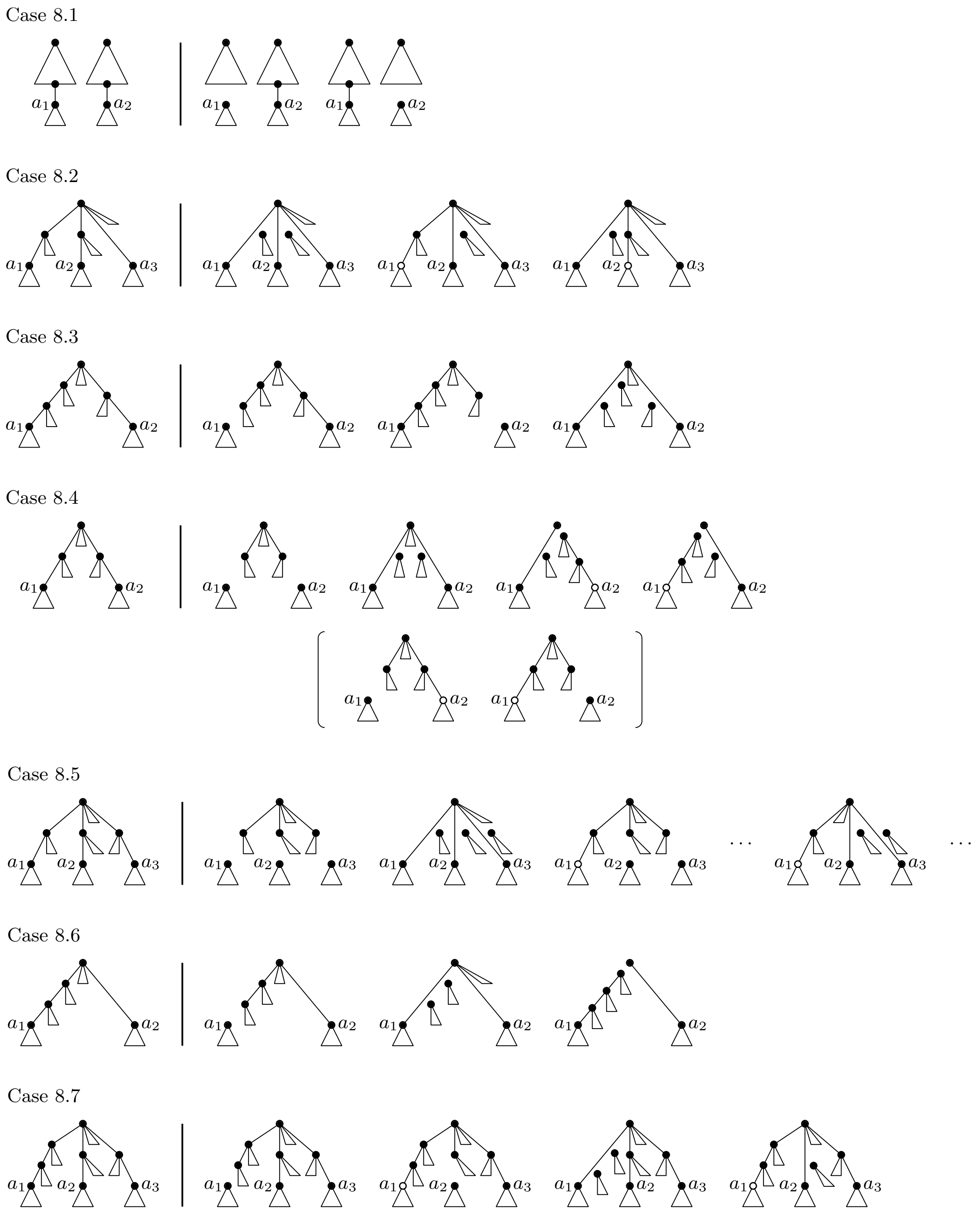


Figure 5.6: The different cases of Step 8. Only the subtree of $\dot{F}_2$ rooted in l is shown. The left side shows a possible input for each case, the right side visualizes the cuts made in each recursive call. Whenever $a_0 \neq \text{nil}$ in a recursive call, it is shown as a hollow circle. The last two calls in Case 8.4 may or may not be made.

$a_0 \neq$ nil are Cases 8.2, 8.4, 8.5 and 8.7. We consider each case in turn.

In Case 8.2, consider a recursive call that cuts edges $e_{B_1}, e_{B_2}, \ldots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \ldots, e_{B_{r-1}}$, for some $1 \leqslant i \leqslant r-1$. Assume w.l.o.g. that $i = 1$. After cutting edges $e_{B_2}, e_{B_3}, \ldots, e_{B_{r-1}}$, there still exist leaves $a_1' \in F_2^{a_1}$, $b_1' \in F_2^{B_1}$, and $a_r' \in F_2^{a_r}$ such that F_1 contains the triple $a_1'a_r'|b_1'$ while F_2 contains the triple $a_1'b_1'|a_r'$. Thus, the sibling group does not agree between F_1 and F_2 yet.

In Case 8.4, if we make only four recursive calls, we can ignore whether setting $a_0 = a_1$ or $a_0 = a_2$ in the third and fourth recursive calls translates into $t = 1$ for these recursive calls because even the recurrence $I(k, 0) = 1 + 4I(k-2, 0)$ is bounded by the recurrence $I(k, 0) = 1 + 2I(k-2, 0) + 2I(k-2, 1) + 2I(k-1, 1)$ for the case when we make six recursive calls. If we make six recursive calls, there exists a member a_3 of the current sibling group that is not a descendant of l. Thus, the conditions for making the fifth and sixth recursive calls imply that there exist leaves $a_3' \in F_2^{a_3}$ and $b_3' \notin F_2^{a_i}$, for all $1 \leqslant i \leqslant m$, such that $a_3' \sim_{F_2} b_3'$ and the path from a_3' to b_3' in F_2 is disjoint from F_2^l. After cutting e_{B_1} and $e_{B_1'}$, there exist leaves $a_2' \in F_2^{a_2}$ and $b_2' \in F_2^{B_2}$ such that $a_2' \sim_{F_2 \div \{e_{B_1}, e_{B_1'}\}} b_2'$. Thus, we have two paths in different connected components (between a_2' and b_2' and between a_3' and b_3') that overlap in F_1, and the sibling group does not agree between F_1 and F_2 yet. After cutting e_{a_1}, we obtain overlapping paths as above if $a_2 \nsim_{F_2} a_3$; otherwise $a_2'b_2'|a_3'$ is a triple of F_2 incompatible with F_1. Thus, once again the sibling group does not agree between F_1 and F_2. Similar arguments show that setting $t = 1$ is correct when cutting edge e_{a_2} or edges e_{B_2} and $e_{B_2'}$.

In Case 8.5, we claim that it is correct to set $t = 1$ for each recursive call that cuts edges $e_{B_1}, e_{B_2}, \ldots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \ldots, e_{B_r}$, for some $1 \leqslant i \leqslant r$. Assume w.l.o.g. that $i = 1$. After cutting edges $e_{B_2}, e_{B_3}, \ldots, e_{B_r}$, there exist leaves $a_1' \in F_2^{a_1}$, $b_1' \in F_2^{B_1}$, and $a_2' \in F_2^{a_2}$ such that $a_1'a_2'|b_1'$ is a triple of F_1 and $a_1'b_1'|a_2'$ is a triple of F_2. Thus, the sibling group does not agree between F_1 and F_2 yet.

In Case 8.7, observe that l is not a root and its parent does not have a member of the current sibling group as a child. Thus, since $m > 2$, there exists an index j and two leaves $a_j' \in F_2^{a_j}$ and $b_j' \notin F_2^{a_h}$, for all $1 \leqslant h \leqslant m$, such that $a_j' \sim_{F_2} b_j'$ and the path from a_j' to b_j' in F_2 is disjoint from the path between any two leaves in $F_2^{a_1}$ and $F_2^{B_{11}}$. After cutting e_{a_2}, there exist leaves $a_1' \in F_2^{a_1}$ and $b_1' \in F_2^{B_{11}}$ such that $a_1' \sim_{F_2 \div \{e_{a_2}\}} b_1'$. If $a_1' \sim_{F_2} a_j'$, we thus have a triple $a_1'b_1'|a_j'$ of F_2 incompatible with F_1. If $a_1' \nsim_{F_2} a_j'$, the path between a'_1 and b'_1 overlaps the path between a'_j and b'_j in F_1. In either case, the current sibling group does not agree between F_1 and F_2 yet. This concludes the correctness proof of the recurrence for $I(k, 0)$.

For $t = 1$, we distinguish whether or not the current invocation $\mathcal{I}$ makes a recursive call with $t = 0$ and whether it makes one or two recursive calls. If $\mathcal{I}$ makes no recursive call with $t = 0$, we obtain $I(k, 1) \leqslant 1 + 2I(k-1, 1)$ because each case of Step 7 makes at most two recursive calls, with parameters no greater than $k - 1$. If $\mathcal{I}$ makes only one recursive call, with $t = 0$, we obtain $I(k, 1) \leqslant 1 + I(k-1, 0)$ because this recursive call has parameter no greater than $k - 1$. Finally, if $\mathcal{I}$ makes two recursive calls, at least one of them with $t = 0$, $\mathcal{I}$ must have applied Case 7.2 or 7.4. Let $\mathcal{I}'$ be one of the invocations $\mathcal{I}$ makes with $t = 0$. If $t = 0$ for invocation $\mathcal{I}'$ because $\mathcal{I}'$ terminates in Step 1 or 2, we obtain $I(k, 1) \leqslant 2 + I(k - 1, 0)$ by counting invocations $\mathcal{I}$ and $\mathcal{I}'$ and the number of recursive calls spawned by the sibling invocation of $\mathcal{I}'$, which cannot be more than $I(k - 1, 0)$. So assume that $t = 0$ for invocation $\mathcal{I}'$ and that $\mathcal{I}'$ does make further recursive calls. Then the sibling group chosen in invocation $\mathcal{I}$ must agree between the input forests of invocation $\mathcal{I}'$.

If invocation $\mathcal{I}$ applies Case 7.2 and makes two recursive calls, we observe that $m \geqslant 3$ because a_0 is a member of $\mathcal{I}$'s sibling group, a_0 is not a descendant of l, and l has at least two descendants in the sibling group. Furthermore, $s_1 > 0$. Thus, after cutting e_{a_1}, a_2 has a sibling forest B'_2 that does not include a_0. Since a_0 also has a sibling forest B_0 that does not include a_2, $\mathcal{I}$'s sibling group cannot agree between $\dot{T}_1$ and $\dot{F}_2$ after cutting e_{a_1}. This implies that $t = 1$ for the first recursive call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k - 1, a_0)$, and $\mathcal{I}'$ is the second recursive call $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\}, k - s_1, a_0\big)$. This gives the recurrence $I(k, 1) = 1 + I(k-1, 1) + I(k-s_1, 0)$. Since no two members of $\mathcal{I}$'s sibling group are siblings in F_2 and a_0 is not a descendant of l, cutting edges $e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}$ can make $\mathcal{I}$'s sibling group agree between F_1 and F_2 only if $r = 2$, $s_2 = 0$, and either l is a root of F_2 or the only pendant nodes of the path from l to the root of its component in F_2 are members of $\mathcal{I}$'s sibling group. Thus, since we assume we make two recursive calls, we must have $s_1 \geqslant 2$, that is, the recurrence for this case is $I(k, 1) \leqslant 1 + I(k-1, 1) + I(k-2, 0)$.

Finally, if invocation $\mathcal{I}$ applies Case 7.4, observe that, since a_0 is a descendant of l and has a group of sibling trees B_0 that do not contain any member a_i of $\mathcal{I}$'s sibling group, this sibling group can be made to agree between F_1 and F_2 only by cutting e_{a_x}. Moreover, since no member a_i of $\mathcal{I}$'s sibling group is a root of F_2, cutting e_{a_x} can make this sibling group agree between F_1 and F_2 only if $m = 2$. Thus, Case 7.4 makes only one recursive call and we obtain $I(k, 1) = 1 + I(k - 1, 0)$ in this case.

By combining the different possibilities for the case when $t = 1$, we obtain the recurrence

$$\begin{aligned} I(k,1) \leqslant \max(&1 + 2I(k-1,1), 2 + I(k-1,0), \\ &1 + I(k-1,1) + I(k-2,0)). \end{aligned}$$

Simple substitution now shows that $I(k,t) \leqslant (1+\sqrt{2})^{2+\max(0,k-t+3)} + 2(t-1)$. □

Lemma 5.13. *For two rooted X-trees T_1 and T_2 and a parameter k, the invocation* $\textsc{Maf}\,(T_1, T_2, k, \mathit{nil})$ *returns "yes" if and only if $d_{SPR}\,(T_1, T_2) = e\,(T_1, T_2, T_2) \leqslant k$.*

Proof. We use induction on k to prove the following two claims, which together imply the lemma: (i) If $e\,(T_1, T_2, F_2) > k$, the invocation $\textsc{Maf}\,(F_1, F_2, k, a_0)$ returns "no". (ii) If $e\,(T_1, T_2, F_2) \leqslant k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0 is not a root of F and $a_0 \nsim_F a_i$, for every sibling a_i of a_0 in F_1, the invocation $\textsc{Maf}\,(F_1, F_2, k, a_0)$ returns "yes".

(i) Assume $e\,(T_1, T_2, F_2) > k$. If $k < 0$, the invocation returns "no" in Step 1. If $k \geqslant 0$, assume for the sake of contradiction that the invocation returns "yes". If it does so in Step 2, then F_2 is an AF of T_1 and T_2, that is, $e\,(T_1, T_2, F_2) = 0 \leqslant k$, a contradiction. Otherwise it returns "yes" in Step 7 or 8. Thus, there exists a child invocation $\textsc{Maf}\,(F_1', F_2', k', a_0')$ that returns "yes", where $F_2' = F_2 \div E$ and $k' = k - |E|$, for some non-empty edge set E. By the inductive hypothesis, we therefore have $e\,(T_1, T_2, F_2') \leqslant k'$ and, hence, $e\,(T_1, T_2, F_2) \leqslant k' + |E| = k$, again a contradiction.

(ii) Assume $e\,(T_1, T_2, F_2) \leqslant k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0 is not a root of F and $a_0 \nsim_F a_i$, for every sibling a_i of a_0 in F_1. In particular, $k \geqslant 0$ and the invocation $\textsc{Maf}\,(F_1, F_2, k, a_0)$ produces its answer in Step 2, 7 or 8. If it produces its answer in Step 2, it answers "yes". Next we consider Steps 7 and 8 and prove that at least one of the recursive calls made in each case returns "yes", which implies that the current invocation returns "yes".

In Step 7, $a_0 \neq \text{nil}$, that is, a_0 is not a root of F and $a_0 \nsim_F a_i$, for every sibling a_i of a_0 in F_1. In Case 7.1, $a_x \nsim_{F_2} a_i$ and, hence, $a_x \nsim_F a_i$, for all $i \neq x$. Thus, a_x is a root of F because otherwise the components of F containing a_x and a_0 would overlap in F_1. This implies that there exists an edge set E such that $e_{a_x} \in E$ and $F = F_2 \div E$, that is, the recursive call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$ returns "yes".

In Case 7.2, $a'_1 \nsim_F b'_1$, for all leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_{1j}}$, $1 \leqslant j \leqslant s_1$, because the path between a'_1 and b'_1 would overlap the component containing a_0. Thus, there exists an edge set E such that $F_2 \div E = F$ and either $e_{a_1} \in E$ or $\{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\} \subseteq E$. If we make two recursive calls in Case 7.2, this shows that one of them returns "yes". If we make only one recursive call, we have $s_1 = 1$ and $s_r = 0$. Assume this recursive call returns "no". Then $e_{a_1} \in E$. Let F' be the forest obtained by cutting e_{B_1} instead of e_{a_1}, resolving the pair $\{a_1, a_r\}$, cutting edge $e_{(a_1,a_r)}$ instead of e_{a_r} if $e_{a_r} \in E$, and otherwise cutting the same edges as in E. F and F' are identical, except that in F', a_1 and a_r are siblings and no leaf in $F_2^{B_1}$ can reach a leaf not in $F_2^{B_1}$. The former cannot introduce any triples incompatible with F_1 because a_1 and a_r are siblings in F_1. The latter cannot introduce any overlapping components because this would imply that $a'_1 \sim_F b'_1$, for two leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_1}$, and we already argued that no such path can exist. Thus, F' is also an MAF of F_1 and F_2. Finally, a_0 is not a root in F' and $a_0 \nsim_{F'} a_1$ because otherwise $a_0 \sim_F a_r$. Thus, since there exists an edge set E' such that $F' = F_2 \div E'$ and $e_{B_1} \in E'$, the recursive call $\textsc{Maf}\,(F_1, F_2 \div \{e_{B_1}\}, k-1, a_0)$ returns "yes", a contradiction.

In Case 7.3, there exists an edge set E such that $F = F_2 \div E$ and $E \cap \{e_{a_x}, e_{B_x}\} \neq \varnothing$. Otherwise we would have $a_x \sim_F a_0$ or $a'_x \sim_F b'_x$, for two leaves $a'_x \in F_2^{a_x}$ and $b'_x \notin F_2^{a_i}$, for all $1 \leqslant i \leqslant m$, but we have $a_x \nsim_F a_0$ and the path between a'_x and b'_x would overlap the component of F that contains a_0. Now, if $e_{B_x} \notin E$, we construct an MAF F' of F_1 and F_2 such that a_0 is not a root in F' and $a_0 \nsim_{F'} a_i$, for all $1 \leqslant i \leqslant m$, and an edge set E' such that $F' = F_2 \div E'$ and $e_{B_x} \in E'$ as in Case 7.2. Thus, the invocation $\textsc{Maf}\,(F_1, F_2 \div \{e_{B_x}\}, k-1, a_0)$ returns "yes".

In Case 7.4, finally, one of the two recursive calls $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$ or $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}\}, k - s_x, a_0\big)$ must return "yes", by the same arguments as in Case 7.2. Thus, if $m > 2$ and $r > 2$, one of the recursive calls we make returns "yes". If $m > 2$ and $r = 2$, we observe that after cutting edges

$e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}$, a_x and a_0 satisfy the conditions of Case 7.3, which shows that we can cut edge B'_x immediately after cutting these edges. Thus, if the recursive call $\text{MAF}\,(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$ returns "no", the recursive call $\text{MAF}\,(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \ldots, e_{B_{xs_x}}, e_{B'_x}\}, k - s_x, a_0\big)$ must return "yes" in this case. Finally, if $m = 2$, since $a_x \nsim_F a_0$ and any path between two leaves $a'_x \in F_2^{a_x}$ and $b'_x \notin F_2^{a_x} \cup F_2^{a_0}$ would overlap the component of F that contains a_0, a_x is a root in F. Thus, the invocation $\text{MAF}\,(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$ returns "yes" in this case.

Now consider Step 8. In Cases 8.1, 8.2, 8.3, and 8.5, Lemmas 5.4, 5.6, 5.8, 5.7 and the inductive hypothesis show that one of the recursive calls returns "yes" and, thus, the current invocation returns "yes".

In Case 8.4, Lemma 5.7 and the inductive hypothesis show that one of the recursive calls $\text{MAF}\,(F_1, F_2 \div \{e_{a_1}, e_{a_2}\}, k-2, \text{nil})$, $\text{MAF}\,(F_1, F_2 \div \{e_{B_1}, e_{B_2}\}, k-2, \text{nil})$, $\text{MAF}\,(F_1, F_2 \div \{e_{B_1}\}, k-1, a_2)$, $\text{MAF}\,(F_1, F_2 \div \{e_{B_2}\}, k-1, a_1)$, $\text{MAF}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ or $\text{MAF}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$ would return "yes". Thus, we need to argue only that we can cut *both* e_{B_i} and $e_{B'_i}$ in the third and fourth recursive calls, and that the last two recursive calls are not necessary when we do not make them.

First consider cutting e_{B_1} and setting $a_0 = a_2$ in the third recursive call. We require this call to return "yes" only if the other calls return "no". The forest $F'_2 = F_2 \div \{e_{B_1}\}$ contains a triple $a'_1|a'_2b'_2$, where $a'_1 \in F_2^{a_1}$, $a'_2 \in F_2^{a_2}$, and $b'_2 \in F_2^{B_2}$, while F_1 contains the triple $a'_1a'_2|b'_2$. Thus, in order to obtain an MAF of F_1 and F'_2 where a_2 exists and is not a root, we need to cut either e_{a_1} or $e_{B'_1}$. Since the first and fifth recursive calls return "no", however, we know that cutting a_1 cannot lead to an MAF of F_1 and F_2, and we can cut $e_{B'_1}$ along with edge e_{B_1}. The case when we cut $e_{B'_2}$ along with edge e_{B_2} is analogous.

If we do not make the recursive call $\text{MAF}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$, then l has exactly one sibling, a_i, and its parent p_l is either a root or has exactly one sibling, a_j. Thus, the forest $F'_2 = F_2 \div \{e_{a_1}\}$ contains a triple $a'_2b'_2|a'_i$, where $a'_2 \in F_2^{a_2}$, $b'_2 \in F_2^{B_2}$, and $a'_i \in F_2^{a_i}$, while F_1 contains the triple $a'_2a'_i|b'_2$. In order to obtain an MAF of F_1 and F'_2 where a_2 exists and is not a root, we therefore need to cut either e_{a_i} or $e_{B_i} = e_l$. If p_l is a root, cutting either edge has the same effect. If p_l has a sibling a_j and we cut e_{a_i}, we can obtain an alternate MAF by cutting e_l instead of e_{a_i}, resolving $\{a_i, a_j\}$, cutting edge $e_{(a_i,a_j)}$ instead of e_{a_j} if $e_{a_j} \in E$, and otherwise cutting the same edges as in E. Thus, we can always replace the recursive call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ with the call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}, e_l\}, k-2, a_2)$ without affecting the correctness of the algorithm. If this call returns "yes", however, then so does the call $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_1}, e_{B'_1}\}, k-2, a_2\big)$ because we can yet again obtain an alternate MAF by cutting edges e_{B_1} and e_{B_2} instead of edges e_{a_1} and e_l, resolving $\{a_1, a_i\}$, cutting $e_{(a_1,a_i)}$ instead of e_{a_i} if $e_{a_i} \in E$, and otherwise cutting the same edges as in E. Thus, the call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ can be eliminated altogether. An analogous argument shows that we can eliminate the call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$.

In Case 8.6, Lemma 5.10 and the inductive hypothesis show that one of the recursive calls $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_1}\}, k-1, \text{nil})$, $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, \text{nil})$, $\textsc{Maf}\,\big(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \ldots, e_{B_{1s_1}}\}, k-s_1, \text{nil}\big)$ or $\textsc{Maf}\,(F_1, F_2 \div \{e_{B_2}\}, k-1, \text{nil})$ would return "yes". We need to show that the call $\textsc{Maf}\,(F_1, F_2 \div \{e_{a_2}\}, k-1, \text{nil})$ is not necessary. To see this, observe that, if l is a root, then cutting e_{a_2} or e_{B_2} has the same effect. If l is not a root but has a member a_i of the current sibling group as a sibling, then we can obtain an alternate MAF by cutting e_{B_2} instead of e_{a_2}, resolving $\{a_2, a_i\}$, cutting $e_{(a_2,a_i)}$ instead of e_{a_i} if $a_i \in E$, and otherwise cutting the same edges as in E.

In Case 8.7, finally, the correctness follows from Lemmas 5.9 and 5.10 if we can show that setting $a_0 \neq a_1$ is correct for the second and fourth recursive calls. This, however, follows because, if neither the first nor the third recursive call returns "yes", then in every MAF F of F_1 and F_2, a_1 exists and there exist two leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_{1j}}$, for some $1 \leqslant j \leqslant s_1$, such that $a'_1 \sim_F b'_1$. □

5.4 A 3-Approximation Algorithm for Rooted MAFs of multifurcating trees

We now show how to modify the FPT algorithm from Section 5.3 to obtain a 3-approximation algorithm with running time $O(n \log n)$. This algorithm is easy to implement iteratively, and this may be preferable in practice. In order to minimize the differences to the FPT algorithm, however, we describe it as a recursive algorithm. There are four differences to the FPT algorithm:

- Instead of deciding whether $e(T_1, T_2, F_2) \leqslant k$, an invocation Maf (F_1, F_2) returns an integer k'' such that $e(T_1, T_2, F_2) \leqslant k'' \leqslant 3e(T_1, T_2, F_2)$. Thus, there is no need for a parameter k to the invocation or for an equivalent of Step 1 of the FPT algorithm, and whenever Step 2 of the FPT algorithm would have returned "yes", we now return 0 as our approximation k'' of $e(T_1, T_2, F_2)$ because F_2 is an AF of T_1 and T_2.

- We execute Step 5 only if the immediately preceding execution of Step 4 chose a new sibling group. This ensures that this step is executed only once per sibling group. As discussed in Lemma 5.11, the cost per sibling group $\{a_1, a_2, \ldots, a_m\}$ is $\mathrm{O}(m)$. Thus, the total cost of all executions of Step 5 in all recursive invocations is $\mathrm{O}(n)$. After executing Step 5, implemented as discussed in Lemma 5.11, every node p in F_2 has at most one node a_i as a child, which is stored as p's *representative* r_p. This allows us to merge sibling pairs $\{a_i, a_j\}$ that arise as a result of edge cuts after we executed Step 5 without re-executing this step: Whenever we contract a degree-2 vertex whose only child is a node a_i in the current sibling group and whose parent is p, we set $r_p := a_i$ if $r_p = \text{nil}$; otherwise we resolve the sibling pair $\{a_i, r_p\}$ as in Step 5 and store (a_i, r_p) as p's new representative.

- We do not execute Step 6, as this would require linear time per invocation. Instead, we assign a *depth estimate* d_x to each node x and use it to choose the order in which to inspect the members of the current sibling group. Initially, d_x is x's depth in T_2, which is easily computed in linear time for all nodes $x \in T_2$. In general, d_x is one more than the depth of p_x in T_2, where p_x is x's parent in F_2. In particular, d_x is an upper bound on x's depth in F_2 and, for two nodes x and y with LCA l, we have $d_y > d_x$ if x is a child of l and y is not. When choosing a new sibling group in Step 4, we insert all group members into a max-priority queue Q, with their depth estimates as their priorities. When contracting the degree-2 parent p_x of a node x, we set $d_x := d_{p_x}$. If x is a member of the current sibling group, we update its priority in Q. When resolving a sibling pair $\{a_i, a_j\}$, we remove a_i and a_j from Q, set $d_{(a_i,a_j)} := d_{a_i}$, and insert (a_i, a_j) into Q. Finally, when cutting an edge e_{a_i}, for a member a_i of the current sibling group, we remove a_i from Q. These updates take $\mathrm{O}(\log n)$ time per modification of F_2. Since we modify F_2 at most $\mathrm{O}(n)$ times, the total cost of all priority queue operations is $\mathrm{O}(n \log n)$.

- We do not distinguish between Steps 7 and 8 (having no concept of a_0) and also do not distinguish between the various cases of the steps. Instead, we have a single Step 7 with four cases that each make one recursive call (see Figure 5.7).

  7.1. If $m = 2$ and this sibling group was chosen in Step 4 of the current invocation, make one recursive call $\textsc{Maf}\left(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}, e_{a_2}\}\right)$ and return 3 plus its return value.

  7.2. If $m = 2$ and this sibling group was chosen in Step 4 of a previous invocation, make one recursive call $\textsc{Maf}\left(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}, e_{a_2}, e_{p_{a_2}}\}\right)$ and return 4 plus its return value.

  7.3. If $m > 2$ and this sibling group was chosen in Step 4 of the current invocation, let a_1 and a_2 be the two entries with maximum priority in Q, ordered so that $d_{a_1} \geqslant d_{a_2}$. If a_2's parent has a single sibling, this sibling is a member a_j of the current sibling group, and a_1's parent is either a root or has a sibling that is not a member of the current sibling group, let $x = 2$; otherwise let $x = 1$. Remove a_x from Q, make one recursive call $\textsc{Maf}\left(F_1, F_2 \div \{e_{a_x}, e_{p_{a_x}}\}\right)$, and return 2 plus its return value.

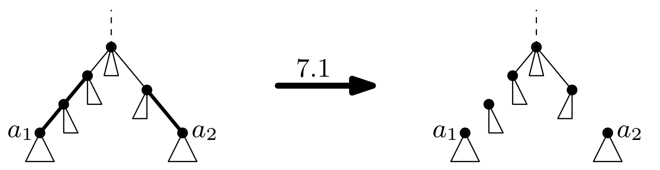
  7.4. If $m > 2$ and this sibling group was chosen in Step 4 of a previous invocation, delete the node a_1 with maximum priority from Q, make one recursive call $\textsc{Maf}\left(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}\}\right)$, and return 2 plus its return value.

Using these modifications, we obtain the following theorem.

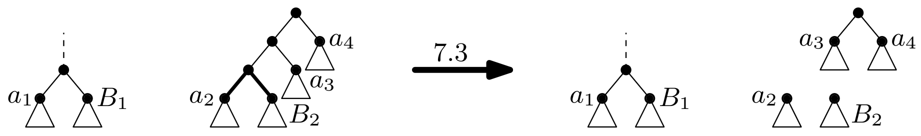
Theorem 5.14. *Given two rooted X-trees T_1 and T_2, a 3-approximation of $e(T_1, T_2, T_2) = d_{SPR}(T_1, T_2)$ can be computed in $\mathrm{O}(n \log n)$ time.*

Proof. We use the algorithm just described. This algorithm consists of Steps 2–5 of the FPT algorithm plus the modified Step 7 above. In addition, there is a linear-time preprocessing step for computing the initial depth estimates of all nodes of T_2. We argued already that all executions of Step 5 take linear time in total. In Lemma 5.11,

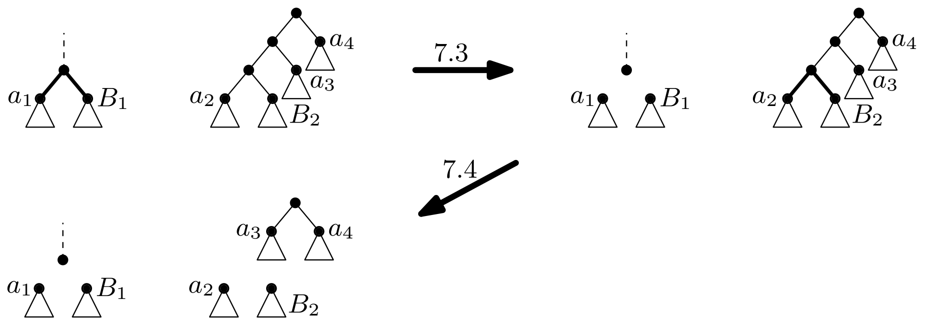
(a) $m = 2$: A single application of Case 7.1 resolves the sibling group $\{a_1, a_2\}$.



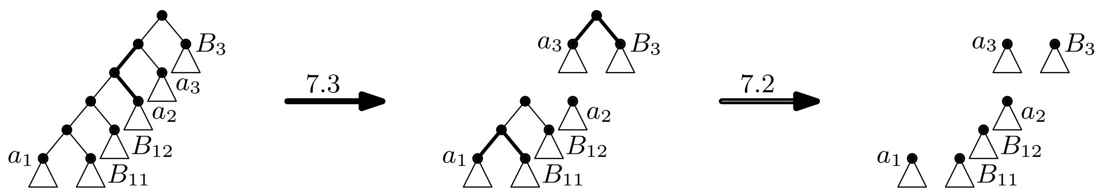
(b) $m = 4$: A single application of Case 7.3 resolves the sibling group $\{a_1, a_2, a_3, a_4\}$.



(c) The set of edges cut on the same input if we did not give preference to a_2 in Case 7.3. Since cutting edge e_{B_2} makes the sibling group agree with F_1 in this case, this would not give a 3-approximation.



(d) $m = 4$: The optimal solution needs to cut 2 edges, while our algorithm cuts 6.



(e) The sequence of cuts on the same input if we did not give preference to a_2 in Case 7.3. Note that we obtain the same forest.

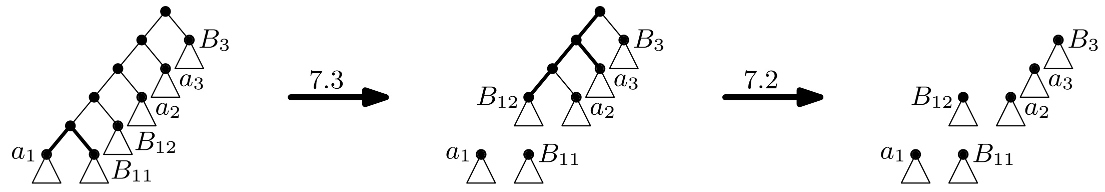


Figure 5.7: Illustration of the various cases of Step 7 of the approximation algorithm. Only $\dot{F}_2$ is shown. Edges that are cut in each step are shown in bold.

we showed that each execution of Step 2, 3 or 4 of the FPT algorithm takes constant time. In the approximation algorithm, if Step 4 chooses a new sibling group, it also needs to insert the members of the sibling group into the priority queue. This takes $\mathrm{O}(m \log m)$ time, $\mathrm{O}(n \log n)$ in total for all sibling groups. Each execution of Step 7 takes $\mathrm{O}(\log n)$ time, constant time for the modifications of F_2 it performs and $\mathrm{O}(\log n)$ time for the $\mathrm{O}(1)$ corresponding priority queue operations. Thus, to obtain the claimed time bound of $\mathrm{O}(n \log n)$ for the entire algorithm, it suffices to show that each step of the algorithm is executed $\mathrm{O}(n)$ times.

This is easy to see for Steps 3, 5, and 7: Each execution of Step 5 reduces the number of nodes in R_t by one, the number of nodes in R_t never increases, and initially R_t contains the n leaves of T_1. Each execution of Step 3 or 7 cuts at least one edge in F_1 or F_2, and initially these two forests have $\mathrm{O}(n)$ edges.

For Steps 2 and 4, we observe that they cannot be executed more often than Steps 3, 5, and 7 combined because any two executions of Step 2 or 4 have an execution of Step 3, 5 or 7 between them.

It remains to bound the approximation ratio of the algorithm. First observe that the value k' returned by the algorithm satisfies $k' \geqslant e(T_1, T_2, T_2)$ because the input forest F_2 of the final invocation $\textsc{Maf}(F_1, F_2)$ is an AF of T_1 and T_2 and the algorithm returns the number of edges cut to obtain F_2 from T_2. To prove that $k' \leqslant 3e(T_1, T_2, T_2)$, let r be the number of descendant invocations of the current invocation $\textsc{Maf}(F_1, F_2)$, not counting the current invocation itself, and let k'' be its return value. We use induction on r to prove that $k'' \leqslant 3e(T_1, T_2, F_2)$ if $\textsc{Maf}(F_1, F_2)$ chooses a new sibling group in Step 4. We call such an invocation a *master invocation.* We also consider the last invocation of the algorithm to be a master invocation. For two master invocations without another master invocation between them, all invocations between the two invocations are *slave invocations* of the first of the two master invocations, as they manipulate the sibling group chosen in this invocation.

As a base case observe that, if $r = 0$, then $e(T_1, T_2, F_2) = 0$ and the invocation $\textsc{Maf}(F_1, F_2)$ returns $k'' = 0$ in Step 2. For the inductive step, consider a master invocation $\mathcal{I} = \textsc{Maf}(F_1, F_2)$ with $r > 0$, and let $\mathcal{I}' = \textsc{Maf}(F_1', F_2')$ be the first master invocation after $\mathcal{I}$. By the inductive hypothesis, $\mathcal{I}'$ returns a value k''' such that $k''' \leqslant 3e(T_1, T_2, F_2')$.

If $m = 2$ in invocation $\mathcal{I}$, we invoke Case 7.1, which cuts edges e_{a_1}, $e_{p_{a_1}}$, and e_{a_2} and, thus, makes the sibling group agree between F_1 and F_2 (see Figure 5.7(a)). This implies that $k'' = k''' + 3$. By Lemma 5.5, we have $e(T_1, T_2, F_2') \leqslant e(T_1, T_2, F_2) - 1$. Since $k''' \leqslant 3e(T_1, T_2, F_2')$, this shows that $k'' \leqslant 3e(T_1, T_2, F_2)$.

If $m > 2$ in invocation $\mathcal{I}$, this invocation applies Case 7.3, each of its slaves applies Case 7.2 or 7.4, and Case 7.2 is applied at most once. Since the sibling group $\{a_1, a_2, \ldots, a_m\}$ does not agree between F_1 and F_2, at least one edge cut in F_2 is necessary to make the sibling group agree between F_1 and F_2. We distinguish whether one or more edge cuts are required.

If one cut suffices (Figures 5.7(b) and 5.7(c)), then there are at most two components of F_2 that contain members of the current sibling group because resolving overlaps in F_1 between q components of F_2 requires at least $q - 1$ cuts in F_2. Consequently, exactly one component C contains at least two members of the sibling group. The existence of at least one such component follows because $m > 2$. If we had another component C' containing at least two members of the current sibling group, then at least one cut would be required in each of C and C' to make the sibling group agree between F_1 and F_2, but we assumed that one cut suffices.

For a single cut to suffice to make the current sibling group agree between F_1 and F_2, C must consist of a single path of nodes $x_1, x_2, \ldots, x_t$ such that, for $1 \leqslant j < t$, x_j has two children: x_{j+1} and a member a_{i_j} of the current sibling group; x_t has a member a_{i_t} of the current sibling group as a child, as well as a group B_{i_t} of siblings of a_{i_t} such that no member a_h of the current sibling group belongs to $F_2^{B_{i_t}}$. Thus, cutting edges $e_{a_{i_t}}$ and $e_{p_{a_{i_t}}}$ makes the sibling group agree between F_1 and F_2. Now observe that a_{i_t} and $a_{i_{t-1}}$ are the two members of the current sibling group with the greatest depth estimates in C. If there exists another component C' of F_2 that contains a member a_h of the current sibling group, then a_h is the only such node in C'. Thus, the two maximum priority entries in Q are either a_{i_t} and $a_{i_{t-1}}$ or a_{i_t} and a_h. In both cases, invocation $\mathcal{I}$ cuts edges $e_{a_{i_t}}$ and $e_{p_{a_{i_t}}}$ because a_h either has no parent or its parent does not have a member of the current sibling group as a sibling, and a_{i_t} is preferred over $a_{i_{t-1}}$ by invocation $\mathcal{I}$ because a_{i_t}'s parent does have a member of the current sibling group as its only sibling (namely $a_{i_{t-1}}$) and has a greater depth estimate than $a_{i_{t-1}}$. Thus, in $\mathcal{I}$'s child invocation, the current sibling group agrees between F_1 and F_2, which implies that this child invocation is $\mathcal{I}'$ and $k'' = k''' + 2$. Moreover, since the sibling group does not agree between F_1 and F_2 in invocation $\mathcal{I}$, we must have $e(T_1, T_2, F_2') \leqslant e(T_1, T_2, F_2) - 1$ and, hence, $k'' = k''' + 2 \leqslant 3e(T_1, T_2, F_2') + 2 \leqslant 3e(T_1, T_2, F_2)$.

For the remainder of the proof assume at least two cuts are necessary to make the sibling group $\{a_1, a_2, \ldots, a_m\}$ agree between F_1 and F_2 (Figures 5.7(d) and 5.7(e)), and assume the members of the sibling group are ordered by their depth estimates. Let $i_1, i_2, \ldots, i_s$ be the indices such that invocation $\mathcal{I}$ and its slaves cut edges $\{e_{a_{i_j}}, e_{p_{a_{i_j}}} \mid 1 \leqslant j \leqslant s\}$ and, for all $1 \leqslant j \leqslant s$, let B_{i_j} be the set of a_{i_j}'s siblings at the time we cut edges $e_{a_{i_j}}$ and $e_{p_{a_{i_j}}}$. Assume for now that invocation $\mathcal{I}$ cuts edges e_{a_1} and $e_{p_{a_1}}$. Then, for all $1 \leqslant j \leqslant s$, $F_2^{B_{i_j}}$ contains no member of the current sibling group because such a member a_h would have a greater depth estimate than a_{i_j} and hence e_{a_h} would have been cut before $e_{a_{i_j}}$. This implies that there exists an edge set E such that $F_2 \div E$ is an MAF of F_1 and F_2 and $|E \cap \{e_{a_{i_j}}, e_{B_{i_j}} \mid 1 \leqslant j \leqslant s\}| \geqslant s - 1$. Since cutting edges $e_{a_{i_j}}$ and $e_{B_{i_j}}$ produces the same result as cutting edges $e_{a_{i_j}}$ and $e_{p_{a_{i_j}}}$, this shows that $e(T_1, T_2, F_2') \leqslant e(T_1, T_2, F_2) - (s - 1)$.

If $s \geqslant 3$, we have $2s \leqslant 3(s-1)$ and, hence, $k'' \leqslant k''' + 3(s-1) \leqslant 3(e(T_1, T_2, F_2') + s - 1) \leqslant 3e(T_1, T_2, F_2)$. If $s < 3$, we observe that $e(T_1, T_2, F_2') \leqslant e(T_1, T_2, F_2) - 2$ because the current sibling group agrees between F_1 and F_2' and we assumed that at least two cuts are necessary in F_2 to make this sibling group agree between F_1 and F_2. Thus, $k'' \leqslant k''' + 2s \leqslant 3e(T_1, T_2, F_2') + 4 \leqslant 3e(T_1, T_2, F_2)$.

It remains to deal with the case when invocation $\mathcal{I}$ cuts edges e_{a_2} and $e_{p_{a_2}}$. If $a_1 \notin F_2^{B_2}$, then again $F_2^{B_{i_j}}$ contains no member of the current sibling group, for all $1 \leqslant j \leqslant s$, and the same argument as above shows that $k'' \leqslant 3e(T_1, T_2, F_2)$. If $a_1 \in F_2^{B_2}$, then observe that either the path in F_2 between a_1 and p_{a_2} has at least two internal nodes or a_2 has at least two siblings because otherwise invocation $\mathcal{I}$ would prefer a_1 over a_2 because a_1 has the greater depth estimate. This implies that a_2 has the same parent before and after cutting edges e_{a_1} and $e_{p_{a_1}}$, and a_1 has the same parent before and after cutting edges e_{a_2} and $e_{p_{a_2}}$. Thus, $\mathcal{I}$ and its child invocation cut the same four edges e_{a_1}, $e_{p_{a_1}}$, e_{a_2}, and $e_{p_{a_2}}$ as would have been cut if invocation $\mathcal{I}$ had not preferred a_2 over a_1. The same argument as above now shows that $k'' \leqslant 3e(T_1, T_2, F_2)$. $\square$

5.5 Conclusions

We developed efficient algorithms for computing MAFs of multifurcating trees. Our fixed-parameter algorithm achieves the same running time as in the binary case and our 3-approximation algorithm achieves a running time of $\mathrm{O}(n \log n)$, almost matching the linear running time for the binary case. Implementing and testing our algorithms will be the focus of future work.

Two other directions to be explored by future work are practical improvements of the running time of the FPT algorithm presented here and extending our FPT algorithm so it can be used to compute maximum *acyclic* agreement forests (MAAFs) and, hence, the hybridization number of multifurcating trees. To speed up our FPT algorithm for computing MAFs, it may be possible to extend the reduction rules used by Linz and Semple [62] for computing MAAFs of multifurcating trees so they can be applied to MAF computations, and combine them with the FPT algorithm in this chapter. The fastest fixed-parameter algorithms for computing MAAFs of binary trees [2, 33, 102], including the algorithm of Chapter 4, are extensions of the binary MAF algorithms of Whidden and Zeh [103] and Whidden et al. [100] (Chapter 3). These algorithms were developed by examining which search branches of the binary MAF algorithm get "stuck" with cyclic agreement forests and consider cutting additional edges to avoid these cycles [2] or refine cyclic agreement forests to acyclic agremeent forests [33, 102]. Similarly, Chen and Wang [31] recently extended the MAF fixed-parameter algorithm for two binary trees to compute agreement forests of multiple binary trees using an iterative branching approach. The proofs of various structural lemmas in this chapter prove that *any* MAF can be obtained by cutting an edge set that includes certain edges. To prove this, we started with an arbitrary MAF and an edge set such that cutting these edges yields this MAF, and then we modified this edge set so that it includes the desired set of edges without changing the resulting forest. As in the binary case [102] (Chapter 4), the same lemmas apply also to MAAFs; since the modifications in the proofs do not change the resulting AF, the only needed change in the proof is to start with an edge set such that cutting it yields an MAAF. Thus, numerous lemmas in this chapter may also form the basis for an efficient algorithm for computing MAAFs.

Finally, we note that our fixed-parameter algorithm becomes greatly simplified when comparing a binary tree to a multifurcating tree. This is common in practice when, for example, comparing many multifurcating gene trees to a binary reference tree or binary supertree, as will be shown in Chapter 8. To see this, suppose F_1 is binary, that is, $m = 2$ in every case of the FPT algorithm. Then only Cases 8.1, 8.2 and 8.3 apply in Step 8 and Step 7 never applies. Using our observation that cutting e_{B_2} is not necessary in Case 8.2 when $m = 2$, our algorithm becomes similar to the MAF algorithm for binary trees [100] (Chapter 3). We further note, in the interest of practical efficiency, that cutting a_2 is unnecessary in this case when the parent of a_1 is a binary node (and, indeed, our algorithm is then identical to the algorithm of [100] (Chapter 3) when applied to two binary trees).

Chapter 6

Edge Protection: Computing MAFs of Binary Trees in $O\left(2^k n\right)$-time

6.1 Introduction

In this chapter, we present an algorithm for computing the SPR Distance of two binary rooted trees in $O\left(2^k n\right)$ time. This algorithm is a refinement for our earlier algorithm with running time $O\left(2.42^k n\right)$ (Chapter 3) and achieves its improved running time using a novel edge protection scheme that reduces the search space to be explored by the algorithm. This is stated formally in the following theorem.

Theorem 6.1. *Given two rooted binary phylogenetic trees with n leaves, it takes $O\left(2^k n\right)$ time to decide whether their SPR distance is at most k.*

As with our previous algorithms, Theorem 6.1 can be used to compute the SPR distance between two given phylogenies by starting with a guess of $k = 0$ for the distance and repeatedly trying larger values of k until the algorithm returns an affirmative answer. Since the last invocation dominates the running time, this gives the following corollary.

Corollary 6.2. *Given two rooted binary phylogenetic trees with n leaves, it takes $O\left(2^k n\right)$ time to compute their SPR distance, where k is the computed distance.*

6.2 Preliminaries

In this section we introduce three new lemmas that will allow us to state our subsequent proofs in an intuitive way. The first is a simplification of Lemma 2.4; Lemma 6.3 states that forests of other forests do not contain incompatible triples and overlapping components. The second, Lemma 6.4 shows that after cutting a set of edges that are sufficient to obtain an agreement forest and any set of additional edges the result is still an agreement forest. The third, Lemma 6.5, shows that, for a tree T_1 with sibling pair (a, c) and a forest F_2, if we cut each pendant edge between a and c in F_2 then it is either unnecessary to cut any edges on the path between a and c in F_2 or all such edges can be replaced by cutting the parent edge of the LCA of a and c in F_2.

Lemma 6.3. *Given two forests F_1 and F_2 with label set X, F_2 is a forest of F_1 if and only if all triples of F_2 are compatible with F_1 and no two components of F_2 overlap in F_1.*

Lemma 6.4. *Assume T_1 and T_2 are two X-trees, F_2 is a forest of T_2, and E is a subset of edges of F_2 such that $F_2 \div E$ is an agreement forest of T_1 and T_2. Then $F_2 \div E'$ is also an agreement forest of T_1 and T_2, for any superset $E' \supseteq E$.*

Proof. It suffices to prove the lemma for the case when $|E' \setminus E| = 1$. The claim for arbitrary $E' \supseteq E$ then follows by applying this case inductively. By Lemma 6.3, we have to prove that all triples of $F_2 \div E'$ are compatible with T_1 and that no two components of $F_2 \div E'$ overlap in T_1. The former is obvious because the triples of $F_2 \div E'$ are a subset of the triples of $F_2 \div E$ and $F_2 \div E$ is an AF of T_1 and T_2. To prove the latter, let $E' \setminus E = \{e\}$, let C be the component of $F_2 \div E$ that contains e, and let C_1 and C_2 be the two components of $C \div \{e\}$. C_1 and C_2 are the only two components of $F_2 \div E'$ that are not components of $F_2 \div E$. Now assume two components C' and C'' of $F_2 \div E'$ overlap in T_1. If $\{C', C''\} \cap \{C_1, C_2\} = \varnothing$, then C' and C'' are also components of $F_2 \div E$, a contradiction because no two components of $F_2 \div E$ overlap in T_1. If $|\{C', C''\} \cap \{C_1, C_2\}| = 1$, then w.l.o.g. $C' = C_1$ and $C'' \notin \{C_1, C_2\}$. Since C_1 and C'' overlap in T_1, so do C and C'', a contradiction again because C and C'' are components of $F_2 \div E$. Finally, if C_1 and C_2 overlap in T_1, then there exists an edge e' of T_1 and four leaves $u, v \in C_1$ and $x, y \in C_2$ such that e' belongs to the paths from u to v and from x to y. Thus, w.l.o.g. $ux|vy$ is a quartet of C. On the other hand, since e splits C into C_1 and C_2, C also contains the quartet $uv|xy$. This is a contradiction because C cannot contain both quartets. This shows that no two components of $F_2 \div E'$ overlap in T_1. Since we argued above that all triples of $F_2 \div E'$ are compatible with T_1, $F_2 \div E'$ is an AF of T_1 and T_2. $\square$

Lemma 6.5. *Assume T_1 and T_2 are two X-trees, F_1 is a forest of T_1, F_2 is a forest of T_2, and E is a subset of edges of F_2 such that $F_2 \div E$ is an agreement forest of*

T_1 and T_2. Assume further there exists a subset of edges $E'' \subseteq E$ such that (a, c) is a sibling pair of F_1 and $F_2 \div E''$, let E' be the set of edges in E that belong to the path from a to c in F_2, and let u be the LCA of a and c in F_2. Let $E_0 := E \setminus E'$ if $|E'| \leqslant 1$, and $E_0 := E \setminus E' \cup \{e_u\}$ if $|E'| \geqslant 2$. Then $F_2 \div E_0$ is also an agreement forest of T_1 and T_2.

Proof. Consider edge sets E, E', E'', and E_0 as in the lemma. We can assume w.l.o.g. that $E' \neq \varnothing$ because otherwise $E_0 = E$. By Lemma 6.3, we have to prove that all triples of $F_2 \div E_0$ are compatible with T_1 and that no two components of $F_2 \div E_0$ overlap in T_1.

First consider the triples of $F_2 \div E_0$. Every such triple that is also a triple of $F_2 \div E$ is compatible with T_1 because $F_2 \div E$ is an AF of T_1 and T_2. Every triple $xy|z$ of $F_2 \div E_0$ that is not a triple of $F_2 \div E$ involves at least one leaf in $F_2^a \cup F_2^c$. If all three leaves of $xy|z$ belong to $F_2^a \cup F_2^c$, the triple is compatible with T_1 because (a, c) is a sibling pair of both F_1 and $F_2 \div E'' \supseteq F_2 \div E_0$. If $|E'| \geqslant 2$, then $e_u \in E_0$. Thus, if at most two leaves of $xy|z$ belong to $F_2^a \cup F_2^c$, we must have $|E'| = 1$. If exactly two leaves of $xy|z$ belong to $F_2^a \cup F_2^c$, we must have $x, y \in F_2^a \cup F_2^c$ and $z \notin F_2^a \cup F_2^c$. This implies that $xy|z$ is also a triple of F_1, and hence of T_1. Finally, if exactly one leaf belongs to $F_2^a \cup F_2^c$, assume w.l.o.g. that $x \in F_2^a \cup F_2^c$ and $y, z \notin F_2^a \cup F_2^c$. Since $|E'| = 1$ and (a, c) is a sibling pair of $F_2 \div E_0$, there exists a leaf $x' \in F_2^a \cup F_2^c$ such that $x' \sim F_2 \div Eu$. Since $xy|z$ is a triple of $F_2 \div E_0$ and E and E_0 contain the same edges outside of F_2^u, we also have $y \sim F_2 \div Eu \sim F_2 \div Ez$. Thus, $x'y|z$ is a triple of $F_2 \div E$. This triple is incompatible with T_1 because $xy|z$ is and x and x' are the only two leaves in the two triples that belong to $F_2^a \cup F_2^c$. However, this is a contradiction because $F_2 \div E$ is an AF of T_1 and T_2.

It remains to show that no components of $F_2 \div E_0$ overlap in T_1. If $|E'| \leqslant 1$, the only component of $F_2 \div E_0$ that is not a component of $F_2 \div E$ is the component C containing a and c. Thus, if two components of $F_2 \div E_0$ overlap in T_1, one of them is C. Let C' be the other component, and let C'' be the component of $F_2 \div E$ containing the leaf x' from the previous paragraph. Since (a, c) is a sibling pair of F_1 and $F_2 \div E_0$, the edge of T_1 shared by C and C' does not belong to $F_1^{p_a}$. Hence, C'' also contains this edge, and C' and C'' also overlap, a contradiction because $F_2 \div E$ is an AF of T_1 and T_2.

If $|E'| \geqslant 2$, we have $e_u \in E_0$, and the only two components of $F_2 \div E_0$ that may not be components of $F_2 \div E$ are the ones containing u and p_u. Let C_1 and C_2 denote these two components. C_1 cannot overlap any component of $F_2 \div E_0$ because all its leaves belong to $F_2^a \cup F_2^c$ and (a, c) is a sibling pair of F_1. C_2 cannot overlap any component of $F_2 \div E_0$ because we have just argued that C_2 does not overlap C_1 and any other component C' overlapped by C_2 would also have to overlap the component C of $F_2 \div E$ containing p_u, but this is impossible because no two components $F_2 \div E$ overlap in T_1. $\square$

6.3 The Algorithm

Our new algorithm is an extension of our previous algorithm with running time $\mathrm{O}\left(2.42^k n\right)$ (Chapter 3). We summarize this algorithm in Section 6.3.1, using more intuitive notation that generalizes to the cases of the new algorithm. In Section 6.3.2, we discuss our edge protection scheme and introduce refinements of the branching rules used in the algorithm in Section 6.3.1. In Section 6.3.3, we present the complete algorithm using edge protection and the new branching rules. The analysis of the algorithm, and thus the proof of Theorem 6.1, is presented in Section 6.3.5.

6.3.1 The Previous Algorithm

Our previous $\mathrm{O}\left(2.42^k n\right)$-time algorithm for computing an MAF of two phylogenetic trees applied a depth-bounded search approach. An invocation $m\left(F_1, F_2, k\right)$ of our algorithm takes a forest F_1 of T_1, a forest F_2 of T_2, and a parameter k as inputs and returns TRUE or FALSE depending on whether or not $e\left(T_1, T_2, F_2\right) \leqslant k$. To make this decision, the algorithm identifies a small number of edge sets $E_1, E_2, \ldots, E_k$ of F_2 such that $e\left(T_1, T_2, F_2\right) \leqslant k$ if and only if $e\left(T_1, T_2, F_2 \div E_i\right) \leqslant k - |E_i|$, for at least one such edge set E_i. We therefore make a recursive call $m\left(F_1', F_2 \div E_i, k - |E_i|\right)$, for each such edge set E_i, and return TRUE if and only if one of these recursive calls does. F_1' is a forest $F_1' := F_1 \div E'$ obtained by cutting an appropriate set E' of edges of F_1. This edge set is chosen so that we make progress towards obtaining an AF of T_1 and T_2 also from F_1 while maintaining the property that, for every invocation $m\left(F_1, F_2, k\right)$ and every subset E of edges of F_2, $F_2 \div E$ is an AF of T_1 and T_2 if and only if it is an AF of F_1 and F_2. The top-level invocation of the algorithm is $m\left(T_1, T_2, k\right)$ and thus decides whether $d_{SPR}\left(T_1, T_2\right) = e\left(T_1, T_2, T_2\right) \leqslant k$.

Each recursive call $m\left(F_1, F_2, k\right)$ considers a sequence of sibling pairs of F_1 until it finds a sibling pair that is “non-trivial” in a sense we make precise below. To help the invocation decide which sibling pairs to consider, the input forests F_1 and F_2 passed to the invocation are represented as follows: The set of components of F_1 can be divided into two sets. The first set contains a single tree $\dot{T}_1$. All components in the second set are also components of F_2. We denote the forest of these components by F. Similarly, the components of F_2 can be divided into two sets. The components in the first set form a forest $\dot{F}_2$ with the same label set as $\dot{T}_1$, while the components in the second set are the components of F. We now represent F by the set R_d (roots-done) of roots of its components. For the representation of $\dot{T}_1$, we use a set R_t (roots-todo) of nodes of $\dot{T}_1$ that exist also in $\dot{F}_2$ and such that every leaf of $\dot{T}_1$ has exactly one ancestor in R_t. For the top-level invocation $m\left(T_1, T_2, k\right)$, R_t contains all leaves of T_1, and R_d is empty.

An invocation $m\left(F_1, F_2, k\right)$ only considers sibling pairs that have both their members in R_t. For such a sibling pair (a, c) of F_1, if a is a root of F_2, we can make progress towards an AF of F_1 and F_2 by cutting edge e_a in F_1 and moving a from R_t to R_d. In other words, we move the component $F_1^a = F_2^a$ to F. The same analysis applies if c is a root of F_2. Cutting e_a or e_c eliminates the sibling pair (a, c), and we continue to consider the next sibling pair. If a and c are siblings in F_2, we eliminate the sibling pair (a, c) by removing a and c from R_t and adding their parent to R_t and again continue to consider the next sibling pair. We consider these two types of sibling pairs to be *trivial*. A *non-trivial* sibling pair is a sibling pair (a, c) of F_1 such that neither a nor c is a root of F_2 and a and c are not siblings in F_2. Such a sibling pair allows us to identify sets of edges to cut in F_2, in order to make progress towards an AF of T_1 and T_2. In [101, 102] (Chapter 3), we distinguished the following three cases:

Separate components (AC): If $a \nsim_{F_2} c$, then make two recursive calls $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$ and $m\left(F_1', F_2 \div \{e_c\}, k-1\right)$ and return True if and only if one of these recursive calls does.

One pendant subtree (B): If $a \sim_{F_2} c$ and the path from a to c in F_2 has exactly one pendant node b, then make one recursive call $m(F_1', F_2 \div \{e_b\}, k-1)$ and return its answer.

Multiple pendant subtrees (ABC): If $a \sim_{F_2} c$ and the path from a to c in F_2 has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$, then make three recursive calls $m(F_1', F_2 \div \{e_a\}, k-1)$, $m\big(F_1', F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\big)$, and $m(F_1', F_2 \div \{e_c\}, k-1)$ and return True if and only if one of these recursive calls does.

In all three cases, F_1' is the forest obtained from F_1 by dealing with trivial sibling pairs as described above before finding the non-trivial sibling pair (a, c) that triggers the recursive calls. Since the members of a non-trivial sibling pair cannot be siblings in F_2, one of these three cases applies to every non-trivial sibling pair. The case labels in parentheses refer to the sets of edges the algorithm cuts in each of these cases. The correctness of these three cases follows from the following three lemmas shown in [101, 102] (Chapter 3).

Lemma 6.6 (Separate components). *If (a, c) is a non-trivial sibling pair of F_1 and $a \not\sim_{F_2} c$, then $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$ or $e(T_1, T_2, F_2 \div \{e_b\}) = e(T_1, T_2, F_2) - 1$.*

Lemma 6.7 (One pendant node). *If (a, c) is a non-trivial sibling pair of F_1, $a \sim_{F_2} c$, and the path from a to c in F_2 has exactly one pendant node b, then $e(T_1, T_2, F_2 \div \{e_b\}) = e(T_1, T_2, F_2) - 1$.*

Lemma 6.8 (Multiple pendant nodes). *If (a, c) is a non-trivial sibling pair of F_1, $a \sim_{F_2} c$, and the path from a to c in F_2 has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$, then then $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$, $e\big(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}\big) = e(T_1, T_2, F_2) - q$ or $e(T_1, T_2, F_2 \div \{e_b\}) = e(T_1, T_2, F_2) - 1$.*

The base case of the recursion (when no further recursive calls are made) is reached when either $k < 0$ or $R_t \leqslant 2$. In the former case, the algorithm can immediately answer False because $e(T_1, T_2, F_2) \geqslant 0 > k$, for any forest F_2 of T_2. In the latter case, it can answer True because $R_t \leqslant 2$ implies that $\dot{F}_2$ is a forest of $\dot{T}_1$, that is, F_2 is a forest of F_1 and thus an agreement forest of T_1 and T_2.

As shown in [101, 102] (Chapter 3), each recursive call of the algorithm we have just sketched takes linear time, the number of recursive calls is a function of k, and the worst case is when each recursive call applies Case ABC with $q = 2$. In this case, the number of recursive calls is given by the recurrence $I(k) = 1 + 2I(k-1) + I(k-2) = \mathrm{O}\left((1+\sqrt{2})^k\right) = \mathrm{O}\left(2.42^k\right)$, and the running time of the algorithm follows. An important observation to be made is that the number of recursive calls would be bounded by the recurrence $I(k) = 1 + 2I(k-1)$ if we did not use Case ABC. This would give $I(k) = \mathrm{O}\left(2^k\right)$ and, thus, a running time of $\mathrm{O}\left(2^k n\right)$. Thus, the main challenge in obtaining a faster algorithm is to reduce the cost of Case ABC.

6.3.2 Edge Protection and New Branching Rules

In order to improve on the running time of our previous algorithm, we introduce the concept of protected edges we do not need to consider cutting in recursive calls. The intuition is the following: Consider Case ABC. Its first recursive call cuts edge e_a, its third recursive call cuts edge e_c. Denote these two invocations by $\mathcal{I}_a$ and $\mathcal{I}_c$, respectively. The branch of the search started by invocation $\mathcal{I}_c$ does not need to consider cutting edge e_a because, if it is necessary to cut this edge in order to obtain an AF of the desired size, such an AF would be found in the branch started by invocation $\mathcal{I}_a$. Thus, we protect edge e_a in invocation $\mathcal{I}_c$ and thereby forbid invocation $\mathcal{I}_c$ and all its descendant invocations to cut edge e_a. This reduces the exploration of identical combinations of edge cuts in different branches of the algorithm. We used this idea before in [99] (Chapter 5), in order to obtain an $\mathrm{O}\left(2.42^k n\right)$-time algorithm for computing the SPR distance between multifurcating trees. The main difference to [99] (Chapter 3) is that in [99] (Chapter 3) the input to any recursive call contained at most one protected edge, while it seems necessary to introduce potentially many protected edges in order to achieve a running time of $\mathrm{O}\left(2^k n\right)$ even for binary trees. This complicates the analysis substantially.

The exact rules for dealing with protected edges in a given invocation $m\left(F_1, F_2, k\right)$ are as follows: In addition to the representation of F_1 and F_2 discussed in Section 6.3.1, the input of this invocation includes a labelling of the edges of F_1 and F_2 as protected or not. The invocation now processes trivial sibling pairs as in Section 6.3.1 and thereby constructs a forest F_1'. Once it finds a non-trivial sibling pair (a, c), it decides which of the three cases AC, B or ABC applies to it. However, if the applicable case specifies that a recursive call $m\,(F_1', F_2 \div E_i, k - |E_i|)$ should be made, we make this recursive call only if E_i does not contain any protected edge, nor an edge incident to a root of F_2 whose sibling edge is protected. When we do make a recursive call $m\,(F_1', F_2 \div E_i, k - |E_i|)$, cutting each edge xp_x in E_i turns p_x into a degree-2 node that is suppressed in $F_2 \div E_i$. This effectively replaces the remaining two edges e_1 and e_2 incident to p_x with a single edge e in $F_2 \div E_i$. If at least one of e_1 and e_2 is protected in F_2, then e inherits this protection in $F_2 \div E_i$.

While it should be clear now how the edge protection affects the set of recursive calls an invocation can make, we have not discussed how edges become protected in the first place. Only Case ABC introduces protected edges. We also split it into two sub-cases, depending on whether the path from a to c in F_2 has $q = 2$ or $q \geqslant 3$ pendant subtree. This gives us the following four cases of the algorithm (see Figure 6.1). In the description of these cases, we use $F \diamond E$ to indicate that the edges in E should be labelled as protected in F. When we say "make a recursive call $m\,(F_1', F_2 \div E_i, k - |E_i|)$", this should from now on be read as "make this call unless E_i includes a protected edge".

Separate components (AC): If $a \nsim_{F_2} c$, then make two recursive calls $m\,(F_1', F_2 \div \{e_a\}, k - 1)$ and $m\,(F_1', F_2 \div \{e_c\}, k - 1)$ and return TRUE if and only if one of these recursive calls does.

One pendant subtree (B): If $a \sim_{F_2} c$ and the path from a to c in F_2 has exactly one pendant node b, then make one recursive call $m\,(F_1', F_2 \div \{e_b\}, k - 1)$ and return its answer.

Two pendant subtrees (A2BC): If $a \sim_{F_2} c$ and the path from a to c in F_2 has two pendant nodes b_1 and b_2, we distinguish whether or not e_c is protected in F_2. If e_c is not protected, then make three recursive calls $m\,(F_1', F_2 \div \{e_a\}, k - 1)$, $m\,(F_1', F_2 \div \{e_{b_1}, e_{b_2}\}, k - 2)$, and $m\,(F_1' \diamond \{e_a\}, F_2 \diamond \{e_a, e_{b_1}, e_{b_2}\} \div \{e_c\}, k - 1)$. If e_c is protected, then make two recursive calls $m\,(F_1', F_2 \div \{e_{b_1}, e_{b_2}\}, k - 2)$ and $m\,(F_1', F_2 \diamond \{e_{b_1}, e_{b_2}\} \div \{e_a\}, k - 1)$. In either case, return TRUE if and only if one of these recursive calls does.

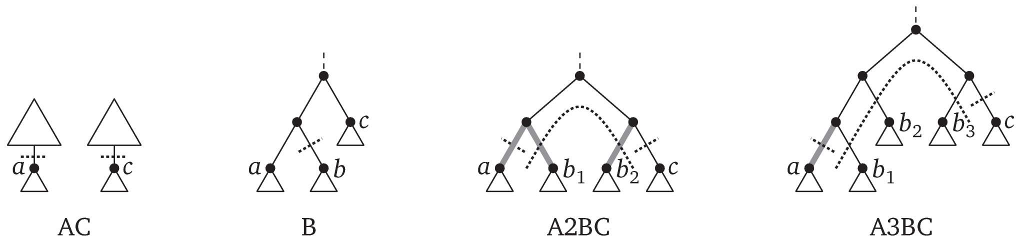


Figure 6.1: The first four cases of the MAF algorithm. Only F_2 is shown. Each dotted line indicates the set of edges cut in a recursive call. The fat grey edges in Cases A2BC and A3BC are the ones protected when cutting edge e_c.

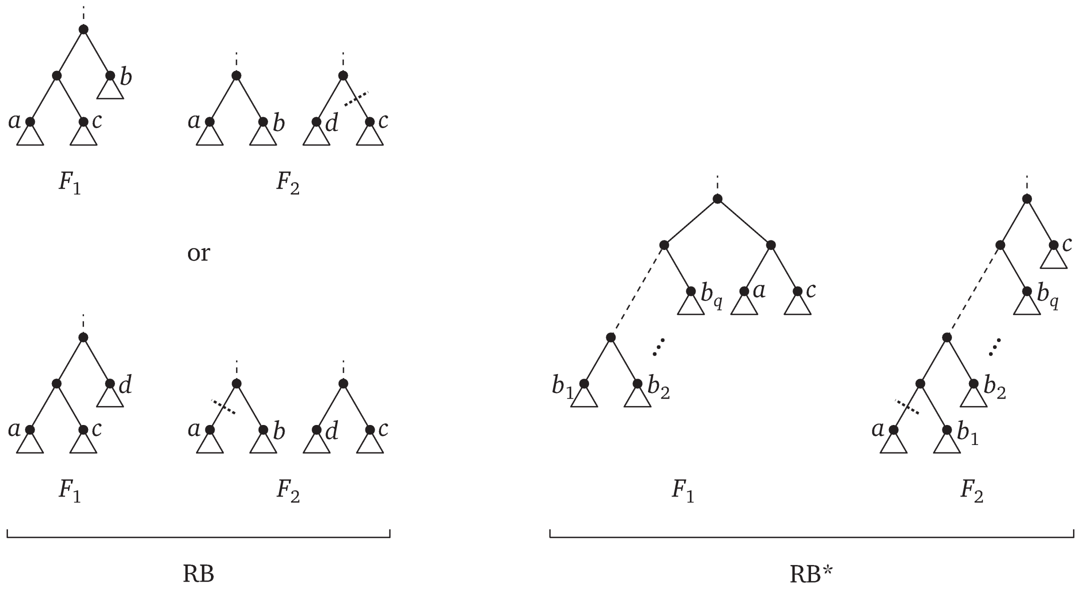


Figure 6.2: The two "reverse" cases of the MAF algorithm.

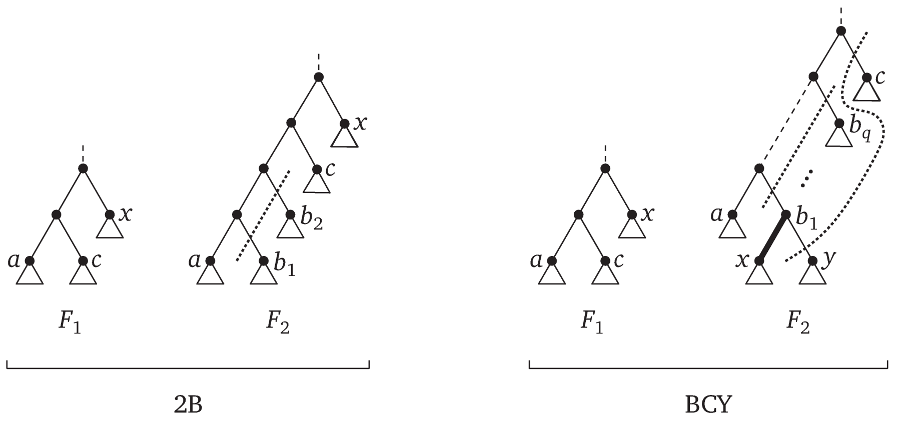


Figure 6.3: The last two cases of the MAF algorithm. The fat edge in Case BCY is protected.

Multiple pendant subtrees (A3BC): If $a \sim_{F_2} c$ and the path from a to c in F_2 has $q \geqslant 3$ pendant nodes $b_1, b_2, \ldots, b_q$, then make three recursive calls $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$, $m\left(F_1', F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\right)$, and $m\left(F_1' \diamond \{e_a\},\right.$ $\left.F_2 \diamond \{e_a\} \div \{e_c\}, k-1\right)$ and return TRUE if and only if one of these recursive calls does.

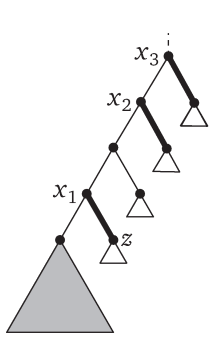
The effect of protecting edges is to reduce the number of recursive calls made in the different cases of our algorithm. This idea is nearly sufficient to achieve the desired running time of $\mathrm{O}\left(2^k n\right)$, except in a small number of special cases. We deal with these special cases by introducing additional branching rules that deal with these cases explicitly (see Figures 6.2 and 6.3).

Reverse Case B (RB): If a's sibling b in F_2 exists in F_1 and is a sibling of a and c's common parent in F_1, then make one recursive call $m\left(F_1', F_2 \div \{e_c\}, k-1\right)$ and return its answer. Otherwise, if c's sibling d in F_2 exists in F_1 and is a sibling of a and c's common parent in F_1, then make one recursive call $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$ and return its answer.

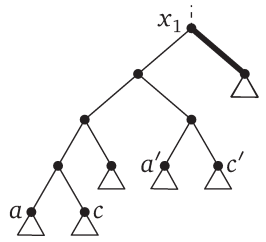
Chained Reverse Case B (RB*): This case applies only if $a \sim_{F_2} c$. Assume that c is a child of a and c's LCA in F_2 and, hence, that a is not. Let $b_1, b_2, \ldots, b_q$ be the pendant nodes of the path from a to c in F_2, numbered in their order of appearance along the path from a to c. If nodes $b_1, b_2, \ldots, b_q$ exist in F_1 and the sibling x of a and c's common parent in F_1 is the root of a subtree consisting of a path of nodes $x_1, x_2, \ldots, x_{q-1} = x$ such that b_1 and b_2 are the children of x_1 and, for $1 < i < q$, x_i is the parent of x_{i-1} and b_{i+1}, then make one recursive call $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$ and return its answer.

Two pendant subtrees and a common uncle (2B): If $a \sim_{F_2} c$, the path from a to c has exactly two pendant nodes b_1 and b_2, and the sibling x of a and c's common parent in F_1 exists in F_2 and is the sibling of the LCA of a and c in F_2, then make one recursive call $m\left(F_1', F_2 \div \{e_{b_1}, e_{b_2}\}, k-2\right)$ and return its answer.

Protected uncle (BCY): Assume $a \sim_{F_2} c$, the sibling x of a and c's commont parent in F_1 exists in F_2, and x's parent edge e_x is protected in F_2. If the path from a to c in F_2 has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$ and x is a



(a) Only z and nodes in the shaded subtree can be members of eligible sibling pairs. The fat edges are the ones that are protected.



(b) The depth rule prefers the sibling pair (a, c) over the sibling pair (a', c').

Figure 6.4: The depth rule.

child of b_1, make two recursive calls $m\left(F_1', F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q\right)$ and $m\left(F_1', F_2 \div \{e_c, e_y\}, k-2\right)$, where y is x's sibling in F_2.

6.3.3 The Complete Algorithm

The final ingredient needed for our new algorithm to achieve the desired running time of $\mathrm{O}\left(2^k n\right)$ is a rule to choose sibling pairs in each invocation based on their depths in F_1:

Depth rule: Given the set of sibling pairs defined by the nodes in R_t, an invocation always chooses the next sibling pair (a, c) based on two criteria:

(i) We will maintain the condition that the protected edges in $\dot{T}_1$, if any, are pendant to a single root-leaf path in $\dot{T}_1$ (see Figure 6.4(a)). Let $x_1, x_2, \ldots, x_q$ be the nodes on this path that are top endpoints of protected edges, sorted by decreasing distance from the root of $\dot{T}_1$. Another condition we maintain is that the bottom endpoints of all protected edges in $\dot{T}_1$ are in R_t. Only sibling pairs whose members are descendants of x_1 are eligible to be chosen in the current invocation.

(ii) The sibling pair (a, c) is chosen from the set of eligible sibling pairs so that there is no other eligible sibling pair (a', c') whose members have a greater distance from the root of $\dot{T}_1$ than a and c (see Figure 6.4(b)).

(iii) If there are two sibling pairs (a, c) and (a', c') with equal distance from the root of $\dot{T}_1$, we choose the one whose members are in different components

m **(F_1, F_2, k):**

1. (Failure) If $k < 0$, return FALSE.
2. (Success) If $|R_t| \leqslant 2$, return TRUE.
3. Choose a sibling pair (a, c) according to the depth rule.
4. If a and c are siblings in F_2, remove a and c from R_t and add their parent to R_t, return to Step 2.
5. If a is a root of F_2, cut edge e_a in F_1 and move a from R_t to R_d, return to Step 2.
6. If c is a root of F_2, cut edge e_c in F_1 and move c from R_t to R_d, return to Step 2.
7. Consider cases SC, B, RB, RB*, 2B, BCY, A2BC, and A3BC in order and choose the first case that applies. Make the recursive calls specified in this case and return TRUE if and one of them does; otherwise return FALSE.

Figure 6.5: The algorithm for deciding whether $e(T_1, T_2, F_2) \leqslant k$.

of $\dot{F}_2$ or whose deepest member has the greater depth in $\dot{F}_2$. Remaining ties are broken arbitrarily.

This gives us the algorithm in Figure 6.5. The order in which the different cases are considered in Step 7 is important because, for example, Case A2BC or A3BC applies also if Case 2B or BCY applies, but we need to give preference to the latter two cases if we want to guarantee a running time of $\mathrm{O}\left(2^k n\right)$.

6.3.4 Correctness

We split the correctness proof into two parts. First we argue that, if we ignore the protection of edges, our algorithm succeeds in finding an AF of size at most $k + 1$ if such an AF exists. Then we show that, even though edge protection leads us to ignore a number of search branches that would have been explored in the absence of edge protection, we still succeed in finding an AF of size at most $k + 1$.

The correctness of Cases AC, B, A2BC, and A3BC is established by Lemmas 6.6, 6.7, and 6.8 because, ignoring the protected edges they introduce, Cases A2BC and A3BC are special cases of Case ABC in Section 6.3.1. The correctness proof of Case RB is identical to that for Case B given in [101, 102] (Chapter 3) after exchanging the roles of F_1 and F_2. Thus, it remains to establish the correctness of Cases RB*, 2B, and BCY. The next lemmas cover these cases.

Lemma 6.9 (Collapsible chain)**.** *Let (a, c) be a non-trivial sibling pair of F_1, assume $a \sim_{F_2} c$, assume c is a child of the LCA of a and c in F_2, and let $b_1, b_2, \ldots, b_q$, $q \geqslant 2$, be the pendant nodes of the path from a to c, numbered by increasing distance from a. If nodes $b_1, b_2, \ldots, b_q$ exist in F_1 and the sibling x of a and c's common parent in F_1 is the root of a subtree consisting of a path of nodes $x_1, x_2, \ldots, x_{q-1} = x$ such that b_1 and b_2 are the children of x_1 and, for $1 < i < q$, x_i is the parent of x_{i-1} and b_{i+1}, then $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$.*

Proof. Nodes b_1 and b_2 exist in F_1 and are siblings in F_1, while the path from b_1 to b_2 in F_2 has a as a pendant node. Thus, Lemma 6.7 applied to the sibling pair (b_1, b_2) shows that $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$. □

Lemma 6.10 (Two pendant nodes and a common uncle)**.** *If (a, c) is a non-trivial sibling pair of F_1, $a \sim_{F_2} c$, the path from a to c in F_2 has two pendant nodes b_1 and b_2, and the sibling x of a and c's common parent in F_1 exists in F_2 and is the sibling of the LCA of a and c in F_2, then $e(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}\}) = e(T_1, T_2, F_2) - 2$.*

Proof. By Lemma 6.8, we have $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$, $e(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}\}) = e(T_1, T_2, F_2) - 2$ or $e(T_1, T_2, F_2 \div \{e_c\}) = e(T_1, T_2, F_2) - 1$. In the second case, the lemma holds, so assume we have the first or third case. Since a and c are interchangeable, we can assume the first case applies. After cutting edge e_a, (x, c) is a sibling pair of $F_1' := F_1 \div \{e_a\}$ such that the path from x to c in $F_2' := F_2 \div \{e_a\}$ has one or two pendant nodes, depending on whether or not b_1 and b_2 were both pendant nodes of the path from a to the LCA of a and c in F_2. Thus, by Lemmas 6.7 and 6.8, we have $e(T_1, T_2, F_2' \div \{e_c\}) = e(T_1, T_2, F_2') - 1$, $e\left(T_1, T_2, F_2' \div \{e_{b_1'}, e_{b_2'}, \ldots, e_{b_q'}\}\right) = e(T_1, T_2, F_2') - q$ or $e(T_1, T_2, F_2' \div \{e_x\}) = e(T_1, T_2, F_2') - 1$, where $b_1', b_2', \ldots, b_q'$ are the pendant nodes of the path from x to c in F_2'. In each case, we obtain an edge set $E \supseteq \{e_a\}$ such that $|E| \geqslant 2$, all edges in E belong to the subtree of F_2 with leaves a, b_1, b_2, c, and x, and such that $e(T_1, T_2, F_2 \div E) = e(T_1, T_2, F_2) - |E|$. In particular, there exists an edge set $E_0 \supseteq E$ of size $e(T_1, T_2, F_2)$ such that $F_2 \div E_0$ is an AF of T_1 and T_2. Let u be the LCA of a and c in F_2. We split E into three subsets

$E_1 := E \cap \{e_x, e_u\}$, $E_2 := E \cap \{e_{b_1}, e_{b_2}\}$, and $E_3 := E \setminus (E_1 \cup E_2)$. Note that $e_a \in E_3$, that is, $E_3 \neq \emptyset$. By Lemmas 6.4 and 6.5, $F_2 \div E_0'$ is an AF of T_1 and T_2, where $E_0' := E_0 \setminus E_3 \cup \{e_{b_1}, e_{b_2}\}$ if $|E_3| = 1$ and $E_0' := E_0 \setminus E_3 \cup \{e_{b_1}, e_{b_2}, e_u\}$ if $|E_3| \geqslant 2$. Now let $F_2'' := F_2 \div \{e_{b_1}, e_{b_2}\} \supseteq F_2 \div E_0'$, and let $E_1' := E_0' \cap \{e_x, e_u\}$. The sibling pair (x, u) exists in F_1 and F_2''. Thus, by Lemma 6.5, $F_2 \div E_0''$ is an AF of T_1 and T_2, where $E_0'' := E_0' \setminus E_1'$ if $|E_1'| \leqslant 1$ and $E_0'' := E_0' \setminus E_1' \cup \{e_{p_x}\}$ if $|E_1'| \geqslant 2$. We distinguish two cases to bound the size of E_0''.

If $|E_3| = 1$, then $|E_0'| = |E_0| + 1 - |E_2|$ and $E_1' = E_1$. The latter implies that $|E_0''| = |E_0'| - |E_1|$ if $|E_1| = 1$ or $|E_0''| = |E_0'| - |E_1| + 1$ if $|E_1| > 1$. Since $|E| \geqslant 2$ and $|E_3| = 1$, we have $|E_1| + |E_2| \geqslant 1$, and we obtain $|E_0''| = |E_0|$ if $|E_1| + |E_2| = 1$, and $|E_0''| \leqslant |E_0| + 2 - (|E_1| + |E_2|) \leqslant |E_0|$ if $|E_2| + |E_2| > 1$.

If $|E_3| > 1$, then $|E_0'| = |E_0| + 3 - (|E_2| + |E_3|) \leqslant |E_0| + 1$ and $|E_1'| \geqslant 1$. Hence, $|E_0''| \leqslant |E_0'| - 1 \leqslant |E_0|$.

Since we obtain that $|E_0''| \leqslant |E_0| = e(T_1, T_2, F_2)$ in each case, and $\{e_{b_1}, e_{b_2}\} \subseteq E_0''$, this proves that $e(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}\}) = e(T_1, T_2, F_2) - 2$. $\square$

Lemma 6.11 (Protected uncle). *Let (a, c) be a non-trivial sibling pair of F_1, let x be the sibling of a and c's common parent in F_1, and assume $a \sim_{F_2} c$. Assume further that x exists in F_2, that the path from a to c in F_2 has $q \geqslant 2$ pendant nodes $b_1, b_2, \ldots, b_q$, and that x is a child of b_1. Then $e(T_1, T_2, F_2 \div \{e_x\}) = e(T_1, T_2, F_2) - 1$, $e\left(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}\right) = e(T_1, T_2, F_2) - q$ or $e(T_1, T_2, F_2 \div \{e_c, e_y\}) = e(T_1, T_2, F_2) - 2$, where y is x's sibling in F_2.*

Note that Case BCY never makes the recursive call corresponding to the case when $e(T_1, T_2, F_2 \div \{e_x\}) = e(T_1, T_2, F_2) - 1$ in Lemma 6.11 because edge e_x is always protected in this case.

Proof. By Lemma 6.8, we have $e(T_1, T_2, F_2 \div \{e_a\}) = e(T_1, T_2, F_2) - 1$, $e(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}) = e(T_1, T_2, F_2) - q$ or $e(T_1, T_2, F_2 \div \{e_c\}) = e(T_1, T_2, F_2) - 1$. In the second case, the lemma holds. In the third case, observe that (a, x) is a sibling pair of $F_1' := F_1 \div \{e_c\}$ such that the path from a to x in $F_2' := F_2 \div \{e_c\}$ has y as its single pendant node. Thus, by Lemma 6.7, we have $e(T_1, T_2, F_2' \div \{e_y\}) = e(T_1, T_2, F_2') - 1$, that is, $e(T_1, T_2, F_2 \div \{e_c, e_y\}) = e(T_1, T_2, F_2) - 2$. It remains to consider the first case. $F_1'' := F_1 \div \{e_a\}$ has (c, x) as a sibling pair and the path from c to x in $F_2'' := F_2 \div \{e_a\}$ has pendant nodes $y, b_2, b_3, \ldots, b_q$. By Lemma 6.8, we have $e(T_1, T_2, F_2'' \div \{e_c\}) = e(T_1, T_2, F_2'') - 1$, $e\left(T_1, T_2, F_2'' \div \{e_y, e_{b_2}, e_{b_3}, \ldots, e_{b_q}\}\right) = e(T_1, T_2, F_2'') - q$ or $e(T_1, T_2, F_2'' \div \{e_x\}) = e(T_1, T_2, F_2'') - 1$. In the first case, we also have $e(T_1, T_2, F_2 \div \{e_c\}) = e(T_1, T_2, F_2) - 1$ because $F_2'' \subseteq F_2$. As we argued above, this implies that $e(T_1, T_2, F_2 \div \{e_c, e_y\}) = e(T_1, T_2, F_2) - 2$. In the third case, we obtain $e(T_1, T_2, F_2 \div \{e_x\}) = e(T_1, T_2, F_2) - 1$ because $F_2'' \subseteq F_2$. Finally, in the second case, we obtain that $e(T_1, T_2, F_2'' \div \{e_y\}) = e(T_1, T_2, F_2'') - 1$ and, hence, $e(T_1, T_2, F_2 \div \{e_y\}) = e(T_1, T_2, F_2) - 1$. Case RB applies to the sibling pair (a, c) after cutting edge e_y in F_2, with (a, x) being a sibling pair in $F_2''' := F_2 \div \{e_y\}$ and c being the only pendant node of the path from a to x in F_1. Thus, $e(T_1, T_2, F_2''' \div \{e_c\}) = e(T_1, T_2, F_2''') - 1$ and, hence, $e(T_1, T_2, F_2 \div \{e_c, e_y\}) = e(T_1, T_2, F_2) - 2$. □

Using Lemmas 6.6–6.11, we can now prove the correctness of our algorithm.

Theorem 6.12. *The invocation* $m(T_1, T_2, k_0)$ *returns* True *if and only if* $e(T_1, T_2, T_2) \leqslant k_0$.

Proof. First consider a variant of our algorithm where Cases A2BC and A3BC do not protect any edges and Case BCY includes an additional recursive call $m(F_1', F_2 \div \{e_x\}, k - 1)$. Consider the recursion tree $\mathcal{T}$ defined by the set of recursive calls made by this algorithm. Each leaf of this tree corresponds to an invocation that does not make any further recursive calls and thus returns False in Step 1 or True in Step 2. The algorithm returns True if and only if at least one of these leaf invocations returns True.

First we prove that $e(T_1, T_2, T_2) \leqslant k_0$ if our algorithm returns True. Since each recursive call $m(F_1', F_2 \div E, k')$ made by an invocation $m(F_1, F_2, k)$ satisfies $k' = k - |E|$, it is easily verified that each leaf invocation $m(F_1, F_2, k)$ satisfies $|F_2| \leqslant k_0 - k + 1$. If our algorithm returns True, then there exists such a leaf invocation that returns True, which implies that $k \geqslant 0$ and F_2 is an AF of T_1 and T_2. Thus, $|F_2| \geqslant e(T_1, T_2, T_2) + 1$, that is, $e(T_1, T_2, T_2) \leqslant k_0 - k \leqslant k_0$.

Next assume $e(T_1, T_2, T_2) \leqslant k_0$. We call an invocation $m(F_1, F_2, k)$ *promising* if $|F_2| + e(T_1, T_2, F_2) = 1 + e(T_1, T_2, T_2)$ and $k_0 - k = |F_2| - 1$. By Lemmas 6.6–6.11, every promising invocation $m(F_1, F_2, k)$ has a promising child invocation $m(F_1', F_2', k')$. Since the root invocation $e(T_1, T_2, T_2)$ is itself promising, this shows that $\mathcal{T}$ has at least one promising leaf invocation $\mathcal{I} = m(F_1, F_2, k)$. Since this is a leaf invocation, we either have $k < 0$ or F_2 is an AF of T_1 and T_2. However, $k = k_0 - |F_2| + 1 = k_0 - e(T_1, T_2, T_2) + e(T_1, T_2, F_2) \geqslant 0$ because $k_0 \geqslant e(T_1, T_2, T_2)$ and $e(T_1, T_2, F_2) \geqslant 0$. Thus, $k \geqslant 0$ and F_2 is an AF of T_1 and T_2, that is, invocation $\mathcal{I}$, and hence the whole algorithm, returns True.

It remains to consider the effect of edge protection. First observe that the introduction of protected edges can only reduce the number of recursive calls made by each invocation. Thus, if the algorithm returns False without using edge protection, it also returns False using edge protection, that is, our algorithm gives the correct answer if $e(T_1, T_2, T_2) > k_0$. For the remainder of the proof assume $e(T_1, T_2, T_2) \leqslant k_0$. Consider the subtree $\mathcal{T}'$ of $\mathcal{T}$ defined by promising invocations. Our argument in the previous paragraph implies that every leaf invocation of $\mathcal{T}'$ is a leaf invocation of $\mathcal{T}$ that returns True. Consider the leftmost root-leaf path $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_h$ in $\mathcal{T}'$, where the children of each invocation $\mathcal{I}$ in $\mathcal{T}'$ are ordered left to right in the order invocation $\mathcal{I}$ makes these recursive calls. We prove that the introduction of protected edges cannot prevent our algorithm from exploring this path, which implies that our algorithm returns True if $e(T_1, T_2, T_2) \leqslant k_0$ even using edge protection.

Assume the contrary. Then there exists an index $1 \leqslant j' < h$ such that our algorithm makes invocations $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{j'}$ but does not make invocations $\mathcal{I}_{j'+1}, \mathcal{I}_{j'+2}, \ldots, \mathcal{I}_h$. Then we have $\mathcal{I}_{j'} = m(F_1'', F_2'', k'')$ and $\mathcal{I}_{j'+1} = m(F_1''', F_2'' \div E'', k'' - |E''|)$, for some edge set E'' that includes a protected edge of F_2'' or an edge incident to a root of F_2'' whose sibling edge is protected. In the former case, let e be a protected edge in E''; in the latter case, let e be the protected sibling edge of an edge in E'' incident to a root of F_2''. Since both invocations $\mathcal{I}_{j'}$ and $\mathcal{I}_{j'+1}$ are promising, we have $e(T_1, T_2, F_2'' \div E'') = e(T_1, T_2, F_2'') - |E''|$, which implies that $e(T_1, T_2, F_2'' \div \{e\}) = e(T_1, T_2, F_2'') - 1$ because either $e \in E''$ or e's sibling is in E'' and both are incident to a root in F_2'', in which case cutting e or its sibling produces the same forest. Since T_2 does not contain any protected edges, there exists an index $1 \leqslant j < j'$ such that the invocations $\mathcal{I}_j = m(F_1, F_2, k)$ and $\mathcal{I}_{j+1} = m(F_1', F_2', k')$ have the property that edge e is protected in F_2' but not in F_2. Therefore, invocation $\mathcal{I}_j$ applies Case A2BC or A3BC and $\mathcal{I}_{j+1} = m(F_1', F_2 \diamond \{e_a, e_{b_1}, e_{b_2}\} \div \{e_c\}, k-1)$ or $\mathcal{I}_{j+1} = m(F_1', F_2 \diamond \{e_a\} \div \{e_c\}, k-1)$. In particular, $e \in \{e_a, e_{b_1}, e_{b_2}\}$. If invocation $\mathcal{I}_j$ applies

Case A3BC, then $e = e_a$, which implies that the invocation $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$ is also promising because F_2'' is a forest of F_2 and, hence $e\left(T_1, T_2, F_2'' \div \{e\}\right) = e\left(T_1, T_2, F_2''\right) - 1$ implies that $e\left(T_1, T_2, F_2 \div \{e\}\right) = e\left(T_1, T_2, F_2\right) - 1$. This contradicts the assumption that $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_h$ is the leftmost root-leaf path in $\mathcal{T}'$. So assume invocation $\mathcal{I}_j$ applies Case A2BC. Since (a, c) is a sibling pair of F_1' and the path from a to c in F_2 has b_1 and b_2 as pendant nodes, Lemma 6.8 shows that $e\left(T_1, T_2, F_2 \div \{e_a\}\right) = e\left(T_1, T_2, F_2\right) - 1$, $e\left(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}\}\right) = e\left(T_1, T_2, F_2\right) - 2$ or $e\left(T_1, T_2, F_2 \div \{e_c\}\right) = e\left(T_1, T_2, F_2\right) - 1$. In the first two cases, we obtain a contradiction because this would imply that invocation $m\left(F_1', F_2 \div \{e_a\}, k-1\right)$ or $m\left(F_1', F_2 \div \{e_{b_1}, e_{b_2}\}, k-2\right)$ is promising, contradicting that $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_h$ is the leftmost root-leaf path in $\mathcal{T}'$. In the last case, let E be the largest subset of $\{e_a, e_{b_1}, e_{b_2}, e_c\}$ such that $e_c \in E$ and $e\left(T_1, T_2, F_2 \div E\right) = e\left(T_1, T_2, F_2\right) - |E|$. We have $e_a \notin E$ and $\{e_{b_1}, e_{b_2}\} \nsubseteq E$. If $E = \{e_c\}$, we arrive at a contradiction because in this case $e\left(T_1, T_2, F_2 \div \{e_c, e'\}\right) = e\left(T_1, T_2, F_2 \div \{e_c\}\right)$, for all $e' \in \{e_a, e_{b_1}, e_{b_2}\}$, but $e \in \{e_a, e_{b_1}, e_{b_2}\}$ and $e\left(T_1, T_2, F_2'' \div \{e\}\right) = e\left(T_1, T_2, F_2''\right) - 1$ implies that $e\left(T_1, T_2, F_2 \div \{e_c, e\}\right) = e\left(T_1, T_2, F_2 \div \{e_c\}\right) - 1$. Thus, since $e_a \notin E$ and $\{e_{b_1}, e_{b_2}\} \nsubseteq E$, we have $E = \{e_{b_i}, e_c\}$, for some $i \in \{1, 2\}$. Let $E_0 \supseteq E$ be an edge set of size $e\left(T_1, T_2, F_2\right)$ such that $F_2 \div E_0$ is an AF of T_1 and T_2. By Lemmas 6.4 and 6.5, $F_2 \div E_0'$ is also an AF of T_1 and T_2, where $E_0' := E_0 \setminus \{e_c\} \cup \{e_{b_1}, e_{b_2}\}$. Since $E_0 \cap \{e_{b_1}, e_{b_2}\} \neq \emptyset$, we have $|E_0'| \leqslant |E_0| = e\left(T_1, T_2, F_2\right)$. Since $\{e_{b_1}, e_{b_2}\} \subseteq E_0'$, this implies that $e\left(T_1, T_2, F_2 \div \{e_{b_1}, e_{b_2}\}\right) = e\left(T_1, T_2, F_2\right) - 2$, that is, the invocation $m\left(F_1', F_2 \div \{e_{b_1}, e_{b_2}\}, k-2\right)$ is once again promising, a contradiction. □

6.3.5 Analysis

In this section, we prove the following theorem, which, together with Theorem 6.12 proves Theorem 6.1.

Theorem 6.13. *The invocation $m\left(T_1, T_2, k_0\right)$ takes $\mathrm{O}\left(2^{k_0} n\right)$ time.*

The running time per invocation of the algorithm, excluding the recursive calls it makes, is easily shown to be $\mathrm{O}(n)$ using a straightforward adaptation of the arguments in [101,102] to encompass the new cases. Thus, it suffices to prove that the invocation $m\left(T_1, T_2, k_0\right)$ makes $\mathrm{O}\left(2^{k_0}\right)$ recursive calls.

To prove this, we augment the algorithm so that each invocation $m(F_1, F_2, k)$ takes two additional arguments P and Q. P is a set of up to two nodes whose parent edges are protected in F_1; Q is a subset of P. Thus, this invocation becomes $m(F_1, F_2, k, P, Q)$. The two additional arguments are used only in the analysis, not by the algorithm, and are defined recursively. For the top-level invocation, we have $P = Q = \varnothing$. The sets passed to any other invocation are chosen by its parent invocation using the rules we specify below. For most invocations, $P = Q = \varnothing$.

Let $I(k, P, Q)$ be the number of descendant invocations of an invocation $\mathcal{I} = m(F_1, F_2, k, P, Q)$. We prove that $I(k, P, Q) \leqslant \alpha(2^{k-|P|} + |Q|2^{k-|P|-1}) + |P| - 3$, where $\alpha \geqslant 0$ is an appropriate constant. This shows that the total number of recursive calls of our algorithm is $I(k_0, \varnothing, \varnothing) \leqslant \alpha 2^{k_0} - 3 = \mathrm{O}\left(2^{k_0}\right)$. The proof is by induction on k.

Base case. For $-3 \leqslant k < 0$, an invocation $m(F_1, F_2, k, P, Q)$ has $\mathrm{O}(1)$ descendant invocations, so we can find an appropriate $\alpha \geqslant 0$ such that $I(k, P, Q) \leqslant \alpha 2^{k-2} - 3$ in this case.[1]

Inductive step. For the inductive step, assume $k \geqslant 0$ and the bound on $I(k', P)$ holds for all $k' < k$ and $P = \varnothing$. Consider an invocation $\mathcal{I} = m(F_1, F_2, k, \varnothing, \varnothing)$. If $\mathcal{I}$ applies Case AC, B, RB, RB*, 2B or BCY, we set $P = Q = \varnothing$ in each of its child invocations. Since $\mathcal{I}$ makes at most two recursive calls, each with parameter at most $k-1$, the inductive hypothesis implies that $I(k, \varnothing, \varnothing) \leqslant 1 + 2I(k-1, \varnothing, \varnothing) \leqslant 1 + 2(\alpha 2^{k-1} - 3) < \alpha 2^k - 3$.

If $\mathcal{I}$ applies Case A2BC or A3BC to a sibling pair (a, c), consider the subtree of F_1 rooted in a's great-grandparent r. By the depth rule, this tree is a subtree of the tree shown in Figure 6.6. We need to consider only cases where a has a great-grandparent because otherwise F_1 has constant size and $\mathcal{I}$ has $\mathrm{O}(1)$ descendant invocations; for an appropriate choice of α and $k \geqslant -3$, this is bounded by $\alpha 2^k - 3$.

We can divide the descendant invocations of $\mathcal{I}$ into six groups:

G1. Invocation $\mathcal{I}$ itself.

G2. Invocations that consider sibling pairs in F_1^d.

G3. Invocations that consider the sibling pair (a, d).

[1] We only need to consider $k \geqslant -3$ because the inductive step is applied only for $k \geqslant 0$, and we only consider recursive calls with parameter no less than $k - 3$ in the inductive step.

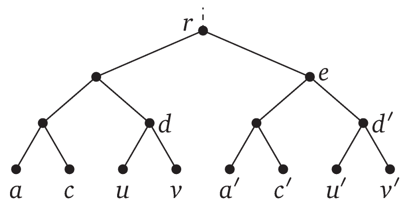


Figure 6.6: The subtree of F_1 relevant to the analysis of an invocation $m\,(F_1, F_2, k, \varnothing)$ that applies Case A2BC or A3BC.

G4. Invocations that consider sibling pairs in F_1^e.

G5. Invocations that consider the sibling pair (a, e).

G6. Invocations that consider sibling pairs outside of F_1^r.

We prove that our bound on $I(k, P, Q)$ holds for the invocations in each group if it holds for all invocations in subsequent groups and for invocations with parameter at most $k - 1$ and $P = Q = \varnothing$. The latter is true by induction, so we only need to argue specifically about invocations in these six groups that satisfy $P \neq \varnothing$.

G1: Invocation $\mathcal{I}$. Invocation $\mathcal{I}$ sets $P = Q = \varnothing$ in its A- and B-branches, and $P = \{a\}$ in its C-branch. If $\mathcal{I}$ applies Case A2BC, we set $Q = \varnothing$ in the C-branch, which gives $I(k-1, \{a\}, \varnothing) \leqslant \alpha 2^{k-2} - 2$ for the number of descendant invocations of $\mathcal{I}'$ and, hence, $I(k, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k-1} - 3 + \alpha 2^{k-2} - 3 + \alpha 2^{k-2} - 2 < \alpha 2^k - 3$ for the number of descendant invocations of $\mathcal{I}$. If $\mathcal{I}$ applies Case A3BC, we set $Q = P$ in the C-branch, which gives $I(k-1, \{a\}, \{a\}) \leqslant \alpha(2^{k-2} + 2^{k-3}) - 2$ for the number of descendant invocations of $\mathcal{I}'$ and, hence, $\mathcal{I}(k, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k-1} - 3 + \alpha 2^{k-3} - 3 + \alpha(2^{k-2} + 2^{k-3}) - 2 < \alpha 2^k - 3$ for the number of descendant invocations of $\mathcal{I}$.

G2: Sibling pairs in $\boldsymbol{F_1^d}$. Let $\mathcal{I}' = m\,(F_1', F_2', k', \{a\}, Q')$ be a descendant invocation of $\mathcal{I}$ that considers a sibling pair in F_1^d. Then $\mathcal{I}'$ is the root of $\mathcal{I}$'s C-branch and considers the sibling pair (u, v). If u and v are siblings in F_2', invocation $\mathcal{I}'$ merges the sibling pair (u, v) and then considers the sibling pair (a, d), so we can apply the argument for invocations in group G3. Now assume u and v are not siblings in F_2'. If $\mathcal{I}'$ makes only one recursive call, which includes an application of Case B, RB, RB* or 2B, we set $P = Q = \varnothing$ in this recursive call and, thus, obtain $I(k', \{a\}, Q') \leqslant 1 + I(k' - 1, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-1} - 3 \leqslant \alpha(2^{k'-1} + |Q'| 2^{k'-2}) - 2$ for the number of descendant invocations of $\mathcal{I}'$. If $\mathcal{I}'$ applies Case BCY and makes two recursive calls, with parameter at most $k'-2$, we set $P = Q = \varnothing$ in both recursive calls and, thus, obtain $I(k', \{a\}, Q') \leqslant 1+2I(k'-2, \varnothing, \varnothing) \leqslant 1+2(\alpha 2^{k'-2}-3) < \alpha(2^{k'-1}+|Q'|2^{k'-2})-2$ for the number of descendant invocations of $\mathcal{I}'$.

If $\mathcal{I}'$ makes two recursive calls using Case AC, A2BC or A3BC, we set $P = P'$ and $Q = Q'$ in its two child invocations. In these two child invocations, d merges with one or both of its children. In both cases, we denote the merged node as d, and both child invocations consider the sibling pair (a, d), that is, belong to group G3. Thus, the number of descendant invocations of $\mathcal{I}'$ is given by $I(k', \{a\}, Q') \leqslant 1 + 2I(k' - 1, \{a\}, Q') \leqslant 1 + 2(\alpha(2^{k'-2} + |Q'|2^{k'-3}) - 2) < \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$.

If $\mathcal{I}'$ applies Case A2BC or A3BC and makes three recursive calls, we set $P = P'$ and $Q = Q'$ in the two child invocations that cut e_u and the pendant edges of the path from u to v in F_2', respectively. For the last branch, which cuts e_v and protects edge u, we set $P = P' \cup \{u\}$ and either $Q = Q'$, if $\mathcal{I}'$ applies Case A2BC, or $Q = Q' \cup \{v\}$, if $\mathcal{I}'$ applies Case A3BC. Note that, once again, all three child invocations consider the sibling pair (a, d) next and, thus, belong to group G3. Thus, if $\mathcal{I}'$ applies Case A2BC, the number of descendant invocations of $\mathcal{I}'$ is given by $I(k', \{a\}, Q) \leqslant 1+I(k'-1, \{a\}, Q)+I(k'-2, \{a\}, Q)+I(k'-1, \{a, u\}, Q) \leqslant 1+\alpha(2^{k'-2}+|Q|2^{k'-3})-2+\alpha(2^{k'-3}+|Q|2^{k'-4})-2+\alpha(2^{k'-3}+|Q|2^{k'-4})-1 < \alpha(2^{k'-1}+|Q|2^{k'-2})-2$. If $\mathcal{I}'$ applies Case A3BC, we obtain $I(k', \{a\}, Q) \leqslant 1+I(k'-1, \{a\}, Q)+I(k'-3, \{a\}, Q)+I(k' - 1, \{a, u\}, Q \cup \{u\}) \leqslant 1 + \alpha(2^{k'-2} + |Q|2^{k'-3}) - 2 + \alpha(2^{k'-4} + |Q|2^{k'-5}) - 2 + \alpha(2^{k'-3} + (|Q| + 1)2^{k'-4}) - 1 < \alpha(2^{k'-1} + |Q|2^{k'-2}) - 2$.

G3: The sibling pair (a, d). Let $\mathcal{I}' = m\,(F_1', F_2', k', P, Q)$ be a descendant invocation of $\mathcal{I}$ that considers the sibling pair (a, d) with $P \neq \varnothing$. We argue first that a and d are not siblings in F_2'.

If $\mathcal{I}'$ is a child of $\mathcal{I}$, then it is the root of $\mathcal{I}$'s C-branch. Thus, if a and d were siblings in F_2', d would be a or c's sibling in F_2. This, however, would imply that $\mathcal{I}$ applies Case RB instead of Case A2BC or A3BC.

If $\mathcal{I}'$ is not a child of $\mathcal{I}$, it is a grandchild of $\mathcal{I}$ and its parent $\mathcal{I}'' = m\,(F_1'', F_2'', k'', P', Q')$ applies Case AC, A2BC or A3BC. Thus, to obtain F_2' from F_2'', we cut either e_u, e_v or all the pendant edges of the path from u to v in F_2''. If cutting e_u makes a and $d = v$ siblings, then a is a sibling of u or v in F_2'', which implies that $\mathcal{I}''$ would have applied Case RB. We obtain a similar contradiction if cutting e_v makes a and $d = u$ siblings. Finally, if cutting the pendant edges of the path from u to v makes a and d siblings, then a must be the sibling of the LCA of u and v in F_2'. Thus, u is deeper in F_2' than a and is also deeper in F_2 than a. Since a is deeper in F_2 than c, this implies that u is deeper in F_2 than both a and c. Since a, c, u, and v all have the same depth in F_1, this contradicts that we chose the sibling pair (a, c) over the sibling pair (u, v) in invocation $\mathcal{I}$.

We have shown that a and d are not siblings in F_2'. Hence, invocation $\mathcal{I}'$ branches on the sibling pair (a, d). We distinguish whether $P' = \{a\}$ or $P' = \{a, u\}$. The latter happens in the C-branch of $\mathcal{I}''$ if $\mathcal{I}''$ applies Case A2BC or A3BC.

First assume $P' = \{a\}$. If $\mathcal{I}'$ applies Case AC, B, RB, RB*, 2B or BCY, the protection of edge e_a ensures that $\mathcal{I}'$ makes at most one recursive call, with parameter at most $k' - 1$. We set $P = Q = \varnothing$ in this recursive call and, thus, obtain $I(k', \{a\}, Q) \leqslant 1 + I(k' - 1, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-1} - 3 \leqslant \alpha(2^{k'-1} + |Q| 2^{k'-2}) - 2$ for the number of descendant invocations of $\mathcal{I}'$. We obtain a similar analysis with $P = Q = \varnothing$ if $\mathcal{I}'$ applies Case A2BC or A3BC but makes only one recursive call due to protected edges.

If $\mathcal{I}'$ applies Case A2BC or A3BC and makes two recursive calls corresponding to cutting e_d and cutting the pendant edges of the path from a to d in F_2', respectively, we distinguish whether $\mathcal{I}$ applied Case A2BC or A3BC. If $\mathcal{I}$ applied Case A3BC, we set $P = Q = \varnothing$ in both of these recursive calls. This gives $I(k', \{a\}, \{a\}) \leqslant 1 + I(k'-1, \varnothing, \varnothing) + I(k'-2, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-1} - 3 + \alpha 2^{k'-2} - 3 < \alpha(2^{k'-1} + 2^{k'-2}) - 3$ for the number of descendant invocations of $\mathcal{I}'$. If $\mathcal{I}$ applied Case A2BC, we set $P = P' = \{a\}$ in the recursive call that cuts e_d and $P = Q = \varnothing$ in the other recursive call; we set $Q = \varnothing$ in the recursive call that cuts edge e_d if $\mathcal{I}'$ applies Case A2BC, and $Q = \{a\}$ otherwise. This gives $I(k', \{a\}, \varnothing) \leqslant 1 + I(k'-1, \{a\}, \varnothing) + I(k'-2, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-2} - 2 + \alpha 2^{k'-2} - 3 < \alpha 2^{k'-1} - 3$ or $I(k', \{a\}, \varnothing) \leqslant 1 + I(k'-1, \{a\}, \{a\}) + I(k'-3, \varnothing, \varnothing) \leqslant 1 + \alpha(2^{k'-2} + 2^{k'-3}) - 2 + \alpha 2^{k'-3} - 3 < \alpha 2^{k'-1} - 3$ for the number of descendant invocations of $\mathcal{I}'$.

Finally consider the case when $P' = \{a, u\}$. Since $d = u$ in this case, this means that both edges e_a and e_d are protected. Thus, if this invocation makes any recursive calls at all, it must cut all pendant edges of the path from a to d in F_2'. We prove that this path has at least two pendant edges. This allows us to set $P = Q = \varnothing$ in this recursive call, which gives $I(k', P', Q') \leqslant 1 + I(k' - 2, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-2} - 3 < \alpha(2^{k'-2} + |Q'|2^{k'-3}) - 1$ for the number of descendant invocations of $\mathcal{I}'$.

Consider the parent invocation $\mathcal{I}'' = m\left(F_1'', F_2'', k'', P'', Q''\right)$ of $\mathcal{I}'$. Since invocation $\mathcal{I}''$ applies Case A2BC or A3BC in this case, the path from u to v in F_2'' has at least two pendant nodes. Let x be the LCA of u and v in F_2''. We observe that the path from u to x has at least one pendant node because otherwise v would have greater depth in F_2'' than u, contradicting how we order the members of each sibling pair we consider. If a is not in $(F_2'')^x$, all pendant nodes of the path from u to x in F_2'' become pendant nodes of the path from u to a in F_2', while the pendant nodes of the path from v to x, if any, merge to become a single pendant node of the path from u to a. Since the path from u to x has at least one pendant node and the path from u to v has at least two pendant nodes, this implies that the path from a to u in F_2' has at least two pendant nodes. If a belongs to the subtree of some pendant node b of the path from u to v, we distinguish two cases. First observe that $b \neq a$ in this case because we set $P' = \{a, u\}$ in invocation $\mathcal{I}'$ only if a is not a pendant node of the path from u to v in F_2''. Thus, if b is not u's sibling, the path from a to u in F_2'' has at least two pendant nodes other than v. These nodes are also pendant nodes of the path from a to u in F_2'. If b is u's sibling and a is b's child, invocation $\mathcal{I}''$ would have applied Case BCY. Thus, a has distance at least two from b, which implies that the path from a to u in F_2'' has at least two pendant nodes. Cutting e_v does not affect the subtree below p_u, so these nodes are also pendant nodes of the path from a to u in F_2'.

G4: Sibling pairs in $\boldsymbol{F_2^e}$. Consider a descendant invocation $\mathcal{I}' = m\left(F_1', F_2', k', P', Q'\right)$ of $\mathcal{I}$ such that all descendant invocations of $\mathcal{I}$ that are proper ancestors of $\mathcal{I}'$ belong to groups G1–G3. We prove that, if $P' \neq \varnothing$, a and e belong to the same connected component in F_2'. Thus, we do indeed consider at least one sibling pair in group G4 or G5 before moving on to group G6. Note that $P' \neq \varnothing$ only if this is true for every ancestor invocation of $\mathcal{I}'$ that is a proper descendant of $\mathcal{I}$ and only if invocation $\mathcal{I}$ applied Case A2BC. For $j \in \{1, 2, 3\}$, let $\mathcal{I}_j = m\left(F_1^{(j)}, F_2^{(j)}, k^{(j)}, P^{(j)}, Q^{(j)}\right)$ be the ancestor invocation of $\mathcal{I}'$ in group Gj.

$\mathcal{I}_1 = \mathcal{I}$. Thus, only the branch that cuts edge e_c sets $P \neq \varnothing$, and cutting this edge clearly cannot separate a and e.

$\mathcal{I}_2$ branches on the sibling pair (u, v). We set $P \neq \varnothing$ in a child invocation only if we apply Case AC, A2BC or A3BC. A branch that cuts edge e_u or e_v once again cannot separate a and e. Cutting the pendant edges of the path from u to v in F_2, however, could cut e_e because e may be a pendant node of this path. To have $P' \neq \varnothing$ in $\mathcal{I}'$, we require that $a \sim_{F_2^{(3)}} d$ and that none of the pendant edges of the path from a to d in $F_2^{(3)}$ are protected because otherwise $\mathcal{I}_3$ makes at most one recursive call, and we set $P = \varnothing$ in this recursive call. Since invocation $\mathcal{I}$ applies Case A2BC, it protects the edge e_{b_1}, where b_1 is a's sibling in $F_2^{(1)}$. Thus, for none of the pendant edges of the path from a to d in $F_2^{(3)}$ to be protected, it is necessary that d is a descendant of b_1 in $F_2^{(1)}$, and the same must therefore also be true for u and v. This, however, implies that u is deeper than a and c in $F_2^{(1)}$ and has the same depth as a and c in $F_1^{(1)}$. Thus, we obtain a contradiction to the depth rule because we chose the sibling pair (a, c) over (u, v) in invocation $\mathcal{I}$.

$\mathcal{I}_3$, finally, branches on the sibling pair (a, d) and sets $P \neq \varnothing$ only in the branch that cuts edge e_d, which once again cannot separate a and e. Thus, $a \sim_{F_2'} e$, as claimed.

Now we divide the invocations in group G4 into three categories. If an invocation makes at most one recursive call, which includes an application of Case B, RB, or RB*, we set $P = Q = \varnothing$ in this child invocation and obtain $I(k', \{a\}, Q') \leqslant 1 + I(k' - 1, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-1} - 3 \leqslant \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$. Similarly, if the invocation applies Case BCY and makes two invocations, we set $P = Q = \varnothing$ in both invocations, which gives $I(k', \{a\}, Q') \leqslant 1 + 2I(k' - 2, \varnothing, \varnothing) \leqslant 1 + 2(\alpha 2^{k'-2} - 3) < \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$.

If an invocation makes two recursive calls but does not apply Case BCY, which includes an application of Case AC, we set $P = P'$ and $Q = Q'$ in its two child invocations, which gives $I(k', \{a\}, Q') \leqslant 1 + 2I(k' - 1, \{a\}, Q') \leqslant 1 + 2(\alpha(2^{k'-2} + |Q'|2^{k'-3}) - 2) < \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$.

Finally, if the invocation makes three recursive calls, it applies Case A2BC or A3BC. This invocation $\mathcal{I}' = m(F_1', F_2', k', P', Q')$ and all its descendant invocations operate on a subtree of F_2^e which, due to the depth rule, has at most four leaves. Thus, we can divide the descendant invocations of $\mathcal{I}'$ in group G4 into three subgroups G4.1–G4.3

that are the equivalent of groups G1–G3. G4.1 is invocation $\mathcal{I}'$. Let (a', c') be the sibling pair $\mathcal{I}'$ branches on. G4.2 is either empty or contains a single invocation that branches on a sibling pair (u', v'), where $d' = p_{u'} = p_{v'}$ and $p_{a'} = p_{c'}$ are siblings in F_1'. G4.3 is either empty or contains a single invocation that branches on the sibling pair (a', d'). These node labels are illustrated in Figure 6.6.

The analysis of the invocations in these three groups is identical to the analysis of groups G1–G3 with a', c', d', u', and v' playing the roles of a, c, d, u, and v. However, whenever we choose particular values of P and Q in a child invocation of an invocation in groups G1–G3, we now need to replace these values with $P \cup P'$ and $Q \cup Q'$, where P' and Q' are the input sets of the invocation $\mathcal{I}'$ above that branches on the sibling pair (a', c'). There are two exceptions to this rule. One exception is invocation $\mathcal{I}'$ itself (in group G4.1). If $\mathcal{I}'$ applies Case A3BC, we set $P = P'$ and $Q = Q'$ in the branch that cuts edge $e_{a'}$, $P = Q = \varnothing$ in the branch that cuts the pendant edges of the path from a' to c' in F_2', and $P = P' \cup \{a'\}$ and $Q = Q' \cup \{a'\}$ in the branch that cuts edge $e_{c'}$. The other exception is an application of Case 2B, which sets $P = Q = \varnothing$ in its only child invocation.

G5: The sibling pair (a, e). If we reach an invocation $\mathcal{I}' = m(F_1', F_2', k', P', Q')$ in group G5 with $P' \neq \varnothing$, we have $P' = \{a\}$ or $P' = \{a, a'\}$. In the former case, edge e_a is protected. In the latter case, edges e_a and e_e are both protected. Also, invocation $\mathcal{I}$ in group G1 must have applied Case A2BC and the invocation in group G3 must have applied Case A2BC or A3BC because otherwise the invocation in group G3 sets $P = Q = \varnothing$ in all its child invocations. First we prove that a and e cannot be siblings in F_2' in both cases. Then we analyze the recurrences for these two cases.

Nodes a and e cannot be siblings in F_2 after the invocation in group G3, which branches on the sibling pair (a, d). This is true because the invocation $\mathcal{I}''$ in this group sets $P = \varnothing$ except in the branch of Case A2BC or A3BC that cuts edge e_d. If cutting e_d makes a and e siblings, however, then invocation $\mathcal{I}''$ would have applied Case RB.

If a and e become siblings after an invocation $\mathcal{I}'' = m(F_1'', F_2'', k'', P'', Q'')$ in group G4, assume this invocation branches on a sibling pair (x, y). The path from x to a in F_1'' has either one or two pendant nodes. If this path has two pendant nodes,

y and z, observe that, no matter whether we cut e_x, e_y or the pendant edges of the path from x to y, at least two of x, y, and z merge with e before a and e become siblings. Thus, x belongs to a's sibling subtree of F_2'' and is not its root. Let b_1 be this root. If b_1 is a's sibling in F_2, we obtain a contradiction to the depth rule because x is deeper than a in F_2 and has the same depth as a in F_1. If b_1 is not a's sibling in F_2, then it cannot be a's sibling in F_2'' either because invocation $\mathcal{I}$ applied Case A2BC and, thus, protected a's sibling edge in F_2, that is, a can merge only with nodes in this sibling subtree. Thus, we obtain a contradiction.

If the path from x to a in F_1'' has one pendant node, assume first that $\mathcal{I}''$ has no ancestor invocation in group G4 that applies Case A2BC or A3BC. If a and e become siblings as a result of cutting edge e_y, then the sibling pair (x, y) would have satisfied Case RB, so we would have $P' = \emptyset$. The same argument applies if a and e become siblings as a result of cutting edge e_x. Finally, assume a and e become siblings as a result of cutting the pendant edges of the path from x to y in F_2''. This path has at least three pendant edges because otherwise Case 2B would have applied and $P' = \emptyset$. Thus, invocation $\mathcal{I}''$ applies Case A3BC and sets $P = Q = \emptyset$ in the branch that cuts these pendant edges, a contradiction.

Finally, consider the case when $\mathcal{I}''$ does have an ancestor invocation $\mathcal{I}'''$ in group G4 that applies Case A2BC or A3BC. By the same arguments as in the previous paragraph, a and e becoming siblings after invocation $\mathcal{I}''$ implies that invocation $\mathcal{I}''$ applies Case RB, 2B or A3BC and, if it applies Case A3BC, it must cut the pendant edges of the path from x to y. In the latter two cases, we have $P = Q = \emptyset$ in the child invocation of $\mathcal{I}''$ that is an ancestor of $\mathcal{I}'$, a contradiction. If invocation $\mathcal{I}''$ applies Case RB and does not set $P = \emptyset$ in its only child invocation, then $\mathcal{I}'''$ is the invocation that branches on the sibling pair (a', c') and $\mathcal{I}''$ is in the branch of $\mathcal{I}'''$ that cuts edge c', protects edge a', and sets $P = \{a, a'\}$. This implies that w.l.o.g. $x = a'$, e_x is protected in invocation $\mathcal{I}''$, and x initially had the same depth in F_1 as a. Once again, let b_1 be a's sibling in F_2. If $x \notin F_2^{b_1}$, then a and x cannot merge after cutting y because e_{b_1} is protected. Thus, since a and e merge after applying Case RB to the sibling pair (x, y), we must in fact cut x to let a and $x = e$ merge. This, however, is a contradiction because x is protected. If $x \in F_2^{b_1}$, we must have $x = b_1$ because otherwise our choice of the sibling pair (a, c) in invocation $\mathcal{I}$ would violate the depth rule.

Since we also have $x = a'$, however, this implies that a is a pendant node of the path from a' to c' in invocation $\mathcal{I}'''$. Therefore, $\mathcal{I}'''$ makes at most two child invocations, which contradicts that we set $P = \{a, a'\}$ in the branch that cuts edge $e_{c'}$. Since this exhausts all possible cases, we conclude that a and e cannot be siblings in F_2' at the beginning of invocation $\mathcal{I}'$. We finish the analysis by considering whether $P' = \{a\}$ or $P' = \{a, a'\}$ in invocation $\mathcal{I}'$.

If $P' = \{a\}$ and invocation $\mathcal{I}'$ applies Case AC, B, RB, RB*, 2B or BCY, it has only one child invocation, with parameter at most $k' - 1$. We set $P = Q = \varnothing$ in this child invocation, which gives $I(k', \{a\}, Q') \leqslant 1 + \alpha 2^{k'-1} - 3 \leqslant \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$. If invocation $\mathcal{I}'$ applies Case A2BC or A3BC and $Q' \neq \varnothing$, we set $P = Q = \varnothing$ in both child invocations of $\mathcal{I}'$, which gives $I(k', \{a\}, Q') \leqslant 1 + I(k' - 1, \varnothing, \varnothing) + I(k' - 2, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-1} - 3 + \alpha 2^{k'-2} - 3 < \alpha(2^{k'-1} + |Q'|2^{k'-2}) - 2$. Finally, if invocation $\mathcal{I}'$ applies Case A2BC or A3BC and $Q' = \varnothing$, then the invocations $\mathcal{I}_1$ and $\mathcal{I}_3$ in groups G1 and G3 must both have applied Case A2BC. Moreover, $\mathcal{I}_3$ must have made two recursive calls. Now consider F_2. The path from a to c in F_2 has two pendant nodes b_1 and b_2, which both become protected after cutting edge e_c. If $d \notin F_2^{b_1}$, then b_1 is also a pendant node of the path from a to d in invocation $\mathcal{I}_3$, and $\mathcal{I}_3$ would in fact make only one recursive call. Thus, $d \in F_2^{b_1}$. Since we set $P = \varnothing$ when cutting the pendant edges of the path from a to d, we must in fact have cut edge e_d and protected the pendant edges of the path from a to d. Let b_1' and b_2' be the bottom endpoints of these two pendant edges. Now consider the possible locations of e. If $e \in F_2^{b_1'}$, then $e_{b_2'}$ is a protected pendant edge of the path from a to e in F_2', contradicting that invocation $\mathcal{I}'$ makes two recursive calls. We obtain a similar contradiction if $e \in F_2^{b_2'}$. Finally, if $e \notin F_2^{b_1}$, then e_{b_1} is a protected pendant edge of the path from a to e in F_2', again contradicting that $\mathcal{I}'$ makes two recursive calls. Thus, the case when $\mathcal{I}_1$ and $\mathcal{I}_3$ both apply Case A2BC and $\mathcal{I}'$ applies Case A2BC or A3BC cannot arise.

If $P' = \{a, a'\}$, observe first that $a', c' \in F_2^e$ in invocation $\mathcal{I}$, where (a', c') is the sibling pair on which the invocation $\mathcal{I}'' = m(F_1'', F_2'', k'', P'', Q'')$ that adds a' to P' branches. We distinguish whether a' is a child of e or has distance two from e. (By the depth rule, a greater depth of a' is not possible and, since both a' and c' are in F_2^e, $a' \neq e$.)

If a' has distance two from e in F_2, then observe that a and a' have the same depth in F_1. Let b_1 be a's sibling in F_2. If $a' \in F_2^{b_1}$, then we must have $a' = b_1$ because otherwise we would not have chosen the sibling pair (a, c) in invocation $\mathcal{I}$. This implies that $c' \notin F_2^{b_1}$ and a is a pendant edge of the path from a' to c' in F_2''. Since e_a is protected, however, this implies that $\mathcal{I}''$ makes at most two recursive calls, contradicting that we add a' to P in one of them. Thus, $a' \notin F_2^{b_1}$.

If $a' \notin F_2^{b_1}$, b_1 is a pendant node of the path from a to a' in F_2'. Since e_a, $e_{a'}$, and e_{b_1} are all protected in F_2', invocation $\mathcal{I}'$ cannot make any recursive calls at all. Thus, we obtain $I(k', \{a, a'\}, Q') = 1 < \alpha(2^{k'-2} + |Q'|2^{k'-3}) - 1$ for the number of descendant invocations of $\mathcal{I}'$.

If a' is e's child in F_2, we distinguish whether $a' \in F_2^{b_1}$. If $a' \notin F_2^{b_1}$, we can apply the argument from the previous paragraph, so assume $a' \in F_2^{b_1}$. We argued above that we obtain a contradiction if $c' \notin F_2^{b_1}$, so $c' \in F_2^{b_1}$. Since invocation $\mathcal{I}''$ applied Case A2BC or A3BC, the path from a' to c' in F_2'' has at least two pendant nodes. Since at least one of them is attached to the path from a' to the LCA of a' and c', this implies that the path from a to a' in F_2' has at least two pendant nodes. Now, since e_a and $e_{a'}$ are protected in F_2', invocation $\mathcal{I}'$ can make at most one recursive call, which cuts the pendant edges of the path from a to a'. We set $P = Q = \varnothing$ in this recursive call and obtain $I(k', \{a, a'\}, Q') \leqslant 1 + I(k'-2, \varnothing, \varnothing) \leqslant 1 + \alpha 2^{k'-2} - 3 < \alpha(2^{k'-2} + |Q'|2^{k'-3}) - 1$.

G6: Sibling pairs outside F_1^r. Observe that our analysis of groups G1–G5 above did not consider any invocations outside groups G1–G5 and with $P \neq \varnothing$. Therefore, we only require that $I(k', \varnothing, \varnothing) \leqslant \alpha 2^{k'} - 3$ for every invocation $m\,(F_1', F_2', k', \varnothing, \varnothing)$ in group G6, which holds by induction. This concludes the proof of Theorem 6.13 and, thus, of Theorem 6.1.

6.4 Conclusions

Our novel edge protection scheme along with several new edge cases greatly improves the running time bound of our MAF algorithm. As we show in Chapter 7, this also provides a great improvement in practice. This scheme and the techniques we introduced to prove its correctness and running time may be of great utility for developing and analyzing improved algorithms for many other exponential search problems. In particular, these techniques should be applied in future work to improve the MAAF algorithm of Chapter 4.

Chapter 7

RSPR: Experimental Evaluation of the Binary MAF Algorithm and Associated Heuristics

In this chapter, we evaluate experimentally the efficiency of our MAF algorithms from Chapters 3, 5, and 6. These algorithms have been implemented in the software program RSPR with the C++ programming language and their source code is available under the open source GPL license [105]. Section 7.1 evaluates version 1.01 of RSPR, an implementation of the MAF algorithm in Chapter 3. These results were presented at the Symposium for Experimental Algorithms (SEA) [100] and are reproduced here with kind permission of Springer Science+Business Media. Section 7.2 evaluates version 1.2.0 of RSPR, an implementation of the binary MAF algorithm in Chapter 6 and a partial implementation of the multifurcating MAF algorithm in Chapter 5 that quickly computes MAFs of a pair of trees such that one tree is binary and the other may be multifurcating. This restricted case is common in supertree analysis when comparing a binary supertree to a set of multifurcating gene trees and this implementation is vital to the supertree and LGT analyses in Chapter 8.

The MAF algorithm of Chapter 3 introduced improved branching rules to be used in the algorithms of [103]. These branching rules, Step 6 of the algorithm, reduce the running times of that $\mathrm{O}\left(3^k n\right)$ algorithm for SPR distance to $\mathrm{O}\left(2.42^k n\right)$. While these theoretical improvements are valuable in their own right, our main contribution from [100] was to evaluate the practical performance of the algorithm of [103] and the impact of our improved branching rules. An additional optimization we apply is to use the linear-time 3-approximation algorithm for SPR distance of [83, 103] to prune branches in the search tree that are guaranteed to be unsuccessful. This reduces the size of the search tree substantially and leads to a corresponding decrease in running time. We demonstrate that each of the improved branching rules and the pruning of unsuccessful branches have a marked and distinct effect on the performance of the algorithm. Our experiments confirm that our algorithm is orders of magnitude faster than the previous best exact alternatives [19,110] based on reductions to integer linear programming and satisfiability testing, respectively. The largest distances reported using implementations of previous methods are a hybridization number of 14 on 40 taxa [21] and an SPR distance of 19 on 46 taxa [110]. In contrast, our method took less than 5 hours to compute SPR distances of up to 46 on trees with 144 taxa and 99 on synthetic 1000-leaf trees. This represents a major step forward towards tools that can infer reticulation scenarios for the thousands of genomes that have been fully sequenced to date.

The MAF algorithm of Chapter 6 introduced the novel technique of edge protection to further reduce the time required for MAF computation to $\mathrm{O}\left(2^k n\right)$. This technique can be combined with the special case of the multifurcating MAF algorithm from Chapter 6, noted in Section 5.5, where only one of the compared trees must be binary. See Section 8.2.4 for more information. As an additional optimization we use the cluster reduction of Linz and Semple [63] to partition the two trees into subproblems that can be solved iteratively. This reduction greatly improves the performance of the MAF algorithm as the time required to compute an MAF scales with the SPR distance computed for each cluster, rather than the full distance between the trees. We extended the cluster reduction to compute general MAFs rather than the weighted MAFs of Linz and Semple and reduced the time required to compute the cluster reduction from $\mathrm{O}\left(n^3\right)$ to $\mathrm{O}\left(n\right)$. See Section 8.2.5 for details. These improvements accelerate our algorithm by orders of magnitude, computing distances of 46 on trees with 144 taxa in an average of 0.0275 seconds. These computations required an average of 2538.462 seconds with the MAF algorithm of Chapter 3, more than 90,000 times longer, and would be infeasible with previous algorithms.

7.1 Evaluation of the $\mathrm{O}\left(2.42^k n\right)$-time SPR Distance Algorithm

In this section, we present an experimental evaluation of our MAF (SPR distance) algorithm that compares the algorithm's performance to that of two competitors using the protein tree data set examined in [13, 14] and using synthetic trees. We also investigate the impact of the improved branching rules in Step 6 on the performance of our algorithm. Our competitors were **sprdist** [110] and **treeSAT** [19], which reduce the problem of computing SPR distances to integer linear programming

(ILP) and satisfiability testing, respectively. We do not provide a comparison with **EEEP** [13] because **sprdist** outperformed it and other heuristics at finding the exact SPR distance between binary rooted phylogenies [110].

For **sprdist** and **treeSAT**, we used publicly available implementations of these algorithms. Note that the publicly available implementation of **sprdist** uses the open source GLPK ILP solver. Wu [110] observed that the commercial CPLEX ILP solver provided a speedup between 3–9x over GLPK, but such an improvement would not materially affect the results obtained below, reaching only the performance of our previous $\mathrm{O}\left(3^k n\right)$-time algorithm.

For our own algorithm, we developed an implementation in C++ that allowed us to individually turn the optimized branching rules in Step 6 on and off. When the optimized branching rule in one of the cases is turned off, the algorithm uses the 3-way branching of [103] in this case. In particular, with all optimizations off, the algorithm is the one of [103]. Source code for our algorithm is available at [105].

We also implemented the linear-time 3-approximation algorithm for MAF of [103] and used it to implement two additional optimizations of our FPT algorithm. The FPT algorithm searches for the correct value of $e\left(T_1, T_2, T_2\right)$ by starting with a lower bound k of $e\left(T_1, T_2, T_2\right)$ and incrementing k until it determines that $k = e\left(T_1, T_2, T_2\right)$. If the 3-approximation algorithm returns a value of k', then $e\left(T_1, T_2, T_2\right) \geqslant \lceil k'/3 \rceil$; by using this as the starting value of our search, we can skip early iterations of the algorithm and thereby obtain a small improvement in the running time. The same approach can be used in a branch-and-bound strategy that prunes unsuccessful branches from the search tree. In particular, we extended Step 1 of the FPT algorithm as follows:

1′. (Failure) If $k < 0$, return “no”. Otherwise compute a 3-approximation k' of $e\left(T_1, T_2, F_2\right)$. If $k' > 3k$, then $e\left(T_1, T_2, F_2\right) > k$; return “no” in this case.

We allowed this optimization of Step 1 to be turned on or off in our algorithm to investigate its effect on the running time, but our implementation always uses the 3-approximation algorithm to provide a starting guess of $e\left(T_1, T_2, T_2\right)$.

7.1.1 Data Sets

The protein tree data set of [13, 14] contains 5689 protein trees with 10 to 144 leaves (each corresponding to a different microbial genome); each of these was compared in turn to a rooted reference tree covering all 144 genomes. The protein trees were unrooted, so we selected a rooting for each tree that gave the minimum SPR distance according to the 3-approximation algorithm of [103].

The synthetic tree pairs were created by first generating a random tree T_1 and then transforming it into a second tree T_2 using a known number of random SPR operations. Note that the SPR distance may be lower because the sequence of SPR operations we generated may not be the shortest such sequence. For n taxa, the label set of T_1 was represented using integers 1 through n, and T_1 was generated by splitting the interval $[1, n]$ into two sub-intervals uniformly at random, recursively generating two trees with these two intervals as label sets and then adding a root to merge these trees. Random SPR operations were generated by choosing an edge xp_x to cut uniformly at random and then choosing the new parent p'_x of x uniformly at random from among all valid locations of p'_x. We constructed pairs of 100-leaf trees with 1–20 SPR operations and with $25, 30, \ldots, 50$ SPR operations. We also constructed pairs of 1000-leaf trees with 1–20 SPR operations and with $25, 30, \ldots, 100$ SPR operations. For each tree size and number of SPR operations we generated ten pairs of trees.

7.1.2 Results

Our experiments were performed on a 3.16Ghz Xeon system with 4GB of RAM and running CentOS 2.6 Linux in a Rocks 5.1 cluster. Our code was compiled using gcc 4.4.3 and optimization -O2. Each run of an algorithm was limited to 5 hours of running time. If it did not produce an answer in this time limit, we say the algorithm did not solve the given input instance in the following discussion. We refer to the FPT algorithm with all optimizations off as **fpt**, and with only the branch-and-bound optimization turned on as **bb**. The activation of the improved branching rules in Step 6 is indicated using suffixes **sc** (Case 6.1: separate components), **cob** (Case 6.2: cut only b), and **cab** (Case 6.3: cut a or b). Thus, the algorithm with all optimizations on is labelled **bb_cob_cab_sc**.

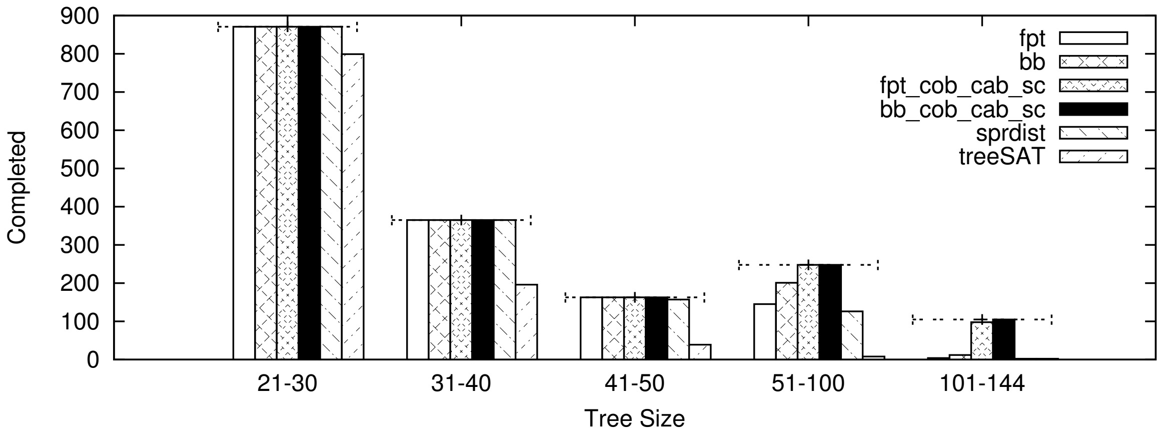


Figure 7.1: Completion on the protein tree runs grouped by tree size. The $O(3^k n)$ algorithm with (**bb**) and without (**fpt**) branch-and-bound is compared to the $O(2.42^k n)$ algorithm of Chapter 3 with (**bb_cob_cab_sc**) and without (**fpt_cob_cab_sc**) branch-and-bound, as well as sprdist, and treeSAT.

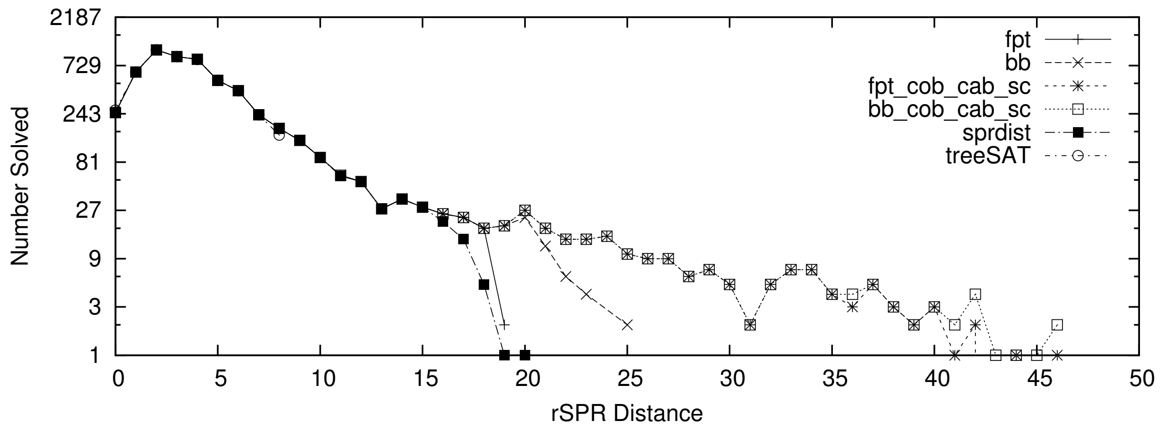


Figure 7.2: Number of protein trees solved by rSPR distance.

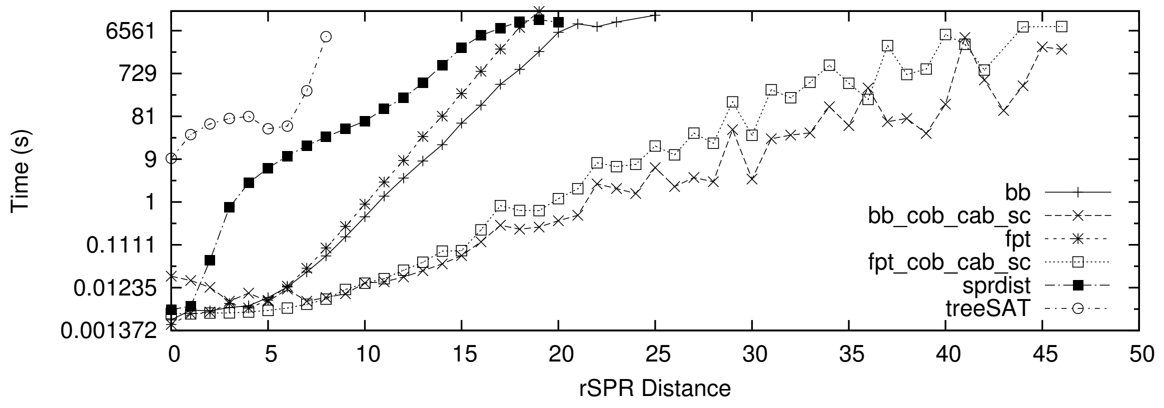


Figure 7.3: Mean running time of the FPT, sprdist, and treeSAT protein tree runs.

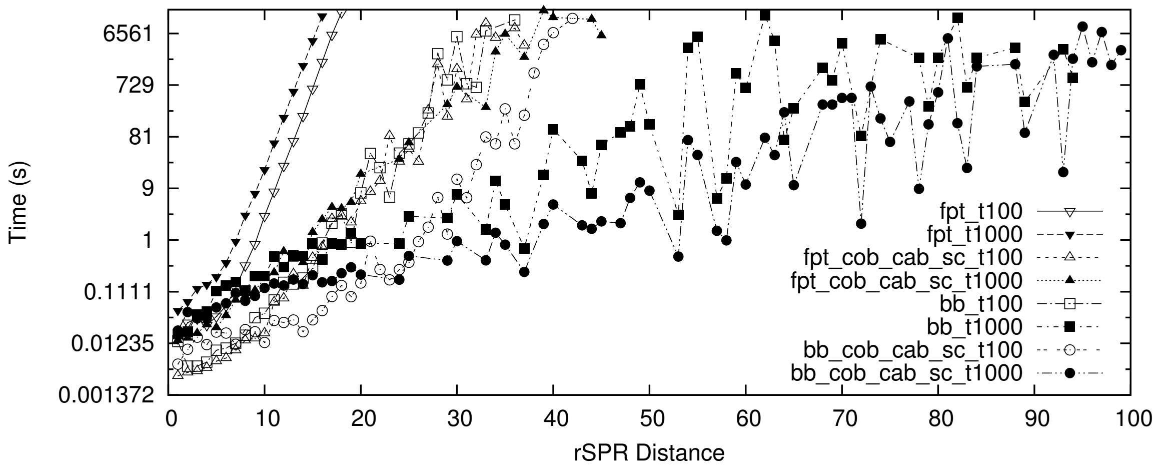


Figure 7.4: Mean running time of the FPT algorithm on the data set of randomly permuted trees. The trees with 100 and 1000 leaves are shown separately.

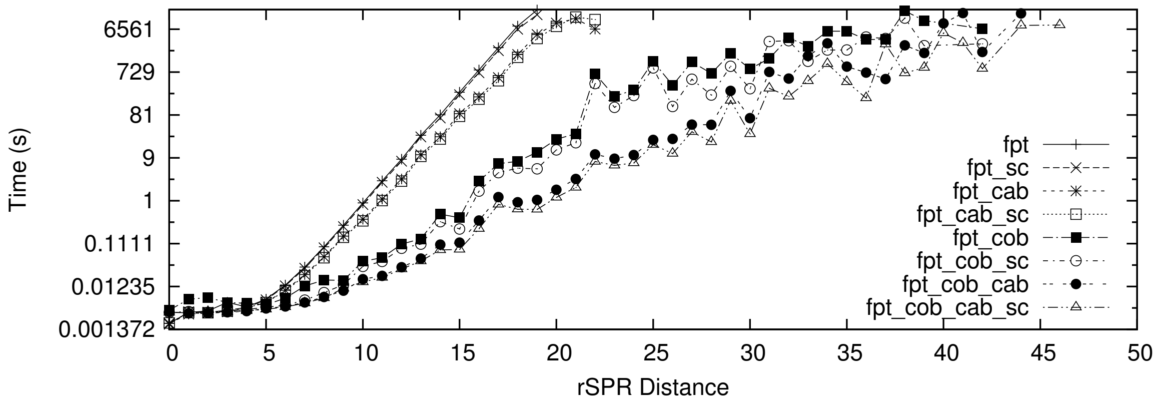


Figure 7.5: Mean running time for combinations of the new cases on the protein tree runs. The improved branching rules in Step 6 are indicated using suffixes **sc** (Case 6.1: separate components), **cob** (Case 6.2: cut only b), and **cab** (Case 6.3: cut a or b).

Number of instances solved.

Figure 7.1 shows the number of solved protein tree instances for the given ranges of tree sizes. Our experiments showed that the average SPR distance for trees of the same size ranged between one sixth and one third of the number of leaves. All of the algorithms solved all instances with 20 or fewer leaves and only **treeSAT** did not solve all instances with 40 or fewer leaves. **sprdist** solved most of the instances with 41-50 leaves, and half of the instances with 51-100 leaves, but very few of the larger instances. **fpt** performed similarly to **sprdist** but solved all of the instances with 41-50 leaves and more of the larger instances than **sprdist**. **bb** improved upon this somewhat. However, adding our new branching rules improved the results greatly. In particular, **bb_cob_cab_sc** solved all of the instances in this data set.

Figure 7.2 shows the number of protein trees found with a given SPR distance from the reference tree. The "number solved" axis is a $\log_3$-scale to allow easy comparison of the trees with small and large SPR distances, as the majority had small SPR distances. **treeSAT** was unable to solve any instances with SPR distance greater than 8. **sprdist** and **fpt** solved instances with a distance as large as 20. Since **bb_cob_cab_sc** solved all the instances in this data set, including instances with an SPR distance of 46, we were able to verify that **sprdist** and **fpt** solved all instances with SPR distance up to 15 and 18, respectively.

Running time.

Figure 7.3 shows the mean running time of the algorithms on *solved* protein tree instances with the given SPR distance. The time axis here and in the following figures is a $\log_3$-scale to highlight the exponential running time of the algorithms and to allow easy comparison of the runs. The curves for some of the algorithms 'dip" for higher distance values, which is a result of taking the average running time only over solved instances. In addition, the curves for larger k values are not smooth due to the small number of samples with these large distances. The slope of the curve for **fpt** is close to 1, indicating that the algorithm is close to its worst-case running time of $\mathrm{O}\left(3^k n\right)$. **bb** shows a marked improvement over **fpt**; however, the improvement achieved using the new branching rules is much more dramatic. **treeSAT** was much slower than all the other algorithms and although **sprdist** solved a similar number of instances as **fpt**, as shown in Figure 7.2, it took much longer to solve them on average. The two instances that **sprdist** solved with an SPR distance of 19 and 20 are an exception to this, but that is likely an artifact of considering only solved instances. **bb_cob_cab_sc** solved all input instances with SPR distance of up to 20 in 5.5 seconds or less, and solved instances with SPR distance up to 46 in well under 2 hours, while none of the previous methods was able to solve instances with SPR distance greater than 20 in under 5 hours.

Figure 7.4 shows the mean running time of the fixed-parameter algorithms on the random data set. As expected, **fpt** took 10 times longer on average for the 1000-leaf trees as for the 100-leaf trees, given the same SPR distance. **fpt_cob_cab_sc** did not show this difference, which suggests that the improved branching rules have a more pronounced impact on larger trees. **bb_cob_cab_sc** was able to solve instances with SPR distances up to 99 on the 1000-leaf trees, while a distance of 42 was the limit on 100-leaf trees. We believe that, since the proportion of SPR operations to the number of leaves is smaller for the bigger trees, the randomly generated SPR operations are more likely to operate on independent subtrees, which brings the approximation ratio of the approximation algorithm closer to its worst-case bound of 3 on these inputs. In our case, this provides better lower bounds on the true SPR distance and, thus, allows us to prune more branches in the search tree than is the case for the smaller trees.

Figure 7.5 shows the mean running time of the fixed-parameter algorithms without branch-and-bound on the protein tree data set and using only some of the improved branching rules. Case 6.1, Case 6.3, and Case 6.2 provide small, moderate and large improvements, respectively. Using all of the cases gives another large improvement, since each occurs under different conditions.

7.1.3 Conclusions

Our theoretical results improve on previous work, and our experiments confirm that these improvements have a tremendous impact in practice. Our algorithm efficiently solves problems with up to 144 leaves and an SPR distance of 20 in less than a second on average; for distance values up to 46, the running time was less than two hours. Our branch-and-bound approach showed a marked improvement on larger

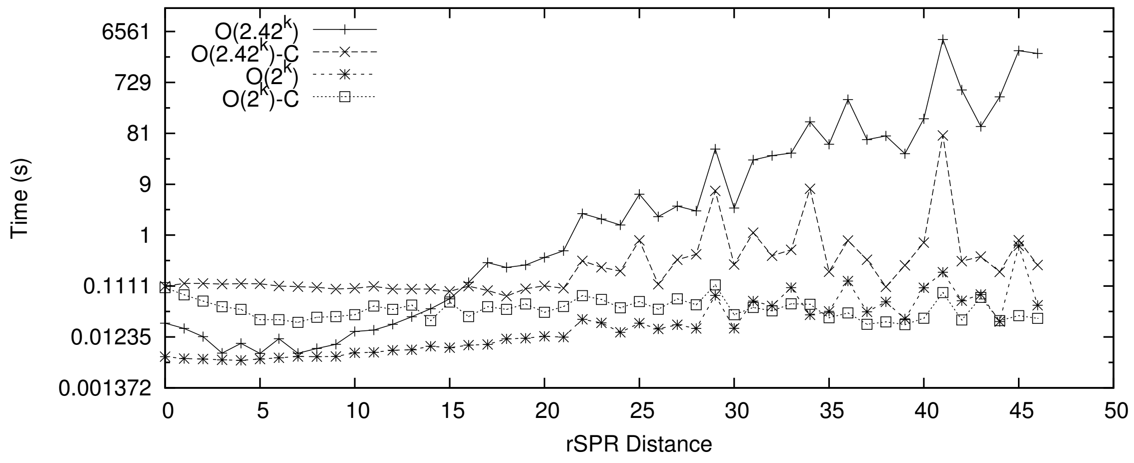


Figure 7.6: Mean running time of the binary MAF algorithms on the protein tree dataset with and without clustering (-C).

trees, allowing us to compute distance values up to 42 on 100-leaf synthetic trees and 99 on 1000-leaf synthetic trees.

7.2 Evaluation of the $\mathrm{O}\left(2^k n\right)$-time SPR Distance Algorithm

In this section, we extend the experimental evaluation of our $\mathrm{O}\left(2.42^k n\right)$-time MAF (SPR distance) algorithm in Section 7.1 to the full $\mathrm{O}\left(2^k n\right)$-time MAF algorithm including edge protection (Chapter 6) and the cluster reduction (Section 8.2.5). We show that these optimizations provide an enormous improvement in practice, decreasing the time required to compute an SPR distance of 46 between two trees with 144 leaves from 2538 seconds to 0.0275 seconds.

Figure 7.6 shows the mean running time of the MAF algorithms on protein tree instances with the given SPR distance. As in Figure 7.3, the time axis is a $\log_3$-scale to highlight the exponential running time of these algorithms. We compared the $O\left(2.42^k n\right)$ and $O\left(2^k n\right)$ algorithms with and without the cluster reduction.

The cluster reduction alone provided a large improvement, reducing the time required to compute large SPR distances from hours to seconds. We observed, however, that the overhead of computing the cluster reduction increased the running time for SPR distances of 15 or less and the improvement with larger distances varied greatly with how "clusterable" the data was. In particular, the two instances with an SPR distance of 41 were solved in an average of 73.7 seconds (a 62x speedup), while the two instances with an SPR distance of 46 were solved in average of 0.276 seconds (a 9,196x speedup).

The $O\left(2^k n\right)$ MAF algorithm alone provided a tremendous improvement, requiring at most 0.633 seconds for any instance. For example, the mean running times for an SPR distance of 41 and 46 was only 0.202 seconds (a 22,926x speedup) and 0.0485 seconds (a 52,330x speedup), respectively. The cluster reduction increased the running time of the $O\left(2^k n\right)$ algorithm on instances with an SPR distance less than 30 but provided some improvement on more difficult instances, requiring an average of 0.0275 seconds to compute SPR distances of 46 (a 92,307x speedup over the $O\left(2.42^k n\right)$ algorithm). The combined algorithm will likely be useful when computing much larger SPR distances.

We found that the $O\left(2^k n\right)$ MAF algorithm with clustering greatly reduced the time required to compare evolutionary trees in practice; for large SPR distances we observed an improvement of nearly 5 orders of magnitude—hours to fractions of a second. Extending the MAAF algorithm of Chapter 4 with these improvements and testing it remain for future work. In Section 8.3 we show that the algorithm performs just as efficiently when comparing a binary and multifurcating tree. Our efficient MAF algorithm thus makes the SPR distance a practical metric for meaningful datasets and enables new evolutionary analyses such as the supertree and LGT analysis of Chapter 8.

Chapter 8

SPR Supertrees: Inferring Trees of Life and Highways of Gene Sharing

In this chapter, we present the first method to construct supertrees by using the subtree prune-and-regraft (SPR) distance as an optimality criterion. Supertree methods reconcile a set of phylogenetic trees into a single structure that is often interpreted as a branching history of species. A key challenge is combining conflicting evolutionary histories that are due to artifacts of phylogenetic reconstruction and phenomena such as lateral gene transfer (LGT). The SPR distance, a measure of LGT, is thus an ideal distance measure to reconcile such conflict. Although they often work well in practice, existing supertree approaches use optimality criteria that do not reflect underlying processes, have known biases and may be unduly influenced by LGT.

Although calculating the rooted SPR distance between a pair of trees is NP-hard, our new maximum agreement forest-based methods can reconcile trees with hundreds of taxa and more than 50 transfers in fractions of a second, which enables repeated calculations during the course of an iterative search. Our approach can accommodate trees in which uncertain relationships have been collapsed to multifurcating nodes. Using a series of simulated benchmark datasets, we show that SPR supertrees are more similar to correct species histories under plausible rates of LGT than supertrees based on parsimony or Robinson-Foulds distance criteria. We successfully constructed an SPR supertree from a phylogenomic dataset of 40,631 gene trees that covered 244 genomes representing several major bacterial phyla. Our SPR-based approach also allowed direct inference of highways of gene transfer between bacterial classes and genera; a small number of these highways connect genera in different phyla and can highlight specific genes implicated in long-distance LGT.

8.1 Introduction

An organism's genome, typically comprising many thousands of genes, provides a detailed record of its past. While sets of homologous genes from a set of genomes can provide evidence about organismal relationships, individual gene trees covering these genomes may be influenced by processes including paralogy and gene loss, lineage sorting and lateral gene transfer (LGT) [41,68]. One approach to reconcile trees that differ due to these processes and to artifacts of the phylogenetic inference process is to construct a single tree that aims to reflect the relationships in the input trees. Supertree methods generate a single tree, which may serve as a hypothesis of organismal descent or relatedness, by optimizing a similarity criterion. Supertrees have been used to represent large-scale phylogenies including the first phylogeny of nearly all extant mammals [17], the first family-level phylogeny of flowering plants [38], and the first species-level phylogeny of non-avian dinosaurs [64]. They have also been used to study the extent of LGT in prokaryotes [14] and to disentangle the origin of eukaryotic genomes [78]. One key advantage of supertree methods is that they can take as input sets of gene trees sampled from overlapping but non-identical sets of taxa, in contrast with consensus tree approaches, which require that all input trees contain exactly the same set of leaves. Simulations have shown that supertrees are more reliable in the presence of a moderate amount of misleading LGT than the supermatrix approach which requires concatenated alignments of many gene sequences [59].

Many optimality criteria have been proposed for supertree construction. Matrix representation with parsimony (MRP) [8, 80] was among the earliest methods proposed and remains the most commonly used, but detailed work with MRP has raised several concerns with the approach. MRP converts input trees into a binary character matrix and solves the parsimony problem on this matrix. Although the parsimony problem is NP-hard, fast hill-climbing heuristics in PAUP* or TNT allow MRP to be applied to large datasets [42, 85, 94]. MRP is very effective in practice, quickly constructing supertrees of competitive quality in every tested metric [18,30,40]. However, it is not clear why the MRP approach performs so well and it may generate relationships that do not belong to any of the source trees [79], has problems resulting from unequal representation of taxa [16], and may include relationships contradicted by the majority of source trees [43]. Other developed supertree criteria include consensus supertrees [1], majority-rule supertrees [35], Quartet supertrees [76] and Triplet supertrees [61]. However, like MRP, other supertree building methods that are not based on symmetric tree-to-tree similarity measures may be unduly influenced by the shapes of the input trees [107].

Bansal et al. [6] recently proposed Robinson-Foulds (RF) supertrees, which aim to minimize the total RF distance [82] between the supertree and the set of input trees. The RF measure captures the number of clusters (clades, in the binary case) that differ between two trees, so the RF supertree approach aims to maintain as much phylogenetic information from the input trees as possible. Fast hill-climbing heuristics make computing rooted RF supertrees feasible from binary input trees and others have begun to extend this to unrooted trees with local search heuristics [29]. While RF appears to be a good criterion for supertrees, it may not be suitable for datasets with substantial amounts of LGT: a single "long-distance" LGT event between distant taxonomic relatives will result in many discordant bipartitions and a high RF distance. If many organisms participate in long-distance LGT, then "phylogenetic compromise" trees [12] may emerge which reflect neither the correct species relationships, nor the dominant pathways of gene sharing. The requirement that all input trees be binary is also potentially limiting, as many relationships in trees inferred from sequence data are unsupported by statistics such as the bootstrap, and should be collapsed into multifurcations.

Another well-studied criterion for expressing differences between trees is the subtree prune-and-regraft (SPR) distance [51]. The SPR operation involves splitting a pendant subtree from the rest of the tree, and reattaching it at a different location, with the rooting of the subtree preserved. Since SPR operations allow the pruned subtree to be reattached anywhere, they can accommodate long-distance transfers in a single step; such a transfer would increase the SPR distance by only 1, whereas the RF distance could be drastically increased. The SPR distance is the minimum number of such operations required to reconcile two trees. The relationship between an SPR operation and the topological consequences of an LGT event [13] makes SPR a natural criterion for assessing a supertree whose constituent trees contain a large number of LGT events. Given its relationship with the RF distance, the SPR criterion may also be suitable for datasets where a phenomenon other than LGT is the principal confounding factor. To date, no SPR-based supertree approach has been developed, in part because computing the SPR distance between two phylogenetic trees is NP-hard [23, 53].

Combining two recent advances makes SPR supertrees feasible. First, using the equivalence between Maximum Agreement Forests (MAFs) and rooted SPR distance [23, 51], Whidden and Zeh [103] and Whidden et al. [100, 102] developed an algorithm with running time $O\left(2.42^k n\right)$ (Chapter 3). The resulting implementation was orders of magnitude faster than any previous algorithm and is able to compute SPR distances of up to 46 on trees with 144 prokaryotic taxa, and 99 on synthetic 1000-leaf trees, in less than 5 hours (Chapter 7). We have extended this algorithm with several enhancements that improve the running time to $O\left(2^k n\right)$ for binary input trees (Chapter 6), and allow the inclusion of input trees in which uncertain relationships have been collapsed into multifurcating nodes (Chapter 5). Second, Linz and Semple [63] developed a cluster reduction technique which can reduce the computation of an MAF into several subproblems, yielding an exponential reduction of the running time in practice. The approach taken by Linz and Semple is similar to the cluster reduction rule of Baroni et al. [7] for computing the hybridization distance but requires more care in choosing which maximum agreement forest to take for each subproblem to build the complete MAF. We have also reduced the time required to compute a cluster reduction to linear from the originally published $O\left(n^3\right)$. Neither refinement alone is fast enough to compute the thousands of SPR distances required to build an SPR-based supertree on interesting numbers of taxa. However, by combining the cluster reduction with our improved MAF-based approach we obtain dramatic improvements in running time, processing tree pairs that previously required 1-5 hours to reconcile in one second or less, thus enabling the many SPR distance computations needed to iteratively construct a supertree.

Our heuristic approach uses a greedy hill-climbing strategy to build an initial supertree, then refines this supertree using iterative global SPR rearrangements. We use a bipartition-based heuristic to identify and ignore proposed rearrangements that violate relationships that are well-supported in many trees, greatly reducing the number of rearrangements that need to be evaluated. These algorithms are implemented in the SPR Supertree software version 1.2.0 [106]. The software is freely available, open source and licensed under the GNU GPL version 3. Here we describe the steps in our approach, and demonstrate the speedups achieved using the algorithmic refinements described above. Our experiments using simulated datasets with LGT show that the SPR approach is more accurate than RF and, for some realistic rates and regimes of LGT, MRP as well. Comparisons based on the eukaryotic datasets used by Bansal et al. [6] for benchmarking show that the SPR approach yields supertrees with lower total SPR distances to the input trees than either RF or MRP, and with slightly higher RF and parsimony scores. To demonstrate the application of the SPR supertree approach on a dataset in which considerable LGT is expected, we also used a phylogenomic data set of 244 bacteria covering 393,876 genes in 40,631 orthologous sets to analyze preferential transfer of genes between bacterial lineages. We were able to reconstruct a highly plausible supertree, and with the SPR approach we identified putative highways of gene sharing. Interestingly, preference for alternative hypotheses of the relatedness between bacterial phyla depended on the choice of gene tree rootings, suggesting that unrooted supertree methods may be ignoring plausible hypotheses.

8.2 Methods

8.2.1 Calculating the Subtree Prune-and-Regraft Distance Between a Pair of Rooted Trees

We can compute the SPR distance between a pair of rooted trees quickly in practice, despite the NP-hardness of the problem [23], using our efficient fixed-parameter bounded search tree algorithm in combination with our linear-time formulation of Linz and Semple's cluster reduction [63] to solve the equivalent Maximum Agreement Forest problem. The MAF problem is a static version of the SPR distance problem that is easier to manipulate and analyze. An agreement forest of two trees is a forest on the same label set that can be created by cutting (deleting) edges from either tree. Bordewich and Semple [23] showed that a maximum agreement forest—an agreement

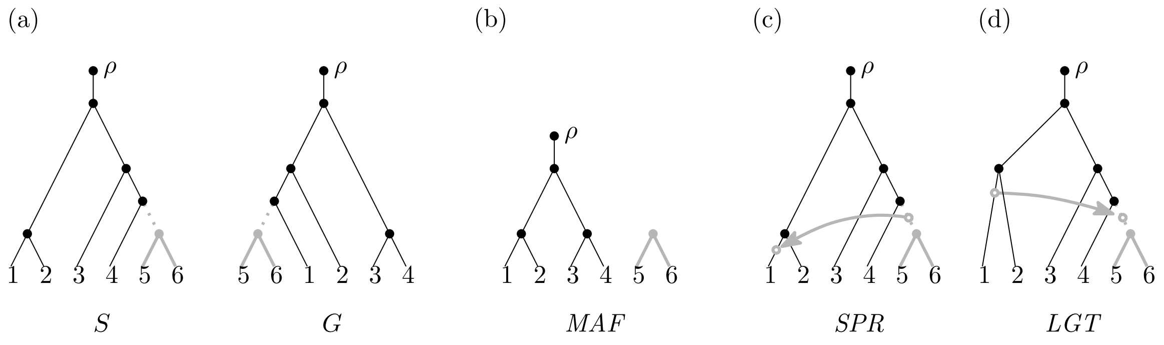


Figure 8.1: The equivalence between the SPR distance and MAF size. (a) The species tree S and gene tree G differ only in the placement of the grey subtree. The roots of these trees are denoted by ρ. (b) The MAF of S and G is produced by cutting the dotted edge in both trees. (c) Each component of an MAF other than the component containing ρ represents an SPR move. A single SPR move transforms S into G by moving the grey subtree in S to its position in G. (d) Each SPR move models an LGT event in the reverse direction. From the MAF of S and G we infer that a transfer of gene G has occurred from an ancestor of taxon 1 to an ancestor of taxon 4.

forest that requires the fewest edge cuts—requires exactly as many edge cuts as the SPR distance between the trees. Indeed, each edge cut represents a transfer and the proposed series of transfers can be quickly inferred from the MAF (Figure 8.1). Our algorithms, like most recent work on the SPR distance, compute such MAFs.

Our published MAF algorithm [100, 102] (Chapter 3) operates in a bottom-up fashion in the first tree, denoted T_1, and reduces the second tree to a forest, denoted F_2. During the algorithm we identify subtrees that are identical in T_1 and F_2 and, in particular, pairs of such trees that are siblings in T_1 (sibling pairs). If any identical subtree is a component of F_2 we cut its corresponding parent edge in T_1. If any sibling pair in T_1 is also a sibling pair of F_2 we note that their parent nodes are identical in T_1 and F_2. If neither of these two situations applies, we identify at most three possible edge cutting scenarios and explore each recursively. We explore each scenario in turn, thus using very little memory, and use our 3-approximation algorithm (which operates similarly but simply cuts all three possible edges so that its running time scales linearly and may return at most 3 times the correct distance) to avoid exploring scenarios that are guaranteed to not return an optimal MAF.

We have enhanced our MAF algorithm to prioritize non-branching edge cut scenarios and ignore duplicate search branches through *edge protection*. First, we examine each sibling pair to select a sibling pair with only one edge cutting scenario, if any exist. This limits the exponential explosion of our search when possible. Second, we *protect* edges that have been cut in previously rejected scenarios. If we have two scenarios that cut edges e_1 and e_2, respectively, and the e_1 scenario fails to find an MAF, then the e_2 scenario will not find an MAF by cutting e_1 so we *protect* e_1 to indicate this and ignore any scenario that would cut e_1. This prevents us from exploring duplicate edge sets and increases the chance of finding a non-branching edge cut scenario. When no non-branching sibling pairs remain, we select a sibling pair with a protected member, if possible, to capitalize on this effect. For further details see Section 8.2.4.

We have also extended our MAF algorithm to allow for reconciliation of multifurcating gene trees with the reference supertree (see Section 8.2.4). For such gene trees we define the *soft* SPR distance [62, 99] to be the minimum number of SPR operations required to transform the reference tree into some binary resolution of the gene tree. This definition accounts for the general assumption that multifurcations imply uncertainty rather than simultaneous speciation. Our algorithm proceeds similarly to the binary case (as the reference tree, required to be T_1, is binary) with modifications to our considered edge scenarios that allow the resolution of multiple siblings and cutting the resulting edge.

The cluster reduction of Linz and Semple [63] splits the input trees into smaller subproblems that can be solved iteratively (but not independently). As our algorithms' running times scale exponentially with the computed distance, this reduction has an enormous impact in practice. Two subtrees of the input trees on the same leaf sets represent a cluster. A cluster MAF with its root edge removed (representing a transfer prior to the LCA of the leaf set) is guaranteed to be part of some complete MAF of the two trees, if any such cluster MAF exists. Alternatively, if every MAF of the cluster must maintain its root edge, every cluster MAF will be part of a complete MAF. We thus modified our search strategy to prefer MAFs with their root edge removed in order to accommodate this reduction. In addition, we removed the complicated weighting scheme of the original cluster reduction method and improved the time required to compute such a cluster reduction to linear in the size of the trees from the cubic scaling reported by Linz and Semple (see Section 8.2.5).

Recently, Chen and Wang proposed a separate improvement to our previous SPR distance algorithm for binary trees called UltraNet [32]). We do not compare our algorithms with UltraNet in detail as UltraNet requires binary trees and failed to find the correct SPR distance in 30 of our tests. However, our improved algorithm for the SPR distance even without the cluster reduction was significantly faster than UltraNet and our previous algorithm with clustering outperformed UltraNet on 65 of our tests.

8.2.2 Supertree Construction

We attempt to find the minimal SPR supertree for a given set of gene trees, that is, the binary rooted tree on the union of the label sets of the gene trees with the minimal cumulative SPR distance to the gene trees (hereafter, simply minimal SPR distance). When the leaf set of the (partially constructed) supertree differs from that of a gene tree, we ignore unique taxa when computing this distance. If no starting tree is provided to initiate the search, we construct an initial SPR supertree through stepwise addition of taxa and then use global SPR rearrangements to optimize the tree. To construct the initial tree, we begin with the four most common taxa in the input trees and select the tree shape on these four taxa with minimal SPR distance to the projected input trees. We then successively add taxa to the supertree, in decreasing order according to the frequency of occurrence in the gene trees. Each taxon is added in the location that minimizes the SPR distance. When determining this location, we only compute the SPR distance to gene trees containing the new taxon, as the SPR distance between the supertree and other gene trees is unchanged. Once we have constructed an initial SPR supertree (or, alternatively, are supplied an initial tree by the user) we begin the SPR rearrangement phase. For a prespecified number of iterations, we look at the $O(n^2)$ trees that can be obtained from the current supertree of n leaves by one SPR operation and select from these the tree with minimal SPR distance. Many of these SPR rearrangements will be obviously flawed, so we incorporate a bipartition clustering approach to ignore such rearrangements. Any bipartition of the supertree that is supported by at least half of the gene trees containing two or more taxa from each of the two sets induced by the bipartition is considered "fixed", and SPR rearrangements that disrupt it are disallowed. This greatly decreases the number of considered bipartitions with little effect on the accuracy of the tree search.

Our methods were developed for rooted gene trees, but we provide three options to accommodate the unrooted gene trees that are typically produced by maximum-likelihood and Bayesian phylogenetic approaches. Our first method is to compute the minimal SPR distance between the supertree and any rooting of each gene tree using an exhaustive search of all possible rootings. Second, given a rooted (partial) supertree and unrooted gene tree we use each bipartition of the gene tree to try and identify the root bipartition of the supertree. We root the gene tree at the bipartition that best matches the supertree root bipartition according to the balanced accuracy score, an average of the similarities between each matching side of the bipartitions. Suppose that the supertree root bipartition splits the taxa into two groups A and B and a gene tree bipartition splits the taxa into two groups C and D. Then the balanced accuracy of the $C|D$ bipartition as compared to the $A|B$ bipartition is the larger of

$$\frac{|A \cap C|}{2(|A \cup C|)} + \frac{|B \cap D|}{2(|B \cup D|)} \text{ or } \frac{|A \cap D|}{2(|A \cup D|)} + \frac{|B \cap C|}{2(|B \cup C|)},$$

depending on whether A and C or A and D are more closely matched. Third, we can root the gene trees at a set of predetermined outgroup taxa, throwing away trees where this outgroup is not monophyletic. We then build a supertree of this reduced tree set and can then, if desired, root the remainder of the trees using our balanced accuracy approach to build a final supertree.

8.2.3 Comparative Evaluation and Data Sets

We evaluated the performance of our SPR supertree algorithm against two other approaches: the widely used matrix representation with parsimony (MRP) approach of Baum [8] and Ragan [80] and the recently published Robinson-Foulds (RF) supertree algorithm [6]. Since the RF supertree approach is also based on topological distances between trees, it is an appropriate comparator for our SPR-based method. To construct MRP supertrees we used the Clann 3.2.2 [36] software package to generate matrices for a PAUP* version 4.0b10 [94] parsimony search using 25 iterations of SPR rearrangements (to match the SPR and RF approaches). RF supertrees were constructed using version 1.8.4 of the software described by Bansal et al. [6] which uses 25 iterations of SPR rearrangements interleaved with partial data ratchet iterations. The three methods were compared in terms of their running time on various datasets as well as their accuracy, either against the known phylogeny in the case of simulated data sets or the three supertree criteria when empirical data sets were used.

Unless otherwise specified, our experiments were performed on a 3.16Ghz Xeon system with 4GB of RAM and running CentOS 2.6 Linux in a Rocks 5.1 cluster. Our code was compiled using gcc 4.4.3 and optimization -O2.

We built simulated data sets to evaluate the accuracy of SPR, MRP and RF on gene trees generated from a completely known species history. EvolSimulator [11] version 2.2 was used to generate 15 replicated speciation and extinction histories in populations limited to 25 extant genomes. 10,000 simulation iterations were run in all cases. For each of the 15 distinct histories, multiple runs were carried out in which the rate of LGT was varied between 0 (no LGT) and 2.5 events per iteration in increments of 0.1. We also simulated two different LGT regimes: random, in which transfers between any donor/recipient pair were equally probable; and divergence-biased, where donor/recipient exchanges were more likely between closely related genomes (i.e., genomes that share a recent common ancestor), with no LGT at all between genomes that diverged less than 5000 generations in the past. The ancestral genome in each simulation (i.e., iteration 1) had 150 genes, and lineages could gain and lose genes to a minimum of 100 and a maximum of 200. The resulting gene trees were used to infer supertrees under the SPR, MRP and RF criteria: supertree accuracy was evaluated based on dissimilarity with the known species tree, and the total distance between the supertree and all gene trees.

We also compared the three methods using published eukaryotic supertree datasets of marsupials [27], seabirds [57], placental mammals [9] and papilionoid legumes [109] obtained from `http://www.cs.utexas.edu/~phylo/datasets/supertrees.html`. These datasets cover between 121-558 taxa in 7-726 trees and were used to compare the supertree methods according to their respective supertree optimization criteria, as was done by Bansal et al. [6].

Finally, we constructed a 244-taxon bacterial SPR supertree from a 40,631-tree subset of the 159,905 unrooted multifurcating prokaryotic phylogenetic trees from

Beiko [10], compared it with an MRP supertree and used the SPR supertree to infer "highways of gene sharing", that is, frequently implied pathways of LGT among major bacterial lineages. From the 1179 taxa in the original dataset, we randomly selected 15 Alphaproteobacteria, Betaproteobacteria and Deltaproteobacteria, 14 Epsilonproteobacteria, 13 Gammaproteobacteria, 40 Bacilli, 34 Clostridia, 74 Actinobacteria, 2 Deferribacteres, 11 Thermotogae, 7 Aquificae, 2 Nitrospira and 2 Synergistetes for a total of 244 taxa (listed in Supplemental Table A.1) covering a subset of well-sampled and sparsely sampled classes of bacteria and restricted the 159,905 trees to this subset. We then collapsed all branches with a bootstrap support value of less than 0.8 and discarded all star trees and trees with fewer than 4 taxa. After this procedure, 40,631 trees remained. In total, there were 393,876 leaves in the trees for an average of 9.7 taxa per tree. To construct a supertree from the set of unrooted gene trees, we used our rooting method described above with the Aquificae as outgroup. Unlike our other results, this supertree construction was performed on a 12-core 2.93Ghz Xeon system with 12GB of RAM and running Ubuntu 12.04.2 LTS Linux. We used all 12 cores in parallel with openMP to quickly compute aggregate SPR distances. We first constructed an initial guiding supertree from the 40 largest gene trees that contained a monophyletic Aquificae group [47]. This required 13 global rearrangement iterations and 87 CPU hours to converge on a local minimum. The remaining trees were then rooted using our balanced accuracy approach, and we constructed our SPR supertree from this data set using the guiding supertree as a base, which required 16 iterations to converge and 1198 CPU hours.

Once the final supertree was obtained, LGT events were inferred using MAF comparisons between our SPR supertree and the gene trees. We computed a single MAF for each gene tree and determined the equivalent sequence of implied LGT events in less than one minute. Transfers where both the putative donor and recipient were contained within two distinct genera were counted, and the results visualized as a heatmap and LGT affinity graph constructed using Cytoscape 2.8.3 [88]. We ignored directionality as it is often possible to identify partners but not the direction of transfer [12]. Heatmap values were scaled such that each row had a mean of 0 and standard deviation of 1 and relationships with fewer than 5% of the maximum transfer events for a row or only a single transfer event were filtered out. Two genera were connected by an edge if the number of inferred LGT events between them exceeded 5% of the total number of homologous genes common to at least one member of both genera.

8.2.4 Fast MAF Algorithm

In this section we discuss the efficiency and practicality improvements of our new MAF algorithm. We first introduce our previous algorithm [100, 102] (Chapter 3) whose running time is bounded by $O\left(2.42^k n\right)$ for two binary trees with n leaves and an SPR distance of k. We then introduce our novel concept of "protecting" edges during the search for an MAF. This "edge protection" scheme allows us to avoid exploring the same edge cutting scenarios multiple times and greatly speeds up the search for an MAF, as we demonstrated in Figure 8.9. In a forthcoming paper (Chapter 6) we give the full details of this algorithm and prove that its running time is bounded by $O\left(2^k n\right)$. Finally, we explain how we extended our algorithm to compute MAFs of a binary and multifurcating tree and thereby account for uncertainty in the gene trees input to our supertree method. In a recently submitted manuscript [99] (Chapter 5) we gave the full details of this algorithm as applied to two multifurcating trees and proved that its running time remains bounded by $O\left(2.42^k n\right)$. However, by requiring that one tree be binary and applying edge protection our new MAF algorithm requires roughly the same time in practice to compute an MAF regardless of whether the other tree is multifurcating, as we demonstrated in Figure 8.9.

Previous MAF Algorithm

Our previous MAF algorithm [100, 102] (Chapter 3) takes two binary trees T_1 and T_2 as input along with a parameter k and returns an agreement forest with at most $k + 1$ components (and thus k edge cuts) if and only if such an agreement forest exists. To find an MAF, we run this algorithm with increasing values of k from 0 until an agreement forest is found. Since the running time of the algorithm scales exponentially with k, this entire procedure only takes a small constant factor more time than the invocation that finds the MAF. Our algorithm proceeds in a bottom-up fashion from the leaves of T_1. T_1 remains a tree through this procedure but T_2 may become a forest, denoted F_2. We maintain a set of sibling pairs, sibling subtrees (a, c)

in T_1 such that identical subtrees a and c exist in F_2. The algorithm examines each such sibling pair in turn and applies one of three cases:

1. If a and c are also siblings in F_2, then the subtree rooted at their parent is identical in T_1 and F_2 and so becomes a candidate for membership in a sibling pair,

2. If a or c is a component of F_2 then it must be cut off in T_1,

3. We identify at most 3 sets of edges in F_2 such that cutting one of these edge sets will lead to an MAF and try each edge set recursively in turn.

Case 3, which defines multiple edge sets to consider for cutting, requires detailed explanation. Assume that a is the deeper subtree of F_2, if a and c are in the same component, and let b be the sibling of a in F_2. If a and c are in separate components of F_2 then cutting off a or c will lead to an MAF. If a and c are in the same component but only one subtree, b, is on the path between them then cutting off b will always lead to an MAF. Otherwise, cutting off a, c, or simultaneously cutting off all of the subtrees between a and c in F_2 will lead to an MAF. Note that this last case is the worst case of our algorithm as it splits our computation into three branches cutting one, one, or at least two edges respectively. We previously showed in [100] (Chapter 3) that this last case bounds our running time of $\mathrm{O}\left(2.42^k n\right)$ with a recurrence relation analysis.

New MAF Algorithm

Our improved algorithm introduces the concept of edge protection to alleviate the bottleneck of the 3-way branching case of our previous MAF algorithm. Observe that if some MAF can be found by a recursive invocation of this case that cuts off subtree a in F_2 then an MAF will be found by this invocation. Thus, we can assume that cutting off subtree a does not lead to an MAF in the recursive invocation that cuts off subtree c, or we would have already found it. We protect edge a in this search branch to denote this and ignore any recursive invocations that cut a protected edge. By ignoring these search paths we reduce the running time of the algorithm to $\mathrm{O}\left(2^k n\right)$. The proof of this bound is highly technical, as it relies on showing that this edge protection either forces our best case, cutting subtree b without branching, or avoids enough search branches to achieve this bound and requires some additional boundary cases. In a forthcoming paper (Chapter 6) we will provide the full details of our algorithm and prove this bound.

We have also developed a theory for MAFs of multifurcating trees to incorporate uncertainty in gene trees. In a recently submitted manuscript [99] (Chapter 5) we developed a general MAF algorithm for two multifurcating trees. This algorithm is based on our $\mathrm{O}\left(2.42^k n\right)$ algorithm for binary trees and achieves the same running time but is significantly more complicated and requires many more cases. For the purposes of constructing SPR supertrees, however, we only need to allow that the gene trees be multifurcating; the supertree is binary. By requiring that T_1 be binary in our MAF algorithm these extra cases disappear and we can use the same overall algorithm structure but with the ability to resolve multifurcations as well as cut edges. Our MAF algorithm when T_2 is multifurcating still examines each sibling pair in turn and applies one of three cases:

1. if a and c are also siblings in F_2, then either the subtree rooted at their parent is identical in T_1 and F_2 and so becomes a candidate for membership in a sibling pair or we resolve the multifurcation of their parent in F_2 to separate them so that this occurs.

2. If a or c is a component of F_2 then it must be cut off in T_1.

3. We identify at most 3 sets of edges in F_2 such that cutting one of these edge sets will lead to an MAF and try each edge set recursively in turn.

We again assume that a is the deeper subtree of F_2, if a and c are in the same component. Since F_2 is multifurcating, a may now have multiple siblings and we represent them collectively by B which we call a pendant subtree. If a and c are in separate components of F_2 then cutting off a or c will again lead to an MAF. If a and c are in the same component separated only by B then either cutting off c or resolving B separately from a and cutting the introduced edge will lead to an MAF. Otherwise, cutting off a, c, or resolving and cutting off all pendant subtrees of the path from a to c in F_2 will lead to an MAF. We further apply edge protection to this last case as in our improved binary algorithm. Note that this procedure is essentially identical to our prior binary algorithm with the exception that our previous best case, where we could bring a and c together in F_2 with a single cut now requires us to branch into two possibilities. Fortunately, cutting off c is never necessary when a's parent is binary, that is, B is a single node b, so this has a negligible running time impact in practice, as we demonstrated in Figure 8.9. This does, however, preclude the argument we used to prove that edge protection reduces the running time of the binary MAF algorithm to $\mathrm{O}\left(2^k n\right)$ so the running time of our MAF algorithm when one tree is multifurcating remains $\mathrm{O}\left(2.42^k n\right)$ in the worst case.

8.2.5 Linear-time Cluster Reduction

In this section we explain how to accelerate the computation of MAFs (and, thus, the SPR distance) using the Cluster Reduction of Linz and Semple [63]. This reduction partitions the input trees into pairs of subtrees, or clusters, that can be solved iteratively and reassembled into a full solution. The time required to solve these clusters with our MAF algorithms scales exponentially with the maximum number of components in an MAF of any cluster rather than the full MAF of the trees so this strategy greatly accelerates the recovery of MAFs in practice. The Cluster Reduction as originally formulated is only suitable to compute an MAF variant, weighted MAFs, that cannot be computed with our algorithms. We first extend the Cluster Reduction to apply to ordinary MAFs and then show how to identify clusters in linear time, greatly improving on the previous cubic time algorithm.

Linz and Semple defined a *cluster* of two trees T_1 and T_2 to be a pair of subtrees T_1^e and T_2^f, for appropriate edges e in T_1 and f in T_2 such that both trees have the same set of labelled leaves. A cluster sequence of T_1 and T_2 is a sequence of tree pairs $T = (T_1^1, T_2^1), (T_1^2, T_2^2), \ldots, (T_1^t, T_2^t), (T_1^\rho, T_2^\rho)$ defined inductively as follows: if $t = 0$, then $T_1^\rho = T_1$ and $T_2^\rho = T_2$. If $t > 0$ then (T_1^1, T_2^1) is a cluster of T_1 and T_2 with at least two taxa, the roots of T_1^1 and T_2^1 are labelled with a new label ρ_1, and $(T_1^2, T_2^2), \ldots, (T_1^t, T_2^t), (T_1^\rho, T_2^\rho)$ is a *cluster sequence* of the two trees obtained from T_1 and T_2 by replacing the subtrees T_1^1 and T_2^2 with a single labelled leaf a_1. This is illustrated in Figure 8.2. Clearly, ρ is the root of T_1^ρ and T_2^ρ. An agreement forest F of T is the disjoint union of forests $F = F_1 \cup F_2 \cup \ldots \cup F_t \cup F_\rho$, where F_i is an agreement forest of T_1^i and T_2^i, for all i in $\{1, 2, \ldots, t, \rho\}$. The weight of F

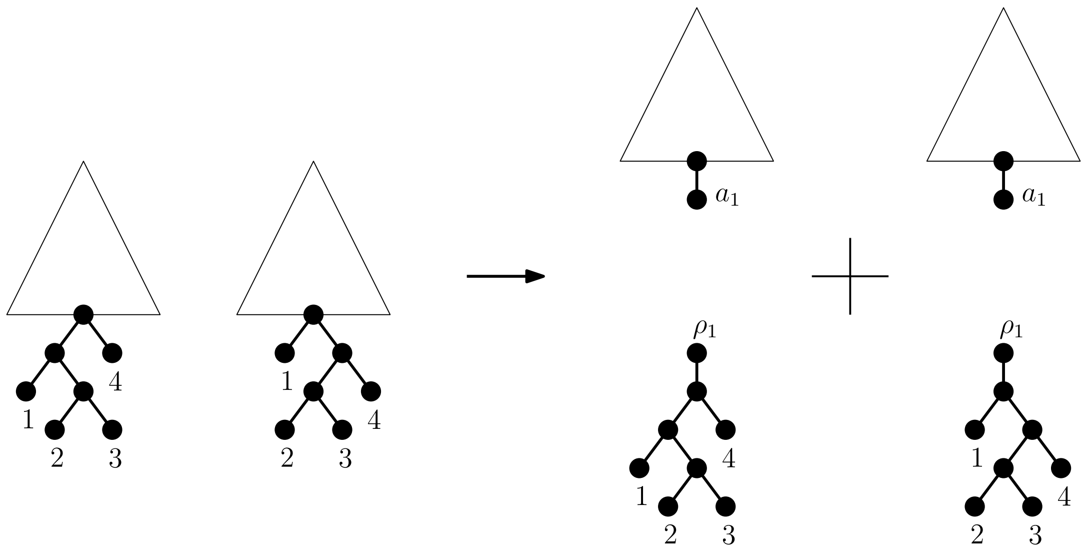


Figure 8.2: One application of the cluster reduction. The subtrees on leaves 1, 2, 3, and 4 are not identical but cover the same leaf set. Thus, they can be split from the trees and solved independently. The original locations of the removed subtrees are represented by a new leaf a_1 and their roots are labelled ρ_1. It is preferable to cut ρ_1 in any sub-MAF as we can then cut the equivalent edge above a_1.

is defined to be $w(F) = |F| - |\{(p_i, a_i) : p_i \text{ and } a_i \text{ are singletons in } F\}| - t$, where $|F|$ denotes the number of trees in F. We say that F is an MAF of T if it has minimum weight among all agreement forests of T. The key result proved by Linz and Semple is that the weight of an MAF of any cluster sequence is exactly the number of components in an MAF of the original trees. They also provided a divide-and-conquer approach for computing an MAF of T: Process the clusters in order, for each i computing an agreement forest F_i of T_1^i and T_2^i. If $F = F_1 \cup F_2 \cup \ldots \cup F_{i-1}$ is the union of forests computed so far (for $i = \rho$, let $i - 1 = t$), then F_i is computed to be an agreement forest of T_1^i and T_2^i that minimizes $w(F_i) = |Fi| - |\{(\rho_j, a_j) : \rho_j \text{ is a singleton in } F \text{ and } a_j \text{ is a singleton in } F_i\}|$. This weight corrects for the fact that we have cut the same edge twice; ρ_j and a_j are nodes introduced by the cluster reduction to represent the intersection of two clusters so the edge below ρ_j and above a_j are the same edge. Thus, for $i \neq p$, we choose F_i to be an agreement forest of T_1^i and T_2^i that minimizes this weight and such that ρ_i is a singleton, if possible, to capitalize on this correction. The final forest defined in this way is an MAF of T_1

and T_2.

We used the key observation of the cluster reduction, that it is best to cut the root edge of each cluster when possible, to modify this procedure to compute unweighted MAFs. We first compute the cluster sequence as above. We then apply a modification (described below) of our MAF algorithm that returns an MAF of the current cluster such that it has the root edge cut if and only if any MAF of the current cluster i has an isolated ρ_i. If the root edge, below ρ_i, was cut in this MAF then we separate the two clusters by simply cutting the edge above a_i in its corresponding cluster and then removing a_i and ρ_i completely to avoid counting this double cut. If the root edge is not cut then we reattach the two clusters by cutting this root edge, removing ρ_i, and then replacing a_i with the subtree formerly rooted by ρ_i (thereby removing this subtree from the agreement forest of the current cluster). We apply this procedure iteratively to the cluster sequence and then take the union of these forests as our MAF. We have removed each ρ_i and a_i so this is an unweighted MAF. To see that this is indeed an MAF, observe that we apply the same procedure as Linz and Semple for each cluster other than our treatment of ρ_i and a_i. If ρ_i is not isolated in a given cluster, then we remove one component from our forest by replacing a_i, whereas the weighted algorithm applies a weight of -1 (from the $-t$ factor) to compensate. If ρ_i is isolated in a given cluster then we remove ρ_i (equivalent to the -1 weight) and remove a_i (equivalent to the singleton portion of the weight calculation, this reduces the weight by 1 if p_i and a_i are singletons in some weighted MAF). Thus, our computed forest has exactly as many components as the weight of some weighted MAF and is indeed an MAF.

We now explain how we modified our MAF algorithm to prefer MAFs with isolated roots. Recall that each recursive step of our algorithm identifies at most three edge sets to cut from the intermediate forests and tries each edge set in turn. If more than one of these edge set choices lead to an MAF then our algorithm arbitrarily chooses one of them. We simply modified our algorithm to instead select between these at most three MAFs by preferring MAFs with their root edge cut. Since our algorithm does not find all MAFs of the two trees, it is not immediately obvious that this change is sufficient to find one MAF where the root edge is cut if such an MAF exists. However, the correctness proof of our previous MAF algorithm [102]

(Chapter 3) and our forthcoming correctness proofs start with an arbitrary agreement forest F and construct an agreement forest F' from F that has no more components than F and such that our algorithms find F'. If we choose F to be an agreement forest where ρ_i is a singleton, then this construction ensures that F' also contain ρ_i as a singleton. In other words, if there exists an MAF that has ρ_i as a singleton, our algorithms find one such MAF.

Finally, we developed a linear-time algorithm for computing a cluster sequence, greatly improving on the naïve cubic algorithm. Let n be the number of leaves in T_1 and T_2. The cubic algorithm compares each of the subtrees of T_1, starting at the leaves, to each subtree of T_2 and appends each found cluster to the cluster sequence. There are $\mathrm{O}(n)$ subtrees in each tree and it takes $\mathrm{O}(n)$ time to compare two leaf sets so this procedure requires $\mathrm{O}(n^3)$ time. We improve on this by using least common ancestors (LCAs). The LCA of two or more nodes in a tree is their common ancestor furthest from the root. Let s_1 be a subtree of T_1 with leaf set L_1 and s_2 be a subtree of T_2 with leaf set L_2. Observe that these subtrees have the same leaf set if and only if the LCA of L_1 in T_2 is s_2 and the LCA of L_2 in T_2 is s_1. Efficient least common ancestor (LCA) query structures exist (e.g., [15]) that can be built in $\mathrm{O}(n)$ time and that allow for constant time LCA queries of two nodes. We use such a structure to compute a mapping M of T_1 subtrees to the LCAs of their leaf sets in T_2. First, for each leaf x in T_1, we set $M(x)$ to the corresponding leaf x of T_2. Then, for any node n of T_1 with children c_1 and c_2 such that the mapping $M(c_1)$ and $M(c_2)$ have been defined, we compute $M(n) = \mathrm{LCA}(M(c_1), M(c_2))$. We apply this procedure again with T_1 and T_2 reversed to compute the mapping M^{-1} of T_2 subtrees to the LCAs of their leaf sets in T_1. Finally, for each subtree s_1 of T_1 in a bottom-up postorder traversal we check if s_1 is a cluster by checking if $M^{-1}(M(s_1)) = s_1$ and, if so, appending s_1 and $M(s_1)$ to the cluster sequence.

8.3 Results

8.3.1 Bacterial SPR Supertree and Large-Scale Analysis of LGT

We first present our supertree of 244 bacterial taxa that was constructed from 40,631 unrooted input gene trees using our two-stage outgroup procedure. The taxa selected

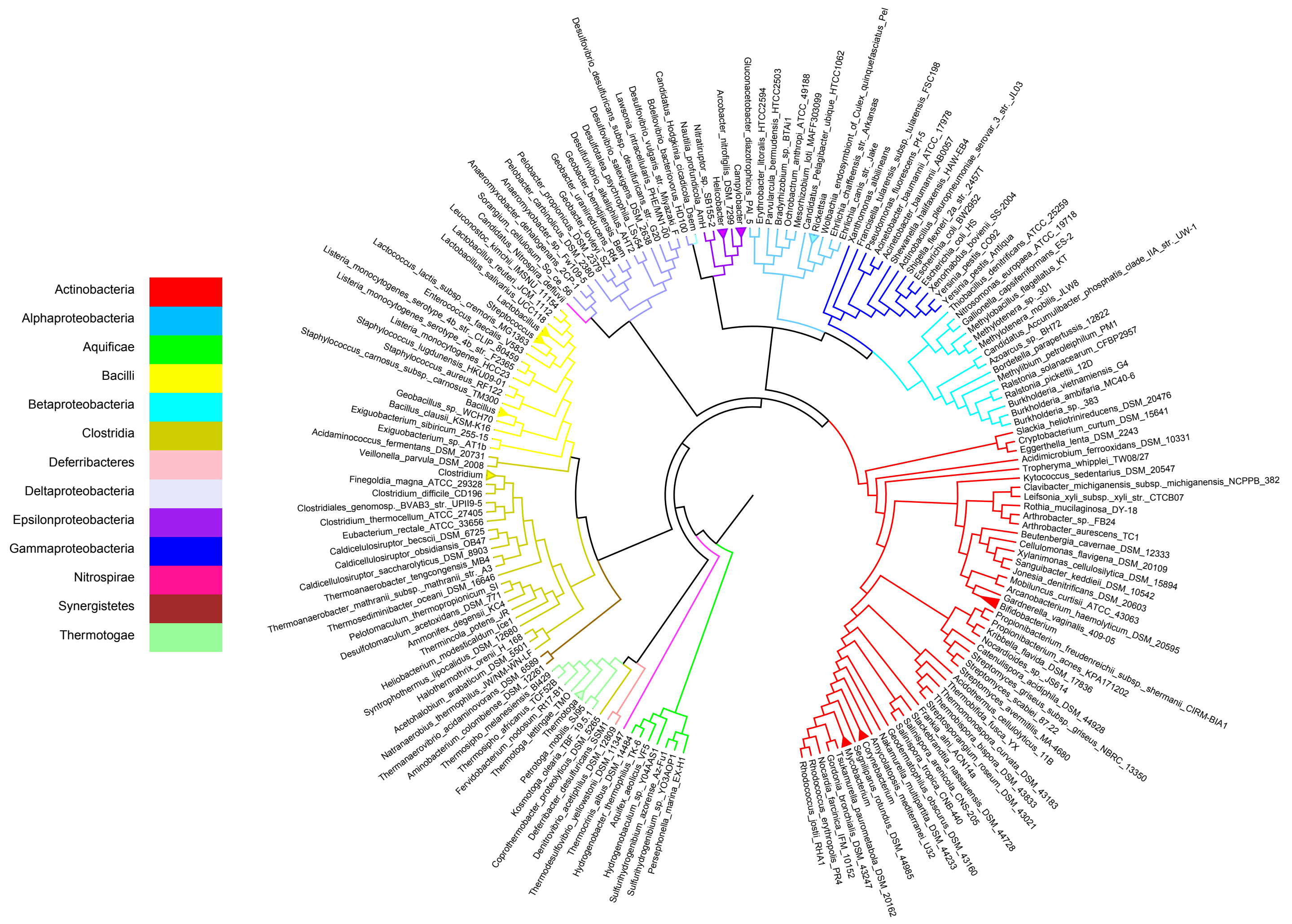


Figure 8.3: The SPR supertree constructed using Aquificae as outgroup. Genera such as *Mycobacterium* with multiple representatives are shown as collapsed subtrees for brevity. Colours indicate the classes of bacteria.

for our bacterial supertree analysis were chosen to examine several interesting phylogenetic questions in Bacteria. For example, there are two competing hypotheses for the placement of the Aquificae. Informational genes such as 16S small subunit ribosomal RNA suggest that the Aquificae are deep-branching and either external to or sister with the Thermotogae but the majority of other proteins suggest that the Aquificae are sister to the Epsilonproteobacteria (or other groups such as the Deltaproteobacteria) and not the Thermotogae [26]. It has been suggested that the Aquificae may be closely related to the Epsilonproteobacteria with either LGT or a thermophilic G+C bias and long-branch attraction responsible for the observed affinity for Thermotogae [47]. Informational proteins are thought to be transferred infrequently, so it has been more recently suggested that there have been large amounts of lateral gene transfer between the Aquificae and Epsilonproteobacteria [26]. Our dataset also includes members of many other groups implicated in LGT, including the Deltaproteobacteria and Clostridia: both of these groups show evidence of frequent LGT with other lineages [10, 37, 50]. Other genera frequently associated with high LGT rates including *Pseudomonas* and *Burkholderia* are also included. Finally, several lineages such as Deferribacteres and Synergistetes with relatively few sequenced representatives and an uncertain phylogenetic position [55] were included to assess their placements in the SPR supertree.

Figure 8.3 shows our SPR supertree of the 244-taxon bacterial dataset. The SPR supertree largely recovered the major bacterial classes as monophyletic groups with several notable exceptions. The Deltaproteobacteria are separated from the other Proteobacteria by the Actinobacteria. The Deltaproteobacteria are also split into a group containing the *Myxobacteria* and *Candidatus* “Nitrospira defluvii”, and a group containing all other orders of the class. Although assigned to phylum *Nitrospirae*, *Ca.* N. defluvii has strong affinities to other phylogenetic groups, with deltaproteobacterial genomes constituting seven of the 15 most frequently observed phylogenetic partners. This is an interesting link as *Sorangium cellulosum* has the largest known bacterial genome [87] and both *Candidatus* Nitrospira defluvii and *Anaeromyxobacter dehalogenans* are gram-negative nitrite reducers. Further, it has been suggested that *Ca.* N. defluvii evolved from microaerophilic or even anaerobic ancestors [60] and *Anaeromyxobacter dehalogenans* exhibits aerobic and anaerobic growth [86]. Two other proteobacteria are separated from their classes: *Bdellovibrio bacteriovorus*, a Deltaproteobacterium that parasitizes other gram-negative bacteria [91] and appears to have acquired genes from the protebacterial cells it parasitises [45], and *Candidatus* Hodgkinia cicadicola, an alphaproteobacterial cicada symbiont with the smallest known genome [70], form a pairing that is sister to the Epsilonproteobacteria.

Among other phylogenetic groups, *Veillonella parvula* and *Acidaminococcus fermentans*, initially assigned to class Clostridia, are sister to the Bacilli. *Veillonellaceae* and *Acidaminococcaceae* have a peculiar cell wall composition which stains Gram-negative, unlike most Firmicutes, and have been suggested to belong to a class Negativicutes, separate from the Bacilli and Clostridia, by Marchandin et al. [69]. *Coprothermobacter proteolyticus* groups with the Thermotagae rather than the Clostridia. *C. proteolyticus* was assigned to class Clostridia using small subunit ribosomal RNA [81] but phylogenomic analysis [10, 111] and newer phylogenetic trees built from many more samples of small subunit ribosomal RNA agree with a closer relationship between *C. proteolyticus* and Thermotogae [72]. With Aquificae as the outgroup, the next-deepest branches in the bacterial tree are *Thermodesulfovibrio yellowstonii*, the other member of phylum Nitrospirae, and the Deferribacteres, followed by Thermotogae. The Synergistetes are sister to the Firmicutes in this tree.

We then inferred LGT events between these bacteria by computing a single MAF for each gene tree and determining the equivalent sequence of implied LGT events. This entire analysis of the 40,631 gene trees required less than one minute using our refined MAF algorithms. Transfer events with source and endpoints both in a monophyletic subtree of the same genus or different genera were identified to focus on relatively recent transfers. Directionality was ignored as it is often possible to identify partners but not the direction of transfer [12]. Figure 8.4 shows the results of this analysis. Clustering based on the strength of their LGT affinities still groups most genera by class and phylum, and the majority of inferred LGT events occur within clusters of taxonomically related genera. However, there are also many linkages between genera of distinct phyla and clusters of genera with distinct classes and phyla. Figure 8.5 shows a heatmap of the relative LGT trends between classes.

A genus-level LGT affinity graph (Figure 8.6) between genera was used to further explore these relationships and identify paths of gene sharing between distinct

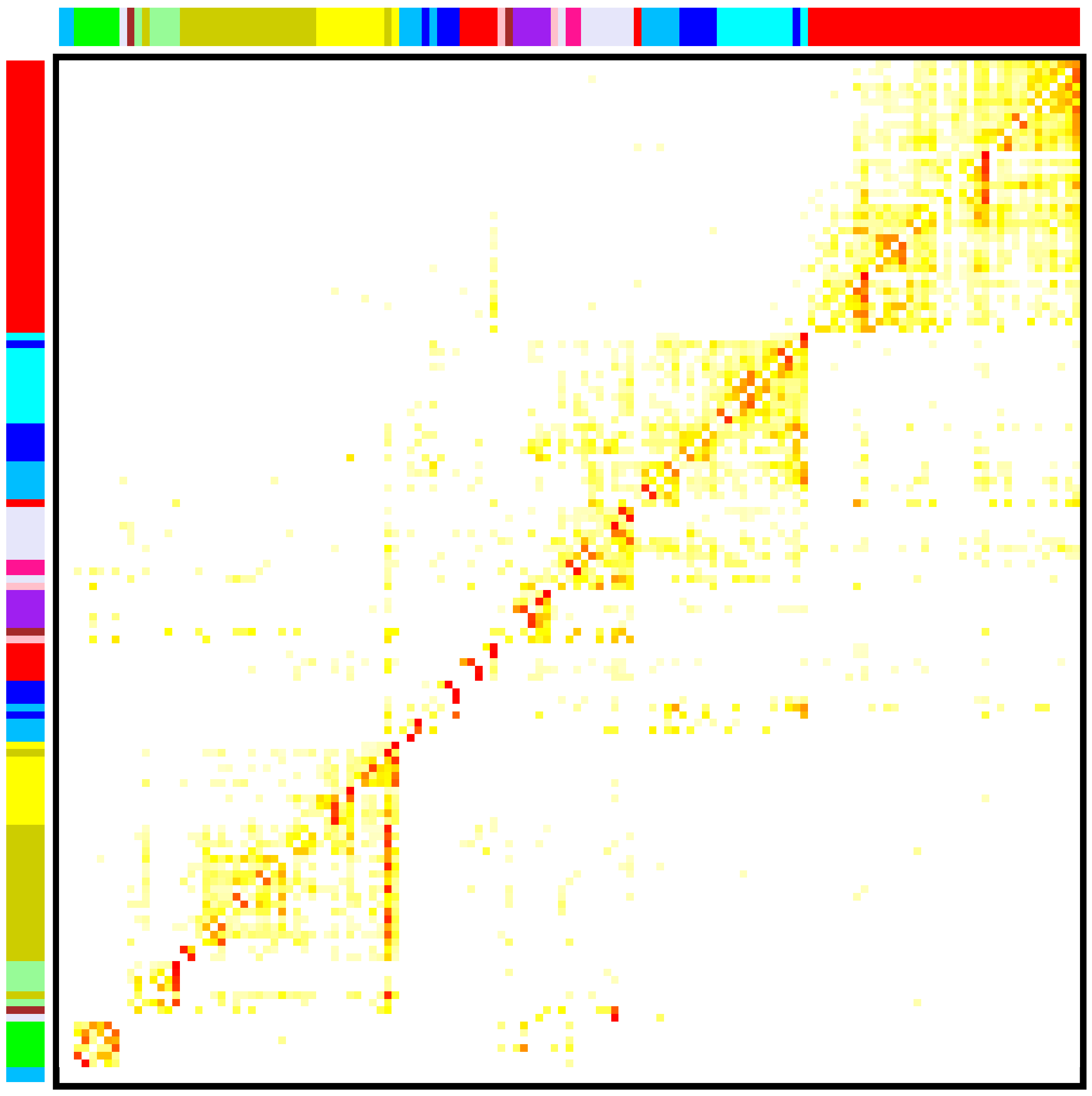


Figure 8.4: Inferred LGT events between 135 distinct bacterial genera as an LGT heatmap. The coloured side bars indicate class using the colour mapping of Figure 8.3. The row and column genus order is the same. The number of transfers is shown in a white-yellow-red colour scale with darker colours indicating a higher proportion of transfer events. Colour intensity is relative to the largest number of transfers in a row. Relationships with fewer than 5% of the maximum transfer events for a row or only a single transfer event were filtered out.

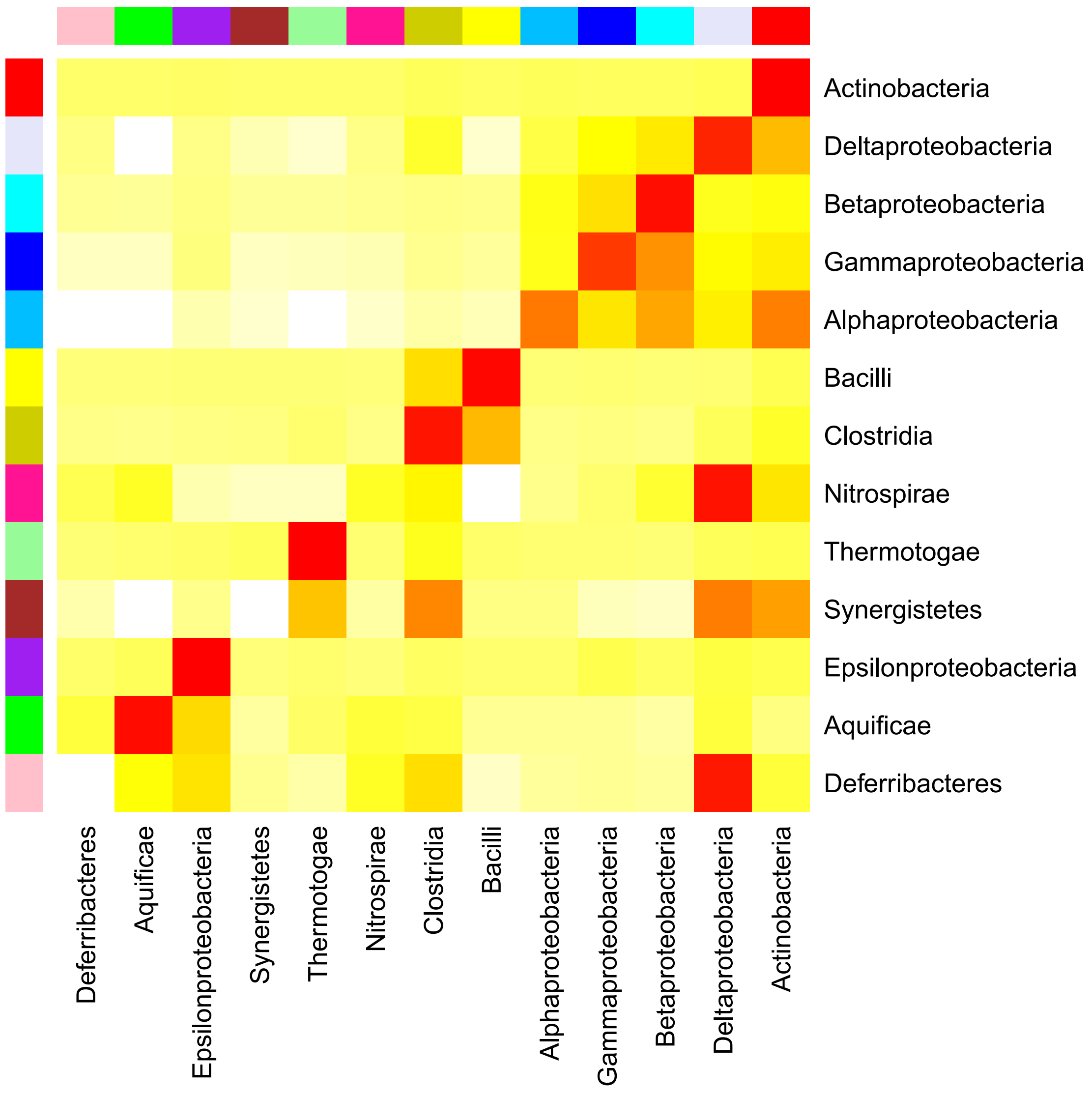


Figure 8.5: Inferred LGT events between 13 bacterial classes as an LGT heatmap. The colour side bars indicate class. The row and column order is the same. The number of transfers is shown in a white-yellow-red colour scale with darker colours indicating a higher proportion of transfer events. Colour intensity is relative to the largest number of transfers in a row. Relationships with fewer than 5% of the maximum transfer events for a row or only a single transfer event were filtered out.

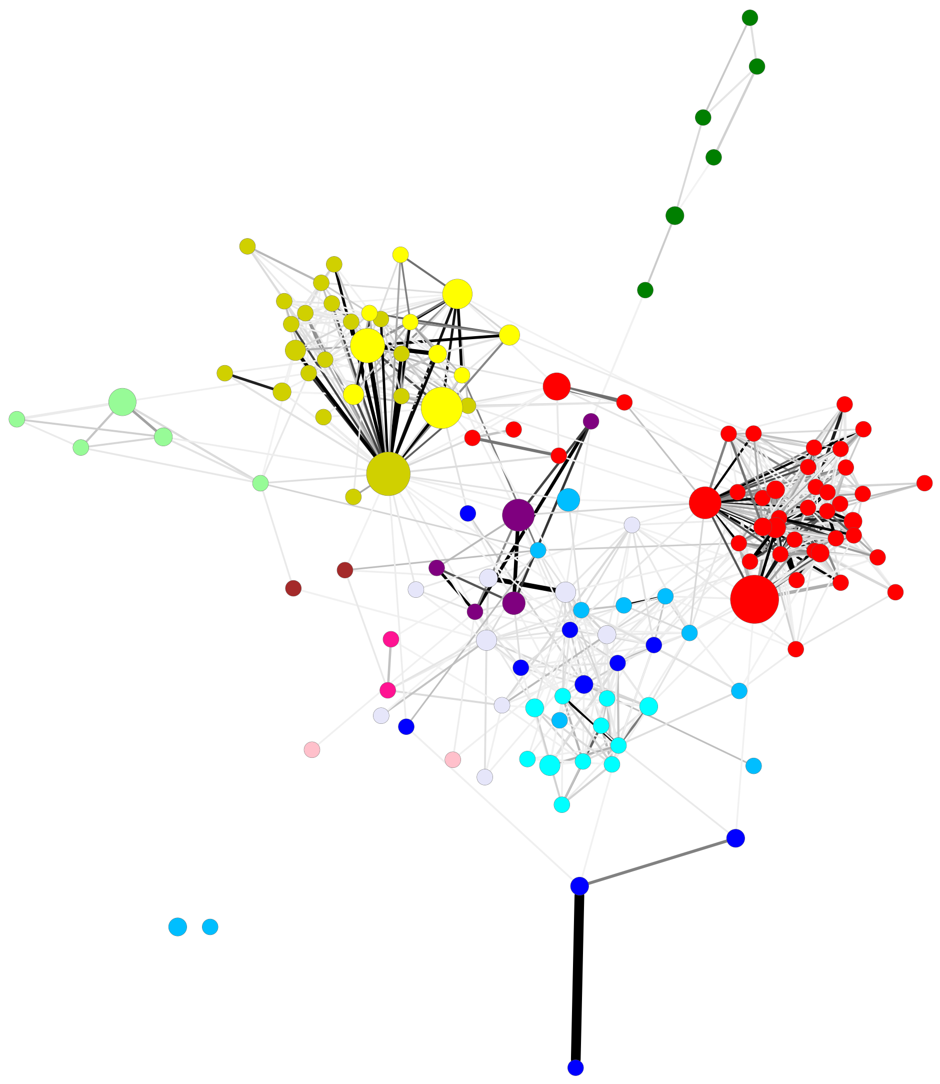


Figure 8.6: Inferred LGT events between 135 distinct bacterial genera as an LGT affinity graph. Each node of the graph represents a bacterial genus, colored by class and scaled relative to the number of genomes representing that genus (1-15). Two genera are connected by an edge if the number of inferred LGT events between them exceeds 5% of the number of homologous genes common to both genomes. The shade of an edge is proportional to this ratio of LGT events to common genome size; black edges indicate relationships with at least as many LGT events as the size of their common genome. The thickness of an edge scales relative to the actual number of inferred transfers (between 2 and 370) with thicker edges indicating more transfers. The graph is shown with a spring-loaded layout.

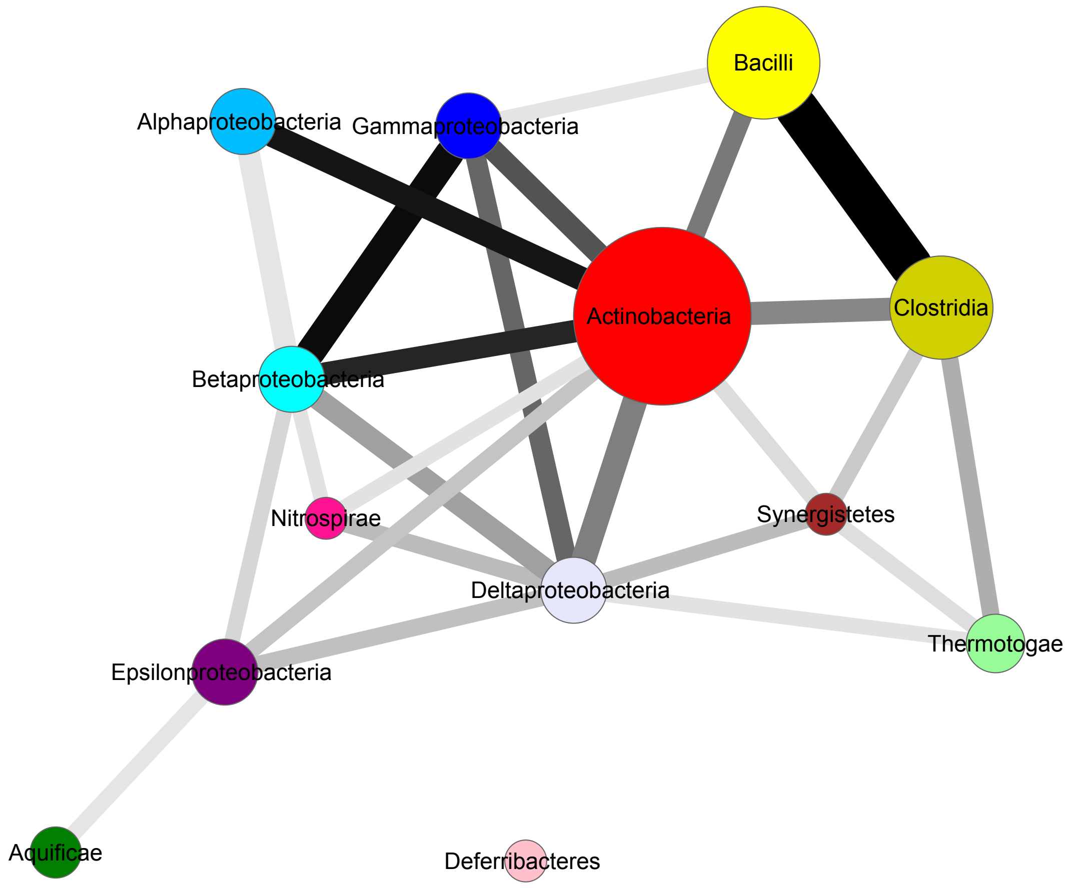


Figure 8.7: Inferred LGT events between 13 bacterial classes as an LGT affinity graph. Each node of the graph represents a bacterial class scaled relative to the number of represented taxa (2-75). Two genera are connected by an edge if the number of inferred LGT events between them exceeds 5% of their shared genes. The shade of an edge is proportional to this ratio of LGT events to shared genes; black edges indicate relationships with at least as many LGT events as shared genes. The thickness of an edge scales relative to the actual number of inferred transfers (30-1414) with thicker edges indicating more transfers.

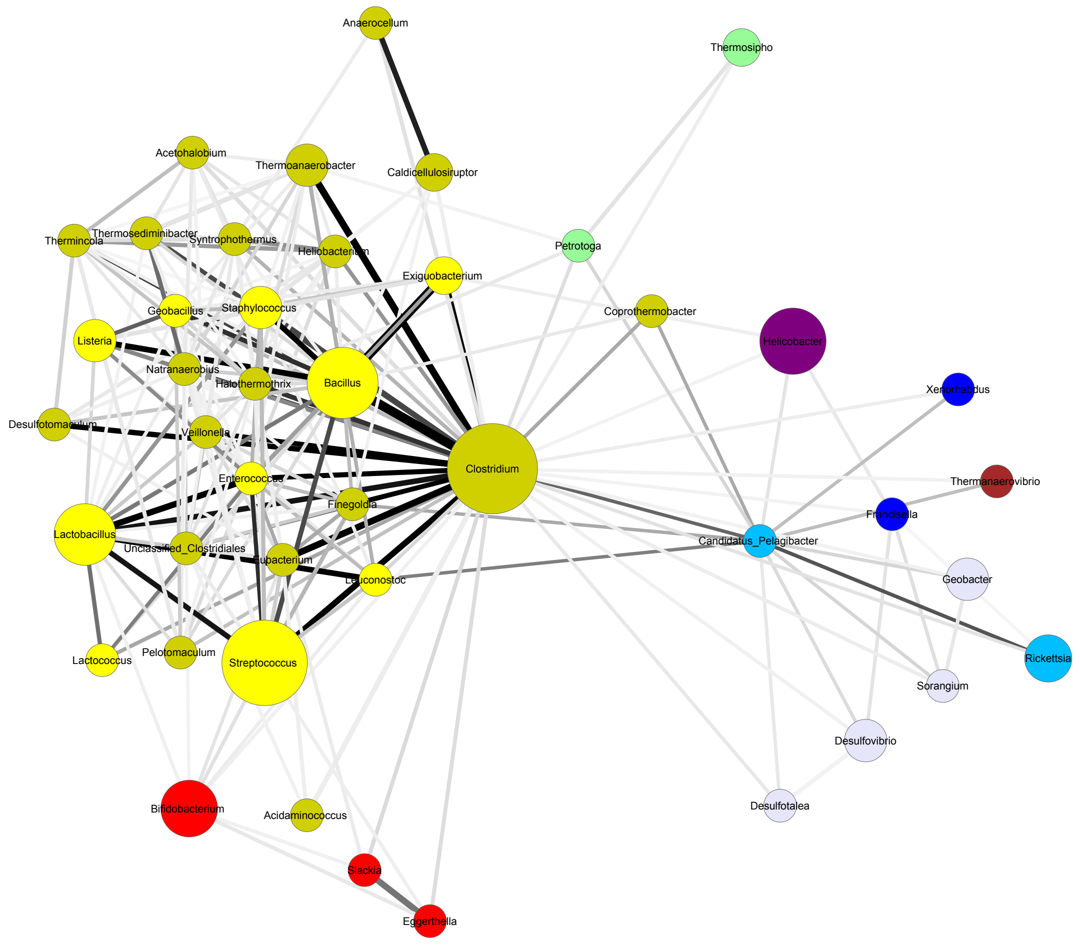
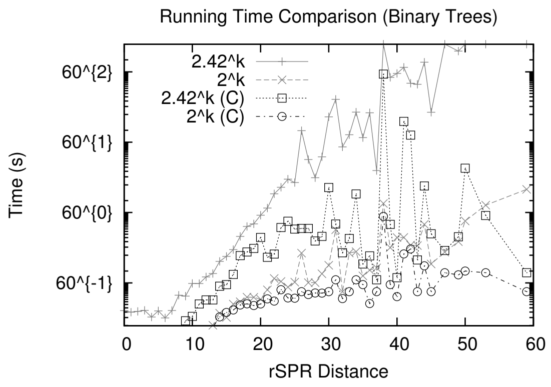


Figure 8.8: The LGT affinity neighbourhood of genus Clostridium. Each node of the graph represents a bacterial genus coloured by class and scaled relative to the number of represented taxa (1-13). Two genera are connected by an edge if the number of inferred LGT events between them exceeds 5% of their shared genes. The shade of an edge is proportional to this ratio of LGT events to shared genes; black edges indicate relationships with at least as many LGT events as shared genes. The thickness of an edge scales relative to the actual number of inferred transfers (2-125) with thicker edges indicating more transfers.

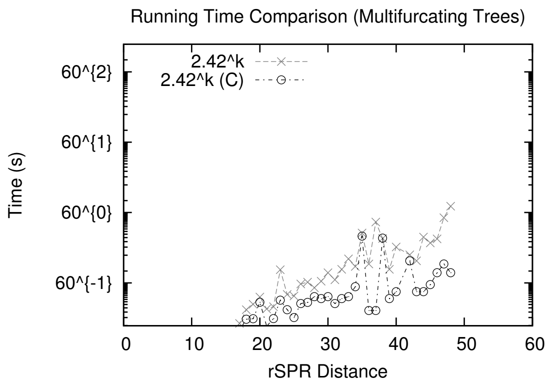
lineages. Genera were connected by edges representing transfer events exceeding 5% of their total number of shared homologous genes. As in Figure 3a, the majority of inferred LGT events connect members of the same class or phylum. Yet many linkages connect different classes and phyla such that all of the genera but two, *Ehrlichia* and *Wolbachia*, are connected. The large and diverse genus *Clostridium*, in particular, connects Actinobacteria, Thermotogae, four of the five classes of Proteobacteria, *Thermoanaerovibrio* (phylum Synergistetes), and has many strong connections with Bacilli and other Clostridia (Figure 8.8(b)). Family *Coriobacteriaceae*, comprising *Slackia*, *Eggerthella*, and *Cryptobacterium*, had linkages with the other Actinobacterial genera *Corynebacterium* and *Bifidobacterium* but was also connected to the Firmicute genera *Clostridium*, *Eubacterium*, and *Streptococcus*. There are numerous pathways of gene sharing between actinobacterial genera such as *Acidimicrobium*, *Corynebacterium* and *Mycobacterium* on the one hand, and proteobacterial genera such as *Helicobacter*, *Sorangium*, *Xanthomonas* and *Mesorhizobium* on the other. A single path between *Nitratiruptor* and *Persephonella* connects the Epsilonproteobacteria with the Aquificae. Many connections are observed between the different classes of Proteobacteria, highlighting the numerous LGT events that occur between distinct lineages of phylum Proteobacteria. The connectedness of higher taxonomic groups is supported by the class-level affinity graph (Figure 8.7, in which each class is connected to 3.92 other classes on average, with the Actinobacteria connected to a total of ten.

8.3.2 Validation of Efficiency and Accuracy

We next demonstrate the improved performance of our MAF algorithms with a single SPR distance analysis of our 244-taxon bacterial supertree as compared to each of the 40,631 gene trees. Figure 8.9 shows the mean running time for tree comparisons with a given SPR distance on a log scale. Our improved algorithms reduced the time required for individual calculations from 5 hours to a maximum of 0.8 seconds on the initial set of binary gene trees. Both the cluster reduction and our improved algorithms are necessary to achieve these running times. Our algorithm requires slightly more time to compare the supertree with multifurcating trees for a given SPR distance but this is balanced by the reduction in SPR distance caused by collapsing



(a)



(b)

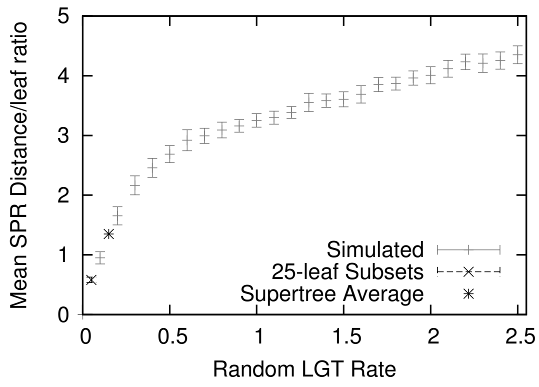
Figure 8.9: Mean time required to compare gene trees with a given SPR distance from an SPR supertree of a 244-genome dataset. The time axis is on a log scale as the time required increases exponentially with the SPR distance. The left panel compares our previous (2.42^k) and new (2^k) algorithms, with (C) and without clustering, on the set of binary trees. The right panel compares our new algorithm with and without clustering on the set of trees with unsupported bipartitions collapsed. Note that collapsing bipartitions reduces the SPR distance.

unsupported bipartitions; clustered comparisons required at most 0.76 seconds. As mentioned previously, a full LGT analysis now requires just 34 seconds on a single CPU. Without our new algorithms, such an analysis would be limited to binary trees and require more than 65 hours.

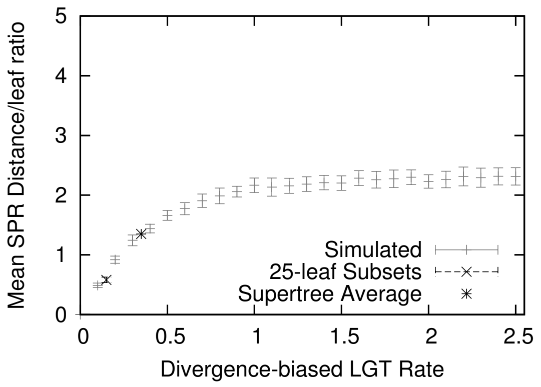
8.3.3 Validation with Simulated Datasets

We next compared the ability of SPR, RF, and MRP based supertrees to recover the species tree in a series of simulated datasets. EvolSimulator [11] was used to evolve sets of genomes under a model of lineage duplication and extinction, with each lineage capable of gene duplication, gene loss, and LGT. Varying the rate of LGT in different sets of replicated simulations allowed us to explore the effectiveness of SPR, RF and MRP at relatively low or high levels of LGT. We also simulated two regimes of LGT: random LGT, which can interfere with the recovery of correct branching patterns, and divergence-biased LGT, which can actually reinforce the true tree due to preferential sharing between close relatives [12].

Simulated LGT rates varied between 0 (no LGT) and 2.5 events per iteration (see Methods for details). To give context to our LGT rate simulation parameter, we



(a)



(b)

Figure 8.10: A comparison of our LGT rate simulation parameter to the bacterial dataset. Supertrees of empirical data have the same mean SPR distance to leaf ratio (within 95% confidence intervals) as our simulations with a random LGT rate less than 0.2 and a divergence-biased LGT rate less than 0.4.

computed the mean ratio of SPR distance to number of leaves in the simulated trees, to similar values inferred for the 244-taxon SPR supertree (Figure 8.10). The inferred frequency of LGT in our empirical data equated to a simulated random LGT rate between 0.1 and 0.2 and a simulated divergence-biased LGT rate between 0.3 and 0.4. Since the bacterial supertree has 244 leaves rather than 25, we also restricted our bacterial supertree and gene trees to 25 randomly sampled subsets of 25 leaves and computed this ratio. We found these subsampled supertrees corresponded to lower simulated rates of LGT. This suggests that our simulations with lower rates of LGT are biologically plausible; also, since the distribution of LGT events is non-uniform across bacterial lineages [14, 58, 96] the higher rates are likely to be relevant to the inference of some relationships in the supertree.

Having established the relevance of our simulated rates of LGT, we then assessed the ability of different supertree algorithms to recover the correct organismal history based on analysis of the gene trees. Figure 8.11 shows the mean SPR difference between the simulated species histories and the RF supertree, SPR supertree, SPR supertree seeded with an MRP starting tree, and SPR supertree seeded with the correct species tree. SPR supertrees were significantly more similar to the simulated species tree than RF supertrees for the LGT rates seen in our bacterial dataset and higher ($p < 0.05$ for random LGT rates of 0.2–1.4 and divergence-biased LGT rates

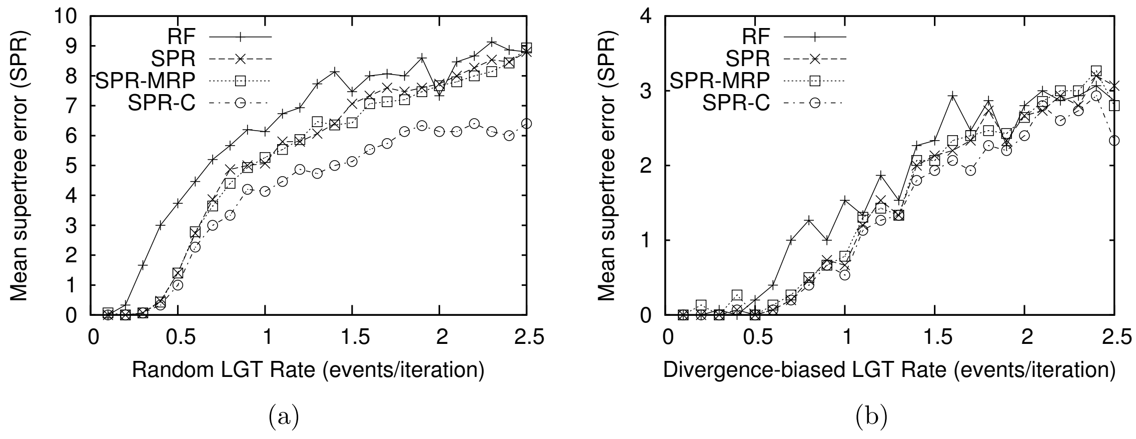
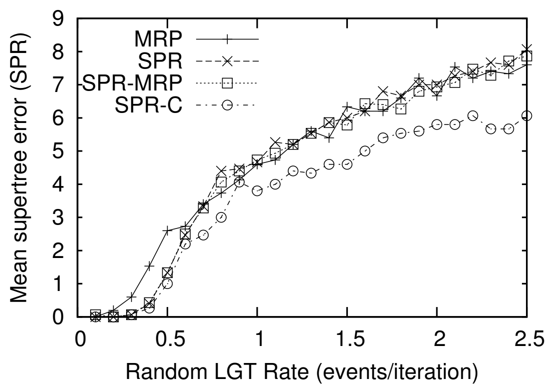


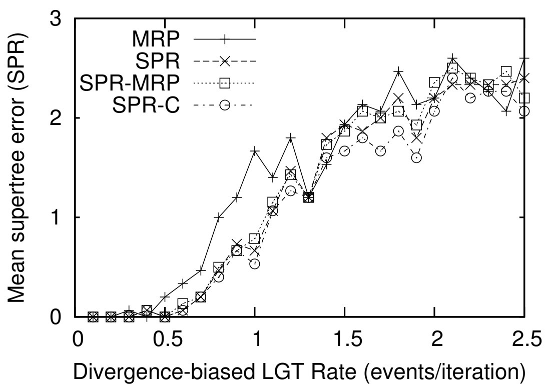
Figure 8.11: A comparison of the mean supertree error (as measured by the SPR distance) of RF supertrees (RF) to SPR supertrees using the default parameters (SPR), seeded with an MRP starting tree (SPR-MRP), or seeded with the correct tree (SPR-C).

of 0.7,0.8 and 1.0 with a 2-tailed paired students t-test; $p < 0.01$ for random LGT rates of 0.2–0.7, 0.9, 1.3, 1.4; the overall results were significant with $p < 10^{-5}$ for both types of LGT). Seeding the SPR supertree search with an MRP tree did not substantially change these results. Seeding the SPR supertree search with the correct tree does not substantially change the results for divergence-biased LGT or plausible rates of random LGT. We see that the SPR supertree and the simulated species tree diverge as the random LGT rate increases, even when seeded with the species tree. These results suggest that datasets with substantially higher rates of LGT than our bacterial data would require a better search strategy or a network-based analysis rather than a supertree.

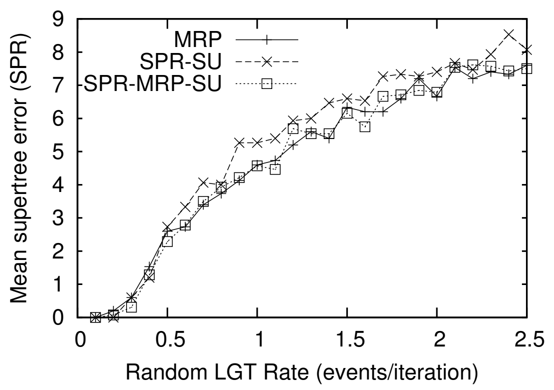
Figure 8.12 compares the accuracy of SPR and MRP supertrees. As MRP constructs unrooted supertrees, the error is measured here as the minimum SPR distance between the simulated species history and any rooting of the inferred supertrees. The upper panels of Figure 8.12 show the mean supertree error between the simulated species histories and the MRP supertree, SPR supertree, SPR supertree seeded with an MRP starting tree, and SPR supertree seeded with the correct species tree. The SPR supertrees were significantly more similar to the simulated species history than the MRP trees under biologically plausible rates of LGT ($p < 0.01$ for random LGT rates of 0.3–0.5 with a two-tailed paired students t-test; the divergence-biased results



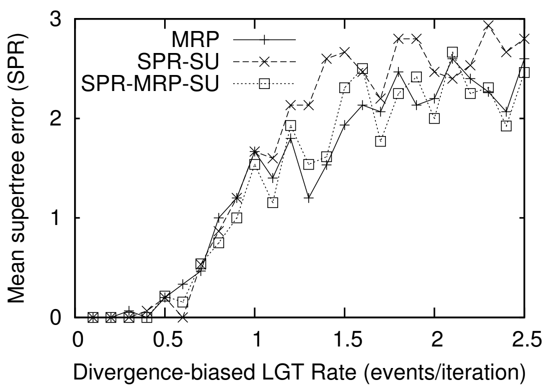
(a)



(b)



(c)



(d)

Figure 8.12: A comparison of the accuracy of SPR and MRP supertrees with known or unknown gene tree roots. The upper panels compare the mean supertree error (as measured by the minimal SPR distance to any rooting of a supertree) when the gene trees are correctly rooted. We compared MRP supertrees (MRP) to SPR supertrees using the default parameters (SPR), seeded with an MRP starting tree (SPR-MRP), or seeded with the correct tree (SPR-C). The lower panels compare the mean error of the MRP supertree to SPR supertrees when the gene tree roots are unknown, using our balanced accuracy based simple unrooted comparison without and with an MRP seed tree (SPR-SU and SPR-MRP-SU, respectively).

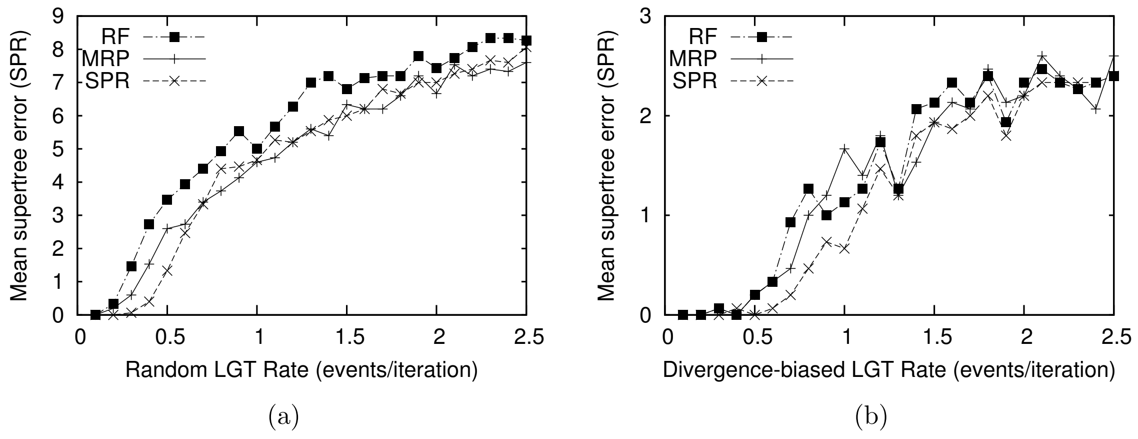


Figure 8.13: A comparison of the accuracy of SPR, RF and MRP supertrees as measured by the minimal SPR distance between simulated species histories and any rooting of the supertree under varying rates of random or divergence-biased simulated LGT events.

were not significantly different for individual rates other than 0.6 and 1.0 due to the small supertree error but were significantly better overall with $p < 0.001$). At higher simulated rates of LGT the accuracy of SPR supertrees matches that of the MRP trees. We observed that this occurs when the accuracy of the SPR supertree and the SPR supertree seeded with the correct tree diverge, suggesting that a better search strategy may improve these results. We also examined the accuracy of RF supertrees with this unrooted measure and found similar results to the unrooted comparison, that is, SPR supertrees and MRP supertrees were both significantly more similar to the simulated species tree than the RF supertrees (Figure 8.13). The lower panels of Figure 8.12 show the mean supertree error between the simulated species histories and the MRP supertree and SPR supertrees using our balanced accuracy based simple unrooted comparison without and with an MRP seed tree. The accuracy of our SPR supertrees when the gene tree roots are unknown matches that of the MRP trees for plausible rates of LGT but the performance of our SPR supertrees declines with increasing rates. Using an MRP seed tree prevented this decline which suggests that our initial tree construction step is not well suited to gene trees with unknown roots. Developing an improved method for building starting trees from unrooted gene trees could improve these results.

8.3.4 Comparison with MRP and RF Supertrees on Eukaryotic Datasets

Bansal et al. [6] validated their RF supertree approach on a series of eukaryotic datasets that varied substantially in the number of input trees and total number of taxa. We compared the accuracy of each supertree method on these datasets as measured by their ability to minimize the three supertree criteria of SPR distance, RF distance, and parsimony score to the gene trees. In addition to the three basic methods, we tested a variant of SPR supertrees that uses the RF distance as a secondary optimization criterion to break ties when multiple supertrees have the same SPR distance, and tested the SPR and RF supertree methods when the MRP supertree was used as the initial tree. As MRP supertrees are unrooted, we computed the RF and SPR distances for each rooting of the MRP supertree and show the minimum value. For these tests each supertree method was run with its default parameters to match the comparisons of Bansal et al. [6] so we used the SPR and RF methods with 25 iterations of SPR rearrangements and the MRP method with 10 iterations of TBR rearrangements. Due to excessive running times (more than 3 days) for the MRP method on the marsupial and legume datasets we disabled the 'multrees' option on these runs which would otherwise retain multiple trees per iteration.

The performance of each approach according to all optimality criteria is shown in Table 8.1. Each supertree method was best at minimizing its respective optimization measure, suggesting that each method has merit and a well-balanced analysis should either include a justification for the choice of method (e.g. the presence of LGT for the SPR distance) or consider multiple optimization criteria. The MRP method required the least amount of time and the SPR method the most. However, the SPR method converged rapidly in 3, 1, 5 and 3 iterations on the marsupial, seabird, placental mammal, and legume datasets respectively and thus produced an optimal result in only a fraction of the reported time. Seeding the search with the MRP tree greatly reduced the time required by the SPR method and reduced the resulting parsimony scores at the expense of increasing the SPR distance. Starting with the MRP tree reduced the time required by the RF method and found supertrees with better RF and MRP scores on the marsupial and placental mammal datasets but increased RF and MRP scores on the legume dataset. Using the RF distance as a tie-breaker with the SPR method found lower SPR distances, RF distances and parsimony scores in a shorter period of

Table 8.1: Experimental results comparing the performance of the SPR supertree method to RF and MRP supertree methods. Six analyses are shown: The SPR supertree method starting from an SPR greedy addition tree (SPR) or MRP supertree (SPR-MRP), the SPR supertree method breaking ties with the RF distance using a greedy addition tree (SPR-RF-TIES), the RF supertree method starting from random addition sequence trees (RF-Ratchet) or MRP supertree (RF-MRP), and MRP with TBR global rearrangements (MRP-TBR). The best optimization criteria or running times for a dataset are shown in bold.

Data Set	Supertree Method	SPR Distance	RF-Distance	Parsimony Score	Time (s)
Marsupial (267 taxa; 158 trees)	SPR	382	1604	2203	1097.79
	SPR-RF-TIES	**373**	1536	2149	767.01
	SPR-MRP	380	1534	2126	219.64
	RF-Ratchet	394	1520	2145	2150.30
	RF-MRP	379	**1502**	2116	2044.07
	MRP-TBR	379	1514	**2112**	**20.52**
Sea Birds (121 taxa; 7 trees)	SPR	**17**	109	235	31.15
	SPR-RF-TIES	**17**	63	**208**	29.44
	SPR-MRP	**17**	**61**	**208**	2.04
	RF-Ratchet	**17**	**61**	**208**	10.43
	RF-MRP	**17**	**61**	**208**	9.16
	MRP-TBR	**17**	**61**	**208**	**1.03**
Placental Mammals (116 taxa; 726 trees)	SPR	1715	5908	8946	5561.84
	SPR-RF-TIES	**1713**	5902	8934	5040.03
	SPR-MRP	**1713**	5876	8921	1819.08
	RF-Ratchet	1790	5738	8827	801.92
	RF-MRP	1780	**5692**	8810	659.32
	MRP-TBR	1783	5702	**8809**	**34.27**
Legumes (558 taxa; 19 trees)	SPR	108	651	1175	21130.08
	SPR-RF-TIES	**92**	471	1037	12376.00
	SPR-MRP	110	511	903	276.49
	RF-Ratchet	117	**401**	1102	1349.56
	RF-MRP	130	429	1068	1558.60
	MRP-TBR	140	519	**891**	**579.76**

time over the basic method and avoided an issue with the seabird dataset where many supertrees have the same SPR distance but poor RF distances and parsimony scores. These results suggest that blended methods have merit even when only considering a single optimization criterion. In particular, the SPR distance with RF as a tie-breaker should be used when nontrivial amounts of lateral gene transfer are expected.

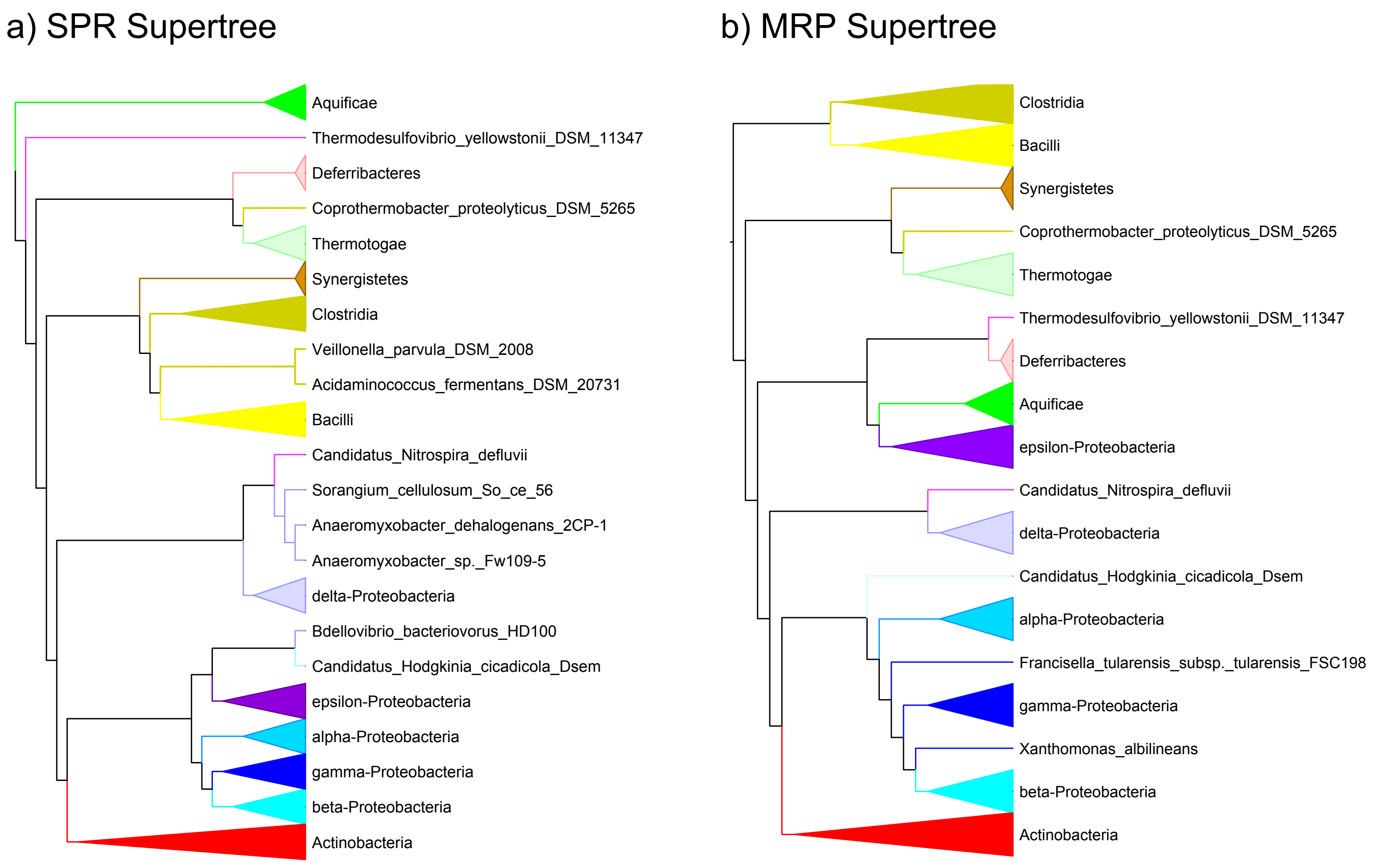

a) SPR Supertree
Aquificae
Thermodesulfovibrio_yellowstonii_DSM_11347
Deferribacteres
Coprothermobacter_proteolyticus_DSM_5265
Thermotogae
Synergistetes
Clostridia
Veillonella_parvula_DSM_2008
Acidaminococcus_fermentans_DSM_20731
Bacilli
Candidatus_Nitrospira_defluvii
Sorangium_cellulosum_So_ce_56
Anaeromyxobacter_dehalogenans_2CP-1
Anaeromyxobacter_sp._Fw109-5
delta-Proteobacteria
Bdellovibrio_bacteriovorus_HD100
Candidatus_Hodgkinia_cicadicola_Dsem
epsilon-Proteobacteria
alpha-Proteobacteria
gamma-Proteobacteria
beta-Proteobacteria
Actinobacteria
b) MRP Supertree
Clostridia
Bacilli
Synergistetes
Coprothermobacter_proteolyticus_DSM_5265
Thermotogae
Thermodesulfovibrio_yellowstonii_DSM_11347
Deferribacteres
Aquificae
epsilon-Proteobacteria
Candidatus_Nitrospira_defluvii
delta-Proteobacteria
Candidatus_Hodgkinia_cicadicola_Dsem
alpha-Proteobacteria
Francisella_tularensis_subsp._tularensis_FSC198
gamma-Proteobacteria
Xanthomonas_albilineans
beta-Proteobacteria
Actinobacteria

Figure 8.14: Comparison of SPR and MRP supertrees of 244 bacterial genomes. The SPR supertree on the left was constructed with the Aquificae as an outgroup while the MRP supertree on the right is unrooted and places the Aquificae as neighbours of the Epsilonproteobacteria. Both figures show the largest monophyletic group of each class as a collapsed subtree and all members of a given class with the same color.

8.3.5 Comparison of SPR and MRP Supertrees of 244 Bacterial Genomes

To contrast with the SPR supertree described above and examine the influence of tree rootings, we constructed an MRP supertree from the 244-taxon bacterial dataset using 25 iterations of an SPR rearrangement search and compared it to our SPR supertree (Figure 8.14). The MRP supertree does not recover the same arrangement of hyperthermophiles as the SPR supertree; notably, it places the Epsilonproteobacteria in close proximity to the Aquificae. If we place the root somewhat arbitrarily between the Firmicutes and all other Bacteria, the MRP supertree like the SPR supertree places the Thermotogae and *C. proteolyticus* as sisters, although this pairing is sister to the Synergistetes and not the Deferribacteres in the MRP supertree. The two Nitrospirae are again split, with *Nitrospira* sister to the Deltaproteobacteria and *Thermodesulfovibrio* with the Aquficae and Deferribacteres. As with the SPR supertree, the Deltaproteobacteria are separated from the other Proteobacteria.

The rooted nature of MAFs allowed the evaluation of our chosen rooting and alternative rootings on inferring phylogenetic relationships from this dataset. We have already described the MRP supertree rooted to separate the Firmicutes from the other taxa (MRP), the SPR supertree constructed from the 40 largest trees with a monophyletic Aquificae group (40-Aquificae) and the SPR supertree constructed using the SPR-Aquificae supertree (SPR-Aquificae). Three more supertrees were constructed to test the influence of starting topology and rooting. The first was an SPR supertree seeded with the MRP supertree (SPR-MRP). We then rooted the gene trees with both the MRP supertree and SPR-Aquificae tree using our balanced accuracy measure and constructed an SPR supertree from these two sets of rooted gene trees (SPR-MRP-Rooting and SPR-Aquificae-Rooting, respectively).

These six supertrees were compared to the two sets of rooted gene trees (see Table 8.2). The three MRP-rooted supertrees had a much smaller aggregate SPR distance (nearly 11% smaller) to the MRP-rooted gene trees than the Aquificae-rooted supertrees but the three Aquificae-rooted supertrees had a much smaller SPR distance (more than 8% smaller) to the Aquificae-rooted gene trees than the three MRP-rooted supertrees. Thus, it is impossible to determine which supertree is more similar to the gene trees without choosing a specific rooting of the gene trees.

Table 8.2: Aggregate SPR distance to supertrees constructed from different rootings of the bacterial protein trees. Six different construction methods were compared: The MRP supertree (MRP), the SPR supertree constructed from the 40 largest trees with a monophyletic Aquificae group (40-Aquificae), the SPR supertrees constructed using the MRP supertree (SPR-MRP) or SPR-Aquificae supertree (SPR-Aquificae), and the SPR supertrees constructed by only rooting the gene trees using the MRP supertree (SPR-MRP-Rooting) or SPR-Aquificae tree (SPR-Aquificae-Rooting) and building a greedy addition supertree. Each supertree was compared to the MRP rooted gene trees or SPR-Aquificae rooted gene trees with the SPR distance.

MRP rooted gene trees

	SPR Distance
SPR-MRP-Rooting	52867
SPR-MRP	52896
MRP	52896
SPR-Aquificae-Rooting	58539
SPR-Aquificae	59561
40-Aquificae	60611

SPR-Aquificae rooted gene trees

	SPR Distance
SPR-Aquificae-Rooting	53534
SPR-Aquificae	54488
40-Aquificae	55570
SPR-MRP-Rooting	58023
SPR-MRP	58057
MRP	58057

The four SPR supertrees constructed from the full bacterial dataset were compared by measuring their pairwise SPR distances (see Table 8.3). The two Aquificae-rooted supertrees differed by only 10 SPRs, despite the fact that one was constructed from the 40-Aquificae tree and the other was constructed with our usual greedy addition procedure and no *a priori* information other than the gene tree roots. Even more telling, the two MRP-rooted supertrees were essentially identical, differing by only 2 SPRs. The SPR-MRP-Rooting supertree also differed from the MRP supertree by only 2 SPRs, so we were able to essentially recover the MRP supertree just by biasing the gene tree roots. This suggests that MRP infers relationships that are consistent with certain gene tree roots despite not implicitly assuming any rooting. As these relationships are also inconsistent with plausible alternative roots, it may be that unrooted supertree methods such as MRP are insufficient to distinguish between controversial evolutionary hypotheses such as the placement of the Aquificae.

8.4 Discussion

Large phylogenies are being built from multiple sequence datasets to reconstruct the histories of many groups of living organisms, and supertrees offer the means to carry this out in a rigorous fashion. The known limitations of widely used approaches

Table 8.3: Dissimilarity of supertrees constructed from the same rooting of bacterial protein trees. We compared the minimal SPR distance between any rooting of the SPR supertree constructed from the 40 largest trees with a monophyletic Aquificae group (40-Aquificae), the SPR supertrees constructed using the MRP supertree (SPR-MRP) or SPR-Aquificae supertree (SPR-Aquificae), and the SPR supertrees constructed by only rooting the gene trees using the MRP supertree (SPR-MRP-Rooting) or SPR-Aquificae tree (SPR-Aquificae-Rooting) and building a greedy addition supertree.

	SPR-Aquificae	SPR-Aquificae-Rooting	SPR-MRP	SPR-MRP-Rooting
SPR-Aquificae	0	10	34	33
SPR-Aquificae-Rooting	10	0	27	25
SPR-MRP	34	27	0	2
SPR-MRP-Rooting	33	25	2	0

such as MRP have motivated the development of new strategies, such as the use of Robinson-Foulds distance as an alternative optimality criterion. Although RF is frequently used to assess the dissimilarity of phylogenetic trees, it is not based on a specific phylogenetic process and can be heavily influenced by shifts in the position of single taxa. A single LGT event will influence the RF distance (and parsimony score) in proportion to the number of branches in the path between the donor and recipient lineages, and many LGT events are likely to confound RF-based supertree inference. The SPR distance is an alternative optimality criterion that is particularly well-suited to analyzing phylogenomic data where LGT or other reticulate evolutionary processes are expected to play an important role in generating phylogenetic discordance. Each SPR operation is equivalent to an LGT event, and the degree of separation between donor and recipient in the tree does not influence the SPR score. The SPR distance may thus avoid some of the phylogenetic compromises of other supertree methods.

Using simulations, we verified that SPR supertrees were significantly more similar to the known species history than RF supertrees given biologically plausible rates of simulated LGT. The effect was more pronounced for random LGT, which produces more "long-distance" transfers, than for divergence-biased LGT. The improved performance of SPR with random LGT events suggests that penalizing phylogenetic discordance in a manner that is insensitive to the number of impacted bipartitions may be preferable to the alternative RF criterion. However, in the future this assertion should be tested under a wider range of scenarios, with larger trees and different types of phylogenetic discordance modelled. SPR also outperformed MRP in a narrower, but still biologically relevant, range of LGT rates. However, the advantage of SPR disappeared when the gene tree roots were unknown, demonstrating that the obligately rooted SPR approach is influenced by alternative rootings of the reference and gene trees. We also verified that each of the three supertree methods is best at minimizing its own criterion. Combining multiple supertree criteria, such as using the RF distance to break ties in an SPR supertree approach, yielded better results than any method did alone. This finding suggests that combinations of criteria that consider different types of phylogenetic discordance may provide even greater accuracy.

Although the history of bacteria may be better represented with a phylogenetic network than a single tree, the supertree we inferred offers a useful backdrop for the inference of highways of gene sharing. As shown in Figures 8.3 and 8.12, both SPR and MRP recovered a majority of bacterial classes as monophyletic groups, regardless of the choice of rooting. Many of the topological differences between the SPR and MRP supertrees are minor, including subtle shifts in the position of taxa such as *Nitrospira defluvii* and the Negativicutes. One point of substantial difference between the two trees related to the controversial placement of Aquificae and the Epsilonproteobacteria: MRP, being unrooted, placed these two groups adjacent to one another, corresponding to a sister relationship under the reasonable assumption that the root of the supertree is placed somewhere outside of this pairing. When the SPR supertree was constructed from trees rooted to reflect the MRP tree topology in the manner described above, the two supertrees were nearly identical; however, if Aquificae were treated as the outgroup then the SPR supertree produced a topology that placed other groups with many thermophiles, such as Thermotogae, as early branches. These results suggest that unrooted supertree criteria such as MRP provide hypotheses that are consistent with certain rootings despite not implicitly assuming any rooting. Furthermore, the Aquificae SPR supertree was much more similar to the Aquificae rooted gene trees than the MRP supertree, but the MRP supertree was much more similar to the MRP-rooted trees. It was thus impossible to distinguish between these two hypotheses of Aquificae placement; either could be plausible given knowledge of the correct gene tree roots. This is a practical example of the fundamental limits of unrooted supertree methods identified by Steel et al. [90].

Using the tree in Figure 8.3 as a basis for LGT inference, we searched for highways of LGT between classes and genera. Not surprisingly, connections were more frequently associated with specific lineages such as *Clostridium* and interactions between the Proteobacteria and other phyla varied considerably. In addition, larger gene trees (those shared by many taxa) required proportionately more transfers to explain, including ribosomal proteins. Such biased LGT could muddy or completely obscure the vertical evolutionary signal. Our improved SPR algorithm allowed the entire set of more than 40,000 trees to be reconciled with the supertree in less than one minute: a similar analysis could have been carried out using any rooted reference tree, regardless of what method was used to construct this tree. The rapid inference of LGT highways raises the possibility of using information about lateral connections to construct phylogenetic networks with reticulations explicitly based on major directions of LGT [13, 65, 73].

The scaling of runtimes with the number and size of trees is a central concern in phylogenomics. The analysis of Beiko et al. [14] required over 20,000 CPU hours to reconcile 22,432 gene trees with a 144-taxon supertree, and the largest trees could not be reconciled at all due to limitations of the breadth-first search of EEEP [13]. Alternative methods of inferring highways of LGT have been proposed based on quartets [5], but such methods are limited to finding the most obvious highways and required on the order of two days to analyze the same dataset of 22,432 gene trees. Repeated applications of SPR distances in large phylogenomic data sets were heretofore not feasible due to the complexity of the algorithm, but our efficient new methods for computing the SPR distance made the computation of these supertrees feasible even for hundreds of taxa and tens of thousands of gene trees. Of particular importance is the adaptation of the clustering strategy of Linz and Semple [63] to subdivide the construction of an MAF for a given pair of trees. Clustering yields no improvement in theoretical runtime, because there is no guarantee that more than one cluster will be identified between a pair of trees. However, our results clearly demonstrate that clustering is effective in practice, because LGT connections are not random and consistent partitioning can usually be identified and used as the basis for subdivision. We are optimistic that our approach will be applicable to much larger phylogenomic data sets with thousands of taxa, for two reasons: first, our fixed-parameter algorithm scales exponentially with the distance between a pair of trees and not their size; and second, as the timing results of Figure 8.9 suggest, clustering increases the speed of the algorithm and reduces the rate of increase of running times with increasing SPR distance. With only a small number of exceptions, all trees with SPR distance less than 60 were resolved in less than one second, with the time of MAF construction dominated by the single cluster with the largest distance. We expect that most large trees will have a cluster size distribution similar to that of the trees we tested here; consequently the size of the largest cluster and the corresponding computational burden may increase only slightly. This hypothesis remains to be tested on larger phylogenomic data sets.

Our methods could be expanded and refined in several ways. As we identified in our results, our current supertree search method could potentially be improved with a better strategy for constructing the initial guide tree such as SuperFine [93], methods for avoiding local optima such as ratchet searches, or using prior knowledge to constrain the supertree search [98]. An RF supertree method has been recently proposed for multi-labelled gene trees [29]; extending our SPR distance algorithms to accept such trees would enable their inclusion in SPR supertrees. The rooting problem remains to be resolved. While in many cases rooting can be performed using an appropriate outgroup taxon, the bacterial case considered here lacks an obvious outgroup: the Archaea could be used to root the Bacteria and vice versa, but many gene trees have shown evidence of interdomain LGT and rooting between domains may be invalid or even impossible. Finally, our approach considers only the history of observed genes, and does not attempt to account for processes such as gene duplication and loss. Methods of reconciling multiple evolutionary processes such as duplications, losses, transfers and incompatible lineage sorting (ILS) show a great deal of promise [4, 95], but are currently limited to smaller datasets [92].

Chapter 9

Conclusion

Maximum agreement forests provide a solid theoretical foundation for the analysis of lateral gene transfer, hybridization, and other reticulate evolutionary processes but have heretofore seen limited application due to the time required for their computation. I developed efficient algorithms to compute MAFs of both binary and multifurcating trees as well as MAAFs of binary trees. The size of these forests is equivalent to the SPR distance and hybridization number, respectively, measures of LGT and hybridization, and these forests can be used to propose a minimal set of such events. The MAF algorithms were shown to be many orders of magnitude faster than previous approaches, reducing the time required to compute MAFs representing 46 LGT events between two 144-leaf trees to fractions of a second. This enabled new evolutionary analyses including a 244-leaf supertree constructed from 40,631 bacterial genomes to minimize the effect of LGT and a large-scale analysis of the inferred LGT events between bacterial classes and genera.

My efficient fixed-parameter bounded search tree algorithms for computing MAFs and MAAFs represent a significant improvement over previous work and their development required several novel theoretical insights. My MAF algorithm for two binary trees takes $\mathrm{O}\left(2^k n\right)$ time, where n is the number of leaves in the trees and k is their SPR distance, a substantial improvement over the previous best algorithm with running time $\mathrm{O}\left(3^k n\right)$. My MAF algorithm for two multifurcating trees takes $\mathrm{O}\left(2.42^k n\right)$ time, where k is the “soft SPR distance” between the trees, the minimal SPR distance between some binary resolutions of the two trees. This new measure allows for uncertaintly in evolutionary analyses, and my algorithm is substantially better than a recently proposed $\mathrm{O}\left(4^k n\right)$ algorithm, the only other algorithm for this problem. My accompanying $\mathrm{O}\left(n \log n\right)$-time 3-approximation algorithm enables the rapid estimation of this measure for massive datasets. My MAAF algorithm for two binary trees runs in $O\left(3.18^k n\right)$ time, where k is the hybridization number, and is the first algorithm with a running time bounded by $O\left(c^k n\right)$ where c is a small constant. These algorithms make use of several novel strategies—edge protection to carry a small amount of useful information throughout an exponential search, bimodal searches to seamlessly combine varied structural information, and favouring provably viable solutions to avoid becoming bogged down by multiple optima—that will be of great use for efficiently solving other exponential search problems.

The MAF algorithms have been implemented as the open source RSPR software package. This software makes use of several additional heuristics—branch-and-bound, prioritizing nonbranching search options, the cluster reduction—and is the first SPR distance algorithm able to compare each 144-leaf tree of the protein tree dataset of Beiko et al. [14]. Previous methods required more than 5 hours to compute any SPR distance greater than 25 on this dataset while my algorithms required no more than 0.764 seconds for any tree pair; computing the maximum distance of 46 in this dataset would be infeasible with previous methods. RSPR can also be used to compare a binary and multifurcating tree with a negligible increase in running time.

These efficient algorithms enable the computation of SPR supertrees that minimize the number of inferred LGT events. Simulations showed that this SPR-based approach is more accurate than RF and MRP approaches given plausible rates and regimes of LGT. A highly plausible supertree was reconstructed for a phylogenomic dataset of 244 bacteria and the inferred highways of gene sharing showed several interesting trends corroborated by other evidence; the majority of inferred LGT events were between taxonomically related genera but substantial amounts of LGT appear to connect disparate classes and phyla. Interestingly, different rootings of the gene trees supported different placements of class Aquificae, suggesting that some evolutionary relationships may not be resolvable with unrooted trees. The effect of gene tree roots requires further investigation.

This body of work is a significant advancement and enables use of the SPR distance to analyze practical datasets. However, like all fixed-parameter algorithms, limitations may arise when the computed distances grow too large due to their exponential scaling on running time. The performance increase from branch-and-bound grew with tree size, somewhat offsetting this effect, but the approximation algorithms should be used to estimate distances in order to test the feasibilty of the fixed-parameter algorithms on large trees.

Several open challenges and avenues of expansion remain. First, I concentrated my implementation efforts on MAFs and the SPR distance to develop practical and useful software, so my MAAF algorithm remains unimplemented. Furthermore, extending my advancements such as edge protection and methods for handling multifurcating trees to quickly compute MAAFs of multifurcating trees would enable the effective study of hybridization. Second, I used a standard greedy addition and SPR treespace search approach to construct SPR supertrees, with the exception of my bipartition-based clustering measure, so there is much room for improvement of the supertree framework. New methods will be necessary to analyze the thousands of bacterial and archaeal genomes currently sequenced and tens of thousands that will be. Third, and finally, the insights and techniques developed in this thesis should be applied to the efficient construction of phylogenetic networks. An MAAF of two trees suffices to construct a minimal phylogenetic network of those trees, but this does not hold for generalized MAAFs of three or more trees. Efficiently constructing a minimal phylogenetic network to represent three or more trees remains a major open problem in the field of phylogenetics.

References

[1] Adams, E.N.III.: Consensus techniques and the comparison of taxonomic trees. Systematic Zoology 21(4), 390–397 (1972)

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

[3] Allen, B.L., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics 5(1), 1–15 (2001)

[4] Bansal, M.S., Alm, E.J., Kellis, M.: Efficient algorithms for the reconciliation problem with gene duplication, horizontal transfer and loss. Bioinformatics 28(12), i283–i291 (2012)

[5] Bansal, M.S., Banay, G., Harlow, T.J., Gogarten, J.P., Shamir, R.: Systematic inference of highways of horizontal gene transfer in prokaryotes. Bioinformatics 29(5), 571–579 (2013)

[6] Bansal, M.S., Burleigh, J.G., Eulenstein, O., Fernández-Baca, D.: Robinson-Foulds supertrees. Algorithms for Molecular Biology 5(1), 18 (2010)

[7] Baroni, M., Grünewald, S., Moulton, V., Semple, C.: Bounding the number of hybridisation events for a consistent evolutionary history. Journal of mathematical biology 51(2), 171–182 (2005)

[8] Baum, B.R.: Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. Taxon 41, 3–10 (1992)

[9] Beck, R.M.D., Bininda-Emonds, O.R.P., Cardillo, M., Liu, F.G.R., Purvis, A.: A higher-level MRP supertree of placental mammals. BMC Evolutionary Biology 6(1), 93 (2006)

[10] Beiko, R.G.: Telling the whole story in a 10,000-genome world. Biology Direct 6(1), 34 (2011)

[11] Beiko, R.G., Charlebois, R.L.: A simulation test bed for hypotheses of genome evolution. Bioinformatics 23(7), 825–831 (2007)

[12] Beiko, R.G., Doolittle, W.F., Charlebois, R.L.: The impact of reticulate evolution on genome phylogeny. Systematic Biology 57(6), 844–856 (2008)

[13] Beiko, R.G., Hamilton, N.: Phylogenetic identification of lateral genetic transfer events. BMC Evolutionary Biology 6(1), 15 (2006)

[14] Beiko, R.G., Harlow, T.J., Ragan, M.A.: Highways of gene sharing in prokaryotes. Proceedings of the National Academy of Sciences 102(40), 14332–14337 (2005)

[15] Bender, M.A., Farach-Colton, M.: The LCA problem revisited, pp. 88–94. LATIN 2000: Theoretical Informatics, Springer (2000)

[16] Bininda-Emonds, O.R.P., Gittleman, J.L., Steel, M.A.: The (super) tree of life: procedures, problems, and prospects. Annual Review of Ecology and Systematics 33, 265–289 (2002)

[17] Bininda-Emonds, O.R.P., Cardillo, M., Jones, K.E., MacPhee, R.D.E., Beck, R.M.D., Grenyer, R., Price, S.A., Vos, R.A., Gittleman, J.L., Purvis, A.: The delayed rise of present-day mammals. Nature 446(7135), 507–512 (2007)

[18] Bininda-Emonds, O.R.P., Sanderson, M.J.: Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. Systematic Biology 50(4), 565–579 (2001)

[19] Bonet, M.L., St. John, K.: Efficiently calculating evolutionary tree measures using SAT. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing. Lecture Notes in Computer Science, vol. 5584, pp. 4–17. Springer-Verlag (2009)

[20] Bonet, M.L., St. John, K., Mahindru, R., Amenta, N.: Approximating subtree distances between phylogenies. Journal of Computational Biology 13(8), 1419–1434 (2006)

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

[22] Bordewich, M., McCartin, C., Semple, C.: A 3-approximation algorithm for the subtree distance between phylogenies. Journal of Discrete Algorithms 6(3), 458–471 (2008)

[23] Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Annals of combinatorics 8(4), 409–423 (2005)

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

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

[26] Boussau, B., Guguen, L., Gouy, M.: Accounting for horizontal gene transfers explains conflicting hypotheses regarding the position of aquificales in the phylogeny of Bacteria. BMC Evolutionary Biology 8(1), 272 (2008)

[27] Cardillo, M., Bininda-Emonds, O.R.P., Boakes, E., Purvis, A.: A species-level phylogenetic supertree of marsupials. Journal of zoology 264(1), 11–31 (2004)

[28] Chataigner, F.: Approximating the maximum agreement forest on k trees. Information Processing Letters 93, 239–244 (2005)

[29] Chaudhary, R., Burleigh, J.G., Fernández-Baca, D.: Inferring species trees from incongruent multi-copy gene trees using the robinson-foulds distance. arXiv preprint arXiv:1210.2665 (2012)

[30] Chen, D., Eulenstein, O., Fernández-Baca, D., Burleigh, J.G.: Improved heuristics for minimum-flip supertree construction. Evolutionary Bioinformatics Online 2, 347–356 (2006)

[31] Chen, Z.Z., Wang, L.: Algorithms for reticulate networks of multiple phylogenetic trees. IEEE/ACM Transactions on Computational Biology and Bioinformatics 9(2), 372–384 (2012)

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

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

[34] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. McGraw-Hill (Jul 2001)

[35] Cotton, J.A., Wilkinson, M.: Majority-rule supertrees. Systematic biology 56(3), 445–452 (2007)

[36] Creevey, C.J., McInerney, J.O.: Clann: investigating phylogenetic information through supertree analyses. Bioinformatics 21(3), 390–392 (2005)

[37] Dagan, T., Roettger, M., Bryant, D., Martin, W.: Genome networks root the tree of life between prokaryotic domains. Genome Biology and Evolution 2, 379–392 (2010)

[38] Davies, T.J., Barraclough, T.G., Chase, M.W., Soltis, P.S., Soltis, D.E., Savolainen, V.: Darwin's abominable mystery: insights from a supertree of the angiosperms. Proceedings of the National Academy of Sciences 101(7), 1904–1909 (2004)

[39] Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. Journal of Classification 2(1), 7–28 (1985)

[40] Eulenstein, O., Chen, D., Burleigh, J.G., Fernández-Baca, D., Sanderson, M.J.: Performance of flip supertree construction with a heuristic algorithm. Systematic Biology 53(2), 299–308 (2004)

[41] Galtier, N., Daubin, V.: Dealing with incongruence in phylogenomic analyses. Philosophical Transactions of the Royal Society B: Biological Sciences 363(1512), 4023–4029 (2008)

[42] Goloboff, P.A.: Analyzing large data sets in reasonable times: solutions for composite optima. Cladistics 15(4), 415–428 (1999)

[43] Goloboff, P.A.: Minority rule supertrees? MRP, Compatibility, and Minimum Flip may display the least frequent groups. Cladistics 21(3), 282–294 (2005)

[44] Goloboff, P.A.: Calculating SPR distances between trees. Cladistics 24(4), 591–597 (2008)

[45] Gophna, U., Charlebois, R.L., Doolittle, W.F.: Ancient lateral gene transfer in the evolution of *Bdellovibrio bacteriovorus*. Trends in Microbiology 14(2), 64–69 (2006)

[46] Gordon, A.D.: Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labeled leaves. Journal of Classification 3(2), 335–348 (1986)

[47] Griffiths, E., Gupta, R.S.: Signature sequences in diverse proteins provide evidence for the late divergence of the order aquificales. International Microbiology 7(1), 41–52 (2004)

[48] Hallett, M.T., Lagergren, J.: Efficient algorithms for lateral gene transfer problems. In: Proceedings of the 5th Annual International Conference on Computational Biology. pp. 149–156. ACM New York, NY, USA (2001)

[49] Hallett, M., McCartin, C.: A faster FPT algorithm for the maximum agreement forest problem. Theory of Computing Systems 41, 539–550 (2007)

[50] He, M., Sebaihia, M., Lawley, T.D., Stabler, R.A., Dawson, L.F., Martin, M.J., Holt, K.E., Seth-Smith, H.M., Quail, M.A., Rance, R., et al.: Evolutionary dynamics of *Clostridium difficile* over short and long time scales. Proceedings of the National Academy of Sciences 107(16), 7527–7532 (2010)

[51] Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Applied Mathematics 71(1), 153–169 (1996)

[52] Hickey, G., Dehne, F., Rau-Chaplin, A., Blouin, C.: The computational complexity of the unrooted subtree prune and regraft distance. Technical Report CS-2006-06, Faculty of Computer Science, Dalhousie University (2006)

[53] Hickey, G., Dehne, F., Rau-Chaplin, A., Blouin, C.: SPR distance computation for unrooted trees. Evolutionary Bioinformatics 4, 17–27 (2008)

[54] Hillis, D.M., Moritz, C., Mable, B.K. (eds.): Molecular Systematics. Sinauer Associates (1996)

[55] Jumas-Bilak, E., Roudiére, L., Marchandin, H.: Description of 'synergistetes' phyl. nov. and emended description of the phylum 'deferribacteres' and of the family syntrophomonadaceae, phylum 'firmicutes'. International Journal of Systematic and Evolutionary Microbiology 59(5), 1028–1035 (May 01 2009)

[56] Kelk, S., van Iersel, L., Lekic, N., Linz, S., Scornavacca, C., Stougie, L.: 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 26(4), 1635–1656 (2012)

[57] Kennedy, M., Page, R.D.M., Prum, R.: Seabird supertrees: combining partial estimates of procellariiform phylogeny. The Auk 119(1), 88–108 (2002)

[58] Kunin, V., Goldovsky, L., Darzentas, N., Ouzounis, C.A.: The net of life: reconstructing the microbial phylogenetic network. Genome Research 15(7), 954–959 (July 2005)

[59] Lapierre, P., Lasek-Nesselquist, E., Gogarten, J.P.: The impact of HGT on phylogenomic reconstruction methods. Briefings in Bioinformatics (2012)

[60] Lcker, S., Wagner, M., Maixner, F., Pelletier, E., Koch, H., Vacherie, B., Rattei, T., Damsté, J.S.S., Spieck, E., Paslier, D.L.: A nitrospira metagenome illuminates the physiology and evolution of globally important nitrite-oxidizing bacteria. Proceedings of the National Academy of Sciences 107(30), 13479–13484 (2010)

[61] Lin, H.T., Burleigh, J.G., Eulenstein, O.: Triplet supertree heuristics for the tree of life. BMC Bioinformatics 10(Suppl 1), S8 (2009)

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

[63] Linz, S., Semple, C.: A cluster reduction for computing the subtree distance between phylogenies. Annals of Combinatorics 15(3), 465–484 (2011)

[64] Lloyd, G.T., Davis, K.E., Pisani, D., Tarver, J.E., Ruta, M., Sakamoto, M., Hone, D.W.E., Jennings, R., Benton, M.J.: Dinosaurs and the Cretaceous terrestrial revolution. Proceedings of the Royal Society B: Biological Sciences 275(1650), 2483–2490 (2008)

[65] MacLeod, D., Charlebois, R.L., Doolittle, W.F., Bapteste, E.: Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement. BMC Evolutionary Biology 5, 27 (2005)

[66] Maddison, W.: Reconstructing character evolution on polytomous cladograms. Cladistics 5(4), 365–377 (1989)

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

[68] Maddison, W.P., Knowles, L.L.: Inferring phylogeny despite incomplete lineage sorting. Systematic Biology 55(1), 21–30 (2006)

[69] Marchandin, H., Teyssier, C., Campos, J., Jean-Pierre, H., Roger, F., Gay, B., Carlier, J.P., Jumas-Bilak, E.: *Negativicoccus succinicivorans* gen. nov., sp. nov., isolated from human clinical samples, emended description of the family *Veillonellaceae* and description of *Negativicutes* classis nov., *Selenomonadales* ord. nov. and *Acidaminococcaceae* fam. nov. in the bacterial phylum *Firmicutes*. International Journal of Systematic and Evolutionary Microbiology 60(6), 1271–1279 (2010)

[70] McCutcheon, J.P., McDonald, B.R., Moran, N.A.: Origin of an alternative genetic code in the extremely small and GC-rich genome of a bacterial symbiont. PLoS genetics 5(7), e1000565 (2009)

[71] Morrison, D.: Carl Woese and new perspectives on evolution. NASA Astrobiology Institute. `http://nai.arc.nasa.gov/news_stories/news_detail.cfm?ID=274` (2003)

[72] Munoz, R., Yarza, P., Ludwig, W., Euzéby, J., Amann, R., Schleifer, K.H., Glöckner, F.O., Rosselló-Móra, R.: Release LTPs104 of the All-Species Living Tree. Systematic and Applied Microbiology 34(3), 169–170 (2011)

[73] Nakhleh, L., Ruths, D.A., Wang, L.S.: RIATA-HGT: a fast and accurate heuristic for reconstructing horizontal gene transfer. In: Proceedings of the 11th International Conference on Computing and Combinatorics. Lecture Notes in Computer Science, vol. 3595, pp. 84–93. Springer-Verlag (2005)

[74] Nakhleh, L., Warnow, T., Lindner, C.R., St. John, K.: Reconstructing reticulate evolution in species—theory and practice. Journal of Computational Biology 12(6), 796–811 (2005)

[75] Nelson, K.E., Clayton, R.A., Gill, S.R., Gwinn, M.L., Dodson, R.J., Haft, D.H., Hickey, E.K., Peterson, J.D., Nelson, W.C., Ketchum, K.A., et al.: Evidence for lateral gene transfer between Archaea and Bacteria from genome sequence of *Thermotoga maritima*. Nature 399(6734), 323–329 (1999)

[76] Piaggio-Talice, R., Burleigh, J.G., Eulenstein, O.: Quartet supertrees. In: Phylogenetic supertrees: Combining information to reveal the Tree of Life, vol. 3, pp. 173–191. Springer Netherlands (2004)

[77] Piovesan, T., Kelk, S.: A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees (2013)

[78] Pisani, D., Cotton, J.A., McInerney, J.O.: Supertrees disentangle the chimerical origin of eukaryotic genomes. Molecular Biology and Evolution 24(8), 1752–1760 (2007)

[79] Pisani, D., Wilkinson, M.: Matrix representation with parsimony, taxonomic congruence, and total evidence. Systematic Biology 51(1), 151 (2002)

[80] Ragan, M.A.: Phylogenetic inference based on matrix representation of trees. Molecular Phylogenetics and Evolution 1(1), 53–58 (1992)

[81] Rainey, F.A., Stackebrandt, E.: Transfer of the type species of the genus *Thermobacteroides* to the genus *Thermoanaerobacter* as *Thermoanaerobacter acetoethylicus* (Ben-Bassat and Zeikus 1981) comb. nov., description of *Coprothermobacter* gen. nov., and reclassification of *Thermobacteroides proteolyticus* as *Coprothermobacter proteolyticus* (Ollivier et al. 1985) comb. nov. International Journal of Systematic and Evolutionary Microbiology 43(4), 857–859 (1993)

[82] Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical Biosciences 53(1), 131–147 (1981)

[83] Rodrigues, E.M., Sagot, M.F., Wakabayashi, Y.: The maximum agreement forest problem: Approximation algorithms and computational experiments. Theoretical Computer Science 374(1-3), 91–110 (2007)

[84] Rosas-Magallanes, V., Deschavanne, P., Quintana-Murci, L., Brosch, R., Gicquel, B., Neyrolles, O.: Horizontal transfer of a virulence operon to the ancestor of *Mycobacterium tuberculosis*. Molecular Biology and Evolution 23(6), 1129–1135 (2006)

[85] Roshan, U.W., Moret, B.M.E., Warnow, T., Williams, T.L.: Rec-I-DCM3: a fast algorithmic technique for reconstructing phylogenetic trees. In: Proceedings of the 3rd International IEEE Computational Systems Bioinformatics Conference. pp. 98–109 (2004)

[86] Sanford, R.A., Cole, J.R., Tiedje, J.M.: Characterization and description of *Anaeromyxobacter dehalogenans* gen. nov., sp. nov., an aryl-halorespiring facultative anaerobic myxobacterium. Applied and Environmental Microbiology 68(2), 893–900 (2002)

[87] Schneiker, S., Perlova, O., Kaiser, O., Gerth, K., Alici, A., Altmeyer, M.O., Bartels, D., Bekel, T., Beyer, S., Bode, E., Bode, H.B., Bolten, C.J., Choudhuri, J.V., Doss, S., Elnakady, Y.A., Frank, B., Gaigalat, L., Goesmann, A., Groeger, C., Gross, F., Jelsbak, L., Jelsbak, L., Kalinowski, J., Kegler, C., Knauber, T., Konietzny, S., Kopp, M., Krause, L., Krug, D., Linke, B., Mahmud, T., Martinez-Arias, R., McHardy, A.C., Merai, M., Meyer, F., Mormann,

S., Munoz-Dorado, J., Perez, J., Pradella, S., Rachid, S., Raddatz, G., Rosenau, F., Ruckert, C., Sasse, F., Scharfe, M., Schuster, S.C., Suen, G., Treuner-Lange, A., Velicer, G.J., Vorholter, F.J., Weissman, K.J., Welch, R.D., Wenzel, S.C., Whitworth, D.E., Wilhelm, S., Wittmann, C., Blocker, H., Puhler, A., Muller, R.: Complete genome sequence of the myxobacterium *Sorangium cellulosum*. Nature Biotechnology 25(11), 1281–1289 (2007)

[88] Smoot, M.E., Ono, K., Ruscheinski, J., Wang, P.L., Ideker, T.: Cytoscape 2.8: new features for data integration and network visualization. Bioinformatics 27(3), 431–432 (2011)

[89] St. John, K.: Comparing phylogenetic trees. In: EMBO Workshop on Current Challenges and Problems in Phylogenetics. Isaac Newton Institute, Cambridge, UK (2007)

[90] Steel, M., Bcker, S.: Simple but fundamental limitations on supertree and consensus tree methods. Systematic Biology 49(2), 363–368 (2000)

[91] Stolp, H., Starr, M.P.: *Bdellovibrio bacteriovorus* gen. et sp. n., a predatory, ectoparasitic, and bacteriolytic microorganism. Antonie van Leeuwenhoek 29(1), 217–248 (1963)

[92] Stolzer, M., Lai, H., Xu, M., Sathaye, D., Vernot, B., Durand, D.: Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. Bioinformatics 28(18), i409–i415 (2012)

[93] Swenson, M.S., Suri, R., Linder, C.R., Warnow, T.: Superfine: fast and accurate supertree estimation. Systematic Biology 61(2), 214–227 (2012)

[94] Swofford, D.: Paup 4.0 b10: phylogenetic analysis using parsimony (2002)

[95] Szöllősi, G.J., Boussau, B., Abby, S.S., Tannier, E., Daubin, V.: Phylogenetic modeling of lateral gene transfer reconstructs the pattern and relative timing of speciations. Proceedings of the National Academy of Sciences 109(43), 17513–17518 (2012)

[96] Thiergart, T., Landan, G., Schenk, M., Dagan, T., Martin, W.F.: An evolutionary network of genes present in the eukaryote common ancestor polls genomes on eukaryotic and mitochondrial origin. Genome Biology and Evolution 4(4), 466–485 (2012)

[97] van Iersel, L., Kelk, S., Lekić, N., Stougie, L.: Computing nonbinary agreement forests. ArXiv e-prints (Oct 2012)

[98] Wehe, A., Burleigh, J.G., Eulenstein, O.: Algorithms for knowledge-enhanced supertrees. In: Bioinformatics Research and Applications. Lecture Notes in Computer Science, vol. 7292, pp. 263–274. Springer (2012)

[99] Whidden, C., Beiko, R.G., Zeh, N.: Fixed-parameter and approximation algorithms for maximum agreement forests of multifurcating trees, submitted to *IEEE/ACM Transactions on Computational Biology and Bioinformatics*

[100] Whidden, C., Beiko, R.G., Zeh, N.: Fast FPT algorithms for computing rooted agreement forests: Theory and experiments. In: Proceedings of the 9th International Symposium on Experimental Algorithms. Lecture Notes in Computer Science, vol. 6049, pp. 141–153. Springer-Verlag (2010)

[101] Whidden, C., Beiko, R.G., Zeh, N.: Fixed-parameter and approximation algorithms for maximum agreement forests. Tech. Rep. arXiv:1108.2664 [q-bio.PE] (2011)

[102] Whidden, C., Beiko, R.G., Zeh, N.: Fixed-parameter algorithms for maximum agreement forests. SIAM Journal on Computing. 42(4), 1431–1436 (2013)

[103] Whidden, C., Zeh, N.: A unifying view on approximation and FPT of agreement forests. In: Proceedings of the 9th International Workshop. Lecture Notes in Bioinformatics, vol. 5724, pp. 390–401. Springer-Verlag (2009)

[104] Whidden, C.: A Unifying View on Approximation and FPT of Agreement Forests. Master's thesis, Dalhousie University, Halifax, NS (October 2009)

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

[106] Whidden, C., `http://kiwi.cs.dal.ca/Software/SPRSupertrees`: SPR Supertree Software

[107] Wilkinson, M., Cotton, J.A., Creevey, C., Eulenstein, O., Harris, S.R., Lapointe, F.J., Levasseur, C., Mcinerney, J.O., Pisani, D., Thorley, J.L.: The shape of supertrees to come: tree shape related properties of fourteen supertree methods. Systematic biology 54(3), 419–432 (2005)

[108] Woese, C.R., Kandler, O., Wheelis, M.L.: Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. Proceedings of the National Academy of Sciences 87(12), 4576–4579 (1990)

[109] Wojciechowski, M.F., Sanderson, M.J., Steele, K.P., Liston, A.: Molecular phylogeny of the temperate herbaceous tribes of papilionoid legumes: a supertree approach. Advances in Legume Systematics 9, 277–298 (2000)

[110] Wu, Y.: A practical method for exact computation of subtree prune and regraft distance. Bioinformatics 25(2), 190–196 (2009)

[111] Yutin, N., Puigbó, P., Koonin, E.V., Wolf, Y.I.: Phylogenomics of prokaryotic ribosomal proteins. PLoS One 7(5), e36972 (2012)

Appendix A

Supplemental Figures and Tables

Table A.1: List of 244 Bacterial Genomes included in this work.

Class	Taxon
Actinobacteria	*Acidimicrobium ferrooxidans* DSM 10331
	Acidothermus cellulolyticus 11B
	Amycolatopsis mediterranei U32
	Arcanobacterium haemolyticum DSM 20595
	Arthrobacter aurescens TC1
	Arthrobacter sp. FB24
	Beutenbergia cavernae DSM 12333
	Bifidobacterium adolescentis ATCC 15703
	Bifidobacterium animalis subsp. lactis AD011
	Bifidobacterium animalis subsp. lactis Bl-04
	Bifidobacterium longum NCC2705
	Bifidobacterium longum subsp. infantis ATCC 15697
	Bifidobacterium longum subsp. longum JDM301
	Catenulispora acidiphila DSM 44928
	Cellulomonas flavigena DSM 20109
	Clavibacter michiganensis subsp. michiganensis NCPPB 382
	Corynebacterium aurimucosum ATCC 700975
	Corynebacterium efficiens YS-314
	Corynebacterium glutamicum ATCC 13032 DSM 20300
	Corynebacterium glutamicum R
	Corynebacterium jeikeium K411
	Corynebacterium kroppenstedtii DSM 44385
	Corynebacterium pseudotuberculosis FRC41
	Corynebacterium urealyticum DSM 7109
	Cryptobacterium curtum DSM 15641
	Eggerthella lenta DSM 2243
	Frankia alni ACN14a
	Gardnerella vaginalis 409-05

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
	Geodermatophilus obscurus DSM 43160
	Gordonia bronchialis DSM 43247
	Jonesia denitrificans DSM 20603
	Kribbella flavida DSM 17836
	Kytococcus sedentarius DSM 20547
	Leifsonia xyli subsp. xyli str. CTCB07
	Mobiluncus curtisii ATCC 43063
	Mycobacterium avium 104
	Mycobacterium avium subsp. paratuberculosis K-10
	Mycobacterium bovis AF2122/97
	Mycobacterium bovis BCG str. Pasteur 1173P2
	Mycobacterium bovis BCG str. Tokyo 172
	Mycobacterium gilvum PYR-GCK
	Mycobacterium leprae Br4923
	Mycobacterium leprae TN
	Mycobacterium marinum M
	Mycobacterium smegmatis str. MC2 155
	Mycobacterium sp. KMS
	Mycobacterium tuberculosis F11
	Mycobacterium tuberculosis H37Ra
	Mycobacterium tuberculosis H37Rv
	Mycobacterium vanbaalenii PYR-1
	Nakamurella multipartita DSM 44233
	Nocardia farcinica IFM 10152
	Nocardioides sp. JS614
	Propionibacterium acnes KPA171202
	Propionibacterium freudenreichii subsp. shermanii CIRM-BIA1
	Rhodococcus erythropolis PR4
	Rhodococcus jostii RHA1
	Rothia mucilaginosa DY-18
	Salinispora arenicola CNS-205
	Salinispora tropica CNB-440
	Sanguibacter keddieii DSM 10542
	Segniliparus rotundus DSM 44985

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
	Slackia heliotrinireducens DSM 20476
	Stackebrandtia nassauensis DSM 44728
	Streptomyces avermitilis MA-4680
	Streptomyces griseus subsp. griseus NBRC 13350
	Streptomyces scabiei 87.22
	Streptosporangium roseum DSM 43021
	Thermobifida fusca YX
	Thermobispora bispora DSM 43833
	Thermomonospora curvata DSM 43183
	Tropheryma whipplei TW08/27
	Tsukamurella paurometabola DSM 20162
	Xylanimonas cellulosilytica DSM 15894
Alphaproteobacteria	*Bradyrhizobium* sp. BTAi1
	Candidatus Hodgkinia cicadicola Dsem
	Candidatus Pelagibacter ubique HTCC1062
	Ehrlichia canis str. Jake
	Ehrlichia chaffeensis str. Arkansas
	Erythrobacter litoralis HTCC2594
	Gluconacetobacter diazotrophicus PAl 5
	Mesorhizobium loti MAFF303099
	Ochrobactrum anthropi ATCC 49188
	Parvularcula bermudensis HTCC2503
	Rickettsia akari str. Hartford
	Rickettsia canadensis str. McKiel
	Rickettsia peacockii str. Rustic
	Rickettsia rickettsii str. Sheila Smith
	Wolbachia endosymbiont of Culex quinquefasciatus Pel
Aquificae	Aquifex aeolicus VF5
	Hydrogenobacter thermophilus TK-6
	Hydrogenobaculum sp. Y04AAS1
	Persephonella marina EX-H1
	Sulfurihydrogenibium azorense Az-Fu1
	Sulfurihydrogenibium sp. YO3AOP1
	Thermocrinis albus DSM 14484

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
Bacilli	Bacillus anthracis str. Sterne
	Bacillus cereus 03BB102
	Bacillus cereus AH187
	Bacillus cereus ATCC 10987
	Bacillus cereus G9842
	Bacillus cereus Q1
	Bacillus clausii KSM-K16
	Bacillus thuringiensis BMB171
	Bacillus thuringiensis str. Al Hakam
	Enterococcus faecalis V583
	Exiguobacterium sibiricum 255-15
	Exiguobacterium sp. AT1b
	Geobacillus sp. WCH70
	Lactobacillus acidophilus NCFM
	Lactobacillus casei ATCC 334
	Lactobacillus casei str. Zhang
	Lactobacillus crispatus ST1
	Lactobacillus reuteri JCM 1112
	Lactobacillus rhamnosus Lc 705
	Lactobacillus salivarius UCC118
	Lactococcus lactis subsp. cremoris MG1363
	Leuconostoc kimchii IMSNU 11154
	Listeria monocytogenes HCC23
	Listeria monocytogenes serotype 4b str. CLIP 80459
	Listeria monocytogenes serotype 4b str. F2365
	Staphylococcus aureus RF122
	Staphylococcus carnosus subsp. carnosus TM300
	Staphylococcus lugdunensis HKU09-01
	Streptococcus gordonii str. Challis substr. CH1
	Streptococcus mitis B6
	Streptococcus mutans NN2025
	Streptococcus pneumoniae 670-6B
	Streptococcus pneumoniae JJA
	Streptococcus pyogenes MGAS10270

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
	Streptococcus pyogenes MGAS10394
	Streptococcus pyogenes MGAS10750
	Streptococcus pyogenes NZ131
	Streptococcus pyogenes str. Manfredo
	Streptococcus suis 98HAH33
	Streptococcus thermophilus LMD-9
Betaproteobacteria	*Azoarcus* sp. BH72
	Bordetella parapertussis 12822
	Burkholderia ambifaria MC40-6
	Burkholderia sp. 383
	Burkholderia vietnamiensis G4
	Candidatus Accumulibacter phosphatis clade IIA str. UW-1
	Gallionella capsiferriformans ES-2
	Methylibium petroleiphilum PM1
	Methylobacillus flagellatus KT
	Methylotenera mobilis JLW8
	Methylotenera sp. 301
	Nitrosomonas europaea ATCC 19718
	Ralstonia pickettii 12D
	Ralstonia solanacearum CFBP2957
	Thiobacillus denitrificans ATCC 25259
Clostridia	*Acetohalobium arabaticum* DSM 5501
	Acidaminococcus fermentans DSM 20731
	Ammonifex degensii KC4
	Caldicellulosiruptor obsidiansis OB47
	Caldicellulosiruptor saccharolyticus DSM 8903
	Caldicelulosiruptor becscii DSM 6725
	Clostridiales genomosp. BVAB3 str. UPII9-5
	Clostridium acetobutylicum ATCC 824
	Clostridium botulinum A str. ATCC 19397
	Clostridium botulinum B str. Eklund 17B
	Clostridium botulinum Ba4 str. 657
	Clostridium botulinum E3 str. Alaska E43
	Clostridium cellulovorans 743B

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
	Clostridium difficile CD196
	Clostridium kluyveri DSM 555
	Clostridium kluyveri NBRC 12016
	Clostridium perfringens ATCC 13124
	Clostridium perfringens SM101
	Clostridium tetani E88
	Clostridium thermocellum ATCC 27405
	Coprothermobacter proteolyticus DSM 5265
	Desulfotomaculum acetoxidans DSM 771
	Eubacterium rectale ATCC 33656
	Finegoldia magna ATCC 29328
	Halothermothrix orenii H 168
	Heliobacterium modesticaldum Ice1
	Natranaerobius thermophilus JW/NM-WN-LF
	Pelotomaculum thermopropionicum SI
	Syntrophothermus lipocalidus DSM 12680
	Thermincola potens JR
	Thermoanaerobacter mathranii subsp. mathranii str. A3
	Thermoanaerobacter tengcongensis MB4
	Thermosediminibacter oceani DSM 16646
	Veillonella parvula DSM 2008
Deferribacteres	*Deferribacter desulfuricans* SSM1
	Denitrovibrio acetiphilus DSM 12809
Deltaproteobacteria	*Anaeromyxobacter dehalogenans* 2CP-1
	Anaeromyxobacter sp. Fw109-5
	Bdellovibrio bacteriovorus HD100
	Desulfotalea psychrophila LSv54
	Desulfovibrio desulfuricans subsp. desulfuricans str. G20
	Desulfovibrio salexigens DSM 2638
	Desulfovibrio vulgaris str. Miyazaki F
	Desulfurivibrio alkaliphilus AHT2
	Geobacter bemidjiensis Bem
	Geobacter lovleyi SZ
	Geobacter uraniireducens Rf4

Continued on next page

Table A.1 – continued from previous page

Class	Taxon
	Lawsonia intracellularis PHE/MN1-00
	Pelobacter carbinolicus DSM 2380
	Pelobacter propionicus DSM 2379
	Sorangium cellulosum So ce 56
Epsilonproteobacteria	*Arcobacter nitrofigilis* DSM 7299
	Campylobacter concisus 13826
	Campylobacter jejuni subsp. doylei 269.97
	Campylobacter jejuni subsp. jejuni 81116
	Campylobacter jejuni subsp. jejuni NCTC 11168
	Helicobacter acinonychis str. Sheeba
	Helicobacter hepaticus ATCC 51449
	Helicobacter mustelae 12198
	Helicobacter pylori B38
	Helicobacter pylori HPAG1
	Helicobacter pylori J99
	Helicobacter pylori Shi470
	Nautilia profundicola AmH
	Nitratiruptor sp. SB155-2
Gammaproteobacteria	*Acinetobacter baumannii* AB0057
	Acinetobacter baumannii ATCC 17978
	Actinobacillus pleuropneumoniae serovar 3 str. JL03
	Escherichia coli BW2952
	Escherichia coli HS
	Francisella tularensis subsp. tularensis FSC198
	Pseudomonas fluorescens Pf-5
	Shewanella halifaxensis HAW-EB4
	Shigella flexneri 2a str. 2457T
	Xanthomonas albilineans
	Xenorhabdus bovienii SS-2004
	Yersinia pestis Antiqua
	Yersinia pestis CO92
Nitrospirae	*Candidatus* Nitrospira defluvii
	Thermodesulfovibrio yellowstonii DSM 11347
Synergistetes	*Aminobacterium colombiense* DSM 12261

Continued on next page

Table A.1 – continued from previous page

Class	**Taxon**
	Thermanaerovibrio acidaminovorans DSM 6589
Thermotogae	*Fervidobacterium nodosum* Rt17-B1
	Kosmotoga olearia TBF 19.5.1
	Petrotoga mobilis SJ95
	Thermosipho africanus TCF52B
	Thermosipho melanesiensis BI429
	Thermotoga lettingae TMO
	Thermotoga maritima MSB8
	Thermotoga naphthophila RKU-10
	Thermotoga neapolitana DSM 4359
	Thermotoga petrophila RKU-1
	Thermotoga sp. RQ2

Appendix B

Copyright Permission Letters

Springer

May 14, 2013

Springer reference
Lecture Notes in Computer Science Volume 6049
Experimental Algorithms. 2010
9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings
Editors: Paola Festa
ISBN: 978-3-642-13192-9
Chapter: "Fast FPT Algorithms for Computing Rooted Agreement Forests: Theory and Experiments" by Chris Whidden, Robert G. Beiko, Norbert Zeh.
pp 141-153.

Your Phd Thesis:
University: Faculty of Graduate Studies at Dalhousie University, Halifax, Nova Scotia, Canada.
Title: Dissertation/Thesis

With reference to your request to reuse material in which Springer Science+Business Media controls the copyright, our permission is granted free of charge under the following conditions:

Springer material
- represents original material which does not carry references to other sources (if material in question refers with a credit to another source, authorization from that source is required as well);
- requires full credit (book title, year of publication, page, chapter title, name(s) of author(s), original copyright notice) is given to the publication in which the material was originally published by adding: "With kind permission of Springer Science+Business Media";
- may not be altered in any manner. Any other abbreviations, additions, deletions and/or any other alterations shall be made only with prior written authorization of the author and/or Springer Science+Business Media.

This permission
- is non-exclusive;
- is valid for one-time use only for the purpose of defending your thesis and with a maximum of 100 extra copies in paper.
- includes use in an electronic form, provided it is an author-created version of the thesis on his/her own website and his/her university's repository, including UMI (according to the definition on the Sherpa website: http://www.sherpa.ac.uk/romeo/);
- is subject to courtesy information to the corresponding author;
- is personal to you and may not be sublicensed, assigned, or transferred by you to any other person without Springer's written permission;
- is valid only when the conditions noted above are met.

Permission free of charge does not prejudice any rights we might have to charge for reproduction of our copyrighted material in the future.

Best regards,

Rights and Permissions
Springer-Verlag GmbH
Tiergartenstr. 17
69121 Heidelberg
Germany

May 2, 2013

SIAM
3600 Market Street, 6th Floor
Philadelphia, PA 19104-2688 USA

I am preparing my PhD thesis for submission to the Faculty of Graduate Studies at Dalhousie University, Halifax, Nova Scotia, Canada. I am seeking your permission to include a manuscript version of the following paper(s) as a chapter in the thesis:

Fixed-Parameter Algorithms for Maximum Agreement Forests. Chris Whidden, Robert G. Beiko, Norbert Zeh. To appear in SIAM Journal on Computing. 2013.

Canadian graduate theses are reproduced by the Library and Archives of Canada (formerly National Library of Canada) through a non-exclusive, world-wide license to reproduce, loan, distribute, or sell theses. I am also seeking your permission for the material described above to be reproduced and distributed by the LAC(NLC). Further details about the LAC(NLC) thesis program are available on the LAC(NLC) website (www.nlc-bnc.ca).

Full publication details and a copy of this permission letter will be included in the thesis.

Yours sincerely,

Christopher Whidden

Permission is granted for:

a) the inclusion of the material described above in your thesis.

b) for the material described above to be included in the copy of your thesis that is sent to the Library and Archives of Canada (formerly National Library of Canada) for reproduction and distribution.

Name: [redacted] Title: Managing Editor

Signature: [redacted] Date: 5/3/13