Computing Hybridization Networks Using Agreement Forests

Benjamin Albrecht

München 2015

Computing Hybridization Networks Using Agreement Forests

Benjamin Albrecht

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Benjamin Albrecht
aus Heilbronn-Neckargartach

München, den 21. Dezember 2015

Erstgutachter: Prof. Dr. Volker Heun
Zweitgutachter: Dr. Katharina Huber
Tag der mündlichen Prüfung: 8. April 2016

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Albrecht, Benjamin

Name, Vorname

Tübingen, 13. Mai 2016	
Ort, Datum	Unterschrift Doktorand/in

Contents

Zusammenfassung **xvii**

Summary **xix**

1 Introduction **1**

1.1 Phylogenetic trees and networks . . . 1

1.2 Process of reticulate evolution . . . 5

1.3 Related work . . . 7

1.3.1 Hybridization Networks . . . 8

1.3.2 Inferring hybridization networks . . . 8

1.4 Basic notations . . . 10

1.4.1 Further definitions for binary trees . . . 12

1.4.2 Further definitions for nonbinary trees . . . 15

1.5 Organization of this thesis . . . 19

2 Hybridization networks for two binary trees **23**

2.1 Reduction rules . . . 23

2.1.1 Subtree reduction . . . 23

2.1.2 Cluster reduction . . . 24

2.2 A fast approach for computing minimum hybridization networks . . . 26

2.2.1 Further definitions . . . 27

2.2.2 The algorithm . . . 27

2.2.3 Simulation study . . . 31

2.2.4 Application to phylogeny of Aegilop/Triticum genera . . . 35

2.2.5 Discussion . . . 37

2.3 The algorithm allMAAFs . . . 39

2.3.1 Related work . . . 39

2.3.2 Further definitions . . . 39

2.3.3 Description of the algorithm . . . 41

2.3.4 Correctness of the algorithm allMAAFs . . . 42

2.3.5 Runtime of the algorithm allMAAFs . . . 44

2.4 Speeding up allMAAFs . . . 45

2.4.1 Further definitions . . . 45

2.4.2 The Algorithm allMAAFs 47
2.4.3 Modifications to allMAAFs 47
2.4.3.1 The Algorithm allMAAFs^1 50
2.4.3.2 The Algorithm allMAAFs^2 52
2.4.3.3 The Algorithm allMAAFs^3 52
2.4.4 Proofs of Correctness 54
2.4.4.1 The algorithm processCherries 54
2.4.4.2 Correctness of allMAAFs^1 55
2.4.4.3 Correctness of allMAAFs^2 62
2.4.4.4 Correctness of allMAAFs^3 67
2.4.5 Simulation Study 68
2.4.6 Runtime of allMAAFs^1, allMAAFs^2 and allMAAFs^3 76
2.4.7 Conclusion 79
2.5 TerminusEst 80
2.5.1 The algorithm TerminusEst 80
2.5.2 Correctness of TerminusEst 82
2.5.3 The algorithm TerminusEst* 82
2.5.4 Correctness of TerminusEst* 83
2.5.5 Runtime of TerminusEst 84
2.5.6 Simulation study 84
2.5.7 Conclusion 88

3 Hybridization networks for multiple binary trees 89
3.1 Further Definitions 89
3.2 Reduction rules 91
3.3 Correctness of the cluster reduction 91
3.3.1 Related work 91
3.3.2 Further definitions 92
3.3.3 Proof of Theorem 13 92
3.4 The algorithm allHNetworks 98
3.4.1 Further definitions 99
3.4.2 The Algorithm allHNetworks 99
3.4.2.1 Inserting Trees into Networks 100
3.4.2.2 Inserting Components into Networks 101
3.4.2.3 Combinatorial complexity 104
3.4.2.4 Pseudocode of allHNetworks 106
3.4.3 Use case 107
3.4.4 Proof of correctness 111
3.4.5 Runtime of allHNetworks 118
3.4.6 Speeding Up the Algorithm allHNetworks 119
3.4.6.1 Parallelization 119
3.4.6.2 Reductions rules 120
3.4.7 Conclusion 126

3.5 A fast approach for computing minimum hybridization networks 127
3.5.1 Further definitions 128
3.5.2 The algorithm 128
3.5.3 Parallelization 129
3.5.4 Additional features 130
3.5.5 Results and discussion 131
3.5.5.1 Simulation study 131
3.5.5.2 Application to a grass dataset 141
3.5.6 Conclusion 144

4 Hybridization networks for nonbinary trees 145
4.1 Further definitions 145
4.2 The algorithm allMulMAFs 148
4.2.1 Notations 148
4.2.2 The algorithm 150
4.2.3 Correctness of allMulMAFs 153
4.2.3.1 The algorithm allMulMAFs* 155
4.2.3.2 The algorithm ProcessCherries 156
4.2.3.3 Proof of Correctness 159
4.2.4 Runtime of allMulMAFs 166
4.2.5 Conclusion 166
4.3 The algorithm allMulMAAFs 167
4.3.1 Refining agreement forests 167
4.3.2 The algorithm 169
4.3.3 Correctness of allMulMAAFs 170
4.3.4 Runtime of allMulMAAFs 172
4.3.5 Robustness of our Implementation 173
4.3.5.1 Synthetic dataset 173
4.3.5.2 Comparison with other software 173
4.3.5.3 Permutation test 174
4.3.6 Conclusion 175
4.4 The algorithm allMulHNetworks 176
4.4.1 Further definitions 176
4.4.2 The algorithm 178
4.4.2.1 Inserting trees into networks 178
4.4.2.2 Preparing networks for tree insertion 179
4.4.2.3 Inserting components into networks 182
4.4.2.4 Application of shifting-patterns of Type 1 187
4.4.2.5 Application of shifting-patterns of Type 2 190
4.4.3 Runtime 190
4.4.4 Correctness 191
4.4.5 Speeding up the algorithm 192
4.4.6 Robustness of our Implementation 192

4.4.6.1 Synthetic dataset 192
4.4.6.2 Permutation test 192
4.4.7 Conclusion 193

5 Studying phylogenetic networks 195
5.1 Filtering relevant networks 195
5.1.1 Further definitions 196
5.1.2 Network constraints 196
5.1.3 Application to a grass data set 201
5.2 Rerooting trees by hybridization number 206
5.3 Hybroscale 208
5.3.1 Distinctive features 208
5.3.2 Details on the algorithm 208
5.3.3 Drawing Rooted Phylogenetic Networks 210
5.4 Conclusion 213

Conclusion and outlook 215

Danksagung 224

List of Figures

1.1 Phylogenetic $\mathcal{X}$-trees and agreement forests 15
1.2 Resolving nonbinary nodes 16
1.3 Ancestor-descendant graph 18

2.1 Cluster reduction 25
2.2 Runtime comparison between Dendroscope and HybridNET – Part 1 . 34
2.3 Runtime comparison between Dendroscope and HybridNET – Part 2 . 35
2.4 Two consensus trees 37
2.5 Three hybridization networks for two consensus trees 38
2.6 Two phylogenetic $\mathcal{X}$ together with a corresponding agreement forest. . . . 51
2.7 Illustration of different types of special cherry actions 58
2.8 Scenario regarding Step 1 of the correctness proof of allMAAFs^1 59
2.9 Scenario regarding Step 2 of the correctness proof of allMAAFs^1 60
2.10 Two forests as defined in Lemma 8 63
2.11 Illustration of Case (1) and Case (2) regarding Lemma 9 64
2.12 Illustration of the Cases (2.3.1) to (2.3.3) regarding Lemma 9 66
2.13 Illustration of a scenario regarding Lemma 9 66
2.14 Runtime comparison between allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 Part 1 70
2.15 Runtime comparison between allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 Part 2 71
2.16 Speedups calculated for allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 72
2.17 Comparisons of recursive calls between allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 Part 1 74
2.18 Comparisons of recursive calls between allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 Part 2 75
2.19 Scatter-plots generated for allMAAFs, allMAAFs^1, allMAAFs^2, and allMAAFs^3 77
2.20 An illustration of terminal nodes 81
2.21 Runtime comparison between allMAAFs^3, TerminusEst, and TerminusEst* 86

2.22 Scatter-plots generated for allMAAFs^3, TerminusEst, and TerminusEst* . . . 87

3.1 Definitions regarding restricted networks . . . 90
3.2 An illustration of stacks of hybridization nodes . . . 90
3.3 Definition regarding restricted pendant subtrees . . . 93
3.4 Illustration of a scenario compensating an edge . . . 94
3.5 Illustration of Case (ii) and (iii) regarding Lemma 14 . . . 95
3.6 Illustration of Case (iv.i), (iv.ii), and (iv.iii) regarding Lemma 14 . . . 96
3.7 Illustration of Case (iv.i) regarding Lemma 14 . . . 97
3.8 An illustration of the definitions of source and target nodes regarding the algorithm allHNetworks . . . 103
3.9 Adding new reticulation edges into networks for binary trees . . . 104
3.10 An illustration of a tree-insertion regarding the algorithm allHNetworks 105
3.11 Figure 1 of a use case referring to the algorithm allHNetworks . . . 107
3.12 Figure 2 of a use case referring to the algorithm allHNetworks . . . 108
3.13 Figure 3 of a use case referring to the algorithm allHNetworks . . . 108
3.14 Figure 3 of a use case referring to the algorithm allHNetworks . . . 109
3.15 Figure 4 of a use case referring to the algorithm allHNetworks . . . 110
3.16 Figure 5 of a use case referring to the algorithm allHNetworks . . . 110
3.17 An example showing why the algorithm allHNetworks has to consider different orderings of the input trees . . . 117
3.18 An illustration of computational paths generated by the algorithm allHNetworks . . . 120
3.19 An illustration of an caught edge within a cluster network . . . 121
3.20 An illustration of the application of linking patterns - Part 2 . . . 124
3.21 An illustration of the application of linking patterns - Part 1 . . . 125
3.22 Visualization of embedded trees within a hybridization network via the software package Hybroscale . . . 130
3.23 An illustration of the cluster degree parameter . . . 132
3.24 Runtime comparison between Hybroscale and PIRNv2.0 for 3 input trees– Part 1 . . . 132
3.25 Runtime comparison between Hybroscale and PIRNv2.0 for 3 input trees – Part 2 . . . 133
3.26 Scatterplots of the runtimes attained by Hybroscale and PIRNv2.0 for 3 input trees . . . 133
3.27 Comparison of the real- and user-runtime of Hybroscale and PIRNv2.0 for 3 input trees . . . 134
3.28 Speedups of Hybroscale versus PIRN2.0 . . . 134
3.29 Scatterplots of the runtimes attained by Hybroscale and PIRNv2.0 for 4 input trees . . . 137
3.30 Scatterplots of the runtimes attained by Hybroscale and PIRNv2.0 for 5 input trees . . . 137

3.31 Runtime comparison between Hybroscale and PIRNv2.0 for 4 input trees– Part 1 138
3.32 Runtime comparison between Hybroscale and PIRNv2.0 for 5 input trees– Part 1 138
3.33 Comparison of the real- and user-runtime of Hybroscale and PIRNv2.0 for 4 input trees 139
3.34 Comparison of the real- and user-runtime of Hybroscale and PIRNv2.0 for 5 input trees 139
3.35 Runtime comparison between Hybroscale and PIRNv2.0 for 4 input trees– Part 2 140
3.36 Runtime comparison between Hybroscale and PIRNv2.0 for 5 input trees– Part 1 140

4.1 Trees reflecting an agreement forest 147
4.2 Two different ways of contracting a cherry regarding the algorithm allMulMAFs 149
4.3 Two different ways of cutting an edge regarding the algorithm allMulMAFs 149
4.4 An illustration of Case 3b following the description the algorithm allMulMAFs 150
4.5 Two different ways of expanding a cherry regarding the algorithm allMulMAFs 152
4.6 An illustration of Case 3a following the description the algorithm allMulMAFs 154
4.7 An illustration of a pseudo cherry regarding the algorithm allMulMAFs* 155
4.8 Two examples of calling the algorithm ProcessCherries 158
4.9 An illustration of the scenario regarding Lemma 22 160
4.10 An illustration of Case 4 regarding the proof of Lemma 23 162
4.11 An example of an expanded ancestor-descendant graph 170
4.12 An illustration of the cluster degree parameter 174
4.13 Definitions regarding restricted networks 177
4.14 An illustration of a stack of hybridization nodes 177
4.15 Preparing a network for tree insertion 181
4.16 Adding new reticulation edges into networks for nonbinary trees 184
4.17 Illustration of inserting a tree into a nonbinary network — Part 1 185
4.18 Illustration of inserting a tree into a nonbinary network — Part 1 186
4.19 An illustration of a highest ancestor in respect to a reticulation edge 188
4.20 An illustration of the shifting-patterns of Type 1 189
4.21 An illustration of the shifting-patterns of Type 2 191

5.1 A particular digraph for a hybridization network 198
5.2 Example of an SQL-statement for filtering hybridization networks 199
5.3 A network satisfying certain direct-hybridization constraints 200
5.4 The main window of the program Hybroscale 212

List of Tables

2.1 Runtime comparison between Dendroscope and HybridNET 36
2.2 Accession numbers in GenBank 36

3.1 Parameters of a use case referring to the algorithm allHNetworks . . . 107
3.2 Parameters of a use case referring to the algorithm allHNetworks . . . 109
3.3 Results calculated by Hybroscale for a grass dataset – Part 1 141
3.4 Results calculated by Hybroscale for a grass dataset – Part 2 143

5.1 Application of a direct-hybridization constraint 202
5.2 Network-constraints applied via Hybroscale for a grass dataset 204
5.3 Network-constraints applied via Hybroscale for a grass dataset 205
5.4 Results for rerooting a grass dataset 207

Zusammenfassung

Gewurzelte phylogenetische Bäume werden häufig in der Biologie dazu verwendet, um den evolutionären Verlauf bestimmter Spezies zu repräsentieren. Gewöhnlich handelt es sich dabei um einfache Binärbäume in denen jeder innere Knoten (außer der Wurzel) einen Eingangsgrad von eins und einen Ausgangsgrad von zwei aufweist und somit lediglich einfache *Speciation Events* repräsentieren. In der angewandten Phylogenie jedoch können Bäume auch innere Knoten mit Ausgangsgrad größer als zwei beinhalten. Dies ist nämlich genau dann der Fall, wenn in den zugrunde liegenden Daten nicht ausreichend Information vorhanden ist, um die Reihenfolge bestimmter *Speciation Events* eindeutig aufzulösen. Die gängige Art und Weise, um so eine Uneindeutigkeit zu modellieren, besteht darin, nicht-binäre Knoten (d.h. Knoten mit Ausgangsgrad drei oder mehr) zu verwenden.

Zusätzlich zu solchen *Speciation Events* gibt es jedoch auch bestimmte biologische Ereignisse, die nicht durch einen Baum dargestellt werden können, sondern vielmehr das generellere Konzept eines *gewurzelten phylogenetischen Netzwerkes* bzw. *Hybridisierungs-Netzwerkes* benötigen. Solch ein *verzweigendes Ereignis* kann dann ein Hinweis auf *horizontalen Gentransfer*, *Hybridisierung* oder *Rekombination* sein.

Um ein phylogenetisches Netzwerk zu konstruieren, lässt sich jedoch das etwas einfachere Konzept eines phylogenetischen Baumes als Grundbaustein wiederverwenden. Genauer gesagt besteht häufig der erste Schritt darin, für eine bestimmte Menge an Spezies mehrere phylogenetische Bäume basierend auf verschiedenen orthologen Genen zu konstruieren. In einem zweiten Schritt wird dann zunächst eine Menge von gemeinsamen Teilbäumen, sog. *maximum acyclic agreement forests*, bestimmt, welche dann letztendlich zu einem gemeinsamen Hybridisierungs-Netzwerk zusammen gefügt werden. In so einem Netzwerk können dann sog. Hybridisierungs-Knoten (Knoten mit Eingangsgrad größer gleich zwei) entstehen, welche potentielle Hybridisierungs-Ereignisse des zugrunde liegenden evolutionären Verlaufs symbolisieren. Da solche Ereignisse jedoch sehr selten sind, sind aus einem biologischen Blickwinkel besonders diejenigen Netzwerke von großer Bedeutung die eine minimale Anzahl dieser verzweigenden Ereignisse aufweisen.

Daher kann man, aus mathematischer Sichtweise, das Problem zur Konstruktion solcher Netzwerke folgendermaßen beschreiben. Für eine gegebene Menge $\mathcal{T}$ an gewurzelten phylogenetischen Bäume, berechne ein Hybridisierungs-Netzwerk, welches eine minimale Anzahl an verzweigenden Ereignissen aufweist und jeden Baum aus $\mathcal{T}$ beinhaltet. In diesem Zusammenhang sagt man, dass ein Netzwerk N einen Baum T genau dann beinhaltet, falls man aus N den Baum T durch Löschen bestimmter Kanten und Unterdrücken bestimmter

Knoten wiedergewinnen kann. Unglücklicherweise ist dies ein sehr schweres Problem (genauer gesagt liegt es in der Klasse *NP-schwer*), auch für den einfachsten Fall, falls nämlich $\mathcal{T}$ nur aus zwei Binärbäumen besteht.

Diese Dissertation beschäftigt sich nun mit mehreren Methoden für dieses NP-schwere Problem. Unser erster Ansatz beschreibt dabei, wie man eine repräsentative Menge minimaler Hybridisierungs-Netzwerke für zwei Binärbäume berechnen kann [6]. Zu diesem Zweck, wurde der erste nicht-naive Algorithmus allMAAFs [57] entwickelt, der in der Lage ist, alle *maximum acyclic agreement forests* zweier gewurzelter Binärbäume zu berechnen. In einem weiteren Schritt, beschreiben wir mehrere Modifikationen des Algorithmus allMAAFs, welche jeweils die Anzahl an Berechnungsschritten deutlich reduziert und somit die praktische Laufzeit signifikant verbessert [4].

Unser zweite Ansatz beschreibt eine Erweiterung unseres ersten Ansatzes, welcher den zugrunde liegende Algorithmus nun für mehrere Binärbäume zugänglich macht [2]. Hierzu verwenden wir den Algorithmus allHNetworks [1], welcher der erste Algorithmus ist der alle relevanten Hybridisierungs-Netzwerke für mehrere Binärbäume berechnen kann. Diese Eigenschaft ist daher wünschenswert, da hierdurch eine vernünftige Interpretation der Netzwerke verbessert (bzw. erst ermöglicht) wird.

Abschließend haben wir wiederum unseren zweiten Ansatz dahingehend erweitert, so dass dieser nun mit mehreren nicht-binären Bäumen umgehen kann. Um dies zu ermöglichen, haben wir den Algorithmus allMulMAAFs [5] entwickelt, der nun in der Lage ist, eine relevante Menge an *nonbinary maximum acyclic agreement foests* zweier Bäume mit nicht-binären Knoten zu berechnen.

Jeder unserer Algorithmen wurde des Weiteren in unser benutzerfreundliches Java-basierende Programm *Hybroscale* [3] integriert, welches insbesondere frei erhältlich ist und auf allen gängigen Plattformen funktioniert. Hybroscale besitzt eine graphische Benutzeroberfläche um Bäume bzw. Netzwerke entsprechend zu visualisieren. Des Weiteren, erleichtert es die Interpretation der berechneten Netzwerke, indem es Methoden bereitstellt, welche die jeweiligen Netzwerke mit zusätzlicher Information versorgen können. Darüber hinaus, besitzt unser Programm eine auf SQL (*Structured Query Language*) basierende Modellierungssprache, um die häufig große Menge an Netzwerke gezielt zu filtern.

Summary

Rooted phylogenetic trees are widely used in biology to represent the evolutionary history of certain species. Usually, such a tree is a simple *binary tree* only containing internal nodes of in-degree one and out-degree two representing specific speciation events. In applied phylogenetics, however, trees can contain nodes of out-degree larger than two because, often, in order to resolve some orderings of speciation events, there is only insufficient information available and the common way to model this uncertainty is to use *nonbinary nodes* (i.e., nodes of out-degree of at least three), also denoted as *polytomies*.

Moreover, in addition to such speciation events, there exist certain biological events that cannot be modeled by a tree and, thus, require the more general concept of *rooted phylogenetic networks* or, more specifically, of *hybridization networks*. Examples for such *reticulate events* are *horizontal gene transfer*, *hybridization*, and *recombination*.

Nevertheless, in order to construct hybridization networks, the less general concept of a phylogenetic tree can still be used as building block. More precisely, often, in a first step, phylogenetic trees for a set of species, each based on a distinctive orthologous gene, are constructed. In a second step, specific sets containing common subtrees of those trees, known as *maximum acyclic agreement forests*, are calculated, which are then glued together to a single hybridization network. In such a network, *hybridization nodes* (i.e., nodes of in-degree larger than or equal to two) can exist representing potential reticulate events of the underlying evolutionary history. As such events are considered as rare phenomena, from a biological point of view, especially those networks representing a minimum number of reticulate events, which is denoted as *hybridization number*, are of high interest.

Consequently, in a mathematical aspect, the problem of calculating hybridization networks can be briefly described as follows. Given a set $\mathcal{T}$ of rooted phylogenetic trees sharing the same set of taxa, compute a hybridization network N *displaying* $\mathcal{T}$ with minimum hybridization number. In this context, we say that such a network N *displays a phylogenetic tree* T, if we can obtain T from N by removing as well as contracting some of its nodes and edges. Unfortunately, this is a computational hard problem (i.e., it is *NP-hard*), even for the simplest case given just two binary input trees.

In this thesis, we present several methods tackling this *NP-hard* problem. Our first approach describes how to compute a representative set of minimum hybridization networks for two binary input trees [6]. For that purpose, our approach implements the first non-naive algorithm — called ALLMAAFs — calculating all *maximum acyclic agreement forests* for two rooted binary phylogenetic trees on the same set of taxa [57]. In a subsequent step, in order to maximize the efficiency of the algorithm allMAAFs, we have developed additionally several modifications each reducing the number of computational steps and, thus, significantly improving its practical runtime [4].

Our second approach is an extension of our first approach making the underlying algorithm accessible to more than two binary input trees [1]. For this purpose, our approach implements the algorithm allHNetworks [2] being the first algorithm calculating all relevant hybridization networks displaying a set of rooted binary phylogenetic trees on the same set of taxa, which is a preferable feature when studying hybridization events.

Lastly, we have developed a generalization of our second approach that can now deal with multiple nonbinary input trees. For that purpose, our approach implements the first non-naive algorithm — called allMulMAAFs — calculating a relevant set of *nonbinary maximum acyclic agreement forests* for two rooted (nonbinary) phylogenetic trees on the same set of taxa [5].

Each of the algorithms above is integrated into our user friendly Java-based software package Hybroscale [3], which is freely available and platform independent, so that it runs on all major operating systems. Our program provides a graphical user interface for visualizing trees and networks. Moreover, it facilitates the interpretation of computed hybridization networks by adding specific features to its graphical representation and, thus, supports biologists in investigating reticulate evolution. In addition, we have implemented a method using a user friendly SQL-style modeling language for filtering the usually large amount of reported networks.

Chapter 1

Introduction

In this chapter, we first introduce the concept of phylogenetic trees and networks as well as processes of reticulate evolution and then briefly describe all important related works regarding the computation of hybridization networks. Next, we give all basic notations that are used throughout this work. Finally, we give an outline of this thesis by giving a short summary of each chapter. Additionally, we discuss the correlations of published work and papers in preparation with the results of this thesis.

1.1 Phylogenetic trees and networks

Phylogenetic analysis aims at uncovering the evolutionary relationship between contemporary species in order to shed light on how biological diversity has evolved on earth. The principal tools that are used for this purpose are based on the concept of phylogenetic trees. However, as we will discuss later, this concept has certain limitations and, thus, in order to expand the scope of evolutionary thinking, the application of the more general concept of phylogenetic networks has become more and more important in molecular evolution.

Charles Darwin is considered to be the first scientist that started to analyze evolutionary history using phylogenetic trees and, thus, is seen as the founder of evolutionary biology. His famous book *On the Origin of Species by Means of Natural Selection* (1859) [22] contains a sketch of a bifurcating phylogenetic tree modeling the evolutionary relationship among certain species. From there on, the concept of phylogenetic trees has been developed further so that now it has become widely accepted and dominates the recognition and interpretation of patterns in genetic data [7].

Roughly speaking, such a tree is a directed acyclic graph, in which each node of out-degree zero represents a set of contemporary species, each node of out-degree unequal to zero a speciation event, and the root the lowest common ancestor of all species under examination. Note that in Section 1.4 we give a formal definition of such trees.

Whereas, at the beginning, phylogenetic trees were calculated in respect to morphological characters, nowadays the inference of such trees is typically based on molecular sequences, i.e., nucleotide or amino acid sequences. This is mainly due to the fact that sequencing technologies have become more and more efficient. Hence, more and more scientist can afford to sequence their desired parts of a genome, which they consider as being appropriate for investigating evolution. Moreover, today there exist several sequence databases, e.g., Genebank [10] or UniProt [21], that are publicly available so that scientist can simply download particular sequences avoiding the rather complex application of sequence technology.

In the last decades, researchers have put a lot of time and effort in developing methods calculating phylogenetic trees trying to reconstruct the past of a set of contemporary *taxa*, which is just an umbrella term for species, individuals, genes or other taxonomic units. Apart from the fact that those methods are either based on morphological characters or molecular sequences, following the book of Huson *et al.* [33], one can distinguish between four other types.

Distance-based methods. Those methods are based on pairwise evolutionary distances between the given set of taxa, which is often calculated under the assumption of certain evolutionary models. Usually, those methods provide a good practical running time. However, in order to apply such methods, the given data has to fulfill certain assumptions, which is, however, not often the case. Nevertheless, those methods are widely used in order to obtain a first approximation, which is then elaborated on by applying more sophisticated methods. The most popular distance methods are *Neighbor Joining* (NJ) [53] as well as the *Unweighted Pair Group Method with Arithmetic mean* (UPGMA) [58].

Maximum parsimony methods. A maximum parsimony method is characterized by always looking for the simplest explanation. Thus, tree building methods that are based on this concept try to construct those trees containing a minimum number of evolutionary events necessary to explain the development of different taxa. Due to the large number of possible trees [25], methods taking all possible trees into account are usually infeasible. Thus, in practice heuristic methods are applied following a certain strategy for exploring the underlying tree space. This, for example, can be done by adding the set of taxa stepwise beginning with the simplest tree containing only two taxa [26] or by starting with a precomputed tree already containing all taxa (e.g., a star), which is then modified by certain tree operations like *branch swapping methods* [62] for instance .

Maximum likelihood methods. The goal of these methods is to find a tree with edge lengths maximizing a certain likelihood. More precisely, this likelihood is determined by taking the topology of a tree, its edge lengths, and a certain evolutionary model, e.g., the *Jukes-Cantor model of evolution* [36], into account. Such methods have the advantage that through the evolutionary model one can include some previous knowledge of underlying processes that generated the given data. As in the case of parsimony, these methods cannot afford it to consider all kinds of possible trees and, thus, again heuristic methods are applied in practice. Two widely used maximum likelihood programs are for example PhyML [29] and RAxML [60].

Bayesian methods. In contrast to all so far introduced methods, Bayesian methods are more sophisticated. These methods essentially aim at sampling a set of appropriate trees, which are then processed further in a certain way. Therefor, the sampling of those trees is usually done by the *Markov Chain Monte Carlo* approach constructing a sequence of trees, in which each tree is slightly modified in terms of its predecessor. The decision whether a tree T_i is accepted in terms of a previously selected tree T', is based on the ratio of the two posterior probabilities corresponding to T_i and T'. Again, this posterior probability takes the tree topology, its edge lengths, and an evolutionary model into account. Moreover, the way the trees are modified in the Markov chain is usually done by applying certain modified branch swapping methods. As example, *MrBayes 3* [52] is a widely used software using such a Bayesian approach.

Notice that, as it is the case for phylogenetic networks, such tree building methods can construct either rooted or unrooted trees. Since for a phylogenetic analysis, one needs an evolutionary direction, one has to turn unrooted trees into rooted trees, which can be simply done by either declaring one of its inner nodes to be the root or by inserting a root into one of its edges. Notice, however, that the placement of the root is not a trivial step and, thus, in order to determine the right location there have been proposed several ways [47]. Given a set of taxa (the so-called *ingroup*), a common strategy is to add a further set of closely related taxa (the so-called *outgroup*) so that, if in the calculated unrooted tree both groups are separated, one can place the root into the edge connecting those two groups.

It turns out that the concept of phylogenetic trees is an inherent part of our research culture and is actually still the preferred way for investigating evolution. This concept, however, is often insufficient as a simple tree diagram cannot be used in order to represent certain more complex signals of the underlying data. Following the book of Morrison [48], those signals can be caused by two fundamental situations arising from *estimation errors* as well as *biological conflicts.*

Estimation errors. Estimation errors include all kind of problems occurring during the calculation of phylogenetic trees, which, for example, can be due to *inaccurate data*, *inappropriate sampling*, or *model mis-specification.* More specifically, one can obtain inaccurate data from poor laboratory techniques including sequence contamination or PCR-mediated recombination. Moreover, inappropriate sampling affects the choice of the data, which can be inappropriate if certain taxa are either too closely related or too evolutionary distant. Lastly, based on the respective tree-building method, model misspecification can happen if the wrong evolutionary model has been chosen or at least some of its parameters have been set to inappropriate values.

Biological conflicts. Biological conflicts can be due to two reasons dealing with *analogy* and *homology.* More specifically, analogy refers to the observation that two quite similar taxa must not have evolved by simple evolutionary descent, which for example can happen if there are two different origins for the evolution of certain characters (known as *parallelism*) or if two similar characters have evolved separately to acquire a certain functionality (known as *convergence*). Furthermore, homology refers to certain processes provoking reticulate evolution, e.g., *hybridization*, *introgression*, *recombination*, and *horizontal gene transfer* (HGT). As those *reticulation events* are actually the motivation regarding phylogenetic networks, we will describe each of them separately in Section 1.2.

Now, when constructing several trees for a set of taxa based on different molecular sequences, e.g., different homologous genes, due to the problems above incongruent trees, each describing a different evolutionary scenario, can arise. Thereby trees that are affected from estimation errors as well as biological conflicts dealing with analogy are trees that are obviously inappropriate for describing evolutionary processes and, thus, are of no interest here. However, each tree affected by biological conflicts dealing with homology may represent a piece of a puzzle of the true underlying evolutionary history and, thus, has to be taken into account, which is, however, only possible using the more general concept of phylogenetic networks [30]. Notice, however, that usually much time and effort has to be spent in order to recognize those trees that are relevant for investigating evolution.

Phylogenetic networks are generalizations of phylogenetic trees that can be used to analyze data whose evolution additionally contains reticulate processes. In contrast to phylogenetic trees, however, investigating evolution under the concept of phylogenetic networks is not of widespread use even though there have been developed several methods available for their computation [34]. An important reason therefor may be that, until now, the research of phylogenetic networks is not well structured, which means there is a lot of confusion in the literature regarding the classification of these networks or the respective methods used for their computation. First of all, as it is the case for phylogenetic trees, there exist rooted and unrooted networks. Moreover, depending on how networks have been generated or should be interpreted, one can distinguish between *implicit and explicit networks* [32]. Unfortunately, for these kind of networks two further terms have been introduced; networks classified by these two categories are also called *abstract or explicit networks* [46], and *data-display or evolutionary networks* [48].

Implicit networks. Mainly all implicit (or abstract or data-display) networks are unrooted networks that are in general used to display incompatibilities of the underlying data. In contrast to rooted networks, unrooted networks do not explicitly describe evolutionary processes and, thus, are rather inappropriate regarding the investigation of reticulate evolution. One of the most important unrooted phylogenetic networks are *Split networks* that can be derived either from distances, trees, sequences, or quartets. More information about these kinds of networks as well as a brief overview of other types of unrooted phylogenetic networks can be found in the work of Huson and Scornavacca [34]. One may notice that the concept of unrooted phylogenetic networks has been developed quite well so far, which means that there exist a lot of methods as well as user-friendly software packages, e.g., SplitsTree [31], that are widely used in phylogenetic analysis [34].

Explicit networks. Explicit (or evolutionary) networks are rooted phylogenetic networks directly outlining the evolution of certain taxa by explicitly indicating reticulation events. There exist a lot of algorithms, as discussed in Section 1.3, dealing with the concept of rooted phylogenetic networks. Most of those algorithms, however, are only so-called *proof-of-concept methods* not having been integrated into user-friendly software packages so far and, consequently, are not widely used in laboratories for investigating reticulate evolution. Recently, this situation has slightly improved as now some appropriate software packages integrating certain algorithms are available, e.g., Dendroscope [35] and PhyloNET [63]. It will, however, still take some time and effort to develop further appropriate methods in order to encourage biologists to investigate evolution using rooted phylogenetic networks.

1.2 Process of reticulate evolution

Calculating phylogenetic trees for a set of taxa based on different molecular sequences, e.g., genes, can lead to incongruent phylogenetic trees (henceforth denoted as gene trees). As previously discussed, the reasons leading to different gene trees are estimation errors and biological conflicts, which again can be due to reasons dealing with *analogy* and *homology*. Whereas estimation errors and problems due to analogy can be simply avoided by carefully creating and checking the underlying data, problems occurring due to homology have to be taken into account when investigating evolutionary processes. More precisely, in this case, incongruent gene trees can be taken as evidence that reticulation events may have occurred during the underlying evolutionary history.

Following the work of Morrison [48], one can distinguish between two different types of reticulation events; those in which horizontal gene flow has actually taken place and those occurring due to other circumstances. Whereas for the first type, a network containing reticulation events is the common way for displaying evolutionary history, the second scenario is usually displayed by presenting different trees all being contained in a superior tree — a so-called *species tree*. Thus, when computing rooted phylogenetic networks for a set of incongruent gene trees, in a first step one has to check due to which reasons those trees differ. This means, in particular, depending on the respective network-building method, one should check carefully if the underlying genetic data fulfills the required assumptions.

In the following, we first introduce two scenarios — known as *Deep Coalescence* (resp. *Incomplete Lineage Sorting*) or *Duplication-loss* — that can lead to incongruent gene trees being part of an underlying species tree and, thus, should *not* be reconciled into phylogenetic networks. Notice that there exist specific methods, e.g., iGTP [16] or PhyloNET [63], calculating species trees from discordant gene trees by taking the likelihood of these two scenarios into account.

Deep coalescence (or incomplete lineage-sorting). Regarding a gene tree, each node representing a speciation event is also called *coalescent event* as by going backward in time such an event coalesce different ancestral populations of certain contemporary species. Now, regarding the corresponding species tree embedding such a gene tree, two ancestral populations of certain contemporary species can pass certain speciation events, which means that, by going backward in time, those populations coalescence deeper than the actually nearest speciation event. This scenario, denoted as *deep coalescence*, can happen due to mutations leading to alternative forms of the same gene (called *alleles*) in ancestral populations, which is shortly denoted as *ancestral polymorphism*. If those alleles are sorted incompletely into several descendant species, two alleles of two distant species can be more similar compared to other parts of the genomes and, consequently, the respective gene tree may differ from the species tree. Notice that the larger the branches of a species tree (referring to the size of the respective ancestral population) and the shorter the branches (referring to short divergent times) the more unlikely a gene tree calculated for random genes matches the species tree [23].

Duplication-loss. The same scenario, as it is the case for deep-coalescence, can happen if genes are duplicated or lost. In this context, alternative forms of genes can arise if duplicated genes diverge by time simply due to different mutation patterns. Moreover, by loosing some of these alternative forms of a gene, these gene copies can be sorted incompletely into descendant species, which again can lead to discordant gene trees.

Next, we will introduce some reticulation events in which horizontal gene flow actually takes place between certain unrelated organisms. Notice that this can happen either between organisms that are closely related (introgression) or evolutionary more distant (hybridization). Moreover, it can affect parts of a genome that are rather small (recombination) or large (genome fusion). Hence, each reticulation event displayed by a phylogenetic network can be due to different reasons and it is the task of the respective scientists to give a reasonable interpretation of these events.

Recombination. Recombination involves the rearrangement of genetic material, e.g., by producing new combinations of certain genes (or alleles). In eukaryotes, for example, recombination automatically happens during meiosis as here, on the one hand, new combinations of the parental set of chromosomes are created and, on the other hand, genetic materiel of different chromosomes is exchanged through *crossing over*. Recombination, however, is not restricted to meiosis but can also happen during mitosis.

Horizontal gene transfer. The main mechanisms being responsible for horizontal gene transfer are in general all those biological processes transporting small genetic material from one organism to a different one including processes like *transformation*, *transduction*, and *conjugation*. The technique is often used, for example, by bacteria cells to exchange particular genes increasing the resistance against certain drugs, e.g., antibiotica [43]. Thus, horizontal gene transfer is supposed to play an important role especially in the evolution of bacteria.

Genome fusion. Genome fusion is a phenomenon that is assumed to happen quite rarely. However, it is considered to have played an important role in the origin of certain eukaryotes (as, for instance, proposed by the endosymbiotic theory).

Introgression. Introgression takes place if two distinct populations produce common offspring which then again reproduces with individuals from one of both parental populations (known as *backcrossing*). As a consequence, regarding the genetic material from those populations, introgression takes place from the one into the other population by gradually exchanging parts of their genomes.

Hybridization. In contrast to introgression, hybridization takes place between evolutionary more distant individuals, i.e., individuals belonging to different species. More precisely, *recombinational or homoploid hybrid speciation* is a mechanism affecting evolution by merging a sizable percentage of genomes of different species [45]. It has been discovered especially in plants [51, 59], but also in certain animals [56]. If two individuals each belonging to different species hybridize, a new species, containing genes from both parental individuals, can arise under the following certain circumstances. First of all, the resulting hybrid has to produce viable gametes, which is often a problem due to the two genetically different parental sets of chromosomes preventing a correct meiotic pairing. Second, if these two sets are similar enough and, thus, the hybrid is able to produce any progeny, its early recombinants have to find and successfully colonize its own unexploited niche that is different from either of its parents, which ensures a reduction of the gene flow between its parental genotypes. Due to these circumstances, homoploid hybrid speciation is considered as a rare phenomenon. However, there exist studies indicating that still about 10% of animal species are involved in hybridization (or introgression) [44].

1.3 Related work

This thesis deals with the computation of explicit (or evolutionary) networks, i.e., rooted phylogenetic networks outlining the evolution of certain taxa by explicitly indicating reticulation events. Those networks can be derived from various kinds of data involving clusters, characters, trees, and triplets or quartets [64]. In the following, however, we will only briefly describe those related work dealing with one certain type of rooted phylogenetic network, namely so-called *minimum hybridization networks*. A good overview of other types can be found in the survey of Huson and Scornavacca [34] as well as in the book of Morrison [48]. Moreover, a detailed description dealing with algorithms and applications of all types of rooted (as well as unrooted) phylogenetic networks can be found in the book of Huson *et al.* [33].

1.3.1 Hybridization Networks

Hybridization networks have originally been developed with respect to hybridization events merging genomes of different parental species. Its nomenclature, however, can be quite misleading as those networks can also model other types of reticulation events, e.g., those listed in Section 1.2. This means, in particular, when giving an interpretation of each reticulation event within a hybridization network, one should not only take hybridization events into account.

Hybridization networks are calculated for a set of rooted phylogenetic trees. More precisely, following the parsimonious principle, the objective of each mathematical method is to compute a rooted phylogenetic network displaying the set of input trees by inferring a minimum number of hybridization events (a formal definition will be given in Section 1.4). Often, a method solving this task focuses on two parts including the computation of sets of subtrees being contained in each of the input trees (so-called *acyclic agreement forests*) and the reattaching of these subtrees back together so that the resulting network displays each input tree. Consequently, there exist methods only dealing with the first part, the computation of the minimum number of hybridization events, shortly denoted as *minimum hybridization number*, necessary for reconciling the set of input trees into a rooted phylogenetic network.

Moreover, given only two input trees, the so-called *rooted subtree prune and regraft distance*, or rSPR-distance for short, describes a lower bound of the hybridization number. More specifically, the rSPR-distance of two phylogenetic trees T_1 and T_2 on the same set of taxa denotes how often one has to detach a subtree from one of those trees, say T_1, and reattach it to a different location in order to receive T_2. As the set of subtrees that has to be relocated fulfills most of the properties of an *acyclic agreement forest*, this problem is closely related to the problem of calculating minimum hybridization networks [12].

1.3.2 Inferring hybridization networks

Unfortunately, the problem of inferring minimum hybridization networks is a computationally hard task, even for the simplest case given only two rooted binary phylogenetic trees as input [13]. Thus, depending on the number and the type (binary/nonbinary) of input trees, the problem gets more and more complicated. Consequently, most of the related work only deals with two binary input trees, whereas for other input types there so far only exist none or at least only few works addressing these more complex problems. Notice that, in this thesis, we close this gap by presenting several algorithms being able to deal with all various kinds of input trees, especially those including more than two input trees containing nonbinary nodes.

Two binary trees. Important theoretical work dealing with hybridization networks for two rooted binary phylogenetic trees is the work of Baroni *et al.* [8, 9] and Bordewich *et al.* [11, 12, 13]. Among other things, these works show that the problem is *NP-hard* but still *fixed-parameter tractable*, which means that the problem is exponential in some parameter related to the problem itself, namely the hybridization number, and only polynomial in the size of the input trees.

Moreover, based on two binary input trees, there have been developed several algorithms calculating the rSPR-distance and the hybridization number [57, 69, 70, 71] as well as minimum hybridization networks [6]. Tools exploiting these algorithms are rSPR [70, 71] for the computation of the rSPR-distance as well as UltraNet [19] and Dendroscope [35] for the computation of minimum hybridization networks as well as exact hybridization numbers.

Furthermore, there exist heuristics for the computation of the exact hybridization number, e.g., the program CycleKiller [68] providing a 2-approximation algorithm, that can handle computational more complex instances (however, not guaranteeing the most parsimonious solution).

Multiple binary trees. Given more than two binary input trees, the problem still remains fixed-parameter tractable as recently shown by van Iersel and Linz [67]. So far, however, there exists only one method, namely the program PIRN [72, 73], calculating minimum hybridization networks for more than two rooted binary phylogenetic trees. In most cases, however, PIRN runs only reasonable efficient if the inferred number of hybridization events is relatively small as indicated by an upcoming simulation study presented in Section 3.5.

Two nonbinary trees. The only proper work dealing with the computation of minimum hybridization networks for two rooted nonbinary trees is the work describing the concept of the program TerminusEst [49]. Moreover, there also exists an implementation in Dendroscope calculating several hybridization networks for two nonbinary trees. So far, however, there does not exist a proper description of its underlying Autumn algorithm.

Again, there exist heuristics for the computation of the exact hybridization number, e.g., the program nonbinaryCycleKiller [68] providing a 2-approximation algorithm, that can handle computational more complex instances (however, not guaranteeing the most parsimonious solution).

Multiple nonbinary trees. To our knowledge, so far there does not exist any related work addressing the computation of minimum hybridization networks for multiple rooted nonbinary trees.

Notice that most of the presented methods, except Dendroscope [35], are only so-called *proof-of-concept methods*, which have not yet been integrated into user-friendly software packages providing a graphical user-interface. Moreover, each of the so far existing methods calculating minimum hybridization networks are ignoring the fact that such a network is typically not unique. This means, in particular, that those methods only report one (or sometimes at least a couple) of hybridization networks. In order to investigate reticulate evolution, however, biologists are interested in all of those networks, since only then it makes sense to apply filtering techniques for testing particular hypotheses.

In the upcoming part of this thesis we will present algorithms calculating minimum hybridization networks for all various kinds of input trees. Moreover, in Chapter 5 we will present a user-friendly software package, called Hybroscale [3], integrating these algorithms as well as giving support in analyzing the set of reported networks.

1.4 Basic notations

In this section, the basic notation and terminology is given that is used throughout this thesis. Notice that most of these definitions principally follow those given in the work of Huson *et al.* [33].

Graphs. A *graph* $G = (V, E)$ consists of a finite set of *nodes* V and a finite set of *edges* E, in which each edge e of E is represented by a set $\{u, v\}$ with $u, v \in V$. Given such an edge $e = \{u, v\}$, both nodes u and v are called the *endpoints of* e and we say e *connects* u *and* v. In general, one assumes that in a graph G two nodes are connected by at most one edge and there is no edge connecting the same node. More specifically, this means that in such a *simple graph* there does neither exist multi-edges nor self-loops.

Directed graphs. A *directed graph* (or *digraph*) $G = (V, E)$ consists of a finite set of *nodes* V and a finite set of *edges* E, in which each edge e of E is represented by a tuple (u, v) with $u, v \in V$. Given such an edge $e = (u, v)$, we say that e is directed from its *source* u to its *target* v. Moreover, in such a case we say e is an *out-going edge of* u and an *in-going edge of* v. Additionally, the *in- and out-degree of a node* v, denoted by $\delta^-(v)$ and $\delta^+(v)$ for short, refers to the number of its in- and out-going edges, respectively.

Given a directed graph $G = (V, E)$, a *path* (or *undirected path*) $\mathcal{P}$ of length k leading from v_0 to v_k is a sequence of nodes and edges $(v_0, e_0, v_1, e_1, \ldots, e_{k-1}, v_k)$ in which the two nodes v_i and v_{i+1}, with $0 \leq i < k$, are endpoints of the edge e_i. If $\mathcal{P}$ is a *directed path*, additionally each edge e_i in $\mathcal{P}$, with $0 \leq i < k$, has to be directed from v_i to v_{i+1}. In order to ease reading, throughout this thesis we will omit the edges of such a path $\mathcal{P}$ so that a path is simply denoted by a sequence of nodes. Now, given such a directed path $\mathcal{P}$ of length k, we say that $\mathcal{P}$ is a *directed cycle* if v_0 equals v_k and we say a directed graph is *acyclic* if it does not contain any directed cycles. Moreover, we say G is *(weakly) connected* if, by replacing all of its directed edges by undirected edges, for each pair of nodes u and v of the resulting undirected graph there exists a path leading from u to v.

Throughout this thesis, we will apply specific modifications to directed graphs which is done basically by deleting nodes and edges. Whereas an edge can be removed from a graph without applying further modifications, a node is deleted by first removing all of its in- and out-going edges and then by removing the node itself. A further modification is *suppressing nodes of both in- and out-degree* 1 which is done as follows. Let v be such a node of in- and out-degree 1 where (u, v) and (v, w) refers to its in- and outgoing edge, respectively. Then, v is suppressed by first inserting a new edge (u, w) connecting the two nodes u and w and then by deleting v (together with removing its in- and out-going edges).

Phylogenetic trees. A *rooted phylogenetic $\mathcal{X}$-tree* (or a *rooted phylogenetic tree on* $\mathcal{X}$) T is a directed acyclic connected graph whose edges are directed from the root to the leaves as defined in the following. There is exactly one node of in-degree 0, denoted as the *root* of T, and no nodes of both in- and out-degree 1. The set of nodes of out-degree 0 is called the *leaf set of* T and is labeled one-to-one by the *taxa set* $\mathcal{X}$, also denoted by $\mathcal{L}(T)$. Additionally, given a set $\mathcal{F}$ of rooted phylogenetic trees, by $\mathcal{L}(\mathcal{F})$ we refer to the union of each taxa set $\mathcal{L}(F)$ of each tree F in $\mathcal{F}$. Here, the taxa set $\mathcal{X}$ usually consists of particular species or genes whose evolution is outlined by T. The tree T is called *binary* if all of its nodes, except the root, provide an in-degree of 1 and if all of its nodes, except all leaves (the so-called *inner or internal nodes*) provide an out-degree of 2. Otherwise, in order to emphasize that a tree T can contain inner nodes of out-degree larger than 2, we call T a *multifurcating* or *nonbinary* tree.

Next, given a node v of a rooted phylogenetic $\mathcal{X}$-tree T, by $T(v)$ we refer to the subtree rooted at v containing all nodes and edges that can be reached from v. Moreover, the label set $\mathcal{L}(v)$ refers to the taxa set of $T(v)$. We will also call this label set $\mathcal{L}(v)$ the *cluster of* v, shortly denoted by $\mathcal{C}_T(v)$ (or $\mathcal{C}(v)$). In addition, the set of clusters of T, denoted by $\mathcal{C}(T)$, consists of all clusters represented by each node.

Given a *rooted phylogenetic $\mathcal{X}$-tree* T, let u and v be two nodes such that there is a directed path leading from u to v. Then, we say u is an *ancestor* of v and v is an *descendant* of u. Moreover, the *lowest common ancestor* of T in terms of a taxa set $\mathcal{X}' \subseteq \mathcal{X}$, denoted by $\text{LCA}_T(\mathcal{X}')$ for short, is the node v in T with $\mathcal{X}' \subseteq \mathcal{L}(v)$ such that there does not exist another node v' in T with $\mathcal{X}' \subseteq \mathcal{L}(v')$ and $\mathcal{L}(v') \subseteq \mathcal{L}(v)$. If there is an edge directed from a node u to a node v, we say that u is a *parent* of v and v is a *child* of u. Consequently, the *children* of a node u are those nodes being a target of an outgoing edge of u. Note that, if T is a binary tree, each inner-node has precisely two children and each node except the root precisely one parent.

Now, based on a rooted phylogenetic $\mathcal{X}$-tree T, throughout the thesis we will make use of the following specific kinds of subtrees each being contained in T in a certain way. Let $\mathcal{X}'$ be a set of taxa with $\mathcal{X}' \subseteq \mathcal{X}$, then, by $T(\mathcal{X}')$ we refer to the minimal connected subgraph of T only containing those leaves that are labeled by a taxon of $\mathcal{X}'$. A *restriction of* T *by* $\mathcal{X}'$, shortly denoted by $T|_{\mathcal{X}'}$, is a rooted phylogenetic tree that is obtained from $T(\mathcal{X}')$ by suppressing each node of both in- and out-degree 1. Moreover, we say that a rooted binary phylogenetic $\mathcal{X}'$-tree T' is a *pendant subtree of* T if we can detach T' from T by deleting exactly one of its edges. In addition, let $\mathcal{P}$ be a path in T and let $e = (u, v)$ be an edge with $u \in V(\mathcal{P})$ and $v \notin V(\mathcal{P})$. Then, we say the subtree $T(v)$ rooted at v is a *pendant subtree lying on* $\mathcal{P}$.

Phylogenetic networks. A *rooted phylogenetic network* N on $\mathcal{X}$ is a rooted connected digraph whose edges are directed from the root to the leaves as defined in the following. There is exactly one node of in-degree 0, namely the *root*, and no nodes of both in- and out-degree 1. The set of nodes of out-degree 0 is called the *leaf set of* N and is labeled one-to-one by the *taxa set* $\mathcal{X}$, also denoted by $\mathcal{L}(N)$. In contrast to a phylogenetic tree, such a network may contain undirected but not any directed cycles. Consequently, N can contain nodes of in-degree larger than or equal to 2, which are called *reticulation nodes*. Moreover, each edge that is directed into such a reticulation node is called *reticulation edge*.

Representation of trees and networks. In order to ease reading, throughout the thesis we will draw each tree and network from top to bottom with the root being at the top and the leaves at the bottom. Moreover, as in such a representation each edge is automatically direct downwards, we will draw each edge by a simple line without an arrowhead denoting its direction. Additionally, we will adopt a frequently used convention that each reticulation node of a phylogenetic network with in-degree of at least 2 has out-degree 1. Note that this is not a necessary property of a phylogenetic network but helps the reader to distinguish between reticulation nodes and other nodes within the graph.

1.4.1 Further definitions for binary trees

In this section, we give further definitions referring to *binary* phylogenetic trees.

Hybridization Networks. A *hybridization network* N for a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}'$-trees, with $\mathcal{X}' \subseteq \mathcal{X}$, is a rooted phylogenetic network on $\mathcal{X}$ *displaying* $\mathcal{T}$ (i.e., *contains an embedding of each tree* T *in* $\mathcal{T}$). More precisely, this means that for each tree T in $\mathcal{T}$ there exists a set $E' \subseteq E(N)$ of reticulation edges *referring* to T. More specifically, this means that T can be derived from N by conducting the following steps.

(1) First, delete each reticulation edge from N that is not contained in E'.

(2) Then, remove each node whose corresponding taxon is not contained in $\mathcal{X}'$.

(3) Next, remove each unlabeled node of out-degree 0 repeatedly.

(4) Finally, suppress each node of both in- and out-degree 1.

From a biological point of view, this means that N displays T (i.e., contains an embedding of T) if each speciation event of T is reflected by N. Moreover, each internal node of in-degree 1 represents a speciation event and each internal node providing an in-degree of at least 2 represents a reticulation event or, in terms of hybridization, a hybridization event. This means, in particular, that such a latter node represents an individual whose genome is a *chimaera* of several parents. Thus, such a node v of in-degree larger than or equal to 2 is called *hybridization node* (or reticulation node) and each edge directed into v is called *hybridization edge* (or reticulation edge). Moreover, each edge that is not a hybridization edge is called *tree edge*.

Now, based on those hybridization nodes, the *reticulation number* $r(N)$ of a hybridization network N is defined by

$$r(N) = \sum_{v \in V : \delta^-(v) > 0} \big(\delta^-(v) - 1\big) = |E| - |V| + 1, \tag{1.1}$$

where V denotes the node set and E the edge set of N. Next, based on the definition of the reticulation number, for a set $\mathcal{T}$ of phylogenetic $\mathcal{X}$-trees the (minimum or exact) *hybridization number* $h(\mathcal{T})$ is defined by

$$h(\mathcal{T}) = \min\{r(N) : \text{N is a hybridization network displaying } \mathcal{T}\}. \tag{1.2}$$

Lastly, we call a *hybridization network* N for a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees a *minimum hybridization network*, if $r(N) = h(\mathcal{T})$.

Notice that the computation of the hybridization number for just two rooted binary phylogenetic $\mathcal{X}$-trees is an *NP-hard* problem [14] which is, however, still fixed-parameter tractable [13]. More specifically, this means that the problem is exponential in some parameter related to the problem itself, namely the hybridization number, but only polynomial in the size of the input trees, which is an important feature facilitating the development of practical algorithms.

Forests. Let T be a rooted nonbinary phylogenetic $\mathcal{X}$-tree T. Then, we call any set of rooted nonbinary phylogenetic trees $\mathcal{F} = \{F_1, \dots, F_k\}$ with $\mathcal{L}(\mathcal{F}) = \mathcal{X}$ a *forest on* $\mathcal{X}$, if we have for each pair of trees F_i and F_j that $\mathcal{L}(F_i) \cap \mathcal{L}(F_j) = \emptyset$. Moreover, if additionally for each component F in $\mathcal{F}$ the tree $T|_{\mathcal{L}(F)}$ equals F, we say that $\mathcal{F}$ is a *forest for* T.

Binary agreement forests. For technical purpose, the definition of agreement forests is based on two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 whose roots are marked by a unique taxon $\rho \notin \mathcal{X}$, which can be done in two different ways as follows.

- Either, let r_i be the root of the tree T_i with $i \in \{1, 2\}$. Then, we first create a new node v_i labeled by a new taxon $\rho \notin \mathcal{X}$ and then attach this node to r_i by inserting the edge (v_i, r_i).
- Or, again let r_i be the root of the tree T_i with $i \in \{1, 2\}$. Then, we first create a new node v_i as well as a new leaf ℓ_i labeled by a new taxon $\rho \notin \mathcal{X}$ and then attach these nodes to r_i by inserting the two edges (v_i, r_i) and (v_i, ℓ_i).

In both cases, v_1 and v_2 is the new root of T_1 and T_2, respectively. Moreover, since we consider ρ as being a new taxon, the taxa set of both trees is $\mathcal{X} \cup \{\rho\}$.

Now, assuming we have given two trees T_1 and T_2 whose roots are marked by a unique taxon ρ, then, a *binary agreement forest for* T_1 *and* T_2 is a set of components $\mathcal{F} = \{F_\rho, F_1, \dots, F_{k-1}\}$ on $\mathcal{X} \cup \{\rho\}$ satisfying the following properties.

(1) Each component F_i with taxa set $\mathcal{X}_i$ equals $T_1|_{\mathcal{X}_i}$ and $T_2|_{\mathcal{X}_i}$.

(2) There is exactly one component, denoted as F_ρ, with $\rho \in \mathcal{L}(F_\rho)$.

(3) Let $\mathcal{X}_\rho, \mathcal{X}_1, \dots, \mathcal{X}_{k-1}$ be the taxa sets of the components $F_\rho, F_1, \dots, F_{k-1}$. All trees in $\{T_1(\mathcal{X}_i) | i \in \{\rho, 1, \dots, k-1\}\}$ and $\{T_2(\mathcal{X}_i) | i \in \{\rho, 1, \dots, k-1\}\}$ are node disjoint subtrees of T_1 and T_2, respectively.

In order to ease reading, throughout the thesis we will call a binary agreement forest simply an agreement forest. Moreover, we call an agreement forest a *maximum agreement forest*, if this agreement forest is of minimal size. This means, in particular, that there does not exist another set of components of smaller size satisfying the conditions of an agreement forest listed above.

Lastly, there is another important property an agreement forest can satisfy. We call an agreement forest $\mathcal{F}$ for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 *acyclic*, if there is no directed cycle in the underlying *ancestor-descendant graph* $AG(T_1, T_2, \mathcal{F})$, which is defined as follows. First, this graph consists of nodes each corresponding to precisely one component of $\mathcal{F}$. Moreover, two different nodes F_i and F_j of this graph are connected via a directed edge (F_i, F_j), if,

(i) regarding T_1, the root of $T_1(\mathcal{X}_i)$ is an ancestor of the root of $T_1(\mathcal{X}_j)$

(ii) or, regarding T_2, the root of $T_2(\mathcal{X}_i)$ is an ancestor of the root of $T_2(\mathcal{X}_j)$,

where $\mathcal{X}_i \subseteq \mathcal{X}$ and $\mathcal{X}_j \subseteq \mathcal{X}$ refers to the taxa set of the two components F_i and F_j, respectively. Again, we call an acyclic agreement forest consisting of a minimum number of components a *maximum acyclic agreement forest*. Notice that for a maximum acyclic agreement forest containing k components there exists a hybridization network with hybridization number $k-1$ [8]. This means, in particular, if a maximum acyclic agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 contains only one component, T_1 equals T_2.

Acyclic orderings. Given an agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, then, if $\mathcal{F}$ is acyclic and, thus, $\text{AG}(T_1, T_2, \mathcal{F})$ does not contain any directed cycles, one can compute an *acyclic ordering* as already described in the work of Baroni *et al.* [9]. First, select the node v_ρ corresponding to F_ρ of in-degree 0 and remove v_ρ together with all its incident edges. Next, again choose a node v_1 of in-degree 0 and remove v_1. By continuing this way, until finally all nodes have been removed, one receives the ordering $\Pi_V = (v_\rho, v_1, \dots, v_k)$ containing all nodes in $\text{AG}(T_1, T_2, \mathcal{F})$. In the following, we call the ordering $(F_\rho, F_1, \dots, F_k)$ of components corresponding to each node in Π_V an *acyclic ordering of* $\mathcal{F}$. Notice that, as during each of those steps there can occur several nodes of in-degree 0, especially if $\mathcal{F}$ contains components consisting only of isolated nodes, such an acyclic ordering is in general not unique.

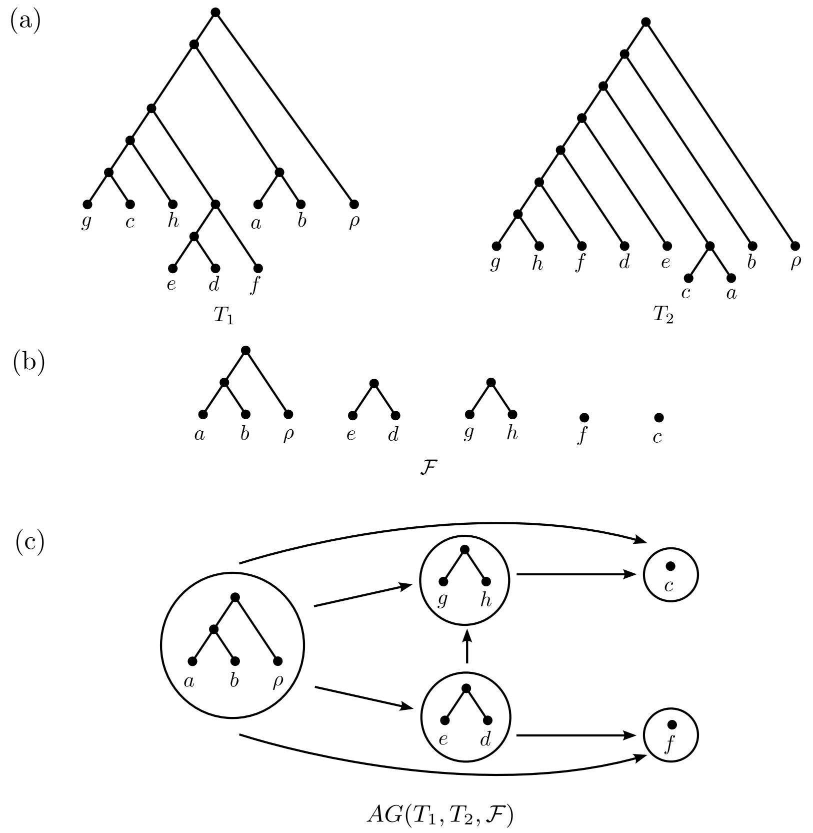


Figure 1.1: (a) Two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 with taxa set $\mathcal{X} = \{a, b, c, d, e, f, g, h, \rho\}$. (b) An acyclic agreement forest $\mathcal{F}$ for T_1 and T_2 in acyclic ordering. (c) The directed graph $AG(T_1, T_2, \mathcal{F})$ not containing any directed cycles and, thus, $\mathcal{F}$ is acyclic.

1.4.2 Further definitions for nonbinary trees

In this section, we give further definitions referring to *nonbinary* phylogenetic trees.

Phylogenetic trees. A rooted nonbinary phylogenetic $\mathcal{X}$-tree T can contain *mulitfurcating* or *nonbinary* nodes, which are nodes of out-degree larger than or equal to 3. We say a rooted phylogenetic $\mathcal{X}$-tree T' is a *refinement* of T, if we can obtain T from T' by *contracting* some of its edges. More precisely, an edge $e = (u, v)$, with C_v being the set of children of v, is contracted by first deleting v together with all of its adjacent edges (including e) and then by reattaching each node c_i in C_v back to u by inserting a new edge (u, c_i). Moreover, in this context we further say that T' is a *binary refinement of* T, if T' is binary.

Similarly, if T' is a refinement of T, we can obtain T' from T by *resolving* some of its multifurcating nodes in the following way (cf. Fig. 1.2). Let v be a multifurcating node and let $C_v = \{c_1, \dots, c_n\}$ be its set of children, then, we can resolve v as follows. First, a new node w is created, which is attached to v by inserting a new edge (v, w). Second, we select a subset C'_v of C_v, with $1 < |C'_v| < |C_v|$, and, finally, we prune each node c_i of C'_v

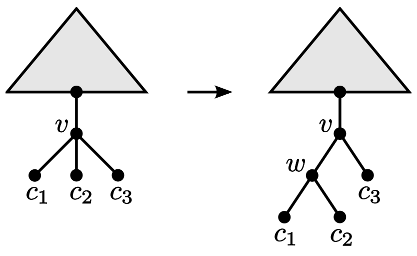


Figure 1.2: Resolving a multifurcating node v by reattaching c_1 and c_2 to a new inserted node w.

from v and reattach c_i to w by inserting a new edge (w, c_i).

Hybridization networks. A hybridization network N for a set of rooted nonbinary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$ is a rooted phylogenetic network on $\mathcal{X}$ *displaying* a refinement T_i' of each tree T_i in $\mathcal{T}$. More precisely, this means that for each tree T_i in $\mathcal{T}$ there exists a set $E_i' \subseteq E(N)$ of reticulation edges *referring* to its refinement T_i'. This means, in particular, that we can obtain the tree T_i' from N by first deleting all reticulation edges that are not contained in E_i' and then suppress all nodes of both in- and out-degree 1. In this context, a reticulation edge (or hybridization edge) is an edge that is directed into a node with in-degree larger than or equal to 2, which is denoted as reticulation node (or hybridization node).

Given a hybridization network for a set of rooted nonbinary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$, the *reticulation number* $r(N)$ is defined by

$$r(N) = \sum_{v \in V : \delta^-(v) > 0} (\delta^-(v) - 1) = |E| - |V| + 1, \tag{1.3}$$

where V refers to the set of nodes of N and $\delta^-(v)$ denotes the in-degree of a node v in V. Moreover, based on the definition of the reticulation number, the (minimum or exact) hybridization number $h(\mathcal{T})$ for $\mathcal{T}$ is defined by

$$h(\mathcal{T}) = \min\{r(N) : N \text{ displays a refinement of each } T_i \in \mathcal{T}\}. \tag{1.4}$$

A hybridization network displaying a set of rooted nonbinary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$ with minimum hybridization number $h(\mathcal{T})$ is called a *minimum hybridization network*. Notice that even in the simplest case, if $\mathcal{T}$ consists only of two rooted binary phylogenetic $\mathcal{X}$-trees, the problem of computing the hybridization number is known to be *NP-hard* but fixed-parameter tractable [11, 14], which means that the problem is exponential in some parameter related to the problem itself, namely the hybridization number of $\mathcal{T}$, but only at most polynomial in its input size, which is, in this context, the number of nodes and edges in $\mathcal{T}$.

Forests. Let T be a rooted nonbinary phylogenetic $\mathcal{X}$-tree T. Then, we call any set of rooted nonbinary phylogenetic trees $\mathcal{F} = \{F_1, \ldots, F_k\}$ with $\mathcal{L}(\mathcal{F}) = \mathcal{X}$ a *forest on* $\mathcal{X}$, if we have for each pair of trees F_i and F_j that $\mathcal{L}(F_i) \cap \mathcal{L}(F_j) = \emptyset$. Moreover, we say that $\mathcal{F}$

is a *forest for* T, if additionally for each component F in $\mathcal{F}$ the tree F is a refinement of $T|_{\mathcal{L}(F)}$.

Lastly, given two forests $\mathcal{F}$ and $\hat{\mathcal{F}}$ for T, we say that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$, if for each component $\hat{F}$ in $\hat{\mathcal{F}}$ there exists a component F in $\mathcal{F}$ such that $\hat{F}$ is a binary refinement of F.

Nonbinary agreement forests. Given two rooted nonbinary phylogenetic $\mathcal{X}$-trees T_1 and T_2. For technical purpose, we consider the root of both trees T_1 and T_2 as being a node that has been marked by new taxon $\rho \notin \mathcal{X}$. More precisely, let r_i be the root of the tree T_i with $i \in \{1, 2\}$. Then, we first create a new node v_i as well as a new leaf ℓ_i labeled by a new taxon $\rho \notin \mathcal{X}$ and then attach these nodes to r_i by inserting the two edges (v_i, r_i) and (v_i, ℓ_i) such that v_i is the new root of the resulting tree. Now, an *agreement forest* for two so marked trees T_1 and T_2 is a forest $\mathcal{F} = \{F_\rho, F_1, \dots, F_k\}$ on $\mathcal{X} \cup \{\rho\}$ satisfying the following three conditions.

(1) Each component F_i with taxa set $\mathcal{X}_i$ equals a refinement of $T_1|_{\mathcal{X}_i}$ and $T_2|_{\mathcal{X}_i}$, respectively.

(2) There is exactly one component, denoted as F_ρ, containing ρ.

(3) Let $\mathcal{X}_\rho, \mathcal{X}_1, \dots, \mathcal{X}_k$ be the taxa sets corresponding to $F_\rho, F_1, \dots, F_k$. All trees in $\{T_1(\mathcal{X}_i) | i \in \{\rho, 1, \dots, k\}\}$ and $\{T_2(\mathcal{X}_i) | i \in \{\rho, 1, \dots, k\}\}$ are edge disjoint subtrees of T_1 and T_2, respectively.

Moreover, a *maximum agreement forest* for two rooted nonbinary phylogenetic $\mathcal{X}$-trees T_1 and T_2 is an agreement forest of minimal size, which implies that there does not exist a smaller set of components fulfilling the properties of an agreement forest for T_1 and T_2 listed above. Additionally, we call an agreement forest $\mathcal{F}$ for T_1 and T_2 *acyclic*, if its underlying *ancestor-descendant graph* $AG(T_1, T_2, \mathcal{F})$ does not contain any directed cycles (cf. Fig. 1.3). This directed graph contains one node corresponding to precisely one component of $\mathcal{F}$ and an edge (F_i, F_j) for a pair of its nodes F_i and F_j, with $i \neq j$, if,

(i) regarding T_1, there is a path leading from the root of $T_1(\mathcal{X}_i)$ to the root of $T_1(\mathcal{X}_j)$ containing at least one edge of $T_1(\mathcal{X}_i)$,

(ii) or, regarding T_2, there is a path leading from the root of $T_2(\mathcal{X}_i)$ to the root of $T_2(\mathcal{X}_j)$ containing at least one edge of $T_2(\mathcal{X}_i)$.

In this context, $\mathcal{X}_i \subseteq \mathcal{X}$ and $\mathcal{X}_j \subseteq \mathcal{X}$ refers to the set of taxa that are contained in F_i and F_j, respectively. Again, we call an acyclic agreement forest consisting of a minimum number of components a *maximum acyclic agreement forest.*

Acyclic orderings. Now, if $\mathcal{F}$ is acyclic and, thus, $\mathrm{AG}(T_1, T_2, \mathcal{F})$ does not contain any directed cycles, based on this graph one can compute an acyclic ordering for $\mathcal{F}$ in the same way as already described in the binary case.

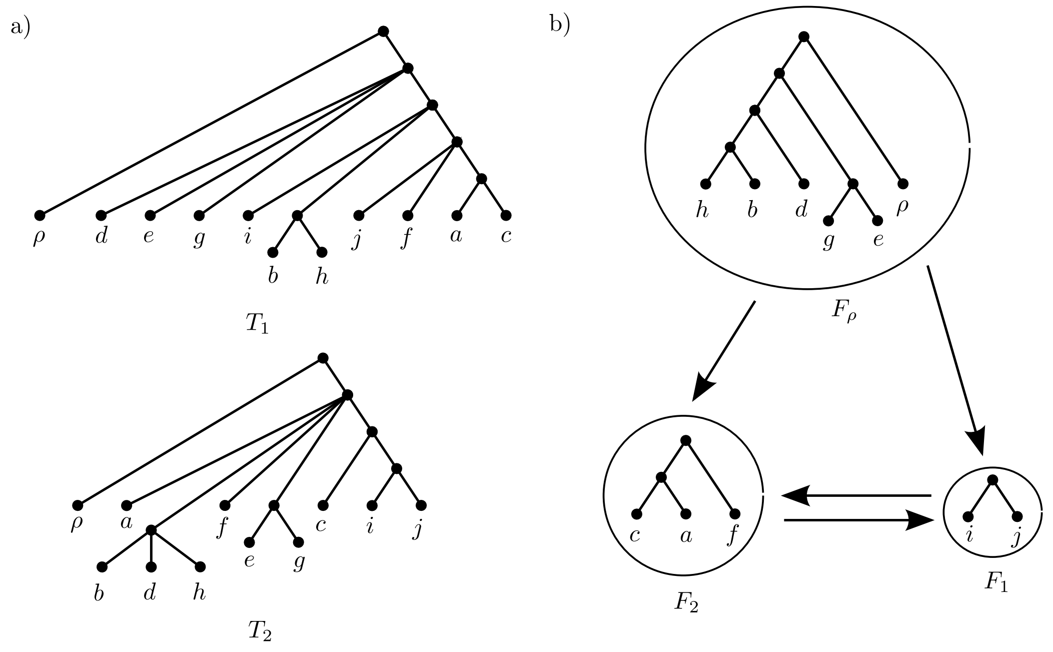


Figure 1.3: (a) Two rooted nonbinary phylogenetic $\mathcal{X}$-trees T_1 and T_2. (b) The ancestor-descendant graph $AG(T_1, T_2, \mathcal{F})$ with $\mathcal{F} = \{F_\rho, F_1, F_2\}$. Notice that the component corresponding to each node of the graph is drawn inside.

1.5 Organization of this thesis

The upcoming part of this thesis is organized as follows.

Chapter 2. In this chapter, we describe an approach calculating the rSPR-distance, the hybridization number as well as a certain set of representative minimum hybridization networks for *two* rooted *binary* phylogenetic $\mathcal{X}$-trees T_1 and T_2. Our approach is based on the concept of agreement forests whose elements can then be used in a subsequent step to construct a minimum hybridization network displaying T_1 and T_2. Thus, the major step of our approach is the computation of such agreement forests which is done by the algorithm allMAAFs. A simulation study, conducted on our own synthetic dataset, indicates that, at this particular time, an implementation of our approach was much faster than all so far existing methods.

The algorithm allMAAFs is the first non-naive approach that enables the computation of all maximum acyclic agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees, which has been shown by a formal proof [57]. To further improve its efficiency, we applied some algorithmic modifications reducing the number of computational steps. Again, by conducting a simulation study on a synthetic dataset, we indicate that these modifications significantly improve the practical runtime of the original algorithm. Additionally, we show how the algorithm TerminusEst, an algorithm calculating the hybridization number for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees, can be used to further speed up the computation of all maximum acyclic agreement forests.

My contribution: *The algorithm* allMAAFs *has been developed during my diploma thesis that was supervised by Celine Scornavacca and Daniel Huson. This work excludes its proof of correctness that has been established afterwards by Celine Scornavacca and Simone Linz. Everything else of this chapter is continuing work, which has been elaborated by myself subsequent to my diploma thesis. More specifically, this includes the development (and implementation) of three modifications of the algorithm* allMAAFs *including the respective proofs of correctness as well as the development and implementation of a modified version of the algorithm* TerminusEST.

Publications: *The following publications deal with the content of this chapter.*

B. Albrecht, C. Scornavacca, A. Cenci, and D. H. Huson. Fast computation of minimum hybridization networks. *Bioinformatics*, 28 (2): 191-197, 2011.

C. Scornavacca, S. Linz, and B. Albrecht. A First Step Toward Computing All Hybridization Networks For Two Rooted Binary Phylogenetic Trees. *Journal of Computational Biology*, 19(11): 1227-1242, 2012.

B. Albrecht. Fast computation of all maximum acyclic agreement forests. *arXiv:1512.05656*, 2015.

Chapter 3. Here, we present the algorithm allHNetworks being an extension of the approach of the previous chapter so that now minimum hybridization networks can be calculated for a set $\mathcal{T}$ of rooted *binary* phylogenetic $\mathcal{X}$-trees. This is done, principally, by step-wise calculating maximum acyclic agreement forests for an input tree T and some embedded tree of a so far computed network N. Those elements of such an agreement forests are then inserted into N by creating further reticulation edges so that T is displayed in the resulting network. Consequently, the algorithm allMAAFs of the previous chapter is an important part of this extended approach.

Moreover, by comparing our approach to the software package PIRNv2.0, we demonstrate that an implementation of the algorithm allHNetworks is much faster than all so far existing methods. Additionally, we show how a set containing all representative minimum hybridization networks can be used in order to calculate specific support values indicating which hybridization events might played an important role during evolution.

My contribution: *Everything in this chapter has been done by myself including the development of the algorithm* allHNetworks, *the proof establishing its correctness, and the simulation study comparing our approach to the software package* PIRNv2.0 *(as well as integrating our approach into the program* Hybroscale *as mentioned below).*

Publications: *The following publications deal with the content of this chapter.*

B. Albrecht. Computing all hybridization networks for multiple binary phylogenetic input trees. *BMC Bioinformatics*, 16:236, 2015.

B. Albrecht. Computing hybridization networks for multiple rooted binary phylogenetic trees by maximum acyclic agreement forests. *arXiv:1408.3044*, 2015.

Chapter 4. In this chapter, we first present an algorithm — called allMulMAFs — calculating all relevant maximum agreement forests for two rooted nonbinary phylogenetic $\mathcal{X}$-trees. Based on a proof showing the correctness of a modification of the algorithm allMAAFs presented in Chapter 2, we give a detailed formal proof showing the correctness of the algorithm allMulMAFs. Next, by introducing the concept of an expanded cycle graph, we extend this algorithm to the algorithm allMulMAAFs now calculating all relevant maximum acyclic agreement forests for two nonbinary phylogenetic $\mathcal{X}$-trees.

Moreover, based on the algorithm allMulMAAFs, we introduce the algorithm allMulHNetworks, which extends our algorithm of Chapter 3 so that now minimum hybridization networks for a set $\mathcal{T}$ of rooted *nonbinary* phylogenetic $\mathcal{X}$-trees can be calculated.

My contribution: *Everything in this chapter has been done by myself including the development of the three algorithms* allMulMAFs, allMulMAAFs, *and* allMulHNetworks *as well as each proof dealing with the correctness of the first two algorithms.*

Publications: *The following publication deals with the content of this chapter.*

B. Albrecht. Computing a relevant set of nonbinary maximum acyclic agreement forests. *arXiv:1512.05703*, 2015.

Chapter 5. Here, we first introduce particular network constraints that can be used to filter a set of rooted phylogenetic networks under both mathematical and biological views. In this connection, we want to point out that phylogenetic networks are not free of interpretive challenges and, in order to improve the interpretation of hybridization networks, it is now time to think about techniques facilitating this interpretation step.

Next, we introduce a naive method re-rooting a set of rooted phylogenetic $\mathcal{X}$-trees in respect of hybridization numbers.

Finally, we present our software package HYBROSCALE. More precisely, we show how HYBROSCALE can be applied in order to first reconcile a set of input trees into minimum hybridization networks and then how to classify the set of reported networks in respect of specific constraints. Moreover, we give a short description of a layout algorithm assigning coordinates to each node of a tree or network in order to visualize the respective graph on the computer screen.

My contribution: *Everything in this chapter has been done by myself including the development and the application of all network constraints as well as of the method re-rooting trees. The software package* HYBROSCALE *has been implemented by myself which includes the implementation of all integrated algorithms as well as the implementation of its graphical user-interface.*

Publications: *The following publication deals with the content of this chapter.*

B. Albrecht. Hybroscale: a software for studying reticulate evolution. *In preparation*, 2015.

Chapter 2

Hybridization networks for two binary trees

In this chapter, we first present an approach calculating a representative set of minimal hybridization networks for two rooted binary phylogenetic $\mathcal{X}$-trees. Here, the determinant step is based on the algorithm allMAAFs calculating all maximum acyclic agreement forests for such two input trees. The workflow of this algorithm is briefly demonstrated and the main ideas of its published proof of correctness are briefly motivated. Next, we introduce three modifications of this algorithm by additionally discussing its correctness as well as indicating its benefit regarding its practical runtime obtained from a synthetic dataset. Finally, we present a modified version of the software package TerminusEst, which is so far the fastest algorithm calculating minimum hybridization numbers for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Furthermore, we show how our approach calculating minimal hybridization networks can benefit from this method.

2.1 Reduction rules

Given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, before applying a method calculating hybridization networks for both trees one can apply two well-known reduction rules, namely the *subtree reduction* and the *cluster reduction*. Through those two rules the computational complexity of both input trees can be reduced which often significantly improves the practical runtime of the respective method.

2.1.1 Subtree reduction

Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees, then the subtree reduction transforms both trees into two rooted binary phylogenetic $\mathcal{X}$-trees T_1' and T_2' by removing all maximal pendants subtrees T' of size ≥ 2 occurring in both trees T_1 and T_2 as follows. Let v_1 and v_2 be the root of such a maximal pendant subtree T' in T_1 and T_2, respectively.

Then, first all nodes that can be reached from v_1 and v_2 are deleted and then v_1 and v_2 is labeled by a unique taxon $a \notin \mathcal{X}$.

2.1.2 Cluster reduction

Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees and let $A \subset \mathcal{X}$ be a *cluster* with $A \geq 2$ such that both trees T_1 and T_2 contain a specific node v_1 and v_2, respectively, with $\mathcal{L}(v_1) = \mathcal{L}(v_2) = A$. Then, the cluster reduction generates two tree pairs $(T_1|_A, T_2|_A)$ and (T_1', T_2') in which the latter tree pair is generated by detaching both subtrees rooted at node v_1 and v_2, respectively. More precisely, the two trees T_1' and T_2' are obtained from T_1 and T_2, respectively, by first deleting each node that can be reached from v_1 and v_2 and then by labeling v_1 and v_2 by a unique taxon $a \notin \mathcal{X}$.

Theorem 1, which has been established in the work of Baroni *et al.* [9, Theorem 1], shows that by applying the cluster reduction to both trees T_1 and T_2, the hybridization number for those two trees can still be calculated by simply summing up the hybridization number of the two resulting tree pairs $(T_1|_A, T_2|_A)$ and (T_1', T_2').

Theorem 1 ([9]). *Given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 containing a common cluster $A \subset \mathcal{X}$, then $h(\{T_1, T_2\}) = h(\{T_1|_A, T_1|_A\}) + h(\{T_1', T_2'\})$.*

This means, in particular, whereas the subtree reduction simply reduces the size of the problem instance, the cluster reduction separates the original problem into several subproblems, which can then be solved on its own. As those subproblems are in general less complex than the original problem itself, by applying a cluster reduction beforehand the practical runtime of a method calculating the hybridization number can be significantly improved even though this method has to be executed on multiple tree pairs instead of just one.

Now, in order to maximize its efficiency, the cluster reduction is usually applied in a way that ensures a maximal number of resulting tree pairs. More specifically, this is done by recursively applying the reduction rule to a minimal cluster of both input trees (cf. Fig 2.1).

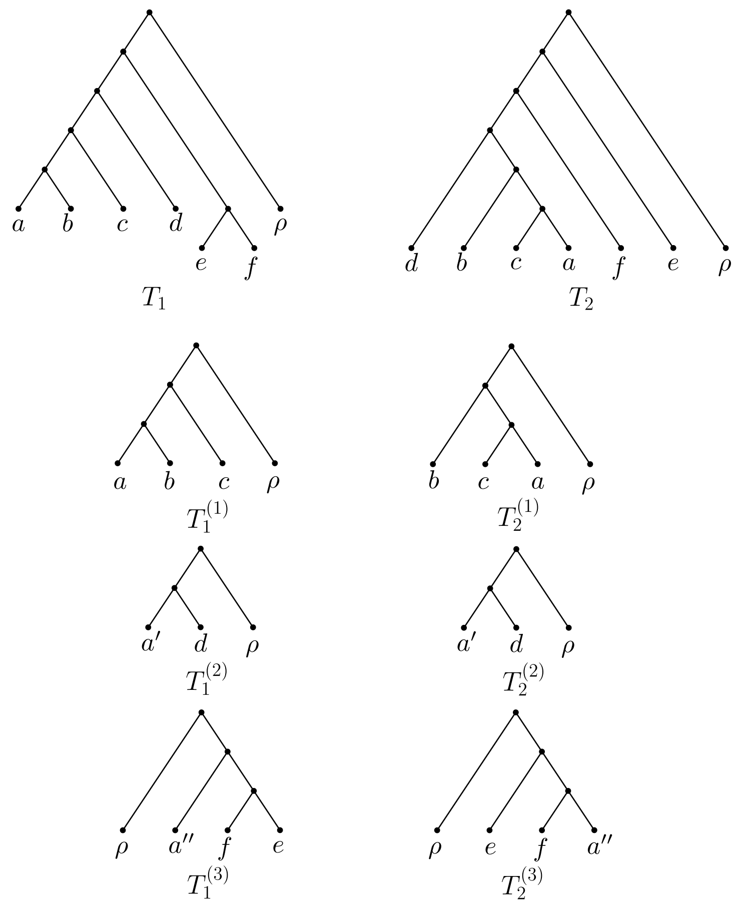


Figure 2.1: The cluster reduction dividing the tree pair (T_1, T_2) into its minimum common clusters resulting in three tree pairs $(T_1^{(1)}, T_2^{(1)})$, $(T_1^{(2)}, T_2^{(2)})$, and $(T_1^{(3)}, T_2^{(3)})$.

2.2 A fast approach for computing minimum hybridization networks

In this section, we focus on the following computational task. Given two rooted binary phylogenetic trees T_1 and T_2 on the same set of taxa, compute a *representative* set of hybridization networks each displaying both trees with minimum hybridization number. In this context, such a representative set provides exactly one network each corresponding to one of all maximum acyclic agreement forests for T_1 and T_2.

Our algorithm adopts some techniques presented in previous works of Baroni *et al.* [8] and Bordewich *et al.* [14]. Moreover, it is an extension of the approach introduced in previous works of Whidden *et al.* [71, 70] that is now able to solve an extended computational task as described above. We are aware of the fact that our work raises some important theoretical questions regarding the correctness of the presented approach. However, in this section we want to focus on the practical aspects of our algorithm and, thus, all theoretical questions have been addressed in an extra paper of Scornavacca *et al.* [57] whose content is briefly summarized in Section 2.3.

To apply the algorithm to biological data, we integrated our method as a plug-in into Dendroscope 3 [35], which is an freely available[1] interactive viewer for rooted phylogenetic networks. The software is platform independent and, thus, runs on all major operating systems. Usually, in our experience, the set of representative hybridization networks can be quite large. Thus, our implementation gives the user support in identifying the number of networks containing certain kinds of hybridization events so that a user can quickly figure out those hybridization events that are contained in most of the networks and, thus, might have played an important role during evolution. In order to improve the interpretation of hybridization events within computed networks, we additionally assign certain edge labels that help to understand how both initial input trees are displayed in there. Moreover, apart from computing minimum hybridization networks, we provide two further variants of our algorithm that can be used for the computation of the *rooted rSPR-distance* and the computation of the *hybridization number* for two binary rooted phylogenetic trees.

In order to indicate the efficiency of our developed algorithm, we have performed a simulation study comparing its implementation to HybridNET [17, 18], which was at this particular time the best available software for computing the exact hybridization number for two rooted binary phylogenetic trees on the same set of taxa. As shown in Section 2.2.3, this study indicates that our approach was, at this particular time, much faster than all so far existing methods. Notice that, regarding the computation of the hybridization number, in the meantime our program has been outperformed by the program *TerminusEst* [49] (cf. Sec 2.5), which is currently the fastest available tool for this purpose. Due to this fact, we decided to integrated this approach, which will be discussed separately in Section 2.5.

Finally, we end this section by giving a short illustration of how our method can be used for the investigation of hybridization events that might had an impact on the evolution of the *Aegilops/Triticum* genera.

[1]`www.dendroscope.org`

2.2.1 Further definitions

In a first step, we give further definitions that are crucial for the description and the discussion of our approach calculating a representative set of hybridization networks for two rooted binary phylogenetic trees. Again, these definitions follow those definitions given in the work of Huson *et al.* [33].

Agreement forests. In this section, given an agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees, we assume that both roots of the two trees are nodes of out-degree 1 that are labeled by the taxon ρ.

rSPR-moves. Given a rooted phylogenetic tree T, a *rooted Subtree Prune and Regraft move*, shortly denoted as *rSPR-move*, is a tree operation including two edges $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$ which is performed as follows. First, the subtree T' rooted at w_1 is pruned by removing e_1 followed by suppressing v_1 if its in- and out-degree equals 1. Second, the subtree T' is re-grafted to e_2, which is done as follows. In a first step, a new node z is created. Next, e_2 is replaced by two edges (v_2, z) and (z, w_2) and, finally, the root w_1 of T' is connected to z. Notice that if T' is re-grafted to the root ρ of T, first a new root ρ' has to be created which is then connected to ρ and w_1.

rSPR-distance. Given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, then the exact rSPR-distance of T_1 and T_2 is defined as the minimum number of rSPR-moves that has to be performed to obtain T_2 from T_1. Notice that the problem of computing the exact rSPR-distance of two rooted binary phylogenetic trees sharing the same set of taxa is known to be NP-hard, but fixed-parameter tractable [12].

2.2.2 The algorithm

In this section, we give a high level description of our developed algorithm computing a *representative* set of minimum hybridization networks for two rooted binary phylogenetic trees T_1 and T_2 sharing the same set of taxa $\mathcal{X}$. More precisely, this set of networks consists of exactly one minimum hybridization network for each maximum acyclic agreement forest of T_1 and T_2. As the computation of the hybridization number and the computation of the rSPR-distance of T_1 and T_2 are two problems that are strongly connected to the problem of computing minimum hybridization networks, our algorithms provides two additional variants solving these two kinds of problems.

Our algorithm is divided into three consecutive phases, namely the *reduction phase*, the *exhaustive search phase*, and the *final phase*. Whereas the first and the last phase have a polynomial runtime, the exhaustive search phase solves an *NP-hard* problem, which means that its theoretical worst-case runtime is exponential. As a direct consequence, the most computational time during the execution of our algorithm, is spent on performing the exhaustive search phase. Thus, before entering this phase, we have applied a reduction phase reducing the complexity of the input trees, which in general significantly decreases the practical runtime of the exhaustive search phase and, consequently, also the practical runtime of the entire algorithm. Notice that at the end of the algorithm, in the final phase, one still has to undo each of those reduction steps. These steps, however, are of rather low complexity and, thus, do not have an large impact on the practical runtime.

Reduction phase. The reductions phase tries to identify certain components of both input trees that can be replaced in a specific way such that the result of the exhaustive search phase can be used to restore the output corresponding to the unreduced input trees. There are two well-known reduction steps of this kind that are performed by our algorithm, namely the subtree reduction and the cluster reduction as previously described in Section 2.1. A more detailed description of each single step can be looked up in the work of Huson *et al.* [33].

Exhaustive search phase. The exhaustive search phase is the determinant step of the algorithm as, compared to all other steps, it provides the highest complexity. Its input consists of one tree pair or, due to the previously applied cluster reduction, of a set of tree pairs each referring to a particular common cluster of both initial input trees. Depending on whether the goal is to compute the rSPR-distance, the exact hybridization number or the set of representative minimum hybridization networks, its output consists of one maximum agreement forest, one maximum acyclic agreement forest, or all maximum acyclic agreement forests.

For the computation of a maximum agreement forest, whose size minus one directly corresponds to the rSPR-distance of a tree pair [12], our algorithm refers to the one described in the work of Whidden *et al.* [71]. Moreover, we use the 3-approximation algorithm, given in Whidden and Zeh [70], calculating a lower bound of the rSPR-distance, for identifying a starting point ($k > 0$) for the exhaustive search phase. This usually leads to a significant speedup, especially, for those tree pairs providing a large rSPR-distance. The work of Whidden *et al.* [71] additionally contains an algorithm computing a maximum acyclic agreement forest for two rooted binary phylogenetic trees on the same set of taxa, which is simply performed by running a subroutine checking if the given maximum agreement forest contains any cycles. Unfortunately, at that time, this subroutine could only be used to recognize cycles of length two and, thus, failed in identifying cycles involving more than two components.

Consequently, we have worked out a modification of that algorithm, which, on the one hand, contains a subroutine that is able to identify cycles of arbitrary size and, on the other hand, can be used to calculate all maximum acyclic agreement forest for two rooted binary phylogenetic trees on the same set of taxa. In this work, however, we just give a high level description of this modified algorithm. A more detailed description can be found in the work of Scornavacca *et al.* [57], which additionally addresses its correctness and its runtime. Note that in Section 2.3 we give a brief overview of this paper.

Given two rooted binary phylogenetic trees T_1 and T_2 on the same set of taxa, our algorithm is initialized by $R = T_1$, $\mathcal{F} = \{T_2\}$ as well as a specific parameter $k \in \mathbb{N}$.

Next, by running a bounded search-type fashion during each recursion, two leaves a and c labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, sharing the same parent are chosen randomly from R. Then, depending on the location of the corresponding two leaves a' and c' in $\mathcal{F}$ labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, R and $\mathcal{F}$ are updated by either cutting or contracting certain subtrees. More precisely, we distinguish between three different cases.

(i) The two leaves a' and c' are both contained in the same component F_i of $\mathcal{F}$ and share the same parent. In this case we run three computational paths. One by recursively calling the algorithm with a' and c' as well as a and c being contracted to a new leaf labeled by $\{\mathcal{L}(a), \mathcal{L}(c)\}$ and two by recursively calling the algorithm with a' and c', respectively, being cut from F_i.

(ii) The two leaves a' and c' are both contained in two different components, say F_i and F_j, of $\mathcal{F}$. In this case we run two computational paths by recursively calling the algorithm with a' and c' being cut from F_i and F_j, respectively.

(iii) The two leaves a' and c' are both contained in the same component F_i in $\mathcal{F}$ but do not share the same parent. In this case, we run three computational paths. Two by recursively calling the algorithm with a' and c', respectively, being cut from F_i and a third one where one pendant subtree lying on the shortest path connecting a' and c' in F_i is cut.

Whenever a tree $\hat{F}$ in $\mathcal{F}$ only consisting of one specific taxon arises at the first time, $\hat{F}$ refers to exactly one leaf labeled by taxon $\hat{a}$ in R. At the beginning of each recursive call, each of those leaves labeled by such a taxon $\hat{a}$ is removed from R. Thus, after each recursive call either the number of leaves in R decreases at least by one (either by contracting two leaves or by deleting a certain leaf corresponding to a fully contracted tree $\hat{F}$ in $\mathcal{F}$), or the size of $\mathcal{F}$ increases, or both. Finally, a computational path stops if the size of $\mathcal{F}$ exceeds k or if R only consists of one single leaf. In the latter case, $\mathcal{F}$ forms an agreement forest of size k of T_1 and T_2, which is then checked by a subsequent subroutine whether it contains any cycles. If this is not the case, our algorithm has successfully computed an acyclic agreement forest of size k. Otherwise, $\mathcal{F}$ is rejected.

Moreover, our algorithm is conducted by incrementing parameter k by one starting with $k = s$ where s denotes a lower bound of the rSPR-distance. Hence, by acting this way, one can be sure that once an acyclic agreement forest $\mathcal{F}$ is reported by our algorithm there cannot exist an acyclic agreement forest of smaller size and, thus, $\mathcal{F}$ has to be of minimum size. If we are interested in the hybridization number, we can immediately stop each existent computational path right after having computed the first maximum acyclic agreement forest. Otherwise, if we are interested in all maximum acyclic agreement forest, we still have to continue the execution of all other computational paths, since these path can produce further results.

Final phase. If we are just interested in the rSPR-distance or the hybridization number, we simply have to sum up the sizes of all maximum agreement forests or of all maximum acyclic agreement forests, respectively, that is reported by the exhaustive search phase for each reduced tree pair.

Otherwise, in order to obtain a representative set of minimum hybridization networks, we first have to generate a certain network for each maximum acyclic agreement forest that has been reported for a reduced tree pair as described by the algorithm HybridPhylogeny given in the work of Baroni *et al.* [9]. Then, we have to undo the cluster reduction by replacing repeatedly each taxon acting as replacement character for a certain minimum common cluster $\mathcal{C}$ by all those networks that have been computed for a maximum acyclic agreement forest corresponding to $\mathcal{C}$. More precisely, assuming there are k reduced tree pairs and each of those pairs has n maximum acyclic agreement forests, then, in this case, this step that is undoing the cluster reduction produces exactly n^k networks. Lastly, we have to undo the subtree reduction, which is simply done by replacing each part of all so far computed networks by its corresponding subtree of the initial tree pair.

Parallelization. As the exhaustive search phase has the largest impact on the practical runtime, our implementation parallelizes this step on two different levels. First, each exhaustive search that has to be performed for the whole set of reduced tree pairs, resulting from the cluster reduction that is performed during the reduction phase at the beginning of the algorithm, is run in parallel. For the computation of the hybridization number or the set of representative minimum hybridization networks, the set of reduced tree pairs forms a set of subproblems that can be run independently from each other. This means, in particular, that the hybridization number can be computed by simply summing up each single hybridization number referring to each reduced tree pair [9]. More precisely, the execution is simply performed by placing each subproblem in a queue distributing each subproblem to individual cores subject to availability.

For the computation of the rSPR-distance the execution of the subproblems has to follow a specific cluster hierarchy $\mathcal{H}$. This means, in particular, that the execution has to respect $\mathcal{H}$ in a specific way. Moreover, the sum of each single rSPR-distance calculated for each reduced tree pair does only provide an upper bound of the rSPR-distance for the initial input trees. Thus, in order to receive the exact rSPR-distance for the initial tree pair, one has to apply some additional steps. However, we omit a detailed description here, and refer the interested reader to the work of Linz and Semple [41].

Independent from the three variants, an exhaustive search phase conducted for a reduced tree pair is parallelized in the following way. First, a thread is created for each possible value of parameter k, denoting the size of the agreement forest the thread is looking for. Second, each thread is placed in a priority queue distributing each thread to individual cores subject to availability. This is done in way such that all threads are executed in increasing order subject to its value k. Whenever a maximum agreement forest or a maximum acyclic agreement forest of size k' could be computed successfully, all other threads, belonging to the same reduced tree pair and are searching for agreement forests of larger sizes, are aborted immediately. Consequently, our implementation exploits a distributed system providing multiple cores even then when the initial tree pair does not share any common clusters.

Additional analysis. During the construction of a minimum hybridization network for two phylogenetic $\mathcal{X}$-trees T_1 and T_2, for each reticulation edge e we keep track to which of both trees e belongs. More precisely, this is done by labeling each edge belonging to T_1 with 1 and each edge belonging to T_2 with 2. Thus, for each minimum hybridization network that is reported by our method the user is able to highlight the embedding of the input trees, which is an important feature helping biologists to interpret hybridization events of so computed networks.

As the resulting set of representative minimum hybridization networks $\mathcal{N}$ can be quite large, we sort this set of networks in decreasing order by the sum of its *support values*, which are computed for each hybridization event. More precisely, for each network N_i in $\mathcal{N}$ and for each reticulation node r_z in N_i, we determine a specific set of taxa $L(r_z)$ containing each taxon that can be reached from r_z through a directed path not crossing any other reticulation nodes. Notice that each of those sets of a network N_i refers to exactly one component of its underlying maximum acyclic agreement forest. Now, based on these sets of taxa being computed for all reticulation nodes in $\mathcal{N}$, for each reticulation node r_z a certain support value $s(r_z)$ is assigned denoting the percentage of networks containing $L(r_z)$. More precisely, given a reticulation node r_z of a network N_i, we say that a network N_j contains $L(r_z)$, if it contains a reticulation node r_y with $L(r_z) = L(r_y)$. Moreover, $s(r_z)$ equals $1/t$ if the set of taxa $L(r_z)$ is contained in exactly t networks of $\mathcal{N}$. Now, given the support values for each reticulation node, we sort the set of representative minimum hybridization networks decreasingly in respect to the sum of its containing support values. Thus, the first networks automatically provide those hybridization events occurring most frequently in the entire set of computed networks. However, when interpreting those support values, one should keep in mind that based on a maximum acyclic agreement forest one usually can compute several networks. Notice that the upcoming network algorithms of this thesis do not have this undesired feature as those algorithms are able to calculate all of these networks.

2.2.3 Simulation study

In this section, we present the results of a simulation study, which we have undertaken to measure the performance of our algorithm. For this purpose, we have first integrated our algorithm as a plug-in into the freely available[2] Java software package Dendroscope 3 [35]. Then, we generated a synthetic dataset consisting of several pairs of rooted binary phylogenetic trees all sharing the same set of taxa. Each of those tree pairs is generated by ranging over all different combinations of three different parameters, namely the number of taxa n, the number of rooted rSPR-moves k (which has been used to obtain the one tree from the other), and the *tangling degree* d (as defined below). Our synthetic dataset contains tree-pairs with parameters $n \in \{20, 50, 100, 200\}$, $k \in \{5, 10, \ldots, 50\}$, and $d \in$

[2]www.dendroscope.org

$\{3, 5, 10, 15, 20\}$. This means, in particular, for all 200 combinations we generated 10 tree-pairs resulting in 2000 tree-pairs in total.

More specifically, we generated a tree-pair based on the three parameters n, k, and d as follows. First, a *rooted* binary phylogenetic tree whose leaves are bijectively labeled by the taxa set $\mathcal{X} = \{x_1, \ldots, x_n\}$ is generated by repeatedly choosing randomly two elements of a specific set of clusters $\mathcal{C}$. At the beginning, each taxon in $\mathcal{X}$ is a cluster in $\mathcal{C}$. After having chosen two clusters c_1 and c_2, these two clusters are removed from $\mathcal{C}$ and a new cluster $c_3 = c_1 \cup c_2$ is added. This is done until $\mathcal{C}$ contains only one element. In a second step, the other tree is obtained by performing exactly k rSPR-moves within this first tree with each of them respecting tangling degree d. More precisely, let R be a rooted rSPR-move pruning the subtree rooted at the target node of an edge $e_1 = (v_1, w_1)$ and re-grafting it to a specific edge $e_2 = (v_2, w_2)$. Then, we say that R respects tangling degree d, if the path leading from the lowest common ancestor of v_1 and v_2 consists of at least d edges.

Based on the way a tree-pair is computed as described above, we want to add two observations. First, the number of rooted rSPR-moves k, which is applied to obtain the second tree from the first tree, is just an upper bound of the true underlying rSPR-distance of both trees. This is the case because when performing a specific rSPR-move one can undo or redo previous rSPR-moves. Second, the tangling degree d has a direct influence on the number of minimum common cluster of both trees. This means, in particular, that tree-pairs, which have been constructed in respect of a small tangling degree, usually provide more minimum common clusters than those having been constructed in respect of larger tangling degrees. We decided to introduce this concept here, since, in respect to our approach, tree-pairs having a large number of minimum common clusters are in general of smaller computational complexity than those having only less. Our simulation study will indicate that this property does not hold for the software package HybridNET, which implies that its approach does not include a cluster reduction.

As mentioned above, we compared the implementation of our algorithm (more precisely, its variant computing the hybridization number) to the software package HybridNET, which, at this particular time, has been the best available software for calculating the exact hybridization number for two rooted binary phylogenetic trees on the same set of taxa. For this purpose, we integrated our approach as a plugin into Dendroscope 3 [35] and downloaded the freely available[3] software package HybridNET [17, 18].

Next, we will discuss the result of our simulation study, in which our method is denoted by Dendroscope 3 and the other one denoted by HybridNET. Both methods have been run on a AMD Phenom X4 955 Processor containing 4 GB RAM. In order to run the whole simulation study within an appropriate time, we decided to set the maximum time limit for each tree-pair to 20 minutes. This means, in particular, that each tree-pair whose hybridization number could not be computed within 20 minutes was aborted. Note that in Figure 2.2 the runtime of each aborted tree-pair has been counted with 20 minutes whereas in Figure 2.3 those aborted tree-pairs were not taken into account.

[3]`http://www.cs.cityu.edu.hk/~lwang/software/Hn`

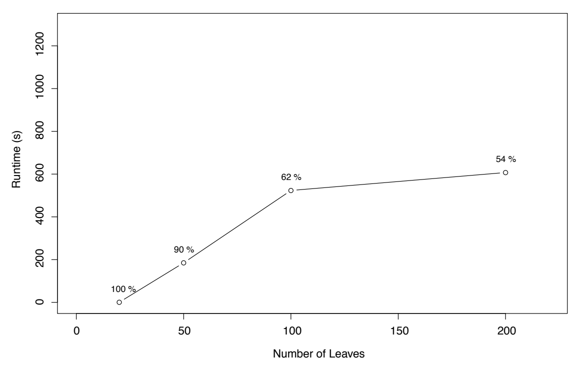
Figure 2.2 shows the average runtimes of both methods as a function of one of the three parameters n, k, and d. More precisely, each plot was generated by first aggregating all tree-pairs corresponding to a certain value of the parameter denoted at the x-axis. Second, the average runtime of all tree-pairs, contained in each of those aggregated subsets, were computed. Notice that, in this case, the runtime of each tree-pair that could not be computed within the time limit was set to 20 minutes. The percentage, which is denoted at each measurement in Figure 2.2, denotes the number of tree-pairs of each aggregated subset whose runtime was less than 20 minutes and, thus, could successfully be computed within the time limit.

The four plots given in Figure 2.2 (a)–(d) show that for both methods the average runtimes of each aggregated subset increases with the number of taxa and the number of rSPR-moves. Moreover, the average runtimes of each aggregated subset attained by our method is less than those attained by the software package HybridNET. Moreover, the percentages denoted at each measurement show that our method is able to compute more tree-pairs within the time limit of 20 minutes and, in addition, HybridNET is not able to compute any tree-pairs providing a rSPR-distance larger than or equal to 40, which is a strong indication that the underlying approach does not perform a cluster reduction.

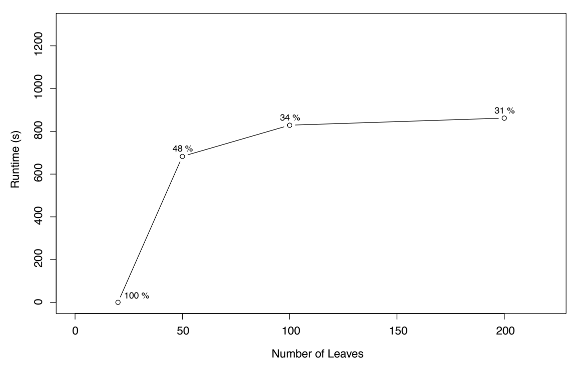
This conjecture is supported by the two plots given in Figure 2.2 (e),(d). These plots clearly show that, in contrast to HybridNET, our method performs better the more minimum common clusters a tree-pair contains. Notice that the plot given in Figure 2.2 (f) is quite erratic which can be explained by the fact that by increasing the number of minimum common clusters c the number of tree-pairs having exactly c minimum common clusters decreases. This is due to the fact because during the generation of tree-pairs rSPR-moves can take place within already existing minimum common clusters and, thus, may not necessarily increase the total number of minimum common clusters of both trees. As a direct consequence, tree pairs having a large number of minimum common clusters are less likely to occur in our synthetic dataset.

Figure 2.3 shows the average runtimes of both methods as a function of the computed hybridization number. More precisely, the plot was generated by first aggregating all tree-pairs having the same hybridization number and then by computing the average runtimes of all tree-pairs corresponding to the same aggregated subset. For each tree-pair that could not be computed within the time limit, its hybridization number is unknown and, consequently, for this analysis those tree-sets could not be taken into account. Figure 2.3 shows that the software package HybridNET was unable to compute hybridization numbers larger than 21 within the given time limit of 20 minutes, whereas our method could compute hybridization numbers up to 40.

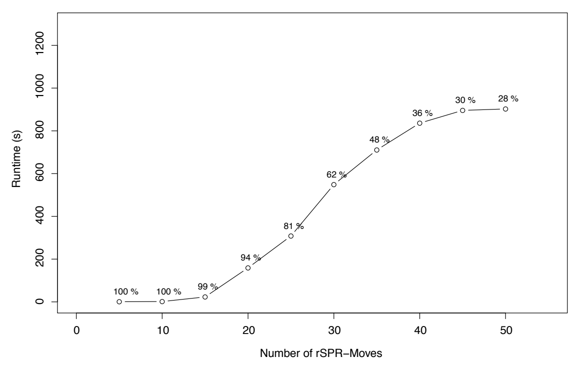
Finally, we computed both the hybridization number and the set of *representative* hybridization networks for a grass (*Poaceae*) dataset provided by the Grass Phylogeny Working Group [27]. Table 2.1 shows the result and a comparison of the runtimes attained by both methods. Note that we decided to add this information because this dataset has been frequently applied to evaluate other methods dealing with the computation of hybridization networks.



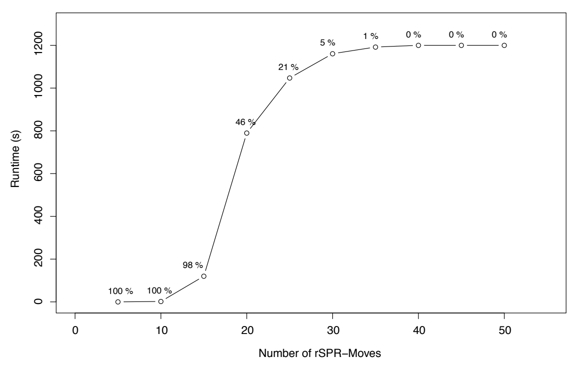
(a) Dendroscope 3



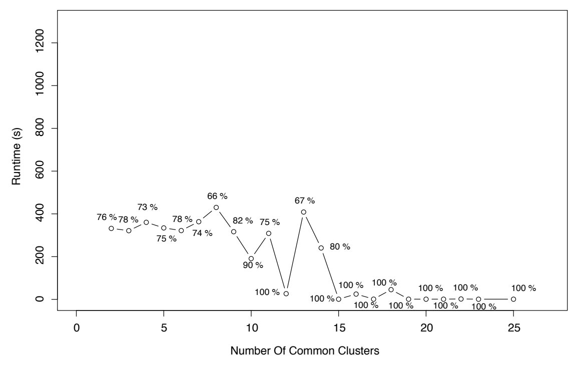
(b) HybridNet



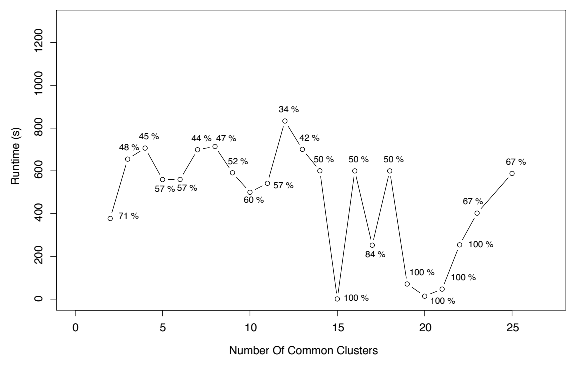
(c) Dendroscope 3



(d) HybridNet

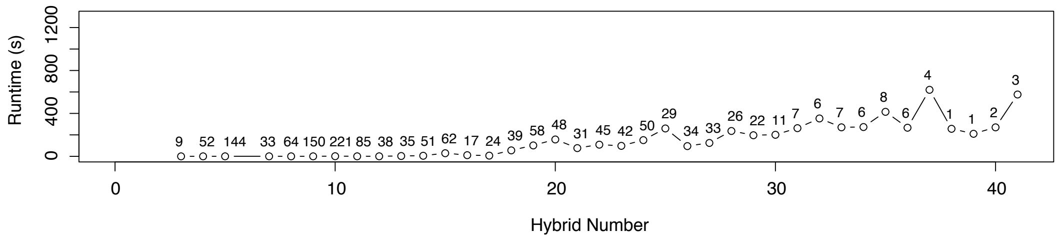


(e) Dendroscope 3

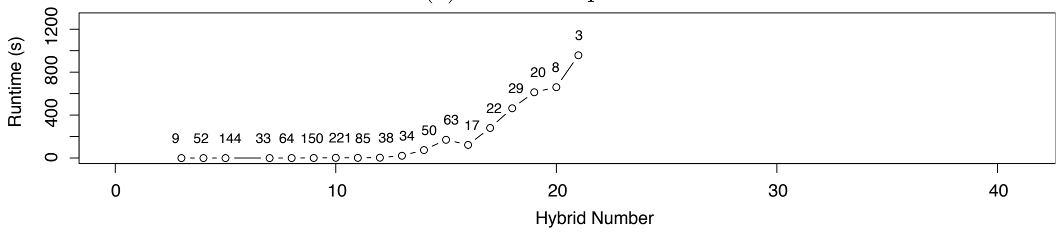


(f) HybridNet

Figure 2.2: Comparison of the runtimes computed for all tree pairs of our synthetic dataset attained by our method, which is implemented as plugin of DENDROSCOPE 3, (left) and the software package HYBRIDNET (right). Each plot shows the average runtime based on one of three specific properties of each tree-pair; namely, the number of leaves (a and b), its underlying number of rSPR-moves having been used for its construction (c and d), and the number of minimum common clusters (e and f). Each percentage refers to the amount of tree pairs that could be successfully computed within a time limit of 20 minutes.



(a) Dendroscope 3



(b) HybridNet

Figure 2.3: Average runtimes based on the hybridization number attained by our method, which is implemented as a plugin of DENDROSCOPE 3, (a) and the software package HYBRIDNET (b). The numbers for each measurement indicate the number of tree pairs that could be computed within a time limit of 20 minutes.

Based on the same synthetic dataset, we additionally studied the performance of our method for the computation of the exact rSPR-distance for two rooted binary phylogenetic trees on the same set of taxa. Therefor, we compared our method with the so far best available software package RSPR. To some unknown reason, the software package RSPR outperforms our approach for each single tree-pair within our synthetic dataset. Since both methods are based on the same algorithm presented in the paper of Whidden and Zeh [71], we assume that the speedup is more due to technical instead of algorithmic reasons. An possible explanation, for instance, could be the fact that the software package RSPR, which is written in C++, was implemented in a more efficient way than our Java based method.

2.2.4 Application to phylogeny of Aegilop/Triticum genera

The worldwide distributed *Triticea* tribe, which is part of the *Poaceae* family, consists of diploid and polyploid grasses. There exist several works dealing with inconsistent phylogenies calculated for certain sequence datasets [38, 54]. More specifically, these inconsistencies occur in analysis that are based on sequences belonging to chloroplast and genomic data [54]. Consequently, for the following application, we have used two datasets both corresponding to sequences derived from diploid species belonging to the genera *Triticum* and *Aegilops*. Whereas the first dataset refers to the *matK* sequence located on the chloroplast, the second one refers to the *PinA* sequence, which is part of the Triticeae chromosome 5. All sequences used in this application are obtained from the sequence database *GenBank*; the respective accession numbers are listed in Table 2.2.

Table 2.1: Output produced by Dendroscope and HybridNET applied to *two* phylogenetic trees belonging to a grass (*Poaceae*) dataset provided by the Grass Phylogeny Working Group. Each runtime given in this table is stated in seconds.

Genes	HNumber	#HNets	Computing HNumbers	
			Runtime Dendroscope	Runtime HybridNET
ndhF phyB	14	2268	< 1	17
ndhF rbcL	13	48	< 1	4
ndhF rpoC2	12	27	< 1	2
ndhF waxy	9	396	< 1	< 1
ndhF ITS	19	81	< 1	1102
phyB rbcL	4	4	< 1	< 1
phyB rpoC2	7	1	< 1	< 1
phyB waxy	3	6	< 1	< 1
phyB ITS	8	9	< 1	< 1
rbcL rpoC2	13	9	< 1	1
rbcL waxy	7	35	< 1	< 1
rbcL ITS	14	156	1	11
rpoC2 waxy	1	1	< 1	< 1
rpoC2 ITS	15	246	< 1	6
waxy ITS	8	18	< 1	< 1

Table 2.2: GenBank accession numbers of the sequences used for inferring each phylogenetic tree given in Figure 2.4.

	PinA	matK
Triticum urartu TU55	EU307589	FJ897889
Triticum monococcum DP57	EU307591	FJ897868
Aegilops tauschii DP16	FJ898213	FJ897861
Aegilops comosa DP13	FJ898210	FJ897858
Aegilops uniaristata DP56	FJ898218	FJ897867
Aegilops bicornis DP18	FJ898215	FJ897863
Aegilops longissima DP17	FJ898214	FJ897862
Aegilops sharonensis DP53	FJ898216	FJ897864
Aegilops speltoides SP6	FJ898222	FJ897884
Hordeum vulgare (Morex)	AY643843	EF115541

Next, based on the two datasets corresponding to both different sequences *matK* and *PinA*, two rooted phylogenetic trees T_1 and T_2 were computed by applying the software packages JModeltest [50] and PhyML [28]. The inferred discordant trees, depicted in Figure 2.4, give rise to several possible hybridization scenarios, which are illustrated by four minimum hybridization networks shown in Figure 2.5. Each of those networks

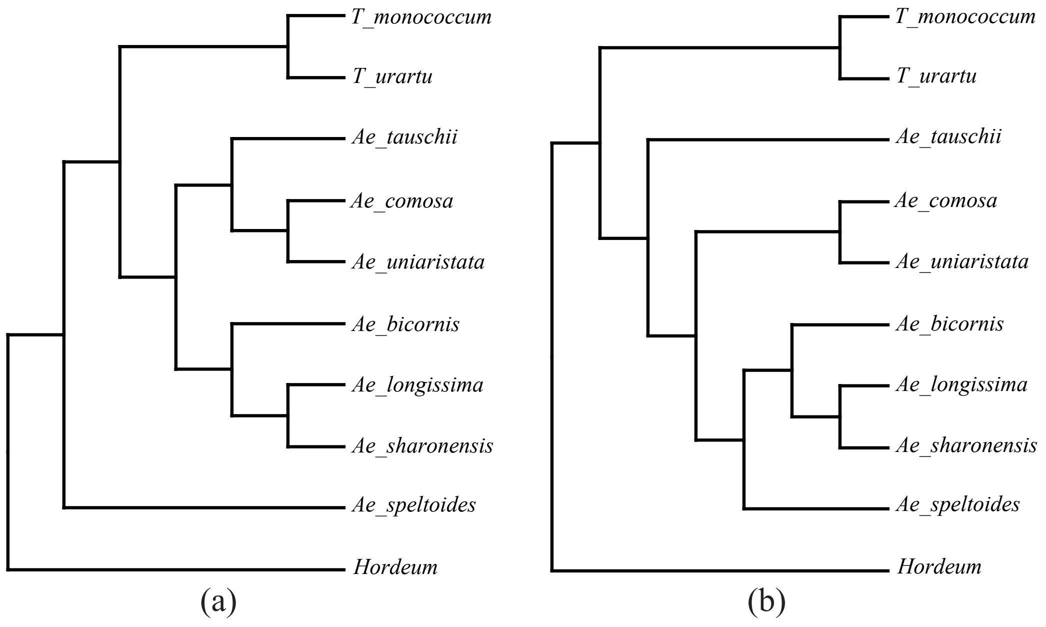


Figure 2.4: Two consensus trees derived from 100 bootstrap replicates for the *matK* **(a)** and *PinA* **(b)** datasets.

indicates a hybridization event between *Aegilops speltoides* and an ancestor of *Aegilops bicornis*, *Aegilops longissima*, as well as *Aegilops sharonensis*. Notice that this hypothesis is additionally supported by the work of Escobar *et al.* [24]. All other hybridization events, however, are rather unlikely and, thus, could be explained by other processes, e.g., *incomplete lineage sorting* or *duplication-loss*.

2.2.5 Discussion

Usually, evolutionary studies are based on incongruent gene trees. If the differences between those trees are significant enough and if hybridization may have an impact on the evolutionary history of the involved species, one can try to reconcile those trees into phylogenetic networks giving rise to hypothesis of putative hybridization events.

There exist a lot of methods dealing with reconciling incongruent gene phylogenies in terms of a network. A major problem, however, is the lack of software packages implementing those methods. Throughout this article, we have shown how to come up with a fast implementation of an algorithm that is able to compute a set of representative minimum hybridization networks for two rooted binary phylogenetic trees on the same set of taxa. Moreover, we have shown how to edit this set of resulting networks such that biologists can quickly figure out the most promising hybridization events.

Furthermore, by presenting a simulation study, we have indicated that, at this particular time, our implementation was faster than any so far existing software package. Moreover, by presenting a workflow generating hybridization networks for a certain type of grass

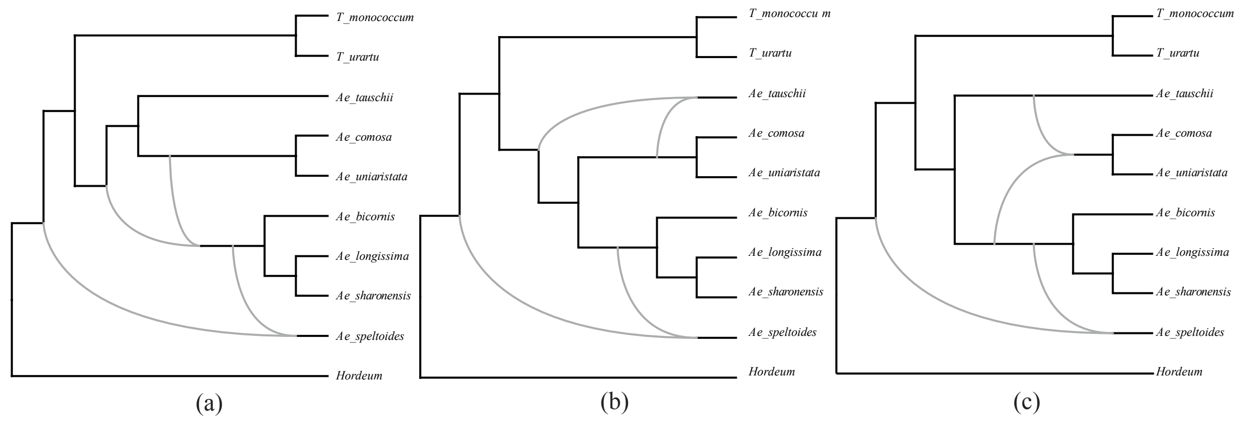


Figure 2.5: Three hybridization networks for the trees given in Figure 2.4 **(a–c)** computed by our algorithm integrated as plug-in into DENDROSCOPE 3.

genera, we have demonstrated how our method can be applied in practice to come up with putative hybridization scenarios.

2.3 The algorithm allMAAFs

In this section, we present the algorithm allMAAFs, which has been developed in respect to the approach introduced in Section 2.2. At this particular time, this algorithm was the first non-naive method calculating all maximum acyclic agreement forests for two binary rooted phylogenetic $\mathcal{X}$-trees. Before going into detail, however, we give an important remark emphasizing the relation of this algorithm to related works and introduce further definitions that are crucial for what follows.

2.3.1 Related work

The algorithm allMAAFs is based on two works of Whidden *et al.*; the first one [71] describing the original algorithm and the second one [70] introducing further improvements. In both works, however, a problematic description regarding the definition of an acyclic agreement forest is used. More precisely, the two works denote an acyclic agreement forest $\mathcal{F}$ for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 as being acyclic, if its underlying graph $AG(T_1, T_2, \mathcal{F})$ (as defined later) contains a cycle of length 2. This means, in particular, if each cycle within the graph $AG(T_1, T_2, \mathcal{F})$ is of length larger than or equal to 3, the agreement forest $\mathcal{F}$ is considered as being acyclic.

However, this makes no sense in a biological point of view, since hybridization networks can only be obtained from those agreement forests not containing any cycles of arbitrary length. Consequently, we fixed this issue, which means that our algorithm allMAAFs takes cycles of arbitrary length into account. Moreover, as shown by a formal proof, the algorithm allMAAFs is able to compute *all* maximum acyclic agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees, which is an important feature for the computation of hybridization networks as demonstrated in the upcoming part of this thesis.

2.3.2 Further definitions

In this section, we first introduce some further definitions, which are crucial for the description of the algorithm allMAAFs.

Agreement forests. In this section, given an agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees, we assume that both roots of the two trees are nodes of out-degree 1 that are labeled by the taxon ρ.

Forests. Let $\mathcal{F}$ be a forest for a rooted binary phylogenetic $\mathcal{X}$-tree T whose root is marked by the taxon ρ as described above. Then, by $\overline{\mathcal{F}}$, we refer to the forest that is obtained from $\mathcal{F}$ by deleting each element only consisting of an isolated node as well as the element containing the node labeled by taxon ρ, if it contains at most one edge.

Cherries. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let a and c be two of its leaves that are adjacent to the same node with label set $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively.

Then, we say the set consisting of the two taxa sets $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a *cherry* in R. Now, let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry in R and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, we say $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a *contradicting cherry* of R and $\mathcal{F}$, if there does not exist a component in $\mathcal{F}$ containing $\{\mathcal{L}(a), \mathcal{L}(c)\}$. Otherwise, if such a component exists in $\mathcal{F}$, the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is called a *common cherry* of R and $\mathcal{F}$. Moreover, if there exists a component in $\mathcal{F}$ containing two leaves labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, we write $a \sim_{\mathcal{F}} c$. Otherwise, if such a component does not exist, we write $a \not\sim_{\mathcal{F}} c$.

Pendant edges. Let F be a rooted binary phylogenetic $\mathcal{X}$-tree and let a and c be two of its leaves that are not adjacent to the same node with label set $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively. Moreover, let $\mathcal{P} = (a, v_1, v_2, \dots, v_n, c)$ be the path of nodes connecting a and c in F. Then, for each edge $e = (v, w)$ with $v \in \{v_1, v_2, \dots, v_n\}$ and $w \notin \mathcal{P}$ we say e is a *pendant edge of a and c.*

Reducing cherries. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry of R, in which a and c are the two leaves with label set $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively. Then, the operation reducing the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$, shortly denoted by $R[\{\mathcal{L}(a), \mathcal{L}(c)\} \to \mathcal{L}(a) \cup \mathcal{L}(c)]$, involves the following steps. First the common parent of a and c is labeled by $\mathcal{L}(a) \cup \mathcal{L}(c)$ and then both nodes a and c are deleted together with their respective labels and in-edges. Moreover, given a forest $\mathcal{F}$ on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R containing a tree F with cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$, by $\mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \to \mathcal{L}(a) \cup \mathcal{L}(c)]$ we refer to the operation reducing the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in F.

Expanding cherries. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Moreover, let F be a tree in $\mathcal{F}$ containing a leaf ℓ labeled by $\mathcal{L}(a) \cup \mathcal{L}(c)$. Then, the operation expanding ℓ, shortly denoted by $\mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \to \{\mathcal{L}(a), \mathcal{L}(c)\}]$, involves the following steps. First, two new leaves a and c are added to F by inserting the two edges (ℓ, a) and (ℓ, c). Next, the label from ℓ is removed and the two leaves a and c are labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively. Notice that, for applying such insertion steps, one has to keep track of the preceding reduction steps.

Cutting edges. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Moreover, let E' be an edge set in $\mathcal{F}$ so that each edge pair in E' is not adjacent. Then, the operation of cutting the edge set E' from $\mathcal{F}$, shortly denoted by $\mathcal{F} - E'$ involves the following two operations. First each edge e in E' is deleted and then each node of both in- and out-degree 1 is suppressed. Note that, after cutting E' from $\mathcal{F}$, the size of the resulting forest is $|\mathcal{F}| + |E'|$.

Labeled leaves. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree, then, $\ell(R)$ denotes the number of labeled leaves. Moreover, let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, $\ell(\overline{\mathcal{F}})$ refers to the total number of labeled leaves being contained in

$\overline{\mathcal{F}}$. Additionally, we write $\ell(R) \equiv \ell(\overline{\mathcal{F}})$, if $\ell(R)$ equals $\ell(\overline{\mathcal{F}})$ and if for each labeled leaf ℓ_R in R there exists a labeled leaf ℓ_F in $\mathcal{F}$ such that $\mathcal{L}(\ell_R) = \mathcal{L}(\ell_F)$.

2.3.3 Description of the algorithm

In the following, based on the definitions above, a description of the recursive algorithm allMAAFs is given. We will assume that at the beginning R is initialized by T_1, $\mathcal{F}$ by $\{T_2\}$, and M by $\emptyset$, where T_1 and T_2 are two rooted binary phylogenetic $\mathcal{X}$-trees and M denotes a map containing information for undoing each cherry reduction (as demonstrated later). Moreover, a parameter $k \in \mathbb{N}$ is given denoting the maximal size of an calculated agreement forest. Then, by calling the algorithm with those parameters, during each recursive call, $\overline{\mathcal{F}}$ is a forest for R such that $\ell(R) \equiv \ell(\overline{\mathcal{F}})$ holds and the output $\boldsymbol{\mathcal{F}}$ of allMAAFs contains all maximum acyclic agreement forests for T_1 and T_2 if and only if $k \geq h(T_1, T_2)$. Now, depending on the size of $\mathcal{F}$ and the cherry that is selected from R, the algorithm acts as follows.

Case 1a. If $\mathcal{F}$ contains more than k components, the computational path is aborted and an empty set is returned.

Case 1b. If R consists of an isolated node only, the forest $\mathcal{F}'$ is obtained from $\mathcal{F}$ by expanding each F in $\mathcal{F}$ as prescribed in M. More precisely, the agreement forest $\mathcal{F}$ is expanded by undoing each cherry reduction as follows. As long as a label $\mathcal{L}(a) \cup \mathcal{L}(c)$ of a leaf ℓ in $\mathcal{F}$ is contained in M, ℓ is expanded by applying the operation $\mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \rightarrow \{\mathcal{L}(a), \mathcal{L}(c)\}]$. Finally, after $\mathcal{F}$ has been expanded, the resulting agreement forest $\mathcal{F}'$ is returned if $AG(T_1, T_2, \mathcal{F}')$ is acyclic. Otherwise, the empty set is returned.

Case 1c. If there exists a leaf ℓ in R such that there exists an isolated node ℓ' in $\mathcal{F}$ with $\mathcal{L}(\ell') = \mathcal{L}(\ell)$, leaf ℓ is removed from R resulting in R'. Next, the algorithm branches into a new computational path by calling the algorithm with R', $\mathcal{F}$, k, and M.

Otherwise, if such a leaf does not exist, the algorithms continues with Case 2.

Case 2. First from R an arbitrary cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is chosen and then the algorithm continues with Case 3a, Case 3b, and Case 3c.

Case 3a. If $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of R and $\mathcal{F}$, the algorithm branches into a new computational path by calling the algorithm with $R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$, $\mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$, k, and M', which is obtained from M by adding the taxa set $\{\mathcal{L}(a), \mathcal{L}(c)\}$.

Case 3b. Let e_a and e_c be the in-edge of the two leaves a and c labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, in $\mathcal{F}$. Then, two new computational paths are initiated by calling the algorithm with R, $\mathcal{F} - \{e_a\}$, k, and M, as well as with R, $\mathcal{F} - \{e_c\}$, k, and M.

Case 3c. If there exists a component F in $\mathcal{F}$ such that $a \sim_{\mathcal{F}} c$ holds, a new computational path is initiated by calling the algorithm with R, $\mathcal{F} - \{e_B\}$, k, and M, where e_B denotes some pendant edge of a and c in $\mathcal{F}$.

We finish the description of the algorithm by noting that the algorithm ALLMAAFS always terminates, which is due to the fact that during each recursive call either the size of R decreases or the number of components in $\mathcal{F}$ increases. More precisely, the size of R is decreased either by deleting a leaf ℓ referring to an isolated node in $\mathcal{F}$ or by contracting a common cherry of R and $\mathcal{F}$. If R is not decreased, an edge in $\mathcal{F}$ is cut and, thus, its size increases by one. Consequently, as each computational path of the algorithm stops if R only consists of an isolated node or if k edges have been cut, a recursive call does always make progress towards one of both abort criteria.

2.3.4 Correctness of the algorithm allMAAFs

The algorithm ALLMAAFS calculates all maximum acyclic agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees, which is a consequence of the following theorem established in the work of Scornavacca *et al.* [57, Theorem 2].

Theorem 2 ([57]). *Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees. Calling*

$$\textsc{allMAAFs}(T_1, T_2, T_1, \{T_2\}, k, \emptyset)$$

returns all maximum acyclic agreement forests for T_1 and T_2 if and only if $k \geq h(t_1, T_2)$ with $k \in \mathbb{N}$.

In the proof of Theorem 2, the authors make use of the algorithm PROCESSCHERRIES (cf. Alg. 1) mimicking a computational path of the algorithm ALLMAAFS. More precisely, such a computational path can be described by a list $\bigwedge$ of cherry actions that are defined as follows. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R and $\ell(R) \equiv \ell(\overline{\mathcal{F}})$. Moreover, let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry of R. Then, we say $\wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e)$ is a cherry action of R and $\mathcal{F}$, if one of the two following properties is satisfied.

- Either, the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of R and $\mathcal{F}$ and $e = \emptyset$,
- or, the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of R and $\mathcal{F}$ and $e = e_a$ (or $e = e_c$), in which e_a (or e_c) is the in-edge of the leaf in $\mathcal{F}$ labeled by $\mathcal{L}(a)$ (or $\mathcal{L}(c)$), or $e = e_B$, in which e_B is a pendant edge of a and c in F.

Now, $\bigwedge = (\wedge_1, \wedge_2, \dots, \wedge_n)$ is called a cherry list for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, if during each iteration i of the algorithm PROCESSCHERRIES $\wedge_i$ is a cherry action for R_i and $\mathcal{F}_i$. Note that this is the case if and only if PROCESSCHERRIES$(T_1, \{T_2\}, \bigwedge)$ does not return the empty set.

Algorithm 1: ProcessCherries($R, \mathcal{F}, (\wedge_1, \wedge_2, \dots, \wedge_n)$)

```
$M \leftarrow \emptyset$;
for $i \leftarrow 1, \dots, n$ do
    $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_i) \leftarrow \wedge_i$;
    $e_a \leftarrow$ in-edge of the node $a$ labeled by $\mathcal{L}(a)$ in $\mathcal{F}$;
    $e_c \leftarrow$ in-edge of the node $c$ labeled by $\mathcal{L}(c)$ in $\mathcal{F}$;
    $E_B \leftarrow$ set of pendant edges of $a$ and $c$ in $\mathcal{F}$;
    if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of $R$ and $\mathcal{F}$ and $e_i = \emptyset$ then
        $R \leftarrow R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
        $\mathcal{F} \leftarrow \mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
        $M \leftarrow M \cup \{\mathcal{L}(a), \mathcal{L}(c)\}$;
    else if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common or contradicting cherry and $e_i = e_a$ or $e_i = e_c$ then
        $\mathcal{F} \leftarrow \mathcal{F} - \{e_i\}$;
    else if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry and $e_i \in E_B$ then
        $\mathcal{F} \leftarrow \mathcal{F} - \{e_i\}$;
    else
        return $(\emptyset)$;
    $R \leftarrow$ from $R$ remove each isolated node in $\mathcal{F}$;
$\mathcal{F} \leftarrow$ expand $\mathcal{F}$ as prescribed in $M$;
return $(\mathcal{F})$;
```

Now, based on the algorithm ProcessCherries mimicking a computational path via a cherry list $\bigwedge$, the proof showing the correctness of allMAAFs is established as follows.

Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees. Then, in a first step, the authors prove that, if ProcessCherries($T_1, T_2, (\wedge_1, \wedge_2, \dots, \wedge_n)$) returns a pre-stage $\mathcal{F}'$ of a maximum acyclic agreement forest $\mathcal{F}$ — a so-called super forest of $\mathcal{F}$ —, then, there still has to exists a cherry in R_n, where R_n denotes the tree R resulting from iteration n. This means, in particular, that a computational path of allMAAFs cannot get stuck while computing a maximum acyclic agreement forest, which implies that the algorithm allMAAFs does not return undesired forests of size smaller than $h(T_1, T_2)$ not fulfilling the properties of an agreement forest.

In a subsequent step, the authors show a stronger property of the algorithm, namely that each forest that is returned by calling the algorithm allMAAFs for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 is an agreement forest for both input trees. Roughly speaking, this is the case, since the operations that are conducted on T_2 while executing allMAAFs always produce a set of node-disjoint subtrees of T_1 and T_2.

Finally, based on those two sub-proofs, the main proof is established showing the correctness of Theorem 2, which is done as follows. Let $\mathcal{F}$ be a maximum acyclic agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2. Then, the algorithm allMAAFs applied to T_1 and T_2 contains a computational path containing for each $l \in \{1, \dots, |\mathcal{F}|\}$ a reduced set of trees $\mathcal{G}_l$ of size l such that by expanding this graph as prescribed by M_l a set of trees $\mathcal{G}'_l$ arises being a super forest of $\mathcal{F}$. This property, which is sufficient to establish Theorem 2, is proved by induction on l in which each case of the algorithm in respect of the so far computed super forest $\mathcal{G}'_l$ and the maximum acyclic agreement forest $\mathcal{F}$ is discussed.

2.3.5 Runtime of the algorithm allMAAFs

The following theorem indicates the theoretical worst-case runtime of the algorithm allMAAFs.

Theorem 3. *Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and $\mathcal{F}$ be a maximum agreement forest for T_1 and T_2 containing $k \in \mathbb{N}$ components. The theoretical worst-case runtime of the algorithm* allMAAFs *applied to T_1 and T_2 is $O(3^{|\mathcal{X}|+k})$.*

Proof. Let $\mathcal{F} = \{F_\rho, F_1, F_2, \dots, F_{k-1}\}$ be an agreement forest for T_1 and T_2 of size k. Then, in order to obtain $\mathcal{F}$ from T_2 there are $k-1$ edge cuttings necessary. Moreover, in order to reduce the size of the leaf set $\mathcal{X}$ of R to 1, to each component F_i in $\mathcal{F}$ we have to apply exactly $|\mathcal{L}(F_i)|-1$ cherry contractions. Consequently, at most $|\mathcal{X}|$ cherry contractions have to be performed in total. Thus, our algorithm has to perform at most $k + |\mathcal{X}|$ recursive calls for the computation of $\mathcal{F}$. Now, as one of these recursive calls can at least branch into three further recursive calls, $O(3^{|\mathcal{X}|+k})$ is an upper bound for the total number of recursive calls that are performed throughout the whole algorithm. Notice that each case that is conducted during a recursive can be performed by iterating a constant number over all nodes of R and $\mathcal{F}$ and, thus, its theoretical worst-case runtime can be neglected here. □

2.4 Speeding up allMAAFs

Phylogenetic studies are often joint works between biologists, mathematicians, and computer scientists; usually biologists produce certain data, mathematicians develop a model for analyzing this data and, based on such models, computer scientists create programs or tools helping to investigate the data. Regarding the investigation of reticulate evolution, the underlying problem is often highly combinatorial so that, due to its underlying complexity, a naive implementation of a model is typically insufficient for analyzing biologically relevant data.

In this work, we tackle this problem by focusing on the previously presented first non-naive algorithm ALLMAAFS [57] calculating *all* maximum acyclic agreement forests for two rooted binary phylogenetic trees, which can be considered, in a mathematical aspect, as an intermediate step in computing *all* possible hybridization networks. More precisely, this is the case, since those networks can be computed by gluing the components of a maximum acyclic agreement forest back together in a specific way, which, for example, has been already demonstrated by the algorithm HYBRIDPHYLOGENY [9]. The computation of a maximum acyclic agreement forest for two binary phylogenetic trees, however, is a well-known *NP-hard* problem [14] and, thus, in order to apply this step to large input trees and thereby making the algorithm accessible to a wider range of biological problems, we have worked out three modifications that significantly improve the practical runtime of the algorithm ALLMAAFS.

This section is organized as follows. In a first step, we give all further definitions for describing and discussing those three modified algorithms of the original algorithm ALLMAAFS, which are ALLMAAFS^1, ALLMAAFS^2, and ALLMAAFS^3. Then, after a detailed description of each modification, we establish the correctness of each of these modified algorithms, i.e., we show that each of those modifications still calculates all maximum acyclic agreement forests. Moreover, in order to indicate the speedup of those modifications, we present the results of a simulation study, which is based on our own synthetic dataset. Finally, we finish this section by discussion the theoretical as well as the practical runtime of all three modified algorithms.

2.4.1 Further definitions

In this section, we give all further formal definitions that we use throughout this section for describing and discussing the original algorithm ALLMAAFS as well as our three modified algorithms ALLMAAFS^1, ALLMAAFS^2, and ALLMAAFS^3.

Phylogenetic trees. Given a set $\mathcal{F}$ of rooted phylogenetic trees on different taxa sets, by $\mathcal{L}(\mathcal{F})$ we refer to the union of each taxa set of all trees in $\mathcal{F}$. Moreover, let T_1 and T_2 be two phylogenetic trees with $\mathcal{L}(T_2) \subseteq \mathcal{L}(T_1) = \mathcal{X}_1$. Then, we say T_2 is *contained* in T_1, shortly denoted by $T_2 \subseteq T_1$, if $T_1|_{\mathcal{L}(T_2)}$ equals T_2. Lastly, given a set of phylogenetic trees $\mathcal{F}$ as well as an edge set E' that is contained in $\mathcal{F}$ such that for each pair of edges $e_1, e_2 \in E'$, with $e_1 \neq e_2$, e_1 is not adjacent to e_2. Then, by $\mathcal{F} - E'$ we refer to the set $\mathcal{F}'$ of trees that is obtained from $\mathcal{F}$ by first deleting each edge in E' and, second, by suppressing each node of both in- and out-degree 1. Notice that by deleting an edge of a tree F in $\mathcal{F}$, this tree is separated into two parts F_a and F_b so that the resulting forest equals $\mathcal{F} \setminus \{F\} \cup \{F_a, F_b\}$. Consequently, after deleting E' from $\mathcal{F}$ the resulting set $\mathcal{F}'$ contains precisely $|\mathcal{F}| + |E'|$ trees.

Agreement forests. In this section, given an agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees, we assume that both roots of the two trees are nodes of out-degree 1 that are labeled by the taxon ρ (cf. Fig. 2.6(a)).

Forests. Let $\mathcal{F}$ be a forest for a rooted binary phylogenetic $\mathcal{X}$-tree T whose root is marked by the taxon ρ as described above. Then, by $\overline{\mathcal{F}}$ we refer to the forest that is obtained from $\mathcal{F}$ by deleting each element only consisting of an isolated node as well as the element containing the node labeled by taxon ρ if it contains at most one edge.

Modified ancestor descendant graphs. Given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 together with a forest $\mathcal{F}$ for T_1 (or T_2), the *modified ancestor descendant graph* $AG^*(T_1, T_2, \mathcal{F})$ is defined as follows. For each component in $\mathcal{F}$ the graph $AG^*(T_1, T_2, \mathcal{F})$ contains exactly one node. Moreover, an edge is directed from a node F_i to a node F_j if

(1) the root of $T_1(\mathcal{L}(F_i))$ is an ancestor of the root of $T_1(\mathcal{L}(F_j))$, or

(2) the root of $T_2(\mathcal{L}(F_i))$ is an ancestor of the root of $T_2(\mathcal{L}(F_j))$.

An illustration of such a modified ancestor descendant graph is given in Figure 2.6(c).

Note that the definition of an ancestor descendant graph given in Section 1.4.1 is different to the one presented above. For the definition of an ancestor descendant graph, $\mathcal{F}$ has to be an agreement forest for T_1 and T_2 and, thus, this definition is more strict than the one given for $AG^*(T_1, T_2, \mathcal{F})$ in which $\mathcal{F}$ just has to be a forest for T_1 (or T_2).

Cherries. For a rooted binary phylogenetic $\mathcal{X}$-tree R, we call two of its leaves a and c a *cherry*, denoted by $\{\mathcal{L}(a), \mathcal{L}(c)\}$, if both nodes have the same parent. Moreover, let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R, then, a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in R is called a *common cherry of R and $\mathcal{F}$*, if there exists a cherry $\{\mathcal{L}(a'), \mathcal{L}(c')\}$ in $\mathcal{F}$ with $\mathcal{L}(a') = \mathcal{L}(a)$ and $\mathcal{L}(c') = \mathcal{L}(c)$. Otherwise, the cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in R is called a *contradicting cherry of R and $\mathcal{F}$*. Moreover, in order to ease reading, if there exists a component in $\mathcal{F}$ containing two leaves labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, we write $a \sim_{\mathcal{F}} c$. Otherwise, if such a component does not exist, we write $a \not\sim_{\mathcal{F}} c$.

Now, let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}$ and let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry in $\mathcal{F}$, in which e denotes the in-edge of the parent p of a (and c). Then, if e exists, by $\mathcal{F} \div \{\mathcal{L}(a), \mathcal{L}(c)\}$ we simply refer to the forest $\mathcal{F} - \{e\}$ and, otherwise, if p has in-degree 0, to $\mathcal{F}$. Furthermore, let $\mathcal{P}$ be the path connecting a and c in $\mathcal{F}$. Then, the *set of pendant edges for* $\{\mathcal{L}(a), \mathcal{L}(c)\}$ contains each edge (v, w) with $v \in V(\mathcal{P}) \setminus \{a, c\}$ and $w \notin V(\mathcal{P})$, where $V(\mathcal{P})$ denotes the set of nodes in $\mathcal{P}$. Notice that in general there exist several, precisely $|V(\mathcal{P})| - 3$, edges satisfying the condition of such an edge.

Cherry Reductions. Let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}$ and let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry of a component F_i in $\mathcal{F}$. Then, a *cherry reduction*, according to a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in one of its components F_i, implies the following two operations.

(1) The parent of the two nodes a and c is labeled by $\mathcal{L}(a) \cup \mathcal{L}(c)$.

(2) Both nodes a and c together with their adjacent edges are deleted from F_i.

Throughout the algorithm, such a reduction step of a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in $\mathcal{F}$ is shortly denoted by $\mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$. Furthermore, given a leaf ℓ labeled by $\mathcal{L}(a) \cup \mathcal{L}(c)$, the reverse notation $\mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \rightarrow \{\mathcal{L}(a), \mathcal{L}(c)\}]$ describes the attachment of two new leaves labeled by $\mathcal{L}(a)$ and $\mathcal{L}(c)$, respectively, to ℓ together with removing the label $\mathcal{L}(a) \cup \mathcal{L}(c)$ from ℓ. Notice that, for applying such insertion steps, one has to keep track of the preceding reduction steps.

Equivalently, for a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in a rooted binary phylogenetic $\mathcal{X}$-tree R, we write $R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$ to denote a cherry reduction of $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in R.

2.4.2 The Algorithm allMAAFs

In this section, we present the algorithm allMAAFs that was first published in the work of Scornavacca *et al.* [57]. To increase its readability, we decided to split the original algorithm allMAAFs into five parts (cf. Alg. 2–6). Note that, apart from its graphical representation, our presentation of the algorithm allMAAFs together with its terminology adheres to the original algorithm.

2.4.3 Modifications to allMAAFs

In this section, we present three modifications of the algorithm allMAAFs that, on the one hand, do not improve its theoretical worst-case runtime but, one the other hand, significantly improve its practical runtime, which is indicated by a simulation study reported in Section 2.4.5.

The first modification improves the processing of a certain type of contradicting cherry whereas for the other two modifications only the processing step of a common cherry is of interest.

Given a contradicting cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ for R and $\mathcal{F}$ with $a \sim_{\mathcal{F}} c$, the original algorithm conducts three recursive calls. One by recursively calling the algorithm with $\mathcal{F} - \{e_a\}$ and $R|_{\overline{\mathcal{F}-\{e_a\}}}$, one by recursively calling the algorithm with $\mathcal{F} - \{e_c\}$ and $R|_{\overline{\mathcal{F}-\{e_c\}}}$ and one by recursively calling the algorithm with $\mathcal{F} - \{e_B\}$ and $R|_{\overline{\mathcal{F}-\{e_B\}}}$, in which e_a and e_c refers to the in-edge of leaf a and c, respectively, in $\mathcal{F}$ and e_B refers to an in-edge of a subtree lying on the path connecting a and c in $\mathcal{F}$. Regarding the latter recursive call,

Algorithm 2: ALLMAAFS(S, T, R, $\mathcal{F}$, k, M)

```
Data: Two rooted binary phylogenetic $\mathcal{X}$-trees $S$ and $T$, a rooted binary phylogenetic tree $R$ and a forest $\mathcal{F}$ such
      that $\mathcal{L}(R) = \mathcal{L}(\overline{\mathcal{F}})$ and $\mathcal{L}(T) = \mathcal{L}(\mathcal{F})$, an integer $k$, and a list $M$ that contains information of previously
      reduced cherries.
Result: A set $\boldsymbol{\mathcal{F}}$ of forests for $\mathcal{F}$ and an integer. In particular, if $\mathcal{F} = T$, $R = S$, $M = \emptyset$, and $k \geq h(S,T)$ is the
        input to ALLMAAFS, the output precisely consists of all maximum-acyclic-agreement forests for $S$ and $T$
        and their respective hybridization number.
if $k < 0$ then
    return $(\emptyset, k-1)$;
if $|\mathcal{L}(R)| = 0$ then
    $\mathcal{F}' \leftarrow$ CHERRYEXPANSION($\mathcal{F}$, $M$);
    if $AG(S,T,\mathcal{F}')$ is acyclic then
        return ($\mathcal{F}'$, $|\mathcal{F}'| - 1$);
    else
        return $(\emptyset, k-1)$;
else
    let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry of $R$;
    if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of $R$ and $\mathcal{F}$ then
        return (PROCESSCOMMONCHERRY($S$, $T$, $R$, $\mathcal{F}$, $k$, $M$, $\{\mathcal{L}(a), \mathcal{L}(c)\}$));
    if $k \neq (|\mathcal{F}| - 1)$ or $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of $R$ and $\mathcal{F}$ then
        return (PROCESSCONTRADICTINGCHERRY($S$, $T$, $R$, $\mathcal{F}$, $k$, $M$, $\{\mathcal{L}(a), \mathcal{L}(c)\}$));
```

Algorithm 3: CHERRYEXPANSION($\mathcal{F}, M$)

```
while $M$ is not empty do
    $M \leftarrow$ remove last element of M, say$\{\mathcal{L}(a), \mathcal{L}(c)\}$;
    $\mathcal{F} \leftarrow \mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \rightarrow \{\mathcal{L}(a), \mathcal{L}(c)\}]$;
return $\mathcal{F}$
```

Algorithm 4: CHERRYREDUCTION($R, \mathcal{F}, M, \{\mathcal{L}(a), \mathcal{L}(c)\}$)

```
$M' \leftarrow$ Add $\{\mathcal{L}(a), \mathcal{L}(c)\}$ as last element of $M$;
$R' \leftarrow R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
$\mathcal{F}' \leftarrow \mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
return $(R', \mathcal{F}', M')$
```

Algorithm 5: ProcessCommonCherry$(S,\, T,\, R,\, \mathcal{F},\, k,\, M,\, \{\mathcal{L}(a), \mathcal{L}(c)\})$

```
1  $(R', \mathcal{F}', M') \leftarrow$ cherryReduction$(R,\, \mathcal{F},\, M,\, \{\mathcal{L}(a), \mathcal{L}(c)\})$;
2  $(\boldsymbol{\mathcal{F}_r},\, k_r) \leftarrow$ allMAAFs$(S,\, T,\, R'|_{\mathcal{L}(\overline{\mathcal{F}'})},\, \mathcal{F}',\, k,\, M')$;
3  if $\boldsymbol{\mathcal{F}_r} \neq \emptyset$ then
4      $k \leftarrow \min(k, k_r)$;
5  if $(k = |\mathcal{F}| - 1)$ then
6      return $(\boldsymbol{\mathcal{F}_r},\, k)$
7  else
8      $(\boldsymbol{\mathcal{F}_a},\, k_a) \leftarrow$ allMAAFs$(S,\, T,\, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_a\}})},\, \mathcal{F} - \{e_a\},\, k - 1,\, M)$;
9      if $\boldsymbol{\mathcal{F}_a} \neq \emptyset$ then $k \leftarrow \min(k, k_a - 1)$ ;
10     if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_a}$ ;
11     $(\boldsymbol{\mathcal{F}_c},\, k_c) \leftarrow$ allMAAFs$(S,\, T,\, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_c\}})},\, \mathcal{F} - \{e_c\},\, k - 1,\, M)$;
12     if $\boldsymbol{\mathcal{F}_c} \neq \emptyset$ then $k \leftarrow \min(k, k_c - 1)$ ;
13     $\boldsymbol{\mathcal{F}} \leftarrow \emptyset$;
14     if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
15     if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
16     if $(k_r = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_r}$ ;
17     return $(\boldsymbol{\mathcal{F}},\, k)$;
```

Algorithm 6: ProcessContradictingCherry$(S,\, T,\, R,\, \mathcal{F},\, k,\, M,\, \{\mathcal{L}(a), \mathcal{L}(c)\})$

```
1  $(\boldsymbol{\mathcal{F}_a},\, k_a) \leftarrow$ allMAAFs$(S,\, T,\, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_a\}})},\, \mathcal{F} - \{e_a\},\, k - 1,\, M)$;
2  if $\boldsymbol{\mathcal{F}_a} \neq \emptyset$ then
3      $k \leftarrow \min(k, k_a - 1)$;
4  $\boldsymbol{\mathcal{F}_c},\, k_c \leftarrow$ allMAAFs$(S,\, T,\, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_c\}})},\, \mathcal{F} - \{e_c\},\, k - 1,\, M)$;
5  if $\boldsymbol{\mathcal{F}_c} \neq \emptyset$ then
6      $k \leftarrow \min(k, k_c - 1)$;
7  $\boldsymbol{\mathcal{F}} \leftarrow \emptyset$;
8  if $a \nsim_{\mathcal{F}} c$ then
9      if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
10     if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
11     return $(\boldsymbol{\mathcal{F}},\, k)$;
12 else
13     $(\boldsymbol{\mathcal{F}_B},\, k_B) \leftarrow$ allMAAFs$(S,\, T,\, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_B\}})},\, \mathcal{F} - \{e_B\},\, k - 1,\, M)$;
14     if $\boldsymbol{\mathcal{F}_B} \neq \emptyset$ then
15         $k \leftarrow \min(k, k_B - 1)$;
16     if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
17     if $(k_B - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_B}$ ;
18     if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
19     return $(\boldsymbol{\mathcal{F}},\, k)$;
```

in the upcoming part of this work we will show that, in order to compute all maximum acyclic agreement forests, instead of cutting just one in-edge e_B one can cut all of those in-edges all at once.

Moreover, given a common cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ for R and $\mathcal{F}$, the original algorithm ALLMAAFs always branches into three new computational paths; one path corresponding to the cherry reduction of $\{\mathcal{L}(a), \mathcal{L}(c)\}$ and two corresponding to the deletion of both in-edges of the two leaves a and c (cf. Alg. 5). To understand the sense of our two modifications, one has to take the necessity of these two additional edge deletions into account.

Therefor, we demonstrate a specific scenario that is outlined in Figure 2.6 showing two phylogenetic trees T_1 and T_2 on the same taxa set as well as a maximum acyclic agreement $\mathcal{F} = \{((a,b),\rho),(e,d),(g,h),c,f\}$ for those two trees. By running the algorithm ALLMAAFs for T_1 and T_2 without deleting in-edges of a common cherry, the given maximum acyclic agreement forest $\mathcal{F}$ is never computed. This is due to the fact that, once the component $F_i = ((g,h),f)$ occurs on any computational path, F_i will be part of the resulting agreement forest. Because of F_i and the component (e,d), such a resulting agreement forest is never acyclic and, thus, does not satisfy the conditions of an acyclic agreement forest. However, by cutting instead of contracting the common cherry $\{(g,h),f\}$, the resulting agreement forest turns into the maximum acyclic agreement forest $\mathcal{F}$. This example implies that sometimes the deletion of in-edges corresponding to taxa of a common cherry is necessary, which is, however, in practice not often the case, and, thus, the original algorithm ALLMAAFs usually produces a lot of additional unnecessary computational steps compared with two of our three modifications offering a different solution for such a scenario.

2.4.3.1 The Algorithm allMAAFs^1

Our first algorithm ALLMAAFs^1 is a modification of the original algorithm ALLMAAFs improving the processing of contradicting cherries. Let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a contradicting cherry of R and $\mathcal{F}$ such that $a \sim_{\mathcal{F}} c$ holds and let e_B be an edge that is defined as follows. Let $\mathcal{P}$ be the path connecting a and c in $\mathcal{F}$. Then, the edge set E_B contains each edge $e_B = (v,w)$ with $v \in V(\mathcal{P}) \setminus \{a,c\}$ and $w \notin V(\mathcal{P})$, where $V(\mathcal{P})$ denotes the set of nodes of $\mathcal{P}$. Note that in this case there exist precisely $|V(\mathcal{P}) - 3|$ edges satisfying the conditions of such an edge e_B. Now, if such a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ occurs, the original algorithm ALLMAAFs branches into a computational path by cutting exactly one of those edges in E_B. We will show, however, that in this case the whole set E_B can be cut from $\mathcal{F}$ all at once without having an impact on the computation of all maximum acyclic agreement forests for both input trees. In Algorithm 7, we give a pseudo code of ALLMAAFs^1. Note that, for the sake of clarity, we just present the modified part in respect of the original algorithm dealing with the processing of contradicting cherries. The remaining parts are unmodified and can be looked up in Section 2.4.2.

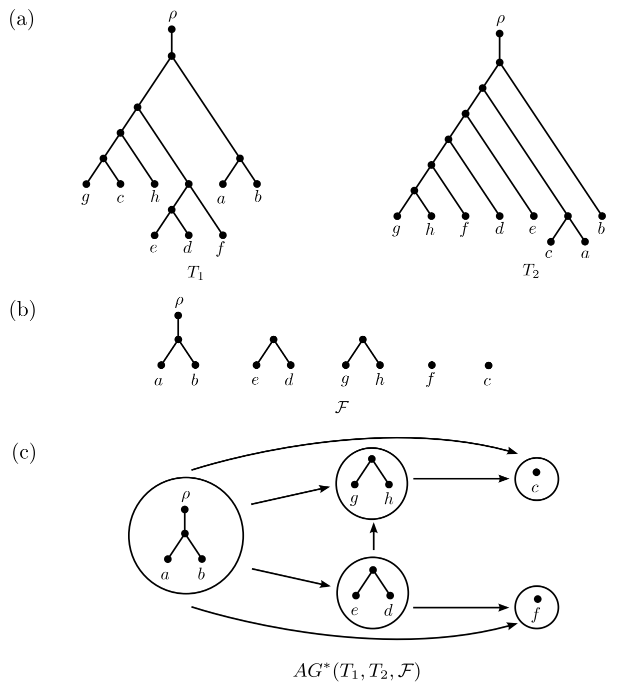


Figure 2.6: (a) Two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 with taxa set $\mathcal{X} = \{a, b, c, d, e, f, g, h, \rho\}$. (b) An acyclic agreement forest $\mathcal{F}$ for T_1 and T_2. (c) The directed graph $AG^*(T_1, T_2, \mathcal{F})$ not containing any directed cycles and, thus, $\mathcal{F}$ is acyclic.

Algorithm 7: ProcessContradictingCherry$^1(S, T, R, \mathcal{F}, k, M, \{\mathcal{L}(a), \mathcal{L}(c)\})$

```
$(\boldsymbol{\mathcal{F}_a}, k_a) \leftarrow$ allMAAFs$^1(S, T, R|_{\mathcal{L}(\overline{\mathcal{F}-\{e_a\}})}, \mathcal{F}-\{e_a\}, k-1, M)$;
if $\boldsymbol{\mathcal{F}_a} \neq \emptyset$ then
    $k \leftarrow \min(k, k_a - 1)$;
$\boldsymbol{\mathcal{F}_c}, k_c \leftarrow$ allMAAFs$^1(S, T, R|_{\mathcal{L}(\overline{\mathcal{F}-\{e_c\}})}, \mathcal{F}-\{e_c\}, k-1, M)$;
if $\boldsymbol{\mathcal{F}_c} \neq \emptyset$ then
    $k \leftarrow \min(k, k_c - 1)$;
$\boldsymbol{\mathcal{F}} \leftarrow \emptyset$;
if $a \nsim_{\mathcal{F}} c$ then
    if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
    if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
    return $(\boldsymbol{\mathcal{F}}, k)$;
else
    $(\boldsymbol{\mathcal{F}_B}, k_B) \leftarrow$ allMAAFs$^1(S, T, R|_{\mathcal{L}(\overline{\mathcal{F}-E_B})}, \mathcal{F}-E_B, k-|E_B|, M)$;
    if $\boldsymbol{\mathcal{F}_B} \neq \emptyset$ then
        $k \leftarrow \min(k, k_B - 1)$;
    if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
    if $(k_B - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_B}$ ;
    if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
    return $(\boldsymbol{\mathcal{F}}, k)$;
```

2.4.3.2 The Algorithm allMAAFs^2

The algorithm allMAAFs^2 is a simple modification of our first modification allMAAFs^1. It is based on the observation that those cutting steps for processing a common cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ often do not lead to a maximum acyclic agreement forest. In Section 2.4.4, we give a formal proof showing that those cutting steps only make sense if the component F of the agreement forest $\mathcal{F} \div \{\mathcal{L}(a), \mathcal{L}(c)\}$, containing the common cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$, is part of a cycle of the underlying directed graph $AG^*(T_1, T_2, \mathcal{F} \div \{\mathcal{L}(a), \mathcal{L}(c)\})$ in which T_1 and T_2 denote the corresponding input trees. Thus, our second modification allMAAFs^2 saves a lot of unnecessary recursive calls provoking a large speedup towards the original algorithm allMAAFs as well as our first modified algorithm allMAAFs^1 as indicated by our simulation study given in Section 2.4.5. In Algorithm 8 we give a pseudo code describing the algorithm allMAAFs^2. For the sake of clarity, we restrict the presentation to only those parts that are modified in respect of the first modified algorithm allMAAFs^1.

Algorithm 8: ProcessCommonCherry$^2(S, T, R, \mathcal{F}, k, M, \{\mathcal{L}(a), \mathcal{L}(c)\})$

```
$(R', \mathcal{F}', M') \leftarrow$ cherryReduction$(R, \mathcal{F}, M, \{\mathcal{L}(a), \mathcal{L}(c)\})$;
$(\boldsymbol{\mathcal{F}_r}, k_r) \leftarrow$ allMAAFs$^2(S, T, R'|_{\mathcal{L}(\overline{\mathcal{F}'})}, \mathcal{F}', k, M')$;
if $\boldsymbol{\mathcal{F}_r} \neq \emptyset$ then
    $k \leftarrow \min(k, k_r)$;
if $(k_r = |\mathcal{F}| - 1)$ then
    return $(\boldsymbol{\mathcal{F}_r}, k)$
else
    $\boldsymbol{\mathcal{F}} \leftarrow \emptyset$;
    $F \leftarrow$ component of $\mathcal{F} \div \{\mathcal{L}(a), \mathcal{L}(c)\}$ containing $a$ and $c$;
    if $F$ is part of a directed cycle in $AG^*(S, T, \mathcal{F} \div \{\mathcal{L}(a), \mathcal{L}(c)\})$ then
        $(\boldsymbol{\mathcal{F}_a}, k_a) \leftarrow$ allMAAFs$^2(S, T, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_a\}})}, \mathcal{F} - \{e_a\}, k - 1, M)$;
        if $\boldsymbol{\mathcal{F}_a} \neq \emptyset$ then
            $k \leftarrow \min(k, k_a - 1)$;
        $\boldsymbol{\mathcal{F}_c}, k_c \leftarrow$ allMAAFs$^2(S, T, R|_{\mathcal{L}(\overline{\mathcal{F} - \{e_c\}})}, \mathcal{F} - \{e_c\}, k - 1, M)$;
        if $\boldsymbol{\mathcal{F}_c} \neq \emptyset$ then
            $k \leftarrow \min(k, k_c - 1)$;
        if $(k_a - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}_a}$ ;
        if $(k_c - 1 = k)$ then $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_c}$ ;
    if $(k_r = k)$ then
        $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \boldsymbol{\mathcal{F}_r}$;
    return $(\boldsymbol{\mathcal{F}}, k)$;
```

2.4.3.3 The Algorithm allMAAFs^3

Our third algorithm allMAAFs^3 is again a modification of our first algorithm allMAAFs^1 and is based on a tool turning agreement forests into acyclic agreement forests. This tool, published by Whidden *et al.* [69], is based on the concept of an *expanded cycle graph* refining cyclic agreement forest. Due to such additional refinement steps, which are performed right after the computation of each maximum agreement forest, both cutting steps for processing a common cherry can be omitted. The simulation study in Section 2.4.5 indicates that this refinement step is efficient enough so that this modification in general outperforms the original algorithm allMAAFs, our first modification allMAAFs^1, as well as our second modification allMAAFs^2. In Algorithm 9 and 10, we present a pseudo code describing the algorithm allMAAFs^3. Again, for the sake of clarity, we restrict the presentation to only those parts that are modified in respect of the modified algorithm allMAAFs^1. Moreover, we omit a description of the subroutine conducting the refinement step, denoted by RefineForest, as it can be looked up in the work of Whidden *et al.* [69]. Notice that, due to this refinement step, an implementation of this modification is quite more expensive compared with our second modification allMAAFs^2.

Algorithm 9: allMAAFs$^3(S, T, R, \mathcal{F}, k, M)$

```
if $k < 0$ then
    return $(\emptyset, k-1)$;
if $|\mathcal{L}(R)| = 0$ then
    $\mathcal{F}' \leftarrow$ cherryExpansion($\mathcal{F}, M$);
    $\boldsymbol{\mathcal{F}}'' \leftarrow$ RefineForest($\mathcal{F}'$);
    $\boldsymbol{\mathcal{F}} \leftarrow \emptyset$;
    foreach $\mathcal{F}'' \in \boldsymbol{\mathcal{F}}''$ do
        if $|\mathcal{F}''| = k$ then
            $\boldsymbol{\mathcal{F}} \leftarrow \boldsymbol{\mathcal{F}} \cup \mathcal{F}''$;
        else if $|\mathcal{F}''| < k$ then
            $k \leftarrow |\mathcal{F}''|$;
            $\boldsymbol{\mathcal{F}} \leftarrow \{\mathcal{F}''\}$;
    return $(\boldsymbol{\mathcal{F}}, k-1)$;
else
    let $\{\mathcal{L}(a), \mathcal{L}(c)\}$ be a cherry of $R$;
    if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of $R$ and $\mathcal{F}$ then
        return (ProcessCommonCherry$^3(S, T, R, \mathcal{F}, k, M, \{\mathcal{L}(a), \mathcal{L}(c)\})$);
    if $k \neq (|\mathcal{F}| - 1)$ or $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of $R$ and $\mathcal{F}$ then
        return (ProcessContradictingCherry($S, T, R, \mathcal{F}, k, M, \{\mathcal{L}(a), \mathcal{L}(c)\}$));
```

Algorithm 10: ProcessCommonCherry$^3(S, T, R, \mathcal{F}, k, M, \{\mathcal{L}(a), \mathcal{L}(c)\})$

```
$(R', \mathcal{F}', M') \leftarrow$ cherryReduction($R, \mathcal{F}, M, \{\mathcal{L}(a), \mathcal{L}(c)\}$);
$(\boldsymbol{\mathcal{F}_r}, k_r) \leftarrow$ allMAAFs$^3(S, T, R'|_{\mathcal{L}(\overline{\mathcal{F}'})}, \mathcal{F}', k, M')$;
if $(k_r = k)$ then return $(\boldsymbol{\mathcal{F}_r}, k)$;
;
return $(\emptyset, k)$;
```

2.4.4 Proofs of Correctness

In this section, we give formal proofs showing the correctness of all three modified algorithms presented in Section 2.4.3. In a first step, however, we give some further definitions that are crucial for what follows.

2.4.4.1 The algorithm processCherries

In the following, we introduce the algorithm processCherries, which has already been utilized in the work of Scornavacca *et al.* [57]. This algorithm is a simplified version of the algorithm allMAAFs describing one of its computational paths by a list of *cherry actions*. Notice that this algorithm is a major tool that will help us to establish the correctness of the three modified algorithms.

Cherry actions. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, a cherry action $\wedge = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e)$ is a tuple containing a cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$ of two taxa sets corresponding to two leaves a and c in R as well as an edge e. We say that $\wedge$ is a *cherry action for R and $\mathcal{F}$*, if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a cherry of R and if, additionally, one of the following three conditions is satisfied.

(1) Either $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of R and $\mathcal{F}$ and $e \in \{\emptyset, e_a, e_c\}$,

(2) or $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of R and $\mathcal{F}$ with $a \not\sim_{\mathcal{F}} c$ and $e \in \{e_a, e_c\}$,

(3) or $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of R and $\mathcal{F}$ with $a \sim_{\mathcal{F}} c$ and $e \in \{e_a, e_B, e_c\}$.

In this context, e_a and e_c is an edge in $\mathcal{F}$ adjacent to both leaves a and c, respectively. Moreover, e_B is part of the set of pendant edges for $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in $\mathcal{F}$.

Cherry lists. Now, given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, we say that $\bigwedge$ is a *cherry list for T_1 and T_2*, if, while calling processCherries$(T_1, \{T_2\}, \bigwedge)$, in the i-th iteration $\wedge_i$ is a cherry action for R_i and $\mathcal{F}_i$. Note that, if $\bigwedge$ is not a cherry list for T_1 and T_2, calling processCherries$(T_1, \{T_2\}, \bigwedge)$ (cf. Alg. 11) returns the empty set.

Further details as well as an example of processCherries can be found in the work of Scornavacca *et al.* [57].

Next, we will introduce the algorithm processCherries^1. This algorithm is a simplified version of the algorithm allMAAFs^1 describing one of its computational paths by a list of *extended cherry actions*. Notice that this algorithm is a major tool that will help us to establish the correctness of our first modified algorithm allMAAFs^1.

Extended cherry action. Let R be a rooted binary phylogenetic $\mathcal{X}$-tree and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, an *extended cherry action* $\wedge = (\{\mathcal{L}(a), \mathcal{L}(c)\}, E')$ is a tuple containing a set $\{\mathcal{L}(a), \mathcal{L}(c)\}$ of two taxa sets corresponding to two leaves a and c in R as well as an edge set E'. We say that $\wedge$ is an *extended cherry*

Algorithm 11: PROCESSCHERRIES($R, \mathcal{F}, \bigwedge = (\wedge_1, \ldots, \wedge_n)$)

```
 1 $M \leftarrow \emptyset$;
 2 foreach $i \in 1, \ldots, n$ do
 3     if $\wedge_i$ is a cherry action for $R$ and $\mathcal{F}$ then
 4         $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_i) \leftarrow \wedge_i$;
 5         if $e_i = \emptyset$ then
 6             $M \leftarrow$ Add $\{\mathcal{L}(a), \mathcal{L}(c)\}$ as last element of $M$;
 7             $R \leftarrow R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
 8             $\mathcal{F} \leftarrow \mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
 9         else
10             $\mathcal{F} \leftarrow \mathcal{F} - \{e_i\}$;
11             $R \leftarrow R|_{\mathcal{L}(\overline{\mathcal{F}})}$;
12     else
13         return $\emptyset$;
14 while $M$ is not empty do
15     $M \leftarrow$ remove the last element, say $\{\mathcal{L}(a), \mathcal{L}(c)\}$, from $M$;
16     $\mathcal{F} \leftarrow \mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \rightarrow \{\mathcal{L}(a), \mathcal{L}(c)\}]$;
17 return $R, \mathcal{F}, M$
```

action for R and $\mathcal{F}$, if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of R and $\mathcal{F}$ with $a \sim_{\mathcal{F}} c$ and $E' = E_B$, where E_B refers to the set of pendant edges for $\{\mathcal{L}(a), \mathcal{L}(c)\}$ in $\mathcal{F}$.

Extended cherry list. Now, given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, we say that $\bigwedge$ is an *extended cherry list for T_1 and T_2*, if, while calling PROCESSCHERRIES[1]($T_1, \{T_2\}, \bigwedge$), in the i-th iteration $\wedge_i$ is either a cherry action or an extended cherry action for R_i and $\mathcal{F}_i$. Note that, if $\bigwedge$ is not an extended cherry list for T_1 and T_2, calling PROCESSCHERRIES[1]($T_1, \{T_2\}, \bigwedge$) (cf. Alg. 12) returns the empty set.

Lemma 4. *Let $\bigwedge'$ be an extended cherry list for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2. Moreover, let $\mathcal{F}$ be a forest for T_2 calculated by $\bigwedge'$. Then, there also exists a cherry list $\bigwedge$ for T_1 and T_2 calculating $\mathcal{F}$.*

Proof. To prove this lemma, it suffices to show how to replace each extended cherry action of $\bigwedge'$ so that the resulting cherry list $\bigwedge$ still computes $\mathcal{F}$. Let $\wedge'_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, E_B)$ be an extended cherry action with $E_B = \{e_1, e_2, \ldots, e_k\}$. Then, we can replace $\wedge'_i$ by the sequence of cherry actions

$$(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_1), (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_2), \ldots, (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_k).$$

Since by these cherry actions the same edges are cut from T_2 as by $\wedge'_i$, the topology of R'_{i+1} equals R_{i+k}, which directly implies that $\mathcal{F}$ is still computed. □

2.4.4.2 Correctness of allMAAFs[1]

In this section, we will discuss the correctness of our first modified algorithm ALLMAAFS[1] by establishing the following theorem.

Algorithm 12: processCherries$^1(R, \mathcal{F}, \bigwedge = (\wedge_1, \dots, \wedge_n))$

```
1  $M \leftarrow \emptyset$;
2  foreach $i \in 1, \dots, n$ do
3      if $\wedge_i$ is a cherry action for $R$ and $\mathcal{F}$ then
4          $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_i) \leftarrow \wedge_i$;
5          if $e_i = \emptyset$ then
6              $M \leftarrow$ Add $\{\mathcal{L}(a), \mathcal{L}(c)\}$ as last element of $M$;
7              $R \leftarrow R[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
8              $\mathcal{F} \leftarrow \mathcal{F}[\{\mathcal{L}(a), \mathcal{L}(c)\} \rightarrow \mathcal{L}(a) \cup \mathcal{L}(c)]$;
9          else
10             $\mathcal{F} \leftarrow \mathcal{F} - \{e_i\}$;
11             $R \leftarrow R|_{\mathcal{L}(\mathcal{F})}$;
12     else if $\wedge_i$ is an extended cherry action for $R$ and $\mathcal{F}$ then
13         $(\{\mathcal{L}(a), \mathcal{L}(c)\}, E_i) \leftarrow \wedge_i$;
14         $\mathcal{F} \leftarrow \mathcal{F} - E_i$;
15         $R \leftarrow R|_{\mathcal{L}(\mathcal{F})}$;
16     else
17         return $\emptyset$;
18 while $M$ is not empty do
19     $M \leftarrow$ remove the last element, say $\{\mathcal{L}(a), \mathcal{L}(c)\}$, from $M$;
20     $\mathcal{F} \leftarrow \mathcal{F}[\mathcal{L}(a) \cup \mathcal{L}(c) \rightarrow \{\mathcal{L}(a), \mathcal{L}(c)\}]$;
21 return $R, \mathcal{F}, M$
```

Theorem 5. *Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees and $k \in \mathbb{N}$. Calling*

$$\text{allMAAFs}^1(T_1, T_2, T_1, \{T_2\}, k, \emptyset)$$

returns all maximum acyclic agreement forests for T_1 and T_2 if and only if $k \geq h(T_1, T_2)$.

Proof. Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees and let $\bigwedge$ be a cherry list for T_1 and T_2 mimicking a computational path of allMAAFs calculating a maximum acyclic agreement forest $\mathcal{F}$ for T_1 and T_2. Then, in the following, we say a cherry action $\wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_i)$ in $\bigwedge$ is a *special cherry action*, if $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a contradicting cherry of R_i and $\mathcal{F}_i$ and if e_i is contained in the set of pendant edges for $\{\mathcal{L}(a), \mathcal{L}(c)\}$. Note that, whereas such special cherry actions may occur in a computational path of allMAAFs, this is not the case for a computational path of allMAAFs^1. In the following, however, we will show that the algorithm allMAAFs calculates a maximum acyclic agreement forest $\mathcal{F}$ for T_1 and T_2 if and only if $\mathcal{F}$ is calculated by allMAAFs^1.

Lemma 6. *Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees and let $\mathcal{F}$ be a maximum acyclic agreement forest for T_1 and T_2 of size k. Then, the agreement forest $\mathcal{F}$ is calculated by calling* allMAAFs$(T_1, T_2, T_1, T_2, k, \emptyset)$ *if and only if $\mathcal{F}$ is calculated by calling* allMAAFs$^1(T_1, T_2, T_1, \{T_2\}, k, \emptyset)$.

Proof. '$\Longrightarrow$': From Lemma 4 we can directly deduce that if there exists a computational path of allMAAFs^1 calculating $\mathcal{F}$, then, there also exists a computational path in allMAAFs calculating $\mathcal{F}$.

'$\Longleftarrow$': Let $\wedge_j = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_j = \emptyset)$ be a cherry action of $\bigwedge$ contracting both leaves a and c. Then, we say a preceding special cherry action $\wedge_i = (\{\mathcal{L}(x), \mathcal{L}(y)\}, e_i)$ $(i < j)$ refers to $\wedge_k$ if the following condition is satisfied. Let a_i and c_i be the lowest common ancestor of $\mathcal{L}(a)$ and $\mathcal{L}(c)$ in R_i, respectively. Then, $e_i \neq \emptyset$ has to be a pendant edge lying on the path connecting both nodes a_i and c_i. In such a case, we say the special cherry action is either of *Type A* or *Type B* (see definition below). Otherwise, if a special cherry action does not refer to another cherry action, we say this cherry action is of *Type C*.

Now, based on the edge $e_i = (v, w)$, being part of the special action $\wedge_i$ referring to the cherry action $\wedge_j$, we further distinguish whether $\wedge_i$ is either of *Type A* or of *Type B*. Therefor, let $T(w)$ be the subtree rooted at w and let $\mathcal{L}(T(w))$ be the set of taxa being contained in $T(w)$. Now, we say $\wedge_i$ is of *Type A*, if there exists a forest $\mathcal{F}_k$ and a tree R_k with $i < k < j$ so that both of the following two conditions are satisfied.

(i) Each taxon in $\mathcal{L}(T(w))$ is part of a taxa set of an isolated node. Notice that, as a direct consequence, there exists a leaf w' in R_k with label $\mathcal{L}(w)$.

(ii) The sibling s' of w' in R_k is a leaf. Notice that, as a direct consequence, $\{\mathcal{L}(s'), \mathcal{L}(w')\}$ is a cherry of R_k.

Otherwise, if there does not exist such a forest $\mathcal{F}_k$ and such a tree R_k satisfying these two conditions, we say $\wedge_i$ is of *Type B*. An illustration of these two types is given in Figure 2.7.

Notice that, due to the so chosen definition, each special cherry action in $\bigwedge$ has to be either of *Type A*, of *Type B*, or of *Type C*. Moreover, as at the end of a cherry list each component of the so calculated forest is an isolated node, for each special cherry action of *Type C* there has to exist a cherry action $\wedge_k$ satisfying both conditions listed for a cherry action of *Type B*. Next, based on these observations, we will show in three steps how to turn $\bigwedge$ into another cherry list for T_1 and T_2 not containing any special cherry actions, but still calculating $\mathcal{F}$, which can be briefly summarized as follows.

In a first step, we will show how to modify $\bigwedge$ by replacing each special cherry action of *Type A* and of *Type C* through a non-special cherry action so that the result is still a cherry list for T_1 and T_2 calculating $\mathcal{F}$. Next, during a second and a third step, we will show how to replace each set of special cherry actions of *Type B* all referring to the same cherry action by a single extended cherry action so that the result is an extended cherry list for T_1 and T_2 still calculating $\mathcal{F}$.

Step 1. Let $\wedge_i = (\{\mathcal{L}(x), \mathcal{L}(y)\}, e_i = (v, w))$ be a special cherry action of *Type A* or of *Type C*, and let $\wedge_k$ with $i < k < j$ be the first cherry action of $\bigwedge$ in which in $\mathcal{F}_k$ each taxon in $\mathcal{L}(T(w))$ is contained in a taxa set of an isolated node so that in R_k the sibling s' of the leaf labeled by $\mathcal{L}(w)$ is also leaf. Then, let $\bigwedge'$ be a cherry list that is obtained from $\bigwedge$ by first removing $\wedge_i = (\{\mathcal{L}(x), \mathcal{L}(y)\}, e_i)$ and then by inserting the cherry action $(\{\mathcal{L}(w'), \mathcal{L}(s')\}, e_{w'})$ right after $\wedge_k$, so that $\bigwedge'$ equals

$$(\wedge'_1, \dots, \wedge'_n) = (\wedge_1, \dots, \wedge_{i-1}, \wedge_{i+1}, \dots, \wedge_k, \wedge'_k, \dots, \wedge'_n),$$

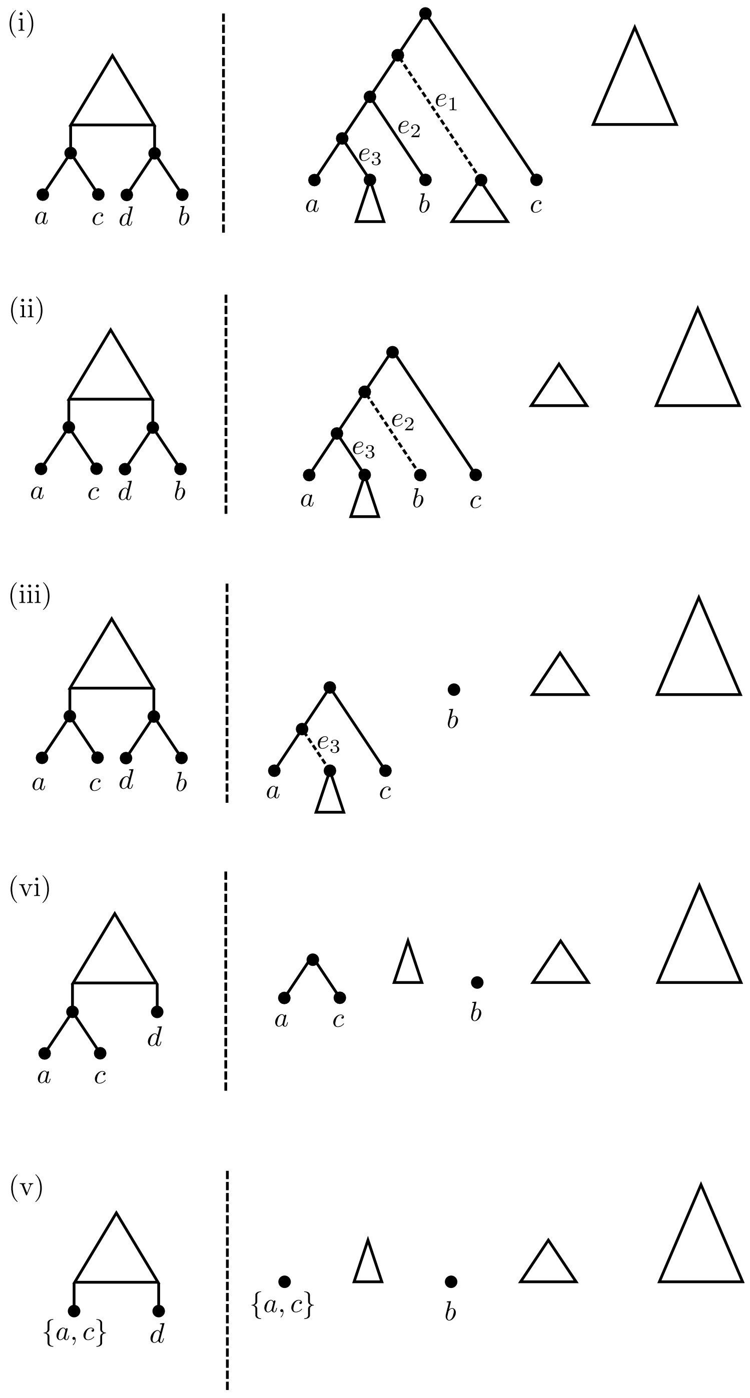


Figure 2.7: An illustration of the cherry list $((\{a,c\},e_1),(\{a,c\}),e_2),(\{a,c\}),e_3),(\{a,c\}),\emptyset))$ in which $(\{a,c\},e_1)$ is a special cherry action of *Type B* and $(\{a,c\},e_2)$ is a special cherry action of *Type A*. Note that the expanded cherry list $((\{b,d\},e_2),(\{a,c\},\{e_1,e_3\}),(\{a,c\}),\emptyset))$ applied to (i) yields the same scenario as depicted in (v) without making use of any special cherry actions.

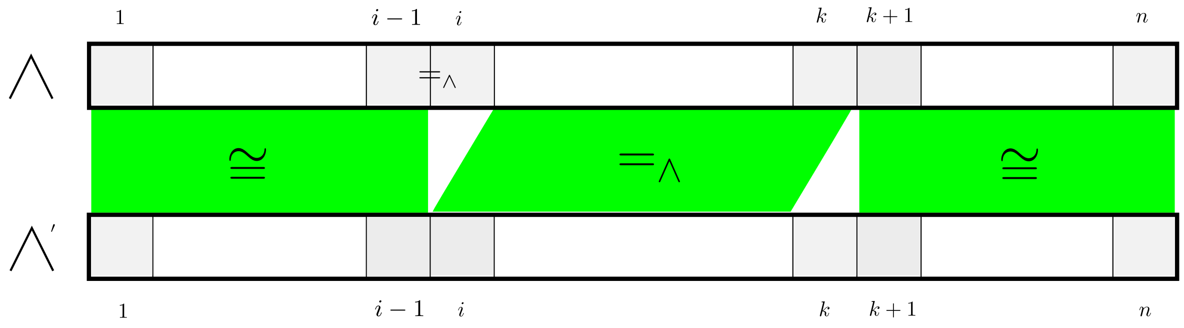


Figure 2.8: An illustration of the scenario as described in Step 1. The figure shows a comparison between two sequences of R_i-trees; one corresponding to the original cherry list and one corresponding to the modified cherry list, in which a special cherry action of *Type A* is shifted from position i to $k+1$.

where $\wedge'_k = (\{\mathcal{L}(w'), \mathcal{L}(s')\}, e_{w'})$ with $e_{w'}$ being the in-edge of w'.

Now, for the following, remember that R_l (or R'_l) refers to the input tree occurring during iteration l while processing the cherry list $\bigwedge$ (or $\bigwedge'$). Moreover, we write $R_i =_\wedge R'_j$ if both trees contain the same set of cherries. Then, based on the position of the cherry action $\wedge_l$ in $\bigwedge$, we can make the following five observations (cf. Fig. 2.8).

- If $l < i$, then R'_l equals R_l: This is the case because R_1 equals R'_1 and through $\wedge_l$ and $\wedge'_l$ the same tree operation is performed on R_l and R'_l, respectively.
- If $l = i$, then $R'_{l-1} =_\wedge R_l$: This is the case because R_{i-1} equals R'_{i-1}, in R_l the node w cannot be part of a cherry (due to the definition of a special cherry action of *Type A*) and, thus, through $\wedge_l$ ($=\wedge_i$) the set of cherries in R_{i-1} remains unchanged.
- If $i < l < k+1$, then $R'_{l-1} =_\wedge R_l$: This is the case because $R_i =_\wedge R'_{i-1}$ and again in R_l node w cannot be part of a cherry. Thus, the two cherry actions $\wedge_l$ and $\wedge'_{l-1}$ produce the same set of cherries in R_l and R'_{l-1}.
- If $l = k+1$, then R'_{k+1} equals R_{k+1}: This is the case because $R_k =_\wedge R'_{k-1}$ and through $\wedge'_k$ first node w is cut and then removed from R'_{k-1}. Consequently, through the cherry actions $\wedge_q$ and $\wedge'_q$, with $1 \leq q < k+1$, the same tree operations are applied to R_1 and R'_1 and, thus, R_{k+1} equals R'_{k+1}.
- If $l > k+1$, then R'_l equals R_l: This is the case because R_{k+1} equals R'_{k+1} and through $\wedge_l$ and $\wedge'_l$ the same tree operation is performed on R_l and R'_l, respectively.

Due to these observations and since through $\bigwedge'$ still the same edges are cut from T_2 as through the original cherry list $\bigwedge$, $\bigwedge'$ is a cherry list for T_1 and T_2 still calculating $\mathcal{F}$. Now, by consecutively replacing all special cherry actions of *Type A* and *Type C* we can turn $\bigwedge$ into the cherry list $\bigwedge^{(1)}$ for T_1 and T_2 only containing special cherry actions of *Type B* and still computing $\mathcal{F}$. Next, we will show how to remove each of those remaining special cherry actions.

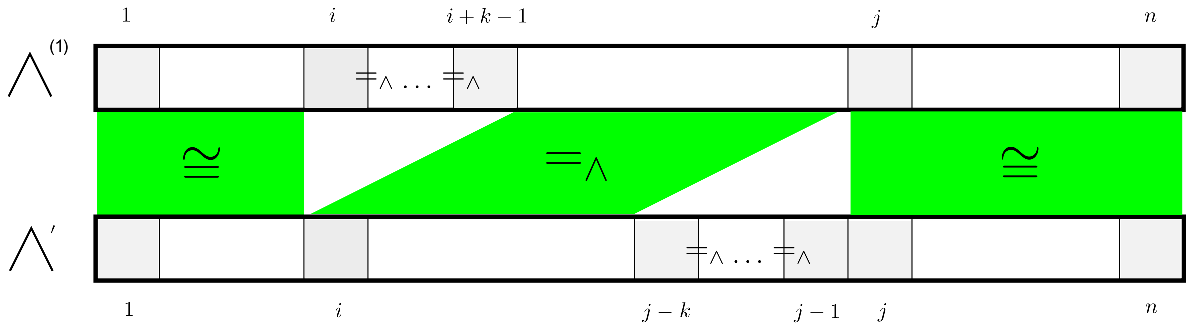


Figure 2.9: An illustration of the scenario as described in Step 2. The figure shows a comparison of the two sequences of R_i-trees; one corresponding to the original cherry list and one corresponding to the modified cherry list, in which a sequence of k special cherry actions of *Type B*, beginning at position i, is shifted to position $j-k$.

Step 2. Let $\bigwedge^* \subset \bigwedge$ with $|\bigwedge^*| = k$ be a set of special cherry actions of *Type B* all referring to a cherry action $\wedge_j = (\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset)$, and let $i = \min_{i'}\{\wedge_{i'} : \wedge_{i'} \in \bigwedge^*\}$ with $\wedge_i = (\{\mathcal{L}(x), \mathcal{L}(z)\}, e_i)$. Moreover, let $\bigwedge'$ be the cherry list that is obtained from $\bigwedge^{(1)}$ as follows. First the cherry of each cherry action in $\bigwedge^*$ is replaced by $\{\mathcal{L}(a), \mathcal{L}(c)\}$ and then all those cherry actions in $\bigwedge^*$ are rearranged such that they are placed in sequential order directly right before position j.

This means, in particular, that $\bigwedge'$ contains a sequence of special cherry actions

$$\begin{aligned} &\vdots \\ \wedge_{j-k} &= (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_{j-k} = (v_{j-k}, w_{j-k})), \\ \wedge_{j-k+1} &= (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_{j-k+1} = (v_{j-k+1}, w_{j-k+1})), \\ &\vdots \\ \wedge_{j-1} &= (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_{j-1} = (v_{j-1}, w_{j-1})), \\ \wedge_{j} &= (\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset), \\ &\vdots \end{aligned}$$

in which each edge of $\bigwedge^*$ is contained in the set of pendant edges for $\{\mathcal{L}(a), \mathcal{L}(c)\}$. Now, just for convenience, in the following we assume, without loss of generality, that all those cherry actions in $\bigwedge^*$ occur in sequential order beginning at position i. Moreover, for the following, let $\mathcal{W}$ be the set of target nodes of each edge in $\bigwedge^*$ and let R_l (or R'_l) be the input tree of iteration l while processing the cherry list $\bigwedge^{(1)}$ (or $\bigwedge'$). Additionally, again we write $R_i =_\wedge R'_j$ if both trees contain the same set of cherries. Then, based on the position of a cherry action $\wedge_l$ in $\bigwedge$, we can make the following five observations (cf. Fig. 2.9).

- If $l < i$, then R'_l equals R_l: This is the case because R_1 equals R'_1 and through $\wedge_l$ and $\wedge'_l$ the same tree operation is performed on R_l and R'_l, respectively.

- If $i-1 < l < i+k$, then $R_l =_\wedge R'_{i-1}$: This is the case because R_{i-1} equals R'_{i-1} and, since in R_l each w_i in $\mathcal{W}$ cannot be part of a cherry, through $\wedge_l$ the set of cherries in R_l remains unchanged.
- If $i+k-1 < l < j$, then $R'_{l-k} =_\wedge R_l$: This is the case because $R_{i+k-1} =_\wedge R'_{i-1}$ and again in R_l each w_i in $\mathcal{W}$ cannot be part of a cherry. Thus, the two cherry actions $\wedge_l$ and $\wedge'_{l-k}$ produce the same set of cherries in R_l and R'_{l-k}.
- If $l = j$, then R'_l equals R_l: This is the case because $R_{j-1} =_\wedge R'_{j-k-1}$ and through each cherry action in $\bigwedge^*$ each subtree $T(w_i)$ is cut (and removed from R'_{j-k-1}, if w_i is a leaf). Consequently, through the cherry actions $\wedge_q$ and $\wedge'_q$, with $1 \leq q < j$, the same tree operations are applied to R_1 and R'_1 and, thus, R_j equals R'_j.
- If $l > j$, then R'_l equals R_l: This is the case because R_j equals R'_j and through $\wedge_l$ and $\wedge'_l$ the same tree operation is performed on R_l and R'_l, respectively.

Now, by consecutively rearranging all special cherry actions of *Type B* as described above, we can turn $\bigwedge^{(1)}$ into the cherry list $\bigwedge^{(2)}$ for T_1 and T_2 in which all special cherry actions referring to the same cherry action are located next to each other. Moreover, as a direct consequence of these observations and since through $\bigwedge^{(2)}$ still the same edges are cut from T_2 as through $\bigwedge^{(1)}$, $\bigwedge^{(2)}$ is still a cherry list for T_1 and T_2 calculating $\mathcal{F}$.

Step 3. Let $\bigwedge^{(2)}$ be the list that is obtained from applying Step 1 and Step 2 as described above. Then, we can further modify the cherry list $\bigwedge^{(2)}$ to $\bigwedge^{(3)}$ by replacing each sequence

$$(\wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_i), \wedge_{i+1} = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_{i+1}), \dots, \wedge_{i+k-1} = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_{i+k-1}))$$

of special cherry actions (i.e., cherry actions of *Type B*) through a single extended cherry action $\wedge_B = (\{\mathcal{L}(a), \mathcal{L}(c)\}, E_B)$, where E_B denotes the pendant edge set for $\{\mathcal{L}(a), \mathcal{L}(c)\}$. As a consequence, since $\wedge_B$ just summarizes all tree operations conducted by the replaced sequence of special cherry actions, by $\bigwedge^{(3)}$ the same edges are cut from T_2 as it is the case for the cherry list $\bigwedge^{(2)}$. As a direct consequence, the maximum acyclic agreement forest $\mathcal{F}$ is still calculated by the extended cherry list $\bigwedge^{(3)}$.

In summary, as described by those three steps, we can consecutively replace all special cherry actions of $\bigwedge$ so that the resulting list $\bigwedge^{(3)}$ satisfies all of the following three conditions.

- $\bigwedge^{(3)}$ is an extended cherry list for T_1 and T_2.
- $\bigwedge^{(3)}$ does not contain any special cherry actions.
- $\bigwedge^{(3)}$ calculates the maximum acyclic agreement forest $\mathcal{F}$.

This directly implies that for each computational path in ALLMAAFS calculating a maximum acyclic agreement forest $\mathcal{F}$ for both input trees there also exists a computational path in ALLMAAFS^1 calculating $\mathcal{F}$. $\square$

Now, from Lemma 6 we can directly deduce the correctness of Theorem 5. $\square$

2.4.4.3 Correctness of allMAAFs^2

In this section, we will discuss the correctness of our second modified algorithm ALLMAAFS^2 by proving the following theorem.

Theorem 7. *Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees and $k \in \mathbb{N}$. Calling*

$$\text{ALLMAAFS}^2(T_1, T_2, T_1, \{T_2\}, k, \emptyset)$$

returns all maximum acyclic agreement forests for T_1 and T_2 if and only if $k \geq h(T_1, T_2)$.

Proof. The proof of Theorem 7 is established by Lemma 8 and 9. Here, we first argument that by contracting common cherries instead of cutting one of its edges, results in an agreement forests containing less components. It might be the case, however, that these agreement forests are not acyclic. In such a case, however, these common cherries have to be part of a cycle in the underlying modified ancestor descendant graph. Consequently, depending on the topology of this graph, one can decide whether it is worth to cut edges of a common cherry or not.

Lemma 8. *Let $\bigwedge = (\wedge_1, \wedge_2, \dots, \wedge_n)$ be a cherry list for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 with $\wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_c) \in \bigwedge$ so that $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of R_i and $\mathcal{F}_i$ and e_c the in-edge of leaf c. Moreover, let $\mathcal{F}$ be the agreement forest for T_1 and T_2 that is calculated by calling PROCESSCHERRIES$(R_1, \mathcal{F}_1, \bigwedge)$ such that $|\mathcal{F}| = k$. Then, there exists an agreement forest $\hat{\mathcal{F}}$ with $|\hat{\mathcal{F}}| = k - 1$ that can be computed by calling PROCESSCHERRIES$(R_i, \mathcal{F}_i, (\wedge'_i, \dots, \wedge'_n))$ in which $\wedge'_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset)$.*

Proof. Let F_a and F_c be the two components in $\mathcal{F}$ that have been derived from expanding both components containing a and c, respectively, and let $\mathcal{X}_a$ be the taxa set of the subtree rooted at a, i.e., $\mathcal{X}_a = \mathcal{L}(F_a(a))$. Then, by attaching F_c back to F_a we can reduce the size of the agreement forest $\mathcal{F}$ by one. Here, depending on whether $\text{LCA}_{F_a}(\mathcal{X}_a)$ is the root of F_a or not, this reattachment step can be done in two different ways.

- If $\text{LCA}_{F_a}(\mathcal{X}_a)$ is not the root of F_a and, thus, has an in-going edge $e = (u, w)$, first e is split into two adjacent edges (u, v) and (v, w) and then F_c is reattached to v by inserting a new edge (v, r_c), where r_c denotes the root of F_c.
- Otherwise, if $\text{LCA}_{F_a}(\mathcal{X}_a)$ is the root of F_a, first a new node v is created and then v is connected to the two roots of F_a and F_c.

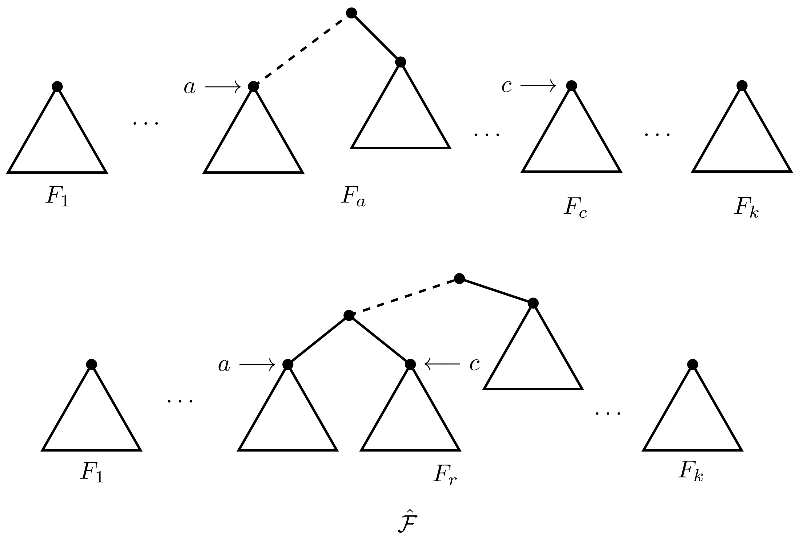


Figure 2.10: The two forests $\mathcal{F}$ and $\hat{\mathcal{F}}$ as defined in Lemma 8. Note that, since F_c is attached to F_a, the size of $\hat{\mathcal{F}}$ is $|\mathcal{F}| - 1$.

In the following, we will denote the component that is obtained from attaching F_c back to F_a by F_r and the resulting set of components by $\hat{\mathcal{F}}$. In Figure 2.10, we give an illustration of the two forests $\mathcal{F}$ and $\hat{\mathcal{F}}$.

Notice that we can calculate the agreement forest $\hat{\mathcal{F}}$ by calling the algorithm processCherries with a specific cherry list $\bigwedge'$ that can be derived from the original cherry list $\bigwedge = (\wedge_1, \wedge_2, \dots, \wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_c), \dots, \wedge_n)$ as follows.

(1) Replace $\wedge_i$ by $(\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset)$.

(2) In each subsequent cherry action $\wedge_{i+1}, \dots, \wedge_n$, replace $\mathcal{L}(a)$ by $\mathcal{L}(a) \cup \mathcal{L}(c)$.

Let R_{i+1} and $\mathcal{F}_{i+1}$ be computed by applying $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_c)$ to R_i and $\mathcal{F}_i$ and let R'_{i+1} and $\mathcal{F}'_{i+1}$ be computed by applying $(\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset)$ to R_i and $\mathcal{F}_i$. Since $\{\mathcal{L}(a), \mathcal{L}(c)\}$ is a common cherry of R_i and $\mathcal{F}_i$, the only difference between the two topologies, disregarding node labels, is that $\mathcal{F}_{i+1}$ contains an additional component consisting of an isolated node labeled by $\mathcal{L}(c)$. Notice, however, that, since this component is fully contracted, this component cannot have an impact on the subsequent cherry actions $\wedge_{i+1}, \dots, \wedge_n$. Consequently, the cherry list $\bigwedge'$ calculates $\hat{\mathcal{F}}$.

Up to now, we have shown that there exists a set of components $\hat{\mathcal{F}}$ of size $|\mathcal{F}| - 1$, which can be computed by calling the algorithm processCherries with a slightly modified cherry list $\bigwedge'$. Now, to establish Lemma 8, we still have to show that $\hat{\mathcal{F}}$ is an agreement forest for both input trees T_1 and T_2. First, note that each expanded forest to which allMAAFs applies the acyclic check (cf. Alg. 2, line 5) has to satisfy each condition of an agreement forest [57, Lemma 3]. As $\bigwedge'$ is imitating a computational path corresponding to the original algorithm allMAAFs calculating $\hat{F}$, this directly implies that $\hat{F}$ is an agreement forest for T_1 and T_2. $\square$

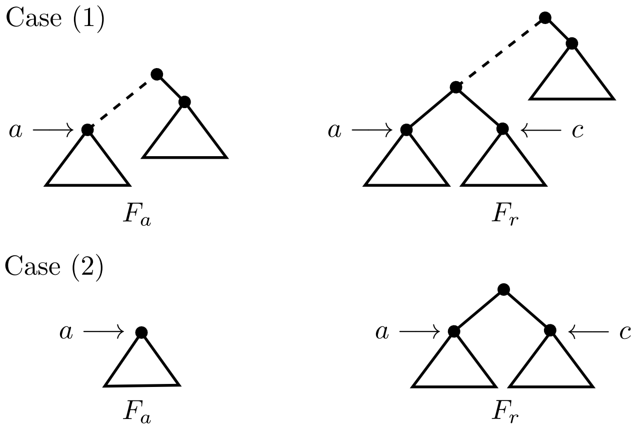


Figure 2.11: An outline of Case (1) and Case (2) corresponding to the proof of Lemma 9.

Now, before entering Lemma 9, we first introduce a further notation. Let $\wedge_i$ be a cherry action of a cherry list $(\wedge_1,\ldots,\wedge_n)$, with $1 \leq i \leq n$, for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, and let F be a component of its corresponding forest $\mathcal{F}_i$. Then, in order to ease reading, we write $\mathcal{F}_i^{\text{ex}}$ to denote the forest that is obtained from calling processCherries$(T_1,\{T_2\},(\wedge_1,\ldots,\wedge_i))$. Moreover, by F^{ex} we refer to the corresponding expanded component in $\mathcal{F}_i^{\text{ex}}$.

Lemma 9. *Let $\mathcal{F}$, $\mathcal{F}_i$, $\hat{\mathcal{F}}$, and $\{\mathcal{L}(a),\mathcal{L}(c)\}$ be as defined in Lemma 8. Moreover, let $\tilde{F}$ be the component in $\mathcal{F}_i \div \{\mathcal{L}(a),\mathcal{L}(c)\}$ containing both nodes a and c. Then, the agreement forest $\hat{\mathcal{F}}$ is acyclic, if $\mathcal{F}$ is acyclic and the expanded component $\tilde{F}^{ex}$ is not part of a directed cycle in $AG^*(T_1,T_2,\tilde{\mathcal{F}}_i^{ex})$, where $\tilde{\mathcal{F}}_i$ equals $\mathcal{F}_i \div \{\mathcal{L}(a),\mathcal{L}(c)\}$.*

Proof. Let $F_a,F_c \in \mathcal{F}$, F_r be the reattached component in $\hat{\mathcal{F}}$ with taxa set $\mathcal{L}(F_a)\cup\mathcal{L}(F_c)$, and $\tilde{F}^{\text{ex}} \in \tilde{\mathcal{F}}_i^{\text{ex}}$. To prove Lemma 9, we first have to take all potentially different topologies of the two graphs $AG^*(T_1,T_2,\mathcal{F})$ and $AG^*(T_1,T_2,\hat{\mathcal{F}})$ into account. Since $\mathcal{F}$ is acyclic, as defined in Lemma 9, each directed cycle that could arise in $AG^*(T_1,T_2,\hat{\mathcal{F}})$ has to contain F_r. Thus, it suffices to consider only those edges of $AG^*(T_1,T_2,\hat{\mathcal{F}})$ that are adjacent to F_r.

Now, let $\mathcal{X}_a$ be the taxa set derived from expanding the contracted node a in F_a, i.e., $\mathcal{X}_a = \mathcal{L}(F_a^{\text{ex}}(a))$. Depending on whether $\text{LCA}_{F_a}(\mathcal{X}_a)$ is the root of F_a or not, we can distinguish between the following two cases (cf. Fig. 2.11).

Case (1). If $\text{LCA}_{F_a}(\mathcal{X}_a)$ is not the root of F_a, then the lowest common ancestor of $\mathcal{L}(F_a)$ and $\mathcal{L}(F_r)$ in both input trees is preserved and, thus, the topology of $AG^*(T_1,T_2,\hat{\mathcal{F}})$ can be derived from $AG^*(T_1,T_2,\mathcal{F})$ by removing node F_c together with all its incident edges and by renaming node F_a to F_r. In this case, we can observe the following properties for each edge in $AG^*(T_1,T_2,\hat{\mathcal{F}})$.

(1.1) If (F_x,F_y) is an edge in $AG^*(T_1,T_2,\hat{\mathcal{F}})$ with $F_x,F_y \neq F_r$, then (F_x,F_y) is contained in $AG^*(T_1,T_2,\mathcal{F})$, too.

(1.2) If (F_x, F_r) is an edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$, then both edges (F_x, F_a) and (F_x, F_c) also exist in $AG^*(T_1, T_2, \mathcal{F})$.

(1.3) If (F_r, F_x) is an edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$, then the edge (F_a, F_x) exists in $AG^*(T_1, T_2, \mathcal{F})$.

This implies that, in this special case, the edge set of $AG^*(T_1, T_2, \hat{\mathcal{F}})$ is a subset of the edge set of $AG^*(T_1, T_2, \mathcal{F})$ (disregarding node labels) and, consequently, since $AG^*(T_1, T_2, \mathcal{F})$ does not contain any directed cycles, this also holds for the modified ancestor descendant graph of $\hat{\mathcal{F}}$. Consequently, $AG^*(T_1, T_2, \hat{\mathcal{F}})$ is acyclic.

Case (2). If $\mathrm{LCA}_{F_a}(\mathcal{X}_a)$ is the root of F_a, due to the following scenario, there can arise a different topology of $AG^*(T_1, T_2, \hat{\mathcal{F}})$ as the one discussed in Case (1). Let F_b be a component whose lowest common ancestor is contained in a subtree lying on the path leading from $\mathrm{LCA}_T(\mathcal{X}_a)$ to $\mathrm{LCA}_T(\mathcal{L}(F_r))$ with $T \in \{T_1, T_2\}$. Because of this component F_b, as $T(\mathcal{L}(F_r))$ is ancestor of $T(\mathcal{L}(F_b))$ in T, there arises a new edge (F_r, F_b) in $AG^*(T_1, T_2, \hat{\mathcal{F}})$. Thus, we can observe the following properties for each edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$.

(2.1) If (F_x, F_y) is an edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$ with $F_x, F_y \neq F_r$, then (F_x, F_y) has to be contained in $AG^*(T_1, T_2, \mathcal{F})$, too.

(2.2) If (F_x, F_r) is an edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$, then (F_x, F_a) and (F_x, F_c) does exist in $AG^*(T_1, T_2, \mathcal{F})$.

(2.3) If (F_r, F_x) is an edge in $AG^*(T_1, T_2, \hat{\mathcal{F}})$, there are three possible scenarios regarding the edge set of $AG^*(T_1, T_2, \mathcal{F})$.

(2.3.1) Either (F_a, F_x) or (F_c, F_x) exists in $AG^*(T_1, T_2, \mathcal{F})$.

(2.3.2) (F_x, F_a) is an edge and (F_x, F_c) is *not* an edge in $AG^*(T_1, T_2, \mathcal{F})$.

(2.3.3) (F_x, F_c) is an edge and (F_x, F_a) is *not* an edge in $AG^*(T_1, T_2, \mathcal{F})$.

An illustration of Case (2.3.1), (2.3.2), and (2.3.3) is given in Figure 2.12.

Regarding Case (2.1), (2.2) and (2.3.1), again the edge set of $AG^*(T_1, T_2, \hat{\mathcal{F}})$ is a subset of the edge set of $AG^*(T_1, T_2, \mathcal{F})$ and, thus, as already discussed in Case (1), since $AG^*(T_1, T_2, \mathcal{F})$ is acyclic this has to hold for $AG^*(T_1, T_2, \hat{\mathcal{F}})$, too. In the following, we will focus on the scenario given in Case (2.3.2). Notice that establishing the correctness of Lemma 9 in respect to Case (2.3.3) can be done in the same way.

Regarding Case (2.3.2), we have that there is a component F_x in $\mathcal{F}$ such that (F_x, F_a) is an edge and (F_x, F_c) is *not* an edge in $AG^*(T_1, T_2, \mathcal{F})$. Consequently, after reattaching F_c back to F_a there exists an edge (F_r, F_x) in $AG^*(T_1, T_2, \hat{\mathcal{F}})$. Moreover, we assume that there exists a set $\mathcal{F}_c = \{F_{\gamma_1}, \ldots, F_{\gamma_n}\}$ of components with $\mathcal{F}_c \subset \mathcal{F}$ such that $AG^*(T_1, T_2, \mathcal{F})$ contains a directed path

$$\mathcal{P} = (F_x, F_{\gamma_1}, F_{\gamma_2}, \ldots, F_{\gamma_n}, F_c)$$

with $n \geq 0$. Otherwise, if there does not exist such a directed path $\mathcal{P}$ connecting F_x and F_c in $AG^*(T_1, T_2, \mathcal{F})$, we can directly deduce that $AG^*(T_1, T_2, \hat{\mathcal{F}})$ is acyclic.

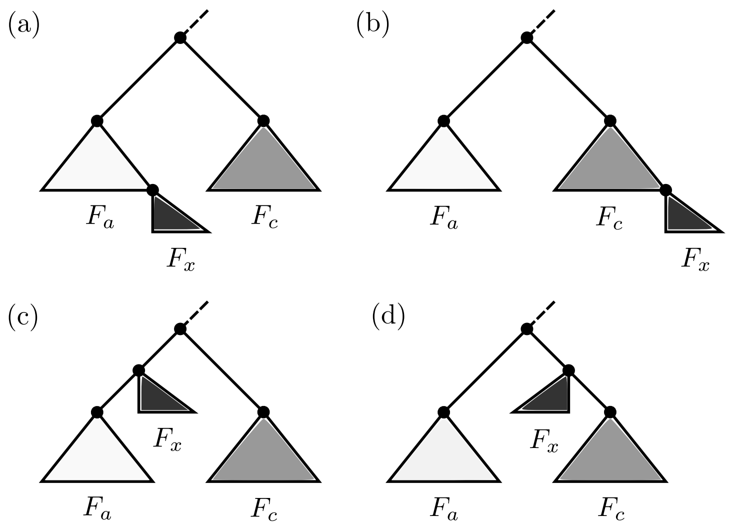


Figure 2.12: The possible locations of F_a, F_c, and F_x in one of the input trees T_1 or T_2 discussed in Case (2.3.1) **(a,b)**, Case (2.3.2) **(c)**, and Case (2.3.3) **(d)**.

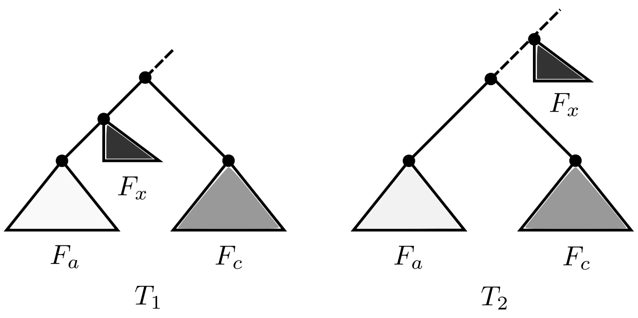


Figure 2.13: An illustration of the scenario as discussed in the proof regarding Case (2.3.2). Notice that, due to location of F_x, after attaching F_c back to F_a the cycle (F_r, F_x, F_r) arises in the underlying modified ancestor descendant graph.

Now, for simplicity, in the following we assume that $\mathcal{P}$ is of minimum length such that $\mathcal{P} = (F_x, F_c)$. Now, due to the edge (F_r, F_x), this directly implies that $AG^*(T_1, T_2, \hat{\mathcal{F}})$ contains a directed cycle $\hat{\mathcal{C}} = (F_r, \hat{F}_x, F_r)$ with $\hat{F}_x \cong F_x$. In this case, however, we will show that there already has to exist a certain cycle in $AG^*(T_1, T_2, \tilde{\mathcal{F}}_i^{\text{ex}})$ containing the expanded component of $\tilde{F}^{\text{ex}}$. For a better understanding we have illustrated the described scenario in Figure 2.13.

Since a and c is a sibling in R_i, there has to exist a component $\tilde{F}_x$ in $\tilde{\mathcal{F}}_i^{\text{ex}}$ with $F_x \subseteq \tilde{F}_x$ and $\tilde{F}_x \neq \tilde{F}^{\text{ex}}$ with, as denoted in Lemma 2.11, $\tilde{F}^{\text{ex}}$ being the expanded component corresponding to $\tilde{F}$ in $\tilde{\mathcal{F}}_i^{\text{ex}}$. This is, in particular, the case, since initially $\tilde{F}_x$ must have been part of a pendant subtree lying on the path leading from $\text{LCA}_T(\mathcal{L}(F_a))$ to $\text{LCA}_T(\mathcal{L}(F_r))$ with $T \in \{T_1, T_2\}$ and, as $\{\mathcal{L}(a), \mathcal{L}(c)\}$ now forms a common cherry, each pendant subtree lying on this path must have been cut so far. Thus, due to $\tilde{F}_x$, there is an edge in $AG^*(T_1, T_2, \tilde{\mathcal{F}}_i^{\text{ex}})$ leading from $\tilde{F}^{\text{ex}}$ to $\tilde{F}_x$. Moreover, due to the existence of $\mathcal{P}$ in $AG^*(T_1, T_2, \mathcal{F})$, this directly implies that there is also an edge leading from $\tilde{F}_x$ to $\tilde{F}^{\text{ex}}$ and, as a direct consequence, $AG^*(T_1, T_2, \tilde{\mathcal{F}}_i^{\text{ex}})$ contains the cycle $\tilde{\mathcal{C}} = (\tilde{F}^{\text{ex}}, \tilde{F}_x, \tilde{F}^{\text{ex}})$.

Note that, if $\mathcal{P}$ is not of minimum size, similar to F_x, each component in $\mathcal{F}_c$ has to be contained in a component in $\mathcal{F}_i^{\text{ex}}$ unequal to $\tilde{F}^{\text{ex}}$. Moreover, because different components in $\mathcal{F}_c$ can be contained in the same component in $\tilde{\mathcal{F}}_i^{\text{ex}}$, the cycle $\tilde{\mathcal{C}}$ can be shorter than $\hat{\mathcal{C}}$. This scenario can occur due to subsequent cherry actions, that, when processing a contradicting cherry, split components that are contained in $\tilde{\mathcal{C}}$ which automatically increases the size of the cycle.

In conclusion, by combining Case (1) and Case (2) we can deduce that if there exists a directed cycle in $AG^*(T_1, T_2, \hat{\mathcal{F}})$ involving the component F_r, there has to exist a directed cycle in $AG^*(T_1, T_2, \tilde{\mathcal{F}}_i^{\text{ex}})$ involving the expanded component of $\tilde{F}$. Hence, regarding Lemma 9, if $\mathcal{F}$ is acyclic and if there does not exist a cycle in $AG^*(T_1, T_2, \tilde{\mathcal{F}}_i^{\text{ex}})$ including the expanded component of $\tilde{F}^{\text{ex}}$, then, $AG^*(T_1, T_2, \hat{\mathcal{F}})$ must be acyclic. □

Now, based on Lemma 8 and 9, we can formulate the following observation.

Observation 1. *Let* PROCESSCHERRIES$(R_i, \mathcal{F}_i, \wedge_i = ((\{\mathcal{L}(a), \mathcal{L}(c)\}, \emptyset), \dots, \wedge_k))$ *be a call in which* $\{\mathcal{L}(a), \mathcal{L}(c)\}$ *is a common cherry for* R_i *and* $\mathcal{F}_i$*. If* $AG^*(S, T, \mathcal{F}_i)$ *does not contain a directed cycle including the component of* $\mathcal{F}_i \div \{\mathcal{L}(a), \mathcal{L}(c)\}$ *containing the taxa set* $\mathcal{L}(a) \cup \mathcal{L}(c)$*, both calls, in which* $\wedge_i$ *is replaced by* $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_a)$ *and* $(\{\mathcal{L}(a), \mathcal{L}(c)\}, e_c)$*, respectively, cannot result in a maximum acyclic agreement forests.*

This means, in particular, when processing a common cherry $\{\mathcal{L}(a), \mathcal{L}(c)\}$, both recursive calls resulting from a deletion of the in-edge of a and c, respectively, can only produce a maximum acyclic agreement forest if $AG^*(T_1, T_2, \mathcal{F})$ contains a directed cycle including the component containing the taxa set $\{\mathcal{L}(a) \cup \mathcal{L}(c)\}$. In return, if $AG^*(T_1, T_2, \mathcal{F})$ does not contain such a directed cycle, one can omit these additional recursive calls and still receives all maximum acyclic agreement forests. Thus, by combining Lemma 8 and 9, Theorem 7 is established. □

2.4.4.4 Correctness of allMAAFs^3

The correctness of our third modified algorithm ALLMAAFS^3 principally directly follows from the correctness of the refinement step, which can be found in the work of Whidden *et al.* [69]. We still have to show, however, that omitting both additional cutting steps when processing a common cherry still computes all of those agreement forests from which all maximum acyclic agreement forest can be obtained by cutting some of its edges.

Theorem 10. *Let* T_1 *and* T_2 *be two rooted binary phylogenetic* $\mathcal{X}$*-trees and* $k \in \mathbb{N}$*. Calling*

$$\text{ALLMAAFS}^3(T_1, T_2, T_1, \{T_2\}, k, \emptyset)$$

returns all maximum acyclic agreement forests for T_1 *and* T_2 *if and only if* $k \geq h(T_1, T_2)$*.*

Proof. Given two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, assume that our first modified algorithm ALLMAAFS^1 contains a computational path calculating a maximum acyclic agreement forest $\mathcal{F}$ of size k by cutting instead of contracting a common cherry. More precisely, let $\bigwedge = (\wedge_1, \ldots, \wedge_i = (\{\mathcal{L}(a), \mathcal{L}(c)\}, e_c), \ldots, \wedge_n)$ be the cherry list mimicking this computational path and, without loss of generality, let $\wedge_i$ be the only cherry action cutting instead of contracting a common cherry. Then, as already discussed in the proof of Lemma 8, there additionally exists a cherry list $\bigwedge'$ calculating a specific agreement forest $\hat{\mathcal{F}}$ of size $k-1$ containing a component $\hat{F}$. More specifically, let F_a and F_c be the expanded component in $\mathcal{F}$ which is derived from the contracted node a and c, respectively. Then, $\hat{\mathcal{F}}$ is obtained from $\mathcal{F}$ by reattaching F_c back to F_a (cf. Fig. 2.10). Thus, as a direct consequence, by cutting the in-edge corresponding to the root of $\hat{F}(\mathcal{L}(F_c))$ the maximum acyclic agreement forest $\mathcal{F}$ arises.

This implies that, if cutting instead of contracting a common cherry yields a maximum acyclic agreement forest $\mathcal{F}$, the algorithm allMAAFs^3 guarantees the computation of an agreement forest $\hat{\mathcal{F}}$ from which $\mathcal{F}$ can be obtained by cutting some of its edges. Note that the refinement step is always able to identify the minimal number of such edges and, thus, as the only difference between the first modified algorithm and the algorithm allMAAFs^3 consists in the way a common cherry is processed, Theorem 10 is established. □

2.4.5 Simulation Study

Our simulation study has been conducted on the same synthetic dataset as the one used for the simulation study reported in the work of Albrecht *et al.* [6], previously presented in Section 2.2.3. It consists of binary phylogenetic tree pairs that have been generated in respect to one combination of the following three parameters: the *number of taxa* $n = \{20, 50, 100.200\}$, the *executed number of rSPR-moves* $k = \{5, 10, \ldots, 50\}$, and the *tangling degree* $d = \{3, 5, 10, 15, 20\}$. For each combination of those parameters 10 tree pairs have been generated, resulting in 2000 tree pairs in total.

More precisely, a tree pair (T_1, T_2) is computed as follows. In a first step, the first tree T_1 with n leaves is computed which is done initially by randomly selecting two nodes u and v of a specific set $\mathcal{V}$ consisting of n nodes of both in- and out-degree 0. Then, those two selected nodes u and v are connected to a new node w and, finally, $\mathcal{V}$ is updated by replacing u and v by its parent w. This process is repeated until $\mathcal{V}$ consists only of one node corresponding to the root of T_1. In a subsequent step, T_2 is computed by applying k rSPR-moves to T_1 each respecting tangling degree d.

When performing a sequence of rSPR-moves, one can undo or redo some of those moves and, thus, k is only an upper bound of the real underlying rSPR-distance corresponding to both trees of a tree pair. The tangling degree, as already described in the work of Albrecht *et al.* [6], is an *ad hoc* concept controlling the number of minimum common clusters during the construction of a tree pair. Since, however, all the four implementations that are tested on the synthetic dataset perform a cluster reduction and, thus, equally benefit from the number of minimum common clusters, this number is irrelevant for our simulation study and, consequently, we do not give any further details about this parameter. Instead, we refer the interested reader to the work of Albrecht *et al.* [6].

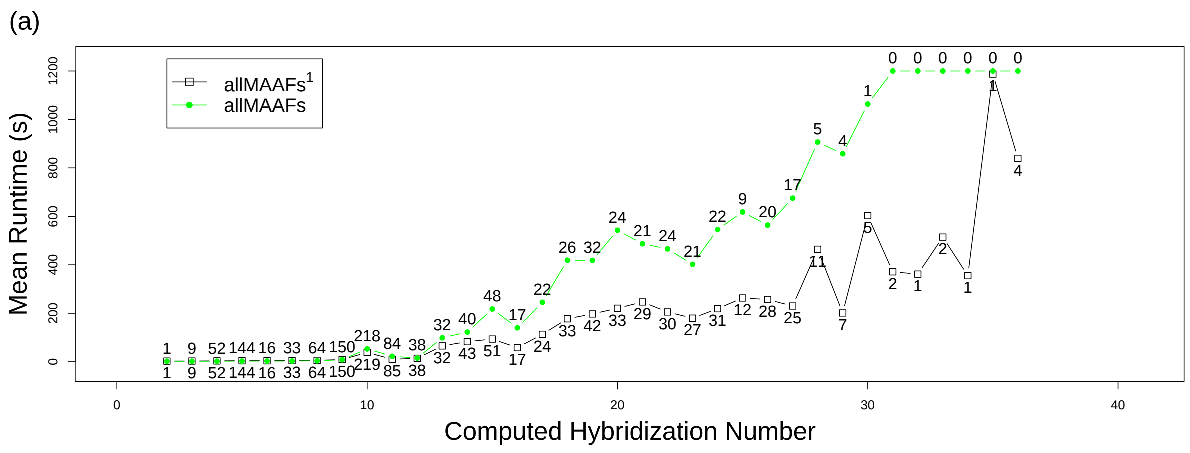
In this section, the practical runtimes produced by the respective implementations of our three modifications are compared with the practical runtimes of an implementation of the original algorithm. More precisely, by applying our synthetic dataset the practical runtime was measured for an implementation of the original algorithm allMAAFs, the algorithm allMAAFs^1, the algorithm allMAAFs^2, and the algorithm allMAAFs^3. Each of those algorithms has been integrated as a plug-in into the freely available Java software Hybroscale[4]. The simulation study has been conducted on a grid computer providing 16 cores and 40 GB RAM. In order to receive a reasonable set of completed data sets within an appropriate time period, we decided to compute only the hybridization number and omitted the computation of all maximum acyclic agreement forests. Moreover, we set the maximum runtime of each tree pair to 20 minutes, which means that each tree pair whose computation of the hybridization number could not be finished within 20 minutes was aborted. Notice that depending on the runtime analysis, which is shown in the following, those aborted tree pairs were either not taken into account or counted as being finished after 20 minutes.

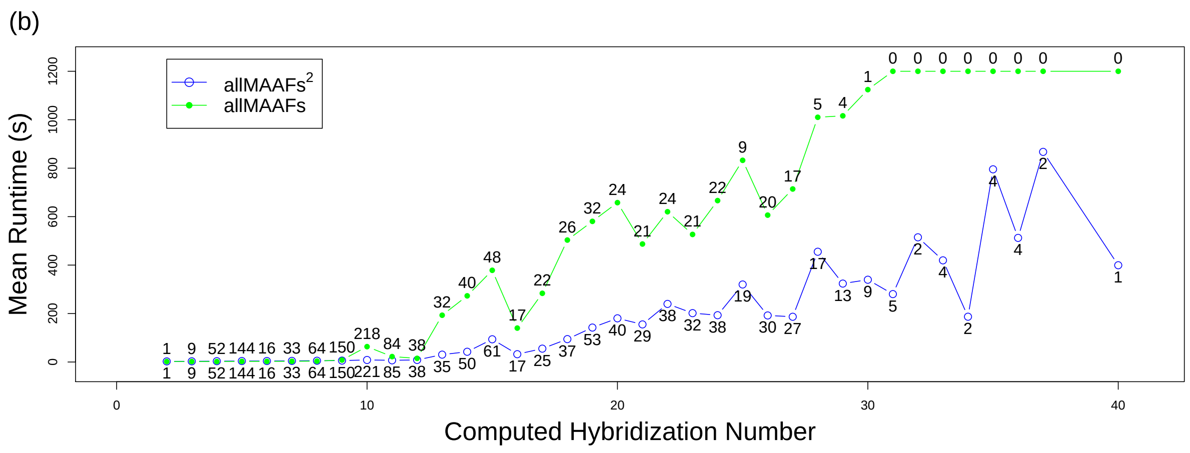
The problem arising when computing all, instead of just one, maximum acyclic agreement forests for certain tree sets is that, typically, there exists a large number of those agreement forests all being distributed in a vast search space. Consequently, even if those tree sets are of low computational complexity, one has to investigate far more than 20 minutes, which means that one could not conduct the simulation study within an appropriate time period. Otherwise, if we would choose a relatively small maximum runtime, such as 20 minutes, we could only process those tree sets of low computational complexity, which are not able to indicate the speedup obtained from applying our three modifications.

In both Figures 2.14 and 2.15, the mean average runtimes in terms of the computed hybridization numbers are shown. More precisely, the plot was generated by first aggregating all tree pairs corresponding to the same hybridization number and then by computing the mean average runtime of each of those aggregated subsets. Therefor, each aborted tree pair whose computation would have taken longer than the given time limit of 20 minutes, was treated as follows. If the tree pair could not be computed by both considered algorithms, this tree pair was not taken into account. However, if at least one algorithm was able to compute the hybridization number for a tree pair, the runtime of the other algorithm was set to 20 minutes if this algorithm was aborted in this case. Regarding both Figures 2.14 and 2.15, the number assigned to each measurement denotes the number of tree pairs whose hybridization number could be computed within the time limit by the corresponding two algorithm.

Figure 2.14 indicates that each of our three modified algorithms outperforms the original algorithm. Moreover, Figure 2.15 demonstrates that our second modified algorithm allMAAFs^2 is significantly more efficient than our first modified algorithm allMAAFs^1, which is also the case for our third modified algorithm allMAAFs^3. Moreover, by comparing the mean average runtimes of the algorithm allMAAFs^2 and allMAAFs^3, it turns out that allMAAFs^3 is slightly more efficient. Due to the large difference between

[4]`www.bio.ifi.lmu.de/softwareservices/hybroscale`





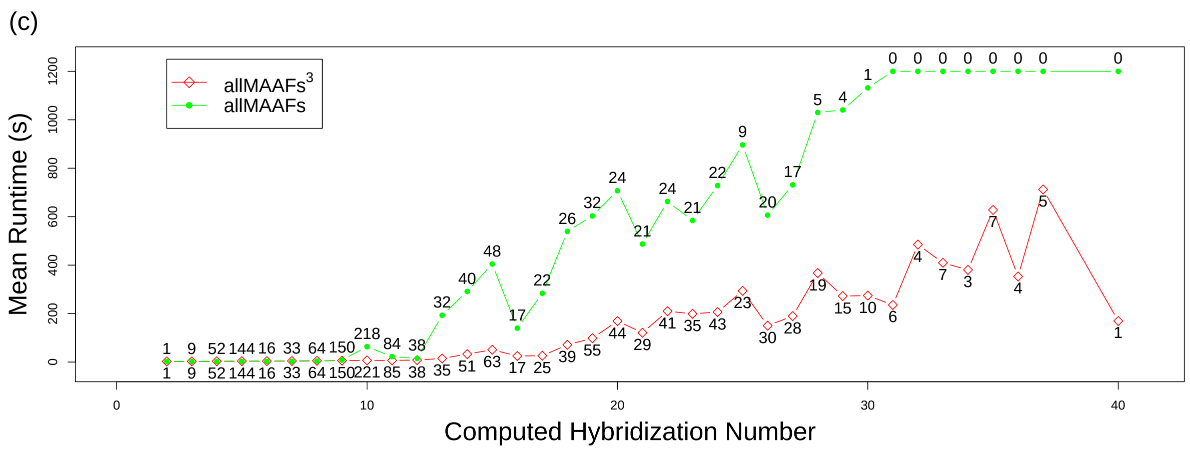


Figure 2.14: Comparisons between the mean average runtimes in terms of the hybridization number of (a) ALLMAAFS and ALLMAAFS[1], (b) ALLMAAFS and ALLMAAFS[2], and (c) ALLMAAFS and ALLMAAFS[3]. For each hybridization number the corresponding number of tree pairs is given that could be computed within the time limit of 20 minutes by the two respective algorithms and, thus, contributed to the mean average runtime denoted by the y-axis.

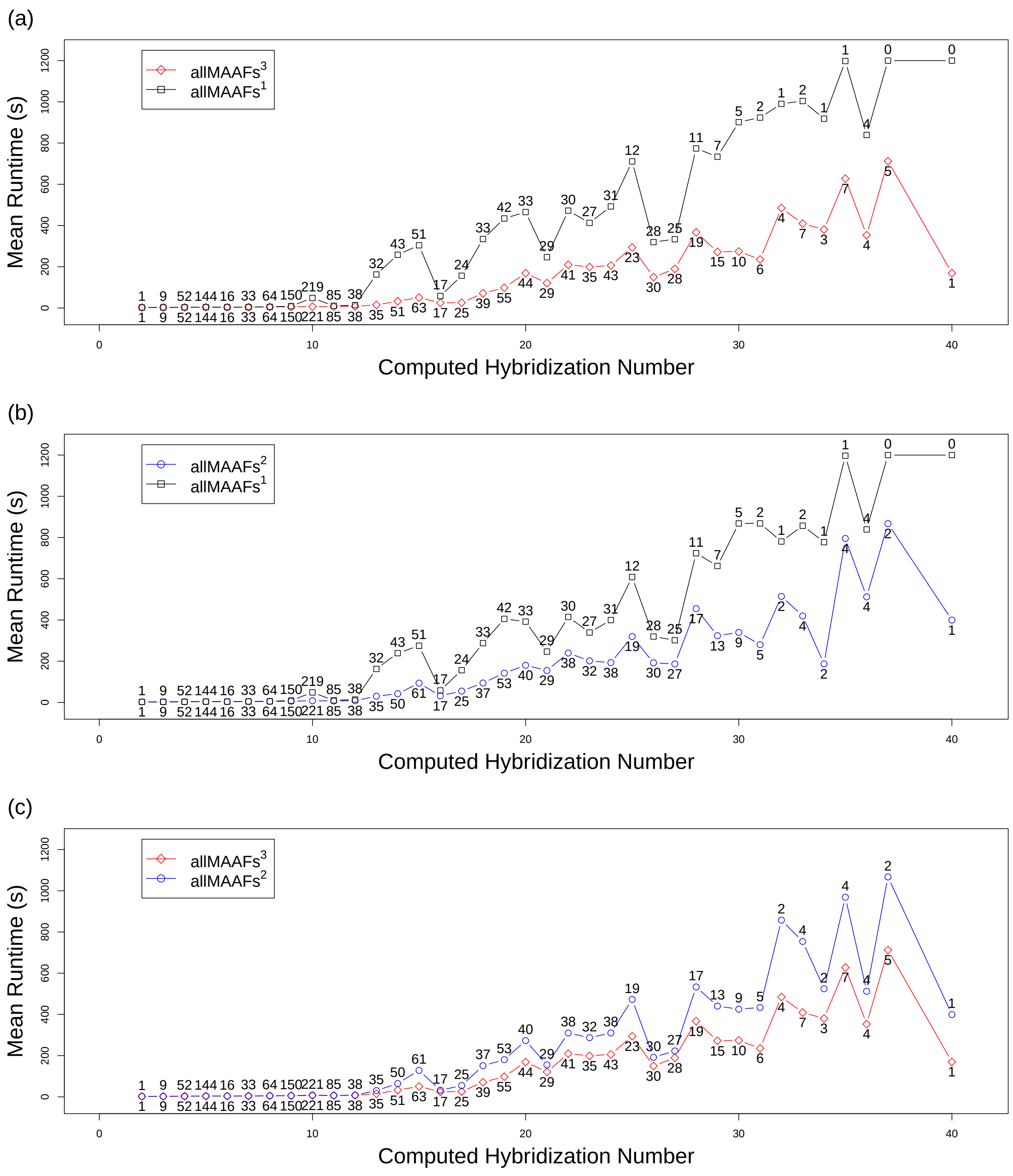


Figure 2.15: Comparisons between the mean average runtimes in terms of the hybridization number of (a) ALLMAAFs^1 and ALLMAAFs^3, (b) ALLMAAFs^1 and ALLMAAFs^2, and (c) ALLMAAFs^2 and ALLMAAFs^3. For each hybridization number the corresponding number of tree pairs is given that could be computed within the time limit of 20 minutes by the two respective algorithms and, thus, contributed to the mean average runtime denoted by the y-axis.

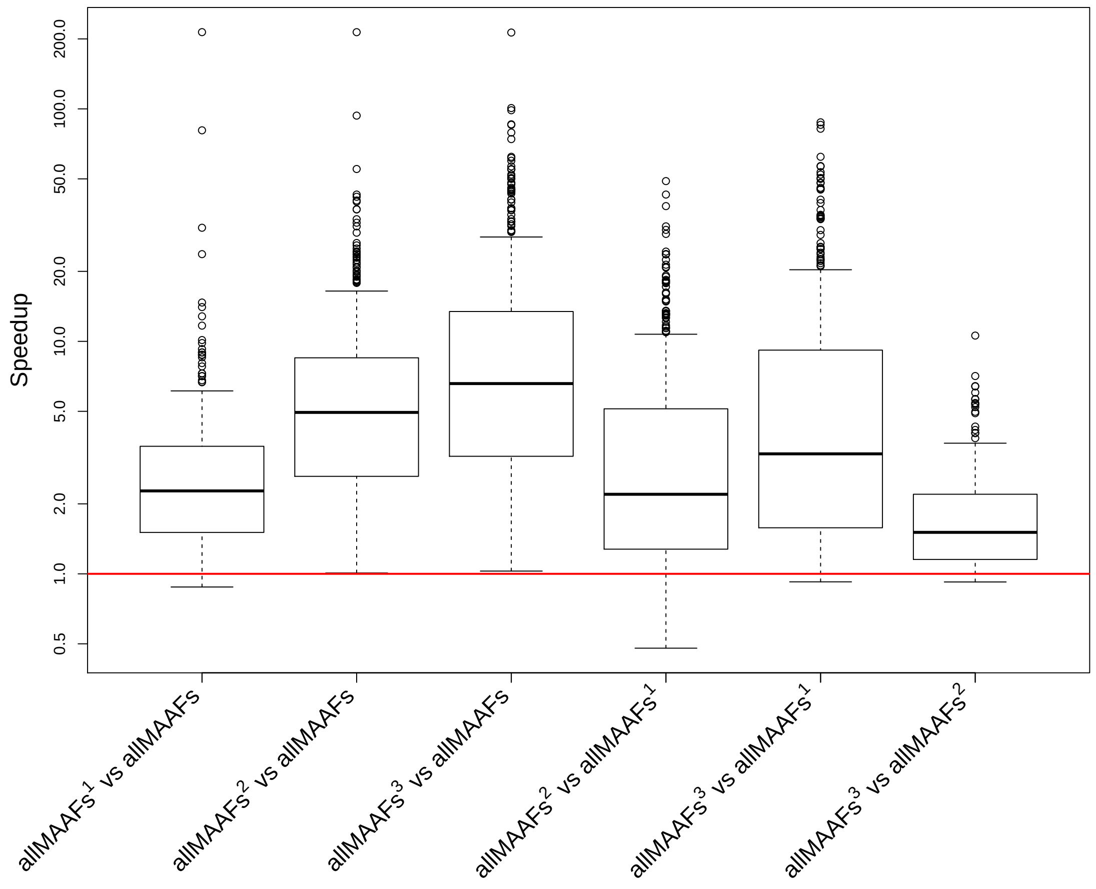


Figure 2.16: The distributions of the speedups, obtained from comparing algorithm A_1 versus algorithm A_2, as denoted at the x-axis, via boxplots. A tree pair d from our synthetic dataset was only considered if at least one of the two algorithms could process d within the time limit of 20 minutes and if at least one algorithm took longer than 50 seconds for its computation. Notice that a base-10 log scale is used for the y-axis.

the number of tree pairs that could be computed within the time limit of 20 minutes, we expect that a higher time limit increases the gap between the runtimes of the first modified algorithm and the other two modified algorithms even more.

Figure 2.16 shows the distributions of the speedups obtained from our three modified algorithms via boxplots. Each of those boxplots was generated by first selecting a relevant set $\mathcal{D}$ of tree pairs from our synthetic dataset and then by computing the speedup of each of those tree pairs by taking the runtime obtained from the corresponding two algorithms into account. More precisely, in a first step, we set the runtime of each tree pair whose hybridization number could not be computed within the time limit to 20 minutes. Second, we consider each tree pair d as relevant if at least one of both algorithms could process d within the time limit of 20 minutes and if at least one computation referring to one of both algorithms took longer than 50 seconds. Consequently, the relevant set $\mathcal{D}$ excludes those tree pairs whose computational complexity is, on the one hand, too low and, on the other hand, too high to reveal a difference between the runtimes of both algorithms.

Figure 2.16 again indicates that all three modified algorithms are more efficient than the original algorithm. More specifically, for the considered set of tree pairs, allMAAFs^1, allMAAFs^2 and allMAAFs^3 is on mean average about 3.9 times, 7.9 times, and 11.9 times (median 2.3, 4.9 and 6.6), respectively, faster than allMAAFs. Moreover, both the second and the third modified algorithms can significantly improve the practical runtime of the first modified algorithm. More specifically, for the considered set of tree pairs, allMAAFs^2 and allMAAFs^3 is on mean average about 4.7 times and 8.15 times (median 2.19 and 3.18), respectively, faster than allMAAFs^1. Lastly, the algorithm allMAAFs^3 is usually slightly faster (mean average 1.88 and median 1.50) than the algorithm allMAAFs^2.

In order to give a reason of the speedup obtained by our three modified algorithms, we measured the number of recursive calls that has been performed for calculating the hybridization number of each tree pair. To draw comparisons, we only took those tree pairs into account whose hybridization number could be processed by each corresponding pair of algorithms within the time limit of 20 minutes. For the first pair, consisting of allMAAFs^1 and allMAAFs^3, these are 1302, for the second pair, consisting of allMAAFs^1 and allMAAFs^2, these are also 1302, and for the third pair, consisting of allMAAFs^2 and allMAAFs^3, these are 1407. Next, all these tree pairs were grouped according to their hybridization number and, finally, the mean average number of recursive calls for each group was computed. Regarding Figure 2.17 and 2.18, the numbers that are attached to each measurement correspond to the number of tree pairs that have contributed to the mean average denoted at the x-axis.

Figure 2.17 indicates that by applying our modified algorithms there are significantly less recursive calls necessary for the computation of a maximum acyclic agreement forest compared with the original algorithm. Moreover, Figure 2.18 (a) and (b) shows that both the second and the third modified algorithms allMAAFs^2 and allMAAFs^3 have to conduct significantly less recursive calls than our first modified algorithm for computing hybridization numbers which, obviously, compensates the effort of preventing additional recursive calls when processing a common cherry. Since the computation of the hybridization number is just an intermediate step in computing all maximum acyclic agreement forests, the difference between the number of recursive calls between the first and both the second and the third modified algorithm is expected to be even larger in this case.

Figure 2.18 (c) shows that the difference between the mean average numbers of recursive calls conducted by the second and the third modification is a rather small. This implies that, regarding the original algorithm, there are only a few computational paths arising from those additional cutting steps that are applied when processing common cherries leading to maximum acyclic agreement forests. Consequently, this plot reveals that the better practical runtime of our second modification allMAAFs^3 compared with our first modification allMAAFs^2 is mainly due to the subroutine (cf. Alg. 8, Line 9,10) deciding if two additional computational paths are necessary when processing common cherries.

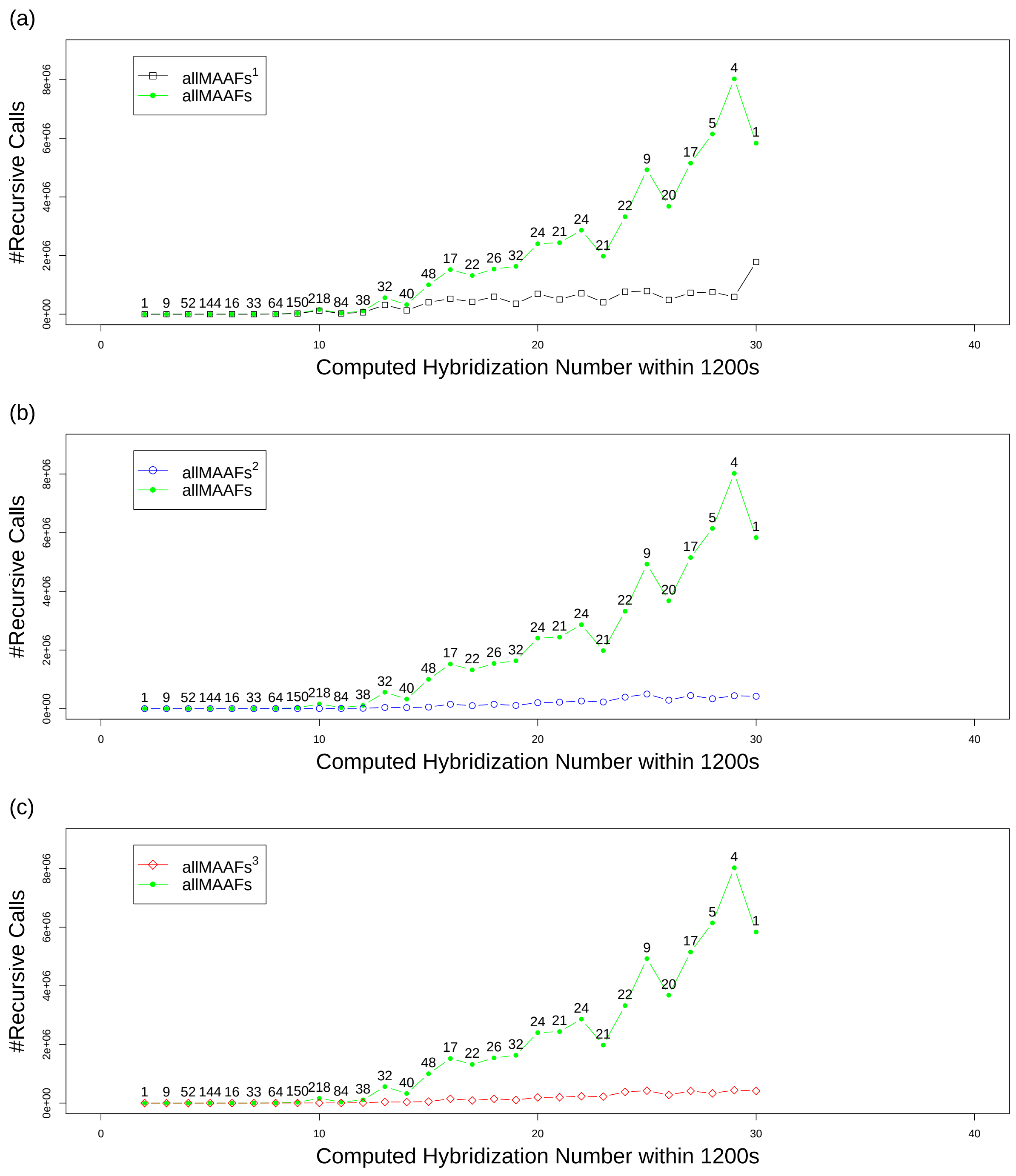


Figure 2.17: Comparisons between the mean average number of recursive calls in terms of the hybridization number of (a) ALLMAAFS and ALLMAAFS^1, (b) ALLMAAFS and ALLMAAFS^2, and (c) ALLMAAFS and ALLMAAFS^3. For each hybridization number the corresponding number of tree pairs is given that could be computed within the time limit of 20 minutes by the two respective algorithms and, thus, contributed to the mean average number of recursive calls denoted by the y-axis.

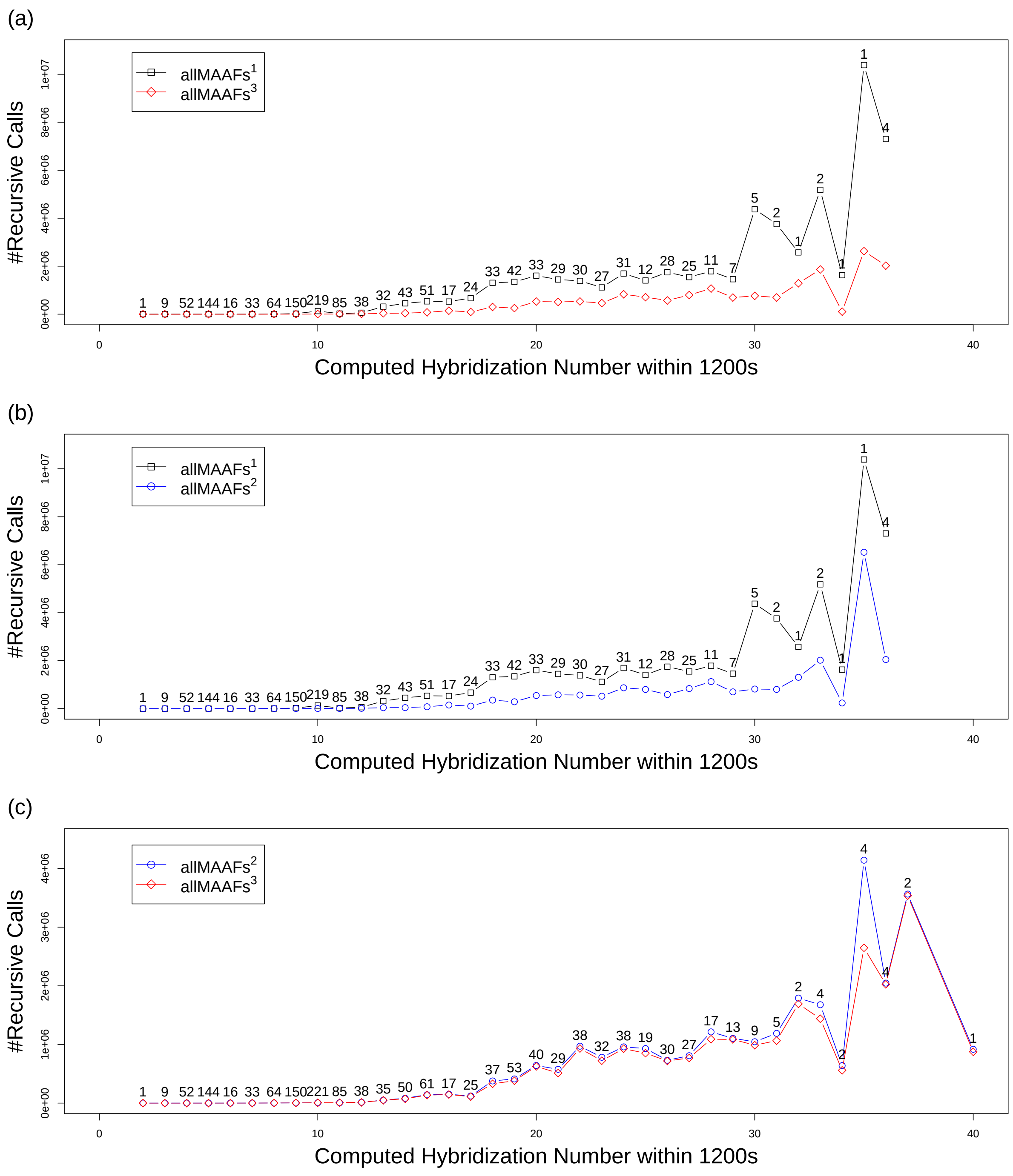


Figure 2.18: Comparisons between the mean average number of recursive calls in terms of the hybridization number of (a) allMAAFs^1 and allMAAFs^3, (b) allMAAFs^1 and allMAAFs^2, and (c) allMAAFs^2 and allMAAFs^3. For each hybridization number the corresponding number of tree pairs is given that could be computed within the time limit of 20 minutes by the two respective algorithms and, thus, contributed to the mean average number of recursive calls denoted by the y-axis.

Even though this subroutine is of rather low computational complexity, it typically has to be performed to a certain extend having a significant impact on the practical runtime.

Finally, we finish this section by comparing the runtimes of the four algorithms via scatter-plots (cf. Fig. 2.19). Again, all tree pairs whose hybridization number could not be computed within the time limit was set to 20 minutes. The plots clearly show that our modified algorithms outperform the original algorithm and that the modified algorithm allMAAFs^3 is most efficient followed by the modified algorithm allMAAFs^2 and the modified algorithm allMAAFs^1. However, as mentioned at the beginning of this section, one has to take into account that our third modification allMAAFs^3 is much more complex than our second modification allMAAFs^2, which means that its implementation is much more sophisticated. Thus, we think that allMAAFs^2 is nevertheless a valuable modification significantly improving the practical runtime of the original algorithm.

2.4.6 Runtime of allMAAFs^1, allMAAFs^2 and allMAAFs^3

The theoretical worst-case runtime of allMAAFs^1 and allMAAFs^2 is still the same as the one of the original algorithm allMAAFs, which is $O(3^{|\mathcal{X}|}|\mathcal{X}|)$ as stated in the work of Scornavacca *et al.* [57, Theorem 3].

Theorem 11. *Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and $k \in \mathbb{N}$. The two algorithms* allMAAFs$^1(T_1, T_2, T_1, \{T_2\}, k)$ *and* allMAAFs$^2(T_1, T_2, T_1, \{T_2\}, k)$ *have a theoretical worst-case runtime of $O(3^{|\mathcal{X}|}|\mathcal{X}|)$.*

Proof. Let $\mathcal{F} = \{F_\rho, F_1, F_2, \ldots, F_{k-1}\}$ be an agreement forest for T_1 and T_2 of size k. To obtain $\mathcal{F}$ from T_2, one obviously has to cut $k-1$ edges. Moreover, in order to reduce the size of leaf set $\mathcal{X}$ of R to 1, to each component F_i in $\mathcal{F}$ we have to contract exactly $|\mathcal{L}(F_i)| - 1$ cherries. Consequently, at most $|\mathcal{X}|$ cherry contractions have to be performed in total. Thus, our algorithm has to perform at most $k + |\mathcal{X}| = O(|\mathcal{X}|)$ recursive calls for the computation of $\mathcal{F}$. Now, as one of these recursive calls can at least branch into three further recursive calls, $O(3^{|\mathcal{X}|})$ is an upper bound for the total number of recursive calls that are performed throughout the whole algorithm. Moreover, as each operation that is performed during a recursive call can be performed in $O(|\mathcal{X}|)$ time, the worst-case runtime of both algorithms is $O(3^{|\mathcal{X}|}|\mathcal{X}|)$. □

When considering the theoretical worst-case runtime of allMAAFs^3, we have to take the runtime of a refinement step into account.

Theorem 12. *Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and $k \in \mathbb{N}$. The theoretical worst-case runtime of the algorithm* allMAAFs$^3(T_1, T_2, T_1, \{T_2\}, k)$ *is $O(3^{|\mathcal{X}|}4^k|\mathcal{X}|)$.*

Proof. As stated in Theorem 11, the algorithm has to conduct $O(3^{|\mathcal{X}|})$ recursive calls. Potentially, for each of these recursive calls we have to apply a refinement step whose theoretical worst-case runtime is stated with $O(4^k|\mathcal{X}|)$ [69]. Moreover, as all other operations that are performed during a recursive call can be performed in constant time, the theoretical worst-case runtime of the whole algorithm is $O(3^{|\mathcal{X}|}4^k|\mathcal{X}|)$. □

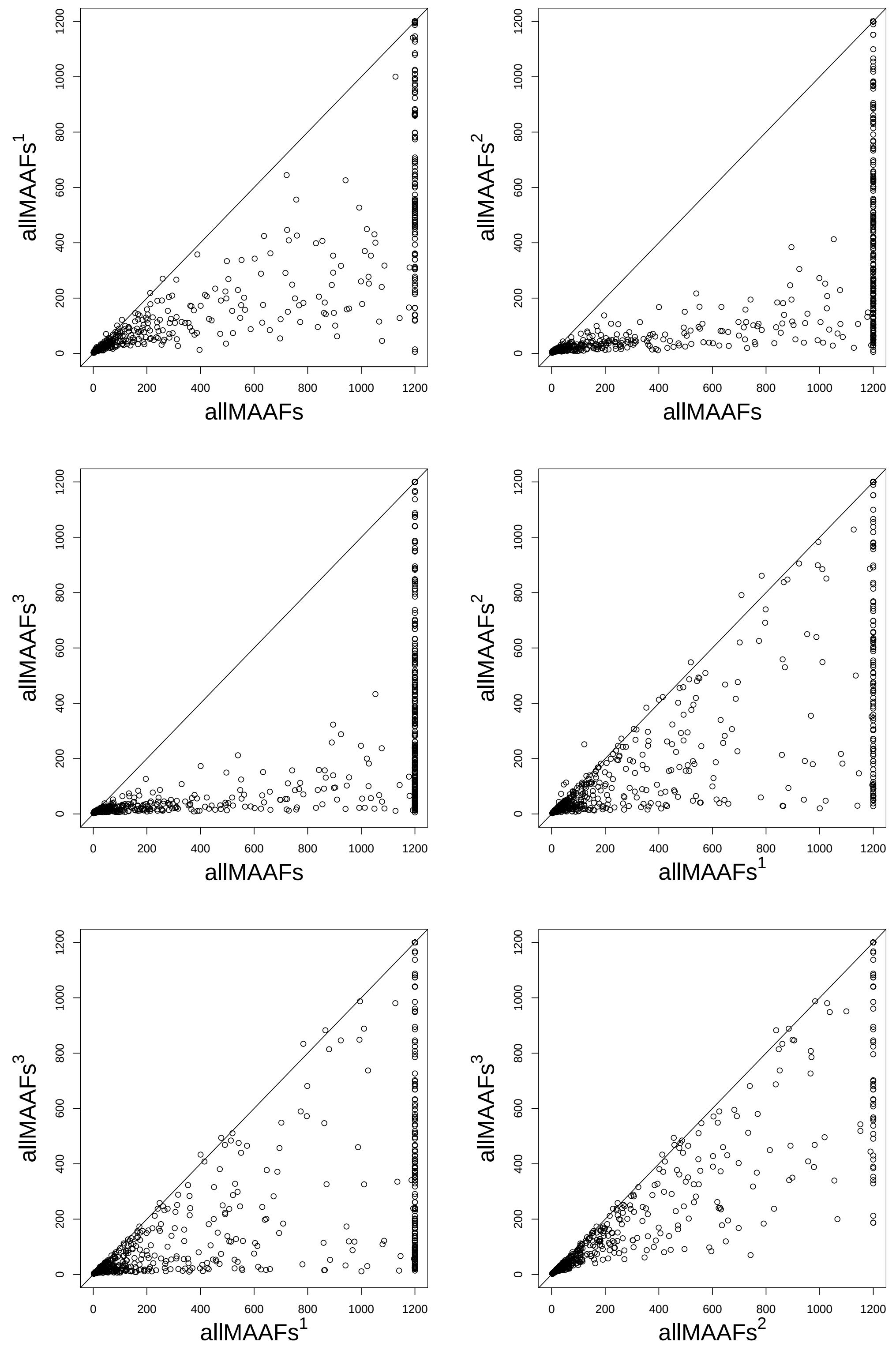


Figure 2.19: Six scatter-plots comparing the runtimes produced by the four different algorithms when applying our synthetic dataset. Each dot refers to a tree set of this data set whose corresponding x- and y-value indicates the runtime attained by the respective two algorithms.

Even though all three presented modifications do not improve the theoretical worst-case runtime, our simulation study indicates that in practice these modifications are significantly faster than the original algorithm allMAAFs. Regarding our first modification allMAAFs^1, this is simply due to the fact that the number of computational paths arising from processing a contradicting cherry is reduced. More precisely, if there is a computational path calculating a maximum acyclic agreement forest in which the set of pendant edges E_B for a particular cherry is cut, the original algorithms produces $|E_B|-1$ redundant recursive calls.

Furthermore, our second modification allMAAFs^2 additionally improves the way of processing a common cherry. Whereas the original algorithm always starts two recursive calls, each for cutting one of both adjacent edges of the chosen cherry, allMAAFs^2 first checks an extra constraint (cf. Alg. 5, line 10) that often prevents the algorithm from initiating computational paths not resulting in the computation of maximum acyclic agreement forests. More precisely, given a particular node v of the underlying modified ancestor descendant graph AG^*, these two additional recursive calls are performed only if there is a directed cycle in AG^* containing v, which obviously can be solved in linear time. As indicated by our simulation study, running this additional check is much more efficient in practice than running all unnecessary recursive calls not leading to a maximum acyclic agreement forest.

Due to the refinement steps that are performed at the end of each recursive call, our third modification allMAAFs^3 always has to start only one recursive call for processing a common cherry. This refinement step, however, has to be conducted for each maximum agreement forest $\mathcal{F}$ in order to transform $\mathcal{F}$ into all maximum acyclic agreement forests. Nevertheless, as indicated by our simulation study, this post processing step is still more efficient than always running two additional recursive calls when processing a common cherry (as it is the case for the original algorithm allMAAFs) even if the number of those recursive calls is limited by first checking the graph AG^* (as it is the case for the algorithm allMAAFs^2).

In conclusion, due to the significant speedup that can be obtained from applying the presented modifications, we feel certain that this work describes a noticeable improvement to the original algorithm allMAAFs, which will make this algorithm accessible for a wider range of biological real-world applications.

2.4.7 Conclusion

An important approach to receive specfic hybridization scenarios is the reconciliation of incongruent gene trees into certain kinds of phylogenetic networks, so-called hybridization networks. As already mentioned, in this context, biologists are in general interested not only in one but in *all* of those networks as, once calculated all of these networks, one can then apply certain filtering techniques testing certain hypothesis. As already discussed in Section 2.2, the recently published algorithm allMAAFs [57] calculating all maximum acyclic agreement forests for two rooted binary phylogenetic trees, is considered to be an important step to achieve this goal.

In this section, we have presented three modifications allMAAFs^1, allMAAFs^2 and allMAAFs^3 of the algorithm allMAAFs still calculating all maximum acyclic agreement forests for both input trees as shown by several formal proofs. Moreover, as we have additionally indicated by a simulation study that each of those modifications significantly improves the practical runtime of the original algorithm, we feel certain that this work will facilitate the computation of hybridization networks for larger input trees and, thus, makes network algorithms accessible to a wider range of biological problems.

2.5 TerminusEst

TerminusEst is, to our knowledge, so far the fastest algorithm calculating the hybridization number for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees [49]. In contrast to our algorithm allMAAFs, it is not based on cherries but on other types of nodes — called *terminals* (as defined later). This algorithm, however, can only be used to calculate one minimum hybridization network instead of all. Nevertheless, for this simplified problem this algorithm is in general much faster than our approach presented in Section 2.3 and, thus, we can use this approach in order to speed up the computation of all hybridization networks for two rooted binary phylogenetic $\mathcal{X}$-trees.

For this purpose, we integrated an optimized version of the algorithm TerminusEst – called TerminusEst*— into our existing approach (cf. Sec. 2.2) as follows. Before applying the algorithm allMAAFs returning all maximum acyclic agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2, we first calculate the minimum hybridization number h for those two trees using TerminusEst*. Consequently, we can directly check the first search space containing maximum acyclic agreement forests of size $h + 1$ such that each search looking for forests of smaller sizes can be skipped. We have undertaken a simulation study indicating that this technique is much faster than simply running the algorithm allMAAFs by step-wise increasing parameter k (beginning with $k = 0$).

2.5.1 The algorithm TerminusEst

The algorithm presented in the work of Piovesan and Kelk [49] is a quite simple practical bound-search algorithm being described on the concept of clusters. As in this thesis we want to focus on graphs, we will introduce the algorithm TerminusEst by transferring the respective definitions on rooted phylogenetic $\mathcal{X}$-trees.

Terminals. Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and let p_1 and p_2 be the parents of a leaf label by taxon x in T_1 and T_2, respectively. Then, taxon x is a *terminal for* T_1 *and* T_2, if $\mathcal{L}(p_1) \setminus \{x\} \cap \mathcal{L}(p_2) \setminus \{x\} = \emptyset$ (cf. Fig. 2.20). Moreover, by $\tau(T_1, T_2)$ we will refer to the set containing all terminals of two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2.

Minimum clusters. Let T be a rooted phylogenetic $\mathcal{X}$-tree and let v be a node whose children are all leaves. Then, $\mathcal{L}(v)$ is a minimum cluster of T (cf. Fig. 2.20).

Cutting taxa. Let T be a rooted phylogenetic $\mathcal{X}$-tree and let a be a leaf in T. Then, the operation of cutting a from T, shortly denoted by $T \div \mathcal{L}(a)$, is conducted by first removing a from T together with its in-going edge and then by suppressing each node of both in- and out-degree 1.

Collapsing maximal common pendant subtrees. Let T_1 and T_2 be two rooted binary phylogenetic $\mathcal{X}$-trees, then all maximal pendants subtrees T' of size ≥ 2 occurring in T_1 and T_2 are collapsed as follows. Let v_1 and v_2 be the root of such a maximal pen-

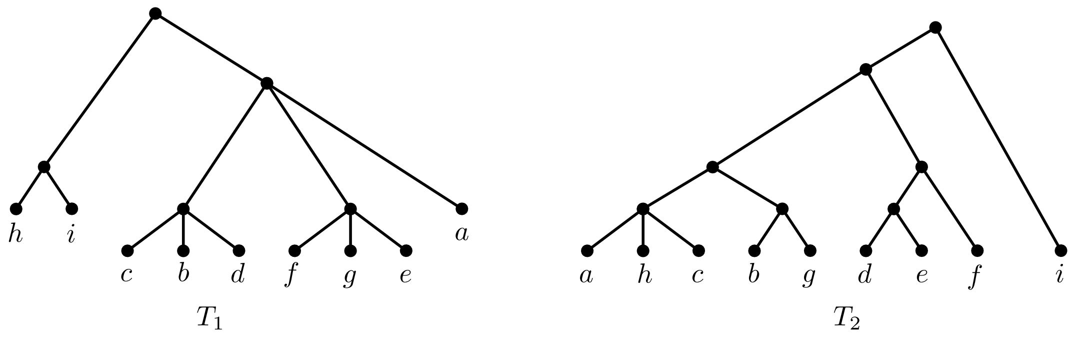


Figure 2.20: Two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, with $\{b, c, d, e, g, h\}$ being the set of terminals for both trees. Note that f, for example, is not a terminal as $\{g, e\} \cap \{e, d\} = \{e\}$ is not the empty set. Moreover, the minimal clusters of T_1 are $\{h, i\}$, $\{c, b, d\}$, and $\{f, g, e\}$, whereas the minimal clusters of T_2 are $\{a, h, c\}$, $\{b, g\}$, and $\{d, e, f\}$.

dant subtree T' in T_1 and T_2, respectively. Then, all nodes that can be reached from v_1 and v_2 are deleted followed by relabeling v_1 and v_2 by a new taxon so far not contained in $\mathcal{X}$.

Now, based on these definitions, we can briefly describe the recursive algorithm TerminusEst. We will assume that, at the beginning, the algorithm is initialized by two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 as well as a parameter $k \in \mathbb{N}$. Then, by calling the algorithm with those parameters the algorithm returns TRUE if and only if $h(T_1, T_2) \leq k$.

Step 1. First, collapse all maximal pendant subtrees of T_1 and T_2 resulting in two rooted binary $\mathcal{X}'$-trees T_1' and T_2' with $|\mathcal{X}'| < |\mathcal{X}|$. If T_1' (and T_2') only consists of one node, the algorithm returns TRUE and all other so far existing computational paths can be aborted immediately. Otherwise, if $k \geq 1$, continue with Step 2 (if $k = 0$, this computational path ends here).

Step 2. Based on T_1' and T_2', compute a set χ containing terminals from both trees as follows. If $|\tau(T_1', T_2')| > 2k$, χ has to be a set of size $2k + 1$ only containing terminals of T_1' and T_2'. Otherwise, if $|\tau(T_1', T_2')| \leq 2k$, χ has to be a set containing from each minimal cluster of T_1' and T_2' two elements such that at least one of both elements is a terminal for T_1' and T_2'. Note that for each minimal cluster there has to exist at least one terminal because, otherwise, there would exist a pendant subtree of T_1' and T_2' which is a contradiction to Step 1. Continue with Step 3.

Step 3. For each taxon x in χ, start a new recursive call with $T_1' \div \{x\}$, $T_2' \div \{x\}$, and $k - 1$.

Now, by running the algorithm for an increasing parameter k (beginning with $k = 0$), the first k for which the algorithm TerminusEst returns TRUE represents the hybridization number for both input trees. Moreover, notice that by recording the sequence of taxa that has been cut during a computational path returning TRUE one can construct a hybridization network for both input trees with reticulation number k. However, as for our purpose we are only interested in the hybridization number, this step is irrelevant and, thus, we omit a detailed description of this procedure here. Instead we refer the interested reader to the original literature [49].

2.5.2 Correctness of TerminusEst

To show the correctness of this algorithm it suffices to consider Step 2 dealing with the selection of a set χ containing certain taxa that have to be removed from both trees T_1 and T_2. We refer the interested reader to the original literature in which the two different ways of constructing χ are discussed in [49, Lemma 5] and [49, Theorem 1], respectively.

2.5.3 The algorithm TerminusEst*

The algorithm TERMINUSEST* is based on the original algorithm TERMINUSEST containing further algorithmic as well as technical modifications.

Terminals. If for both trees T_1 and T_2 there exist more than $3k$ terminals, each outgoing computational path will fail and, thus, the respective computational path can be aborted.

Conflicting clusters. Regarding Step 2 of the algorithm as described above, if for both trees T_1' and T_2' there exist two minimal clusters $\mathcal{L}(v_1)$ and $\mathcal{L}(v_2)$ of size two sharing exactly one common taxon, χ can be simply set to $\mathcal{L}(v_1) \cup \mathcal{L}(v_2)$.

Minimizing χ. Regarding Step 2 of the algorithm as described above, if $|\tau(T_1', T_2')| \leq 2k$, χ has to contain from each minimal cluster of T_1' and T_2' two elements such that at least one of both elements is a terminal for T_1' and T_2'. By maximizing the overlap of the two sets of elements separately calculated for T_1' and T_2', one obviously is minimizing the size of χ, which reduces the number of recursive calls. For example, regarding Figure 2.20, a valid choice for χ would be either $\{i, h, c, d, f, g, a, b, e\}$ or $\{h, i, c, b, g, e, d\}$, whereat the first choice would result in 9 recursive calls and the second one only in 7.

Our applied method for maximizing the overlap between those two sets is quite simple. When choosing elements from T_1, we simply take all potential elements from T_2 into account, which means that we predominantly choose those elements being elements that are also contained in a minimal cluster of T_2. Similarly, when choosing elements from T_2, we take the already chosen elements from T_1 into account. Note that this strategy does not guarantee the computation of a maximal overlap. Our simulation study, however, indicates that this approach performs quite well at least compared with the naive method not taking any potential taxa of the other tree into account.

In order to solve this problem in an optimal way, one could think of the following. First solve a minimum edge covering problem in which the underlying graph contains a node for each minimal cluster of both trees and an edge labeled with a terminal t, if t is contained in both corresponding clusters. Next, solve a minimum edge covering problem in which the underlying graph again contains a node for each minimal cluster of both trees and an edge labeled with taxon x, if x is contained in both corresponding clusters and has not been selected so far during the first edge cover. Finally, if for a minimal cluster less than two elements have been selected, choose arbitrary elements so that, however, at least one of both selected elements is a terminal. As the underlying graph of those edge covering problems are bipartite, we think that one could apply an efficient algorithm solving this specific problem in polynomial time.

Hashing. In order to avoid the processing of unsuccessful computational paths, one can set up a look-up table storing intermediate results. More precisely, let T'_1 and T'_2 be two trees for which each out-going computational path has failed, i.e., could not find a sequence of k' taxa so that both trees could be collapsed into a single node. Then, for this tree pair k' is a lower bound, which means that the hybridization number for T'_1 and T'_2 is at least $k' + 1$.

Now, based on this look-up table, given a tree pair T_1 and T_2 together with a parameter k, at the beginning of each recursive call one can check if a lower bound for T_1 and T_2 has already been calculated and, if this is the case, one can check, if k is larger than the recorded lower bound. If this is not the case, based on previous experience, one can be sure that each out-going computational path will fail and, thus, one can immediately abort the processing of this computational path.

Note that the implementation of TerminusEst, which is open source[5], only contains the first, the second, and the fourth modification. Moreover, our technique for hashing trees occurring on already visited computational paths to so far computed lower bounds is realized in a more space efficient way. This means, in particular, that our implementation needs less space for recording lower bounds, which is an important feature as the amount of lower bounds that are stored in the look-up table is exponential in the size of the input trees.

2.5.4 Correctness of TerminusEst*

Tho show the correctness of TerminusEst*, it obviously suffices to consider the first and the second modification presented above. The correctness of the first one, aborting computational paths whose corresponding trees contain more than $3k$ terminal, is established in [49, Lemma 3]. Moreover, the correctness of the second modification can be found in [49, Lemma 4], which is a subsequent work of the original literature.

[5]`http://skelk.sdf-eu.org/terminusest`

2.5.5 Runtime of TerminusEst

The theoretical worst-case runtime of both algorithms TerminusEst and TerminusEst*, respectively, applied to two phylogenetic $\mathcal{X}$-trees and a parameter $k \in \mathbb{N}$ is stated with $O(6^k k!\text{poly}(|\mathcal{X}|))$. This estimation is based on the observation that at the beginning χ can contain at most $6k$ elements and during each recursion k is always decreased by one [49, Theorem 1]. Moreover, the polynomial term covers all basic tree operations that are conducted during a recursive call, i.e., finding and collapsing all maximal common subtrees, computing the set of terminals for both trees followed by constructing χ, and pruning taxa of χ from both trees.

2.5.6 Simulation study

We conducted a simulation study on the same dataset as already used for the previously presented simulation study dealing with different modifications of the algorithm allMAAFs (cf. Sec. 2.4). For each tree set within this dataset, we calculated the minimum hybridization number by three different implementations, each based on our approach presented in Section 2.2. For technical purpose, we restricted the reductions rules on the subtree reduction, which means that before calculating the hybridization number for two rooted binary phylogenetic $\mathcal{X}$-trees by each of the following three different implementations, we did just replace all of its common maximal pendant subtrees. Moreover, for each tree set we set the maximal time limit to 20 minutes, which means that if the hybridization number could not be calculated within 20 minutes the computation of the respective tree set was aborted.

Implementation allMAAFs^3. This implementation conducts a search after the hybridization number for two rooted binary phylogenetic $\mathcal{X}$-trees by using the algorithm allMAAFs^3 presented in Section 2.4.3.3.

Implementation TerminusEst. This implementation conducts a search after the hybridization number for two rooted binary phylogenetic $\mathcal{X}$-trees by using the freely available[6] implementation of the algorithm TerminusEst.

Implementation TerminusEst*. This implementation conducts a search after the hybridization number for two rooted binary phylogenetic $\mathcal{X}$-trees by using our implementation of the algorithm TerminusEst*.

In Figure 2.21 and 2.22, we compare the runtime for the respective tree sets yielded by each of the three different implementations.

In Figure 2.21, the mean average runtimes in terms of the computed hybridization numbers are shown. More precisely, the plot was generated by first aggregating all tree pairs corresponding to the same hybridization number and then by computing the mean average

[6]`http://skelk.sdf-eu.org/terminusest`

runtime of each of those aggregated subsets. Therefor, each aborted tree pair whose computation would have taken longer than the given time limit of 20 minutes, was treated as follows. If the tree pair could not be computed by both considered implementations, this tree pair was not taken into account. However, if at least one implementation was able to compute the hybridization number for a tree pair, the runtime of the other implementation was set to 20 minutes, if this implementation was aborted in this case. Regarding Figure 2.21, the number assigned to each measurement denotes the number of tree pairs whose hybridization number could be computed within the time limit by the respective two implementations.

Figure 2.21 indicates that for calculating hybridization numbers an approach that is based on terminals is more efficient than an approach that is based on cherries. Whereas the implementation allMAAFs^3 just enables the computation of hybridization numbers up to 22, the other two implementation yield hybridization numbers up to 30. Moreover, the mean average runtime for calculating hybridization numbers larger than 17 attained by those implementations that are based on terminals are significantly lower than those corresponding to the implementation allMAAFs^3. In addition, due to the same reasons, Figure 2.21 demonstrates that our improved implementation TerminusEst* is significantly faster than the implementation TerminusEst, which shows that our algorithmic modifications can indeed improve the practical runtime.

Figure 2.22 compares the runtimes attained by the three implementations via scatterplots. Again, it turns out that both implementations that are based on terminals are in general faster than those corresponding to the implementation of allMAAFs^3. However, there are still some outliers for which the implementation allMAAFs^3 is still faster.

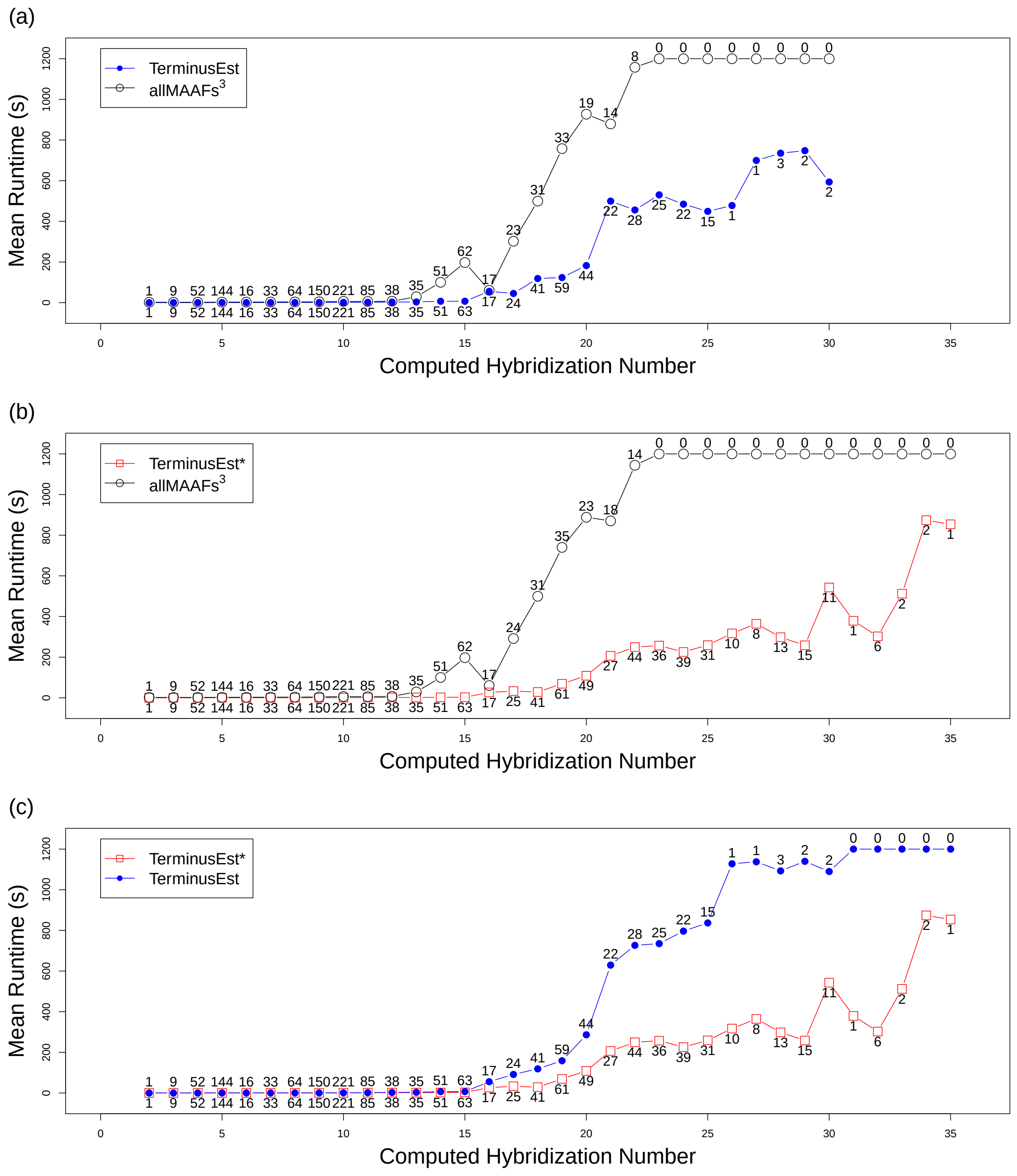


Figure 2.21: Comparisons between the mean average runtimes in terms of the hybridization number of (a) TerminusEst and allMAAFs^3, (b) TerminusEst*and allMAAFs^3, and (c) TerminusEst and TerminusEst*. For each hybridization number the corresponding number of tree pairs is given that could be computed within the time limit of 20 minutes by the two respective algorithms and, thus, contributed to the mean average runtime denoted by the y-axis.

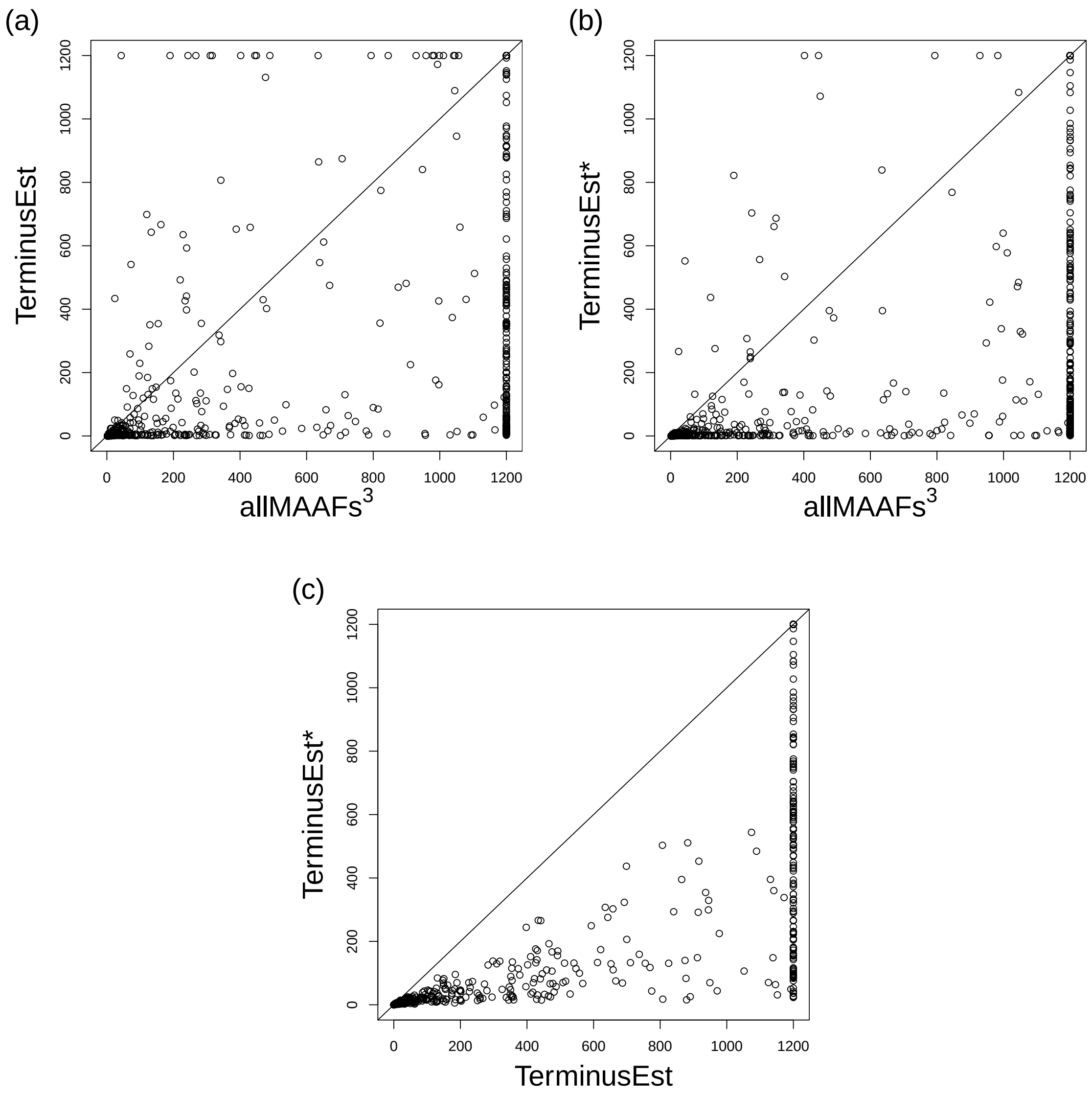


Figure 2.22: Three scatter-plots comparing the runtimes produced by the three different implementations when applying our synthetic dataset. Each dot refers to a tree set of this data set whose corresponding x- and y-value indicates the runtime attained by the respective two implementations.

2.5.7 Conclusion

In this section, we have presented the algorithm TerminusEst, which is so far the fastest algorithm calculating hybridization numbers for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. By presenting the result of a simulation study, we have indicated that this algorithm is indeed faster than our approach presented in Section 2.2, which is based on the algorithm allMAAFs. Notice, however, that, in contrast to the algorithm allMAAFs, the algorithm TerminusEst calculates only one instead of all maximum acyclic agreement forests. Nevertheless, this algorithm can still be valuable for an approach calculating minimum hybridization networks as it can be used to improve its efficiency.

Moreover, as indicated by an implementation of the algorithm TerminusEst*, we have shown that one can apply some simple modifications improving the practical runtime of the original algorithm. As noted above, one of those speed-up techniques, dealing with the maximization of an overlap between two node sets of minimal clusters (cf. Sec. 2.5.3), is indeed an interesting algorithmic challenging problem.

To further improve the runtime, one could still think of recursively applying the cluster reduction as follows. Right after collapsing all maximal common subtrees, one could first check if both trees T_1 and T_2 contain a common cluster A and, if this is the case, start two new recursive calls processing $T_1|_A$ and $T_2|_A$ as well as T_1' and T_2', where T_1' and T_2' are those trees obtained from T_1 and T_2 by removing the subtree whose taxa set equals A. Notice, however, that in order to maximize efficiency both recursive call have to communicate which minimum hybridization number its complementary subproblems requires.

Chapter 3

Hybridization networks for multiple binary trees

In this chapter, we first show that, regarding the computation of minimum hybridization numbers, the concept of a cluster reduction can be also applied to more than just two rooted binary phylogenetic $\mathcal{X}$-trees. Next, we present the algorithm allHNetworks calculating minimum hybridization networks for multiple rooted binary phylogenetic $\mathcal{X}$-trees and establish its correctness by a formal proof. Finally, for an approach, making use of this algorithm, first its efficiency is indicated and then its practical application to a biological dataset is demonstrated.

3.1 Further Definitions

In a first step, we give all further definitions that are crucial for this chapter.

Phylogenetic trees. Given a rooted binary phylogenetic $\mathcal{X}$-tree T, throughout this chapter by $\overline{T}$ we refer to the tree that is obtained from T by suppressing each node of both in- and out-degree 1.

Agreement forests. In this section, given an agreement forest, we assume that both roots of the two corresponding binary phylogenetic $\mathcal{X}$-trees are nodes of out-degree 2 that are connected by an edge to a leaf labeled by the taxon ρ.

Hybridization networks. Given a hybridization network N on $\mathcal{X}$ and an edge set E' referring to an embedded rooted phylogenetic $\mathcal{X}'$-tree T' of N with $\mathcal{X}' \subseteq \mathcal{X}$, the *restricted network* $N|_{E',\mathcal{X}'}$ refers to the minimal connected subgraph T only containing leaves labeled by $\mathcal{X}'$ and edges that are either tree edges or contained in E'. Consequently, $N|_{E',\mathcal{X}'}$ is a directed graph that corresponds to $T'|_{\mathcal{X}'}$ but still contains nodes of both in- and out-degree 1, and, thus, each node in $N|_{E',\mathcal{X}'}$ can be mapped back to exactly one specific node of the unrestricted network N (cf. Fig.3.1(c)).

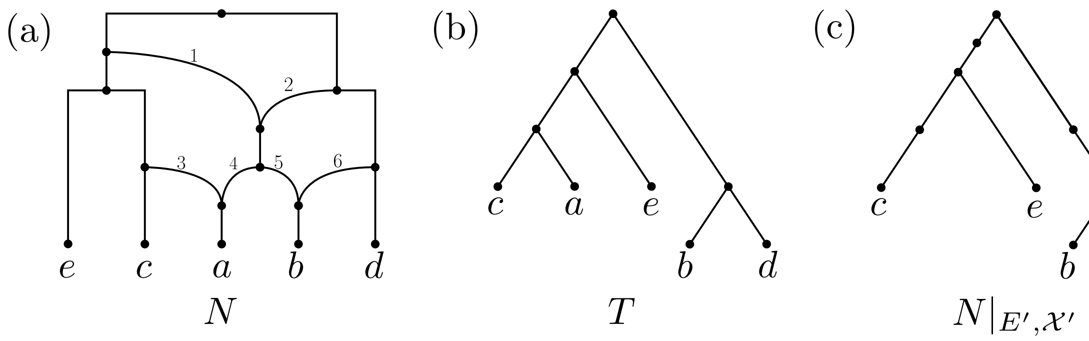


Figure 3.1: **(a)** A hybridization network N with taxa set $\mathcal{X} = \{a, b, c, d, e\}$ whose reticulation edges are consecutively numbered. **(b)** A phylogenetic $\mathcal{X}$-tree T that is displayed by N. Based on N, both edge sets $E' = \{3, 6, 1\}$ and $E'' = \{3, 6, 2\}$ refer to T and, thus, $\overline{N|_{E',\mathcal{X}}}$ as well as $\overline{N|_{E'',\mathcal{X}}}$ equals T. **(c)** The restricted network $N|_{E',\mathcal{X}'}$ with $\mathcal{X}' = \{b, c, d, e\}$ still containing nodes of both in- and out-degree 1.

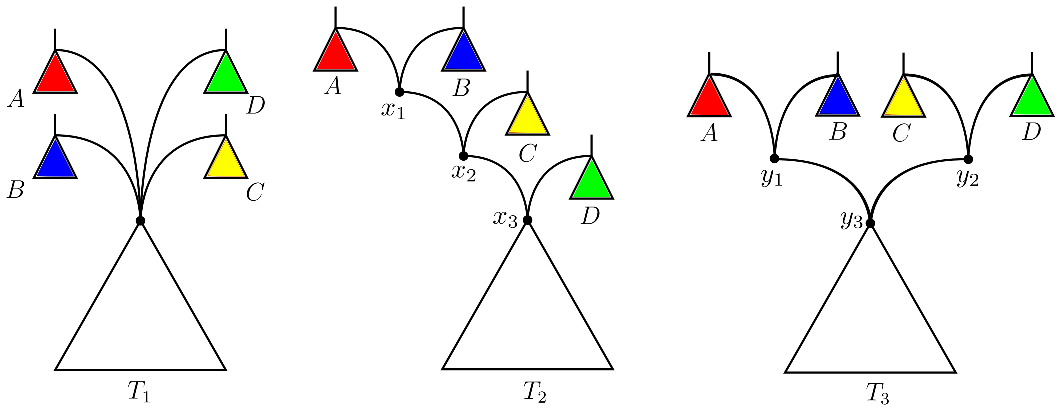


Figure 3.2: An illustration of stacks of hybridization nodes. The hybridization node with in-degree 4 of the left-hand tree T_1 can be resolved (amongst others) into distinctive stacks of hybridization nodes, e.g., (x_1, x_2, x_3) and (y_1, y_3), as demonstrated by T_2 and T_3, respectively. Notice that resolving a hybridization node into a stack of hybridization nodes does not produce new embedded trees compared with those of the unresolved network.

Stacks of hybridization nodes. Given a hybridization network displaying a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees and containing a node v of in-degree of at least 3, one can generate further networks still displaying $\mathcal{T}$ by dragging some of its reticulation edges upwards resulting in so-called *stack of hybridization nodes.* More precisely, such a stack is a path $(v_1, \ldots, v_n)$, with $n > 1$, of hybridization nodes in which each node v_i is connected through a reticulation edge to v_{i+1} (cf. Fig. 3.2).

3.2 Reduction rules

Given a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees, before applying a method calculating minimum hybridization networks for those trees one can apply two well-known reduction rules, namely the *subtree reduction* and the *cluster reduction*. By applying those two rules the computational complexity of the input trees can be reduced which often significantly improves the practical runtime of the respective method.

Subtree reduction. Let $\mathcal{T}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees, then the subtree reduction transforms all of those trees into a set $\mathcal{T}'$ of rooted binary phylogenetic $\mathcal{X}$-trees by replacing each maximal pendant subtree T' of size ≥ 2 occurring in all trees of $\mathcal{T}$. More precisely, let v be the root of such a maximal pendant subtree T'. Then, in each tree of $\mathcal{T}$, first all nodes that can be reached from v are deleted and afterwards v is labeled by a new taxon $a \notin \mathcal{X}$. Notice that, in order to undo the subtree reduction at a given time, one has to keep track which of these new taxa belongs to which common subtree.

Cluster reduction. Let $\mathcal{T}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees and let $A \subset \mathcal{X}$ be a *cluster* with $A \geq 2$ such that for each tree T_i in $\mathcal{T}$ there exists a specific node v_i with $\mathcal{L}(v_i) = A$. Then, the cluster reduction separates $\mathcal{T}$ into two tree sets $\mathcal{T}|_A$ and $\mathcal{T}_a$, where $\mathcal{T}|_A$ contains each tree $T_i|_A$ and $\mathcal{T}_a$ contains each tree T_i where $T(v_i)$ is replaced by a new taxon a. More precisely, the tree set $\mathcal{T}_a$ is obtained from $\mathcal{T}$ by first deleting from each tree T_i all nodes that can be reached from v_i and then by labeling v_i by a new taxon $a \notin \mathcal{X}$. Notice that, in order to reattach those clusters back together at a given time, one has to keep track which of these new taxa belongs to which common cluster.

3.3 Correctness of the cluster reduction

In the following, we will give a formal proof showing that the cluster reduction is safe for a set $\mathcal{T}$ of *multiple* rooted binary phylogenetic $\mathcal{X}$-trees as noted in Theorem 13.

Theorem 13. *Given a set of rooted binary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$ all containing a common cluster $A \subset \mathcal{X}$, then, $h(\mathcal{T}) = h(\mathcal{T}|_A) + h(\mathcal{T}_a)$.*

3.3.1 Related work

In general, there are two important works dealing with the cluster reduction of rooted binary phylogenetic $\mathcal{X}$-trees.

Baroni, 2006. The well-known work of Baroni *et al.* [9] contains a proof showing that the hybridization number of *two* rooted binary phylogenetic $\mathcal{X}$-trees can be computed by simply summing up the hybridization numbers of its common clusters. More precisely, given two rooted binary phylogenetic $\mathcal{X}$-trees T and T' containing a common cluster $A \subset \mathcal{X}$, then $h(T,T') = h(T|_A, T'|_A) + h(T_a, T'_a)$, where T_a and T'_a refers to the respective input tree in which the common cluster has been replaced by a new taxon a.

Linz, 2008. A more general proof, showing that a similar fact also holds for more than two rooted binary phylogenetic $\mathcal{X}$-trees, is given in the PhD thesis of Linz [40, Theorem 2.5]. This proof, however, in contrast to our definition of the hybridization number h (cf. Eq. 1.2), is based on a different definition, denoted by h', only considering the total number of hybridization nodes of a network. More precisely, given a set of rooted binary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$, then $h'(\mathcal{T}) = h'(\mathcal{T}|_A) + h'(\mathcal{T}_a)$ with

$$h'(\mathcal{T}) = \min\{h'(N) : \text{N is a hybridization network displaying } \mathcal{T}\},$$

where $h'(N)$ just counts the number of hybridization nodes in N. This means, in particular, that, in contrast to the reticulation number $r(N)$ as defined here in this thesis (cf. Eq. 1.1), $h'(N)$ does not take the number of edges that are directed into a hybridization node into account. Consequently, the two values $r(N)$ and $h'(N)$ differ, if the network N provides a hybridization node with in-degree larger than two.

3.3.2 Further definitions

In the following, we will first give some further definitions that are crucial for establishing Theorem 13.

Hybridization networks. Given a hybridization network N on $\mathcal{X}$ and a subset E' of reticulation edges in N, then, by writing $N - E'$ we denote the network that is obtained from N by first deleting E' and then by suppressing each node of both in- and out-degree 1.

Restricted pendant subtrees. Let E' be an edge set referring to a rooted binary phylogenetic $\mathcal{X}'$-tree T' that is displayed in a hybridization network N. Then for a path P connecting two nodes both contained in $N|_{E',\mathcal{X}'}$, we denote by $\mathcal{R}_N(P, E', \mathcal{X}')$ the set of *non-empty restricted pendant subtrees* of each node lying on P. More precisely, each subtree R_i in $\mathcal{R}_N(P, E', \mathcal{X}')$ refers to a non-empty subgraph of $N|_{E',\mathcal{X}'}$ with root $v \notin P$, which is connected through an edge to a node $w \in P$, such that R_i equals $\overline{N|_{E',\mathcal{X}'}(v)}$ (cf. Fig. 3.3(b)).

3.3.3 Proof of Theorem 13

Proof. As defined in Theorem 13, we have that $A \subset \mathcal{X}$. Now, in a first step, we show that

$$h(\mathcal{T}) \leq h(\mathcal{T}|_A) + h(\mathcal{T}_a) \tag{3.1}$$

by contradiction. Let N be a hybridization network displaying $\mathcal{T}$ with minimum hybridization number $h(\mathcal{T})$, and let N_A and N_a be a hybridization network displaying $\mathcal{T}|_A$ with minimum hybridization number $h(\mathcal{T}|_A)$ and $\mathcal{T}_a$ with minimum hybridization number

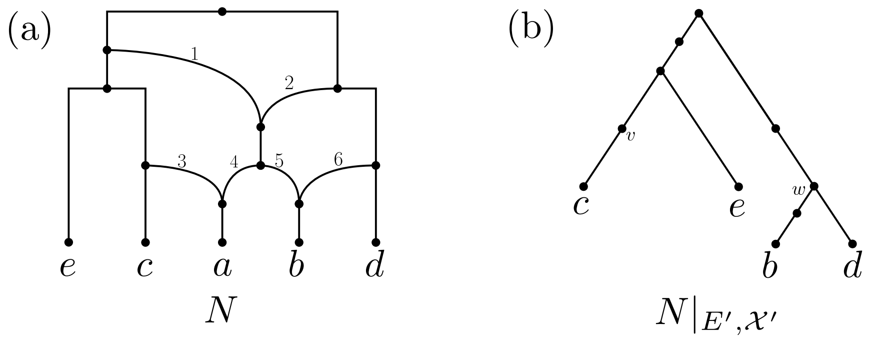


Figure 3.3: **(a)** A hybridization network N with taxa set $\mathcal{X} = \{a, b, c, d, e\}$ whose reticulation edges are consecutively numbered. **(b)** The restricted network $N|_{E',\mathcal{X}'}$ with $E' = \{3, 6, 1\}$ and $\mathcal{X}' = \{b, c, d, e\}$ still containing nodes of both in- and out-degree 1. Let P be the path connecting both nodes v and w in $N|_{E',\mathcal{X}'}$, then $\mathcal{R}_N(P, E', \mathcal{X}')$ consists of the four non-empty restricted pendant subtrees (c), (e), (b), and (d).

$h(\mathcal{T}_a)$, respectively. Moreover, let $N_{A,a}$ be the network that is obtained from N_a by replacing taxon a through N_A. This is done, in particular, by first attaching each in-going edge of the leaf v_a labeled by taxon a to the root of N_A and then by removing label a from v_a. Now, if $h(\mathcal{T}) > h(\mathcal{T}|_A) + h(\mathcal{T}_a)$ holds, then simultaneously $r(N) > r(N_{A,a})$ must hold which is a contradiction to the choice of N. $\lightning$

Next, we will show that

$$h(\mathcal{T}) \geq h(\mathcal{T}|_A) + h(\mathcal{T}_a) \tag{3.2}$$

by discussing several cases.

In a first step, however, we have to establish a new lemma that is crucial for proving this inequation. Given a hybridization network N containing a reticulation edge e, we say that e *can be compensated* if $N - \{e\}$ still displays $\mathcal{T}$. This is the case if and only if N displays the scenario as described in Lemma 14 (cf. Fig. 3.4).

Lemma 14. *Given a hybridization network N displaying a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees. Then, a reticulation edge e in N can be compensated if and only if for each tree T_i in $\mathcal{T}$, whose referring edge set E_i contains e, there exists another edge set $E_i' \neq E_i$ such that the following condition is satisfied. There exist two node-disjoint paths P and P' both connecting two nodes u and w with $e \in P$ and $e \notin P'$ such that $\mathcal{R}_N(P, E_i, \mathcal{X}) = \mathcal{R}_N(P', E_i', \mathcal{X})$.*

Proof. '$\Longleftarrow$': For each tree T_i, whose referring edge set E_i contains e, let $\hat{E}_i = E_i \setminus \{e\} \cup E_H(P')$, where $E_H(P')$ denotes the set of reticulation edges in P'. Then, since $\mathcal{R}_N(P, E_i, \mathcal{X}) = \mathcal{R}_N(P', E_i', \mathcal{X})$, $\hat{E}_i$ refers to T_i and, thus, $N - \{e\}$ still displays T_i.

'$\Longrightarrow$': If e can be compensated, this implies that $N - \{e\}$ still displays $\mathcal{T}$. This means, in particular, that for each edge set E_i containing e and referring to a tree T_i in $\mathcal{T}$, there has to exist a further edge set $E_i' \neq E_i$ not containing e but still referring to T_i. Now, based on the two restricted networks $N|_{E_i,\mathcal{X}}$ and $N|_{E_i',\mathcal{X}}$, we can define two particular paths P and P'.

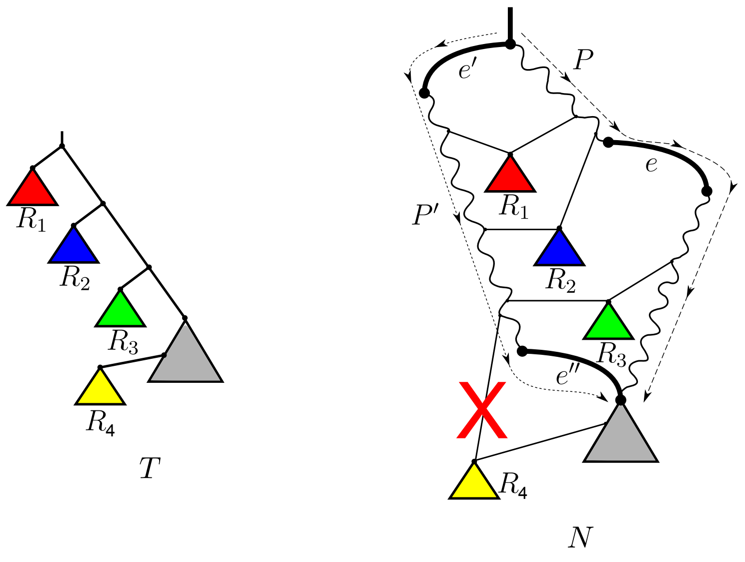


Figure 3.4: Illustration of a scenario compensating the reticulation edge e regarding the embedding of T in N (see proof of Lemma 14). Note that e could not be compensated if R_4 would be a pendant subtree of P'.

Let e' be an edge of N satisfying the following two conditions. First the source node u of e' is part of $N|_{E_i,\mathcal{X}}$ as well as of $N|_{E'_i,\mathcal{X}}$ and its target node is only part of $N|_{E'_i,\mathcal{X}}$ but not of $N|_{E_i,\mathcal{X}}$. Second, there is no other edge in N fulfilling this property and is closer to the root. Similarly, let e'' be an edge of N satisfying the following two conditions. First, the target node w of e'' is part of $N|_{E_i,\mathcal{X}}$ as well as of $N|_{E'_i,\mathcal{X}}$ and its source node is only part of $N|_{E'_i,\mathcal{X}}$ but not of $N|_{E_i,\mathcal{X}}$. Second, there is no other edge in N fulfilling this property and is closer to the root.

Then, there are two specific paths in N running from u to w; one being part of $N|_{E_i,\mathcal{X}}$ (and, thus, containing e), denoted by $P = (u, a_1, \dots, a_k, w)$, and the other one being part of $N|_{E'_i,\mathcal{X}}$ (and, thus, not containing e), denoted by $P' = (u, b_1, \dots, b_{k'}, w)$ (cf. Fig. 3.4). Moreover, as both edges e' and e'' are chosen such there exist no other edges fulfilling the respective properties and are closer to the root, $a_i \neq b_j$ for each node $a_i \in \{a_1, \dots, a_k\}$ and each node $b_j \in (b_1, \dots, b_{k'})$. Additionally, since E_i and E'_i both refer to T_i, $\mathcal{R}_N(P, E_i, \mathcal{X}) = \mathcal{R}_N(P', E'_i, \mathcal{X})$, which finally establishes Theorem 13. □

Now, let each network N, N_A, N_a, and $N_{A,a}$ be as defined above. Moreover, let E_A and E_a be a specific subset of reticulation edges of those contained in the subnetwork corresponding to N_A and N_a, respectively, satisfying the following condition. For each edge e in E_A (resp. E_a), this edge can be reattached to a specific edge of the subnetwork corresponding to N_a (resp. N_A), so that the resulting network $N'_{A,a}$ still displays $\mathcal{T}$ (cf. Fig. 3.5). Additionally, let N'_a and N'_A be the two subgraphs in $N'_{A,a}$ consisting of each element in N_a and N_A, respectively. Now, if $h(\mathcal{T}) < h(\mathcal{T}|_A) + h(\mathcal{T}_a)$ holds, based on E_A and E_a, we have to consider the following four cases (cf. Fig. 3.5).

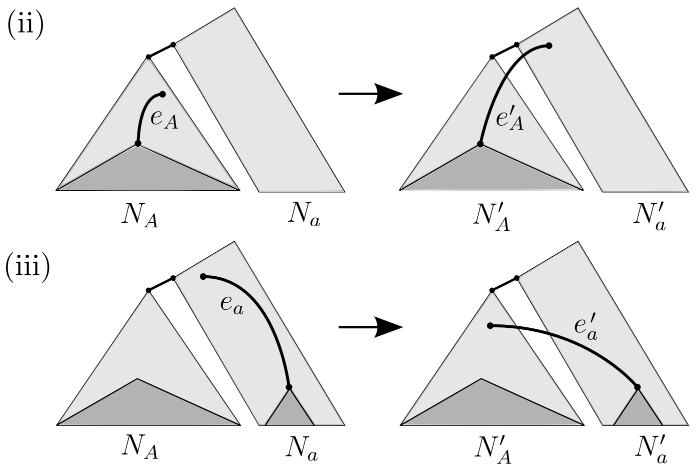


Figure 3.5: Illustration of the scenario referring to Case (ii) and Case (iii).

(i) Let $E_A = \emptyset$ and $E_a = \emptyset$. There exists a set of reticulation edges $E'_{A,a} \neq \emptyset$ in N_A or N_a that can be compensated.

(ii) Let $E_A \neq \emptyset$ and $E_a = \emptyset$. There exists a set of reticulation edges $E'_{A,a} \neq \emptyset$ in $N'_{A,a}$ that can be compensated.

(iii) Let $E_A = \emptyset$ and $E_a \neq \emptyset$. There exists a set of reticulation edges $E'_{A,a} \neq \emptyset$ in $N'_{A,a}$ that can be compensated.

(iv) Let $E_A \neq \emptyset$ and $E_a \neq \emptyset$. There exists a set of reticulation edges $E'_{A,a} \neq \emptyset$ in $N'_{A,a}$ that can be compensated.

In the following, we will show that each scenario, which is described by one of the four cases, cannot occur due to certain circumstances.

Case (i). In this case, either $r(N_A) \neq h(\mathcal{T}|_A)$ or $r(N_a) \neq h(\mathcal{T}_a)$, which is a contradiction to the choice of N_A or N_a, respectively. $\lightning$

Case (ii). Let $N'_{A,a}$ be the network that is obtained from $N_{A,a}$ by reattaching the source nodes of each edge in E_A to the subnetwork corresponding to N_a such that $N'_{A,a}$ still displays $\mathcal{T}$. Now, first notice that from those shifted edges there does not arise a new path whose start- and end-node both lie in N'_a. As a consequence, due to Lemma 14, each edge of N'_a that could be compensated, could be also compensated in the original network $N_{A,a}$, which is a contradiction to the choice of both networks N_a and N_A. $\lightning$
The same argument holds for the subnetwork corresponding to N_A and, thus, in this case the network $N'_{A,a}$ cannot contain any reticulation edges that can be compensated.

Case (iii). The argumentation regarding this case equals the one of Case (ii).

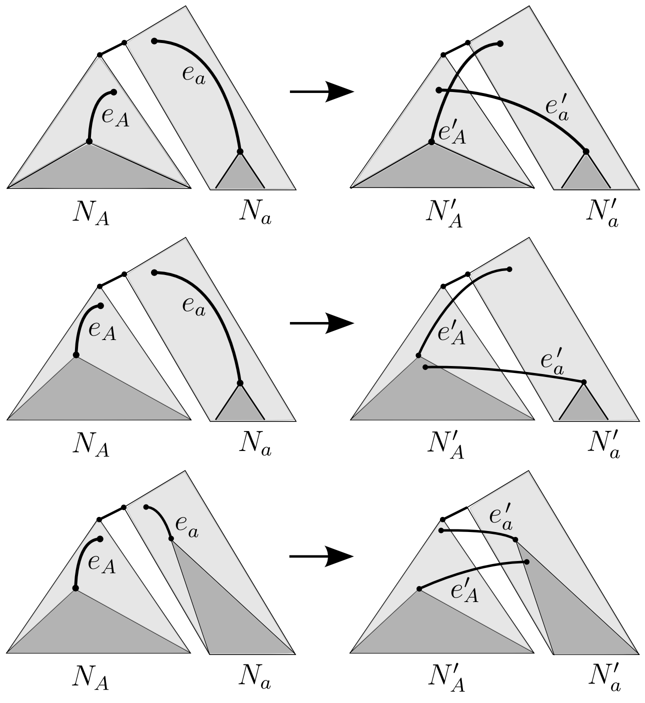


Figure 3.6: Illustration of the scenario referring to Case (iv.i), Case (iv.ii), and Case (iv.iii).

Case (iv). Again, let E'_A and E'_a be the set of edges in $N'_{A,a}$ whose source nodes have been reattached to the subnetwork corresponding N_a and N_A, respectively. Now, there additionally exist three out of four sub-cases that have to be considered here (cf. Fig. 3.6).

(iv.i) Neither a source node of an edge in E'_a is contained in a subnetwork rooted at a target node of an edge in E'_A nor a source node of an edge in E'_A is contained in a subnetwork rooted at a target node of an edge in E'_a.

(iv.ii) There exists a source node of an edge e'_a in E'_a that is contained in a subnetwork rooted at the target node of an edge e'_A in E'_A.

(iv.iii) There exists a source node of an edge e'_A in E'_A that is contained in a subnetwork rooted at the target node of an edge e'_a in E'_a.

(iv.iv) There exists a source node of an edge e'_A in E'_A that is contained in a subnetwork rooted at the target node of an edge e'_a in E'_a and, simultaneously, there exists a source node of an edge e'_A in E'_A that is contained in a subnetwork rooted at the target node of an edge e'_a in E'_a. This directly implies that the graph contains a directed cycle and, thus, does not apply to the definition of hybridization networks. Consequently, this case has not to be considered here.

Case (iv.i). Again, similar to Case (ii), there does not arise a new path whose start- and end-node both lie in the subnetwork corresponding to N_a and N_A, respectively. Thus,

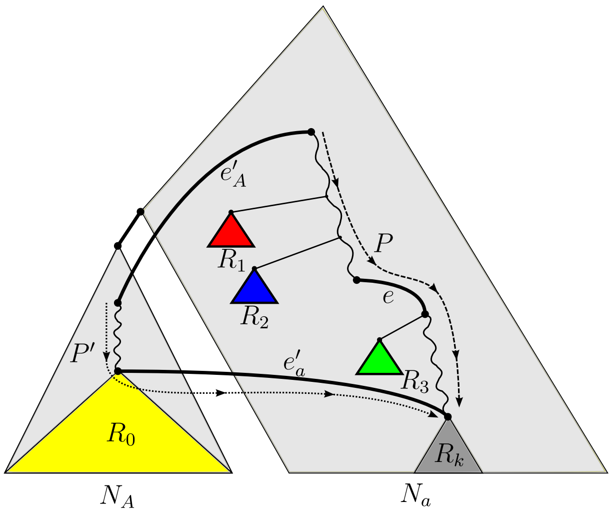


Figure 3.7: An illustration of the scenario concerning Case (iv.ii).

each edge that is contained in this part of the network and could be compensated, could be also compensated in the original network $N_{A,a}$ which is a contradiction to the choice of both networks N_a and N_A. $\lightning$

Case (iv.ii). In this certain case there exists a path leading from a target node of e'_A in E'_A back to N'_a (cf. Fig. 3.7). Thus, potentially, there could exist a reticulation edge e in N'_a such that $N'_{A,a} - \{e\}$ still displays $\mathcal{T}$. More precisely, this would be the case if e'_A and e'_a could compensate a deletion of e.

Now, let E_i be an edge set referring to an input tree T_i and let P be the path of $N'_{A,a}|_{E_i,\mathcal{X}}$ leading from the source node of e'_A to the target node e'_a. Moreover, without loss of generality, we assume that there does not exist a further edge set referring to another input tree T_j with $j \neq i$ containing e. Now, if there would exist an edge set E'_i with $e \notin E'_i$ referring to T_i, this would automatically imply that e could be compensated.

If e is not part of such a path P, e cannot be compensated by the two shifted edges e'_A and e'_a. Otherwise, let $\mathcal{R}_{N'_{A,a}}(P, E_i, \mathcal{X}) = (R_0, \dots, R_k)$ be the ordered set of non-empty pendant subtrees of each node lying on P in which the first restricted subtree R_0 corresponds to $\overline{N'_A|_{\{E_i,\mathcal{X}\}}}$. Now, only if there exists a path P' leading from the target node of e'_A to the target node of e'_a such that $\mathcal{R}_{N'_{A,a}}(P, E_i, \mathcal{X})$ equals $\mathcal{R}_{N'_{A,a}}(P', E_i, \mathcal{X})$, e could be compensated by using e'_A and e'_a. However, as A is a cluster of T_i, in this case R_1 to R_{k-1} may not exist, meaning that $\mathcal{R}_{N'_{A,a}}(P', E_i, \mathcal{X})$ could only consist of the two elements R_0 and R_k. Thus, if e could be compensated, this would directly imply that e could be also compensated in N_a by e_a, which is a contradiction to the choice of N_a. $\lightning$

Case (iv.iii). The argumentation regarding this case equals the one of Case (iv.ii).

Finally, combining both Inequations 3.1 and 3.2 completes the proof of Theorem 13. $\square$

3.4 The algorithm allHNetworks

As demonstrated in the previous chapter, a method computing minimum hybridization networks for two rooted binary phylogenetic trees can be divided into the following two major steps. First *maximum acyclic agreement forests* are calculated by cutting down the input trees in a specific way and then the components of such an agreement forest are again reattached by introducing further reticulation edges in a way that the resulting network displays both input trees. In general, there exists not just one but a large number of minimum hybridization networks. To recognize putative hybridization events, biologists are interested in all of those networks, since given such set of networks one can then test specific hypotheses. Thus, given two incongruent trees, there is a need for two types of algorithms; one calculating all maximum acyclic agreement forests and another one constructing all hybridization networks based on these agreement forests.

While there exist some software packages providing methods for computing minimum hybridization networks for two rooted binary phylogenetic trees on the same set of taxa [17, 35], in this section, we will present an algorithm computing all minimum hybridization networks of a certain type, which we will denote later as relevant networks, for an arbitrary number of rooted binary phylogenetic $\mathcal{X}$-trees. The workflow of this algorithm can be briefly summarized as follows. Starting with one input tree, all other input trees are embedded in sequential ordering into a growing number of networks by adding further reticulation edges each corresponding to a certain component of a maximum acyclic agreement forest. In order to guarantee the computation of relevant networks, it is necessary that each input tree is added to a so far computed network in all possible ways. This implies, in particular, that at the beginning, when adding the second input, say T_2, to the first input tree, say T_1, all relevant networks embedding T_1 and T_2 have to be calculated. Missing one of those networks could mean that a computational path leading to a relevant network embedding the whole set of input trees is lost, and, as a consequence, either not all relevant networks are computed or the resulting output only consists of hybridization networks not providing a minimum hybridization number. Note that, later in this section, we will show that for this purpose it suffices to take only maximum acyclic agreement forests into account.

At this particular time, the only other software that is able to compute minimum hybridization networks for multiple rooted binary phylogenetic $\mathcal{X}$-trees is PIRNv2.0 [72, 73]. A recently conducted simulation study, however, has indicated that an implementation of our algorithm provides the clearly better practical runtime and, additionally, in general PIRNv2.0 is able to output only a small subset of those networks we consider as being relevant [1], which obviously complicates the testing of hypothesis on the reported networks.

This section is organized as follows. In a first step, further definitions are introduced, which are crucial for describing and discussing our algorithm. Next, we give a detailed description of our algorithm ALLHNETWORKS whose correctness is shown in a subsequent section by a detailed formal proof. Finally, we end the presentation of our algorithm by briefly discussing its theoretical worst-case runtime and by giving some concluding remarks.

At the end of this section, we still discuss some techniques that can be applied to improve the practical application of our algorithm.

3.4.1 Further definitions

Here, we give a definition of those networks that are constructed by our algorithm allHNetworks.

Relevant networks. Given a set $\mathcal{T}$ of rooted phylogenetic $\mathcal{X}$-trees and a phylogenetic network N on $\mathcal{X}$, then, we say *N is a relevant network for $\mathcal{T}$*, if N is a hybridization network displaying $\mathcal{T}$ with minimum hybridization number and if N does not contain any stacks of hybridization nodes. Notice that such a network leaves the interpretation of the ordering of the hybridization events adhering to a hybridization node of in-degree larger than or equal to 3 open.

Furthermore, we demand that each relevant network is a binary network not containing any nodes of out-degree larger than 2. Notice that by allowing nonbinary nodes the set of relevant networks usually shrinks, since a nonbinary network can contain multiple binary networks. Moreover, in order to improve its readability, we further demand that all hybridization nodes of a relevant network have out-degree one. Notice that, in order to identify stacks of hybridization nodes, in such networks the out-edges of all hybridization nodes have to be suppressed.

Lastly, just for clarity, given two relevant networks N_1 and N_2 for a set $\mathcal{T}$ of rooted phylogenetic $\mathcal{X}$-trees, we say that N_1 equals N_2 if their graph topologies (disregarding the embedding of $\mathcal{T}$) are isomorphic.

3.4.2 The Algorithm allHNetworks

Given a set of rooted binary phylogenetic $\mathcal{X}$-trees $\mathcal{T} = \{T_1, \ldots, T_n\}$ and a parameter $k \in \mathbb{N}$, our algorithm allHNetworks follows a *branch-and-bound* approach conducting the following major steps. For each order of $\mathcal{T}$, the trees are added sequentially to a set of networks $\mathcal{N}$. In the beginning, $\mathcal{N}$ consists only of one element, which is the first input tree of the ordering. By sequentially adding the other input trees to each so far computed network, the size of $\mathcal{N}$ growths rapidly, since in general an input tree T_i can be added to each network in $\mathcal{N}$ in potential several ways. Each time the reticulation number of a so far extended network exceeds k, the processing of this network can be aborted. This is possible because by adding further input trees the reticulation number of the respective network is never decreased.

Given a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees, based on two different objectives, our algorithm provides two different abort criteria:

1. **Objective:** Computation of the hybridization number of $\mathcal{T}$.
   Abort criterion: As soon as *one* hybridization network with hybridization number

k is computed and each search after hybridization networks providing a hybridization number less than k has failed.

2. **Objective:** Computation of all relevant networks for $\mathcal{T}$.
   Abort criterion: As soon as *all* hybridization networks with hybridization number k are computed and each search after hybridization networks providing a hybridization number less than k has failed.

For the computation of a minimum hybridization network, parameter k is set to an initial value and is increased by one if a network displaying $\mathcal{T}$ with hybridization number smaller than or equal to k could not be computed so far. At the beginning, k can be either simply set to 0 or to a lower bound, e.g.,

$$max\{R(T_i, T_j) : i \neq j\}.$$

A more sophisticated method for the computation of such a lower bound is described in the work of Wu [72]. In practice, however, the lower bound does not significantly improve the runtime, since the required steps for those k's that can be skipped at the beginning are usually of rather low computational complexity.

3.4.2.1 Inserting Trees into Networks

Given a hybridization network N, we say that a tree T is displayed in N, if there exists a set of reticulation edges E such that $\overline{N|_{E,\mathcal{X}}}$ equals T (cf. Sec. 3.4.1). This implies, if such a subset does not exist, we have to insert new reticulation edges for displaying T in N. Given an edge set E' referring to an embedded tree T' that is already displayed in N, those edges can be derived from each component of an agreement forest for T' and T. The here presented algorithm is based on the observation, that, in order to compute all relevant networks, it suffices to take only maximum acyclic agreement forests into account (cf. Sec 3.4.4).

Hence, we can summarize the basic steps that are necessary for adding an input tree T to a so far computed network N as follows.

1. Choose an edge set E' referring to an embedded tree T' of N by selecting precisely one in-edge of each hybridization node.

2. First compute a maximum acyclic agreement forest $\mathcal{F}$ for the two trees T' and T and then choose an acyclic ordering $\Pi_{\mathcal{F}}$ of $\mathcal{F}$.

3. Based on $\Pi_{\mathcal{F}}$, for each component of $\mathcal{F}$, except F_ρ, create a valid pair of source and target nodes (as defined later) such that, by connecting each node pair, T is embedded in the resulting network. Notice that this step will be discussed separately in the upcoming section.

It is easy to see, that the resulting network depends on the chosen edge set E' referring to the embedded tree T', which is the case because different embedded trees lead to different maximum acyclic agreement forests which consequently lead to different reticulation edges that are necessary for the embedding of T. Thus, to guarantee the computation of all relevant networks, all three steps have to be conducted for each edge set referring to an embedded tree in N. Note that, given a network containing r hybridization nodes, this network can contain up to 2^r different embedded trees. Moreover, all maximum acyclic agreement forests of the chosen embedded tree T' and the current input tree T have to be taken into account, which can be done by applying the algorithm allMAAFs [57] previously presented in Section 2.3 (or one of its modification presented in Section 2.4).

The insertion of components of a maximum acyclic agreement forest to a so far computed network is not a trivial step, since, usually, depending on other so far existing reticulation edges, there exist several potential ways of how T can be inserted with the help of those components. Thus, this step will be discussed separately in the following section.

3.4.2.2 Inserting Components into Networks

Given an ordering of rooted phylogenetic $\mathcal{X}$-trees, say $(T_1, T_2, \ldots, T_n)$, and a network N displaying each tree T_j with $1 \leq j < i \leq n$ together with an edge set E' referring to some embedded tree T' of N, we can add T_i to N by inserting further reticulation edges each corresponding to a specific component of a maximum acyclic agreement forest $\mathcal{F}$ for T' and T_i. Consequently, for each component a specific target and source node in N has to be determined. Since different source and target nodes can lead to topologically different networks containing different sets of embedded trees, in order to obtain all relevant networks, we have take all valid combinations of source and target nodes for each component of $\mathcal{F}$ into account. More precisely, we consider a pair (s, t) of source and target nodes as being *valid*, if s cannot be reached from t. Furthermore, we have to consider each possible acyclic ordering of $\mathcal{F}$.

Hence, we can summarize all important steps for inserting components of a maximum acyclic agreement forest $\mathcal{F}$ into a network N as follows.

1. Choose an acyclic ordering $(F_\rho, F_1, \ldots, F_k)$ of $\mathcal{F}$.

2. Add each component F_j of this ordering, except F_ρ, sequentially to N by inserting a new reticulation edge connecting a certain source and target node.

The output of these two steps is usually a large number of new networks, since, in general, there exist several pairs of source and target nodes enabling an embedding of T_i. Whereas all acyclic orderings of $\mathcal{F}$ can be simply computed with the help of the directed graph $\mathrm{AG}(T', T_i, \mathcal{F})$ (cf. Sec. 1.4.1), the second step inserting its components is quite more sophisticated. We will describe the way of adding a component F_j of an acyclic ordering to a so far computed network N by first describing the computation of source and target nodes and then, based on these two nodes, by describing the way new reticulation edges are generated.

I Computation of target and source nodes. The set of source and target nodes corresponding to a component F_j in $\mathcal{F}$ is described in Step I.I–I.III. Therefor, let

$$\mathcal{F}' = \{F_\rho, F_1, \dots, F_{j-1}\} \subset \mathcal{F} = \{F_\rho, F_1, \dots, F_k\}$$

be the set of components that has been added so far. Note that, since N is initialized with F_ρ, at the beginning $\mathcal{L}(\mathcal{F}')$ equals $\mathcal{L}(F_\rho)$ and the first component that is added is F_1.

I.I Computation of target nodes. The set $\mathcal{V}_t$ of target nodes contains all nodes v with $\overline{N|_{E',\mathcal{L}(\mathcal{F}')\cup\mathcal{L}(F_j)}}(v)$ isomorphic to $T_i|_{\mathcal{L}(F_j)}$. Due to the restriction of the network to $\mathcal{L}(\mathcal{F}')$, this set usually contains more than one node. Moreover, since we are only interested in relevant networks, we omit those target nodes that are source nodes of reticulation edges. This is a necessary step preventing the computation of networks containing stacks of hybridization nodes (cf. Sec. 3.4.1).

I.II Computation of source nodes of *Type A*. For each edge set E_i referring to the embedded tree $T_i|_{\mathcal{L}(\mathcal{F}')}$ in N, the set $\mathcal{V}_s^A$ of source nodes of *Type A* contains all nodes v with $\overline{N|_{E_i,\mathcal{L}(\mathcal{F}')}}(v)$ isomorphic to $T_i|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$, where v_{sib} denotes the sibling of the node v' with $\mathcal{L}(v') = \mathcal{L}(F_j)$ in $T_i|_{\mathcal{L}(\mathcal{F}')\cup\mathcal{L}(F_j)}$. Note that, due to the restriction of the network to $\mathcal{L}(\mathcal{F}')$, this set usually consists of more than one node. However, as we want to construct networks in which each hybridization node has out-degree one, we disregard those nodes having more than one in-edge.

I.III Computation of source nodes of *Type B*. The set $\mathcal{V}_s^B$ of source nodes of *Type B* is computed such that it contains each node v of a subtree, whose root is a sibling of a node in $\mathcal{V}_s^A$ not containing any leaves labeled by a taxon of $\mathcal{L}(\mathcal{F}')$. Moreover, its leaf set $\mathcal{L}(v)$ has to consist only of those subsets representing the total taxa set $\mathcal{L}(F)$ of a component F in $\mathcal{F}$, which means that v must not be part of a subtree corresponding to a component that is added afterwards. However, as we want to construct networks in which each hybridization node has out-degree one, we disregard those nodes having more than one in-edge.

For a better understanding, the definitions of source and target nodes are illustrated in Figure 3.8.

Remark. Regarding two components F_p and F_q of an acyclic ordering of $\mathcal{F}$ with $p < q$ and $\mathcal{F}^* = \{F_\rho, F_1, \dots, F_q\}$, it might be the case that both roots of $T_i|_{\mathcal{L}(\mathcal{F}^*)}(v_p)$ and $T_i|_{\mathcal{L}(\mathcal{F}^*)}(v_q)$ are siblings in $T_i|_{\mathcal{L}(\mathcal{F}^*)}$, where v_p and v_q denotes the lowest common ancestor of $\mathcal{L}(F_p)$ and $\mathcal{L}(F_q)$ in $T_i|_{\mathcal{L}(\mathcal{F}^*)}$. In this case, F_p could be either added before F_q or vice versa as both variants are acyclic orderings of $\mathcal{F}$. If F_p is inserted before F_q, a node whose leaf set corresponds to $\mathcal{L}(F_p)$ in $N|_{T_i,\mathcal{L}(\mathcal{F}^*)}$ acts as source node when adding F_q to

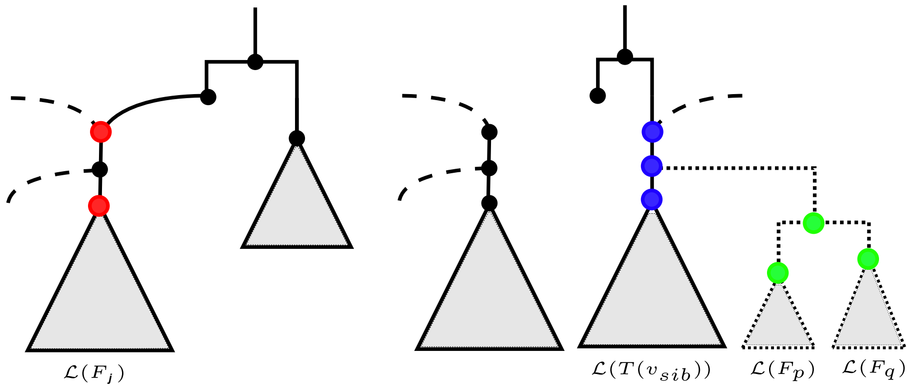


Figure 3.8: An illustration of the definitions of target (left) and source nodes (right) for a component F_j (with $j < p, q$) in which red nodes correspond to target nodes, blue nodes to source nodes of *Type A*, and green nodes to source nodes of *Type B*. Moreover, dashed edges and dotted edges are those edges that are disregarded when considering the restricted network in terms of the chosen embedded tree and the taxa set of the so far added components, respectively.

N. Similarly, by adding F_q before F_p this happens the other way round which leads to a topologically different network. This implies that, in order to receive all relevant networks displaying T_i, we have to consider different acyclic orderings of a maximum acyclic agreement forest.

II Adding new reticulation edges. Now, given a valid pair (s, t) of source and target nodes, a new reticulation edge is inserted as follows (cf. Fig. 3.9).

1. First, the in-edge e of s is split by inserting a new node s', i.e., $e = (p, s)$ is first deleted and then two new edges (p, s') ans (s', s) are inserted. Second, if the parent of t has in-degree one, the in-edge of t is split two times in the same way by inserting two nodes t' and t''. Let t' be the parent of t after splitting its in-edge. In this case, notice that t' is necessary to receive only hybridization nodes of out-degree one and t'' is necessary to provide an attaching point for further reticulation edges as discussed below. Otherwise, if t has an in-degree of at least two, t' is set to t, which prevents the computation of networks containing stacks of hybridization nodes.

2. Now, the two nodes, s' and t', are connected through a path P consisting of two edges. As we do not allow nodes of in-degree larger than one as source nodes, this provides an attaching point for further reticulation edges within already inserted reticulation edges. Notice that, as direct consequence, in each completely processed network, in which all input trees have been inserted so far, one still has to suppress the source nodes of all reticulation edges as these nodes have both in- and out-degree one.

In order to compute all relevant networks, one has to generate for each valid pair (s, t) of source and target nodes a new network $\hat{N}$. This is necessary, since each of those networks contains different sets of embedded trees which can then be used for the insertion of further input trees and, thus, can initiate new computational paths leading to relevant

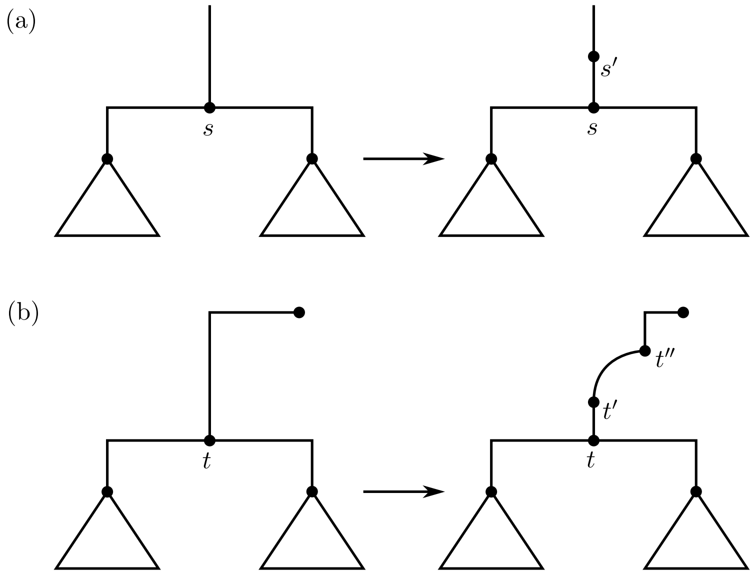


Figure 3.9: Generating a source node **(a)** and a target node **(b)** for adding a new reticulation edge as described in Step II.

networks.

For a better understanding, in Figure 3.10 we illustrate the insertion of an input tree into a so far computed network.

3.4.2.3 Combinatorial complexity

We finish the description of the algorithm by giving an idea of its combinatorial complexity. Given a set of rooted binary phylogenetic $\mathcal{X}$-trees $\mathcal{T}$, in order to guarantee the computation of all relevant networks for $\mathcal{T}$, one has to consider the following combinations.

(1) Take all possible orderings of $\mathcal{T}$ into account.

(2) When adding a tree T_i to a so far computed network N, each possible tree T' that is displayed by N has to be considered.

(3) When processing a so far computed network N by adding a tree T_i based on an a tree T' that is displayed by N, take all acyclic orderings of each maximum acyclic agreement forest for T_i and T' into account.

(4) When adding a certain component of a maximum acyclic agreement forest for T_i and T' to a so far computed network, consider all valid pairs of source and target nodes.

Missing one those combinatorial elements could imply that a computational path leading to a relevant network is not visited. As a direct consequence, possibly either not all relevant networks are computed or the output consists only of those hybridization networks not providing a minimum hybridization number.

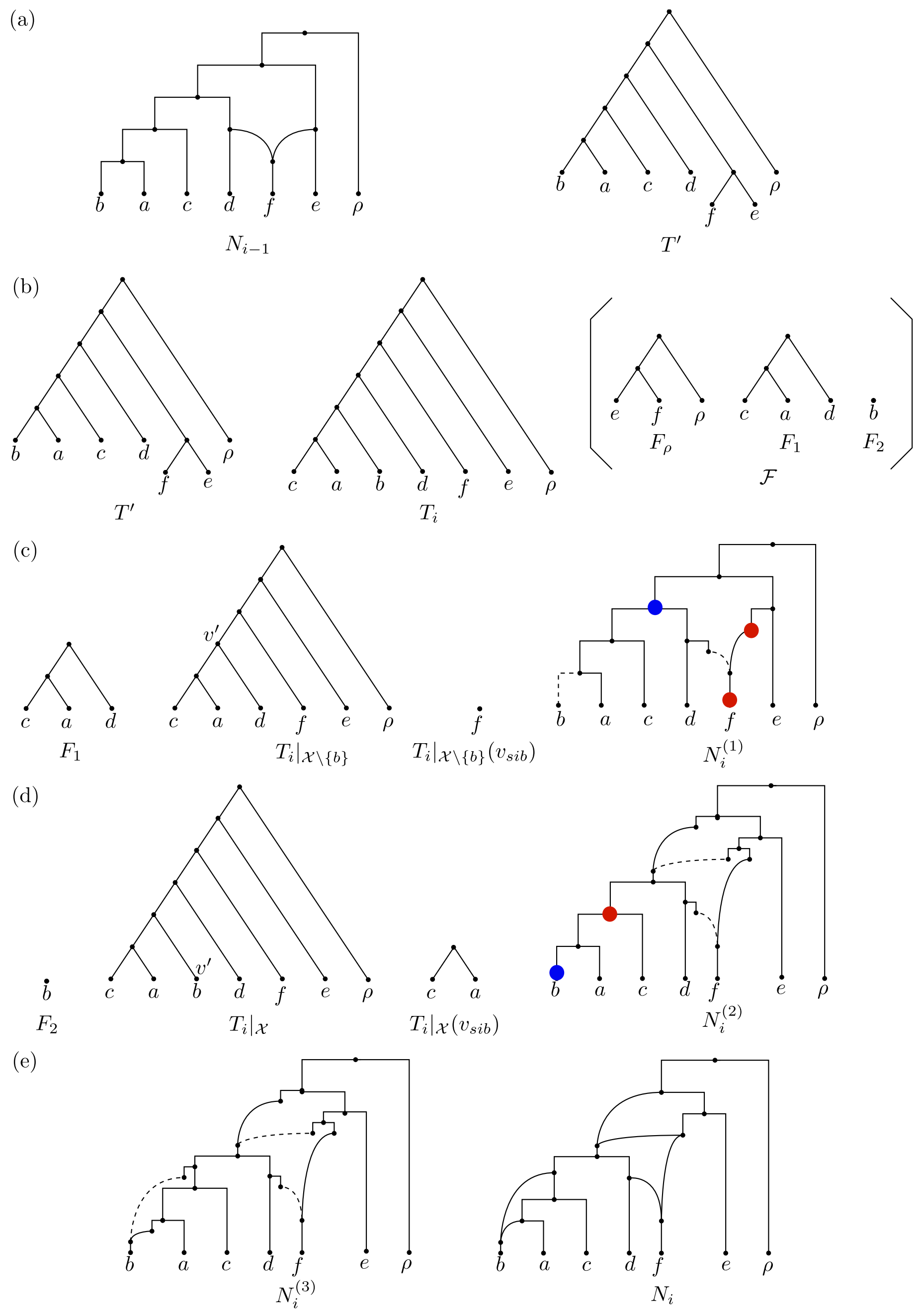

(a)
b a c d f e ρ
N_{i-1}
b a c d ρ f e
T'
(b)
b a c d ρ f e
T'
c a b d f e ρ
T_i
e f ρ c a d b
F_ρ F_1 F_2
$\mathcal{F}$
(c)
c a d
F_1
v'
c a d f e ρ
$T_i|_{\mathcal{X}\setminus\{b\}}$
f
$T_i|_{\mathcal{X}\setminus\{b\}}(v_{sib})$
b a c d f e ρ
$N_i^{(1)}$
(d)
b
F_2
v'
c a b d f e ρ
$T_i|_{\mathcal{X}}$
c a
$T_i|_{\mathcal{X}}(v_{sib})$
b a c d f e ρ
$N_i^{(2)}$
(e)
b a c d f e ρ
$N_i^{(3)}$
b a c d f e ρ
N_i

Figure 3.10: An illustration of how an input tree T_i is inserted into a network N_{i-1}. **(a)** The network N_{i-1} together with an embedded tree T'. **(b)** The input tree T_i, which will be embedded into N_{i-1} by inserting the maximum acyclic agreement forest $\mathcal{F}$ of T_i and T' consisting of three components F_ρ, F_1, and F_2. **(c,d)** The important elements that have to be considered during the insertion of both components F_1 and F_2. Blue dots correspond to source nodes and red nodes to target nodes. Note that, regarding $N_i^{(1)}$, there is only one valid pair of source and target nodes. **(e)** The resulting network N_i, which is obtained from $N_i^{(3)}$ by suppressing each node of both in- and out-degree 1.

3.4.2.4 Pseudocode of allHNetworks

We end this section by giving a pseudocode summarizing all important steps of the algorithm allHNetworks described in the previous section. Some of those steps are denoted by a roman numeral that refers to the equally marked part of Section 3.4.2.2.

Algorithm 13: allHNetworks($\mathcal{T}$)

```
Input: Set $\mathcal{T}$ of rooted phylogenetic $\mathcal{X}$-trees
Output: All topologically different hybridization networks $\mathcal{N}$ with minimum hybridization number
for $k = 1, \dots$ do
    $\mathcal{N} = \emptyset$;
    foreach ordering $\pi$ of $\mathcal{T}$ do
        $T_1 = \pi(1)$;
        $\mathcal{N} = \{T_1\}$;
        for $i = 2$ to $n$ do
            $T_i = \pi(i)$;
            $\mathcal{N}' = \emptyset$;
            foreach $N \in \mathcal{N}$ do
                foreach $T'$ displayed in $N$ do
                    foreach maximum acyclic agreement forest $\mathcal{F}$ for $T'$ and $T_i$ do
                        foreach acyclic ordering $(F_\rho, F_1 \dots, F_m)$ of $\mathcal{F}$ do
                            $\mathcal{N}'' = \{N\}$;
                            for $j = 1$ to $m$ do
                                $\mathcal{N}''' = \emptyset$;
                                foreach $N'' \in \mathcal{N}''$ do
                                    I.I Compute all target nodes $\mathcal{V}_t$ of $F_j$ in $N''$;
                                    I.II Compute all source nodes $\mathcal{V}_a$ of Type A of $F_j$ in $N''$;
                                    I.III Compute all source nodes $\mathcal{V}_b$ of Type B of $F_j$ in $N''$;
                                    $\mathcal{V}_s = \mathcal{V}_a \cup \mathcal{V}_b$;
                                    foreach $(s,t) \in \mathcal{V}_s \times \mathcal{V}_t : s \notin N(t)$ do
                                        $N''' = N''$;
                                        II Insert reticulation edge $(s,t)$ in $N'''$;
                                        $\mathcal{N}''' = \mathcal{N}''' \cup \{N'''\}$;
                                $\mathcal{N}'' = \mathcal{N}'''$
                            foreach $N'' \in \mathcal{N}''$ do
                                if $R(N'') < k$ then
                                    $\mathcal{N}' = \mathcal{N}' \cup \{N''\}$;
            $\mathcal{N} = \mathcal{N}'$;
    if $\mathcal{N} \neq \emptyset$ then
        return $\mathcal{N}$;
```

3.4.3 Use case

In the following, we give a demonstration of the algorithm allHNetworks by presenting a use case for three input trees with taxa set $\mathcal{X} = \{\text{rho}, 1, 2, \ldots, 10\}$. Each of the following Figures 3.11–3.16 and Tables 3.1, 3.2 refers to a particular substep of the algorithm, which is discussed in the corresponding captions.

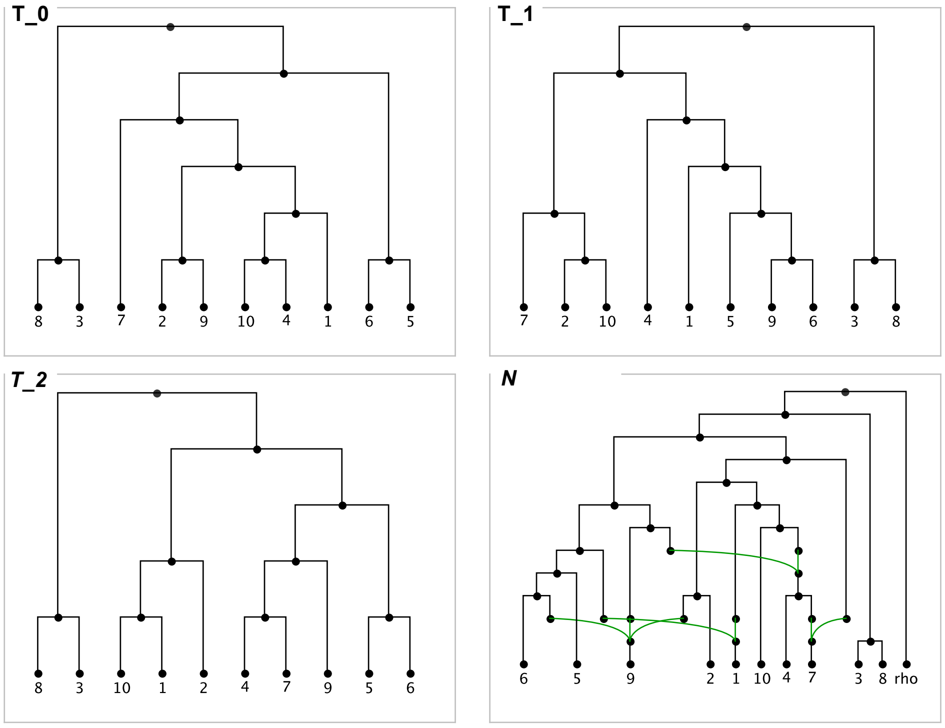


Figure 3.11: The Figure shows the input set consisting of three rooted binary phylogenetic $\mathcal{X}$-trees, namely T_0, T_1, and T_2, with $\mathcal{X} = \{\text{rho}, 1, 2, \ldots, 10\}$. The minimum hybridization network N is one out five relevant networks for those input trees whose computation is now demonstrated step by step. The network is computed by applying the algorithm to the ordering (T_1, T_2, T_0). Consequently, in a first step, T_2 has to be inserted into T_1.

Table 3.1: The computation of all pairs of source and target nodes based on T_2 given T_1 and the components depicted in Figure 3.12. Notice that the notation refers to the one introduced in Section 3.4.2.2.

j	F_j	$\mathcal{L}(\mathcal{F}')$	$T_1\|_{\mathcal{L}(F_j)}$	$\mathcal{V}_t$	$T_1\|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$	$\mathcal{V}_a$	$\mathcal{V}_b$
1	(7);	$\mathcal{X} \setminus \{7, 9, 1\}$	(7);	7	(4);	4	-
2	(9);	$\mathcal{X} \setminus \{9, 1\}$	(9);	9	(5,6);	16	-
3	(1);	$\mathcal{X} \setminus \{1\}$	(1);	1	(10);	10	-

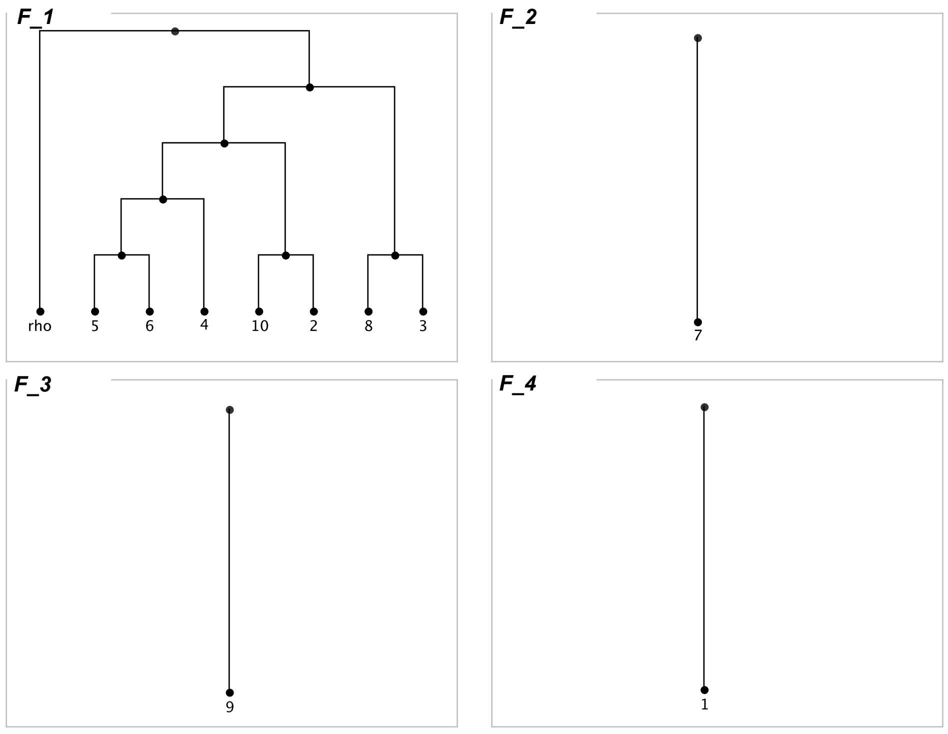


Figure 3.12: At the beginning, the first network as well as its embedded tree both refer to T_1. Hence, in a first step, the maximum acyclic agreement forest $\{F_1, F_2, F_3, F_4\}$ for T_1 and T_2 is computed whose components are used in a subsequent step to receive N_0 displaying both trees.

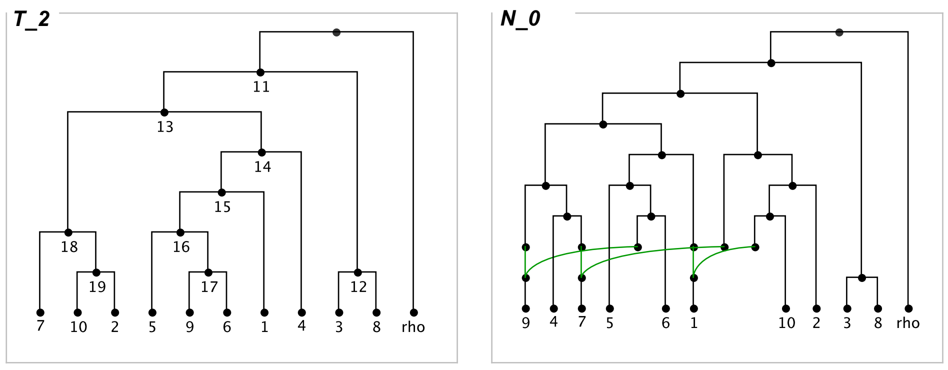


Figure 3.13: Network N_0 is computed by adding the components F_2, F_3, and F_4 (cf. Fig. 3.12) sequentially in acyclic order to T_2. This is done by first computing pairs of target and source nodes (cf. Step I.I–III of the algorithm ALLHNETWORKS) and then by inserting new reticulation edges for each of those pairs (cf. Step II of the algorithm ALLHNETWORKS). Table 3.1 indicates the computation of these source and target nodes by referring to the notation used in Section 3.4.2.2.

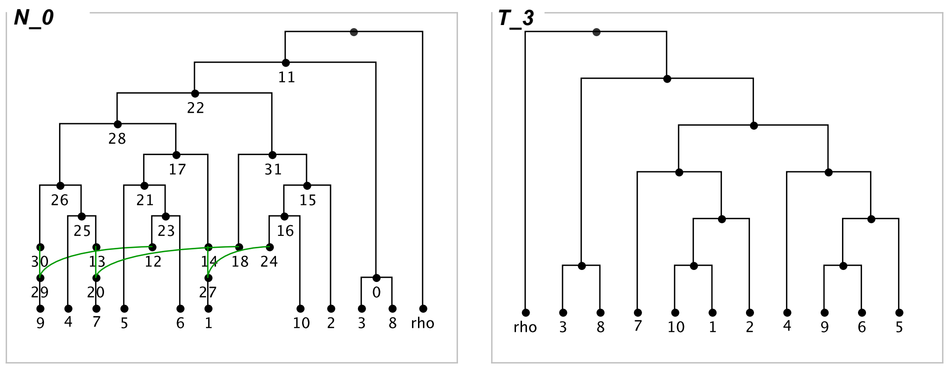


Figure 3.14: Next, all embedded binary phylogenetic $\mathcal{X}$-trees are extracted from N_0 each by selecting one in-edge of each hybridization node. One of those trees is T_3, which is received by selecting the edges $(12, 29)$, $(18, 20)$, and $(24, 27)$.

Table 3.2: The computation of all valid pairs of source and target nodes based on T_2 given N_0, the extracted tree T_3 and the components depicted in Figure 3.15. Note that the notation refers to the one introduced in Section 3.4.2.2.

i	F_j	$\mathcal{L}(\mathcal{F}')$	$T_0\|_{\mathcal{L}(F_j)}$	$\mathcal{V}_t$	$T_0\|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$	$\mathcal{V}_a$	$\mathcal{V}_b$
1	(4);	$\mathcal{X} \setminus \{4, 9\}$	(4);	4, 25	(10);	10	-
2	(9);	$\mathcal{X} \setminus \{9\}$	(9);	9	(2);	2	-

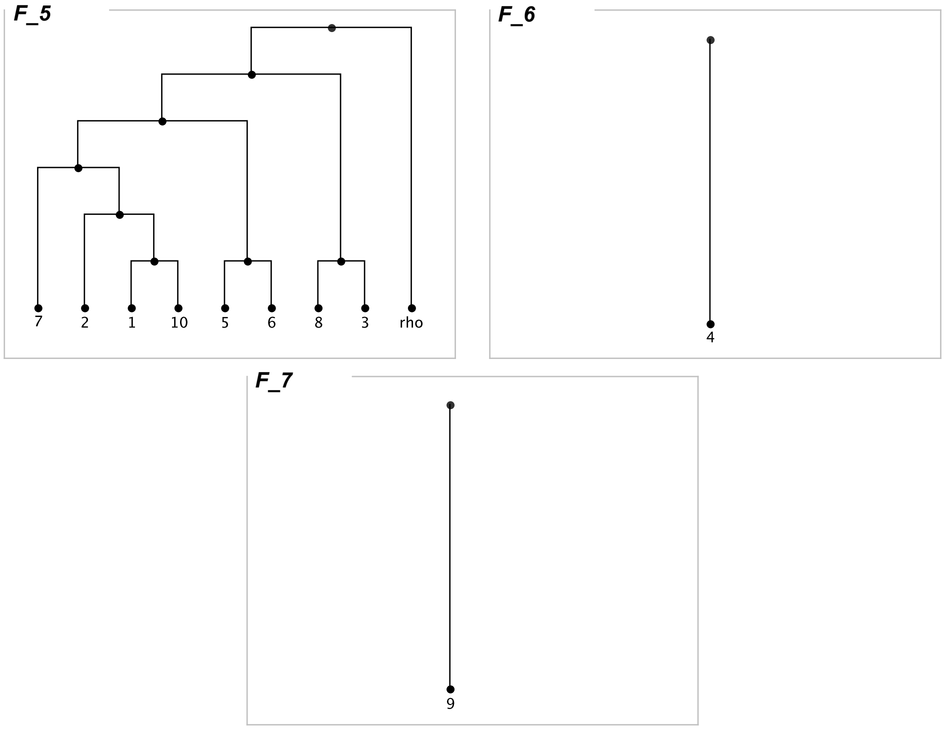


Figure 3.15: Now, again a maximum acyclic agreement forest $\{F_5, F_6, F_7\}$ for the extracted tree T_3 and the input tree T_0 is computed whose components are used in a subsequent step to receive the final network N displaying all input trees.

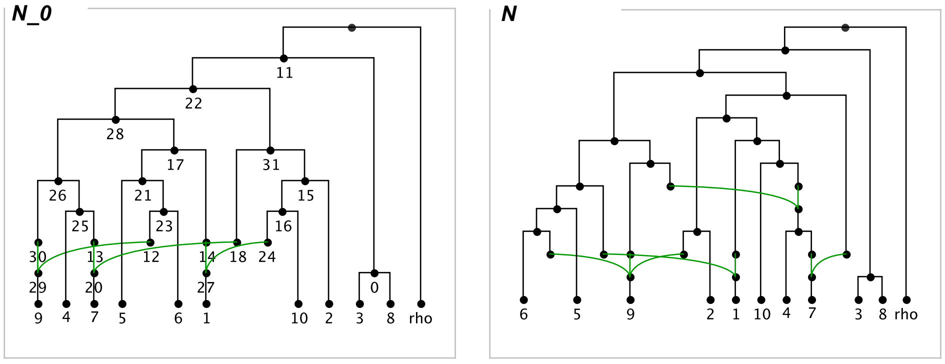


Figure 3.16: The relevant network N is computed by adding the components F_6 and F_7 (cf. Fig. 3.15) sequentially in acyclic order to N_0. This is done by first computing target and source nodes (cf. Step I.I–III of the algorithm ALLHNETWORKS) and then by inserting new reticulation edges for each pair of source and target nodes (cf. Step II of the algorithm ALLHNETWORKS). Table 3.2 indicates the computation of all pairs of source and target nodes by referring to the notation used in Section 3.4.2.2.

3.4.4 Proof of correctness

Here, we prove the main result of this section, namely that for a set of rooted binary phylogenetic $\mathcal{X}$-trees the algorithm ALLHNETWORKS calculates all relevant networks as defined in Section 3.4.1.

Theorem 15. *Given a set of binary rooted phylogenetic $\mathcal{X}$-trees $\mathcal{T}$, by calling*

ALLHNETWORKS($\mathcal{T}$)

all relevant networks for $\mathcal{T}$ are calculated.

For clarity, here we consider two relevant networks N_1 and N_2 as being different if both graph topologies of N_1 and N_2 (disregarding the embedding of $\mathcal{T}$) differ.

Proof. The proof of Theorem 15 is based on the following three Lemmas 16–18. Here, we first show that the concept of acyclic agreement forest suffices to generate all of the desired networks. Next, we argue that for inserting acyclic agreement forests the algorithm takes all necessary pairs of source and target nodes into account. Finally, we prove that by taking all orderings of the input trees into account it suffices to focus only on acyclic agreement forests of minimum size, i.e., maximum acyclic agreement forests. Before entering the first lemma, however, we first have to introduce some further notations.

Let N and N' be two rooted phylogenetic networks on $\mathcal{X}$. Then, we say that N' *is displayed by* N, shortly denoted by $N \supset N'$, if N' can be obtained from N by first deleting some of its reticulation edges and then by suppressing all nodes of both in- and out-degree 1.

Similarly, let N be a hybridization network on $\mathcal{X}$ displaying two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2. Now, given an acyclic agreement forest $\mathcal{F}$ for those two trees, we say that N *displays* $\mathcal{F}$, shortly denoted by $N \supset \mathcal{F}$, if we can obtain $\mathcal{F}$ from N as follows. Regarding N, let E_1 and E_2 be two sets of reticulation edges referring to T_1 and T_2, respectively. First in N all reticulation edges are deleted that are not contained in $E_1 \cap E_2$ and then all nodes of both in- and out-degree 1 are suppressed. Notice that, by deleting those edges the network is disconnected into a set of disjoint trees each corresponding to exactly one of the components in $\mathcal{F}$.

Let $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees and let N be a hybridization network displaying $\mathcal{T}$. Moreover, let E_i be an edge set in N referring to a tree $T_i \in T$. Then, for a tree $T_k \in \mathcal{T}$ the edge set $\hat{E}^{(k)}$ refers to the edge set

$$E_k \setminus E_1 \cup E_2 \cdots \cup E_{k-1} \cup E_{k+1} \cdots \cup E_n.$$

This means, in particular, that $\hat{E}^{(k)}$ consists of those reticulation edges that are only necessary for displaying T_k and none of the other trees in $\mathcal{T}$.

Next, let N be a phylogenetic network and let E' be a subset of its reticulation edges. Then, by writing $N \ominus E'$ we refer to the network that is obtained from N by first deleting each edge in E' and then by suppressing each node of both in- and out-degree 1.

Lemma 16. *Let $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ with $n > 1$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees and let N be a hybridization network displaying $\mathcal{T}$. Moreover, let E_i with $i \in [1:n]$ be an edge set referring to the respective tree $T_i \in \mathcal{T}$ in N. Then, for each tree T_k in $\mathcal{T}$ the network $N \ominus \hat{E}^{(k)}$ contains an embedded tree T' such that N contains an acyclic agreement forest $\mathcal{F}$ for T' and T_k, i.e., $N \supset \mathcal{F}$ holds.*

Proof. Let E_k be an edge set in N referring to T_k and, based on E_k, let $\hat{E}^{(k)}$ be the edge set in N as defined above. Moreover, let $\mathcal{F}$ be a set of subtrees that is derived from N as follows. First, the network N' is computed by removing each edge e with $e \notin E_k$. Next, each edge e in N' with $e \in \hat{E}^{(k)}$ is removed and, finally, each node of both in- and out-degree 1 is suppressed. As the tree that can be derived from N' by suppressing its nodes of both in- and out-degree one corresponds to T_k, it is easy to see that $\mathcal{F}$ consists of common subtrees of T_k. Furthermore, as $\mathcal{F}$ is obtained from N' by cutting some of its edges, this implies that $\mathcal{F}$ is a set of node-disjoint subtrees in T_k.

Next, we will show how one can derive an edge set E' referring to a phylogenetic $\mathcal{X}$-tree T' displayed in N so that $\mathcal{F}$ is an agreement forest for T' and T_k. Therefor, we say a reticulation edge e of N is of *Type A*, if $e \in E_k \setminus \hat{E}^{(k)}$, and of *Type B*, if $e \notin E_k$. Now, let E' be a subset of reticulation edges that is obtained from N by visiting all of its reticulation nodes as follows. If, for a reticulation node, there exists an in-edge e of *Type A*, this edge is selected, otherwise, an arbitrary in-edge of *Type B* is selected. As each edge in E' is also contained in $N \ominus \hat{E}^{(k)}$, it is easy to see that T' is also displayed by $N \ominus \hat{E}^{(k)}$.

Now, let $\hat{E}'$ be the set of reticulation edges that is removed from N by restricting N on E' and let $E_{\mathcal{F}}$ be the set of reticulation edges that has been removed from N in order to obtain $\mathcal{F}$. Then, the target of each reticulation edge in $\hat{E}' \setminus E_{\mathcal{F}}$ is a reticulation node providing an in-edge of $E^{(k)}$, which has been removed from N' (and, thus, actually from T_k) in order to obtain $\mathcal{F}$. As a direct consequence, each component in $\mathcal{F}$ can be also obtained from T' by cutting some of its edges, which directly implies that $\mathcal{F}$ is a set of node-disjoint subtrees in T'.

As a direct consequence, $\mathcal{F}$ is an agreement forest for both trees T_k and T'. Moreover, since N is a hybridization network and, consequently, does not contain any directed cycles, $\mathcal{F}$ has to be an acyclic agreement forest for both trees. □

This means that for inserting further rooted binary phylogenetic $\mathcal{X}$-trees into so far computed networks it is sufficient to focus only on acyclic agreement forests. Notice, however, that the insertion of further reticulation edges based on such agreement forests can be conducted in several ways. Thus, in order to calculate all relevant networks, our algorithm has to guarantee that all of those possibilities are exploited, which is stated by the following lemma.

Lemma 17. *Let $\mathcal{T} = \{T_1, T_2, \dots, T_i\}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees, N_{i-1} be a network displaying each tree in $\mathcal{T} \setminus \{T_i\}$, E' be an edge set referring to some embedded tree T' of N_{i-1}, and $\mathcal{F}$ be an acyclic agreement forest for T' and T_i. Then, the algorithm* ALLHNETWORKS *inserts $\mathcal{F}$ into N_{i-1} so that each hybridization network N_i displaying $\mathcal{T}$ with $N_i \supset \mathcal{F}$ and $N_i \supset N_{i-1}$ is calculated.*

Proof. Given an acyclic ordering $(F_\rho, F_1, \dots, F_k)$ of the maximum acyclic agreement forest $\mathcal{F}$ for the two trees T' and T_i, then, when inserting each component F_j in ascending order, beginning with F_1, all possible target and source nodes in N_{i-1} are taken into account. More precisely, let $\mathcal{X}' = \mathcal{L}(\mathcal{F}')$ with $\mathcal{F}' = \{F_\rho, F_1, \dots, F_{j-1}\}$ and let v_{sib} be the sibling of a node v with $\mathcal{L}(v) = \mathcal{L}(F_j)$ in $T_i|_{\mathcal{L}(\mathcal{F}')\cup\mathcal{L}(F_j)}$.

- Since for each target node $w \in \mathcal{V}_t$ the two trees $\overline{N_{i-1}|_{E',\mathcal{L}(F_j)}(w)}$ and $T_i|_{\mathcal{L}(F_j)}$, with E' referring to T', are isomorphic, each node w' not in $\mathcal{V}_t$ automatically does not fulfill this property and, consequently, by using such a node w' as target node the resulting network N_i would not display F_j, and, thus, $N_i \supset \mathcal{F}$ would not hold.

- For each source node $u \in \mathcal{V}_s^A \cup \mathcal{V}_s^B$ either the two trees $\overline{N_{i-1}|_{E',\mathcal{L}(\mathcal{F}')}(u)}$ and $T_i|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$ are isomorphic (if $u \in \mathcal{V}_s^A$) or, after the insertion of all components in $\mathcal{F}$, there exists a certain path leading to such a node whose edges can be used for displaying T_i (if $u \in \mathcal{V}_s^B$). Choosing a node $u' \notin \mathcal{V}_s^A \cup \mathcal{V}_s^B$ as source node, the reticulation edge e inserted for u' and some node $w \in \mathcal{V}_t$, could not be used for displaying T_i in N, since $T_i|_{\mathcal{X}'\cup\mathcal{L}(F_j)}$ does not contain a node v whose subtree $\overline{T_i|_{\mathcal{X}'\cup\mathcal{L}(F_j)}(v)}$ is isomorphic to $\overline{N_i|_{E_i,\mathcal{X}'\cup\mathcal{L}(F_j)}(u')}$, with E_i referring to T_i.

Thus, following an acyclic ordering of $\mathcal{F}$, the algorithm allHNetworks considers all possible source and target nodes that can be used for inserting one of its components into the so far computed network N_{i-1}.

However, as already discussed, for $\mathcal{F}$ there may exist different acyclic orderings and, depending on these acyclic orderings, the set $\mathcal{F}'$ of so far added components varies. Consequently, for different acyclic orderings the tree $T_i|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$ can differ, which may lead to different sets of source nodes. However, since for inserting an acyclic agreement forest $\mathcal{F}$ the algorithm allHNetworks takes all of its acyclic orderings into account, all of these different sets of target nodes are automatically considered and, thus, Lemma 17 is established. □

We have shown so far that, given an ordering of input trees $\Pi^* = (T_1, T_2, \dots, T_n)$, each input tree T_i can be added sequentially to a so far computed network N_{i-1} displaying all previous trees $\{T_1, T_2, \dots, T_{i-1}\}$ by inserting an acyclic agreement forest $\mathcal{F}$ for some embedded tree T' and T_i in all possible ways such that there does not exist a network N_i displaying $\{T_1, T_2, \dots, T_i\}$ with $N_i \supset \mathcal{F}$ and $N_i \supset N_{i-1}$. Notice that, as for inserting T_i all embedded trees are taken into account, if the algorithm would additionally consider all acyclic agreement forests of arbitrary size, Lemma 16 and 17 would be sufficient to establish Theorem 15.

However, in order to maximize efficiency, the algorithm allHNetworks only focuses on maximum acyclic agreement forests and, thus, we still have to show why we only have to consider acyclic agreement forests of minimum size. For instance, as depicted in Figure 3.17, it can happen that for a specific ordering of the input trees more reticulation edges have to be added when inserting leading input trees so that the resulting networks contain embedded trees that are necessary to obtain so-called *hidden relevant networks* at the end. In the following, however, we will show that, if such a hidden relevant network for a specific ordering of input trees exists, this network has to be contained in a set of relevant networks calculated for another ordering of the input trees.

Now, before presenting the third lemma, we will first introduce a simple modification of the algorithm allHNetworks. Let $\mathcal{T}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees, then, allHNetworks* denotes a modification of the algorithm allHNetworks that considers for the insertion of an input tree $T_i \in \mathcal{T}$ to so far computed networks all acyclic agreement forests of arbitrary size (instead of just those of minimum size).

Lemma 18. *Let $\mathcal{T}$ be a set of rooted binary phylogenetic $\mathcal{X}$-trees. A relevant network N for $\mathcal{T}$ is calculated by calling* allHNetworks*$(\mathcal{T})$ *if and only if it is calculated by calling* allHNetworks$(\mathcal{T})$.

Proof. '$\Longleftarrow$': As each computational path of the algorithm allHNetworks is also conducted by the modified algorithm allHNetworks*, each relevant network calculated by calling allHNetworks$(\mathcal{T})$ is obviously also calculated by calling allHNetworks*$(\mathcal{T})$.

'$\Longrightarrow$': Here, we have to discuss why the algorithm allHNetworks has *not* to consider non-maximum acyclic agreement forests leading to relevant networks. For this purpose, we will first show by induction on $n = |\mathcal{T}|$ that, if for a specific ordering Π of the input trees a relevant network N^* can be only computed by applying a non-maximum acyclic agreement forest $\mathcal{F}_i^*$, then, in this case, there exists a different ordering Π^* computing N^* by only taking components of maximum acyclic agreement forests into account.

Base case. The assumption, obviously, holds for $n = 1$. For $n = 2$ an agreement forest that is *not* maximal cannot lead to relevant networks, since the insertion of a maximum acyclic agreement forest $\mathcal{F}$ directly leads to a network whose reticulation number is smaller. This is, in particular, the case, since the algorithm inserts a reticulation edge for all components of an agreement forest, except F_ρ, and, thus, in this simple case, the hybridization number simply equals $|\mathcal{F}| - 1$. Note that, due to Lemma 16 and 17, in the case of two input trees, all relevant networks are calculated.

Inductive step. Now, let $\Pi^* = (T_1, \ldots, T_i, \ldots, T_n)$, with $n > 2$, be an ordering of input trees for which the algorithm allHNetworks calculates the set $\mathcal{N}_{n-1}$ consisting of all relevant networks for $\mathcal{T} \setminus \{T_n\}$ and there exists a hidden relevant network N^* for Π^* that could only be computed by inserting reticulation edges for a *non*-maximum acyclic agreement forest $\mathcal{F}_i^*$ for an input tree T_i $(1 \leq i < n)$ and an embedded tree T_i' of the network N_{i-1}^* displaying $\{T_1, T_2, \ldots, T_{i-1}\}$. Notice that this directly implies that in N^* there exist $x > 0$ reticulation edges only necessary for displaying both trees T_i and T_n, where x denotes the difference between $|\mathcal{F}_i^*|$ and the size of a maximum acyclic agreement forest $\hat{\mathcal{F}}_i^*$ for T_i and T_i', i.e., $x = |\mathcal{F}_i^*| - |\hat{\mathcal{F}}_i^*|$. In this case, however, as we will show in the following, N^* can be also calculated by applying the algorithm to the ordering $\Pi = (T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n, T_i)$, where T_i is inserted right after T_n.

For this purpose, let N_{n-2} be the relevant network displaying each tree except T_i and T_n in the same topological way as it is the case for N^*. More precisely, N_{n-2} equals the network that is obtained from N^* by first deleting a set of reticulation edges E_i^*, containing each edge that is not necessary for displaying an input tree in $\mathcal{T} \setminus \{T_i\}$, then by deleting a set of reticulation edges E_n^*, containing each remaining edge that is not necessary for displaying an input tree in $\mathcal{T} \setminus \{T_n\}$, and finally by suppressing all nodes of both in- and out-degree 1. Notice that we can calculate N_{n-2} by applying the algorithm allHNetworks to $\mathcal{T} \setminus \{T_i, T_n\}$ since, by induction hypothesis, the algorithm is able to calculate all relevant networks embedding $\mathcal{T} \setminus \{T_n\}$.

Next, let N_{n-1} be the relevant network displaying each tree except T_i in the same topological way as it is the case for N^*. More precisely, N_{n-1} equals the network that is obtained from N^* by first deleting each reticulation edge that is not necessary for displaying an input tree in $\mathcal{T} \setminus \{T_i\}$ and then by suppressing all nodes of both in- and out-degree 1. Notice that, based on N_{n-2}, due to both previous Lemmas 16 and 17, this network can be calculated by inserting the components of an specific acyclic agreement forest $\mathcal{F}_n$ for T_n and the embedded tree T_i' of N_{n-2} with $|\mathcal{F}_n| = |E_n^*| + 1$. Moreover, T_i' is still contained in N_{n-2}, since for displaying this tree no reticulation edge is necessary that has been added during the insertion of T_i and T_n and, thus, would not exist in N_{n-2}.

It still remains to show, however, why this acyclic agreement $\mathcal{F}_n$ is of minimum size. For this purpose, we will establish a proof by contradiction showing that in this case we could construct a hybridization network N' for $\mathcal{T}$ providing a smaller reticulation number than N^*. In a first step, however, we have to recall each acyclic agreement forest that is used in Π^* as well as in Π in order to insert the two trees T_i and T_n. The reader should keep in mind that by inserting a tree based on an acyclic agreement forest of size k, the algorithm allHNetworks inserts precisely $k-1$ reticulation edges.

- Regarding Π^*, first the tree T_i is inserted by a non-maximum acyclic agreement forest $\mathcal{F}_i^*$ of size k_i^* and then the tree T_n is inserted by a maximum acyclic agreement forest $\mathcal{F}_n^*$ of size k_n^*.
- Regarding Π, first the tree T_n is inserted by a maximum acyclic agreement forest $\mathcal{F}_n'$ of size k_n' and then the tree T_i is inserted by a maximum acyclic agreement forest $\mathcal{F}_i$ of size k_i'.

Now, in order to establish a contradiction, let us assume that $k_n' = |\mathcal{F}_n'| < |\mathcal{F}_n|$. Notice that through $\mathcal{F}_n$ the edge set E_n^* is reinserted, which implies that $\mathcal{F}_n$ has to contain precisely $|E_n^*| + 1 = k_n^* + x$ components, where x, as already mentioned above, denotes the difference between $|\mathcal{F}_i^*|$ and the size of a maximum acyclic agreement forest $\hat{\mathcal{F}}_i^*$ for T_i and T_i', i.e., $x = |\mathcal{F}_i^*| - |\hat{\mathcal{F}}_i^*|$. Regarding $\mathcal{F}_n'$, this means that we could insert T_i and T_n to $\mathcal{N}_{n-2}$ by inserting precisely $r_1 = k_n' - 1 + k_i^* - x - 1$ reticulations edges. Next, by considering the number of reticulation edges that are added in Π^* for T_i and T_n, which are $r_2 = k_i^* - 1 + k_n^* - 1$, we can establish the following inequation:

$$r_1 = k_n' - 1 + k_i^* - x - 1 < k_n^* + x - 1 + k_i^* - x - 1 = k_i^* - 1 + k_n^* - 1 = r_2.$$

In summary, this means that, if $|\mathcal{F}'_n| < |\mathcal{F}_n|$ holds, we could construct a network N' with $r_1 = r(N') < r(N^*) = r_2$ by inserting both trees T_i and T_n into N_{n-2} in respect to $\hat{\mathcal{F}}^*_i$ and $\mathcal{F}'_n$, which implies that N^* would not be a relevant network for $\mathcal{T}$; a contradiction to the choice of N^*.

Lastly, based on N_{n-1}, again due to both previous Lemmas 16 and 17, the network N^* can be calculated by inserting the components of an specific acyclic agreement forest for T_i and some embedded tree of N_{n-1}. Notice that this acyclic agreement forest has to be of minimum size, since, otherwise, by simply taking only maximum acyclic agreement forests into account we could directly construct networks providing a smaller reticulation number than N^*. Again, this would directly imply that N^* could not be a relevant network for $\mathcal{T}$; a contradiction to the choice of N^*, which finally establishes the induction step.

Based on the induction above, we can make the following observation. If for a specific ordering of the input trees there exists a tree T_i that has to be added by a *non*-maximum acyclic agreement forest in order to enable an insertion of another input tree T_j $(i < j)$, which is necessary for the computation of a relevant network N, then, in this case, we can compute N by applying the algorithm allHNetworks to an ordering where T_i is located after T_j. Thus, for each relevant network N that could only be computed by our algorithm by applying *non*-maximum acyclic agreement forests, there exists a certain ordering of the input trees such that our algorithm is able to compute N by only taking maximum acyclic agreement forests into account. Finally, as a direct consequence, since our algorithm takes all possible orderings of input trees into account, our algorithm obviously guarantees the computation of all relevant networks without considering *non*-maximum acyclic agreement forests. Thus, the correctness of Lemma 18 is established. □

Now, based on the fact that the algorithm allMAAFs returns all maximum acyclic agreement forests for two binary phylogenetic $\mathcal{X}$-trees [57, Theorem 2], by combining Lemma 16–18 the correctness of Theorem 15 is established.

More precisely, this is the case, because due to Lemma 16 we can derive a network displaying a further input tree T_i from an acyclic agreement forest $\mathcal{F}$ for T_i and an embedded tree of a so far computed network. Moreover, due to Lemma 17, by taking all orderings of the input trees into account, for this purpose it suffices to consider only acyclic agreement forests of minimum size. Furthermore, by considering all possible ways of how such a maximum acyclic agreement forest $\mathcal{F}$ can be inserted (cf. Lemma 17), the algorithm allHNetworks calculates each network displaying $\mathcal{F}$. Now, since T_i is added to all so far computed networks by taking all maximum acyclic agreement forests for all embedded trees into account, all networks embedding T_i are calculated. Consequently, by adding all input trees sequentially for all orderings in this way all relevant networks for all input trees are calculated. □

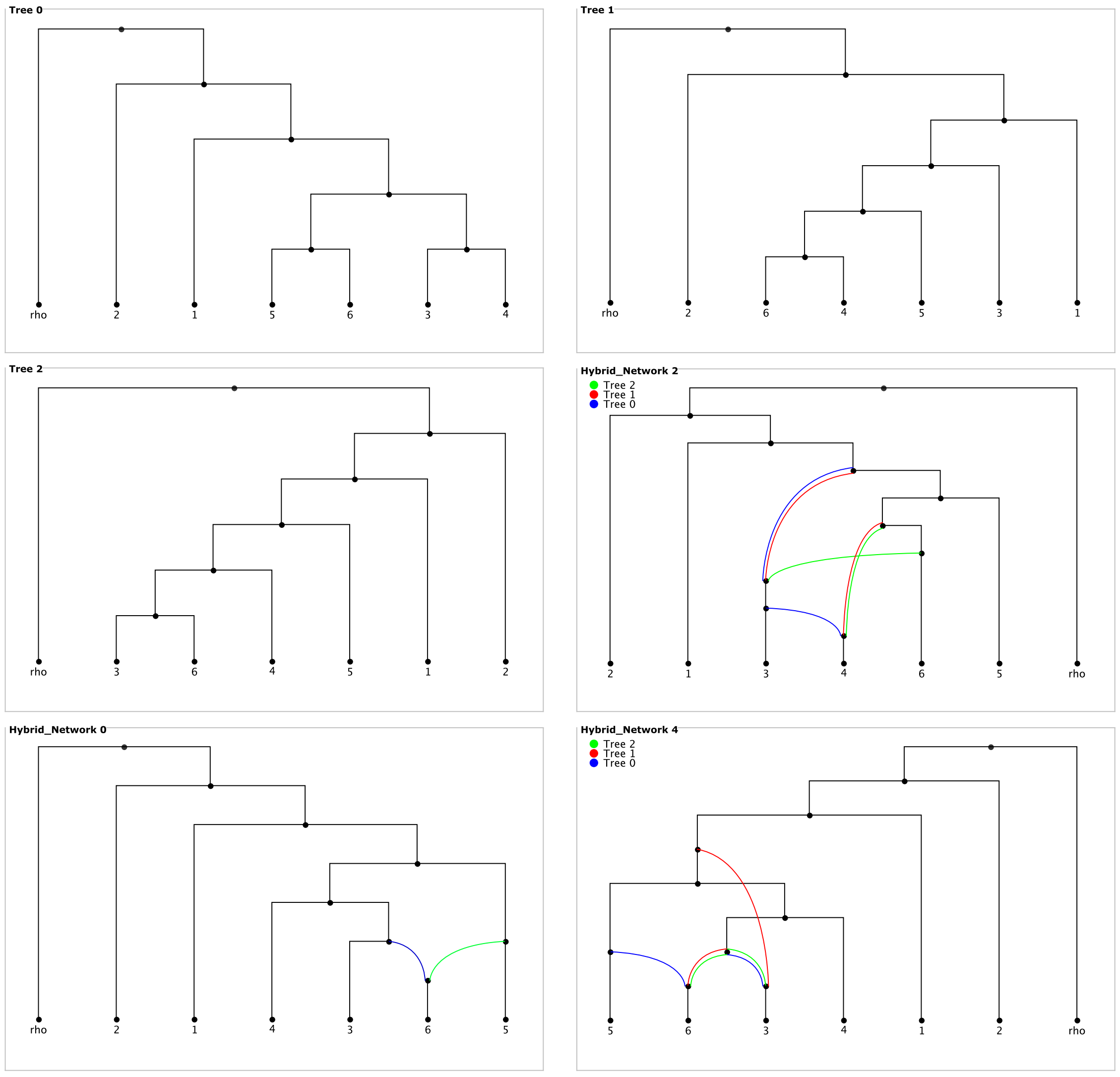


Figure 3.17: An example showing why the algorithm allHNetworks has to consider different orderings of the input trees. By running the algorithm allHNetworks for the ordering $\Pi_1 = (\mathit{Tree\ 0}, \mathit{Tree\ 1}, \mathit{Tree\ 2})$ only those networks with hybridization number two, as the one denoted by *Hybridization_Network 2*, are computed providing a hybridization node whose subtree consists of taxon 4. This is the case, since the only maximum acyclic agreement forest for *Tree 0* and *Tree 1* is of size two containing the component consisting of the single taxon 4. To compute the network denoted as *Hybridization_Network 4* at bottom right, you have to apply the algorithm to the ordering $\Pi_2 = (\mathit{Tree\ 0}, \mathit{Tree\ 2}, \mathit{Tree\ 1})$, since now, in a first step, by adding *Tree 2* to *Tree 0* the network at bottom left, denoted by *Hybridization_Network 0*, is computed. Based on this network you can select an embedded tree T' by choosing its blue in-edge. As a direct consequence, the only maximum acyclic agreement forest for T' and *Tree 2* is of size two containing the component consisting of the single taxon 3 and, thus, by adding this component to the network *Hybridization_Network 0*, finally, the network *Hybridization_Network 4* is computed. Regarding the first mentioned ordering Π_1, this network could only be computed by our algorithm by considering the *non*-maximum acyclic agreement forest for *Tree 0* and *Tree 1* of size three containing the two components consisting of the single taxa 3 and 6.

3.4.5 Runtime of allHNetworks

In order to analyze the theoretical worst-case runtime of the presented algorithm allHNetworks, we have to discuss the complexity of three major steps including the computation of embedded trees, the computation of all maximum acyclic agreement forests of size k, and the computation of all possible reticulation edges that can be added for a given maximum acyclic agreement forest. Given an ordering of the input trees, each of those major steps has to be applied sequentially to each input tree in order to insert this tree into a set of so far computed networks. At the beginning, when adding the second input tree, this set of networks only consists of the first tree of the ordering. However, as shown in the upcoming part, this set grows exponentially in the number of input trees.

Theorem 19. *The theoretical worst-case runtime of the algorithm* allHNetworks *for computing all relevant networks for a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees with minimum hybridization number k is*

$$O\left(n!\left(2^k\tbinom{|E|}{k}k!\tbinom{|V|}{2}k\right)^{n-1}\left(|V|+3^{|\mathcal{X}|}\right)\right),$$

where E denotes the edge set and V denotes the node set of a binary tree in $\mathcal{T}$.

Proof. To show the correctness of Theorem 19, we divide the stated runtime estimation into four parts A–D and discuss each of those parts separately:

$$O\left(\underbrace{n!}_{A}\left(\underbrace{2^k}_{B}\underbrace{\tbinom{|E|}{k}k!}_{C}\underbrace{\tbinom{|V|}{2}k}_{D}\right)^{n-1}\left(\underbrace{|V|}_{B}+\underbrace{3^{|\mathcal{X}|}}_{C}\right)\right)$$

Part A. Since different orderings of the input trees can lead to different relevant networks, the insertion of the trees has to be performed for all $n!$ possible orderings.

Part B. The number of embedded trees of a network is at most 2^r where r denotes its number of hybridization nodes. This upper bound, however, is achieved only if each hybridization node has in-degree 2. Otherwise, if a hybridization node has more than two in-edges, the number of embedded trees is smaller as only one of those edges can be part of an embedded tree. Moreover, extracting a tree from a given network is a process of rather low complexity, which can be solved by iterating a constant number of times over all nodes of the network. Thus, the complexity of extracting one certain embedded tree is linear in the number of nodes.

Part C. The number of all maximum acyclic agreement forests of size k for two input trees T_1 and T_2 can be estimated by $O(\binom{|E(T_1)|}{k})$. In practice, however, this number is clearly smaller since, in general, less than k hybridization events, say r, are necessary for the insertion of one of the input trees. Moreover, only a few number of all $\binom{|E|}{r}$ possible sets of components fulfills the definition of an acyclic agreement forest.

Given an agreement forest of size k, there exist at most $k!$ acyclic orderings. Note that, similar to the number of all maximum acyclic agreement forests, there exist, in general, clearly less orderings. This number, however, can be large if there are a lot of components consisting of isolated nodes. The runtime for the computation of those maximum acyclic agreement forests is stated in the work of Scornavacca *et al.* [57, Theorem 3] by $O(3^{|\mathcal{X}|})$, where $\mathcal{X}$ denotes the taxa set of each input tree.

Part D. As mentioned during the presentation of the algorithm ALLHNETWORKS, a component of a maximum acyclic agreement forest can potentially be added in several ways to a so far computed network N. This number is, obviously, bounded by $\binom{|V|}{2}$ where V denotes the set of nodes corresponding to N. In practice, however, this number is clearly smaller, since only a small fraction of all possible node pairs enable a valid embedding of an input tree. Lastly, given a source and a target node, a new reticulation edge can be simply added by performing a constant number of basic tree operations. □

3.4.6 Speeding Up the Algorithm allHNetworks

To handle the huge computational effort, which is indicated in Theorem 19, it is very important to implement the algorithm in an efficient way. This can be done by parallelizing its execution on distributed systems, by initially applying certain reductions to the input trees, and by reducing the computation of isomorphic networks.

3.4.6.1 Parallelization

In order to improve the practical runtime of our algorithm, each exhaustive search looking for relevant networks with minimum hybridization number k can be parallelized as follows. As described in Section 3.4.2, the insertion of a tree T_i to a so far computed network results in several new networks which are then processed by inserting the next input tree T_{i+1} of the chosen ordering (cf. Fig. 3.18).

Since the processing of networks runs independently from each other, these steps can be parallelized in a simple manner. Notice, however, that, based on the reticulation number of so far computed networks, each of those steps is more or less likely to result in relevant networks. Thus, one can set up a priority queue to process the most promising networks first, which depends, on the one hand, on the number of so far embedded input trees and, on the other hand, on its current reticulation number. One should keep in mind, however, that such a priority queue can only speed up the computation of the hybridization number, since, only in this case, the search can be aborted immediately as soon as the first relevant network has been calculated. Otherwise, if one is interested in all relevant networks, each network has to be processed anyway until either it can be early aborted (which is the case if the reticulation number exceeds k) or it results in relevant networks.

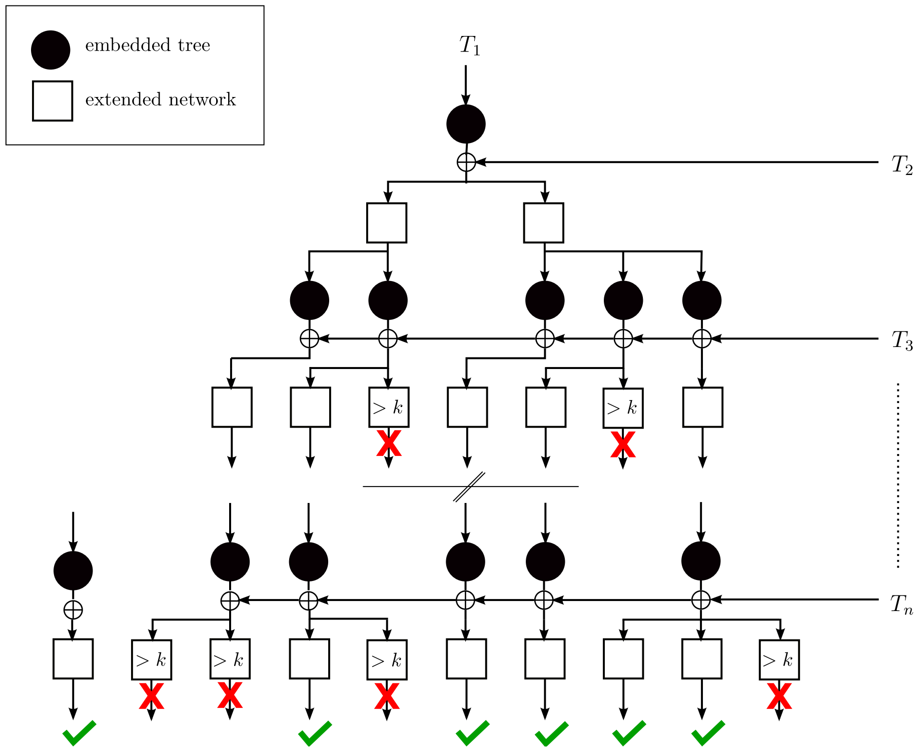


Figure 3.18: An illustration of how the insertion of the input trees is conducted by the algorithm ALLHNETWORKS in respect of the parameter k bounding the maximal reticulation number of resulting networks. Beginning with the first input tree T_1, repeatedly, first, an embedded tree T' of a so far computed network N is selected, and, second, the current input tree T_i is inserted into N by sequentially adding the components of a maximum acyclic agreement forest for T' and T_i. As soon as the reticulation number of a so far computed network exceeds k one can be sure that this network cannot lead to a network whose reticulation number is smaller than or equal to k and, thus, the corresponding computational path can be early aborted.

3.4.6.2 Reductions rules

In order to reduce the size of the input trees, before entering the exhaustive part of the algorithm, one can apply the reduction rules previously presented in Section 3.2. This is, on the one hand, the subtree reduction, following the work of Bordewich and Semple [12], and, on the other hand, the cluster reduction, following the work of Baroni *et al.* [9] and Linz [40]. The subtree reduction simply replaces all maximum common subtrees of all input trees which, however, usually only affects a relatively small number of small subtrees.

The cluster reduction, in contrast, cuts down the set of input trees into all minimum common clusters whose relevant networks can then be computed independently by running the presented algorithm for each of those clusters separately. Consequently, the cluster reduction usually provokes a significant speedup, because often a problem of high computational complexity can be separated into several subproblems providing low computational complexities, which can be solved efficiently on its own. Notice that, in Section 3.3, we give a proof showing that the cluster reduction is save for multiple rooted binary phylogenetic

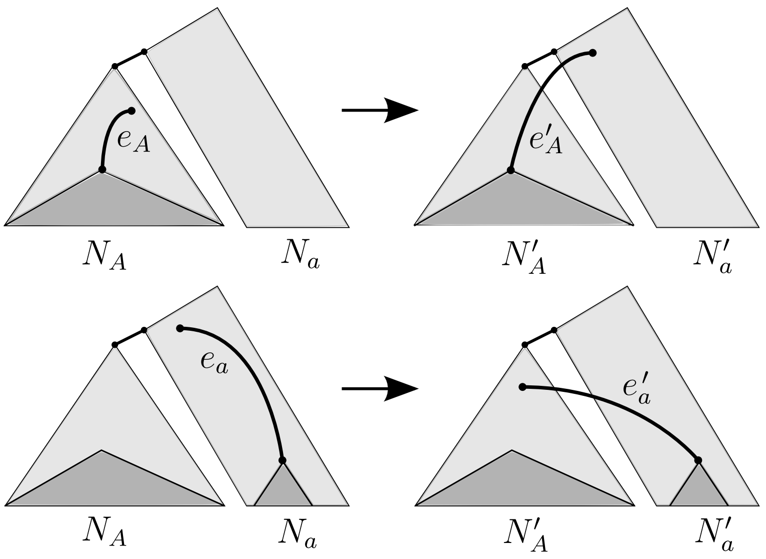


Figure 3.19: An illustration of an edge in $\mathcal{N}_{A,a}$ that is caught in a subgraph corresponding to $\mathcal{N}_A$ and $\mathcal{N}_a$, respectively. This means in particular that both relevant networks on the right hand side are missing in $\mathcal{N}_{A,a}$.

$\mathcal{X}$-trees $\mathcal{T}$, which means that $h(\mathcal{T})$ corresponds to the sum of the minimum hybridization numbers each calculated for a different common cluster.

However, when applying the cluster reduction to a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees, in order to obtain a set consisting of all relevant hybridization networks $\mathcal{N}$ displaying $\mathcal{T}$, due to the following observation one still has conduct further combinatorial steps. Let $A \subset \mathcal{X}$ be a common cluster of $\mathcal{T}$ and let $\mathcal{T}_a$ be the set of trees obtained from $\mathcal{T}$ by replacing each cluster A through a leaf labeled by taxon $a \notin \mathcal{X}$. Then, in a further step, one still has to reattach the networks computed for $\mathcal{T}|_A$ and $\mathcal{T}_a$, shortly denoted by $\mathcal{N}_A$ and $\mathcal{N}_a$, respectively, as follows.

First, replace each taxon a of a network in $\mathcal{N}_A$ by each network in $\mathcal{N}_a$ resulting in a set of networks $\mathcal{N}_{A,a}$. However, due to the following observation this set $\mathcal{N}_{A,a}$ might be just a subset of all relevant networks $\mathcal{N}$. Since $\mathcal{N}_A$ and $\mathcal{N}_a$ are calculated separately, the source node of a reticulation edge is caught in $\mathcal{N}_A$ and $\mathcal{N}_a$, respectively. This means, in particular, that, regarding the set of reattached networks $\mathcal{N}_{A,a}$, each network whose source node of a specific edge e could be also located outside of its subgraph referring to $\mathcal{N}_A$ or $\mathcal{N}_a$, is missing (cf. Fig. 3.19). For example, regarding Figure 3.21, the network at the bottom left would not be calculated, which is due to the fact that the blue in-edge referring to node 6 could not "leave" the common cluster $\{2, 3, 4, 9\}$. However, keep in mind, that, as already proven in Section 3.3, this fact does not have an impact on the calculation of the minimum hybridization number $h(\mathcal{T})$ and, as demonstrated in the following, we can still generate the set $\mathcal{N} \setminus \mathcal{N}_{A,a}$ of missing networks by applying further linking patterns.

Let N_a and N_A be a network of $\mathcal{N}_a$ and $\mathcal{N}_A$, respectively. Then, this second step generates each missing network by performing *legal shifting steps* reattaching reticulation edges to subgraphs beyond the border of two joined networks being part of N_A and N_a, respectively. More precisely, we call a *legal shifting step of a reticulation edge* e that is part of a subgraph corresponding to N_A (resp. N_a), if it is possible to reattach $e = (x, y)$ to a node $s \neq x$ located in the subgraph corresponding N_a (resp. N_A), so that still all input trees are displayed in the resulting network $N'_{A,a}$. Note that by saying reattaching we mean that first e is deleted, then a new edge (s, y) is inserted, and finally all nodes of both in- and out-degree one are suppressed. Now, in order to guarantee the computation of all relevant networks, for each network in $\mathcal{N}_{A,a}$, one simply has to take all combinations of legal shifting steps into account.

For a better illustration of this concept, we will describe some linking-patterns that can be used to apply legal shifting steps in respect to a node v and a subset $\mathcal{T}'$ of all input trees $\mathcal{T}$. In general, those pattern can be separated into three different types. A linking-pattern of Type A can be used to shift reticulation edges downwards in the given network, which means that the new source will be a successor of the original source node being part of a network separately calculated for a particular cluster of $\mathcal{T}$. Similarly, a linking-pattern of Type B can be used to shift reticulation edges upwards in the given network, which means that the new source will be a predecessor of the original source node being part of a network separately calculated for a particular cluster of $\mathcal{T}$. Once an edge has been shifted in terms of a pattern of Type A or Type B (or Type C), one can apply an additional linking-pattern of Type C (as defined below).

Just for convenience, in the following we will assume that the set of input trees $\mathcal{T}$ only consists of two trees, which means that each reticulation edge is only necessary for the embedding of one of both trees and not for more than one tree (which obviously could be the case if $\mathcal{T}$ contains more than two trees). Therefor, let N be a relevant network displaying two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2 and let E_i be an edge set referring to $T_i \in \{T_1, T_2\}$.

Linking-pattern of Type A. Let $e = (x, y)$ be a reticulation edge of E_i and let $\mathcal{P} = (s_0, s_1, \ldots, s_k)$ be a path in N in which $s_0 = x$, $y \notin \mathcal{P}$, and $v = s_i$ with $0 < i \leq k$. Moreover, let the out-degree of each node s_i, with $0 < i \leq k$, in $N|_{E_1,\mathcal{X}}$ be 1. Then, we can conduct a legal shifting step by first pruning e and then by reattaching it to any node of $\mathcal{P}$ (except s_0) (cf. Fig. 3.20).

Linking-pattern of Type B. Let $e = (v, y)$ be a reticulation edge of E_i and let $\mathcal{P} = (s_0, s_1, \ldots, s_k)$ be a path in N in which $s_k = v$. Moreover, let the out-degree of each node s_i, with $0 \leq i < k$, in $N|_{E_1,\mathcal{X}}$ be 1. Then, we can conduct a legal shifting step by first pruning e and then by reattaching it to any node of $\mathcal{P}$ (except s_k) (cf. Fig. 3.21).

Linking-pattern of Type C. Let x be the source node of a reticulation edge e_s that has already been shifted by applying a pattern of Type A or Type B. Moreover, let $e_t = (x, y)$ be an out-going tree edge of x not necessary for displaying T_i. Then, we can conduct a legal shifting step by first pruning e and then by reattaching it to y (cf. Fig. 3.20).

Now, let N be a network of the set $\mathcal{N}_{A,a}$ as defined above. Moreover, let v be the root of the subgraph corresponding to N_a. Then, by applying those three linking-patterns to each network in $\mathcal{N}_{A,a}$ and repeatedly to all resulting networks, one can produce the missing set of relevant networks.

Note that, when applying a linking-pattern of Type B, the initial node v might gets suppressed, if its in- and out-degree is 1. In such a case, v has to be redefined by the target of its out-going edge. Moreover, once an edge has been shifted downwards, one has to take care not shifting it back again upwards (and vice versa). This means, in particular, that edges that have been shifted in terms of a linking-pattern of Type A or B must not be shifted again by applying of one those two patterns.

Lastly, by applying those linking patterns, the resulting networks not necessarily have to match the definition of a relevant network as given in Section 3.4.1. Thus, one additionally has to apply the following two modifications.

Modification of Type A. By the linking patterns from above one automatically generates multifurcating nodes. Consequently, in order to turn those nonbinary networks into binary networks, one still has to resolve these nodes in all possible ways.

Modification of Type B. Moreover, by applying a linking-pattern of Type B, one can attach an edge e to a hybridization node, which consequently means that a network is generated containing hybridization nodes of out-degree larger than one. As a consequence, one either has to reject those networks or, if such a hybridization node provides an in-edge e_h that can be used for displaying the same set of trees as for e, one can first split e_h and then attach e to the new inserted node (cf. Fig. 3.21).

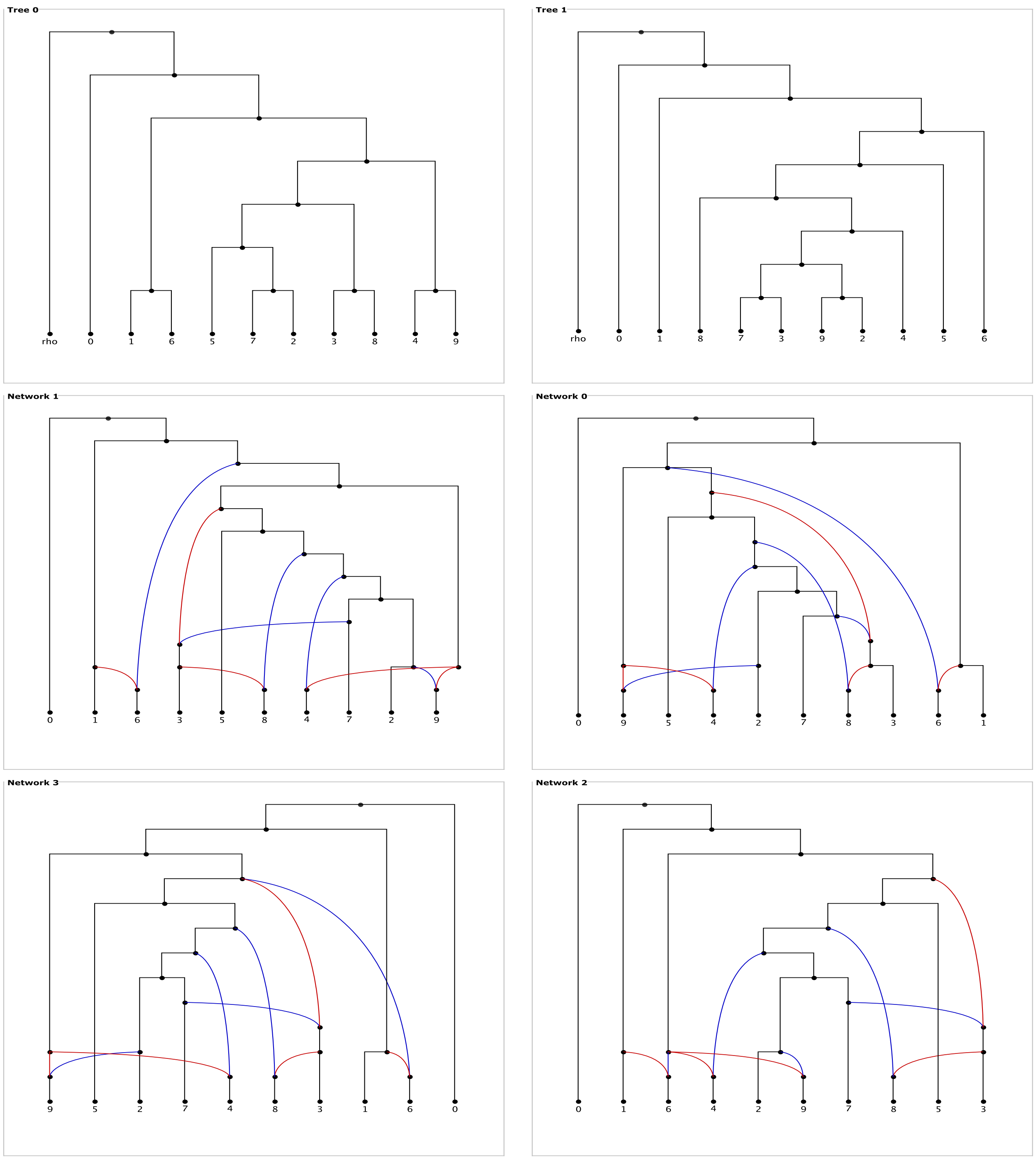


Figure 3.20: (**Top**) Two rooted binary phylogenetic $\mathcal{X}$-trees sharing the common cluster $\{2, 3, 4, 5, 7, 8, 9\}$. (**Rest**) Minimum hybridization networks displaying both trees from the top where red edges refer to the left and blue edges to the right tree. Both networks, Networks 0 and Networks 2, can be obtained from Network 1 by applying a linking-pattern of Type A, whereas Network 3 can be obtained from Network 0 by applying a linking-pattern of Type C.

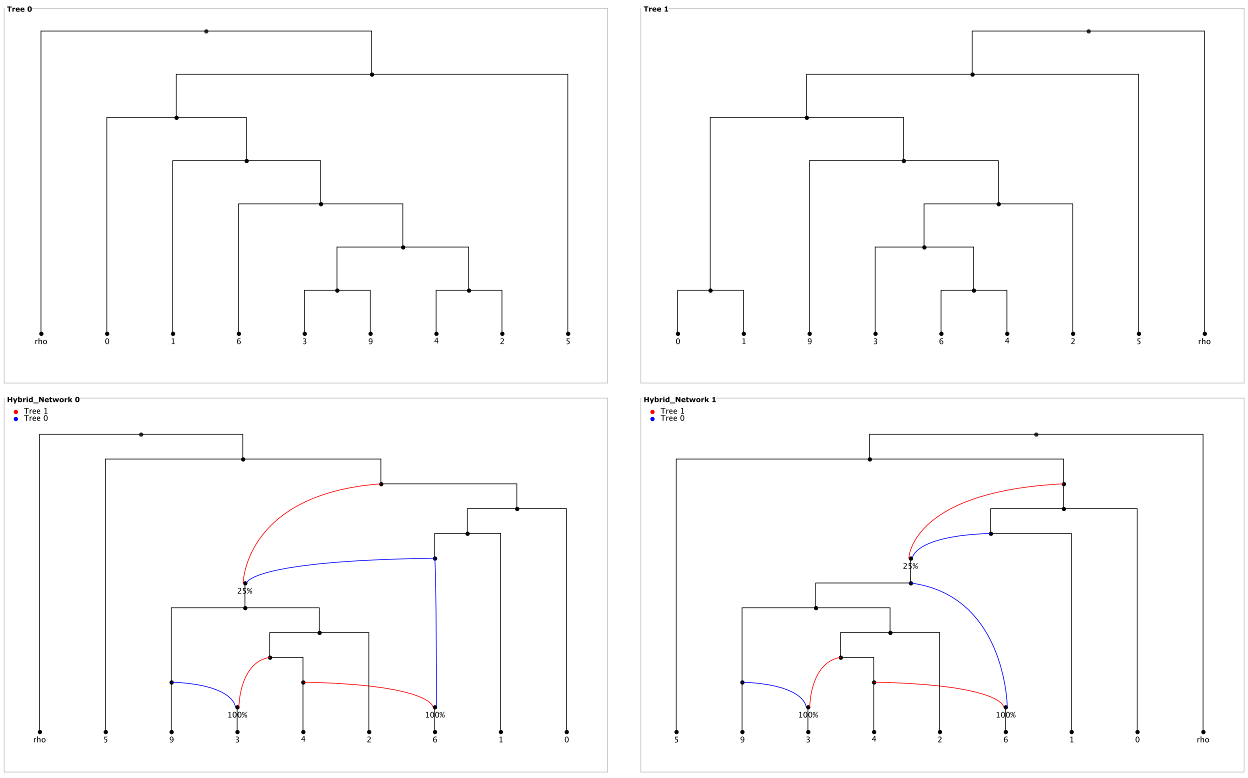


Figure 3.21: (**Top**) Two rooted binary phylogenetic $\mathcal{X}$-trees sharing the common cluster $\{2, 3, 4, 6, 9\}$. (**Bottom**) Two networks displaying both trees from the top where blue edges refer to the left and red edges to the right tree. The left network can be obtained from the right one by first applying a linking-pattern of Type B, attaching the blue in-edge referring to node 6 to the hybridization node labeled by 25%, and then by applying a modification of Type B.

3.4.7 Conclusion

To analyze hybridization events, it is of high interest to compute all hybridization networks, since the more frequently an event occurs in all those networks the more likely it may be part of the true underlying evolutionary scenario. In this work, we first presented the algorithm allHNetworks calculating all relevant networks for an input consisting of multiple rooted binary phylogenetic $\mathcal{X}$-trees and then established its correctness by a detailed formal proof. Its theoretical worst-case runtime, however, reveals that the number of those networks growths in a strong exponential manner in terms of the number and the size of the input trees which obviously complicates its application to real biological problems.

As a consequence, it is very important to technically implement the algorithm in an efficient way (e.g., parallelizing particular substeps) what will be discussed in more detail the upcoming section. Moreover, in an algorithmic point of view, in order to improve the practical runtime, one can apply a cluster reduction to the input trees as already described in Section 3.3. We have demonstrated, however, that when separating those input trees into several clusters, in order to obtain all relevant networks, one still has to spend some work in attaching back different networks separately computed for each of those clusters. However, if one is only interested in the hybridization number, this post-processing step is not necessary as already proven in Section 3.3.

3.5 A fast approach for computing minimum hybridization networks

In this section, we present the *first* approach that is able to compute the *exact* hybridization number as well as a certain set containing *all* representative networks (as defined later) for, not just only two, but an *arbitrary number* of rooted binary phylogenetic $\mathcal{X}$-trees all sharing the same set of taxa. Note that, until now, the software PIRNv2.0 [72, 73] is the most efficient software that guarantees the computation of the *exact* hybridization number for more than two input trees. In most cases, however, PIRN runs only reasonable efficient if the number of hybridization events is relatively small and, moreover, PIRN does usually output only a small subset of those networks that are computed by our method which plays an important role for the interpretation of the networks as discussed later. The algorithm, presented in this section, is based on previous work of Albrecht *et al.* [6] describing an algorithm for just *two* input trees (cf. Sec 2.2), which itself is based on several works including Baroni *et al.* [9], Bordewich and Semple [13], and Whidden *et al.* [70]. Moreover, this previous approach could only compute a subset of all representative networks and, thus, the motivation for this work was to extend this former algorithm such that now all of those networks for an *arbitrary number* of input trees can be computed.

As we state that our algorithm guarantees the computation of the *exact* hybridization number, we are aware of the fact that this algorithm raises some questions regarding its correctness. However, since in this section we want to focus on the efficiency of the presented algorithm as well as on the practical aspects of our software Hybroscale regarding the interpretation of hybridization networks, those rather complex theoretical issues have already been discussed separately in the previous section.

Given a hybridization network displaying several input trees, it is often visually challenging for a user to figure out the embedding of those trees. Thus, we have developed the software Hybroscale providing a function for highlighting each input tree by coloring its corresponding edges within a resulting network, which makes it easier for a biologist to analyze hybridization events. Moreover, Hybroscale sorts the set of computed networks by support values indicating how often a particular hybridization event occurs in the set of representative networks.

To demonstrate the efficiency of our implementation, we computed the hybridization number for a specific synthetic dataset and compared the respective runtime with the best currently available software PIRNv2.0 [72, 73]. Note that there are two main differences between our approach and the one corresponding to PIRN. On the one hand, our software provides the better practical runtime for computing hybridization numbers because of parallelization, certain reduction steps, and other algorithmic issues as discussed in the upcoming part of this section. On the other hand, our approach additionally enables the computation of *all* representative networks allowing the assignment of meaningful support values to each internal node representing a putative hybridization event which helps biologists to figure out hybridization events that might played an important role.

3.5.1 Further definitions

The upcoming definitions are crucial for describing and discussing our algorithm.

Hybridization networks. Given a hybridization network N with node v, the network $N(v)$ denotes a network rooted at v that is obtained from N by first removing each node that cannot be reached from v and then by suppressing each node of both in- and out-degree 1.

Representative networks. As mentioned above, our algorithm ensures the computation of *all representative networks*, which are those hybridization networks with minimum hybridization number (cf. Eq. (1.2)) not containing any stacks of hybridization nodes leaving the interpretation of the ordering of the hybridization events open. Moreover, just for simplicity, we claim that each of those networks has to be binary not containing any nodes of out-degree greater than 2. Notice that by introducing multifurcating nodes, which are nodes having an out-degree of at least 3, the set of representative networks typically shrinks because due to those nodes a network can display several binary networks.

Moreover, in order to improve its readability, we further demand that all hybridization nodes of a representative network have out-degree one. Notice that, in order to identify stacks of hybridization nodes, in such networks the out-edges of all hybridization nodes have to be suppressed.

Lastly, given two representative networks N_1 and N_2, we say that N_1 differs from N_2 if either their graph topologies (disregarding edge labels) are not isomorphic or their edge sets indicating the embedding of each input tree differ.

3.5.2 The algorithm

In this section, we give a high level description of our algorithm. More information, involving a more detailed description of the upcoming steps as well as some theoretical issues have already been discussed in the previous section.

The input of the algorithm is a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees and its output is either just the hybridization number or all representative networks showing the embedding of all those input trees. Similar to the approach described in Section 2.2, our algorithm is separated into three phases. The reduction phase (consisting of a subtree reduction following the work of Bordewich and Semple [12] and a cluster reduction following the work of Baroni *et al.* [9] and Linz [40]), the exhaustive search phase, and the output phase (combining the result of all clusters and undoing each subtree reduction). Whereas the reduction and the output phase can be conducted in polynomial time, the second phase solves an *NP-hard* problem and, thus, its theoretical worst-case runtime is exponential [14]. However, as recently shown by van Iersel and Linz [67], specific parts of the problem still remain fixed-parameter tractable, which is an important feature that is exploited by our algorithm to maximize its efficiency.

At this point, we have to give a remark regarding the correctness of the cluster reduction. The well-known work of Baroni *et al.* [9] contains a proof showing that the *exact* hybridization number of *two* binary phylogenetic $\mathcal{X}$-trees can also be computed by summing up the *exact* hybridization numbers of its minimum common clusters. A more general proof, showing that this concept also holds for *multiple* binary phylogenetic $\mathcal{X}$-trees, has been previously discussed in Section 3.3.

In the upcoming part, we will briefly discuss the exhaustive search phase and its parallel execution. A description of the other two phases is omitted but can be looked up in Section 2.2. The exhaustive search phase runs for an increasing parameter k bounding the reticulation number of each computed network. If a hybridization network with reticulation number less than or equal to k does not exist, the search is continued with $k+1$ until a hybridization network displaying all input trees can be computed.

Exhaustive search phase. Given a set $\mathcal{T}$ of rooted binary phylogenetic $\mathcal{X}$-trees and a parameter $k \in \mathbb{N}$, we first apply the algorithm allHNetworks (cf. Sec 3.4.2) calculating the set $\mathcal{N}$ of all *relevant* networks each displaying $\mathcal{T}$ with minimum hybridization number. Notice that in the output $\mathcal{N}$ only those networks are contained in respect to one single embedding of the input trees. More precisely, this is the case because we consider two relevant network as being different if their topology disregarding the embedding of $\mathcal{T}$ differs. Consequently, since by comparing *representative* networks the embedding of each input tree is now taken into account, we still have to compute for each network in $\mathcal{N}$ all possible combinations of edge sets each referring to the embedding of one input tree.

3.5.3 Parallelization

In order to improve the practical runtime, our implementation runs the exhaustive search phase in parallel as described in Section 3.4.6.1 including the application of a priority queue.

Moreover, as the algorithm computes networks for all different orderings of input trees and all different acyclic orderings of a maximum acyclic agreement forest, a representative network can be computed several times. As a consequence, to ensure that the output only consists of unique networks, one has to filter the set of networks obtained from the exhaustive search step. For this purpose, we, first, group this set after the sum of *support values* computed for each network (as defined later), and, second, check each of those subgroups for isomorphic networks in parallel. Due to the typically large number of computed networks (cf. Tab. 3.3, 3.4), the restriction of the filtering step to small subgroups usually provokes a large speedup. Note that, as already mentioned above, we consider two networks as being different if either their graph topologies (disregarding edge labels) are not isomorphic or their sets of edges necessary for displaying each input tree differ.

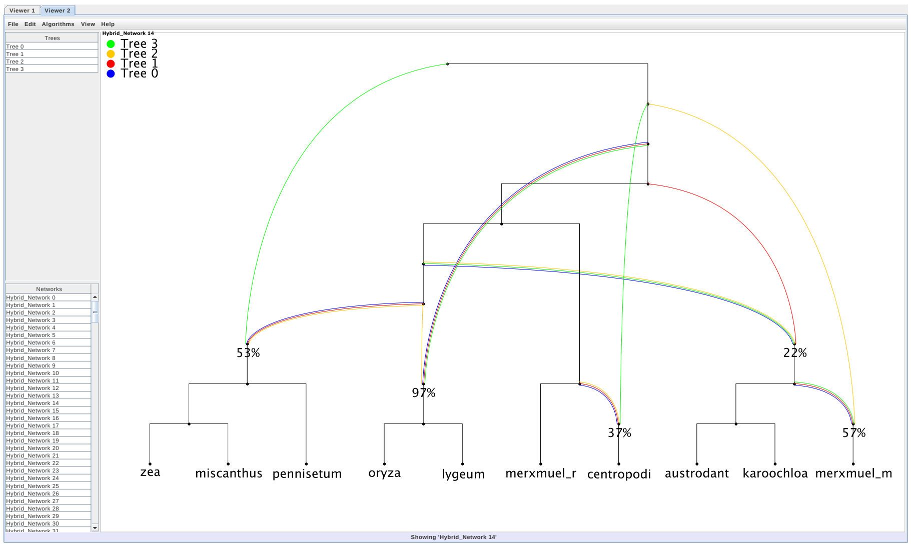


Figure 3.22: Our software Hybroscale showing a hybridization network displaying the embedding of three input trees by the colors blue, red, and green.

3.5.4 Additional features

Given just the *extended newick format* [15] of a hybridization network, its topology is in general hard to interpret. Although there exist software packages, which are able to display rooted phylogenetic networks, e.g., the software Dendroscope [33], most of them are not able to visualize the embedding of all input trees, which is a preferable feature for studying hybridization events. In order to close this gap, we have developed the software Hybroscale, which is specifically designed for studying hybridization networks. Beside the computation of a graphical layout of rooted trees and rooted networks, which is optimized by minimizing the number of crossings between all hybridization edges, Hybroscale can additionally highlight each hybridization edge that is necessary for displaying all embedded input trees by assigning a specific color to each tree (cf. Fig 3.22). Thus, Hybroscale is a tool that, on the one hand, enables an easy handling of our algorithm and, on the other hand, ensures the readability of the so computed networks.

Furthermore, Hybroscale assigns each hybridization node a *support value* indicating the fraction of networks containing this node and, additionally, sorts the reported networks by the sum of those values in decreasing order. More specifically, the computation of support values is done as follows. Given a network N, each edge set E_i referring to one of the input trees T_i, and a certain hybridization node v, we first compute the following ordering of taxa sets $\Pi(v) = (\mathcal{L}(N|_{E_1,\mathcal{X}}(v)), \dots, \mathcal{L}(N|_{E_n,\mathcal{X}}(v)))$. More precisely, each element $\mathcal{L}(N|_{E_i,\mathcal{X}}(v))$ consists of those taxa adhering to each leaf that can be reached from v by directed paths only crossing those hybridization edges in E_i indicating the embedding of T_i. For example, regarding Figure 3.22, the set referring to the hybridization edges indicating the embedding of *Tree 2* and the node labeled by 22% is $\{austrodant, karoochloa\}$. Second, we determine the fraction of networks containing $\Pi(v)$. This step ensures that the user can instantly look at those networks containing the most promising hybridization events, which is an important feature, because usually a large number of networks is reported (cf. Tab. 3.3, 3.4).

3.5.5 Results and discussion

In this section, we first report a simulation study indicating that our approach is much faster than the only so far existing competitive method PIRN and then illustrate how Hybroscale can be used for studying hybridization networks by applying the software to a well known grass (*Poaceae*) dataset.

3.5.5.1 Simulation study

To show the efficiency of our implementation, we have integrated our algorithm into the Java software Hybroscale and conducted a simulation study comparing its runtime to PIRNv2.0 [72, 73], which is so far the best available software for computing exact hybridization numbers for multiple rooted binary phylogenetic $\mathcal{X}$-trees.

Our synthetic dataset is freely available[1] and consists of several tree sets each containing multiple rooted phylogenetic $\mathcal{X}$-trees. Each $\mathcal{X}$-tree is generated by ranging over all different combinations of four parameters, namely the number of input trees n, the number of leaves ℓ, an upper bound for the hybridization number k, and the *cluster degree* c. Each of the n input trees is obtained from a bicombining network N, which means that N only contains hybridization nodes of in-degree 2. This network N is computed with respect to these four different parameters as follows. In a first step, a random binary tree T with ℓ leaves is computed which is done in the following way. First, at the beginning two nodes u and v of a specific set V, which is initialized by ℓ nodes of both in- and out-degree 0, are randomly selected. Those two selected nodes u and v are then connected to a new node w and, finally, V is updated by replacing u and v by its parent node w. This process is repeated until V consists only of one node corresponding to the root of T. In a second step, k hybridization edges are created in T with respect to parameter c such that the resulting network N contains exactly k hybridization nodes of in-degree 2.

In this context, the *cluster degree* is an *ad hoc* concept influencing the computational complexity of a tree set similar to the concept of the *tangling degree* introduced in the work of Scornavacca *et al.* [57]. When adding a hybridization edge e with target node v_2 and source node v_1, we say that e respects the cluster degree c if v_1 cannot be reached from v_2 and there is a path of length less than or equal to c leading from v_2 to a certain node p such that v_1 can be reached from p. Consequently, networks providing a small cluster degree in general contain more minimum common clusters than networks of large cluster degrees and, thus, typically can be processed quite fast when applying a cluster reduction beforehand. For a better understanding, in Figure 3.23 an example of this concept is depicted.

[1]`www.bio.ifi.lmu.de/softwareservices/hybroscale`

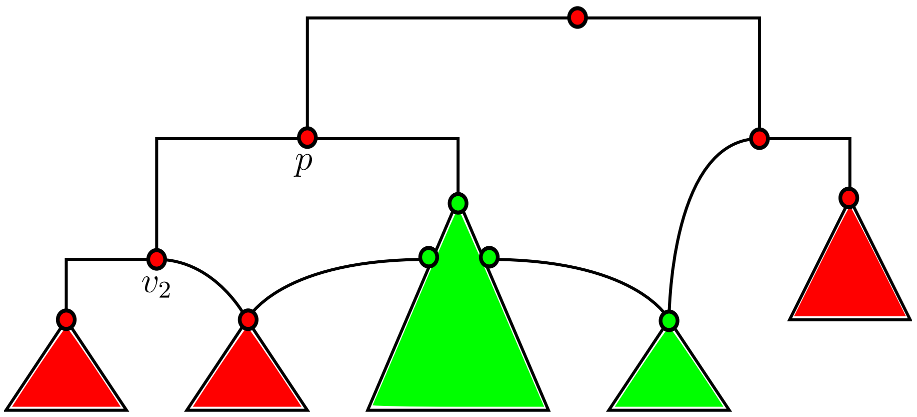


Figure 3.23: An illustration of the cluster degree parameter $c = 1$. When inserting an in-going edge e to node v_2 that is respecting c, each node that is marked green or is part of a green marked subnetwork forms a potential source node.

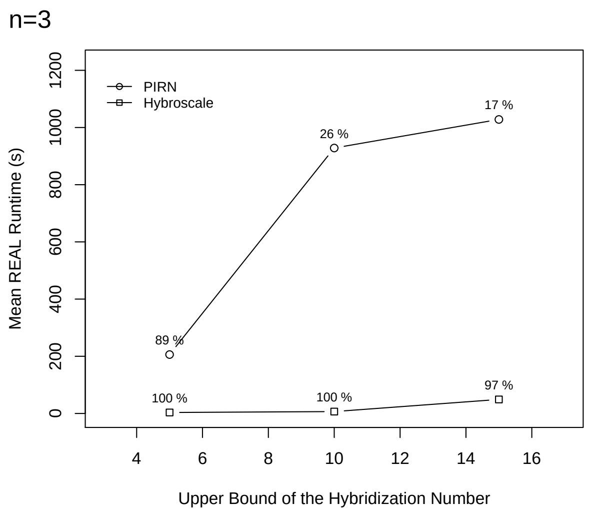


Figure 3.24: The figure shows the mean average runtime corresponding to HYBROSCALE and PIRNV2.0 grouped by parameter k denoting the hybridization number of the network that was used to obtain the tree set $\mathcal{T}$ from. Thus, this parameter k acts as an upper bound of the hybridization number of $\mathcal{T}$. Each percentage indicates the proportion of tree sets that could be computed within the time limit of 20 minutes.

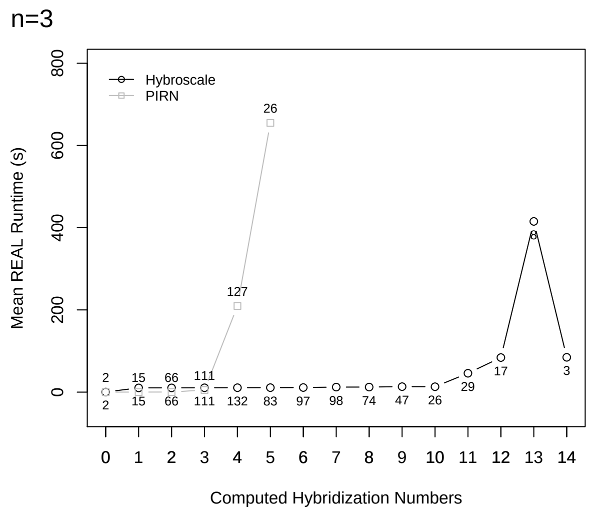


Figure 3.25: The figure shows the mean average runtime of all tree sets grouped by the computed hybridization numbers. The numbers inside the plot indicate how many tree sets could be computed for the corresponding hybridization number within the time limit of 20 minutes. Note that for the hybridization numbers 0 to 3 all corresponding tree sets could be computed by Hybroscale and PIRNv2.0 within comparable runtimes.

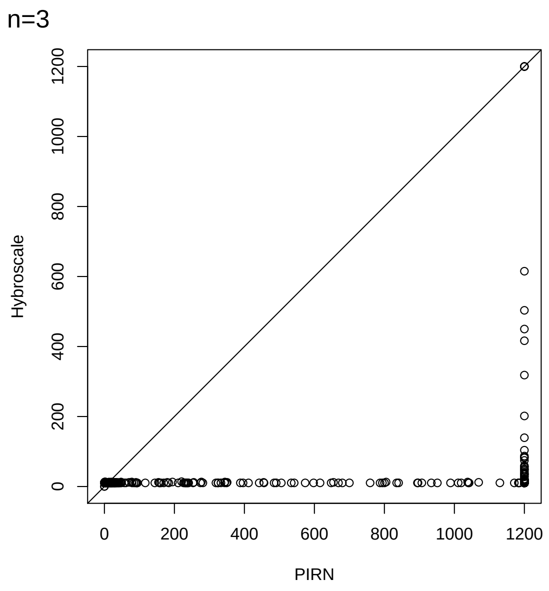


Figure 3.26: A scatterplot of the runtimes generated by PIRNv2.0 (x-axis) against the runtimes generated by Hybroscale (y-axis) of all 810 data sets consisting of *three* input trees. Note that PIRNv2.0 is not able to compute the result for 463 tree sets corresponding to each dot in the figure whose x-value is 1200. From those tree sets just 4 according to the dots whose y-value is also 1200 could not be computed by Hybroscale.

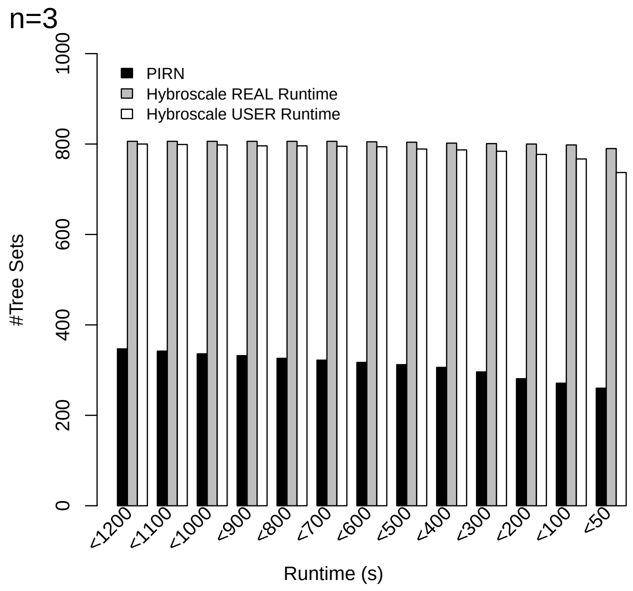


Figure 3.27: The figure shows the number of tree sets that could be computed within the runtime given on the x-axis by considering the real-runtime of PIRN and both real- and user-runtime of Hybroscale. The plot reveals that the massive parallelization with 16 cores does not significantly improve the runtime of Hybroscale in this case. This is the case because for three input trees the time limit of just 20 minutes as well as the complexity of the considered tree sets is simply too low.

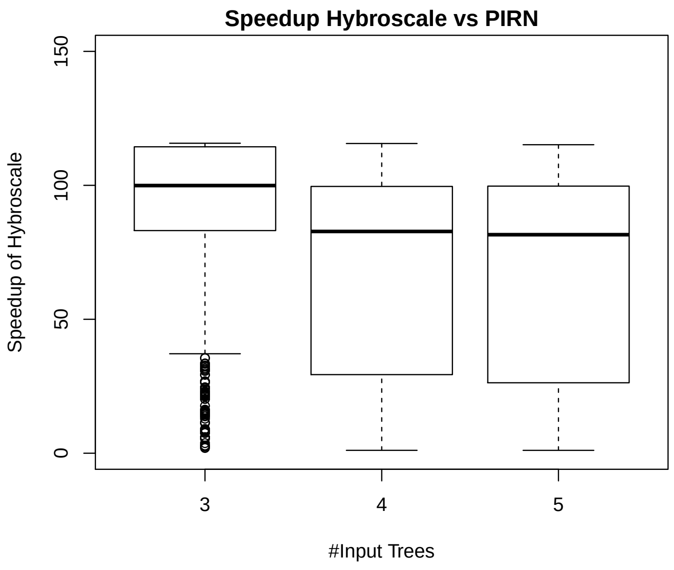


Figure 3.28: Distribution of the speedups of Hybroscale versus PIRN2.0 computed for each *relevant* tree set of our synthetic dataset. For three input trees Hybroscale is on mean average about 100 times faster than PIRN, for four input trees on mean average about 80 times, and for five input trees on mean average about 80 times.

To compare the efficiency, both programs have been run on a grid computer providing 16 cores and 40 GB RAM for our synthetic dataset containing tree sets with parameters

$n \in \{3, 4, 5\}$, $\ell \in \{10, 25, 50\}$, $k \in \{5, 10, 15\}$, and $c \in \{1, 3, 5\}$. More precisely, we have generated for all 81 combinations of the four parameters 30 tree sets as described above resulting in 2430 tree sets in total. The results for three input trees ($n = 3$) are presented in Figure 3.24–3.27. Due to time limitations, if the hybridization number of a certain tree set could not be computed within 20 minutes, the computation of this tree set was aborted. In Figure 3.24, 3.26, 3.27, and 3.28 those unfinished tree sets were taken into account with a runtime of 20 minutes whereas in Figure 3.25 these tree sets were omitted. The results for four and five input trees ($n = 4, 5$) are given in Figure 3.29–3.36. Notice that we do not give a discussion of these results because they are quite similar to those attained for three input trees.

Each of the simulation results given in Figure 3.24–3.28, which are now discussed in more detail, clearly demonstrates that our implementation is much faster than PIRN.

Figure 3.24 shows that, by increasing the upper bound of the hybridization number k, the mean average runtime of the datasets computed by PIRN increases up to 1000 seconds whereas the mean average runtime corresponding to Hybroscale is always below 100 seconds. Note that, as the runtime of each unfinished dataset was set to 1200 seconds, if we would set the time limit to a higher value, the maximal mean average runtime produced by PIRN is expected to be even higher — otherwise, to produce a reasonable comparison between both programs, we would have to leave out each dataset, which could not be computed by one of both programs, which means that we would end up with only those non-representative datasets that are quite easy to compute. Figure 3.25 shows that Hybroscale, in comparison to PIRN, can compute more datasets within the time limit and datasets having a significant larger hybridization number. Whereas PIRN is just able to compute hybridization numbers up to 5, Hybroscale is able to compute hybridization numbers up to 13.

Thus, it is obvious that our implementation outperforms PIRN, which becomes even clearer by looking at Figure 3.26 showing a scatterplot of the runtimes produced by both programs. The figure shows that for each runtime of a specific data set produced by PIRN the corresponding runtime of Hybroscale is smaller or equal. Moreover, looking at the bottom right of the figure, there exist a lot of data sets that could be computed by Hybroscale quite fast in less than 200 seconds, whereas PIRN is not able to come up with a result in less than 1200 seconds.

By comparing real- with user-runtimes, Figure 3.27 demonstrates that the better performance of Hybroscale is not only due to the applied massive parallelization. As the user-runtime indicates the total CPU time, which means that the time spent on all available cores is simply added up, this time indication corresponds to the runtime produced by a program that is executed on a system only providing a single core with no parallel execution taking place. Figure 3.27 shows the number of tree sets that could be computed within the runtime given at the x-axis. For example, the leftmost bar-group shows that PIRN could only finish 360 of 810 tree sets consisting of three trees within 1200 seconds whereas Hybroscale could finish 804 by taking parallelization into account and 793 by not taking parallelization into account. Note that the difference between both bars corresponding to the real- and user-runtime of Hybroscale would be even larger if, on the one hand, the dataset would contain tree sets of higher computational complexity and, on the other hand, the time limit would be set to a higher value. A possible explanation for the speedup without taking advantage of parallelization, apart from the reduction steps, is the proven method allMAAFs [57] that is used for solving the *NP-hard* problem of computing all maximum acyclic agreement forests. The efficiency of this method has been indicated recently in the work of Albrecht *et al.* [6].

Finally, we have computed the speedup of Hybroscale versus PIRN by comparing its runtimes produced for each *relevant* tree set within our synthetic dataset. More specifically, we consider each tree set d as being *relevant* in this case if at least one of both programs could process d within the time limit of 20 minutes and if at least one computation took longer than 50 seconds. Now, for each of those relevant tree sets d we computed its speedup $s(d) = T_P(d)/T_H(d)$, in which T_P and T_H denotes the real-runtime produced by PIRN and Hybroscale, respectively. Figure 3.28, showing the distribution of the speedups corresponding to each of those tree sets, reveals that for three input trees Hybroscale is on mean average about 100 times faster than PIRN, for four input trees on mean average about 80 times, and for five input trees on mean average about 80 times.

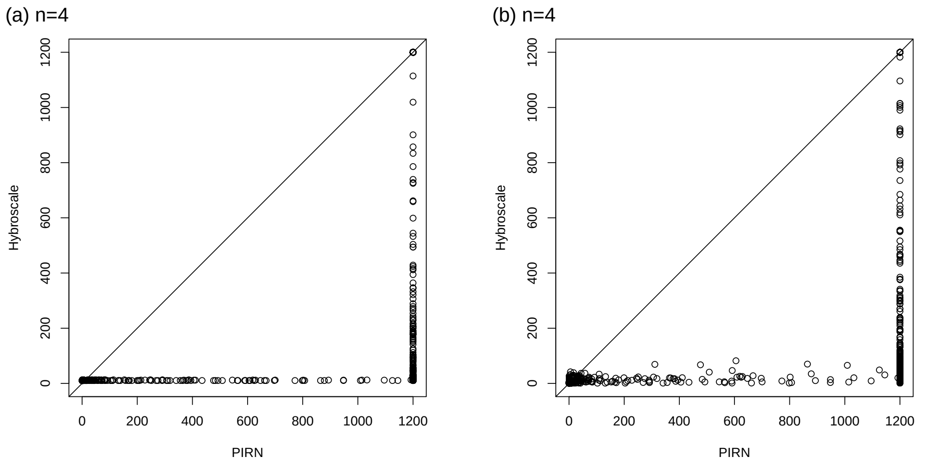


Figure 3.29: A scatterplot of the runtimes generated by PIRN (x-axis) against the runtimes generated by HYBROSCALE (y-axis) of all 810 data sets consisting of *four* input trees. The plot is generated for the real-time **(a)** and the user-time **(b)** of HYBROSCALE. Note that PIRN is not able to compute the result for 565 tree sets corresponding to each dot in the figure whose x-value is 1200. From those tree sets just 87 could not be computed by HYBROSCALE.

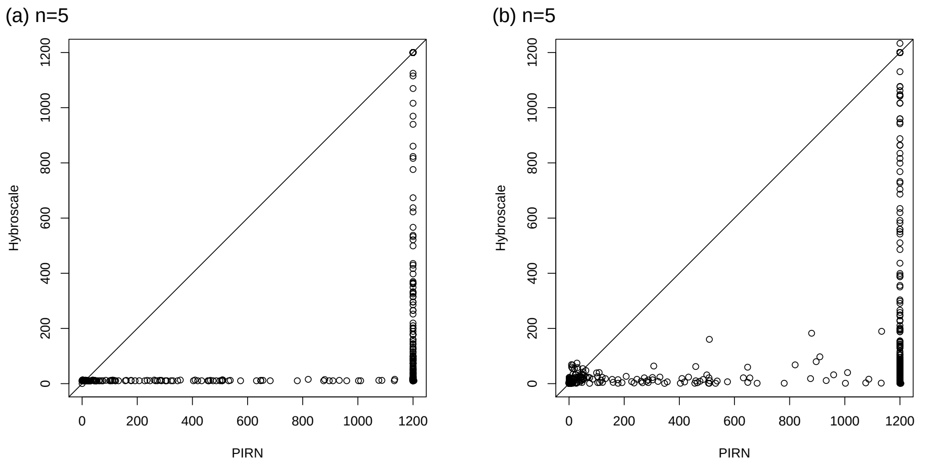


Figure 3.30: A scatterplot of the runtimes generated by PIRN (x-axis) against the runtimes generated by HYBROSCALE (y-axis) of all 810 data sets consisting of *five* input trees. The plot is generated for the real-time **(a)** and the user-time **(b)** of HYBROSCALE. Note that PIRN is not able to compute the result for 624 tree sets corresponding to each dot in the figure whose x-value is 1200. From those tree sets just 201 could not be computed by HYBROSCALE.

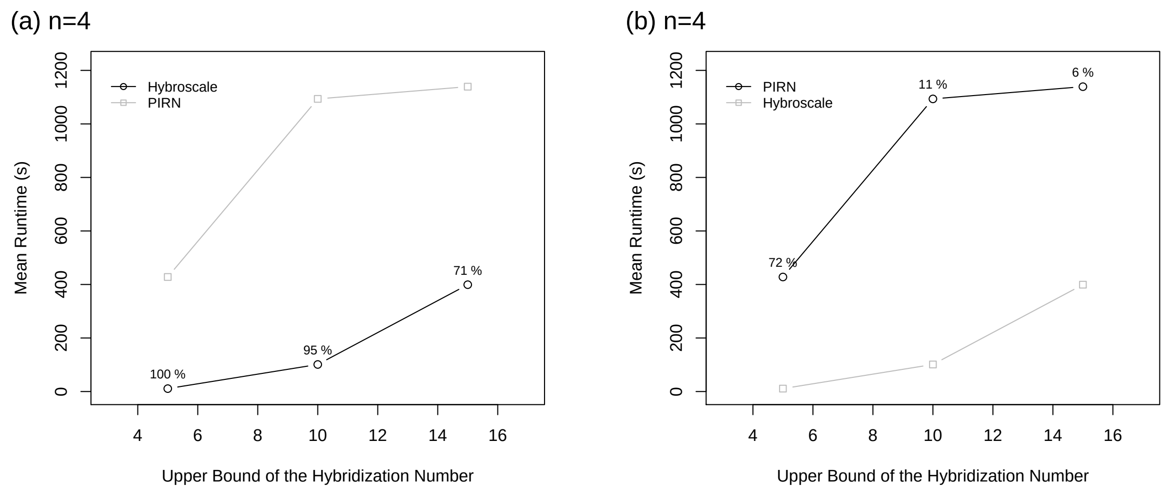


Figure 3.31: Figure **(a)** on the left hand side corresponds to Hybroscale and figure **(b)** on the right hand side to PIRN. The two figures show the average runtime grouped by parameter k, the upper bound of the hybridization number. Each percentage indicates the proportion of tree sets which could be computed within the time limit of 20 minutes.

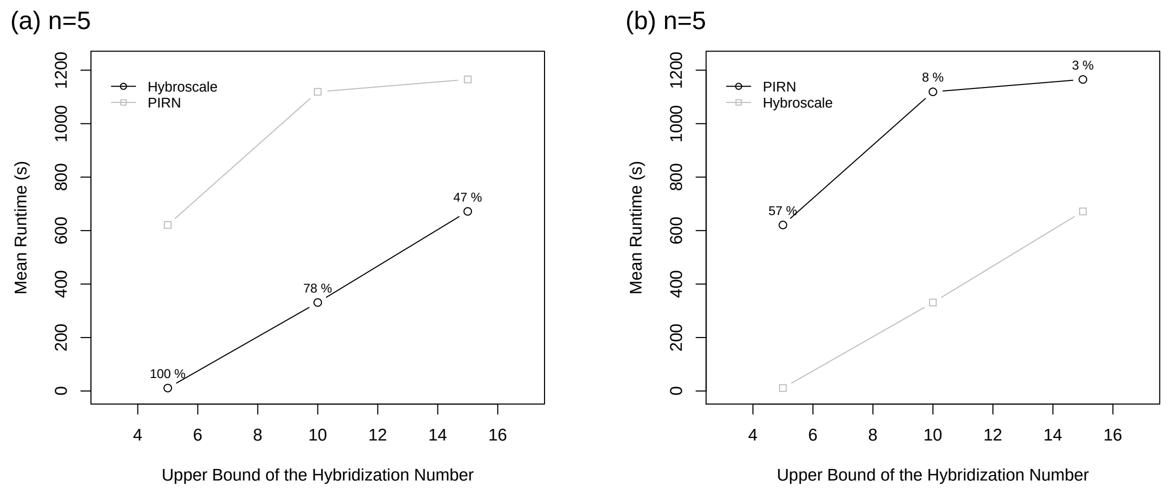


Figure 3.32: Figure **(a)** on the left hand side corresponds to Hybroscale and figure **(b)** on the right hand side to PIRN. The two figures show the average runtime grouped by parameter k, the upper bound of the hybridization number. Each percentage indicates the proportion of tree sets which could be computed within the time limit of 20 minutes.

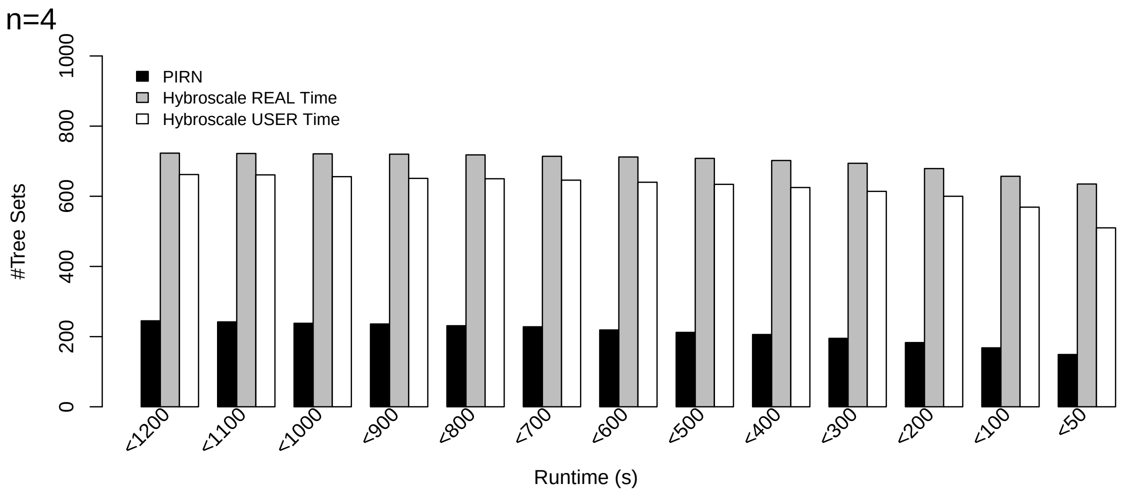


Figure 3.33: The figure shows the number of tree sets that could be computed within the runtime given at the x-axis by considering the real-runtime of PIRN and HYBROSCALE and, additionally, the user-runtime of HYBROSCALE.

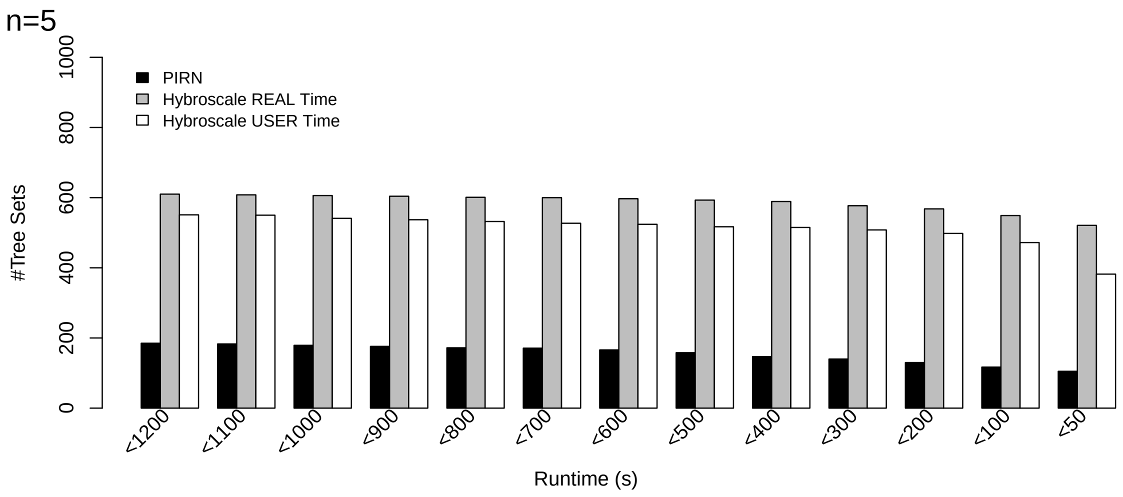


Figure 3.34: The figure shows the number of tree sets that could be computed within the runtime given at the x-axis by considering the real-runtime of PIRN and HYBROSCALE and, additionally, the user-runtime of HYBROSCALE.

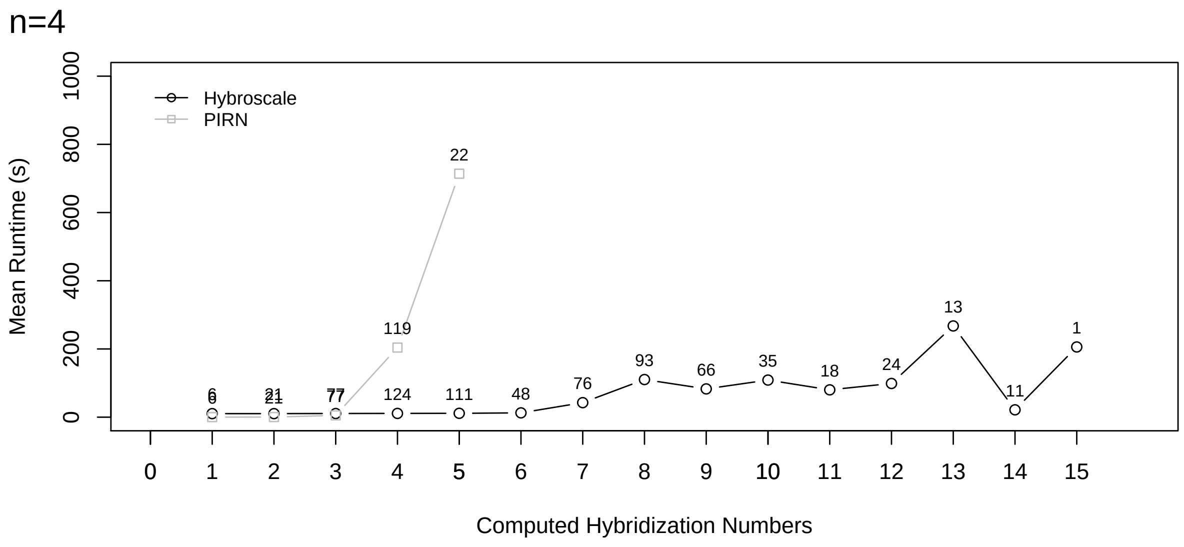


Figure 3.35: The figure shows the average runtime of all tree sets grouped by the computed hybridization numbers. The numbers inside the plot indicate how many tree sets could be computed for the corresponding hybridization number within the time limit. Note that for the hybridization numbers 0 to 3 this could be achieved by both programs for all corresponding tree sets.

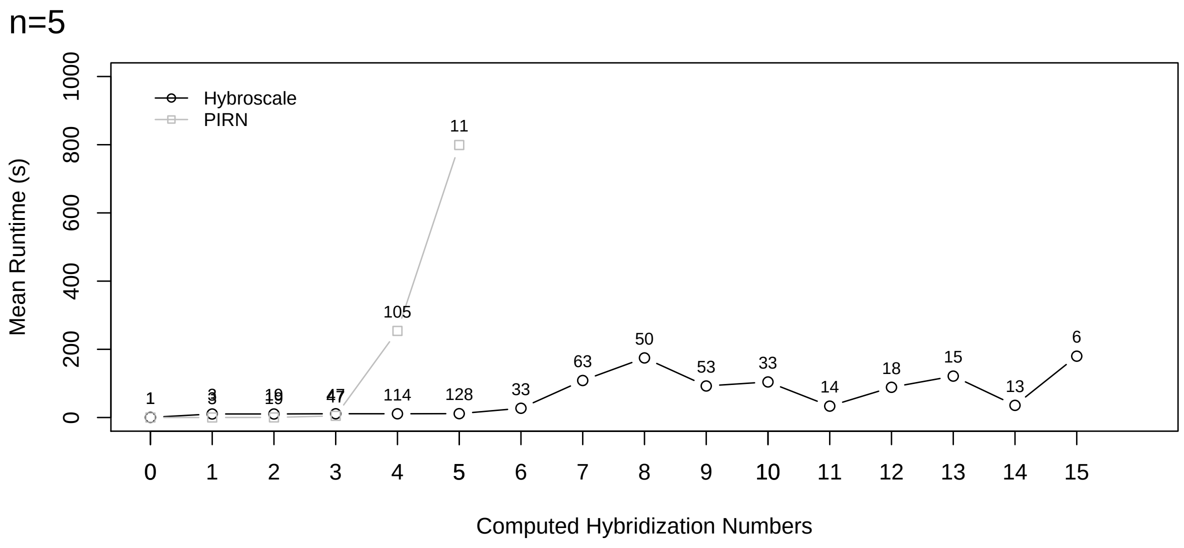


Figure 3.36: The figure shows the average runtime of all tree sets grouped by the computed hybridization numbers. The numbers inside the plot indicate how many tree sets could be computed for the corresponding hybridization number within the time limit. Note that for the hybridization numbers 0 to 3 this could be achieved by both programs for all corresponding tree sets.

Table 3.3: Output produced by HYBROSCALE applied to *two* phylogenetic trees belonging to a well known grass (*Poaceae*) dataset.

Genes	Taxa	HNumber	#MAAFs	#HNetworks
ndhf phyB	40	8	459	2079
ndhf rbcl	36	8	72	1488
ndhf rpoc	34	9	144	264
ndhf waxy	19	6	46	599
phyB its	30	8	21	195
phyB rbcl	21	4	4	6
phyB rpoc	21	4	5	9
phyB waxy	14	3	6	10
rbcl rpoc	26	7	18	111
rbcl waxy	12	4	10	84
rpoc its	31	12	12	3480
rpoc waxy	10	2	1	1
waxy its	15	5	6	15

3.5.5.2 Application to a grass dataset

As already mentioned above, our algorithm computes all representative networks for a set of input trees. In particular, given only two input trees, this means that HYBROSCALE in general outputs multiple networks for each maximum acyclic agreement forest instead of only one as it is the case for the method described in the work of Albrecht *et al.* [6] previously presented in Section 2.2. As a consequence, the output usually consists of a huge number of different hybridization networks, which is demonstrated by Table 3.3 and 3.4 presenting the results of our software HYBROSCALE applied to a well known grass (*Poaceae*) dataset[2] consisting of three nuclear loci and three chloroplast genes. This dataset, which is also used in the work of van Iersel *et al.* [66], was originally published by the Grass Phylogeny Working Group [27] and reanalyzed by Schmidt [55].

Again, we ran HYBROSCALE on a grid computer providing 16 cores and 40 GB RAM for each tree set within the grass dataset and summarized the results in Table 3.4. This table shows that HYBROSCALE is able to calculate the hybridization number for 51 out of 57 tree sets. This means, in particular, that for 6 tree sets HYBROSCALE cannot produce a result within a time limit of 80 minutes. Moreover, even though for 3 tree sets the hybridization number could successfully be calculated, the respective set of representative networks could not. Consequently, this biological example demonstrates that, although our algorithm seems to be faster than all so far existing methods, calculating minimum hybridization networks remains a computationally hard problem, which is still not solved sufficiently.

In Figure 3.22, one out of 324 possible hybridization networks reconciling four different binary phylogenetic trees corresponding to the sequences *ndhf*, *rpoC*, *waxy*, and *ITS* is

[2] `www.sites.google.com/site/cassalgorithm/data-sets`

given. The embedding of the trees is demonstrated by four colors blue, red, green, and orange. This means, for example, that we can simply determine the embedding of the tree corresponding to *rpoC*, which is denoted as *Tree 1* in this case, by taking the red colored edges into account. Moreover, the support values assigned to each hybridization node reveal that a hybridization event involving the two species *oryza* and *lygeum* occurs in 97% of all 324 networks, which could be a strong signal that this event is also part of the true underlying evolutionary history. However, the reader should be aware of the fact that there still exist other mechanisms explaining such inconsistencies, as for example incomplete lineage sorting. Hence, such networks just help to build hypothesis that still have to be tested by applying further experiments.

Table 3.4: Output produced by HYBROSCALE applied to phylogenetic trees belonging to a grass (*Poaceae*) dataset. Each runtime given in this table is stated in seconds. A missing result for a certain tree set means that our software HYBROSCALE could not compute the *exact* hybridization number (resp. set of representative networks) within 80 minutes.

		Computing HNumbers		Computing HNetworks	
Genes	#Taxa	HNumber	Runtime	#HNetworks	Runtime
ndhf its	46	17	3.262	554736	836.678
ndhf phyB	40	8	0.199	2079	33.035
ndhf rbcl	36	8	0.175	1488	32.1
ndhf rpoc	34	9	0.197	264	5.353
ndhf waxy	19	6	0.179	599	5.693
phyB its	30	8	0.238	195	8.304
phyB rbcl	21	4	0.083	6	1.65
phyB rpoc	21	4	0.091	9	1.678
phyB waxy	14	3	0.071	10	1.615
rbcl its	29	12	4.41	-	-
rbcl rpoc	26	7	0.147	111	3.836
rbcl waxy	12	4	0.126	84	4.338
rpoc its	31	12	4.5	3480	217.575
rpoc waxy	10	2	0.07	1	1.582
waxy its	15	5	0.118	15	2.712
ndhf phyB its	30	13	243.411	-	-
ndhf phyB rbcl	21	9	7.226	10889	55.684
ndhf phyB rpoc	21	8	6.189	36948	206.114
ndhf phyB waxy	14	4	1.599	42	2.87
ndhf rbcl its	28	-	-	-	-
ndhf rbcl rpoc	26	11	7.223	36600	444.027
ndhf rbcl waxy	12	5	4.198	114	5.577
ndhf rpoc its	31	17	1130.732	39016	4721.959
ndhf rpoc waxy	10	3	2.583	14	2.632
ndhf waxy its	15	8	4.213	5466	26.697
phyB rbcl its	17	8	9.437	8661	233.768
phyB rbcl rpoc	15	6	4.652	40	4.867
phyB rbcl waxy	7	2	2.568	11	2.592
phyB rpoc its	19	7	3.774	57	4.633
phyB rpoc waxy	5	0	0.045	1	0.075
phyB waxy its	10	4	3.122	146	3.844
rbcl rpoc its	24	-	-	-	-
rbcl rpoc waxy	9	3	1.585	5	1.62
rbcl waxy its	11	6	6.224	63	7.49
rpoc waxy its	10	4	2.626	4	2.635
ndhf phyB rbcl its	17	-	-	-	-
ndhf phyB rbcl rpoc	15	9	224.934	1079	360.922
ndhf phyB rbcl waxy	7	2	2.581	1	2.594
ndhf phyB rpoc its	19	9	984.937	-	-
ndhf phyB rpoc waxy	5	0	0.056	1	0.071
ndhf phyB waxy its	10	5	4.159	4709	26.434
ndhf rbcl rpoc its	24	-	-	-	-
ndhf rbcl rpoc waxy	9	4	3.165	396	14.864
ndhf rbcl waxy its	11	6	54.399	2	159.99
ndhf rpoc waxy its	10	5	4.213	324	16.663
phyB rbcl rpoc its	14	-	-	-	-
phyB rbcl rpoc waxy	4	0	0.057	1	0.06
phyB rbcl waxy its	6	2	2.574	3	2.589
phyB rpoc waxy its	5	0	0.064	1	0.065
rbcl rpoc waxy its	9	5	7.205	335	38.471
ndhf phyB rbcl rpoc its	14	-	-	-	-
ndhf phyB rbcl rpoc waxy	4	0	0.066	1	0.084
ndhf phyB rbcl waxy its	6	3	4.232	135	22.506
ndhf phyB rpoc waxy its	5	0	0.059	1	0.083
ndhf rbcl rpoc waxy its	9	5	35.899	235	587.54
phyB rbcl rpoc waxy its	4	0	0.066	1	0.076
ndhf phyB rbcl rpoc waxy its	4	0	0.062	1	0.083

3.5.6 Conclusion

As already discussed in in Section 2.2, it makes sense to consider hybridization if there is a significant difference between certain gene trees and if other effects, as for example incomplete lineage sorting, could be excluded. The number of genes affected by hybridization, however, is of course not limited to a fixed value, e.g., two, and, thus, a method computing hybridization networks for an arbitrary number of input trees is of high interest.

While some approaches only focus on reconciling two binary phylogenetic $\mathcal{X}$-trees [6, 17], in this section, we present an algorithm that is able to cope with multiple input trees. Moreover, instead of reporting just the hybridization number or only a small fraction of all possible hybridization networks, our approach contains the first algorithm that is able to output all representative networks, which is an important feature enabling the computation of meaningful support values indicating which of the computed hybridization events might have played an important role during evolution. Additionally, in combination with our software HYBROSCALE, we improve the interpretation of the reported hybridization networks by assigning certain support values to each hybridization node and by highlighting the embedding of all input trees.

Moreover, we a carried out a simulation study indicating that our algorithm is much faster than the only so far existing software PIRNV2.0 [72, 73] calculating the exact hybridization number for more than two rooted binary phylogenetic trees on the same set of taxa. As shown in Figure 3.27, the better performance is not only due to parallelization but consequently also due to algorithmic issues.

Chapter 4

Hybridization networks for nonbinary trees

In this chapter, we first present the algorithm allMulMAFs calculating all *relevant* maximum agreement forests for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Next, we present a tool that can be used to further modify the output of this approach such that now all relevant maximum *acyclic* agreement forests can be calculated. Finally, we present the algorithm allMulHNetworks that enables the computation of minimum hybridization networks for a set of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees by making use of such relevant maximum acyclic agreement forests.

4.1 Further definitions

In this section, we give a definition of those (nonbinary) agreement forests we consider as being relevant as well as a definition of certain trees reflecting a given agreement forest.

Forests. Let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}$. Then, by $\overline{\mathcal{F}}$, we refer to the forest that is obtained from $\mathcal{F}$ by deleting each element only consisting of an isolated node.

Relevant agreement forests. Let $\mathcal{F}$ be an agreement forest for two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2 and let E be a subset of edges in $\mathcal{F}$. Then, by $\mathcal{F} \ominus E$ we refer to the forest $\mathcal{F}'$ that is obtained from $\mathcal{F}$ by contracting each edge in E. Furthermore, we say that $\mathcal{F}$ *is relevant*, if there does not exist an edge e in $\mathcal{F}$ such that $\mathcal{F} \ominus \{e\}$ is still an agreement forest for T_1 and T_2. Otherwise, if such an edge e exists, we say that *e is contractible* in $\mathcal{F}$.

Trees reflecting agreement forests. Let $\mathcal{F} = \{F_\rho, F_1, F_2, \dots, F_{k-1}\}$ be an agreement forest for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, then, a tree $T_i(\mathcal{F})$ for $i \in \{1, 2\}$ corresponds to the tree T_i reflecting each component in $\mathcal{F}$ (cf. Fig. 4.1). Generally speaking, in such a tree some of its nodes are resolved such that the definition of an agreement forest can be applied in terms of the resulting tree and $\mathcal{F}$. Technically speaking, such a tree $\hat{T}_i = T_i(\mathcal{F})$ satisfies the following two properties.

(1) Each component F_j in $\mathcal{F}$ refers to a restricted subtree of $\hat{T}_i|_{\mathcal{L}(F_j)}$.

(2) All trees in $\{\hat{T}_i(\mathcal{L}(F_j))|j \in \{\rho, 1, \ldots, k-1\}\}$ are node disjoint subtrees in $\hat{T}_i$.

We can construct such a tree $T_i(\mathcal{F})$ by reattaching the components of $\mathcal{F}$ back together in a specific way as follows. Let $\Pi_{\mathcal{F}} = (F_0, F_1, \ldots, F_k)$, with $F_0 = F_\rho$, be an acyclic ordering that can be obtained from $AG(T_i, T_i, \mathcal{F})$ as discussed above. Notice that, as this graph is based only on one of both trees, this graph cannot contain any directed cycles and, thus, $\Pi_{\mathcal{F}}$ always exists. Now, each of those components in $\Pi_{\mathcal{F}}$, beginning with F_1, is added sequentially to a growing tree T^* (initialized with F_0) as follows.

(i) Let $\mathcal{X}_{<m}$ be the union of each taxa set corresponding to each component F_l in $\Pi_{\mathcal{F}}$ with $l < m$, i.e., $\mathcal{X}_{<m} = \bigcup_l^{m-1} \mathcal{L}(F_l)$, and let $\mathcal{X}_m$ be the taxa set corresponding to F_m. Moreover, let $P_m = (v_0^m, v_1^m, \ldots, v_n^m)$ be those nodes lying on the path connecting the node $v_0^m = \mathrm{LCA}_{T_i}(\mathcal{X}_m)$ and the root v_n^m of T_i such that v_q^m, with $q \in \{1, \ldots, n\}$, is the parent of v_{q-1}^m. Then,
$$v' = \min_q \{v_q^m : v_q^m \in P \wedge \mathcal{L}(v_q^m) \cap \mathcal{X}_{<m} \neq \emptyset\}.$$

(ii) Let $\mathcal{X}'$ be the set of taxa corresponding to the leaf set of $T_i(v')$ restricted to $\mathcal{X}_{<m}$. Notice that, due to the definition of v', this set $\mathcal{X}'$ is not empty. Moreover, based on $\mathcal{X}'$, let v^* be the node in T^* corresponding to $\mathrm{LCA}_{T^*}(\mathcal{X}')$.

(iii) Now, given v^*, the component F_m is added to T^* by connecting its root node ρ_m to the in-edge of v^*. More precisely, first a new node x is inserted into the in-edge of v^* and then ρ_m is connected to x by inserting a new edge (x, ρ_m).

Notice that, since there can exist multiple acyclic orderings for an acyclic agreement forest $\mathcal{F}$, the tree $T_i(\mathcal{F})$ is in general not unique.

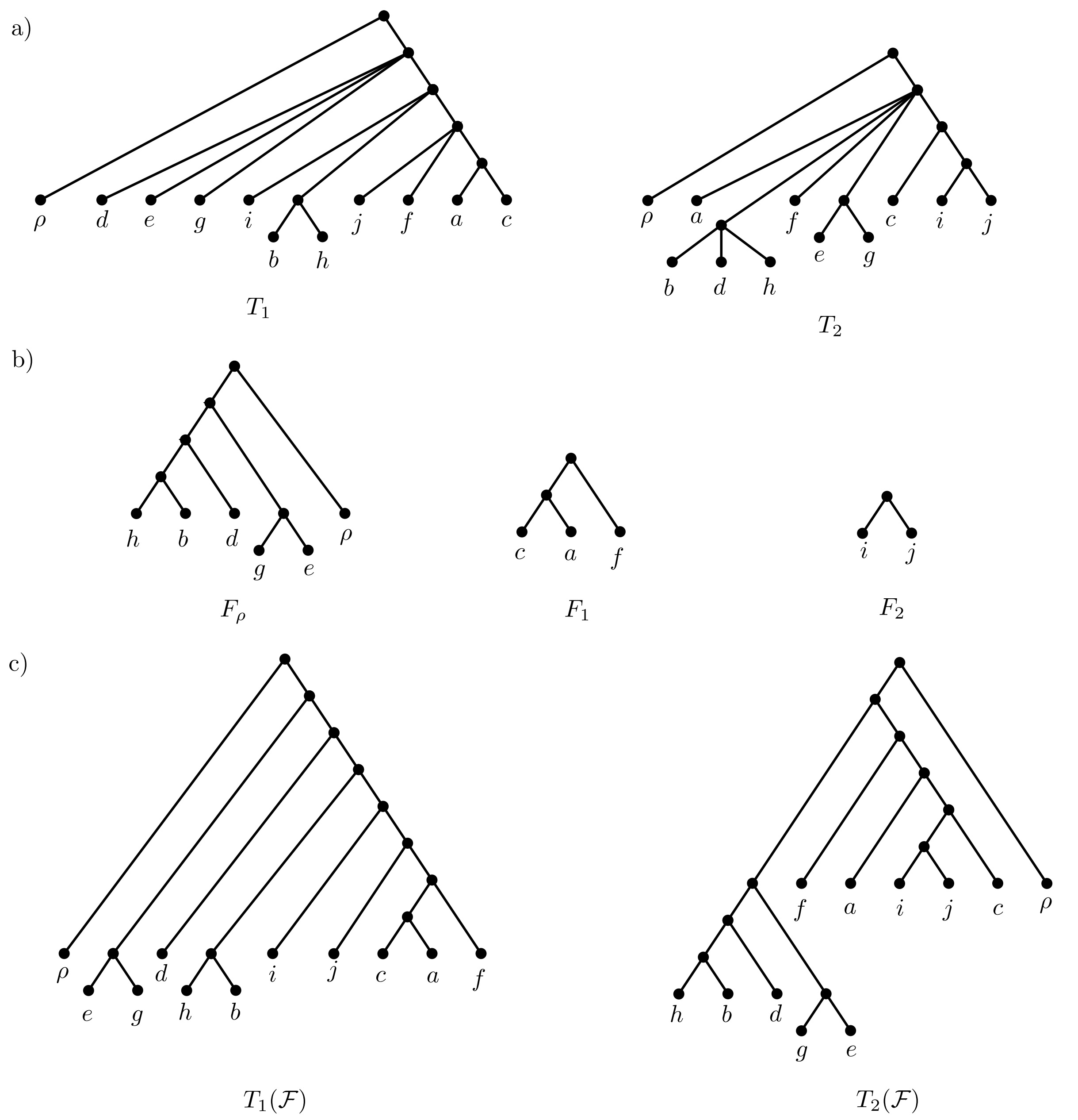

a)
ρ d e g i b h j f a c
T_1
ρ a b d h f e g c i j
T_2
b)
h b d g e ρ
F_ρ
c a f
F_1
i j
F_2
c)
ρ e g d h b i j c a f
$T_1(\mathcal{F})$
h b d g e f a i j c ρ
$T_2(\mathcal{F})$

Figure 4.1: (a) Two rooted nonbinary phylogenetic $\mathcal{X}$-trees T_1 and T_2. (b) An agreement forest $\mathcal{F}$ for T_1 and T_2. (c) Two trees $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ both reflecting $\mathcal{F}$ and being calculated in terms of the acyclic ordering (F_ρ, F_2, F_1) and (F_ρ, F_1, F_2), respectively.

4.2 The algorithm allMulMAFs

In this section, we show how to modify the algorithm ALLMAAFS presented in Section 2.3 such that the output consists of all relevant maximum agreement forests for two rooted *nonbinary* phylogenetic $\mathcal{X}$-trees. Similar to the algorithm ALLMAAFS, this algorithm is again based on processing common and contradicting cherries. In order to cope with nonbinary nodes, however, now for an internal node one has to consider more than one cherry and, before cutting a particular set of edges, one first has to resolve some nonbinary nodes. Furthermore, in respect to the definition of relevant maximum agreement forests, when expanding contracted nodes one has to take care on not generating any contractible edges.

In the following, we will first introduce some further notations necessary for describing the algorithm ALLMULMAFS. Moreover, we give a detailed formal proof establishing the correctness of the algorithm, which means, in particular, that we will show that the algorithm calculates all relevant maximum agreement forests for two rooted *nonbinary* phylogenetic $\mathcal{X}$-trees. Finally, we end this section by discussing its theoretical worst-case runtime.

4.2.1 Notations

Before going into details, we have to give some further notations that are crucial for the following description of the algorithm.

Removing leaves. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R, a *leaf ℓ is removed* by first deleting its in-edge and then by suppressing its parent p, if, after ℓ has been deleted, p has out-degree 1.

Cherries. Let R be a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree and let ℓ_a and ℓ_c be two of its leaves that are adjacent to the same parent node p and labeled by taxon a and c, respectively. Then, we call the set consisting of the two taxa $\{a, c\}$ a *cherry of R*, if the children of p are all leaves. Now, let $\{a, c\}$ be a cherry of R and let $\mathcal{F}$ be a forest on a taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, we say $\{a, c\}$ is a *contradicting cherry of R and $\mathcal{F}$*, if $\mathcal{F}$ does not contain a tree containing $\{a, c\}$. Otherwise, if such a tree exists in $\mathcal{F}$, the cherry $\{a, c\}$ is called a *common cherry of R and $\mathcal{F}$*.

Remark 1. The definition of a cherry is equal to both previous definitions given Section 2.3.2 and 2.4.1. Again, a cherry is a set of two taxa a and c corresponding to two leaves ℓ_a and ℓ_c, respectively. However, in contrast to those previous definitions, now for describing a cherry we explicitly refer to these two taxa. This means, in particular, that in this chapter instead of writing $\{\mathcal{L}(\ell_a), \mathcal{L}(\ell_c)\}$ we will write $\{a, c\}$.

Contracting cherries. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R, a cherry $\{a, c\}$ of R can be contracted in two different ways (cf. Fig. 4.2). Either, if the two leaves

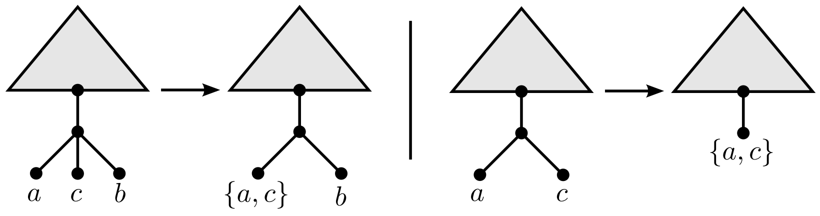


Figure 4.2: Two different ways of contracting a cherry $\{a, c\}$ depending on its number of siblings which is one on the left hand side and zero on the right hand side and .

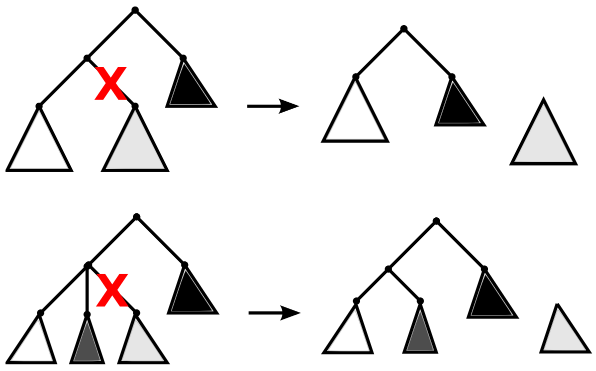


Figure 4.3: Two different ways of cutting an edge depending on the out-degree of its source node which is 2 at the top and 3 at the bottom.

ℓ_a and ℓ_c labeled by a and c, respectively, are the only children of its parent node p, first the in-edge of both nodes is deleted and then the label of p is set to $\{a, c\}$. Otherwise, if the two leaves ℓ_a and ℓ_c contain further siblings and, thus, its parent p has an out-degree larger than 2, just the in-edge of ℓ_a is deleted and the label of ℓ_c is replaced by $\{a, c\}$.

Cutting edges. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree F, the in-edge e_v of a node v is cut as follows (cf. Fig. 4.3). First e_v is deleted and then its parent node p is suppressed, if, after the deletion of e_v, p has out-degree 1. Note that by cutting an edge in F, two rooted (nonbinary) phylogenetic trees with taxa set $\mathcal{X}'$ and $\mathcal{X}''$ are generated with $\mathcal{X}' \cup \mathcal{X}'' = \mathcal{X}$ and $\mathcal{X}' \cap \mathcal{X}'' = \emptyset$.

Moreover, let $\mathcal{F}$ be a set of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees and let E be a set of edges in which each edge e is part of a tree in $\mathcal{F}$. Then, in order to ease reading, we write $\mathcal{F} - E$ to denote the cutting of each edge $e \in E$ within its corresponding tree in $\mathcal{F}$.

Pendant edges. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree F and two leaves ℓ_a and ℓ_c labeled by taxon a and c, respectively, that are not adjacent to the same node. Then, the set of *pendant edges* E_B *for* a *and* c is based on a refinement of F, shortly denoted by $F[a \sim c]$, which is obtained from F as follows. Let $(\ell_a, v_1, v_2, \dots, v_n, \ell_c)$ be the path connecting the two leafs ℓ_a and ℓ_c in F. Then, each node $v \in \{v_1, v_2, \dots, v_n\}$ with $\delta^+(v) > 2$ is turned into a node of out-degree 2 as follows. First a new node w is created that is attached to v by inserting a new edge (v, w) and then each out-going edge (v, x)

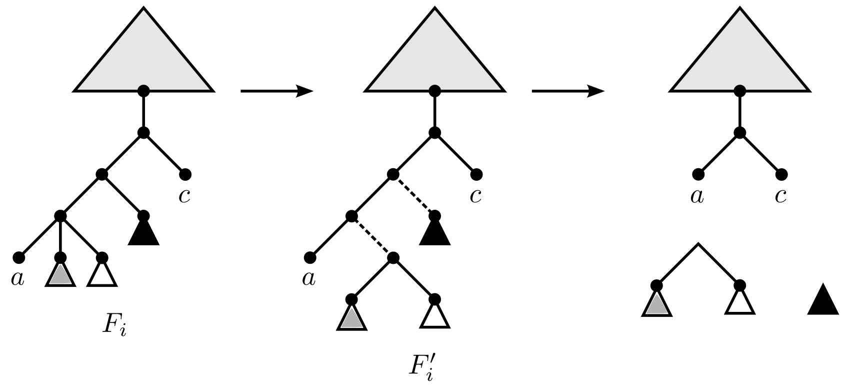


Figure 4.4: An illustration of Case 3b. First F_i is refined into $F'_i = F_i[a \sim c]$ and then each pendant subtree of the path connecting both nodes labeled by a and c, respectively, are cut. Note that each dashed edge of F'_i is part of the pendant edge set E_B for a and c.

with $x \notin \{w, \ell_a, v_1, v_2, \ldots, v_n, \ell_c\}$ is deleted followed by reattaching the node x to w by inserting a new edge (w, x). Now, regarding $F[a \sim c]$, let $(\ell'_a, v'_1, v'_2, \ldots, v'_n, \ell'_c)$ be the path connecting the two nodes ℓ'_a and ℓ'_c labeled by a and c, respectively. Then, E_B consists of each edge (u', v') with $u' \in \{v'_1, v'_2, \ldots, v'_n\}$ and $v' \notin \{\ell'_a, v'_1, v'_2, \ldots, v'_n, \ell'_c\}$ (cf. Fig. 4.4).

Moreover, given a forest $\mathcal{F}$ on $\mathcal{X}$ containing a tree F with two leaves ℓ_a and ℓ_c labeled by taxon a and c, respectively, that are not adjacent to the same node, then, by $\mathcal{F}[a \sim c]$ we refer to $\mathcal{F}$ in which F is replaced by $F[a \sim c]$.

Labeled nodes. Let R be a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree, then, by $\ell(R)$ we denote the number of its labeled nodes. Moreover, let $\mathcal{F}$ be a forest on $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R. Then, $\ell(\overline{\mathcal{F}})$ refers to the number of labeled nodes that are contained in each tree of $\overline{\mathcal{F}}$. Additionally, we write $\ell(R) \equiv \ell(\overline{\mathcal{F}})$, if $\ell(R)$ equals $\ell(\overline{\mathcal{F}})$ and if for each labeled node v_R in R there exists a labeled node v_F in $\overline{\mathcal{F}}$ such that $\mathcal{L}(v_R) = \mathcal{L}(v_F)$. Moreover, if $\ell(R) \equiv \ell(\overline{\mathcal{F}})$ holds, we say that a leaf ℓ in R *refers* to a leaf ℓ' in $\mathcal{F}$, if $\mathcal{L}(\ell)$ equals $\mathcal{L}(\ell')$.

4.2.2 The algorithm

In this section, we give a description of the algorithm allMulMAFs calculating a particular set of nonbinary maximum agreement forests for two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2. More specifically, as shown by an upcoming formal proof, this set consists of all relevant agreement forests for both trees. Before that, however, we want to give a remark emphasizing that the algorithm is based on a previous published algorithm, also presented here in Section 2.3, that solves a similar problem dealing with rooted binary phylogenetic $\mathcal{X}$-trees.

Remark 2. Our algorithm is an extension of the algorithm allMAAFs [57] computing all maximum *acyclic* agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees T_1 and T_2. Notice that the work of Scornavacca *et al.* [57] also contains a formal proof showing the correctness of the presented algorithm. The algorithm allMulMAFs presented here has a similar flavor and, thus, our notation basically follows the notation that has already been used for the description of the algorithm allMAAFs.

Broadly speaking, given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2 as well as a parameter $k \in \mathbb{N}$, our algorithm acts as follows. Based on the topology of the first tree T_1, the second tree T_2 is cut into several components until either the number of those components exceeds k or the set of components fulfills each property of an agreement forest for T_1 and T_2. To ensure that there does not exist an agreement forest consisting of less than k components, the following steps can be simply conducted by step-wise increasing parameter k beginning with $k = 0$. Thus, as far as our algorithm reports an agreement forest for T_1 and T_2 of size k, this agreement forest must be of minimum size and, hence, must be a maximum agreement forest for both input trees. In order to speed up computation, one can either set k to a lower bound calculated by particular approximation algorithms as, for instance, given in van Iersel *et al.* [65], or directly to the hybridization number calculated by applying less complex algorithms, e.g., the algorithm TerminusEst (cf. Sec. 2.5).

The algorithm allMulMAFs takes as input two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2 as well as a parameter $k \in \mathbb{N}$. If $k < h(T_1, T_2)$ holds, an empty set is returned. Otherwise, as we will show later in Section 4.2.3, if k is larger than or equal to $h(T_1, T_2)$, the output $\boldsymbol{\mathcal{F}}$ of allMulMAFs contains all relevant maximum agreement forests for T_1 and T_2. Throughout the algorithm three specific tree operations are performed on both input trees. Either a leaf is removed, subtrees are cut, or a common cherry is contracted.

The algorithm allMulMAFs contains a recursive subroutine, in which the input of each recursion consists of a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R, a forest $\mathcal{F}$ on some taxa set $\mathcal{X}'$ with $\overline{\mathcal{F}}$ being a forest for R, a parameter $k \in \mathbb{N}$, and a map M. This map M is necessary for undoing each cherry reduction that has been applied to each component of the resulting forest. For that purpose, M maps a set of taxa $\tilde{\mathcal{X}}$ to a triplet $(\mathcal{X}_1, \mathcal{X}_2, B)$ with $\mathcal{X}_1 \cup \mathcal{X}_2 = \tilde{\mathcal{X}}$, $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, and $B \in \{\top, \bot\}$, where B denotes the way of how a cherry is expanded (as discussed below). In order to ease reading, by $M[\tilde{\mathcal{X}}] \leftarrow B$ we refer to the operation on M mapping $\tilde{\mathcal{X}}$ to B. This means, in particular, if M already contains an element with taxa set $\tilde{\mathcal{X}}$ this element is replaced. For instance, if $M[\tilde{\mathcal{X}}] = (\mathcal{X}_1, \mathcal{X}_2, \top)$, by $M[\tilde{\mathcal{X}}] \leftarrow \bot$ the taxa set $\tilde{\mathcal{X}}$ is remapped to $(\mathcal{X}_1, \mathcal{X}_2, \bot)$ so that after this operation $M[\tilde{\mathcal{X}}] = (\mathcal{X}_1, \mathcal{X}_2, \bot)$.

Expanding agreement forests. The expansion of an agreement forest $\mathcal{F}$ is done by applying the following steps to $\mathcal{F}$. Choose a leaf v corresponding to a component F_i in $\mathcal{F}$ whose taxon $\mathcal{L}(v)$ is contained in M. Let $(\mathcal{X}_1, \mathcal{X}_2, B)$ be the triplet referring to $M[\mathcal{L}(v)]$, then, depending on $B \in \{\top, \bot\}$, one of the following two operations is performed as illustrated in Figure 4.5.

- If B equals $\bot$, replace v in F_i by first creating two new nodes w_1 and w_2 and then by labeling w_1 and w_2 by $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively. Finally, both nodes w_1 and w_2 are attached to v by inserting a new edge (v, w_1) and (v, w_2)

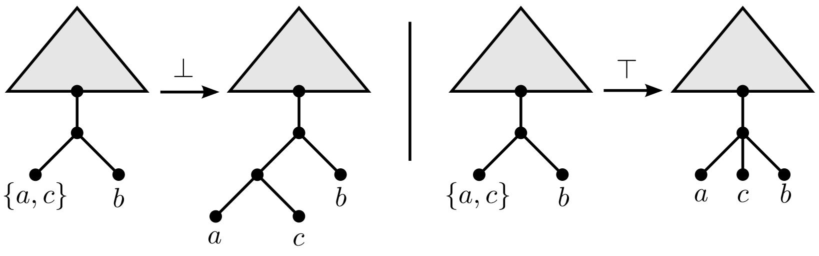


Figure 4.5: Expanding a cherry $\{a,c\}$ in respect of $\perp$ (left) and in respect of $\top$ (right).

- Otherwise, if B equals $\top$, replace v in F_i by first creating a new node w and then by labeling w and v by $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively. Finally, w is attached to the parent p of v by inserting a new edge (p,w).

These steps are repeated in an exhaustive way until each taxon of each leaf in $\mathcal{F}$ is not contained in M. As a result, the expanded forest corresponds to a nonbinary agreement forest for the two input trees T_1 and T_2. Notice that in the following, by saying a cherry is expanded *in respect of* $\perp$ or *in respect of* $\top$, we refer to one of both ways as described above.

In the following, a description of the recursive algorithm allMulMAFs is given. Here we assume that, at the beginning, R is initialized by T_1, $\mathcal{F}$ by $\{T_2\}$ and M by $\emptyset$, with T_1 and T_2 being two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Then, during each recursive call, $\overline{\mathcal{F}}$ is a forest for R with $\ell(R) \equiv \ell(\overline{\mathcal{F}})$ and, depending on the size of $\mathcal{F}$ (cf. Case 1a–c) and the choice of the next cherry that is selected from R (cf. Case 2 and 3), the following steps are performed.

Case 1a. If $\mathcal{F}$ contains more than k components, the computational path is aborted immediately and the empty set is returned.

Case 1b. If R only consists of a single leaf, each F_i in $\mathcal{F}$ is expanded as prescribed in M, and, finally, returned.

Case 1c. If there exists a specific leaf ℓ in R that refers to an isolated node in $\mathcal{F}$, this leaf ℓ is removed from R resulting in R'. Next, the algorithm branches into a new path by recursively calling the algorithm with R', $\mathcal{F}$, k, and M' corresponding to M where $\mathcal{L}(\ell)$ is re-mapped to $M[\mathcal{L}(\ell)] \leftarrow \perp$.

Otherwise, if such a leaf ℓ does not exist continue with Case 2.

Case 2. If there exists a common cherry $\{a,c\}$ of R and $\mathcal{F}$, the cherry $\{a,c\}$ is contracted in R and $\mathcal{F}$ resulting in R' and $\mathcal{F}'$. Second, the algorithm branches into a new path by recursively calling the algorithm with R', $\mathcal{F}'$, k, and M', where M' corresponds to M that has been updated as follows. If both parents of $\{a,c\}$ in R and $\mathcal{F}$ have out-degree ≥ 3, $\mathcal{L}(a) \cup \mathcal{L}(c)$ is mapped to $(\mathcal{L}(a), \mathcal{L}(c), \top)$, otherwise, $\mathcal{L}(a) \cup \mathcal{L}(c)$ is mapped to $(\mathcal{L}(a), \mathcal{L}(c), \perp)$.

Otherwise, if such a common cherry does not exist, continue with Case 3.

Case 3. If there does not exist a common cherry of R and $\mathcal{F}$, a node v in R whose children are all leaves is selected. Now, *for each* cherry $\{a, c\}$ of v, depending on the location of the leaves referring to a and c in $\mathcal{F}$, one of the following two cases is performed.

Case 3a. If $a \not\sim_{\mathcal{F}} c$ holds, and, thus, the leaves referring to a and c in $\mathcal{F}$ are located in two different components, the algorithm branches into two computational paths by recursively calling the algorithm by R, $\mathcal{F} - \{e_a\}$, k, and M' as well as R, $\mathcal{F} - \{e_c\}$, k, and M', where e_a and e_c correspond to the in-edge of the leaf of $\mathcal{F}$ referring to a and c, respectively, and M' is obtained from M as follows. Let p be the parent in $\mathcal{F}$ of the leaf referring to a (resp. c). If p has out-degree larger than 2, nothing is done. Otherwise, if p has out-degree 2, let ℓ be the sibling of the leaf labeled by a (resp. c). Then, if ℓ is a leaf M is updated so that $M' = M[\mathcal{L}(\ell)] \leftarrow \perp$.

Case 3b. If $a \sim_{\mathcal{F}} c$ holds, and, thus, in $\mathcal{F}$ both leaves ℓ_a and ℓ_c referring to a and c, respectively, are located in the same component F_i, the algorithm branches into the following three computational paths. First, similar to Case 3a, the algorithm is called by R, $\mathcal{F} - \{e_a\}$, k, and M' as well as R, $\mathcal{F} - \{e_c\}$ k, and M'. Second, a third computational path is initiated by calling the algorithm with R, $\mathcal{F}[a \sim c] - E_B$, k, and M'', where E_B refers to the set of pendant edges in $\mathcal{F}[a \sim c]$ and M'' is obtained from M as follows.

Let $(\ell_a, v_1, \dots, v_n, \ell_c)$ denote the path connecting ℓ_a and ℓ_c in $\mathcal{F}$. Then, M'' is obtained by updating M as follows. If v_1 does not correspond to $\mathrm{LCA}_{F_i}(\{a, c\})$, $\mathcal{L}(\ell_a)$ is remapped to $M[\mathcal{L}(\ell_a)] \leftarrow \perp$. Similarly, if v_n does not correspond to $\mathrm{LCA}_{F_i}(\{a, c\})$, $\mathcal{L}(\ell_c)$ is remapped $M[\mathcal{L}(\ell_c)] \leftarrow \perp$.

An illustration of this case is given in Figure 4.4

We end the description of the algorithm by noting that the algorithm allMulMAFs always terminates, since during each recursive call either the size of R decreases or the number of components in $\mathcal{F}$ increases. More precisely, the size of R is decreased by one either by deleting one of its leaves ℓ referring to an isolated node in $\mathcal{F}$ (cf. Case 1c) or by contracting a common cherry of R and $\mathcal{F}$ (cf. Case 2). If R is not decreased, at least one edge in $\mathcal{F}$ is cut (cf. Case 3) and, thus, its size increases at least by one. As each computational path of the algorithm stops if R only consists of an isolated node or if k edges have been cut, each recursive call does always make progress towards one of both abort criteria.

4.2.3 Correctness of allMulMAFs

In this section, we establish the correctness of the algorithm allMulMAFs. However, before doing so, we want to give an important remark emphasizing the relation of our algorithm allMulMAFs to the previously presented algorithm allMAAFs[1](cf. Sec. 2.4.3.1),

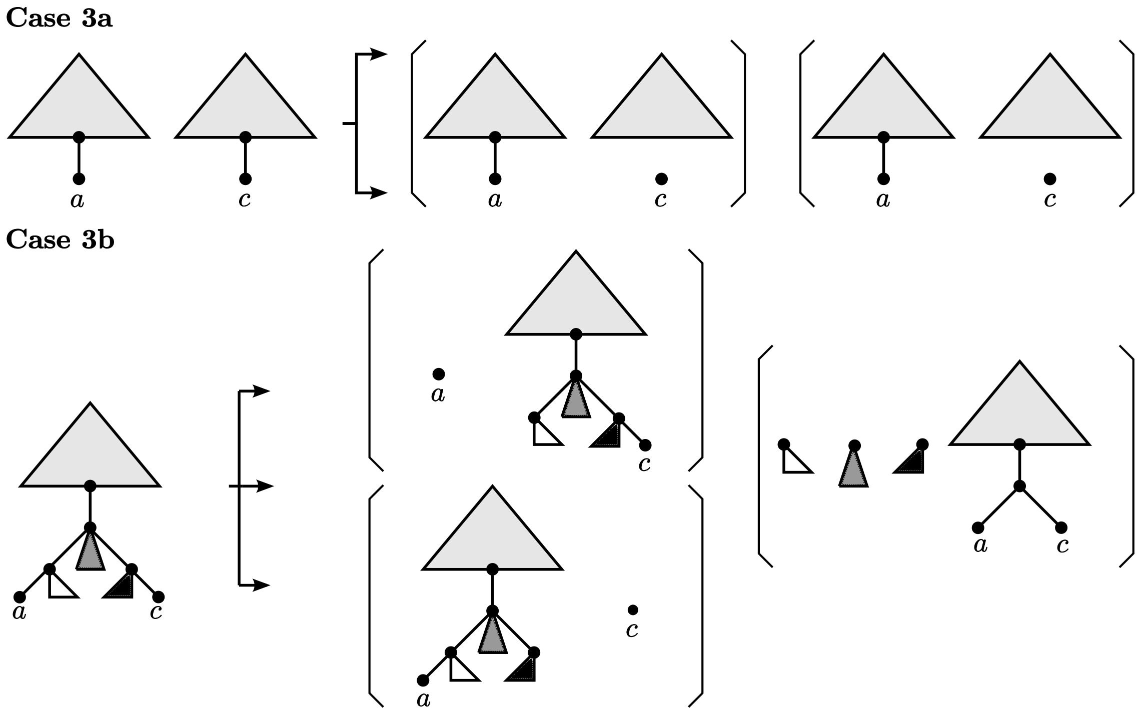


Figure 4.6: An illustration of Case 3a branching into two computational paths and Case 3b branching into three computational paths. Regarding Case 3b, one additional computational path is created in which all pendant subtrees lying on the path connecting the two nodes labeled by a and c, respectively, are cut.

which is a modification of the algorithm allMAAFs (cf. Sec. 2.3) improving the processing of contradicting cherries.

Remark 3. Given two binary phylogenetic $\mathcal{X}$-trees, the algorithm allMulMAFs processes an ordered set of cherries in the same way as the algorithm allMAAFs[1] omitting its acyclic check (henceforth denoted as allMAFs[1]) testing an agreement forests for acyclicity. This means, in particular, that our algorithm allMulMAFs is simply an extension of the algorithm allMAFs[1] that is now able to handle nonbinary trees, but for binary trees still acts in the same way.

As a consequence of Remark 3, the upcoming proof showing the correctness of allMulMAFs refers to the correctness of allMAAFs[1] calculating all maximum acyclic agreement forests for two rooted binary phylogenetic $\mathcal{X}$-trees [4, Theorem 2]. In a first step, however, in order to ease the understanding of our proof, we will introduce a connective element between both algorithms, which is a modified version of our original algorithm — called allMulMAFs* — processing types of cherries that are not considered by computational paths corresponding to allMulMAFs.

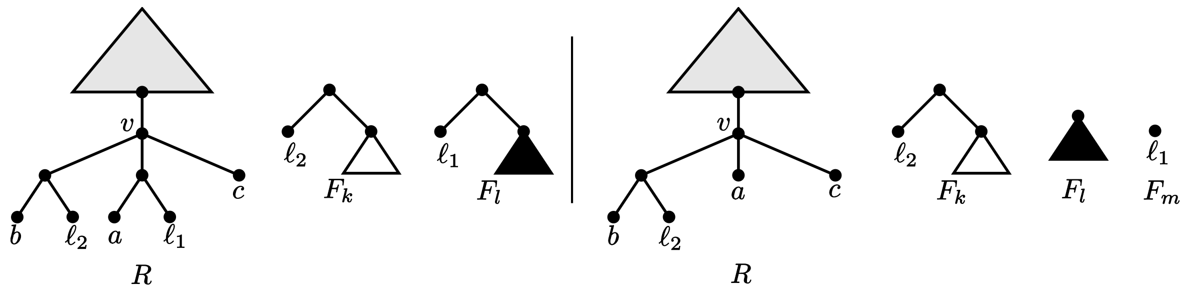


Figure 4.7: (left) An illustration of a pseudo cherry $\{a, c\}$. (right) The result of preparing the pseudo cherry $\{a, c\}$ given on the left hand side.

4.2.3.1 The algorithm allMulMAFs*

Before describing the algorithm, we have to add further definitions that are crucial for what follows.

Proper leaves. Given a leaf ℓ of a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R labeled with taxon a as well as a forest $\mathcal{F}$ on some taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R, ℓ is called a *proper leaf of* R and $\mathcal{F}$, if the corresponding leaf in $\mathcal{F}$ labeled by taxon a is a child of some root.

Pseudo cherries. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R as well as a forest $\mathcal{F}$ on some taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R, we call a set of two taxa $\{a, c\}$ a *pseudo cherry for* R *and* $\mathcal{F}$, if the following two properties hold. First for each child v of $\text{LCA}_R(\{a, c\})$ its leaf set $\mathcal{L}(T(v))$ of size n contains at least $n - 1$ proper leaves. Second, the path connecting the two leaves in R labeled by a and c contains at least one pendant proper leaf.

Preparing cherries. Given a rooted (nonbinary) phylogenetic $\mathcal{X}$-tree R, a forest $\mathcal{F}$ on some taxa set $\mathcal{X}'$ such that $\overline{\mathcal{F}}$ is a forest for R as well as a cherry $\{a, c\}$, then, $\{a, c\}$ *is prepared* as follows. If $\{a, c\}$ is not a pseudo cherry for R and $\mathcal{F}$, nothing is done. Else, the following two steps are conducted. First, each pendant proper leaf $\{\ell_1, \dots, \ell_n\}$ in R lying on the path connecting the two leaves labeled by a and c is removed. Second, the two nodes in $\mathcal{F}$ labeled by a and c are cut. Notice that, after preparing $\{a, c\}$, the node $\text{LCA}_R(\{a, c\})$ in R is the parent of the two leaves labeled by taxon a and c and each component in $\mathcal{F}$ referring to $\{\ell_1, \dots, \ell_n\}$ only consists of a single isolated node (cf. Fig. 4.7).

Now, similar to the original algorithm, the algorithm allMulMAFs* is called by the same four parameters R, $\mathcal{F}$, M and k. Given two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2, R is initialized by T_1, $\mathcal{F}$ by $\{T_2\}$ and M by $\emptyset$. Depending on the size of $\mathcal{F}$ (cf. Case 1a–c) and the choice of the next cherry that is selected from R (cf. Case 2), the following steps are performed.

Case 1a. If F contains more than k components, the computational path is aborted immediately and an empty set is returned.

Case 1b. If R only consists of a single leaf, each F_i in $\mathcal{F}$ is expanded with the help of M, and, finally, returned.

Case 1c. If there exists a specific leaf ℓ in R that refers to an isolated node in F, this leaf ℓ is removed from R resulting in R'. Next, the algorithm branches into a new path by recursively calling the algorithm with R', $\mathcal{F}$, k, and M' corresponding to M updated by $M[\mathcal{L}(\ell)] \leftarrow \perp$.

Otherwise, if such a leaf ℓ does not exist continue with Case 2.

Case 2. Select a subtree in R in which each pair of taxa either represents a cherry or a pseudo cherry. Now, *for each* (pseudo) cherry $\{a, c\}$, depending on the location of the leaves referring to a and c in $\mathcal{F}$, one of the following three cases is performed. In a first step, however, the chosen cherry $\{a, c\}$ is prepared as described above. Moreover, M is updated by $M[\mathcal{X}_i] \leftarrow \perp$, where $\mathcal{X}_i$ denotes the taxa set of each proper leaf that has been cut during the preceding preparation step.

Case 2a. If $\{a, c\}$ is a common cherry, $\{a, c\}$ is processed as described in Case 2 corresponding to the original algorithm ALLMULMAFS.

Case 2b. If $a \not\sim_{\mathcal{F}} c$ holds, and, thus, the leaves referring to a and c in $\mathcal{F}$ are located in two different components, $\{a, c\}$ is processed as described in Case 3a corresponding to the original algorithm ALLMULMAFS.

Case 2c. If $a \sim_{\mathcal{F}} c$ holds, and, thus, the leaves referring to a and c in $\mathcal{F}$ can be found in the same component F_i, $\{a, c\}$ is processed as described in Case 3b corresponding to the original algorithm ALLMULMAFS.

Notice that there are two main differences between the algorithm ALLMULMAFS* and the original algorithm ALLMULMAFS. First, a computational path corresponding to ALLMULMAFS* can additionally process pseudo cherries. Second, if during a recursive call R contains a common cherry $\{a, c\}$ as well as a contradicting cherry $\{a, b\}$, ALLMULMAFS* additionally branches into a computational path processing $\{a, b\}$. In the following, we will call such a cherry $\{a, b\}$ a *needless cherry* as we will show later that it can be neglected for the computation of maximum agreement forests.

4.2.3.2 The algorithm ProcessCherries

Lastly, we present a simplified version of the algorithm ALLMULMAFS* — called PROCESSCHERRIES — mimicking one of its computational by a *cherry list* $\bigwedge = (\wedge_1, \wedge_2, \ldots, \wedge_n)$, in which each of its elements $\wedge_i$ denotes a *cherry action*. Such a cherry action $\wedge_i =$

Algorithm 14: ProcessCherries($R, \mathcal{F}, (\wedge_1, \ldots, \wedge_n)$)

```
$M \leftarrow \emptyset$;
for $i = 1, \ldots, n$ do
    $(\{a,c\}, \phi_i) \leftarrow \wedge_i$;
    if $\{a,c\}$ is a cherry of $R$ or a pseudo cherry for $R$ and $\mathcal{F}$ then
        if $\{a,c\}$ is a pseudo cherry for $R$ and $\mathcal{F}$ then
            $(R, \mathcal{F}, M) \leftarrow$ prepare pseudo cherry $\{a,c\}$;
        if $\{a,c\}$ is a common cherry of $R$ and $\mathcal{F}$ and $\phi_i == \cup_{ac}$ then
            $(R, \mathcal{F}, M) \leftarrow$ contract cherry $\{a,c\}$;
        else if $\{a,c\}$ is a contradicting cherry of $R$ and $\mathcal{F}$ and $\phi_i == \nmid_a$ then
            $e_a \leftarrow$ in-edge of node labeled by $a$ in $\mathcal{F}$;
            $(R, \mathcal{F}, M) \leftarrow$ cut edge $e_a$ in $\mathcal{F}$;
        else if $\{a,c\}$ is a contradicting cherry of $R$ and $\mathcal{F}$ and $\phi_i == \nmid_c$ then
            $e_c \leftarrow$ in-edge of node labeled by $c$ in $\mathcal{F}$;
            $(R, \mathcal{F}, M) \leftarrow$ cut edge $e_c$ in $\mathcal{F}$;
        else if $\{a,c\}$ is a contradicting cherry of $R$ and $\mathcal{F}$ then
            $\mathcal{F} \leftarrow \mathcal{F}[a \sim c]$;
            $E_B \leftarrow$ set of pendant edges for $a$ and $c$ in $\mathcal{F}$;
            $(R, \mathcal{F}, M) \leftarrow$ cut each edge in $E_B$;
        else
            return $(\emptyset)$;
        $(R, \mathcal{F}, M) \leftarrow$ from $R$ remove each leaf referring to an isolated node in $\mathcal{F}$;
    else
        return $(\emptyset)$;
$\mathcal{F} \leftarrow$ expand $\mathcal{F}$ as prescribed in $M$;
return $(\mathcal{F})$;
```

$(\{a,c\}, \phi_i)$ is a tuple that contains a (pseudo) cherry $\{a,c\}$ of the corresponding rooted phylogenetic $\mathcal{X}$-tree R_i and the forest $\mathcal{F}_i$ as well as a variable $\phi_i \in \{\cup_{ac}, \nmid_a, \nmid_c, \cap_{ac}\}$ denoting the way $\{a,c\}$ is processed in iteration i. More precisely,

- $\cup_{ac}$ refers to contracting the cherry $\{a,c\}$ following Case 2a of the algorithm allMulMAFs*.
- $\nmid_a$ and $\nmid_c$ refers to cutting taxon a and c, respectively, of the cherry $\{a,c\}$ following Case 2b of the algorithm allMulMAFs*.
- $\cap_{ac}$ refers to cutting each pendant subtree connecting taxon a and taxon c in $\mathcal{F}_i$ following Case 2c of the algorithm allMulMAFs*.

Now, given a cherry list $\bigwedge$, we say that $\bigwedge$ is a *cherry list for T_1 and T_2*, if in each iteration i the cherry $\{a,c\}$ of $\wedge_i = (\{a,c\}, \phi_i)$ is either contained in R_i or $\{a,c\}$ is a pseudo cherry for R_i and F_i. Moreover, after having prepared the cherry $\{a,c\}$, one of the following two conditions has to be satisfied.

- Either $\{a,c\}$ is a common cherry of R_i and F_i and $\phi_i = \cup_{ac}$,
- or $\{a,c\}$ is a contradicting cherry of R_i and F_i.

Notice that this is the case, if and only if calling ProcessCherries($T_1, \{T_2\}, \bigwedge$) does not return the empty set (cf. Alg. 14).

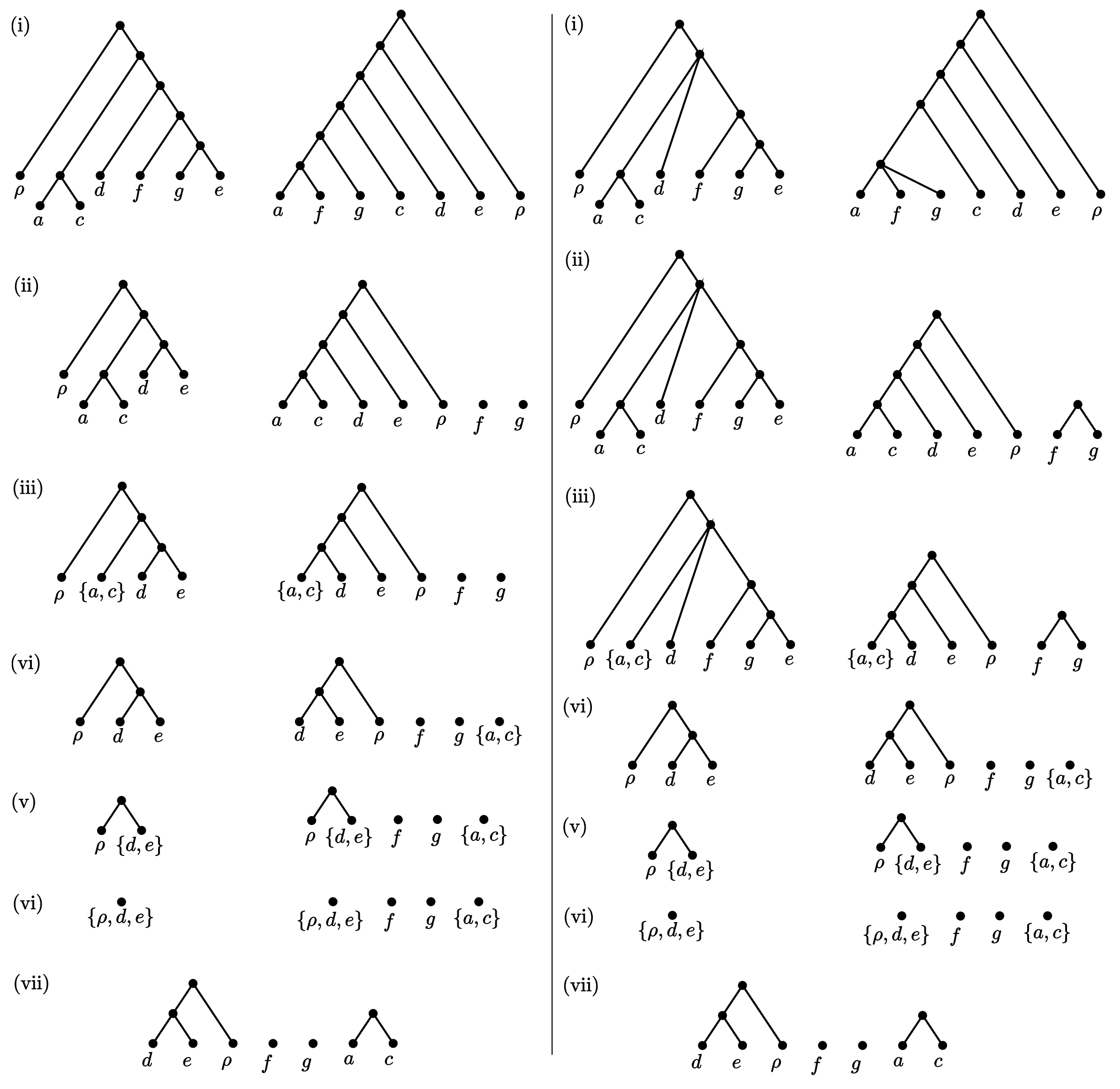


Figure 4.8: Two examples of calling ProcessCherries for two binary and nonbinary trees according to the cherry actions $(\{a,c\},\cap_{ac})$, $(\{a,c\},\cup_{ac})$, $(\{d,e\},\cap_{de})$, $(\{d,e\},\cup_{de})$, and $(\{\rho,\{d,e\}\},\cup_{\rho de})$ conducted in sequential order. Notice that, as the two binary trees are binary resolutions of the two nonbinary trees, the resulting forest on the left hand side is a binary resolution of the resulting forest on the right hand side. Moreover, regarding Step (iii) on the right hand side, notice that the chosen cherry $\{d,e\}$ is a pseudo cherry and, thus, in Step (iv) the two components consisting of the single nodes labeled by f and g, respectively, arise from preparing $\{d,e\}$.

4.2.3.3 Proof of Correctness

In this section, we will establish the correctness of the algorithm allMulMAFs by establishing the following theorem.

Theorem 20. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, by calling*

$$\textsc{allMulMAFs}(T_1, \{T_2\}, \emptyset, k)$$

all relevant maximum agreement forests for T_1 and T_2 are calculated, if and only if $k \geq h(T_1, T_2)$.

Proof. The proof of Theorem 20 is established in several substeps. First, given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, we will show that a binary resolution of each maximum agreement forest for T_1 and T_2 can be computed by applying the algorithm allMAFs^1 to a binary resolution of T_1 and T_2, where, as already mentioned, allMAFs^1 denotes a modification of the algorithm allMAAFs^1 omitting the acyclic check. Next, we will show that for an agreement forest $\mathcal{F}$ calculated by allMulMAFs there does not exist en edge e such that $\mathcal{F} \ominus \{e\}$ is still an agreement forest for T_1 and T_2, which directly implies that $\mathcal{F}$ is relevant. Moreover, we will show that, if a cherry list $\bigwedge$ for two binary resolutions $\hat{T}_1$ and $\hat{T}_2$ of T_1 and T_2, respectively, computes a maximum agreement forest $\hat{\mathcal{F}}$ for $\hat{T}_1$ and $\hat{T}_2$, $\bigwedge$ is mimicking a computational path of the algorithm allMulMAFs* calculating an agreement forest $\mathcal{F}$ for T_1 and T_2 such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$. Lastly, we will show that each maximum agreement forest $\mathcal{F}$ computed by allMulMAFs* is also computed by allMulMAFs.

Lemma 21. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, for each relevant maximum agreement forest $\mathcal{F}$ for T_1 and T_2 of size k there exists a binary resolution $\hat{\mathcal{F}}$ of $\mathcal{F}$ that is calculated by*

$$\textsc{allMAFs}^1(\hat{T}_1, \hat{T}_2, \hat{T}_1, \hat{T}_2, k),$$

where $\hat{T}_1$ and $\hat{T}_2$ refers to binary resolutions of T_1 and T_2, respectively.

Proof. First notice that the algorithm allMAAFs^1 without conducting the acyclic check computes all maximum agreement forest for two rooted binary phylogenetic $\mathcal{X}$-trees, which is a direct consequence from Lemma 6. Moreover, given a relevant maximum agreement forest $\mathcal{F}$ for T_1 and T_2, a binary resolution $\hat{\mathcal{F}}$ of $\mathcal{F}$ is automatically a maximum agreement forest corresponding to $T_1(\hat{\mathcal{F}})$ and $T_2(\hat{\mathcal{F}})$. This is, in particular, the case, since just by definition each component $\hat{F}$ in $\hat{\mathcal{F}}$ is a subtree of $T_1(\hat{\mathcal{F}})$ and $T_2(\hat{\mathcal{F}})$ and, as in $\mathcal{F}$ all components are edge disjoint subtrees in T_1 and T_2, this has to hold for each of its binary resolution as well. Furthermore, $\hat{\mathcal{F}}$ has to be of minimum cardinality, since, otherwise, $\mathcal{F}$ would not be a maximum agreement forest for T_1 and T_2. Consequently, by applying the algorithm allMAFs^1 to both trees $T_1(\hat{\mathcal{F}})$ and $T_2(\hat{\mathcal{F}})$ the maximum agreement forest $\hat{\mathcal{F}}$ is calculated if $k \geq |\mathcal{F}|$, which, finally, establishes the proof of Lemma 21. □

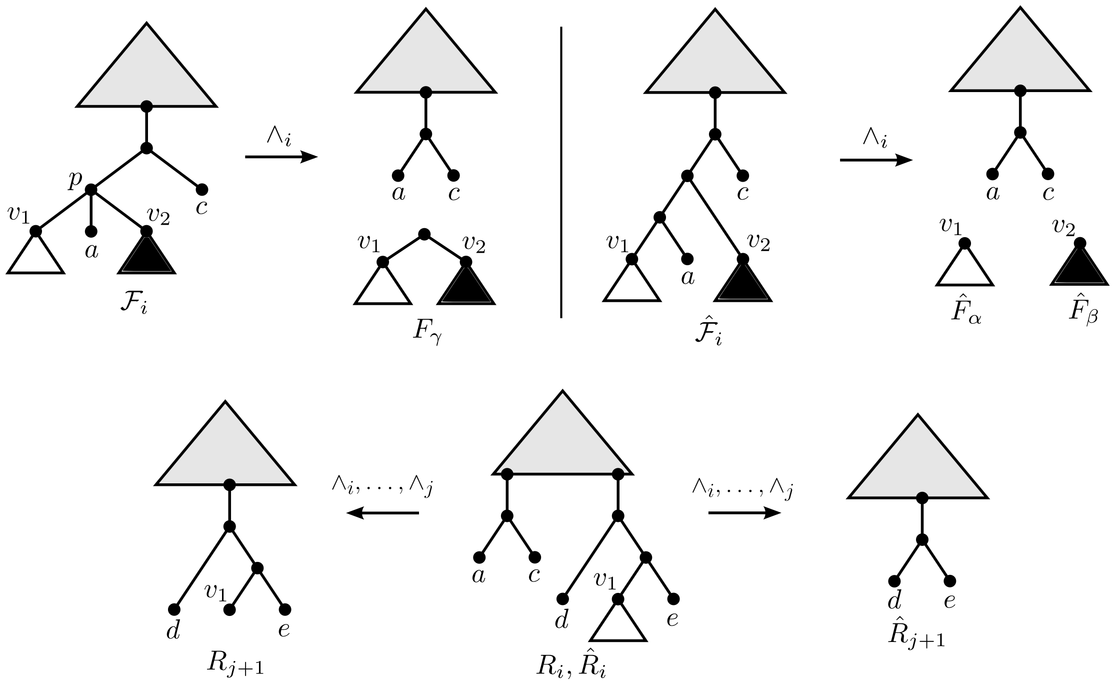


Figure 4.9: An illustration of the scenario described in the proof of Lemma 22.

In the following, we will show that a cherry list $\bigwedge$ for two binary resolutions of two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2 is also mimicking a computational path of ALL-MULMAFS* applied to T_1 and T_2.

Lemma 22. *Let $\hat{T}_1$ and $\hat{T}_2$ be two binary resolutions of two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, respectively. Moreover, let $\bigwedge$ be a cherry list for $\hat{T}_1$ and $\hat{T}_2$. Then, $\bigwedge$ is automatically a cherry list for T_1 and T_2.*

Proof. Lemma 22 obviously holds, if the cherry list $\bigwedge$ only exists of cherry actions $\wedge_i = (\{a,c\},\phi_i)$ with $\phi_i \in \{\cup_{ac}, \dagger_a, \dagger_c,\}$. This is, in particular, the case because these cherry actions only affect those nodes whose corresponding subtree has been fully contracted so far. When processing a cherry action $\wedge_i = (\{a,c\},\phi_i)$ with $\phi_i = \cap_{ac}$, however, two slightly different forests $\hat{\mathcal{F}}_{i+1}$ and $\mathcal{F}_{i+1}$ can arise. More precisely, this is the case, if there is a multifurcating node x lying on the path P_{ac} connecting taxon a and c in $\mathcal{F}_i$ providing a set $V_x = \{v_0, v_1, v_2, \ldots, v_n\}$ of at least 3 children, where v_0 denotes the node which is also part of P and, if, additionally, R_i contains two nodes d and e whose path connecting d and e contains a set of pendant subtrees each corresponding to $R_i(v_i)$, with $i > 0$ and $v_i \in V_x$.

Now, for simplicity, we assume that there is only one such multifurcating node x of out-degree three so that $V_x = \{v_0, v_1, v_2\}$. In this case, as $\hat{\mathcal{F}}_i$ only consists of binary trees, by processing $\wedge_i = (\{a,c\},\cap_{ac})$ two components $\hat{F}_\alpha$ and $\hat{F}_\beta$ rooted at v_1 and v_2, respectively, are added to $\hat{\mathcal{F}}_{i+1}$ whereas to $\mathcal{F}_{i+1}$ only one component F_γ is added whose root contains two children corresponding to v_1 and v_2 (cf. Fig. 4.9).

Now, let $\wedge_j$ be a cherry action in $\bigwedge$ with $j > i$ in which both components $\hat{F}_\alpha$ and $\hat{F}_\beta$ have been fully contracted so far. As a consequence, since the components $\hat{F}_\alpha$ and $\hat{F}_\beta$ only consist of a single taxon, which has been removed from $\hat{R}_{j+1}$ (cf. Alg. 14, line 21), the two taxa d and e are now cherries in $\hat{R}_{j+1}$ which is, however, not the case in R_{j+1} because F_γ still contains the two nodes v_1 and v_2 (cf. Fig. 4.9). Nevertheless, since in F_γ the node v_1 and v_2 are leaves directly attached to the root, the cherry $\{d, e\}$ is a pseudo cherry in R_{j+1} and, thus, an upcoming cherry action $\wedge_k$ containing $\{d, e\}$ represents a pseudo cherry for R_k and $\mathcal{F}_k$ in this case.

As a consequence, each cherry of $\hat{R}_i$ and $\hat{\mathcal{F}}_i$ corresponding to a cherry action $\wedge_i$ in $\bigwedge$ is either a cherry or a pseudo cherry of R_i and $\mathcal{F}_i$ and, thus, Lemma 22 is established. □

Next, we will show that by expanding a forest $\mathcal{F}'$ on $\mathcal{X}$ as prescribed in M derived from calling allMulMAFs, the resulting forest is automatically an agreement forest for both input trees T_1 and T_2.

Lemma 23. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, let $\mathcal{F}$ be a forest on $\mathcal{X}$ that has been expanded as prescribed in M after* allMulMAFs*$(T_1, \{T_2\}, \emptyset, k)$ *has been called. Then, $\mathcal{F}$ is an agreement forest for T_1 and T_2.*

Proof. Since, obviously, $\mathcal{F}$ is a partition of $\mathcal{X}$, it suffices to consider each of the following cases describing a putative scenario leading to a forest that is not an agreement forest for both input trees T_1 and T_2, because either the refinement property or the node disjoint property in terms of T_1 or T_2 is not fulfilled. We will show, however, that during the execution of our algorithm allMulMAFs* each of those scenarios can be excluded.

Case 1. Assume there exists a component F_i in $\mathcal{F}$ such that F_i is not a refinement of $T_2|_{\mathcal{X}_i}$, where $\mathcal{X}_i$ denotes the taxa set of F_i. As $\mathcal{F}'$ has been derived from T_2 by cutting and contracting its edges, this automatically implies that a cherry has been expanded as prescribed in M in respect of $\top$ instead of $\bot$. However, in M a cherry is only then mapped to $\top$, if and only if, during the i-th recursive call, it is a common cherry of R_i and $\mathcal{F}_i$ and if both parents corresponding to the cherry in R_i and F_i are multifurcating nodes (cf. Case 2a of allMulMAFs*). Moreover, such a cherry is immediately mapped back to $\bot$, if either the cherry itself or all its siblings have been cut (cf. Case 1c,2c of allMulMAFs*). Thus, such a component F_i cannot exist in $\mathcal{F}$.

Case 2. Assume there exists a component F_i in $\mathcal{F}$ such that F_i is not a refinement of $T_1|_{\mathcal{X}_i}$, where $\mathcal{X}_i$ denotes the taxa set of F_i. This automatically implies that either a cherry has been expanded as prescribed in M in respect of $\top$ instead of $\bot$ or, during the i-th recursive call, a cherry was not a common cherry of R_i and F_i. Similar to Case 1, the first scenario can be excluded. Moreover, the latter scenario cannot take place either, since, in order to reduce F_i to a single node, this common cherry must have been contracted which can only take place, if it was a common cherry of R_i and $\mathcal{F}_i$. Thus, such a component F_i cannot exist in $\mathcal{F}$.

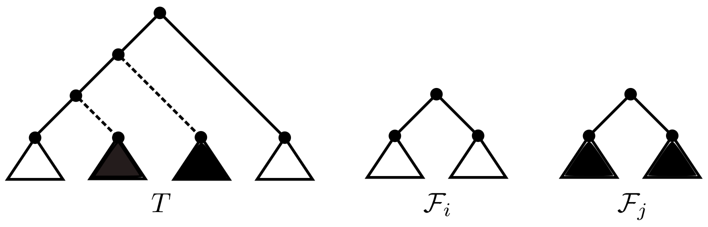


Figure 4.10: An illustration of the scenario described in Case 4 corresponding to the proof of Lemma 23. In order to obtain F_i from T, both dashed edges have to be cut whereas, in order to obtain F_j, these two edges have to be contracted which is a contradiction.

Case 3. Assume there exist two components F_i and F_j in $\mathcal{F}$, with taxa set $\mathcal{X}_i$ and $\mathcal{X}_j$, respectively, such that $T_2(\mathcal{X}_i)$ and $T_2(\mathcal{X}_j)$ are not edge disjoint in T_2. As F_i and F_j must be a refinement of $T_2|_{\mathcal{X}_i}$ and $T_2|_{\mathcal{X}_j}$, respectively, (cf. Case 1) and both components have been derived from T_2 by cutting some of its edges, only one of both components can be part of $\mathcal{F}$. As a direct consequence, such two components cannot exist in $\mathcal{F}$.

Case 4. Assume there exist two components F_i and F_j in $\mathcal{F}$ such that $T_1(\mathcal{X}_i)$ and $T_1(\mathcal{X}_j)$ are not edge disjoint in T_1. As shown in Case 2, F_i and F_j must be a refinement of $T_1|_{\mathcal{X}_i}$ and $T_1|_{\mathcal{X}_j}$, respectively, and, thus, in order to obtain F_i and F_j the following steps must be performed during the execution of allMulMAFs*. Let E_i be an edge set that is only contained in $T_1(\mathcal{X}_i)$ and not in $T_1(\mathcal{X}_j)$. In order to obtain F_i, some of those edges in E_j must be cut, whereas, in order to obtain F_j, all of them must be contracted which leads to a contradiction (cf. Fig. 4.10). Thus, such two components cannot exist in $\mathcal{F}$.

Finally, by combining all four cases Lemma 23 is established. □

Moreover, in the following, we will show that each agreement forest $\mathcal{F}$ that is reported by allMulMAFs* is relevant which means that $\mathcal{F}$ does not contain an edge e such that $\mathcal{F} \ominus \{e\}$ is still an agreement forest for both input trees.

Lemma 24. *Let T_1 and T_2 be two rooted (nonbinary) phylogenetic trees, then, each agreement forest that is reported by applying* allMulMAFs* *to T_1 and T_2 is relevant.*

Proof. Just by definition, given two rooted (nonbinary) phylogenetic trees T_1 and T_2, an agreement forest $\mathcal{F}$ for T_1 and T_2 that is not relevant has to contain an edge e such that $\mathcal{F} \ominus \{e\}$ is still an agreement forest for T_1 and T_2. Such an edge e, however, can only arise, if a cherry of a multifurcating node is expanded in respect of $\perp$ instead of $\top$. Initially, such a cherry must have been set to $\top$ because during the i-th recursive call both corresponding parents in R_i and $\mathcal{F}_i$, respectively, must have been multifurcating nodes. The only scenario setting $\top$ to $\perp$ would arise, if the cherry itself or all its siblings are cut during subsequent recursive calls. In this case, however, this cherry has to be set to $\perp$, since, otherwise the resulting forest would not be an agreement forest for T_1 and T_2. Thus, such an edge e cannot exist and, consequently, Lemma 24 is established. □

Since both algorithms allMulMAFs* and allMulMAFs process common cherries and contradicting cherries in the same way, Lemma 24, obviously, has to hold for allMulMAFs as well.

Corollary 25. *Let T_1 and T_2 be two rooted (nonbinary) phylogenetic trees, then, each agreement forest that is reported by applying* allMulMAFs *to T_1 and T_2 is relevant.*

Now, let T_1 and T_2 be two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. By the following Lemma 26, we will show that for each maximum binary agreement forest $\hat{\mathcal{F}}$, which can be computed by applying ProcessCherries to a cherry list $\bigwedge$ for two binary resolutions of T_1 and T_2, by calling ProcessCherries$(T_1, T_2, \bigwedge)$ a forest $\mathcal{F}$ is computed such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$.

Lemma 26. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, let $\hat{T}_1$ and $\hat{T}_2$ be two binary resolutions of T_1 and T_2, respectively. Moreover, let $\hat{\mathcal{F}}$ be an agreement forest for $\hat{T}_1$ and $\hat{T}_2$ obtained from calling* ProcessCherries$(\hat{T}_1, \{\hat{T}_2\}, \bigwedge)$*, where $\bigwedge$ denotes a cherry list for $\hat{T}_1$ and $\hat{T}_2$. Then, a relevant agreement forest $\mathcal{F}$ is calculated by calling* ProcessCherries$(T_1, \{T_2\}, \bigwedge)$ *such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$.*

Proof. We will first show by induction a slightly modified version of Lemma 26.

Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2. Let $\hat{T}_1$ and $\hat{T}_2$ be two binary resolutions of T_1 and T_2, respectively, and let $\hat{\mathcal{F}}_i$ and $\mathcal{F}_i$ be each forest corresponding to iteration i while executing

$$\textsc{ProcessCherries}(\hat{T}_1, \{\hat{T}_2\}, \bigwedge) \text{ and } \textsc{ProcessCherries}(T_1, \{T_2\}, \bigwedge),$$

respectively, where $\bigwedge = (\wedge_1, \wedge_2, \dots, \wedge_n)$ is a cherry list for $\hat{T}_1$ and $\hat{T}_2$. Then, $\hat{\mathcal{F}}_i$ is a called *pseudo binary resolution* of $\mathcal{F}_i$, which is defined as follows. Given two forests $\hat{\mathcal{F}}$ and $\mathcal{F}$ for a phylogenetic $\mathcal{X}$-tree, we say that $\hat{\mathcal{F}}$ is a pseudo binary resolution of $\mathcal{F}$, if for each component $\hat{F}$ in $\hat{\mathcal{F}}$ there exists a component F in $\mathcal{F}$ such that one of the two following properties hold.

(i) $\hat{F}$ is a binary resolution of F.

(ii) $\hat{F}$ is a binary resolution of $F(v)$, where v is a child of the root of F.

The following proof is established by an induction on i denoting the position of a cherry action in $\bigwedge = (\wedge_1, \wedge_2, \dots, \wedge_n)$.

Base case. At the beginning, F_1 only consists of $\hat{T}_2$, which is a binary resolution of T_2. Thus, the assumption obviously holds for $i = 1$.

Inductive step. Depending on the cherry action $\wedge_i = (\{a, c\}, E)$, the forest $\hat{\mathcal{F}}_{i+1}$ can be obtained from $\hat{\mathcal{F}}_i$ in the following ways.

(i) If $\{a,c\}$ is a pseudo cherry, a set of nodes V' that is attach to the root of a component in $\mathcal{F}_i$ is cut. Since $\bigwedge$ is a cherry list for $\hat{T}_1$ and $\hat{T}_2$ and, thus, $\{a,c\}$ is a cherry in $\hat{R}_i$, each of node in V' already refers to components in $\hat{\mathcal{F}}_i$ all consisting only of isolated nodes. Thus, after cutting the in-edge of each node in V', $\hat{\mathcal{F}}_i$ is still a pseudo binary resolution of $\mathcal{F}_i$.

(ii) If $\{a,c\}$ is a common cherry and, thus $\phi_i = \cup_{ac}$, in both forests $\hat{\mathcal{F}}_i$ and $\mathcal{F}_i$ the two taxa a and c are contracted. Consequently, since $\hat{\mathcal{F}}_i$ is a pseudo binary resolution of $\mathcal{F}_i$, this directly implies that $\hat{\mathcal{F}}_{i+1}$ is a pseudo binary resolution of $\mathcal{F}_{i+1}$ as well.

(iii) If $\{a,c\}$ is a contradicting cherry and $\phi_i = \nmid_a$ (or $\phi_i = \nmid_c$), then, in both forests $\hat{\mathcal{F}}_i$ and $\mathcal{F}_i$ the node labeled by taxon a (or taxon c) is cut. Again, no matter if $a \sim c$ or $a \not\sim c$ holds, since $\hat{\mathcal{F}}_i$ is a pseudo binary resolution of $\mathcal{F}_i$, this directly implies that $\hat{\mathcal{F}}_{i+1}$ is a pseudo binary resolution of $\mathcal{F}_{i+1}$ as well.

(iv) If $\{a,c\}$ is a contradicting cherry and $\phi_i = \cap_{ac}$, in both forests $\hat{\mathcal{F}}_i$ and $\mathcal{F}_i[a \sim c]$ each pendant subtree lying on the path connecting both leaves labeled by a and c, respectively, is cut. Let $\hat{\mathcal{F}}'$ and $\mathcal{F}'$ be those component arising from cutting $\hat{\mathcal{F}}_i$ and $\mathcal{F}_i[a \sim c]$, respectively. Since $\hat{\mathcal{F}}_i$ is a binary resolution of $\mathcal{F}_i$, $|\mathcal{F}'| \geq |\hat{\mathcal{F}}'|$ holds which means, in particular, that each $\hat{F}'_i$ in $\hat{\mathcal{F}}'$ is either a binary resolution of F'_j or a binary resolution of $F'_j(v)$ in $\mathcal{F}'$, where v corresponds to a child whose parent is the root of F'_j. Thus, since $\hat{\mathcal{F}}_i$ is a pseudo binary resolution of $\mathcal{F}_i$, this directly implies that $\hat{\mathcal{F}}_{i+1}$ is a pseudo binary resolution of $\mathcal{F}_{i+1}$ as well.

Now, from the induction we can deduce that, independent from the cherry action $\wedge_i$, $\hat{\mathcal{F}}_i$ is always a pseudo binary resolution of $\mathcal{F}_i$. Moreover, let $\hat{\mathcal{F}}_{n+1}$ and $\mathcal{F}_{n+1}$ be the two forests obtained from $\hat{\mathcal{F}}_n$ and $\mathcal{F}_n$, respectively, by applying $\wedge_n$. Then, since $\hat{\mathcal{F}}$ is an agreement forest for $\hat{T}_1$ and $\hat{T}_2$, all components in $\hat{\mathcal{F}}_{n+1}$ only consist of single isolated nodes which directly implies that $\mathcal{F}_{n+1}$ does not contain any cherries. Furthermore, due to Lemma 23, by expanding $\mathcal{F}_{n+1}$ as prescribed in M a relevant agreement forest $\mathcal{F}$ arises such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$ which completes the proof of Lemma 26. □

In the following, we will show that Lemma 26 also holds for the original algorithm allMulMAFs.

Lemma 27. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, let $\hat{\mathcal{F}}$ be a binary maximum agreement forest for T_1 and T_2. Then, by calling*

$$\text{allMulMAFs}(T_1, \{T_2\}, \emptyset, k)$$

a relevant maximum agreement forests $\mathcal{F}$ for T_1 and T_2 is computed such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$, if and only if $k \geq h(T_1, T_2)$.

Proof. Notice that, as proven in Lemma 26, Theorem 27 holds for the modified algorithm allMulMAFs*. Thus, in order to establish Lemma 27, we just have to show that the following two differences between both algorithms allMulMAFs* and allMulMAFs do not have an impact on the computation of maximum agreement forests.

Needless cherries. First of all, let $\bigwedge$ be a cherry list for T_1 and T_2 mimicking a computational path of ALLMULMAFS* resulting in an agreement forest $\mathcal{F}$ for T_1 and T_2. Moreover, let $\bigwedge$ contain a cherry action $\wedge_i = (\{a,b\},\phi_i)$ in which $\{a,b\}$ is a contradicting cherry of R_i and $\mathcal{F}_i$. Now, if R_i contains a taxon c such that $\{a,c\}$ is a common cherry, we call $\{a,b\}$ a *needless cherry.* Notice that a computational path corresponding to the original algorithm ALLMULMAFS does not consider needless cherries as it always prefers common cherries to contradicting cherries. In the following, however, we will show that for the computation of maximum agreement forests each computational path processing needless cherries can be neglected.

Let $\mathcal{F}$ be an agreement forest resulting from a computational path of ALLMULMAFS* in which, instead of processing a common cherry $\{a,c\}$, a needless cherry $\{a,b\}$ is processed by the cherry action $\wedge_i = (\{a,b\},\nmid_a)$. This implies that $\mathcal{F}$ contains a component F_a corresponding to the expanded taxon a, which has been cut during the i-th iteration. Moreover, let F_c be the component in $\mathcal{F}$ containing the node v_c corresponding to taxon c in $\mathcal{F}_i$. Since $\{a,c\}$ has been a common cherry in iteration i, by attaching F_a back to the in-edge of v_c an agreement forest of size $k-1$ arises and, thus, $\mathcal{F}$ cannot be a maximum agreement forest. Consequently, from cutting instead of contracting common cherries a maximum agreement forest cannot arise and, thus, for the computation of maximum agreement forests each computational path of ALLMULMAFS* processing needless cherries can be neglected.

Notice that, in this case, the cherry action $\wedge_i = (\{a,b\},\nmid_b)$ would be also not be considered. However, after having contracted the common cherry $\{a,c\}$, b could still be cut selecting a cherry action involving one of its siblings.

Pseudo cherries. Furthermore, in contrast to the modified algorithm ALLMULMAFS*, a computational path corresponding to the original algorithm ALLMULMAFS does not consider pseudo cherries. In the following, however, we will show that for an agreement forest $\mathcal{F}$ resulting from a computational path processing pseudo cherries, there exists a different computational path calculating $\mathcal{F}$ without considering any pseudo cherries.

Let $\bigwedge$ be a cherry list for T_1 and T_2 mimicking a computational path of ALLMULMAFS* resulting in an agreement forest $\mathcal{F}$ and let $\wedge_i$ be a cherry action whose corresponding cherry $\{a,c\}$ is a pseudo cherry of R_i and $\mathcal{F}_i$. Moreover, let be $(b_1,b_2,\dots,b_k)$ and $(b_{k+1},b_2,\dots,b_n)$ be those taxa corresponding to each pendant node lying on the path connecting a and $\mathrm{LCA}_{R_i}(\{a,c\})$ as well as c and $\mathrm{LCA}_{R_i}(\{a,c\})$, respectively. Then, we can replace $\wedge_i = (\{a,c\},\phi_i)$ through the sequence of cherry actions

$$(\{a,b_1\},\nmid_{b_1}),\dots,(\{a,b_k\},\nmid_{b_k}),(\{c,b_{k+1}\},\nmid_{b_{k+1}}),\dots,(\{c,b_n\},\nmid_{b_n}),(\{a,c\},\phi_i)$$

neither containing pseudo cherries nor needles cherries such that the agreement forest $\mathcal{F}$ is still computed. This means, in particular, that each tree operation that is conducted for preparing a pseudo cherry can be also realized by a sequence of cherry actions neither containing needless cherries nor pseudo cherries.

As shown above, for a relevant maximum agreement forest $\mathcal{F}$ our modified algorithm allMulMAFs* always contains a computational path calculating $\mathcal{F}$ by neither taking needless cherries nor pseudo cherries into account. Thus, each relevant maximum agreement forest for T_1 and T_2 that is calculated by allMulMAFs* is also calculated by allMulMAFs and, as a direct consequence, Lemma 27 is established. $\square$

Now, in a last step, we can finish the proof of Theorem 20. Let $\hat{T}_1$ and $\hat{T}_2$ be two binary resolutions of two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2, respectively, and let $\hat{\mathcal{F}}$ be an agreement forest for $\hat{T}_1$ and $\hat{T}_2$. Then, by combining Corollary 25 and Lemma 27 we can deduce that the algorithm allMulMAFs computes a relevant maximum agreement forest $\mathcal{F}$ for T_1 and T_2 such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$. This automatically implies, that our algorithm calculates all relevant maximum agreement forests for T_1 and T_2 and, thus, Theorem 20 is finally established. $\square$

4.2.4 Runtime of allMulMAFs

In this section, we discuss the theoretical worst-case runtime of the algorithm allMulMAFs in detail.

Theorem 28. *Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and $\mathcal{F}$ be a relevant maximum agreement forest for T_1 and T_2 containing k components. The theoretical worst-case runtime of the algorithm* allMulMAFs *applied to T_1 and T_2 is $O(3^{|\mathcal{X}|+k}|\mathcal{X}|)$.*

Proof. Let $\mathcal{F} = \{F_\rho, F_1, F_2, \dots, F_{k-1}\}$ be an agreement forest for T_1 and T_2 of size k. To obtain $\mathcal{F}$ from T_2, obviously $k-1$ edge cuttings are necessary. Moreover, in order to reduce the size of the leaf set $\mathcal{X}$ of R to 1, to each component F_i in $\mathcal{F}$ we have to apply exactly $|\mathcal{L}(F_i)| - 1$ cherry contractions. Consequently, at most $|\mathcal{X}|$ cherry contractions have to be performed in total. Thus, our algorithm has to perform at most $O(|\mathcal{X}| + k)$ recursive calls for the computation of $\mathcal{F}$. Now, as one of these recursive calls can at least branch into 3 further recursive calls, $O(3^{|\mathcal{X}|+k})$ is an upper bound for the total number of recursive calls that are performed throughout the whole algorithm. Moreover, each case that is conducted during a recursive (cf. Sec. 4.2.2) can be done in $O(|\mathcal{X}|)$ time and, thus, the theoretical worst-case runtime of the algorithm can be estimated with $O(3^{|\mathcal{X}|+k}|\mathcal{X}|)$. $\square$

4.2.5 Conclusion

In this section, we have presented the algorithm allMulMAFs calculating all relevant maximum agreement forests for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Therefor, we have established a detailed formal proof showing the correctness of the algorithm which is based on both previously presented algorithms allMAAFs and allMAAFs[1]. In the next section, we will show how to further modify the algorithm allMulMAFs so that now all relevant maximum *acyclic* agreement forests are calculated.

4.3 The algorithm allMulMAAFs

In this section, we show how to extend the algorithm allMulMAFs, presented in Section 4.2.2, such that the reported agreement forests additional satisfy the acyclic constraint, which automatically implies that the extended algorithm will calculated all relevant maximum *acyclic* agreement forests for two rooted *nonbinary* phylogenetic $\mathcal{X}$-trees. As mentioned previously, the acyclic constraint plays an important role for the construction of hybridization networks as, for example, demonstrated by the algorithm HybridPhylogeny [9]. More specifically, this algorithms generates a hybridization network displaying two rooted *bifurcating* phylogenetic $\mathcal{X}$-trees from the components of an acyclic agreement forest of those two trees. Thus, we consider the computation of nonbinary maximum acyclic agreement forests as a first step to come up with minimum hybridization networks displaying the refinements of two rooted nonbinary phylogenetic $\mathcal{X}$-trees.

Broadly speaking, the algorithm allMulMAFs can be used to make progress towards an agreement forest for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2 as long as the set of components $\mathcal{F}$ does not satisfy all properties of an agreement forest. Once our algorithm has successfully computed a maximum agreement forest $\mathcal{F}$ for T_1 and T_2, we can apply a specific tool that is able to check, if we can refine $\mathcal{F}$ to a maximum acyclic agreement forest. Such a refinement of an agreement forest is done by cutting a minimum number of edges within its components such that each directed cycle of the underlying ancestor-descendant graph $AG(T_1, T_2, \mathcal{F})$ is dissolved.

Notice that this problem is closely related to the *directed feedback vertex set problem.* More specifically, given a directed graph G with node set V, a feedback vertex set V' is a subset of V containing at least one node of each directed cycle of G. This implies, by deleting each node of V' together with its adjacent edges, each directed cycle is automatically removed. Now, based on a directed graph, the directed feedback vertex set problem consists of minimizing the size of such a feedback vertex set.

4.3.1 Refining agreement forests

In this section we present a tool that enables the refinement of an agreement forest. We call this tool an *expanded ancestor-descendant graph.* Notice that this tool has been previously published under a different term as we state in the following.

Remark 4. The following concept of an *expanded ancestor-descendant graph* corresponds to the concept of an *expanded cycle graph* given in the work of Whidden *et al.* [69]. The latter concept, however, can be only applied to agreement forests corresponding to rooted *binary* phylogenetic $\mathcal{X}$-trees. Hence, we have adapted this concept such that it can be also applied to agreement forests corresponding to rooted *nonbinary* phylogenetic $\mathcal{X}$-trees. Notice that, adapting the concept of an expanded cycle graph to nonbinary agreement forests has been also examined in the master thesis of Li [39]. In this work, each step that is necessary to compute the hybridization number for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees is presented in detail by, additionally, discussing its correctness.

In the following, we give a short overview of how an expanded ancestor-descendant graph is defined and how this graph can be used to transform agreement forests into acyclic agreement forests.

Expanded ancestor-descendant graph. The tool that enables the refinement of an agreement forest $\mathcal{F}$ for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2 to an acyclic agreement forest is an *expanded ancestor-descendant graph* $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$. In contrast to the ancestor-descendant graph, each node of this graph corresponds to exactly one particular node of a component in $\mathcal{F}$. Thus, from such a graph one can directly figure out those edges of a component that have to be cut in order to remove a directed cycle (cf. Fig. 4.11).

Given two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2 and a nonbinary maximum acyclic agreement forest $\mathcal{F}$ for T_1 and T_2, the corresponding expanded ancestor-descendant graph $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ consists of the following nodes and edges. First of all, $\mathcal{F}$ is a subset of $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$, which means that the graph contains all nodes and edges corresponding to all components in $\mathcal{F}$. Moreover, $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ contains a set of *hybrid edges* each connecting two specific nodes each being part of two different components. More precisely, those edges are defined as follows.

Given a node v of a component F_j in $\mathcal{F}$, the function $\phi_i(v)$ refers to the lowest common ancestor in $T_i(\mathcal{F})$, with $i \in \{1, 2\}$, of each leaf that is labeled by a taxon contained in $\mathcal{L}(F_j(v))$. Notice that the node $\phi_i(\cdot)$ is well defined, which means there exists exactly one node in $T_i(\mathcal{F})$ to which $\phi_i(\cdot)$ applies. Equivalently, the function $\phi_i^{-1}(\cdot)$ maps nodes from T_i, with $i \in \{1, 2\}$, back to a component in $\mathcal{F}$. More precisely, let E_i, with $i \in \{1, 2\}$, be the set of edges consisting of all in-edges of all lowest common ancestors in T_i of the taxa set of each F_j in $\mathcal{F} \setminus \{F_\rho\}$. Then, the node $v \in T_i$ maps back to the node in $\mathcal{F}$ representing the lowest common ancestor of those taxa that can be reached from v by not using an edge in E_i. Notice, however, that this function is only defined for those nodes that are either labeled or are part of a path connecting two labeled nodes a and b such that $\phi_i^{-1}(a)$ and $\phi_i^{-1}(b)$ are contained in the same component F_j in $\mathcal{F}$. Similar to the binary case, since the graph is built for the trees $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ reflecting $\mathcal{F}$, the function $\phi_i^{-1}(\cdot)$ is well defined, which means that, if defined, there exists exactly one node in $\mathcal{F}$ to which $\phi_i^{-1}(\cdot)$ applies.

Now, based on the definitions of these two functions, $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ contains the following hybrid edges. Let w be a node in this graph corresponding to the root of a component F_j not equal to F_ρ. Moreover, for the tree $T_i(\mathcal{F})$ with $i \in \{1, 2\}$, let v' be the lowest ancestor of $\phi_i(w)$ such that $\phi_i^{-1}(v')$ is defined. In more detail, let $P_\phi = (v_1, \dots, v_n)$ be those nodes lying on the path connecting the parent v_1 of v and the root v_n of T_1 such that v_j with $j \in [2 : n]$ is the parent of v_{j-1}. Then,

$$v' = \min_j \{v_j : v_j \in P_\phi \wedge \phi_i^{-1}(v_j) \text{ is defined}\}.$$

Based on v' and w, $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ contains a hybrid edge $(\phi_i^{-1}(v'), w)$. Notice that, if $\mathcal{F}$ contains k components, for each component except F_ρ two hybrid edges corresponding to T_1 and T_2 are inserted which are $2k-2$ hybrid edges in total. Furthermore, the target node of a hybrid edge does always refer to a root node of a component F_j in $\mathcal{F}$ whereas the source node never does.

Exit nodes. Given two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2 as well as a nonbinary maximum acyclic forest $\mathcal{F}$ for T_1 and T_2, an *exit node of* $AG^{ex}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ is defined as follows. Let H_i be the set of hybrid edges in $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ resulting from T_i with $i \in \{1,2\}$. Now, given a directed cycle in $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ running through the hybrid edges $E_H = \{h_0, \dots, h_{n-1}\}$ in sequential order, then, the source node v_i of a hybrid edge $h_i = (v_i, w_i)$ in E_H is called an exit node, if h_i is contained in H_1 and h_j, with $j = (i-1) \mod n$, is contained in H_2 or vice versa.

Now, based on an expanded ancestor-descendant graph we can refine an agreement forest by *fixing its exit nodes.* An exit node v belonging to the component F_j is fixed by cutting each edge lying on the path connecting v with the node referring to the root node of F_j. Notice that by cutting k of those edges, the resulting agreement forest $\mathcal{F}'$ consists of $|\mathcal{F}| + k$ components.

4.3.2 The algorithm

We can easily turn the algorithm allMulMAFs into the algorithm allMulMAAFs by applying a post-processing step refining agreement forests. More precisely, given an agreement forest $\mathcal{F}$ for two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2, by applying the following refinement procedure only those relevant acyclic agreement forests are returned whose size is smaller than or equal to k.

(1) Compute two trees $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ reflecting $\mathcal{F}$.

(2) Build the expanded ancestor-descendant graph $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$.

(3) Compute the set of exit nodes V_H of $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$.

(4) For each exit node v in V_H turn $\mathcal{F}$ into $\mathcal{F}'$ by fixing v.

(5) For each agreement forest $\mathcal{F}'$ with $|\mathcal{F}'| \leq k$ continue with step 5a or 5b.

  (5a) If $\mathcal{F}'_i$ is acyclic, return $\mathcal{F}'$.

  (5b) Otherwise, if $\mathcal{F}'$ is *not* acyclic, repeat step 2–5 with $\mathcal{F}'$.

Based on these steps, by modifying Case 1b as follows, we can easily turn the algorithm allMulMAFs into an algorithm computing a set of maximum acyclic agreement forests.

Case 1b'. If R only consists of a single leaf, first each F_i in $\mathcal{F}$ is expanded as prescribed in M and then $\mathcal{F}$ is refined with the help of $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ into $\mathcal{F}'$. Finally, $\mathcal{F}'$ is

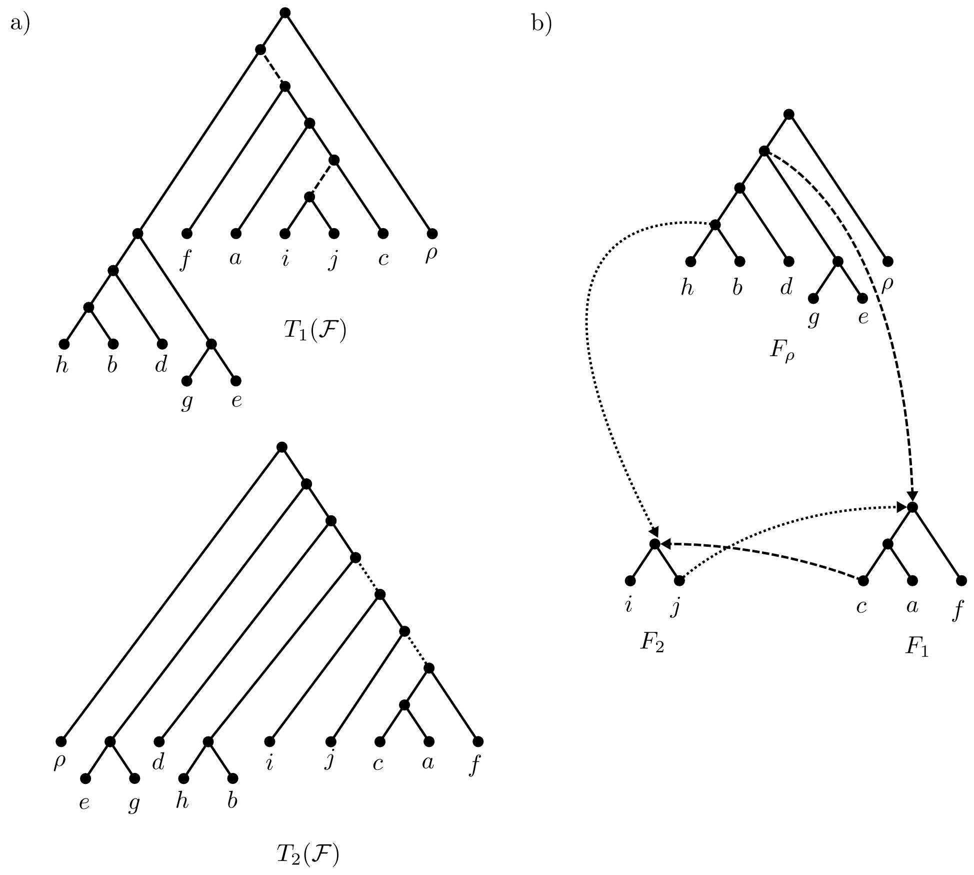


Figure 4.11: **(a)** The same two trees $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ as depicted in Figure 4.1. Here, the set of dashed edges and the set of dotted edges refers to the in-edges of the nodes corresponding to the lowest common ancestors of $\mathcal{L}(F_1)$ and $\mathcal{L}(F_2)$. **(b)**The expanded ancestor-descendant graph $AG^{\text{ex}}(T_1(\mathcal{F}), T_2(\mathcal{F}), \mathcal{F})$ with $\mathcal{F} = \{F_\rho, F_1, F_2\}$. Dashed edges are hybrid edges resulting from $T_1(\mathcal{F})$ and dotted edges are hybrid edges resulting from $T_2(\mathcal{F})$. Here, for a better overview, the directions of edges corresponding to components of $\mathcal{F}$ are omitted. Notice that, by fixing the exit node corresponding to taxon j, all directed cycles are removed and a maximum acyclic agreement forest for T_1 and T_2 with size 4 arises.

returned.

This means that, each time before reporting an agreement forest $\mathcal{F}$, we first check, if we can refine $\mathcal{F}$ to an acyclic agreement forest $\mathcal{F}'$ of size smaller than or equal to k. If this is possible, we return $\mathcal{F}'$, else, we return the empty set.

4.3.3 Correctness of allMulMAAFs

In this section, we show that by applying the presented algorithm ALLMULMAAFS one can calculate all relevant maximum acyclic agreement forests for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees.

Theorem 29. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees, by calling*

$$\textsc{allMulMAAFs}(T_1, \{T_2\}, \emptyset, k)$$

all relevant maximum acyclic agreement forests for T_1 and T_2 are calculated, if and only if $k \geq h(T_1, T_2)$.

Proof. The correctness of the algorithm as stated in Theorem 29 directly depends on the following two Lemmas 30 and 31.

Lemma 30. *Let T_1 and T_2 be two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees and let $\mathcal{F}$ be a relevant maximum acyclic agreement forest T_1 and T_2. Then, a relevant agreement forest $\mathcal{F}'$ by calling* $\textsc{allMulMAFs}(T_1, T_2, \emptyset, h(T_1, T_2))$ *is calculated that can be turned into $\mathcal{F}$ by first resolving some of its multifurcating nodes and then by cutting some of its edges.*

Proof. As the first point holds for the algorithm $\textsc{allMAAFs}^3$ (cf. Theorem 10), from Lemma 26 we can deduce that this has to hold for the algorithm $\textsc{allMulMAFs}$ as well. More precisely, let T_1 and T_2 be two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees and let $\hat{\mathcal{F}}$ be a binary agreement forest for $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ that can be turned into a maximum acyclic agreement forest for $T_1(\mathcal{F})$ and $T_2(\mathcal{F})$ by cutting some of its edges. Then, due to Lemma 26, by calling $\textsc{allMAAFs}^3(T_1, T_2, \emptyset, h(T_1, T_2))$ a relevant acyclic agreement forest $\mathcal{F}$ for T_1 and T_2 is calculated such that $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$. Moreover, as $\hat{\mathcal{F}}$ can be turned into a maximum acyclic agreement forest by cutting some of its edges $\hat{E}$, $\mathcal{F}$ can be turned into a relevant maximum acyclic agreement forest as well by first resolving some of its nodes and then by cutting a certain edge set E with $|\hat{E}| = |E|$. More specifically, for each edge $\hat{e}$ in $\hat{E}$ there exists an edge e that can be obtained from $\mathcal{F}$ as follows. Let $\hat{e} = (\hat{v}, \hat{w})$ be an edge in $\hat{E}$ of a component $\hat{F}$ in $\hat{\mathcal{F}}$, then, as $\hat{\mathcal{F}}$ is a binary resolution of $\mathcal{F}$, $\mathcal{F}$ has to contain a component F with node w' such that $\mathcal{L}(\hat{F}(\hat{w})) \subseteq \mathcal{L}(F(w'))$. Now, e is the in-edge of a node w that can be obtained from resolving node w' such that $\mathcal{L}(F(\hat{w})) = \mathcal{L}(F(w))$. □

Lemma 31. *Given two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees T_1 and T_2 as well as a relevant agreement forest $\mathcal{F}$ for T_1 and T_2, the refinement step resolves a minimum number of nodes and cuts a minimum number of edges such that $\mathcal{F}$ is turned into all relevant acyclic agreement forests of minimum size.*

Proof. Due to the following two observations that are both discussed in the master thesis of Li [39], the refinement procedure, which is based on fixing exit nodes as described above, leads to the computation of acyclic agreement forests.

Observation 2. *Let $\mathcal{F}$ be an agreement forest for two rooted phylogenetic $\mathcal{X}$-trees T_1 and T_2 and let $\mathcal{F}'$ be an agreement forest that is produced by fixing an exit node of $AG^{ex}(T_1, T_2, \mathcal{F})$. Then, the set of exit nodes corresponding to $AG^{ex}(T_1, T_2, \mathcal{F}')$ is a subset of the set of exit nodes corresponding to $AG^{ex}(T_1, T_2, \mathcal{F})$.*

Observation 3. *Given an agreement forest $\mathcal{F}$ for two rooted phylogenetic $\mathcal{X}$-trees, there exists an acyclic agreement forest $\mathcal{F}'$, if and only if there exists a set of exit nodes such that fixing theses nodes leads to the computation of $\mathcal{F}'$.*

A formal proof showing the correctness of these two observations can be looked up in the master thesis of Li [39]. More precisely, Observation 2 is a consequence of [39, Lemma 12 and 13], which ensures that by fixing an exit node one makes progress towards an acyclic agreement forest, and Observation 3 is a consequence of [39, Lemma 10], which ensures that it is possible to obtain all relevant maximum acyclic agreement forests from applying the refinement procedure. Notice that, as by fixing exit nodes a minimum number of nodes are resolved and a minimum number of edges are cut, each resulting maximum acyclic agreement forest is automatically relevant. □

Now, from those two separate proofs each regarding two successive parts, namely the computation of specific relevant agreement forests followed by the refinement procedure establishing the acyclicity of each those forests, we can finally finish the proof of Theorem 29. □

4.3.4 Runtime of allMulMAAFs

In this section, we discuss the runtime of the algorithm allMulMAAFs in detail.

Theorem 32. *Let T_1 and T_2 be two rooted phylogenetic $\mathcal{X}$-trees and $\mathcal{F}$ be a maximum agreement forest for T_1 and T_2 containing k components. The theoretical worst-case runtime of the algorithm* allMulMAAFs *applied to T_1 and T_2 is $O(3^{|\mathcal{X}|+k}4^k|\mathcal{X}|)$.*

Proof. As stated in Theorem 28, the algorithm has to conduct $O(3^{|\mathcal{X}|+k})$ recursive calls. Potentially, for each of those recursive calls we have to apply a refinement step whose theoretical worst-case runtime can be estimated as follows. First notice that the order of fixing exit nodes is irrelevant. Thus, in an expanded ancestor-descendant graph, corresponding to an agreement forest of size $k+1$ and, hence, containing $2k$ exit nodes, at most 2^{2k} different sets of potential exit nodes have to be considered. As the processing of such a set of potential exit nodes takes $O(|\mathcal{X}|)$ time, the theoretical worst-case runtime of the algorithm is $O(3^{|\mathcal{X}|+k}4^k|\mathcal{X}|)$. □

In general, however, due to the following observation, the runtime of the refinement step is not a problem when computing maximum acyclic agreement forests of size k. Either the size k' of an agreement forest $\mathcal{F}$ is close to k and, thus, fixing an exit node immediately leads to an agreement forest of size larger than k (and, consequently, most of the sets of potential exit nodes have not to be considered in full extend). Otherwise, if the size k' of $\mathcal{F}$ is small and, thus, the gap between k' and k is large, the expanded ancestor-descendant graph is expected to contain no or at least only less cycles (and, consequently, there exist only few sets of potential exit nodes). Nevertheless, in the master thesis of Li [39] a method is presented that allows to half the number of exit nodes that have to be taken into account throughout the refinement of an agreement forest, so that by applying this modification the algorithm yields a theoretical worst-case runtime of $O(3^{|\mathcal{X}|+k}2^k k)$.

4.3.5 Robustness of our Implementation

In order to make the algorithm available for research, we added an implementation to our Java based software package Hybroscale [3] providing a graphical user interface, which enables a user friendly interactive handling (cf. Sec. 5.3). Next, we conducted two specific test scenarios demonstrating the robustness of our implementation which means, in particular, that Hybroscale guarantees the computation of all relevant nonbinary acyclic agreement forests for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Each of those test scenarios was conducted on a particular synthetic dataset, which was generated as described below.

4.3.5.1 Synthetic dataset

Our synthetic dataset consists of several tree sets each containing two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Each $\mathcal{X}$-tree is generated by ranging over all different combinations of four parameters, namely the number of leaves ℓ, an upper bound for the hybridization number k, the *cluster degree* c, and an additional parameter p. Each of both trees of a particular tree set corresponds to an embedded tree T of a particular network N only containing hybridization nodes of in-degree 2. With respect to the four different parameters such a tree T is computed as follows. First a random binary tree $\hat{T}$ containing ℓ leaves is computed. This is done, in particular, by randomly selecting two nodes u and v of a specific set V, which is initialized by creating ℓ nodes of both in- and out-degree 0. The two selected nodes u and v are then connected to a new node w. Finally, V is updated by replacing u and v by its parent node w. This is done until V only consists of one node corresponding to the root of $\hat{T}$. In a second step, k reticulation edges are inserted in $\hat{T}$ with respect to parameter c such that the resulting network N contains precisely k reticulation nodes of in-degree 2. Finally, after extracting a binary T' from N, based on parameter p, a certain percentage of its edges are contracted such that a nonbinary tree T is obtained from T'.

In this context, the *cluster degree* is an *ad hoc* concept influencing the computational complexity of a tree set similar to the concept of the *tangling degree* first presented in the work of Albrecht *et al.* [6] (cf. Fig. 4.12). When adding a reticulation edge e with target node v_2 and source node v_1, we say that e respects the cluster degree c, if v_1 cannot be reached from v_2 and there is a path of length less than or equal to c leading from v_2 to a certain node p such that v_1 can be reached from p. This means, in particular, that networks respecting a small cluster degree, in general, contain more minimum common clusters than networks respecting a large cluster degree and, thus, often provide a smaller computational complexity when applying a cluster reduction beforehand.

4.3.5.2 Comparison with other software

First, we generated a synthetic dataset, as described above, containing tree pairs each consisting of two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees with parameters $\ell \in \{10, 25, 50\}$, $k \in \{5, 10, 15\}$, $c \in \{1, 3, 5\}$, and $p \in \{30\}$. More specifically, for all 81 combinations

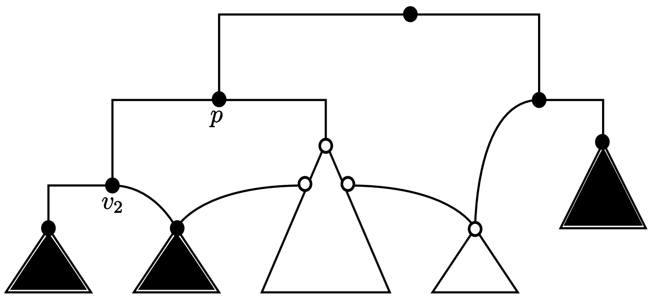


Figure 4.12: An illustration of the cluster degree parameter. Given a cluster degree $c = 1$. When inserting an in-going edge e to node v_2 that is respecting c, each node that is marked white or is part of a white marked subnetwork forms a potential source node.

of the four parameters 30 tree sets were generated resulting in 810 tree sets in total. Next, based on this dataset, we compared the result of our implementation to the two software packages Dendroscope[1] [33] and TerminusEst[2] [49] so far being the only known available software packages computing exact hybridization numbers for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees.

Our simulation study pointed out that our implementation could always reproduce the hybridization numbers that were computed by both software packages Dendroscope and TerminusEst. Moreover, the number of maximum acyclic agreement forests computed by our algorithm was always larger than the number of networks that were reported by Dendroscope and TerminusEst. Notice that, regarding TerminusEst, this is not surprising as this program does only output one network. This fact, however, gives further indication that our program is actually able to compute all relevant maximum acyclic agreement forests. Nevertheless, we applied a further test scenario examining this fact in more detail.

4.3.5.3 Permutation test

To check the robustness of our implementation in more detail, we generated a further synthetic dataset containing thousands of tree pairs of low computational complexity, such that each of those tree pairs could be processed by our implementation within less than a minute. More precisely, the dataset contains tree pairs that have been generated in respect to precisely one value for each of the four parameters ℓ, k, c, and p, i.e., $\ell \in \{10\}$, $k \in \{5\}$, $c \in \{1\}$, and $p \in \{30\}$. Next, for each of those tree pairs, we computed two sets of relevant maximum acyclic agreement forests each corresponding to one of both orderings of the two input trees and compared both results.

For each of those tree pairs, both sets of maximum acyclic agreement forests were identical, which means that each maximum acyclic agreement forest that could be computed was always contained in both sets. Notice that by switching the order of the input trees our algorithm runs through different recursive calls, which means that each computational

[1]`ab.inf.uni-tuebingen.de/software/dendroscope/`
[2]`skelk.sdf-eu.org/terminusest/`

path leading to a maximum acyclic agreement forest usually differs. Nevertheless, due to the fact that the hybridization number is independent from the order of the input trees, those two sets of maximum acyclic agreement forests have to be identical. As we applied this permutation test to thousands of different tree pairs, this is a further strong indication that Hybroscale is actually able to compute all relevant maximum acyclic agreement forests for two rooted (nonbinary) phylogenetic $\mathcal{X}$-trees.

4.3.6 Conclusion

In this section, we have presented the algorithm allMulMAAFs computing a set of relevant maximum acyclic agreement forests for two rooted nonbinary phylogenetic $\mathcal{X}$-trees. allMulMAAFs was developed in respect to the algorithm allHNetworks computing a particular set of minimum hybridization networks for two rooted binary phylogenetic $\mathcal{X}$-trees and is considered to be a first step for making this algorithm accessible to nonbinary phylogenetic $\mathcal{X}$-trees. Additionally, we have established a formal proof showing that the algorithm allMulMAAFs guarantees the computation of all relevant nonbinary maximum acyclic agreement forests.

Moreover, we have integrated our algorithm into the freely available software package Hybroscale and, by conducting two specific test scenarios, we have demonstrated the robustness of our implementation. In the next section, we will demonstrate how this algorithm can be used to extend the algorithm allHNetworks so that now minimum hybridization networks displaying the refinements of multiple rooted nonbinary phylogenetic $\mathcal{X}$-trees can be calculated.

4.4 The algorithm allMulHNetworks

In this section, we will describe the algorithm allMulHNetworks calculating all relevant hybridization networks for a set $\mathcal{T}$ of rooted nonbinary phylogenetic $\mathcal{X}$-trees. As defined later, we consider a network as being relevant, if it displays a refinement of each tree in $\mathcal{T}$ with a minimum hybridization number and additionally neither contains *stacks of hybridization nodes* nor *contractible edges*.

The algorithm is an extension of the algorithm allHNetworks using the algorithm allMulMAAFs instead of the algorithm allMAAFs as well as some additional combinatorial steps in order to cope with nonbinary nodes. Due to its complexity, in contrast to other algorithms of this thesis, we will only indicate its correctness by presenting a concept of a proof similar to the one showing the correctness of the algorithm allHNetworks. It will be future work to set up a detailed formal proof showing that this approach actually calculates all relevant minimum hybridization networks for several rooted (nonbinary) phylogenetic $\mathcal{X}$-trees.

4.4.1 Further definitions

In a first step, we give all further definitions that are crucial for describing the algorithm allMulHNetworks.

Phylogenetic trees. Given a rooted binary phylogenetic $\mathcal{X}$-tree T, throughout this section by $\overline{T}$ we refer to the tree that is obtained from T by suppressing each node of both in- and out-degree 1.

Hybridization networks. Given a hybridization network N on $\mathcal{X}$ and an edge set E' referring to an embedded rooted phylogenetic $\mathcal{X}'$-tree T' of N with $\mathcal{X}' \subseteq \mathcal{X}$, the *restricted network* $N|_{E',\mathcal{X}'}$ refers to the minimal connected subgraph T only containing leaves labeled by $\mathcal{X}'$ and edges that are either tree edges or contained in E'. Consequently, $N|_{E',\mathcal{X}'}$ is a directed graph that corresponds to $T'|_{\mathcal{X}'}$ but still contains nodes of both in- and out-degree 1, and, thus, each node in $N|_{E',\mathcal{X}'}$ can be mapped back to exactly one specific node of the unrestricted network N (cf. Fig.4.13(c)).

Relevant networks. Let N be a minimum hybridization network for a set $\mathcal{T}$ of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees. Then, if N contains a node v of in-degree of at least 3, one can generate further networks by dragging some of its reticulation edges upwards resulting in a *stack of hybridization nodes.* More precisely, such a stack is a path $(v_1, \dots, v_n)$, with $n > 1$, in which each hybridization node v_i is connected through a reticulation edge to v_{i+1} (cf. Fig. 4.14). Moreover, if N contains a node of out-degree of at least 3, by resolving this node one can generate further networks still displaying a refinement of each tree in $\mathcal{T}$.

Consequently, in order to keep the number of resulting networks small, which facilitates the work of biologists analyzing these networks, we consider a minimum hybridization

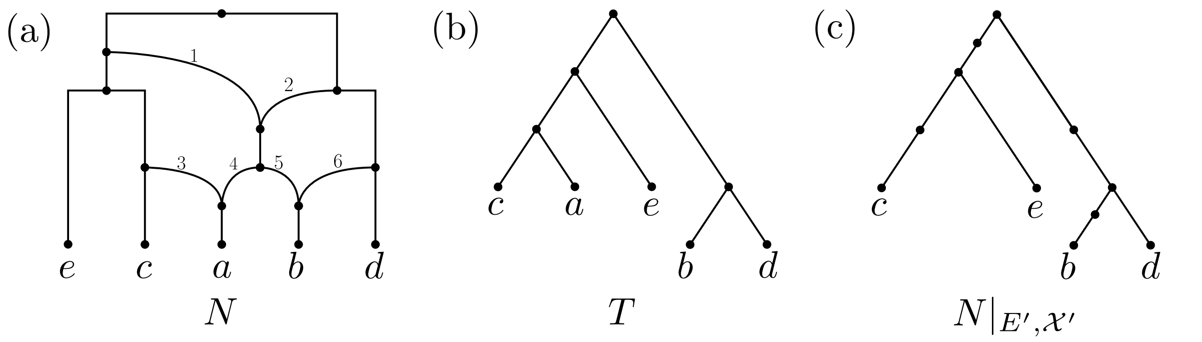


Figure 4.13: **(a)** A hybridization network N with taxa set $\mathcal{X} = \{a, b, c, d, e\}$ whose reticulation edges are consecutively numbered. **(b)** A phylogenetic $\mathcal{X}$-tree T that is displayed by N. Based on N, both edge sets $E' = \{3, 6, 1\}$ and $E'' = \{3, 6, 2\}$ refer to T and, thus, $N|_{E',\mathcal{X}}$ as well as $N|_{E'',\mathcal{X}}$ equals T. **(c)** The restricted network $N|_{E',\mathcal{X}'}$ with $\mathcal{X}' = \{b, c, d, e\}$ still containing nodes of both in- and out-degree 1.

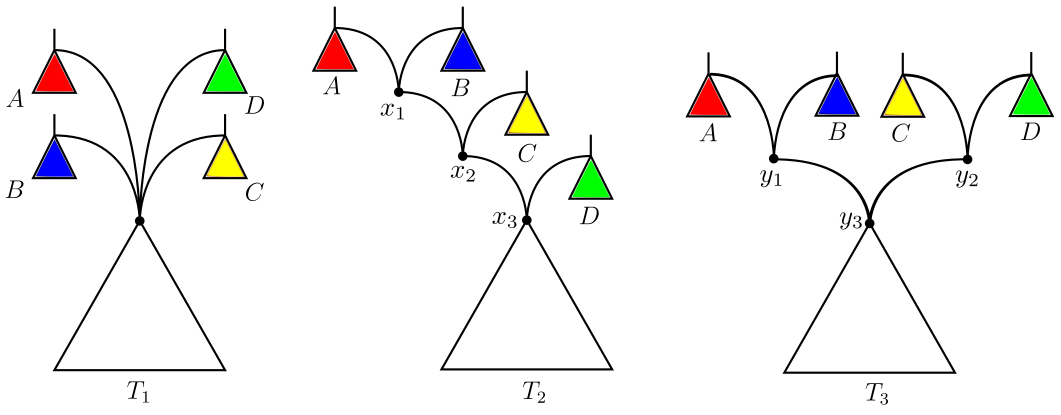


Figure 4.14: An illustration of stacks of hybridization nodes. The hybridization node with in-degree 4 of the left-hand tree T_1 can be resolved (amongst others) into distinctive stacks of hybridization nodes, e.g., (x_1, x_2, x_3) and (y_1, y_3), as demonstrated by T_2 and T_3, respectively. Notice that resolving a hybridization node into a stack of hybridization nodes does not produce new embedded trees compared with those of the unresolved network.

network N for a set $\mathcal{T}$ of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees as being relevant, if N does not contain any stacks of hybridization nodes and if, by contracting an arbitrary edge, a refinement of a tree in $\mathcal{T}$ is not displayed anymore. In the following, we will refer to those networks as *relevant networks*. Notice that such networks leave the interpretation of the ordering of hybridization events corresponding to a node of in-degree ≥ 3 as well as the ordering of speciation events corresponding to a node of out-degree ≥ 3 open. This obviously makes sense because the information for resolving these orderings is simply not given in the input.

Again, in order to improve its readability, we further demand that all hybridization nodes of a relevant network have out-degree one. Notice that in this case, in order to identify stacks of hybridization nodes, in such networks the out-edges of all hybridization nodes have to be suppressed.

4.4.2 The algorithm

Our algorithm is based on the observation that, given two rooted nonbinary phylogenetic $\mathcal{X}$-trees T_1 and T_2, an acyclic agreement forest for T_1 and T_2 of size k can be turned into a hybridization network N that displays both trees and provides a reticulation number of $k-1$. Hence, in order to calculate a network with reticulation number k displaying a set $\mathcal{T}$ of input trees, the algorithm allMulHNetworks follows a *branch-and-bound* approach conducting the following major steps each being discussed in more detail in the upcoming part of this section.

At the beginning, we choose an ordering $\Pi_{\mathcal{T}} = (T_1, T_2, \dots, T_n)$ of the input trees $\mathcal{T}$ and initialize the set $\mathcal{N}$ of networks with T_1. Next, we add each remaining input tree T_i consecutively in increasing order to a so far calculated network N in $\mathcal{N}$. This is done by first extracting a phylogenetic $\mathcal{X}$-tree T' that is displayed by N (but not necessarily contained in $\mathcal{T}$) and then by calculating a maximum acyclic agreement forest $\mathcal{F}$ for T' and T_i. Next, based on $\mathcal{F}$ and both trees T' and T_i, we can add $|\mathcal{F}| - 1$ edges to N such that the resulting network displays each of the first i input trees of $\Pi_{\mathcal{T}}$. As by adding an input tree to a so far computed network its reticulation number never decreases, one can terminate the processing of a network as soon as its reticulation number exceeds the upper bound k.

Now, in order to guarantee the computation of all minimum hybridization networks displaying $\mathcal{T}$, all possible computational paths have to be taken into account. More specifically, this means that we have to consider all possible orderings of the input trees, all trees displayed by a so far computed network and all maximum acyclic agreement forests $\mathcal{F}$ for a displayed tree T' and an input tree T_i. Moreover, as discussed later, there exist several ways of how a component of $\mathcal{F}$ can be inserted into a so far computed network N. Additionally, due to multifurcating nodes, such an insertion of a component can provoke a cascade of rearrangement operations each representing different ways of how T_i can be embedded in N.

The naive way of running our algorithm allMulHNetworks is to start the search for minimum hybridization networks with $k = 0$ and step-wise increasing k until finally networks are found displaying the entire set of input trees and having a reticulation number of k. In order to improve the runtime of this process, one can first apply an approximation algorithm calculating a lower bound for $h(\mathcal{T})$ and then setting the initial value of k to $h(\mathcal{T})$. By doing so, one can skip the very first iterations which can improve the practical runtime especially for computational complex instances.

4.4.2.1 Inserting trees into networks

Given a rooted phylogenetic network N, we say that N displays a rooted nonbinary phylogenetic $\mathcal{X}$-tree T_i, if it contains a refinement T_i' of T_i. More precisely, this is the case, if there exists a set E' consisting of reticulation edges such that $\overline{N|_{E',\mathcal{X}}}$ equals T_i'. If such an edge set E' does not exist, this directly implies that we have to insert further reticulation edges to N, which can be done by conducting the following steps.

1. Select a displayed tree T' from N referring to an edge set E' containing precisely one in-edge of each reticulation node.

2. Calculate a relevant maximum acyclic agreement forest $\mathcal{F}$ for T_i and T'.

3. Based on $\mathcal{F}$, T_i, and T' insert a new reticulation edge in N such that T_i is displayed by N. Note that this step will be discussed separately in Section 4.4.2.2.

The resulting network obviously depends on the choice of the embedded tree T'. This is the case, because different embedded trees yield different relevant maximum acyclic agreement forests which, then, provoke an insertion of different reticulation edges. Thus, in order to guarantee an output containing all relevant hybridization networks displaying $\mathcal{T}$, one has to consider all different edge sets E' each referring to an embedded tree T' and, additionally, all relevant maximum acyclic agreement forests for T' and T_i. Note that, whereas the first step can be easily performed by considering different combinations of reticulation edges, the latter step can be achieved by applying the previously presented algorithm allMulMAAFs (cf. Sec. 4.3).

Finally, in order to obtain from N a network displaying a refinement of T_i, for each component in $\mathcal{F}$ one has to take specific source and target nodes into account which can then be used to add further reticulate edges. This can be done by taking both trees T' and T_i into account as demonstrated now in the upcoming section.

4.4.2.2 Preparing networks for tree insertion

Before inserting new reticulation edges, one has to do some preparation steps resolving some parts of the network. More precisely, given a phylogenetic network N displaying a set $\mathcal{T}$ of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees as well as an edge set E' referring to an embedded tree T' of N, we can prepare N for the insertion of new reticulation edges that are necessary for displaying a further rooted (nonbinary) phylogenetic $\mathcal{X}$-tree T as follows. Let $\mathcal{F}$ be a maximum acyclic agreement forest for T' and T, then, the following steps have to be performed (cf. Fig4.15).

1. Calculate an acyclic ordering $\Pi_{\mathcal{F}} = (F_\rho, F_1, F_2, \dots, F_k)$ of $\mathcal{F}$.

2. Based on $\Pi_{\mathcal{F}}$, calculate the two trees $T'(\mathcal{F})$ and $T(\mathcal{F})$ representing $\mathcal{F}$.

3. Based on N and $T'(\mathcal{F})$, calculate the network N' by resolving some nodes of N so that the resulting network displays $T'(\mathcal{F})$. Note that, if $T'(\mathcal{F})$ is isomorphic to T', one can skip this step.

4. Based on $T'(\mathcal{F})$ and $T(\mathcal{F})$, for each component F_j in $\Pi_{\mathcal{F}}$ insert a new reticulation edge into N', beginning with F_1, consecutively in increasing order so that each resulting network displays $T(\mathcal{F})|_{\mathcal{X}_{\leq j}}$, where $\mathcal{X}_{\leq j} = \mathcal{L}(F_\rho) \cup \mathcal{L}(F_1) \cup \dots \cup \mathcal{L}(F_j)$.

More specifically, each of those steps can be conducted as follows.

Step 1. As previously described in Section 1.4.2, based on the graph $G_{\mathcal{F}} = AG(T', T, \mathcal{F})$, the first step can be conducted by gradually selecting and removing each node in $G_{\mathcal{F}}$ that has no in-going edge. Note that, as in each iteration there can occur multiple nodes of in-degree zero, for an acyclic agreement forest there usually exist more than one acyclic ordering. Moreover, due to the upcoming steps, different acyclic orderings of $\mathcal{F}$ can yield different networks displaying $\mathcal{T} \cup \{T\}$. As a consequence, in order to calculate all of those networks, for the upcoming steps one has to take all acyclic orderings of $\mathcal{F}$ into account.

Step 2. As previously described in Section 4.1, both trees $T'(\mathcal{F})$ and $T(\mathcal{F})$ can be calculated be iteratively reattaching all components of an acyclic ordering of $\mathcal{F}$ back together in a specific way. Note that, as there can exist different acyclic orderings of $\mathcal{F}$, the two trees $T'(\mathcal{F})$ and $T(\mathcal{F})$ are not unique. Consequently, in order to calculate all of those networks, for the upcoming steps one has to take all of those trees reflecting $\mathcal{F}$ into account.

Step 3. If T' is not isomorphic to $T'(\mathcal{F})$, some multifurcating nodes in N have to be resolved so that N displays $T'(\mathcal{F})$. More precisely, this is the case if $\mathcal{F}$ contains two different components F_p and F_q such that $\mathrm{LCA}_{T'}(\mathcal{L}(F_p))$ equals $\mathrm{LCA}_{T'}(\mathcal{L}(F_q))$. In this case, due to the conditions $T'(\mathcal{F})$ has to satisfy, this lowest common ancestor has to be resolved in T' which must then be transferred back to N.

Step 4. Finally, for each component of $\mathcal{F}$, except F_ρ, one has to insert a new reticulation edge, which is a more complex step that will be discussed in the upcoming section on its own.

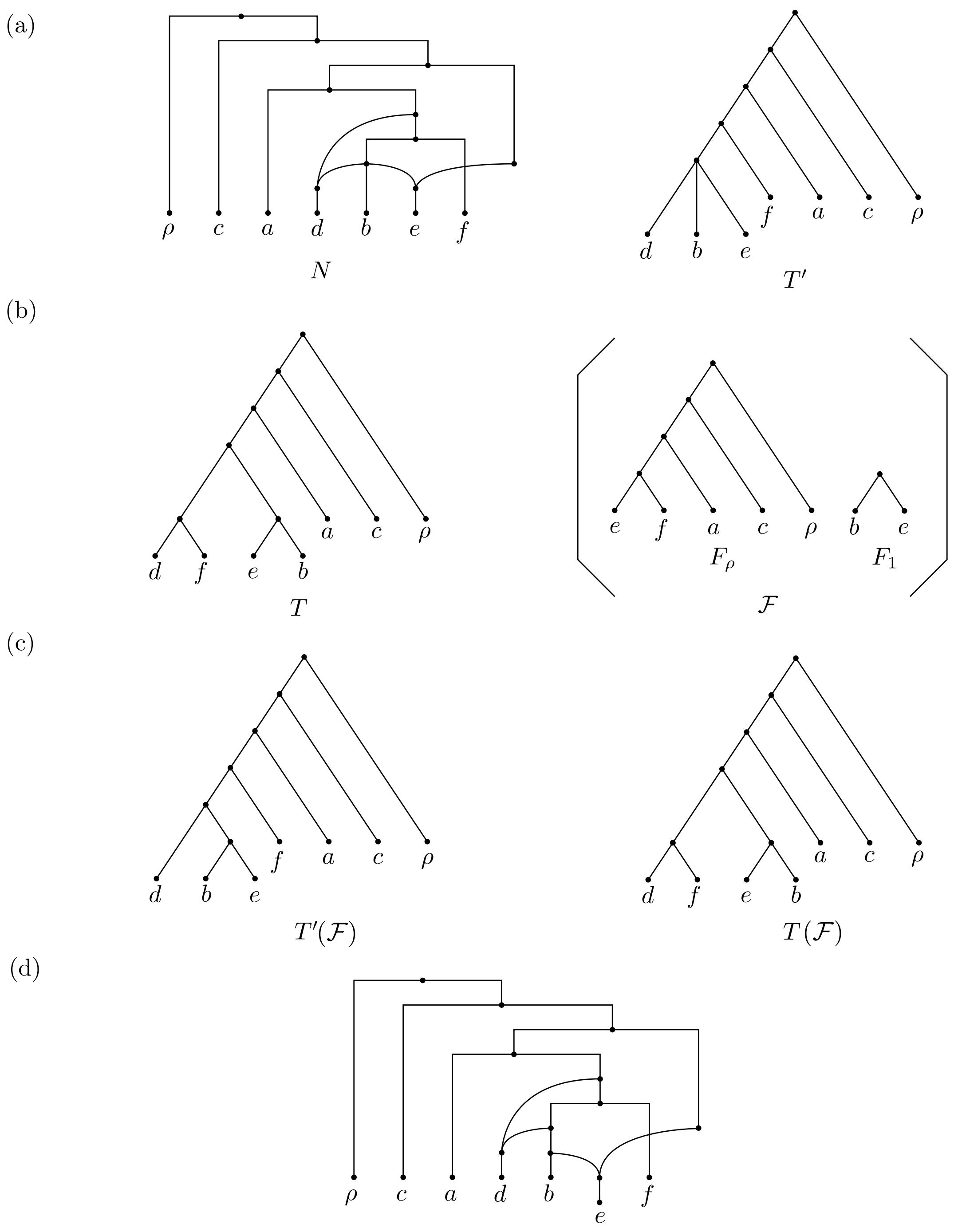


Figure 4.15: An example illustrating the preparation steps of a network N in terms of a tree T. (a) The network N displaying the tree T'. (b) The tree T together with a maximum (nonbinary) acyclic agreement forest for T and T'. (c) The two trees $T'(\mathcal{F})$ and $T(\mathcal{F})$ both representing $\mathcal{F}$. (d) The resolved network N' now displaying $T'(\mathcal{F})$ and, thus, ready for inserting T.

4.4.2.3 Inserting components into networks

In the following, let $\Pi_{\mathcal{F}} = (F_\rho, F_1, F_2, \ldots, F_k)$ be an acyclic ordering of an agreement forest $\mathcal{F}$ for two (nonbinary) phylogenetic $\mathcal{X}$-trees T' and T_i with both trees reflecting $\mathcal{F}$. Moreover, let N be a network that is obtained by applying Step 3 from above. This means, in particular, that N displays T' such that that there exists an edge set E' referring to T', which implies that T' can be extracted from N by first deleting some reticulation edges and then by suppressing all nodes of both in- and out-degree 1 (and not by additionally resolving some of its nodes). Then, similarly to the binary case, for each F_j (beginning with F_1) we can sequentially calculate a set of source and target nodes, which can then be used for the insertion of those reticulation edges that are necessary for an embedding of T_i in N.

I Computation of target and source nodes. Let $\mathcal{F}' = \{F_\rho, F_1, \ldots, F_{j-1}\} \subset \mathcal{F} = \{F_\rho, F_1, \ldots, F_k\}$ be the set of components that has already been added to the network so far and let F_j be the component that is added in the current step. Note that at the beginning $\mathcal{L}(\mathcal{F}')$ equals $\mathcal{L}(F_\rho)$, since the first component being added is the second component of the acyclic ordering which is F_1. Then, the set of source and target nodes for F_j is defined as follows.

I.I Computation of target nodes. The set $\mathcal{V}_t$ of target nodes contains each node v with $\overline{N|_{E',\mathcal{L}(\mathcal{F}')\cup\mathcal{L}(F_j)}(v)}$ isomorphic to $T_i|_{\mathcal{L}(F_j)}$. Due to the restriction of the network to $\mathcal{L}(\mathcal{F}')$ and due to other so far added reticulation edges, this set can contain more than one node. Moreover, since we are only interested in relevant networks, we omit those target nodes that are source nodes of reticulation edges. This is a necessary step preventing the computation of networks containing stacks of hybridization nodes.

I.II Computation of source nodes of *Type A*. For each edge set E_i referring to a refinement of the tree $T_i|_{\mathcal{L}(\mathcal{F}')}$ in N, the set $\mathcal{V}_s^A$ of source nodes of Type A contains all nodes v with $\overline{N|_{E_i,\mathcal{L}(\mathcal{F}')}(v)}$ being a refinement of $T_i|_{\mathcal{L}(\mathcal{F}')}(v_{\text{sib}})$, where v_{sib} denotes the sibling of the node v' with $\mathcal{L}(v') = \mathcal{L}(F_j)$ in $T_i|_{\mathcal{L}(\mathcal{F}')\cup\mathcal{L}(F_j)}$. Note that, due to the restriction of the network to $\mathcal{L}(\mathcal{F}')$, this set can contain several source nodes. However, as we want to construct networks in which each hybridization node has out-degree one, we disregard those nodes having more than one in-edge.

I.III Computation of source nodes of *Type B*. The set $\mathcal{V}_s^B$ of source nodes of *Type B* contains each node v of a subtree that, on the one hand, is attached to a node in $\mathcal{V}_s^A$ and, on the other hand, does not contain any taxa of $\mathcal{L}(\mathcal{F}')$. Moreover, either $\mathcal{L}(F)\cap\mathcal{L}(v) = \emptyset$ or $\mathcal{L}(F) \subseteq \mathcal{L}(v)$ must hold for each component $F \neq F_\rho$ in $\mathcal{F}$. This means, in particular, that v must not be part of a subtree rooted at a source node corresponding to a component, which is added afterwards. Lastly, since we want to construct networks in which each hybridization node has out-degree one, we disregard those nodes having more than one in-edge.

Moreover, we can classify each source node $s \in \mathcal{V}_s^A \cup \mathcal{V}_s^B$ by the three Types α, β, and γ, which have to be taken into account in respect to the definitions of relevant networks (cf Fig. 4.17).

Type $\boldsymbol{\alpha}$. We say that s is of Type α, if we can directly use this node as source node for the insertion of a reticulation edge necessary for displaying T_i. More specifically, this is the case if, on the one hand, based on the chosen edge set E_i referring to the embedded tree $T_i|_{\mathcal{L}(\mathcal{F}')}$, the number of out-going edges of s that are necessary for displaying $T_i|_{\mathcal{L}(\mathcal{F}')}$ is one. Or, on the other hand, if $\text{LCA}_{T_i}(\mathcal{L}(\mathcal{F}'))$ equals $\text{LCA}_{T_i}(\mathcal{L}(\tilde{\mathcal{F}}) \cup \mathcal{L}(F_j))$ with $\tilde{\mathcal{F}} \subseteq \mathcal{F}'$, which implies that the root of the subgraph $T_i|_{\mathcal{L}(\tilde{\mathcal{F}}) \cup \mathcal{L}(F_j)}$ is a nonbinary node.

Type $\boldsymbol{\beta}$. Let p be the parent of s. Notice that p is unique, since as source nodes only those nodes of in-degree one have been selected. Now, if p is of Type α and s is not of Type α, we say that s is of Type β. We will neglect source nodes of Type β, since otherwise, by inserting reticulation edges in terms of such nodes, we would generate edges that could be contracted and, thus, the resulting networks would not be relevant.

Type $\boldsymbol{\gamma}$. If s is neither of Type α nor of Type β, we say that s is of Type γ.

II Inserting new reticulation edges. Now, for each pair (s, t) of source and target nodes, we first check if (s, t) is *valid*, which is the case if s cannot be reach from t and s is not of Type β. If this is the case, we can insert a new reticulation edge by first generating two nodes s' and t' and then by connecting those two nodes, as described now in the following.

1. If s is of Type γ, its in-edge is split by inserting a new node s', i.e., $e = (p, s)$ is first deleted and then two new edges (p, s') ans (s', s) are inserted (cf. Fig. 4.16 (a)). Otherwise, if s is of Type α, there is no need for inserting an extra node and, thus, s directly acts as s' (cf. Fig. 4.16 (b)).

2. If the parent of t has in-degree one, the in-edge of t is split two times in the same way as described above by inserting two nodes t' and t'' (cf. Fig. 4.16 (c)). Let t' be the parent of t after splitting its in-edge. In this case, notice that t' is necessary to receive only hybridization nodes of out-degree one and t'' is necessary to provide an attaching point for upcoming reticulation edges as discussed below. Otherwise, if t has an in-degree of at least two, t' is set to t, which prevents the computation of networks containing stacks of hybridization nodes (cf. Fig. 4.16 (d)).

3. Finally, the two nodes s' and t' are connected through a path P consisting of two edges. Since we do not allow nodes of in-degree larger than one as source nodes, this creates an attaching point for upcoming reticulation edges within already inserted reticulation edges. Consequently, in each final network, in which all input trees have

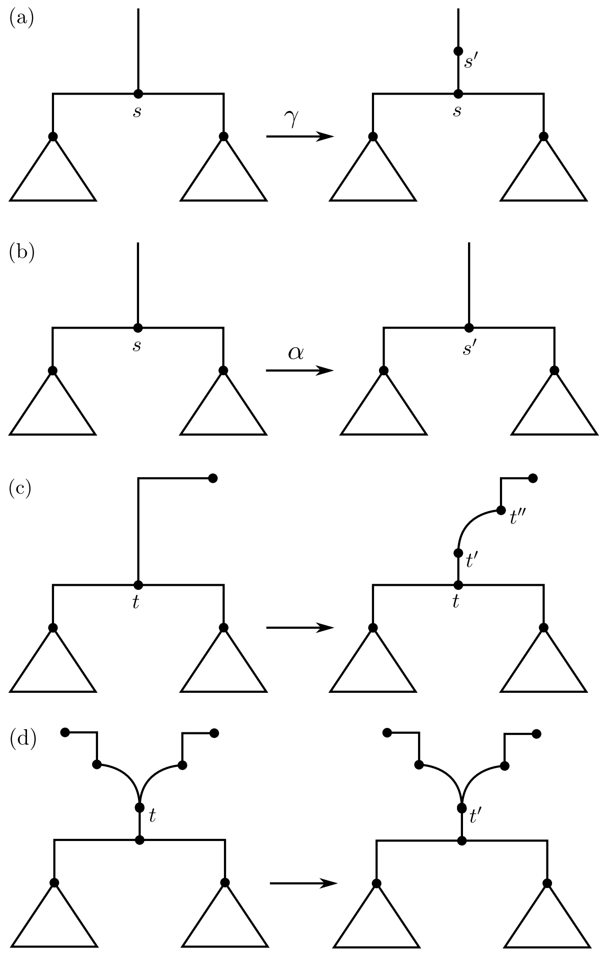


Figure 4.16: An illustration of generating a source node s' if s is of Type γ **(a)**, a source node s' if s is of Type α **(b)**, a target node t' if t has in-degree one **(c)**, and a target node t' if t has in-degree two **(d)** for adding new reticulation edges as described in Step II.

been inserted successfully, one still has to suppress all reticulation edges of both in- and out-degree one (cf Fig. 4.18).

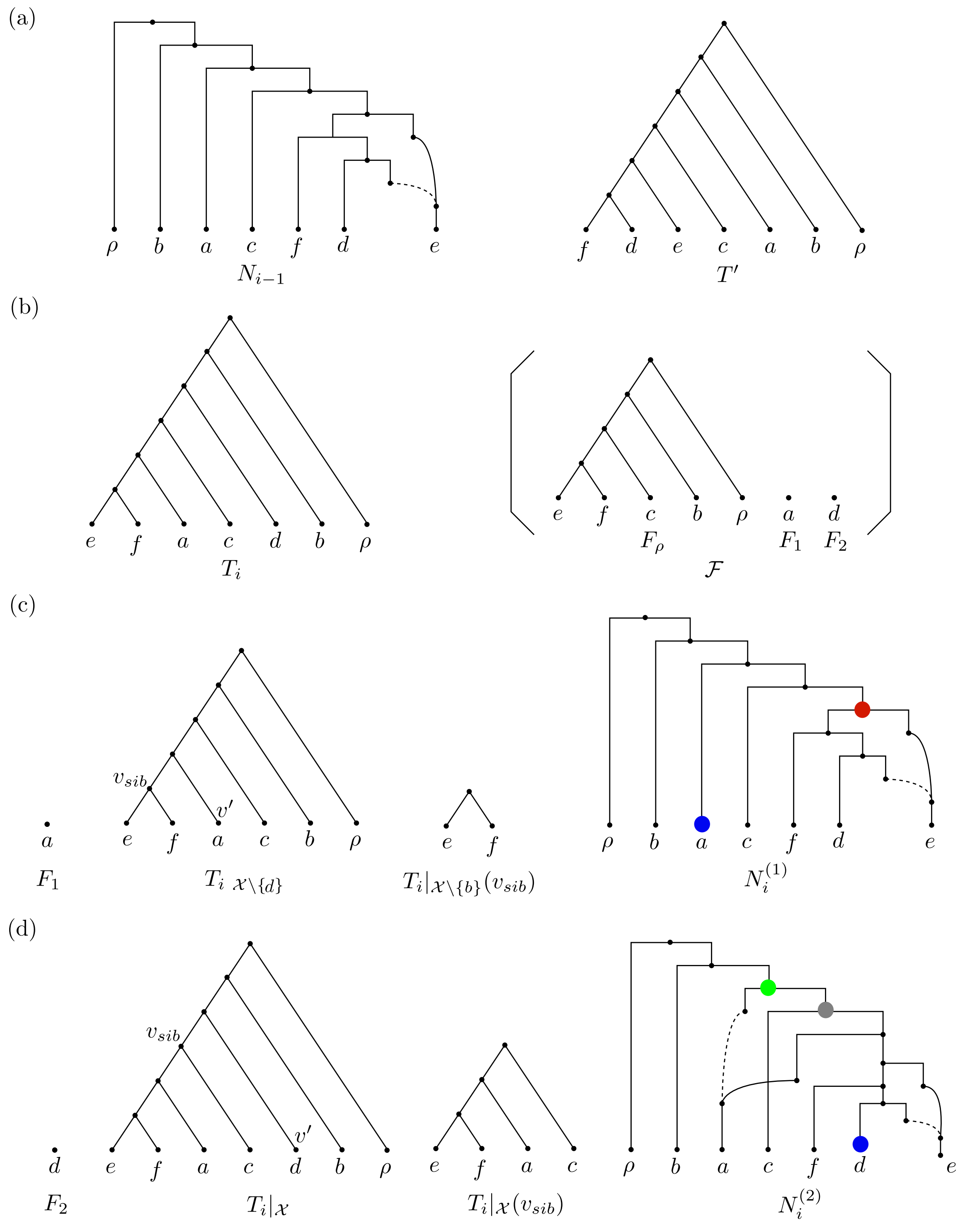
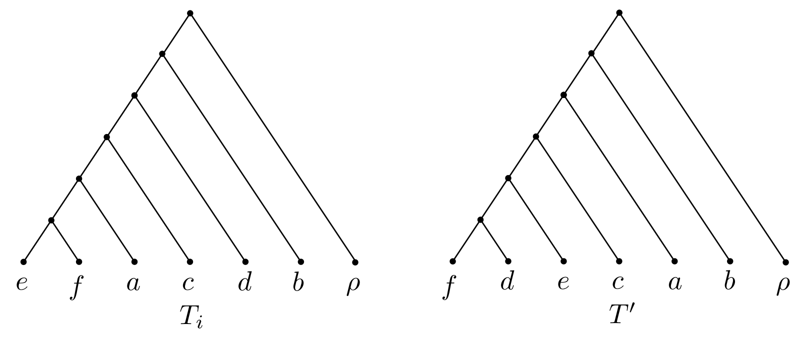


Figure 4.17: An illustration of how a tree T_i is inserted into a network N_{i-1}. **(a)** The network N_{i-1} displaying the tree T'. **(b)** The tree T_i that is inserted into N_{i-1} in respect of the maximum acyclic agreement forest $\mathcal{F}$ of T_i and T' consisting of three components F_ρ, F_1, and F_2. **(c,d)** All important elements that have to be considered during the insertion of both components F_1 and F_2. Blue dots correspond to source nodes and green, grey, and red nodes to target nodes of Type α, β, and γ, respectively.

(a)



(b)

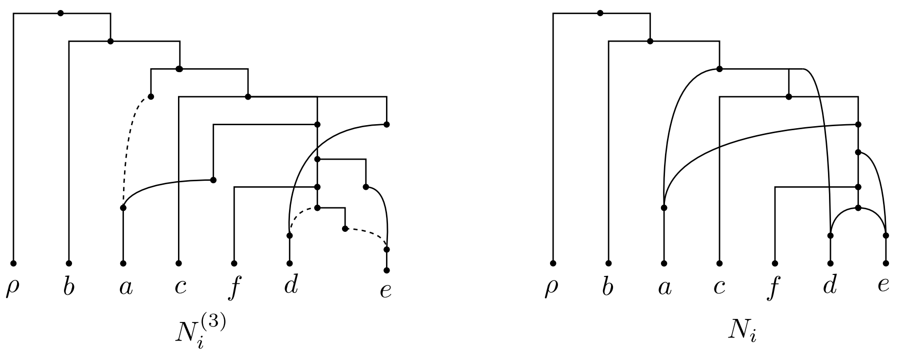


Figure 4.18: Continued example from Figure 4.17. **(b)** The resulting network N_i, which is obtained first from $N_i^{(2)}$ by inserting a reticulation edge in respect of the green and the blue node and then from $N_i^{(3)}$ by suppressing each node of both in- and out-degree 1. Notice that in N_i now T_i (and still T') is displayed.

III Application of shifting-patterns of Type 1. In order to obtain all relevant networks, based on a just inserted reticulation edge, one still has to apply specific shifting operations, which are separately addressed in Section 4.4.2.4.

Now, in order to compute all relevant networks, one has to generate for each valid pair (s, t) of source and target nodes and for each application of a shifting-pattern of Type 1 a new network $\hat{N}$. This is necessary, since each of those networks contains a different set of embedded trees which can then be used for the insertion of further input trees and, thus, can initiate new computational paths leading to relevant networks. Lastly, for each network $\hat{N}$, now displaying a refinement of each so far added input tree (including T_i), one still can apply further shifting-patterns producing additional relevant networks.

III Application of shifting-patterns of Type 2. In contrast to the algorithm ALL-HNETWORKS, to each network $\hat{N}$ one still has to apply further shifting operations. The application of those shifting-patters are separately addressed in Section 4.4.2.5.

In the following, as these steps are quite sophisticated, we will only motivate the application of those shifting-patterns of Type 1 and 2 which means that we will omit a detailed description of how those patterns have to be applied in detail to a given network. Broadly speaking, the challenge when applying those patterns in an exhaustive way in order to obtain all relevant networks remains in taking care on the following.

1. After the application of a shifting-pattern still all so far inserted input trees have to be displayed by the network.
2. By applying a shifting-pattern one possibly generates contractible edges that have to be removed afterwards.
3. In order to prevent the algorithm from running into cycles, one has to reject those shifting-patterns producing an already calculated network.

4.4.2.4 Application of shifting-patterns of Type 1

Let e be a reticulation edge that has just been inserted for a component of a maximum acyclic agreement forest for T_i and a certain embedded tree by applying the steps from above. Moreover, let $\mathcal{T}' \subset \mathcal{T}$ be those input trees that have already been inserted into N so far. Then, due to nonbinary nodes, one still has to apply particular shifting-patterns in order to generate all relevant networks dealing with edges that have already existed before the insertion of T_i. Before going into details, we have to introduce some further definitions that are crucial for what follows.

Due to certain previously discussed circumstances, the source of a reticulation edge $e = (s, t)$ within a so far computed network is of both in- and out-degree one. Consequently, in the following, we will refer to the highest ancestor of s that can be reached through edges only necessary for displaying parts of T_i as the *highest ancestor of e in respect to T_i*, shortly

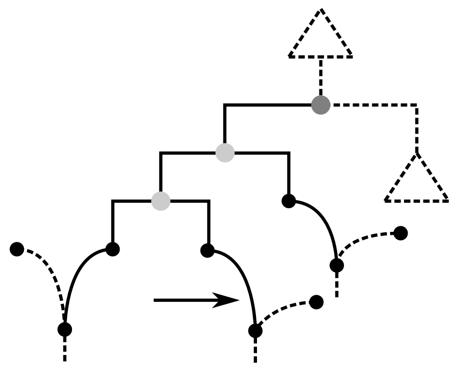


Figure 4.19: An illustration of a network displaying the refinements of several trees, say $\mathcal{T} \cup \{T_i\}$. In this context, in contrast to the dashed edges, the plain edges are only necessary for displaying T_i. Moreover, the node colored dark grey refers to the highest ancestor of the edge marked with an arrow in respect to T_i. Furthermore, each node colored light grey as well as the node colored dark grey, is a proper successor of this highest ancestor.

denoted by $\text{HA}(e, T_i)$. Moreover, we call those nodes of in-degree one that can be reached from s (including s) by only visiting edges only necessary for displaying parts of T_i as the *proper set of successors of* $\text{HA}(e, T_i)$. Similarly, due to certain circumstances discussed in the definition of a relevant network, the target t of a reticulation edge within a so far computed network is one. Consequently, in the following, we will refer to the target of the out-edge of t as the *proper target of* e. Lastly, given two nodes v and w of a hybridization network N for a set $\mathcal{T}$ of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees, we call the reattaching of an out-going edge of w to v *a legal shifting operation from* w *to* v, if the so modified network still displays a refinement of each tree in $\mathcal{T}$. Moreover, we demand that such a shifting operation does not produce any contractible edges which implies that, if before applying a shifting operation the network does match the definition of a relevant network, this also holds afterwards.

In the following, we will present four patterns affecting those edges that have already existed before all parts of T_i have been inserted. By applying the first three of these patterns, edges are shifted upwards which means that an edge is reattached to a predecessor of its original source, whereas by applying the fourth pattern edges are shifted downwards which means that edges are reattached to a successor of its original source (cf. Fig. 4.20).

Shifting-pattern of Type 1a. Let e be a reticulation edge inserted for displaying T_i so that $\text{HA}(e, T_i)$ has exactly two out-going edges e_1 and e_2, where e_1 is only necessary for displaying T_i. Then, one possibly can apply legal shifting operations from the target of e_2 to each proper successor of $\text{HA}(e, T_i)$.

Shifting-pattern of Type 1b. Let e be a reticulation edge inserted for displaying T_i so that $\text{HA}(e, T_i)$ has at least three out-going edges. Moreover, let E' contain those out-going edges that are not only necessary for displaying T_i. Then, one can generate new relevant networks, if one can conduct a certain preparation step followed by certain legal

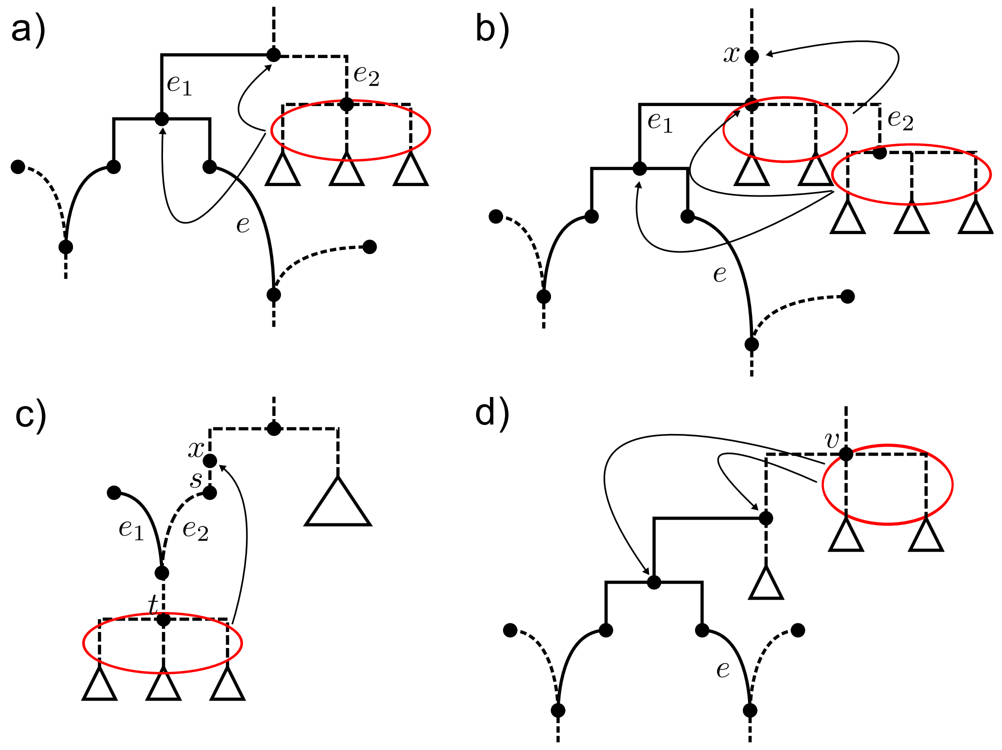


Figure 4.20: An illustration of the scenario referring to the description of a shifting-pattern of Type 1a–d. In this context, in contrast to the dashed edges, the plain edges are only necessary for displaying T_i.

shifting steps. During such a preparation step, one first inserts a node x into the in-edge of $\mathrm{HA}(e, T_i)$ and, then, has to be able to apply exactly $|E'| - 1$ legal shifting operations from $\mathrm{HA}(e, T_i)$ to x so that in the resulting network $\mathrm{HA}(e, T_i)$ has only two out-going edges e_1 and e_2. If this is possible, one possibly can apply further legal shifting operations from the target of e_2 to each proper successor of $\mathrm{HA}(e, T_i)$, where e_2 refers to the out-edge of $\mathrm{HA}(e, T_i)$ that is not only necessary for displaying T_i.

Shifting-pattern of Type 1c. Let s be the source and t be the proper target of a reticulation edge e_2 being adjacent to a reticulation edge e_1 that is only necessary for displaying T_i. Then, by first inserting a new node x into the in-edge of s, one can possibly apply legal shifting operations from t to x.

Shifting-pattern of Type 1d. Let e be a reticulation edge inserted for displaying T_i and let v be the source of the in-edge of $\mathrm{HA}(e, T_i)$. Then, one possibly can apply legal shifting operations from v to each proper successor of $\mathrm{HA}(e, T_i)$.

4.4.2.5 Application of shifting-patterns of Type 2

Now, let $\hat{N}$ be a network being the result from step-wise inserting all reticulation edges necessary for embedding T_i each followed by potentially applying a shifting-pattern of Type 1. Then, in order to receive all relevant networks, one still may have to apply some further legal shifting operations as indicated in the following. Note that, in contrast to the description of the shifting-patterns of Type 1, now the entire tree T_i is embedded in $\hat{N}$ and those patterns of Type 2 have to be applied to all edges (not necessarily reticulation edges) that have been inserted for its embedding.

Given a hybridization network N for a set $\mathcal{T}$ of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees and a reticulation edge only necessary for displaying the lastly added input tree T_i, we reuse those definitions from above, including the definition of a *highest ancestor* and its set of *proper successors* as well as the definition of a *legal shifting operation* from one node v to another node w.

Shifting-pattern of Type 2a. Let e be a reticulation edge inserted for displaying T_i so that $v = \text{HA}(e, T_i)$ provides a set E' of more than two out-going edges, where $e_1 = (v, w_1) \in E'$ is only necessary for displaying T_i and $e_2 = (v, w_2) \in E'$ is a tree edge not necessary for displaying T_i. Then, the reattaching of e_1 to w_2 by first deleting e_1 and then inserting a new edge $e'_1 = (w_2, w_1)$ might represents a legal shifting operation from w_1 to w_2. Moreover, if in the so modified network w_2 is a nonbinary node, one possibly can apply further shifting operations from w_2 to each proper successor of $\text{HA}(e, T_i)$.

Shifting-pattern of Type 2b. Again, as described above, let e be a reticulation edge inserted for displaying T_i so that $v = \text{HA}(e, T_i)$ provides a set E' of more than two out-going edges, where $e_1 = (v, w_1) \in E'$ is only necessary for displaying T_i and $e_2 = (v, w_2) \in E'$ is a tree edge not necessary for displaying T_i. Then, one can possibly generate new relevant networks by applying two kinds of legal shifting operation. During the first shifting operation, one first inserts a node x into e_2 and then reattaches e_1 to x by first deleting e_1 and then inserting a new edge $e'_1 = (x, w_1)$. In a second step, one has to be able to apply at least one legal shifting operation from w_2 to x so that in the resulting network x has more than two out-going edges. If this is possible, one possibly can apply further legal shifting operations from x to each proper successor of $\text{HA}(e, T_i)$. Note that therefor, however, $\text{HA}(e, T_i)$ has to provide at least two proper successors.

An illustration of these shifting-patterns can be found in Figure 4.21.

4.4.3 Runtime

Compared with our algorithm allHNetworks, in order to deal with nonbinary nodes, the presented algorithm allMulHNetworks has to conduct some additional combinatorial steps. Generally speaking, these steps include the calculation of all trees reflecting a given maximum acyclic agreement forest as well as the execution of specific shifting-patterns

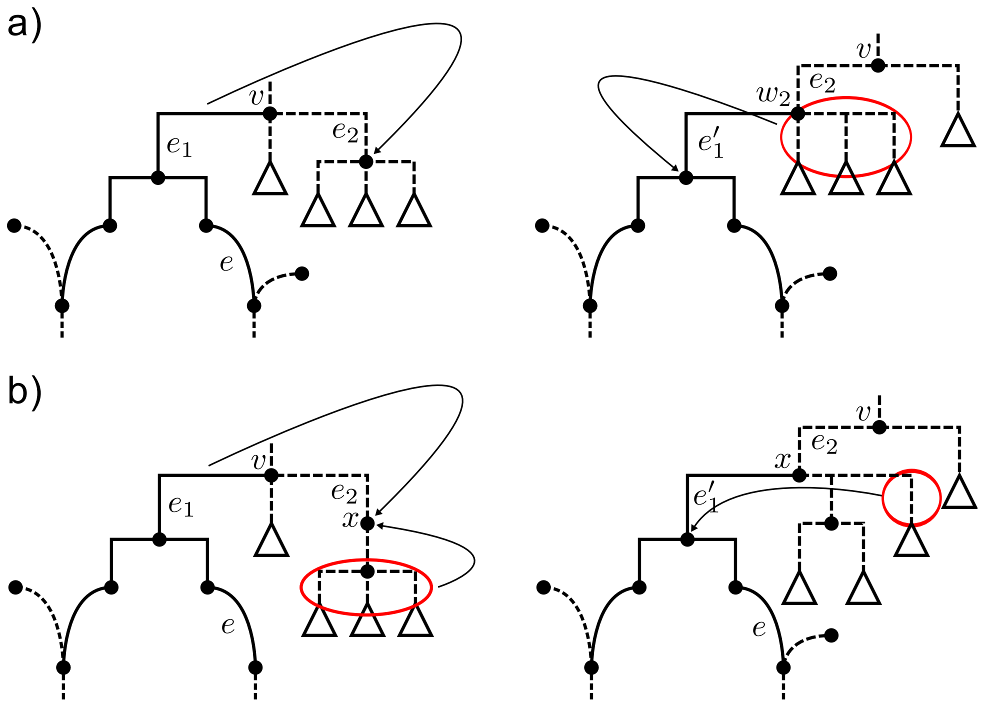


Figure 4.21: An illustration of the scenario referring to the description of a shifting-pattern of Type 2a,b. In this context, in contrast to the dashed edges, the plain edges are only necessary for displaying T_i.

rearranging edges of a so far calculated network. As those additional steps merely increase the complexity of the algorithm allHNetworks, we omit here a detailed discussion of the theoretical worst-case runtime and refer interested readers to Section 3.4.5, where the theoretical worst-case runtime of its binary variant is given.

4.4.4 Correctness

Although we omit a formal proof here, we think that the basic idea dealing with the correctness of the algorithm allHNetworks can be reused to show the correctness of the algorithm allMulHNetworks, i.e., that, given a set $\mathcal{T}$ of several rooted (nonbinary) phylogenetic $\mathcal{X}$-trees, the algorithm allMulHNetworks calculates all relevant networks for $\mathcal{T}$.

More precisely, this proof should consider the following points. Let N' be a relevant network displaying refinements of a subset $\mathcal{T}'$ of all input trees $\mathcal{T}$ and let N be a hybridization network (not containing any stacks of hybridization nodes as well as any contractible edges) that is based on N' and displays a further refinement of an input tree $T_i \notin \mathcal{T}$. Then, one can show that N can be obtained from N' as follows. First, by inserting a set E' of reticulation edges whose source and target nodes can be derived from an (nonbinary) acyclic agreement forest $\mathcal{F}$ for T_i and some tree that is displayed by N' and, additionally, reflects $\mathcal{F}$ and, second, by exhaustively applying all possible sequences of shifting-patterns as described above in Section 4.4.2.4 and 4.4.2.5. Moreover, one has to show that by constructing networks for all possible orderings of the input trees, it suffices to take only (nonbinary) maximum acyclic agreement forests into account.

4.4.5 Speeding up the algorithm

Similar to the algorithm ALLHNETWORKS, in order to improve the practical runtime, one can apply each technique presented in Section 3.4.6 including the application of the subtree and the cluster reduction in its typical way (cf. Sec. 3.2).

Although we omit a proof here, we think that the proof given in Section 3.3, dealing with the correctness of the cluster reduction for multiple binary trees, can be easily adapted to multiple nonbinary trees. This means, in particular, that by summing up the hybridization number calculated by the algorithm ALLMULHNETWORKS for each single cluster represents the hybridization number of the initial unreduced set of input trees. Moreover, due to a permutation test that is described below, we think that the linking-patterns given in Section 3.4.6.2, which can be applied to restore each missing network when reattaching back the networks separately calculated for each cluster, can be also used in this context in combination with the shifting-patterns presented in Section 4.4.2.4 and 4.4.2.5.

4.4.6 Robustness of our Implementation

In order to make the algorithm ALLMULHNETWORKS available for research, we added an implementation to our Java based software package HYBROSCALE [3] providing a graphical user interface, which enables a user friendly interactive handling (cf. Sec. 5.3). Notice that we also applied the cluster reduction involving specific linking-patterns in order to obtain all missing networks that could otherwise only be calculated by omitting this particular reduction step.

Next, we conducted a permutation test indicating the robustness of our implementation which means, in particular, that HYBROSCALE enables the computation of all relevant networks for a set of rooted (nonbinary) phylogenetic $\mathcal{X}$-trees.

4.4.6.1 Synthetic dataset

The synthetic dataset used for the permutation test was generated as already described in Section 4.3.5.1. More specifically, the dataset contains thousands of tree pairs that have been generated in respect to precisely one value for each of the four parameters ℓ, k, c, and p, i.e., $\ell \in \{10\}$, $k \in \{5\}$, $c \in \{1\}$, and $p \in \{30\}$. Consequently, each of those tree pairs is of relatively low computational complexity such that it can be processed by our implementation in less than one minute.

4.4.6.2 Permutation test

For each tree pair $\{T_1, T_2\}$ of the synthetic dataset, we calculated two sets of relevant networks; one corresponding to (T_1, T_2) and the other one corresponding to (T_2, T_1). Notice that by switching the order of the input trees our algorithm runs through different recursive calls, which means that each computational path leading to a relevant network usually differs. Nevertheless, due to the fact that the set of relevant networks is independent from the order of these input trees, those two calculated sets have to be identical.

This permutation test helped us to identify those shifting-patterns previously described in Section 4.4.2.4 and 4.4.2.5. Moreover, by additionally calculating relevant networks with and without applying a cluster reduction, we could recognize that the linking-patterns for reattaching all cluster networks, previously presented in Section 3.4.6.2, can be also used in this case. The reader, however, should keep in mind that the application of those shifting-patterns (as well as of those linking-patterns) is a quite sophisticated combinatorial step and, thus, for implementing the algorithm allMulHNetworks in an appropriate way one has to spent much time and effort.

4.4.7 Conclusion

In this section, we have presented the algorithm allMulHNetworks, which is an extension of the algorithm allHNetworks that can now deal with nonbinary input trees. Consequently, in contrast to its binary variant, one has to apply additional steps including the computation of nonbinary agreement forests as well as the execution of certain shifting-patterns. In this context, especially the latter step is what makes the algorithm quite sophisticated and, thus, quite difficult to implement. Nevertheless, the correctness of the algorithm directly depends on applying those shifting-patterns in an exhaustive way. This means, in particular, omitting some of these shifting-patterns could mean that computational paths leading to minimum hybridization networks are not visited and, thus, either some relevant networks are missing or only those hybridization networks are calculated not providing a minimum hybridization number.

Chapter 5

Studying phylogenetic networks

In this chapter, we first demonstrate that phylogenetic networks are not free of interpretive challenges and present some network constraints that can be used for this purpose. Moreover, we present our software package HYBROSCALE providing implementations of each algorithm previously presented in this thesis. Additionally, we point out that the graphical user interface of HYBROSCALE provides several important features that are of high interest for studying reticulate evolution.

5.1 Filtering relevant networks

As already discussed in the previous part of this thesis, there often exist a huge number of different ways of how trees can be reconciled into phylogenetic networks (or minimum hybridization networks). Consequently, due to the vast network space, so far existing network building methods are usually impractical for many biological problems. Moreover, in contrast to phylogenetic trees, for phylogenetic networks there do not exist standard techniques for calculating specific values supporting biologists in figuring out the most promising networks. This means, in particular, even though if one has successfully applied a network building method, one possibly still has to spent a lot of time and effort in analyzing the output. Notice that, as until now there only exist few methods calculating more than one hybridization network, in this context this is still a poorly understood problem, which has not attracted much attention so far.

As a consequence, in order to overcome these problems, Kelk *et. al* [37] suggested that biologists should be able to generate individual constraints describing the kind of networks that are worth to be taken into account. On the one hand, such constraints could be used *a priori* to prune the search space that is examined by network-building methods so that the runtime of those methods gets improved. On the other hand, however, such constraints could be also used *a posteriori* to filter a large set of reported networks so that, based on previously made investigations, biologists have to study only those networks they consider as being relevant under a particular evolutionary hypothesis. In the following, we give some examples of how such constraints could look like in respect of minimum hybridization networks. Moreover, we demonstrate the benefit of such constraints by applying some of them to a well-known grass (*Poaceae*) dataset.

5.1.1 Further definitions

Before going into details, we have to introduce further definitions that are crucial for describing some of the following network constraints.

Biconnected components. A *biconnected component* of a phylogenetic network N is a maximal induced subgraph not containing any cut nodes (or articulation nodes), i.e., nodes whose removal would disconnect N.

Time-consistent networks. The definition of *time-consistent networks* is motivated by the concept of *real-time hybrids* first introduced by previous work of Baroni *et al.* [9]. Given a network N, we say that N *is time-consistent* if it contains a time-consistent labeling, which can be computed by using a particular *digraph* $D_N = ([V], [E])$ that is defined in Baroni *et al.* [9] as follows.

Let $N = (V, E)$ be a hybridization network, then, each $v \in V$ is part of a node $[v]$ in D_N with

$$[v] = \{v\} \cup \{u \in V : \exists \text{ an undirected path of hybridization edges leading from } u \text{ to } v\}.$$

Moreover, two nodes $[v]$ and $[w]$ in D_N are connected via an edge if there exists a node $a \in [v]$ and a node $b \in [w]$ so that $(a, b) \in E$. An example illustrating the concept of such a digraph can be found in Figure 5.1.

Now, based on such a digraph, N has a time-consistent labeling, and thus is time-consistent, if and only if D_N is acyclic (cf. [9, Theorem 3]).

5.1.2 Network constraints

In general, one can distinguish between two different types of network constraints. Either such a constraint refers to each hybridization event itself or takes the interaction of several hybridization events into account. Consequently, constraints can be of rather low or high complexity. In the following we will give an example for both of those types. Notice, however, that one could also think of other types of network constraints. For instance, one could be also interested in only those networks consisting of particular subtrees providing a specific set of taxa.

Tree-child property. A network N fulfills the *tree-child property*, if from each inner node there exists a directed path leading to a leaf not containing any hybridization nodes. More precisely, for each inner node v_0 there exists a directed path $P = (v_0, v_1, \ldots, v_n)$ with v_n being a leaf such that $\delta^-(v_i) = 1$ for each v_i with $i \geq 1$.

Tree-sibling property. A network fulfills the *tree-sibling property*, if each hybridization node has at least one non-hybridization node as sibling.

Minimum-level constraint. Let N be a bicombining rooted phylogenetic network. Then, we say N *is a level-k network*, if any biconnected component of N properly contains at most k hybridization nodes. This means, in particular, that a bicombining network with only one hybridization node is a level-1 network. Consequently, given a set $\mathcal{N}$ of hybridization networks, we say that a level-k network fulfills the *minimum-level constraint* (henceforth ML-constraint), if $\mathcal{N}$ does not contain any networks whose level is smaller than k.

Direct-hybridization constraint. The *direct-hybridization constraint*, henceforth DH-constraint, defines a subset of taxa that have to be direct descendants of a hybridization event. More precisely, given a set of phylogenetic $\mathcal{X}$-trees, a network N fulfills a DH-constraint for a subset $\mathcal{X}'$ of $\mathcal{X}$, if N contains a hybridization node v with $\ell(v) = \mathcal{X}'$, where $\ell(v)$ denotes the set of taxa that can be reached from v disregarding hybridization edges. An example is given in Figure 5.3

Min-time-consistency constraint. The *min-time-consistency constraint*, henceforth MTC-constraint, filters all those networks providing a *time-consistent labeling with minimal costs*. More precisely, let $N = (V, E)$ be a hybridization network and let $V' \subseteq V$ be those nodes with $\delta^-(v) = 1$, which are all nodes except all hybridization nodes as well as the root. Then, we say that N *has a time-consistent labeling with costs* k, henceforth shortly denoted by $\tau(N) = k$, if there exists a mapping $f : V \to \mathbb{N}$ satisfying the following properties.

(i) Let v be a node with $\delta^-(v) \geq 2$ and u be an immediate ancestor of v, then $f(u) = f(v)$ must hold.

(ii) Let v be a node with $\delta^-(v) = 1$ and u be an immediate ancestor of v, then $f(u) \neq f(v)$ must hold.

(iii) There exist exactly $|V'| - k$ nodes in V', such that for each of such a node v with immediate ancestor u, $f(u) < f(v)$ holds .

(iv) There exist exactly k nodes in V', such that for each of such a node v with immediate ancestor u, $f(u) \geq f(v)$ holds. Notice that this is a direct consequence of property (ii) and (iii).

Notice that, in respect to these properties, we assume that the out-degree of each hybridization node in N is either 0 or at least 2, but *not* 1 as it is usually the case in order to increase the readability of these graphs. An example of such a network can be found in Figure 5.1.

Practically, we tackle this problem by solving a *Directed Feedback Arc Set* problem on the underlying digraph D_N. More specifically, we are looking for the smallest set of

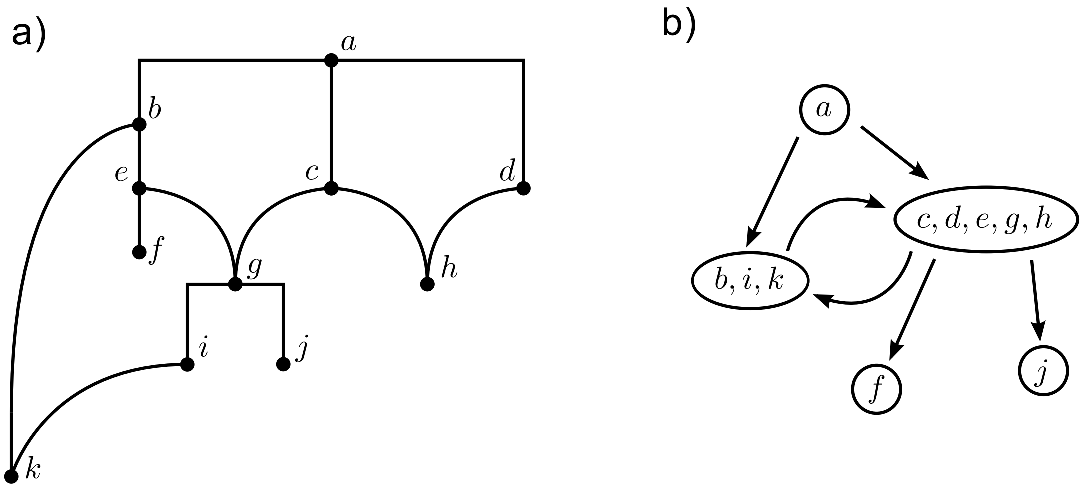


Figure 5.1: (a) A hybridization network N with $\tau(N) = 1$ and (b) its corresponding digraph D_N. Note that, in order to turn N into a time-consistent network, the edge (b, k) is *mandatory* which means that to each tree that is displayed by using this edge a new taxon has to be added.

edges that has to be deleted from D_N so that each cycle within the graph is removed. In practice, this can be done by testing for each sequence of nodes $\Pi([V]) = ([v_1], \dots, [v_n])$, if $w(\Pi([V]))$ is minimal, where

$$w(\Pi([V])) = \sum_{i=1}^{n} \delta^-([v_i], D_N - \{[v_1], \dots, [v_{i-1}]\}),$$

with $\delta^-([v_i], D_N - \{[v_1], \dots, [v_{i-1}]\})$ denoting the in-degree of $[v_i]$ in the digraph restricted on each node $[v] \notin \{[v_1], \dots, [v_{i-1}]\}$. For this purpose, we use a *branch-and-bound* approach, stopping the processing of each sequence $\Pi([V'])$, with $[V'] \subseteq [V]$, as soon as $w(\Pi([V']))$ exceeds the minimum costs of all so far considered sequences.

Min-add-taxa constraint. The *min-add-taxa constraint*, henceforth MAT-constraint, is based on the work of Linz *et al.* [42] introducing the AddTaxa problem for hybridization networks (as well as for HGT networks). More precisely, given a hybridization network N for a set $\mathcal{T}$ of phylogenetic trees, the problem remains in computing the minimum number of new so far unsampled or extinct taxa that have to be added to $\mathcal{T}$ such that there exists a time-consistent hybridization network N' displaying the so modified input trees $\mathcal{T}'$ (and still providing the same hybridization number as N). Notice that this problem is equivalent to finding the minimum number of taxa that have to be added to N so that the resulting network N' is time-consistent. This means, in particular, that one can solve this problem by finding the minimum number of hybridization edges of N in which a new node has to be inserted such that the so modified network N' is time-consistent (cf. [42, Corollary 3.4]).

Practically, we tackle this problem in a quite naive way by simply inserting new nodes in N in respect to each subset containing exactly $k = 0, 1, 2, \dots$ hybridization edges and then by checking for each of those subsets if the so modified network is time-consistent. Notice

```
SELECT
all minimum hybridization networks
WHERE
['Taxon_A', 'Taxon_B', must be direct descendants of an hybridization event.]
AND
(
[Network minimizes time-consistency constraint.]
OR
[Network minimizes add-taxa constraint.]
)
```

Figure 5.2: An example of an SQL-statement that could be used for filtering a set of networks or to prune the underlying network space.

that one can speed up this approach by first identifying so-called *mandatory hybridization edges* that have to be modified anyway. More specifically, let v be a hybridization node of a minimum hybridization network and let V' be its set of immediate ancestors, then, in this context we say *a hybridization edge of v with source $v_1 \in V'$ is mandatory*, if there exists a different node $v_2 \in V'$ that is a descendant of v_1 and if there does not exist a path only consisting of hybridization nodes connecting v_1 and v_2. Notice that, as it is the case for the MTC-constraint, we suppose that the out-degree of each hybridization node in N is either 0 or at least 2, but not 1 (cf. Fig. 5.1).

So far, we have introduced five types of constraints, in which the first three types are rather of mathematical interest and the latter two types have more relevance from a biological point of view. Nevertheless, these constraints show that there already exist a couple of concepts that can help to control the number of networks one has to take under further examination. We are aware of the fact that there might exist some more appropriate constraints and, by presenting these very first prototypes, we hope to encourage biologists to develop their own individual constraints, which might get picked up by mathematicians as well as computer scientists in order to develop more practical network methods. Notice that Gambette provided a good overview[1] of other so far existing subclasses of explicit phylogenetic networks, which could potentially also be used for this purpose.

Once having built a set of network constraints, one can go one step further and use those constraints as building blocks to come up with even more sophisticated compound constraints. For example, those basic constraints could be combined by using an SQL-type language consisting of SELECT, WHERE, AND, and OR statements as, for instance, demonstrated in Figure 5.2. Notice that we have already integrated such an approach into our software package Hybroscale.

[1] `http://phylnet.univ-mlv.fr/isiphync/`

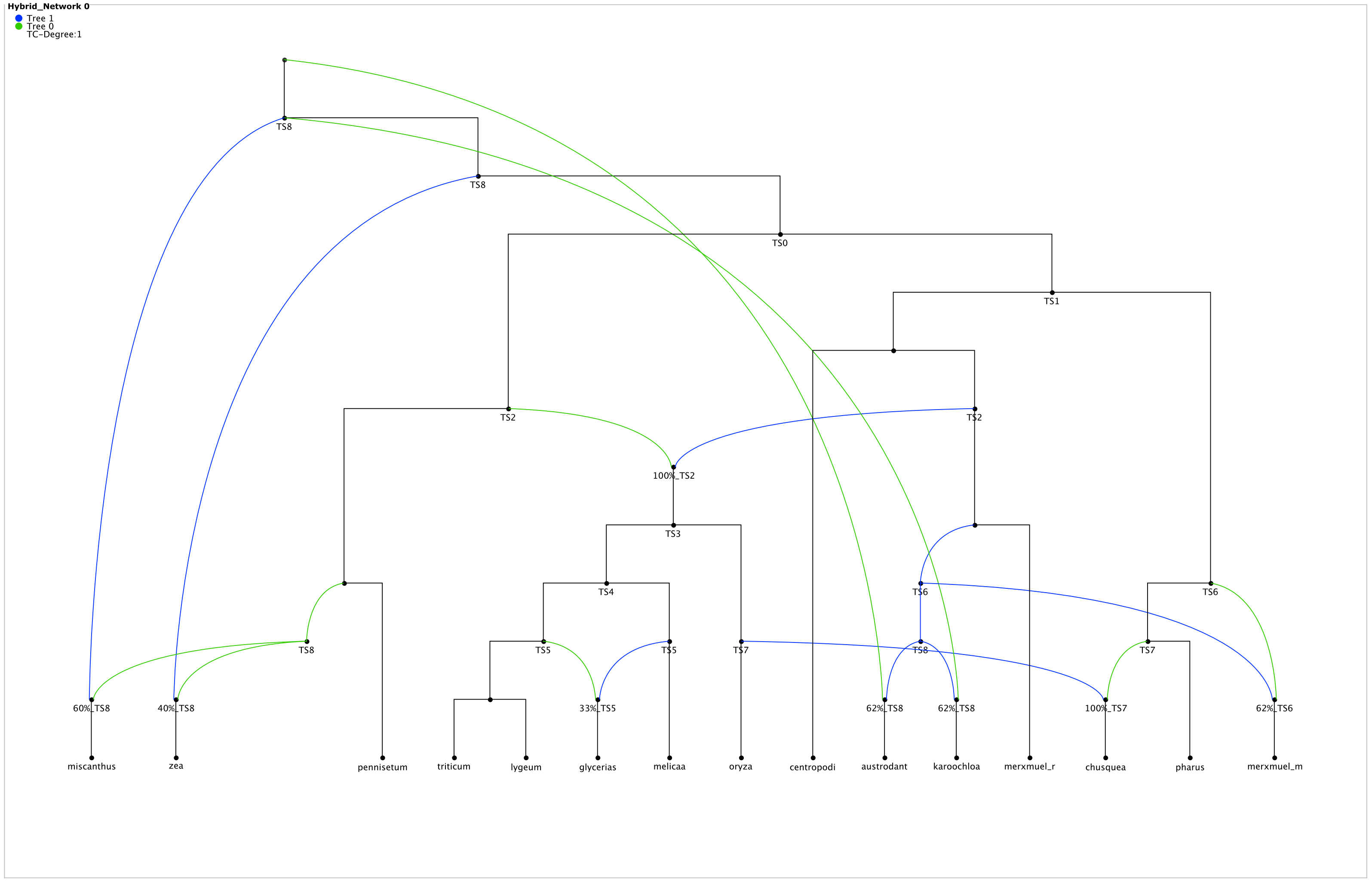


Figure 5.3: A network N satisfying each of the following *DH-constraints*: $\{$*miscanthus*$\}$, $\{$*zea*$\}$, $\{$*miscanthus*$\}$, $\{$*triticum, lygeum, melicaa, oryza*$\}$, $\{$*austrodant*$\}$, $\{$*karoochloa*$\}$, $\{$*chusquea*$\}$, and $\{$*merxmuel_m*$\}$. Moreover, the time-stamps, shortened by *TS*, that are assigned to each of its inner nodes show that $\tau(N)$ equals 1 since there is only one edge whose source node provides a larger time-stamp (*TS8*) than its target node (*TS0*).

5.1.3 Application to a grass data set

In this section, we demonstrate the practical benefit of some network constraints presented above. Our example is based on the well-known grass (*Poaceae*) dataset containing phylogenetic trees based on six different genetic loci, namely *ndhF*, *phyB*, *ropC2*, *waxy*, *rbcL*, and *ITS*. This freely available dataset[2], which is also used in the work of van Iersel *et al.* [66], was originally published by the Grass Phylogeny Working Group [27] and reanalyzed by Schmidt [55].

DH-constraint. Table 5.1 demonstrates how the set of networks calculated for the trees referring to the genes *ndhf* and *waxy* can be separated using the DH-constraint. For example, whereas taxon *merxmuel_m* is a direct result of a hybridization event in all of the 599 computed networks, for the taxa set {*meilicaa, triticum, eremitis, pariana*} this is only the case for 6 out of 599 networks. This means, in particular, if one would only be interested in networks satisfying the latter constraint, one could exclude 593 networks and, thus, for further examinations one would only have to take 6 instead of 599 networks into account.

In the following we give an example of how the number of hybridization networks can be filtered with the help of the MTC-constraint, the MAT-constraint as well as the ML-constraint. For this purpose, we integrated two different kinds of filtering techniques into our software package Hybroscale. The first technique can be used to filter a set of already calculated networks. This means, in particular, that we have implemented a method identifying among all reported networks those networks satisfying a particular constraint. The result is shown in Table 5.2.

Our second technique has a direct influence on the way of how the hybridization networks are computed and, thus, can significantly improve the practical runtime. This is mainly done by allowing a more aggressive bounding of computational paths of our underlying network method, which is based on the algorithm allMulHNetworks (cf. Sec. 4.4) that can be briefly summarized as follows. Given a set of input trees, each tree is added step-wise to a set of so far computed networks so that all resulting networks display all trees that have been added so far. At the beginning, this set of networks contains only one input tree. However, by adding further input trees, this set can grow rapidly as usually there exist many ways of how a tree can be added to a so far calculated network. Notice that, when running a search for networks with hybridization number k, one can immediately stop the processing of each network providing a hybridization larger than k, which is simply due to the fact that by adding further trees the hybridization number can only increase (but never decrease).

Now, regarding our network method, by using network constraints we can perform a more aggressive bounding of computational paths as follows. First, the so far minimum costs (depending on the respective constraint) of all so far calculated minimum hybridization networks displaying all input trees is stored in a global variable k'. Second, let N' be

[2]`www.sites.google.com/site/cassalgorithm/data-sets`

Table 5.1: An example showing numbers of minimum hybridization networks, calculated for two phylogenetic trees referring to the two sequences *ndhf* and *waxy*, fulfilling DH-constraints for different sets of taxa.

DH-Constraint	Frequency
{*merxmuel_m*}	599/599 (100%)
{*eremitis, pariana*}	461/599 (77%)
{*chusquea*}	378/599 (63%)
{*melicaa*}	343/599 (57%)
{*lygeum*}	306/599 (51%)
{*oryza*}	305/599 (51%)
{*glycerias*}	289/599 (48%)
{*triticum*}	260/599 (43%)
{*oryza, lygeum, trticum*}	132/599 (22%)
{*melicaa, glycerias*}	117/599 (20%)
{*lygeum, triticum*}	72/599 (12%)
{*oryza, eremitis, pariana*}	54/599 (9%)
{*trticum, glycerias*}	38/599 (6%)
{*melicaa, glycerias, eremitis, pariana*}	36/599 (6%)
{*oryza, melicaa, glycerias*}	36/599 (6%)
{*oryza, triticum, glycerias*}	28/599 (5%)
{*melicaa, triticum*}	21/599 (4%)
{*oryza, lygeum, glycerias*}	20/599 (3%)
{*lygeum, tritcum, eremitis, pariana*}	18/599 (3%)
{*lygeum, glycerias*}	17/599 (3%)
{*oryza, melicaa, lygeum*}	12/599 (2%)
{*tritcum, glycerias, eremitis, pariana*}	12/599 (2%)
{*oryza, melicaa, trticum, glycerias, eremitis, pariana*}	12/599 (2%)
{*melicaa, lygeum*}	10/599 (2%)
{*melicaa, lygeum, eremitis, pariana*}	6/599 (1%)
{*lygeum, glycerias, eremitis, pariana*}	6/599 (1%)
{*meilicaa, triticum, eremitis, pariana*}	6/599 (1%)

a network of a computational path (displaying a subset of all input trees) and let k be an upper bound of the hybridization number. Then, this path is aborted if the reticulation number of N' is larger than k or, additionally, if its costs exceeds k'. Notice, however, that the latter abort criterion can only be used for search spaces containing minimum hybridization networks. This means, in particular, that in our case the MTC-constraint (or the MAT-constraint) has only an effect on the search space in which k is equal to the hybridization number of the respective input trees.

Moreover, regarding the MTC-constraint as well as the MAT-constraint, by acting this way, some optimal solutions might get lost as by adding new reticulate edges the costs of a so far computed network can still be decreased. This can be the case, for example, if the source of a new reticulate edge is placed into an existing reticulate edge so that a cycle within the underlying digraph is dissolved. The results achieved by applying this second filtering technique based on the MTC-constraint as well as the MAT-constraint are summarized in Table 5.3.

ML-constraint. Table 5.2 shows that in general only a few biologically relevant networks satisfy the ML-constraint (denoted as ML-networks in this case).

MTC-constraint. Again, Table 5.2 indicates that through the MTC-constraint a large set consisting of computed networks can usually be reduced to a small subset. For example, this table reveals that only 2 out of 2079 networks, computed for the trees referring to the genes *ndhF* and *phyB*, satisfy the MTC-constraint.

Moreover, as indicated in Table 5.3, by setting this constraint *before* running Hybroscale, due to the so restricted search space, sometimes the practical runtime can be significantly improved. For example, the computation of all minimum hybridization networks for the trees referring to the genes *ndhF* and *its* takes approximately 15 minutes whereas by restricting the search space to minimum-time-consistent networks (MTC-networks) all 4400 networks are calculated within less than 4 minutes. Notice, however, that by comparing the results of Table 5.2 and Table 5.3 it turns out that by applying our second filtering technique not all MTC-networks can be calculated.

MAT-constraint. As indicated in Table 5.2 and 5.3, the same issues as discussed for the MTC-constraint also holds for the MAT-constraint. Again, the better practical runtimes result from a more aggressive bounding of computational paths. Notice, however, that, due to the same reasons as mentioned above, the computation of optimal solutions is not guaranteed in this case. Consequently, usually either not all MAT-networks are calculated or the reported networks do not provide minimum costs. For example, in our case, this applies to the genes *ndhF* and *rbcl*.

Moreover, Table 5.2 indicates that the costs for a minimum time-consistent labeling is a lower bound for the minimum number of taxa that have to be added to all input trees in order to receive time-consistent networks. Usually, this lower bound matches or is at least close to the minimum number of those edges. However, as indicated by the result calculated for the sequences *ndhf*, *rbcl*, *waxy*, and *its*, this lower bound can be also quite distant (2 vs. 5 in this case). This happens, for example, if there are many hybridization nodes that are connected through hybridization edges in which some of those edges are *mandatory*.

Table 5.2: Constraints applied to an output produced by running Hybroscale on phylogenetic trees belonging to a grass (*Poaceae*) dataset. A missing result for a certain tree set means that Hybroscale could not compute all minimum hybridization networks within 2 hours.

Genes	#Taxa	HNumber	#HNetworks	#MTC-Networks	#AT-Networks	#ML-Networks
ndhf its	46	17	554736	24500 (3)	28014 (7)	-
ndhf phyB	40	8	2079	2 (0)	2 (0)	33 (2)
ndhf rbcl	36	8	1488	18 (1)	48 (3)	288 (3)
ndhf rpoc	34	9	264	9 (3)	6 (3)	165 (4)
ndhf waxy	19	6	599	47 (2)	1 (2)	20 (3)
phyB its	30	8	195	25 (3)	1 (3)	105 (4)
phyB rbcl	21	4	6	2 (1)	2 (1)	6 (2)
phyB rpoc	21	4	9	1 (0)	1 (0)	9 (3)
phyB waxy	14	3	10	8 (2)	3 (2)	10 (2)
rbcl its	29	12	-	-	-	-
rbcl rpoc	26	7	111	23 (1)	19 (1)	91 (5)
rbcl waxy	12	4	84	26 (1)	4 (1)	3 (2)
rpoc its	31	12	3480	64 (3)	198 (5)	120 (9)
rpoc waxy	10	2	1	1 (1)	1 (1)	1 (2)
waxy its	15	5	15	5 (2)	5 (3)	3 (3)
ndhf phyB its	30	13	-	-	-	-
ndhf phyB rbcl	21	9	10889	190 (2)	14 (3)	11 (3)
ndhf phyB rpoc	21	8	36978	180 (1)	72 (1)	30 (3)
ndhf phyB waxy	14	4	42	22 (2)	4 (2)	26 (2)
ndhf rbcl its	28	-	-	-	-	-
ndhf rbcl rpoc	26	11	36600	3421 (2)	8 (2)	162 (6)
ndhf rbcl waxy	12	5	116	16 (1)	1 (1)	2 (3)
ndhf rpoc its	31	17	39016	660 (4)	-	-
ndhf rpoc waxy	10	3	14	9 (1)	4 (1)	1 (2)
ndhf waxy its	15	8	5466	658 (2)	27 (2)	8 (4)
phyB rbcl its	17	8	8661	224 (1)	16 (1)	12 (5)
phyB rbcl rpoc	15	6	40	12 (2)	6 (2)	9 (3)
phyB rbcl waxy	7	2	11	7 (0)	7 (0)	11 (2)
phyB rpoc its	19	7	57	2 (1)	4 (3)	57 (6)
phyB rpoc waxy	5	0	1	1 (0)	1 (0)	1 (0)
phyB waxy its	10	4	146	6 (1)	3 (1)	109 (2)
rbcl rpoc its	24	-	-	-	-	-
rbcl rpoc waxy	9	3	5	2 (1)	1 (1)	1 (2)
rbcl waxy its	11	6	63	10 (1)	12 (3)	1 (4)
rpoc waxy its	10	4	4	2 (1)	4 (3)	2 (3)
ndhf phyB rbcl its	17	-	-	-	-	-
ndhf phyB rbcl rpoc	15	9	1079	231 (2)	222 (3)	258 (5)
ndhf phyB rbcl waxy	7	2	1	1 (1)	1 (1)	1 (2)
ndhf phyB rpoc its	19	9	-	-	-	-
ndhf phyB rpoc waxy	5	0	1	1 (0)	1 (0)	1 (0)
ndhf phyB waxy its	10	5	4709	184 (1)	79 (1)	678 (2)
ndhf rbcl rpoc its	24	-	-	-	-	-
ndhf rbcl rpoc waxy	9	4	396	149 (1)	53 (1)	1 (2)
ndhf rbcl waxy its	11	6	2	2 (2)	2 (5)	2 (6)
ndhf rpoc waxy its	10	5	324	67 (1)	16 (1)	4 (3)
phyB rbcl rpoc its	14	-	-	-	-	-
phyB rbcl rpoc waxy	4	0	1	1 (0)	1 (0)	1 (0)
phyB rbcl waxy its	6	2	3	1 (0)	1 (0)	3 (2)
phyB rpoc waxy its	5	0	1	1 (0)	1 (0)	1 (0)
rbcl rpoc waxy its	9	5	335	87 (1)	2 (1)	2 (3)
ndhf phyB rbcl rpoc its	14	-	-	-	-	-
ndhf phyB rbcl rpoc waxy	4	0	1	1 (0)	1 (0)	1 (0)
ndhf phyB rbcl waxy its	6	3	135	18 (0)	18 (0)	38 (2)
ndhf phyB rpoc waxy its	5	0	1	1 (0)	1 (0)	1 (0)
ndhf rbcl rpoc waxy its	9	5	235	72 (1)	20 (1)	2 (3)
phyB rbcl rpoc waxy its	4	0	1	1 (0)	1 (0)	1 (0)
ndhf phyB rbcl rpoc waxy its	4	0	1	1 (0)	1 (0)	1 (0)

Table 5.3: Output produced by Hybroscale applied to phylogenetic trees belonging to a grass (*Poaceae*) dataset. Each runtime given in this table is stated in seconds. A missing result for a certain tree set means that Hybroscale could not compute the resp. set of minimum hybridization networks within 2 hours.

			HNetworks		MTC-Networks		MAT-Networks	
Genes	#Taxa	HNumber	#Nets	Runtime	#Nets	Runtime	#Nets	Runtime
ndhf its	46	17	554736	836.678	4400 (3)	203.483	11340 (7)	288.596
ndhf phyB	40	8	2079	7.606	2 (0)	3.278	2 (0)	3.288
ndhf rbcl	36	8	1488	8.26	18 (1)	4.867	168 (4)	10.604
ndhf rpoc	34	9	264	5.353	3 (3)	2.344	3 (3)	2.34
ndhf waxy	19	6	599	5.693	11 (2)	2.821	1 (2)	3.1
phyB its	30	8	195	8.304	9 (3)	3.168	1 (3)	2.808
phyB rbcl	21	4	6	1.65	2 (1)	1.639	2 (1)	1.637
phyB rpoc	21	4	9	1.678	1 (0)	1.649	1 (0)	1.646
phyB waxy	14	3	10	1.615	8 (2)	1.648	3 (2)	1.618
rbcl its	29	12	-	-	-	-	-	-
rbcl rpoc	26	7	111	3.836	18 (1)	3.019	15 (1)	3.064
rbcl waxy	12	4	84	4.338	26 (1)	5.641	4 (1)	4.259
rpoc its	31	12	3480	74.765	64 (3)	69.576	198 (5)	74.73
rpoc waxy	10	2	1	1.582	2 (1)	1.809	1 (1)	1.72
waxy its	15	5	15	2.712	5 (2)	3.858	5 (3)	6.271
ndhf phyB its	30	13	-	-	16947 (2)	1826.908	24 (2)	1245.95
ndhf phyB rbcl	21	9	10889	17.191	114 (2)	9.105	10 (3)	7.453
ndhf phyB rpoc	21	8	36978	54.516	108 (1)	36.006	66 (1)	35.872
ndhf phyB waxy	14	4	42	2.87	22 (2)	2.777	4 (2)	2.659
ndhf rbcl its	28	-	-	-	-	-	-	-
ndhf rbcl rpoc	26	11	36600	35.407	2091 (2)	22.254	1428,(3)	28.299
ndhf rbcl waxy	12	5	114	5.577	16 (1)	5.295	1 (1)	5.246
ndhf rpoc its	31	17	39016	4721.959	690 (4)	4364.465	28 (5)	2539.037
ndhf rpoc waxy	10	3	14	2.632	9 (1)	2.651	4 (1)	2.622
ndhf waxy its	15	8	5466	10.661	613 (2)	12.754	27 (2)	6.495
phyB rbcl its	17	8	8661	111.506	210 (1)	57.572	16 (1)	70.702
phyB rbcl rpoc	15	6	40	4.867	6 (2)	4.715	3 (2)	4.689
phyB rbcl waxy	7	2	11	2.592	7 (0)	2.592	7 (0)	2.597
phyB rpoc its	19	7	57	4.633	2 (1)	4.294	4 (3)	4.318
phyB rpoc waxy	5	0	1	0.075	1 (0)	0.074	1 (0)	0.073
phyB waxy its	10	4	146	3.844	6 (1)	3.2	3 (1)	3.207
rbcl rpoc its	24	-	-	-	-	-	-	-
rbcl rpoc waxy	9	3	5	1.62	2 (1)	3.205	1 (1)	2.667
rbcl waxy its	11	6	63	7.49	10 (1)	10.481	12 (3)	10.514
rpoc waxy its	10	4	4	2.635	2 (1)	2.683	4 (3)	2.893
ndhf phyB rbcl its	17	-	-	-	-	-	-	-
ndhf phyB rbcl rpoc	15	9	1079	259.138	125 (2)	224.468	125 (3)	233.487
ndhf phyB rbcl waxy	7	2	1	2.594	1 (1)	2.596	1 (1)	2.592
ndhf phyB rpoc its	19	9	-	-	10 (1)	1411.024	4 (2)	926.402
ndhf phyB rpoc waxy	5	0	1	0.071	1 (0)	0.084	1 (0)	0.086
ndhf phyB waxy its	10	5	4709	15.787	170 (1)	6.131	79 (1)	5.756
ndhf rbcl rpoc its	24	-	-	-	-	-	-	-
ndhf rbcl rpoc waxy	9	4	396	14.864	149 (1)	9.151	53 (1)	6.565
ndhf rbcl waxy its	11	6	2	138.462	2 (2)	90.395	2 (5)	105.409
ndhf rpoc waxy its	10	5	324	13.513	67 (1)	10.055	16 (1)	7.441
phyB rbcl rpoc its	14	8	-	-	42 (1)	1528.832	4 (3)	1972.775
phyB rbcl rpoc waxy	4	0	1	0.06	1 (0)	0.079	1 (0)	0.086
phyB rbcl waxy its	6	2	3	2.589	1 (0)	2.595	1 (0)	2.601
phyB rpoc waxy its	5	0	1	0.065	1 (0)	0.083	1 (0)	0.092
rbcl rpoc waxy its	9	5	335	21.438	87 (1)	12.244	2 (1)	9.21
ndhf phyB rbcl rpoc its	14	-	-	-	-	-	-	-
ndhf phyB rbcl rpoc waxy	4	0	1	0.084	1 (0)	0.076	1 (0)	0.089
ndhf phyB rbcl waxy its	6	3	135	18.246	18 (0)	8.331	18 (0)	8.029
ndhf phyB rpoc waxy its	5	0	1	0.083	1 (0)	0.101	1 (0)	0.105
ndhf rbcl rpoc waxy its	9	5	235	257.34	72 (1)	78.044	20 (1)	27.446
phyB rbcl rpoc waxy its	4	0	1	0.076	1 (0)	0.089	1 (0)	0.095
ndhf phyB rbcl rpoc waxy its	4	0	1	0.083	1 (0)	0.088	1 (0)	0.093

5.2 Rerooting trees by hybridization number

As already discussed during the introduction (cf. Sec. 1.1), when inferring phylogenetic trees the placement of the root is often a quite problematic step. Regarding the investigation of reticulation events based on such trees, a wrongly chosen location for its roots can produce misleading results. This is due to the fact that for different locations of those roots there exist different hybridization numbers and, consequently, different hybridization networks potentially outlining improper evolutionary scenarios.

In order to detect such wrongly chosen root locations, one can test whether other locations provoke smaller hybridization numbers. If this is the case and, especially, if these hybridization numbers are clearly smaller than the one calculated for the original chosen location, one possibly should review the entire respective anterior tree building process. We have integrated a quite naive method for rerooting trees in respect of hybridization numbers, which runs as follows. In a first step, all maximal common subtrees of the respective tree set are reduced. Then, each of those trees are rerooted in all possible ways by first removing the original root and then placing the root into one of its edges. Lastly, for all combinations of these rerooted trees, the respective hybridization number is calculated and, finally, the minimum of those numbers is reported. Notice that this approach is usually impractical as the computation of hybridization numbers, which is already a quite time consuming step, has to be applied several times. Consequently, so far our approach can only be applied to those trees providing low computational complexities, i.e., trees containing only few taxa as well as a small hybridization numbers.

Nevertheless, we could successfully apply this rerooting process to some gene trees of the *Poaceae* dataset. These results are given in Table 5.4.

Table 5.4: Results for rerooting the trees of a grass (*Poaceae*) dataset.

Genes	#Taxa	HNumber	HNumber (re-rooted)
ndhf phyB	40	8	7
ndhf rbcl	36	8	8
ndhf rpoc	34	9	9
ndhf waxy	19	6	5
phyB its	30	8	8
phyB rbcl	21	4	4
phyB rpoc	21	4	4
phyB waxy	14	3	3
rbcl rpoc	26	7	7
rbcl waxy	12	4	3
rpoc waxy	10	2	2
waxy its	15	5	5
ndhf phyB its	30	13	-
ndhf phyB rbcl	21	9	-
ndhf phyB rpoc	21	8	-
ndhf phyB waxy	14	4	3
ndhf rbcl its	28	-	-
ndhf rbcl rpoc	26	11	-
ndhf rbcl waxy	12	5	4
ndhf rpoc its	31	-	-
ndhf rpoc waxy	10	3	-
ndhf waxy its	15	8	-
phyB rbcl its	17	8	-
phyB rbcl rpoc	15	6	-
phyB rbcl waxy	7	2	2
phyB rpoc its	19	7	-
phyB rpoc waxy	5	0	0
phyB waxy its	10	4	4
rbcl rpoc its	24	-	-
rbcl rpoc waxy	9	3	3
rbcl waxy its	11	6	5
rpoc waxy its	10	4	4
ndhf phyB rbcl its	17	-	-
ndhf phyB rbcl rpoc	15	9	-
ndhf phyB rbcl waxy	7	2	-
ndhf phyB rpoc its	19	9	-
ndhf phyB rpoc waxy	5	0	0
ndhf phyB waxy its	10	5	-
ndhf rbcl rpoc its	24	-	-
ndhf rbcl rpoc waxy	9	4	-
ndhf rbcl waxy its	11	6	-
ndhf rpoc waxy its	10	5	-
phyB rbcl rpoc its	14	-	-
phyB rbcl rpoc waxy	4	0	0
phyB rbcl waxy its	6	2	2
phyB rpoc waxy its	5	0	0
rbcl rpoc waxy its	9	5	-
ndhf phyB rbcl rpoc its	14	-	-
ndhf phyB rbcl rpoc waxy	4	0	0
ndhf phyB rbcl waxy its	6	3	-
ndhf phyB rpoc waxy its	5	0	0
ndhf rbcl rpoc waxy its	9	5	-
phyB rbcl rpoc waxy its	4	0	0
ndhf phyB rbcl rpoc waxy its	4	0	0

5.3 Hybroscale

There exist a lot of software packages calculating specific types of rooted phylogenetic networks. Most of those methods, however, are just so-called *proof-of-concept* methods being not very user-friendly and, thus, are in general not used in practice in order to investigate reticulate evolution. Thus, we decided to develop our own tool — called Hybroscale — integrating the most important algorithms of this thesis.

Hybroscale [3] is a new Java-based software package that has been developed specifically for the investigation of minimum hybridization networks. It is based on a slightly extended version of the algorithm allMulHNetworks, previously presented in Section 4.4, that is able to deal with various kinds of input trees, i.e, a set of nonbinary phylogenetic trees sharing an overlapping set of taxa. Its graphical user interface is similar to Dendroscope [35], a well known software for visualizing phylogenetic trees and rooted networks. The software package Hybroscale is freely available from

`www.bio.ifi.lmu.de/softwareservices/hybroscale`

and runs on all major operating systems.

5.3.1 Distinctive features

There exist some programs for visualizing rooted trees or networks and a few programs for both computing and analyzing minimum hybridization networks embedding two rooted binary input trees sharing a common set of taxa (e.g., Dendroscope [35]). So far, however, there exists no program for both computing and visualizing minimum hybridization networks for an arbitrary number of rooted nonbinary input trees sharing an overlapping set of taxa. As Hybroscale can cope with this latter requirement, it is apparently of high interest for the research of reticulate evolution.

The program contains a command-line interface for running several applications via scripting as well as a graphical user interface (GUI) enabling an easy handling of each implemented function. Therefor, the main window of our GUI consists of several tab separated *Viewer*, in which each of them again consists of a data table and a grid window for selecting and visualizing trees and networks (cf. Fig. 5.4).

5.3.2 Details on the algorithm

Studying hybridization events based on a set of orthologous genes belonging to a particular set of species, can involve three major steps, namely the computation of well-supported phylogenetic trees for all respective genes, the inferring of minimum hybridization networks reconciling each of those gene trees, and the interpretation of the hybridization events being displayed by each of those networks. Hybroscale is specifically designed to perform the latter two of those three steps, which can be done by making use of the following subroutines.

Data import. Trees or networks, given in extended newick format [15], can be loaded from a file or simply pasted into a graphical window directly highlighting incorrect parts such that the user can easily fix those parts if necessary. Each inserted tree or network is then automatically added to a particular table on the left hand side.

Visualizing data. The size of the grid that is located to the right of the data tables can be divided into an arbitrary $n \times m$ matrix, in which each field contains a single panel so that different trees or networks can be visualized simultaneously next to each other. By selecting a panel of the grid followed by clicking on an entry contained in a data table, the corresponding tree or network is visualized by computing a special hierarchical layout. Such a layout follows the *Sugiyama approach* [61] and is specifically designed to guarantee a good readability. More precisely, Hybroscale runs an optimization step looking for a layout that, on the one hand, contains no crossings between tree edges and, on the other hand, a minimal number of crossings between reticulation edges. A detailed description of our layout algorithm is given in Section 5.3.3.

Moreover, in order to separate several analyses, based for example on different input trees, the user can create several distinctive *Viewers* each consisting of its own data tables and its own grid which ensures a clear separation between those analyses.

Computation of hybridization networks. Hybroscale contains a fast algorithm computing either the hybridization number or, even more sophisticated, all minimum hybridization networks for an arbitrary number of rooted nonbinary phylogenetic input trees sharing an overlapping set of taxa. Since usually there exist a large number of mathematical possible solutions, we added an option allowing to set specific constraints, as presented in Section 5.1.2, describing particular biological meaningful network features before running the algorithm. Moreover, the implementation of the algorithm is parallelized, which offers the user the opportunity to speed up its computation by running complex instances on distributed systems. After the computation, each hybridization network is automatically loaded into the network table of the corresponding *Viewer* so that it can be visualized easily as already described above. Moreover, the user can generate an extra window containing networks of special interest.

Analyzing hybridization networks. For analyzing hybridization networks it is important to visualize the embedding of each input tree. For this purpose Hybroscale can compute a special layout, where the user can select a color for each reticulation edge that is necessary for displaying one of the input trees. To avoid multiple colors for reticulation edges, each edge that is necessary for displaying k input trees is split into k single edges. Note that Hybroscale additionally offers its own file format storing the content of a *Viewer* by keeping track of the mapping of the input trees to those reticulate edges that are necessary for its embedding.

Furthermore, Hybroscale assigns each hybridization node a support value indicating the fraction of networks containing such a node and sorts all networks after the sum of these values in descending order. Roughly spoken, the computation of this value is done as follows. Given a hybridization node v of a set of networks, we first compute each subtree rooted at v restricted to one of the input trees and then just count in how many networks such a set consisting of all restricted subtrees occurs. A more detailed description of the computation of those support values is given in Section 3.5.4.

Exporting trees/networks. For subsequent work, the trees and networks of a *Viewer* can be easily exported in three different file formats, PDF, JPG, and PNG, or simply stored by its corresponding newick-string.

5.3.3 Drawing Rooted Phylogenetic Networks

Following the book "Phylogenetic Networks" [33], one can distinguish between two distinctive types of diagrams visualizing phylogenetic networks. Whereas in a *cladogram* only the topology of a network matters, in a *phylogram* each edge length provides additional information by reflecting a particular distance or a particular number of mutations. Moreover, a cladogram can be drawn either in a *triangular*, *rectangular*, or *circular manner*. For each of those different layout types, the book "Phylogenetic Networks" [33] presents different algorithms assigning coordinates to each node such that the resulting diagram satisfies constraints that are important to ensure an appropriate readability of the network. Note that the readability of a graph in general depends on the respective user. Nevertheless, there have been assessed particular aesthetic criteria for drawing abstract graphs [20], which obviously also play an important role in terms of phylogenetic networks. For instance, some of those criteria are the following.

- Minimize the number of bends.
- Minimize the number of edge crossings.
- Minimize the number of overlapping nodes and edges.
- Maximize the smallest angle between two edges incident to the same node.

Based on these criteria, we decided to implement our own method computing a rectangular diagram following the steps of the *Sugiyama algorithm* [61], which is a well-known approach for computing hierarchical layouts in terms of directed graphs. Therefore, the main goal of our algorithm is to compute a plain drawing of a network providing no edge crossings if possible and, otherwise, to calculate a layout containing a minimum number of crossings between all reticulation edges (and no crossings between tree edges). As this optimization problem is quite complex, we decided to use a heuristic approach, which tries to produce as little of those edge crossings as possible and, in return, provides a good runtime so that the layout of a network can be computed instantly.

In the following, we will briefly describe the four major steps that are conducted by our layout algorithm.

Step 1: Layer Assignment. First, an arbitrary embedded tree T of the network is chosen randomly. Second, each node of the network is assigned to a layer having a minimum distance to the root so that simultaneously each ancestor is located in an upper layer. In a third step, the nodes of each layer are sorted according to the ordering in which they occur during a depth-first traversal through T. Note that for this initial ordering of the layers, there do not exist any crossings between all edges in T (which are shortly denoted by E_T in the following).

Step 2: Crossing Minimization. In this step, the number of crossings between those edges that are not contained in E_T is minimized by changing the order of its target nodes within the corresponding layers in respect to its neighboring nodes. This step is done sequentially for all layers repeatedly iterating from the first to the last one and back again until no significant progress, in terms of the so far best ordering, is obtained. Finally, the best ordering for each layer, not providing any edge crossings between all edges in E_T and a minimum number of crossings between all edges not in E_T, is reported.

Step 3: Computation of Vertical Coordinates. The y-coordinates are assigned such that each node within a layer gets the same coordinate and the distance between two neighboring layers is identical by simultaneously maximizing the distance between the first and the last layer.

Step 4: Computation of Horizontal Coordinates. The x-coordinates are assigned such that, on the one hand, the number of bends is minimized and, on the other hand, all nodes keep a minimum horizontal distance.

Note that the first and the second step are repeated until, within a certain number of iterations, there is no significant progress in terms of the number of edge crossings within the best so far computed layout.

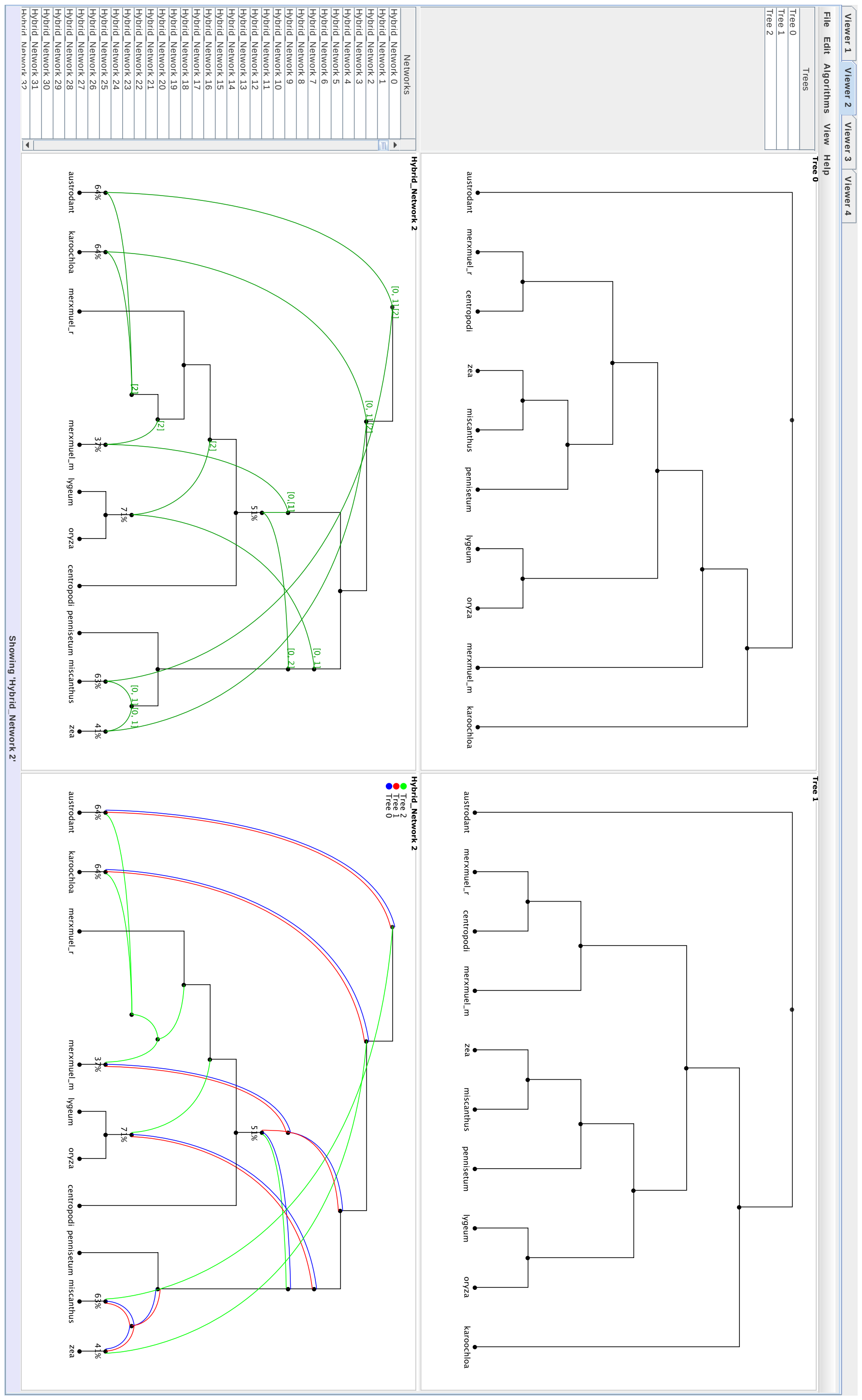


Figure 5.4: The main window of the program HYBROSCALE containing 4 different tab separated *Viewers*. The selected *Viewer*, which is denoted as VIEWER 2, consists of the following graphical elements: (left) The two tables showing its containing trees and networks. (right) The grid divided into 4 panels showing two trees and two networks — the right network in basic layout and the left network in a specific layout showing the embedding of each input tree by the three different colors blue, red, and green.

5.4 Conclusion

In this chapter, we have first presented some network constraints that can be used to speed up the execution of specific network algorithms as well as to filter resulting sets of hybridization networks. Two of those constraints are the MTC- and MAT-constraint dealing with time-consistency as well as the AddTaxa problem. By applying these constraints to a biological dataset, we could demonstrate the need for such constraints regarding the practical application of network building methods. Whereas the MTC-constraint selects all networks providing a minimum number of edges violating the time consistency constraint, the MAT-constraint selects all networks which have to be modified in a minimal biological way in order to establish time-consistency. We are aware of the fact that, from a biological as well as from a mathematical point of view, there might exist more appropriate network constraints. Thus, by presenting these very first prototypes, we hope to encourage biologists as well as mathematicians to work out their own individual constraints, which can then be used to further improve practical application of network building methods.

In a second part, we introduced a method rerooting trees by hybridization numbers. However, as this approach acts in a rather naive way, it is only applicable to input trees of rather low computational complexity. Nevertheless, it is the first approach enabling this functionality to more than two input trees as well as to nonbinary input trees.

Lastly, we have presented our software package Hybroscale that can be used for computing and studying a set of minimum hybridization networks displaying multiple rooted nonbinary phylogenetic trees sharing an overlapping set of taxa. As this program provides a graphical user interface, it guarantees an easy interactive handling including importing trees as well as representing and filtering reported networks. More specifically, the user can highlight the input trees that are embedded in each of those calculated networks and, additionally, by applying particular network constraints via an SQL-type language, the user can specifically select particular kinds of networks, which is an important feature for testing particular hypotheses. Consequently, Hybroscale is more than just a so-called *proof-of-concept* method ensuring the practical application of our developed algorithms and, thus, makes them available for further investigations of reticulate evolution.

Conclusion and outlook

In this thesis, we have presented several methods calculating particular types of rooted phylogenetic networks, namely so-called *minimum hybridization networks*, for a set consisting of rooted phylogenetic trees. As those networks explicitly indicate reticulation events, e.g., *horizontal gene transfer* and *hybridization*, these kind of networks can be directly used for studying reticulate evolution. So far most of our underlying algorithms are *novel* as, on the one hand, they can be applied to several input trees containing nonbinary nodes and, on the other hand, they guarantee the computation of all of those networks that are potentially relevant from a biological point of view. More precisely, until now there neither exist any methods aiming at calculating all minimum hybridization networks nor there exist any methods calculating minimum hybridization networks for more than two nonbinary trees.

Calculating all minimum hybridization networks is of great importance as, once having calculated all of these networks, one can then check which of those networks constitute plausible biological hypotheses. Moreover, since so far there only exist few methods focusing on calculating more than one possible solution, with our work we hope to attract interest in developing more sophisticated ways of describing specific topological network constraints. This means, in particular, that with this work we want to point out that networks are not free of interpretive challenges and it is now time to think about techniques facilitating this interpretation step (e.g., by using a specific SQL-type language).

As demonstrated here by several simulation studies, an approach computing minimum hybridization networks is often impractical for complex biological problems, i.e., several highly discordant trees providing a large number of taxa. Of course this limitation is problematic even though complex trees usually provoke complex networks that are hard to interpret (and, thus, are potentially meaningless). Nevertheless, a possible way of getting such complex data under control is again the application of network constraints. This is due to the fact that through well-defined constraints the space containing all potential solutions usually shrinks, which automatically facilitates the search after minimum hybridization networks and, thus, can improve the practical runtime of network building methods (especially of those providing *branch-and-bound* approaches).

Generally speaking, our methods for calculating minimum hybridization networks, can be distinguished in the properties the respective input has to fulfill. More specifically, we have developed methods for two binary input trees, for several binary input trees, as well as for several nonbinary input trees. For each of those methods, however, we could reuse the same approach, which means that we could extend our initial approach for two binary trees on the other more general inputs. This approach, beginning with one input tree, inserts the other trees step-wise by using the concept of *maximum acyclic agreement forests*. Notice that one reason why our methods working on more than two input trees are novel is that until now it has not been clear that maximum acyclic agreement forests can be used for that purpose. Moreover, as indicated in this thesis, by allowing trees providing multifurcating nodes further complicates the problem, since now additionally refinements of these nodes have to be taken into account requiring further sophisticated combinatorial subroutines.

Since through our approach networks are constructed by step-wise inserting all input trees, our methods are quite sensitive regarding the number of those trees. This means, in particular, that for increasing numbers of input trees our methods tend to become more and more impractical. Notice, however, that the computational complexity of an input does not only depend on its number of trees, but additionally on the size of its taxa set as well as on its underlying hybridization number. Nevertheless, there exist other methods for two input trees, e.g., the method TerminusEst [49], which seem to have the potential to be less sensitive in respect to the number of input trees. It still remains to show, however, that these methods can be extended in a way enabling the computation of all minimum hybridization networks for two and, especially, for more than two input trees.

Finally, we like to note that we have spent much time and effort in developing a user-friendly tool — called Hybroscale — enabling an easy application of our developed algorithms calculating minimum hybridization networks for a set of rooted (nonbinary) phylogenetic trees. As so far there do not exist any comparable software packages providing such a functionality, we are convinced that Hybroscale will make its contribution in broadening network methodology so that in near future more and more biologists will discover the advantages of modern network thinking.

Bibliography

[1] B. Albrecht. Computing hybridization networks for multiple rooted binary phylogenetic trees by maximum acyclic agreement forests. *arXiv:1408.3044*, 2014.

[2] B. Albrecht. Computing all hybridization networks for multiple binary phylogenetic input trees. *BMC Bioinformatics*, 16:236, 2015.

[3] B. Albrecht. Hybroscale: a software for studying reticulate evolution. *In preparation*, 2015a.

[4] B. Albrecht. Fast computation of all maximum acyclic agreement forests. *arXiv:1512.05656*, 2015b.

[5] B. Albrecht. Computing a relevant set of nonbinary maximum acyclic agreement forests. *arXiv:1512.05703*, 2015c.

[6] B. Albrecht, C. Scornavacca, A. Cenci, and D. H. Huson. Fast computation of minimum hybridization networks. *Bioinformatics*, 28(2):191–197, 2011.

[7] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L. Stougie, and J. Whitfield. Networks: expanding evolutionary thinking. *Trends in Genetics*, 29(8):439–441, 2015.

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

[9] M. Baroni, C. Semple, and M. Steel. Hybrids in real-runtime. *Systematic Biology*, 55(1):46–56, 2006.

[10] D. A. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers. Genbank. *Nucleic Acids Research*, 41(D1):D36–D42, 2013.

[11] M. Bordewich, S. Linz, K. John, and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics*, 7:86–98, 2007.

[12] M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8:409–423, 2005.

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

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

[15] G. Cardona, F. Rossello, and G. Valiente. Extended newick: It is time for a standard representation of phylogenetic networks. *BMC Bioinformatics*, 9:532, 2008.

[16] R. Chaudhary, M. S. Bansal, A. Wehe, D. Fernndez-Baca, and O. Eulenstein. iGTP: A software package for large-scale gene tree parsimony analysis. *BMC Bioinformatics*, 11:574, 2010.

[17] Z. Z. Chen and L. Wang. Algorithms for reticulate networks of multiple phylogenetic trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 26:2912–1913, 2012.

[18] Z. Z. Chen and L. Wang. HybridNet: a tool for constructing hybridization networks. *Bioinformatics*, 9(2):372–84, 2012.

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

[20] M. K. Coleman and D. S. Parker. Aesthetics-based graph layout for human consumption. *Softw. Pract. Exper.*, 26(12), 1996.

[21] The UniProt Consortium. UniProt: a hub for protein information. *Nucleic Acids Research*, 43(D1):D204–D212, 2015.

[22] C. Darwin. *The Origin of Species by Means of Natural Selection.* John Murray, London, 1st edition, 1859.

[23] J. H. Degnan and N. A. Rosenberg. Gene tree discordance, phylogenetic inference and the multispecies coalescent. *Trends in Ecology & Evolution*, 24(6):332–340, 2009.

[24] J. S. Escobar, C. Scornavacca, A. Cenci, C. Guilhaumon, S. Santoni, E. J. P. Douzery, V. Ranwez, S. Glmin, and J. David. Multigenic phylogeny and analysis of tree incongruences in Triticeae (Poaceae). *BMC Evolutionary Biology*, 11:181, 2011.

[25] J. Felsenstein. The number of evolutionary trees. *Systematic Biology*, 27(1):27–33, 1978.

[26] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[27] Grass Phylogeny Working Group. Phylogeny and subfamilial classification of the grasses (Poaceae). *Annals of the Missouri Botanical Garden*, 88(3):373–457, 2001.

[28] S. Guindon, J. F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: Assessing the performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, 2010.

[29] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003.

[30] B. M. Hallstrm and A. Janke. Mammalian evolution may not be strictly bifurcating. *Molecular Biology and Evolution*, 27(12):2804–2816, 2010.

[31] D. H. Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[32] D. H. Huson. Split networks and reticulate networks. In O. Gascuel and M. Steel, editors, *Reconstructing Evolution: New Mathematical Models for Evolution*, chapter 9, pages 247–276. Oxford University Press, 2007.

[33] D. H. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic networks: concepts, algorithm and applications.* Cambridge University Press, Shaftesbury Road, Cambridge, 2011.

[34] D. H. Huson and C. Scornavacca. A survey of combinatorial methods for phylogenetic networks. *Genome biology and evolution*, 3:23–35, 2011.

[35] D. H. Huson and C. Scornavacca. Dendroscope 3: An interactive tool for rooted phylogenetic trees and networks. *Systematic Biology*, 61:1061–1067, 2012.

[36] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academy Press, 1969.

[37] S. Kelk, S. Linz, and D. A. Morrison. Fighting network space: it is time for an SQL-type language to filter phylogenetic networks. *arXiv:1310.6844*, 2013.

[38] E. A. Kellogg, R. Appels, and R. J. Mason-Gamer. When genes tell different stories: The diploid genera of Triticeae (Gramineae). *Systematic Botany*, 21(3):321–347, 1996.

[39] Z. Li. Fixed-parameter algorithm for hybridization number of two multifurcating trees. Master's thesis, Dalhousie University, Halifax, Canada, 2014.

[40] S. Linz. *Reticulation in evolution.* PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät der Heinrich-Heine-Universität Düsseldorf, Germany, 2008.

[41] S. Linz and C. Semple. A cluster reduction for computing the subtree distance between phylogenies. *Annals of Combinatorics*, 15(3):465–484, 2011.

[42] S. Linz, C. Semple, and T. Stadler. Analyzing and reconstructing reticulation networks under timing constraints. *Journal of Mathematical Biology*, 61(5):715–737, 2010.

[43] M. C. J. Maiden. Horizontal genetic exchange, evolution, and spread of antibiotic resistance in bacteria. *Clinical Infectious Diseases*, 27(Supplement 1):S12–S20, 1998.

[44] J. Mallet. Hybridization as an invasion of the genome. *Trends in Ecology & Evolution*, 20(5):229–237, 2005.

[45] J. Mallet. Hybrid speciation. *Nature*, 446:279–283, 2007.

[46] D. A. Morrison. Networks in phylogenetic analysis: new tools for population biology. *International Journal for Parasitology*, 35(5):567–582, 2005.

[47] D. A. Morrison. Phylogenetic analyses of parasites in the new millennium. In J. R. Baker, R. Muller, and D. Rollinson, editors, *Advances in Parasitology*, volume 63 of *Advances in Parasitology*, pages 1–124. Academic Press, 2006.

[48] D. A. Morrison. *Introduction to Phylogenetic Networks*. RJR Production, 2011.

[49] T. Piovesan and S. Kelk. A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(1):18–25, 2013.

[50] D. Posada. jModelTest: phylogenetic model averaging. *Molecular Biology and Evolution*, 25(7):1253–1256, 2008.

[51] L. H. Rieseberg. Hybridization, introgression, and linkage evolution. *Plant Molecular Biology*, 42:205–224, 2000.

[52] F. Ronquist and J. P. Huelsenbeck. MrBayes 3: bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003.

[53] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[54] T. Sasanuma, K. Chabane, T. R. Endo, and J. Valkoun. Characterization of genetic variation in and phylogenetic relationships among diploid aegilops species by aflp: incongruity of chloroplast and nuclear data. *Theoretical and Applied Genetics*, 108(4):612–618, 2004.

[55] H. A. Schmidt. *Phylogenetic trees from large datasets*. PhD thesis, Mathematisch-Naturwissenschaftliche Fakultät der Heinrich-Heine-Universität Düsseldorf, Germany, 2003.

[56] K. Schwenk, N. Brede, and B. Streitl. Introduction. extent, processes and evolutionary impact of interspecific hybridization in animals. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 363(1505):2805–2811, 2008.

[57] C. Scornavacca, S. Linz, and B. Albrecht. A first step towards computing all hybridization networks for two rooted binary phylogenetic trees. *Journal of Computational Biology*, 19(11):1227–1242, 2010.

[58] R. R. Sokal, C. D. Michener, and University of Kansas. *A Statistical Method for Evaluating Systematic Relationships*, volume 28 of *University of Kansas science bulletin*. University of Kansas, 1958.

[59] P. Soltis and D. Soltis. The role of hybridization in plant speciation. *J. Comput. Biol.*, 60:561–588, 2009.

[60] A. Stamatakis. RAxML Version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[61] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(2):109–125, 1981.

[62] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis. Phylogeny reconstruction. *Molecular systematics*, 3:407–514, 1990.

[63] C. Than, D. Ruths, and L. Nakhleh. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinformatics*, 9:322, 2008.

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

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

[66] L. van Iersel, S. Kelk, R. Rupp, and D. H. Huson. Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters. *Bioinformatics*, 26(12):124–131, 2010.

[67] L. van Iersel and S. Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Inform. Process. Lett.*, 113(9):318–323, 2013.

[68] L. van van Iersel, S. Kelk, N. Lekić, and C. Scornavacca. A practical approximation algorithm for solving massive instances of hybridization number for binary and nonbinary trees. *BMC Bioinformatics*, 15:127, 2014.

[69] C. Whidden, R. Beiko, , and N. Zeh. Fixed-parameter and approximation algorithms for maximum agreement forests. *arXiv:1108.2664*, 2011.

[70] C. Whidden, R. Beiko, and N. Zeh. Fast FPT algorithms for computing rooted agreement forests: Theory and experiments. In P. Festa, editor, *Proceedings of the 9th International Conference on Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science (SEA 2010)*, pages 141–153. Springer-Verlag, Berlin, Heidelberg, 2010.

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

[72] Y. Wu. Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. *Bioinformatics*, 26(12):i140–i148, 2009.

[73] Y. Wu. An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. *J. Comput. Biol.*, 20(10):792–804, 2013.

List of abbreviations

AF	Agreement Forest
AG	Ancestor-descendant Graph
GUI	Graphical User Interface
HGT	Horizontal Gene Transfer
ILS	Incomplete Lineage Sorting
L1H	Level-1-Hybridization
MAF	Maximum Agreement Forest
MAAF	Maximum Acyclic Agreement Forest
MTC	Min-Time-Consistency
NJ	Neighbor Joining
PCR	Polymerase Chain Reaction
rSPR	rooted Subtree Prune and Regraft
SQL	Structured Query Language
UPGMA	Unweighted Pair Group Method with Arithmetic mean

Danksagung/Acknowledgments

Zunächst möchte ich mich bei *Volker Heun* und *Ralf Zimmer* bedanken, die mir die Möglichkeit gegeben haben meine in Tübingen begonnene wissenschaftliche Arbeit hier in München fortsetzen zu können. Dabei möchte ich mich besonders bei Volker Heun für die Unterstützung bei der Fertigstellung meiner Dissertation bedanken.

Next, i would like to thank my former supervisors of my diploma theses, *Daniel Huson* and *Celine Scornavacca*, for introducing me into the exciting field of phylogenetics. Moreover, i would like to thank two colleagues *Steven Kelk* and *Simone Linz*. Steven Kelk for showing so much interest in my program HYBROSCALE and for encouraging me to work out all the underlying theoretical work in a proper mathematical way. Simone Linz for answering a numerous number of emails, for proofreading some of my very first (and, thus, partially badly written) manuscripts, and for giving me a lot of useful advices.

Zusätzliche möchte ich mich noch bei meinen Kollegen hier am Institut für die abwechslungsreiche Zeit bedanken; sei es durch früh morgendliche Squashspiele oder durch zahlreiche mittäglichen Kerbenspiele.

Der größte Dank jedoch gebührt meiner Familie. Vor allem meinen Eltern die mich immer großartig unterstützt haben, auch wenn es vermutlich teilweise sehr schwer zu begreifen ist womit ich mich im Detail beschäftige, und vor allem natürlich meiner Freundin Sarah die mich auch immer großartig unterstützt hat und immer optimistisch geblieben ist auch wenn manchmal die Dinge nicht so liefen wie ich mir das im Vorfeld vorgestellt hatte. Ich freue mich auf eine gemeinsame spannende Zukunft und vor allem auf den nächsten Sommer der unser beider Leben wohl erstmal etwas auf den Kopf stellen wird.