Creating Phylogenetic Networks from Clusters

by

J. Keur

TU Delft Delft University of Technology

Creating Phylogenetic Networks from Clusters

by

J. Keur

A thesis submitted
in partial fulfilment of the requirements for the degree of

Master of Science

in

Applied Mathematics

for the specialization Discrete Mathematics and Optimization,
at Delft University of Technology,
to be defended publicly on Wednesday February 22, 2023 at 11:00 AM

Faculty EEMCS, Delft, The Netherlands

Student number:	4223292	
Project duration:	April, 2022 - February, 2023	
Thesis committee:	Dr. ir. L.J.J. van Iersel	TU Delft, supervisor
	Prof. dr. ir. C. Vuik	TU Delft

TU Delft Delft University of Technology

Contents

Abstract

In biology, phylogenetics is the study of the evolutionary history of and relations between e.g. species. Such data are often represented in trees. Remarkably, trees lack the representation of reticulation events, such as hybridization, while such events are believed to be important. One of the reasons why trees are widely used, is the enormous complexity of inferring a network directly from DNA data.

Therefore, indirect approaches are studied. One option is to construct a network from trees, which have been constructed from DNA data. Another option, which is studied in this work, is to create a network which represents all clusters from the trees. We are interested in finding a network having the lowest level possible, i.e. the lowest number of reticulations per biconnected component.

The Cass algorithm[12] is one of the best algorithms in this respect. However, it does not always give optimal results. In this thesis, an optimal algorithm is presented called STCass, of which many ideas are inspired from Cass. We lay theoretical foundations on how to build an optimal network. Furthermore, (greedy) steps are proposed to solve the problem, which seem to work very well in practice. Based on several optimizations that are proposed, an improved algorithm called MSTCass is presented, which runs faster than STCass, in general.

In order to speed up any Cass-based algorithm or derivatives of it, several lower bounds are presented on the reticulation number $r(N)$ per biconnected component N of an optimal network. Besides, a new upper bound on $r(N)$ is conjectured which can be evaluated in seconds, although e.g. an upper bound obtained by the HybridInterleave[3] or PIRN[26] algorithm is mostly lower and which can be evaluated often in twenty minutes[1] for the tested instances.

Index terms: Phylogenetic network, phylogenetic tree, level minimization, reticulation number, optimization, hybrid, hybridization event, level-k, cluster, compatibility, incompatibility graph, STCass, MSTCass.

Nomenclature

Abbreviations

LCA stands for the lowest common ancestor of multiple nodes in a network. ST-set stands for 'strict tree set'. MST-set means a maximal ST-set.

Symbols

Symbol	Meaning
$\mathcal{C}$	A set of clusters
$\mathcal{C}(N)$	The set of clusters represented by network N
$\mathcal{C}\|S$	This is equivalent to $\mathcal{C}\backslash(\mathcal{X}\backslash S)$, i.e. the set of clusters $\mathcal{C}$ restricted to the taxa $S \subset \mathcal{X}$
$Cl(\mathcal{T})$	The set of clusters which the trees $\mathcal{T}$ represent, i.e. $\bigcup_{T\in\mathcal{T}} \mathcal{C}(T)$
$IG(\mathcal{C})$	The incompatibility graph of the set of clusters $\mathcal{C}$
$\ell(N)$	The level of network N
$Nb(C)$	The set of neighbours of cluster $C \in \mathcal{C}$ in $IG(\mathcal{C})$
$LCA(C)$	The set of lowest common ancestors of cluster C
$minimal(\mathcal{C})$	The set of the minimal clusters in $\mathcal{C}$, i.e. $\{C \in \mathcal{C} : \nexists C' \in \mathcal{C}$ such that $C' \subset C\}$
N	A phylogenetic network
$r(G)$	Given a set of clusters $\mathcal{C}$ and some connected component G in $IG(\mathcal{C})$, $r(G)$ denotes the reticulation number of the corresponding biconnected component in a phylogenetic network that represents the clusters in $V(G)$
$r(N)$	The reticulation number of network N
T	A (phylogenetic) tree
$\mathcal{T}$	A set of trees
$\mathcal{X}$	A set of taxa
$\mathcal{X}(N)$	The set of taxa in network N

1

Introduction

In biology, phylogenetics is the study of the evolutionary history of and relations between e.g. species or taxa[7][21]. Such data are represented in phylogenetic trees and networks. Examples of phylogenetic trees and a network are shown in Fig. 1.1. The two trees in that figure are so-called *rooted phylogenetic trees*. The difference with *unrooted trees* is that rooted trees display the ancestry relationships between species, whereas unrooted trees only show the relations between them. In a rooted tree, the most recent common ancestor can be derived for several species, however, this is not possible in unrooted trees or networks. In particular, a rooted tree is a representation of the evolution of species by mutation and speciation. The leaves represent the currently living species and the root represents their most recent common ancestor. Both a rooted tree and network are directed acyclic graphs. Besides, each edge represents some cluster. I.e. in a tree, the cluster represented by an edge is the union of all leaf descendants of (the head of) that edge. In a network, such a cluster is only one of the clusters represented by an edge, since there could be more, due to reticulations (for details, see the next chapter).

In Fig. 1.1, the red node represents a reticulation event, which can represent either a hybridization, horizontal gene transfer or recombination (depending on the context). It is widely believed that reticulation events are important in phylogenetic models. Often, trees are used to represent evolutionary relationships, although reticulation events are hard to model in them. One of the main reasons is the enormous complexity of inferring a network directly from DNA data.

Therefore, indirect approaches are studied. One option is to construct a network from trees, which have been constructed from DNA data. In particular, an approach

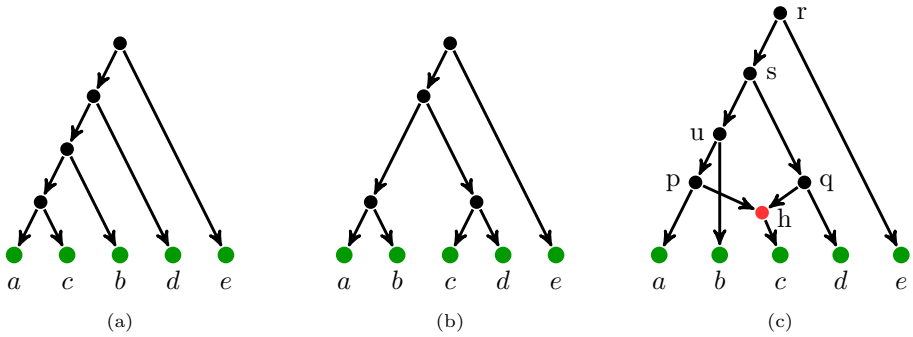


Figure 1.1: The subfigures (a) and (b) both represent a phylogenetic tree. The first one represents the clusters $\{a, c\}$, $\{a, b, c\}$ and $\{a, b, c, d\}$. The second tree represents the clusters $\{a, b\}$, $\{c, d\}$ and $\{a, b, c, d\}$. Subfigure (c) displays a simple phylogenetic network which represents all these clusters: Edge (s, u) represents the clusters $\{a, b\}$ (in the softwired sense) and $\{a, b, c\}$, edge (u, p) cluster $\{a, c\}$, edge (s, q) cluster $\{c, d\}$ and edge (r, s) represents cluster $\{a, b, c, d\}$. The red vertex h symbolizes a reticulation event.

(algorithm) is demanded to convert the trees to a phylogenetic network in which all trees are displayed. In order to create a network that is not more complex than necessary, one can aim at minimizing the number of reticulation events, or a related parameter called *level*. We call a phylogenetic network which minimizes the number of reticulation events or the level (depending on the context) an *optimal phylogenetic network*.

Many attempts have been made in the past to find networks that display the input trees, see e.g. [6], [5] and [21]. However, this is a hard problem; locating a tree in a given phylogenetic network is an NP-complete problem[14] and intuitively, finding an optimal network is much harder. For several restricted problems, locating a tree is possible in polynomial time[9]. Furthermore, a special type of phylogenetic network, a so-called tree-child network, can be created from an arbitrary number of binary trees using the FPT-algorithm proposed in [10]. However, this algorithm is only suitable for binary input trees, whereas we are interested in general input trees.

Nowadays, there is still a lack of a robust method to create such an optimal network from multiple non-binary trees. Therefore, instead of letting the target network display the input trees, we focus on letting it represent the clusters which are represented by the trees. The problem is then: give an optimal phylogenetic network which represents all clusters in the given trees. This is the main topic of this work.

A *cluster* is a proper subset of the taxa in a network; Given some edge $(u, v) \in E$ in any network $N = (V, E)$, the set of leaf descendants of v form a cluster. If for each reticulation in the network, exactly one incoming edge would be 'switched on' and the others 'switched off', then the leaf descendants reachable from v form a

cluster too. For example, by switching on edge (q, h) in Fig. 1.1c (and edge (p, h) off), cluster $\{a, b\}$ is represented by edge (s, u). Furthermore, if for each cluster C from the input clusters $\mathcal{C}$, network N contains and edge that represents C, then it is said that N *represents* $\mathcal{C}$.

For a more complex example, consider the set of four trees $\mathcal{T}$ in Fig. 1.2. You can check that the clusters $Cl(\mathcal{T})$ which are represented in these trees, except the singletons (and the cluster $\mathcal{X}(\mathcal{T})$), are 12, 13, 15, 34, 56, 58, 67, 134, 234, 567, 678, 1234, 1567, 1345, 5678, 12345 and 15678 (where e.g. 12 denotes cluster $\{1, 2\}$). Our goal is to find an optimal network which represents these clusters. Observe that the network presented in Fig. 1.3, is such one, comprising three reticulations. In order to see this, switch on exactly one incoming edge per reticulation and then list the clusters that are represented in the remained tree. Repeat this for any combination of edges that can be switched on and off. One example is illustrated in cyan for cluster 234 and in red for cluster 567.

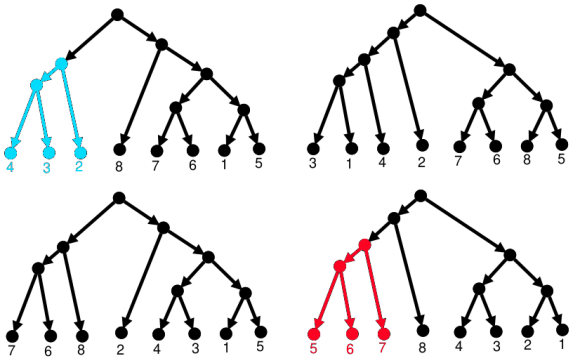


Figure 1.2: This is an edited copy of Fig. 8 in [15]. The coloured subtrees represent two examples of clusters.

Before specifying our interpretation of an optimal network, let us introduce some terms beforehand. A node (edge) is called a *cut-node* (*cut-edge*) if its removal would disconnect the graph. A *biconnected component* of a phylogenetic network is a maximal subgraph that cannot be disconnected by removing a single vertex. The *level* of a network is the maximum reticulation number per biconnected component. For example, say that some network contains three biconnected components, having reticulation number 3, 2, 1, respectively, then the level of the network is 3.

Now, some researchers regard an optimal network as one having the minimum number of reticulations. Such a network is called a *minimum reticulate network*. Lower and upper bounds on this number have been studied in e.g. [26].

However, we aim at minimizing the *level* of the network. Minimizing the number of reticulations is our secondary goal. The main reason for minimizing the level is inspired by phylogenetics, since if we would not do this, unrelated parts of the network may get interconnected. This is illustrated in e.g. Fig. 3 in [8], which is repeated below in Fig. 1.4.

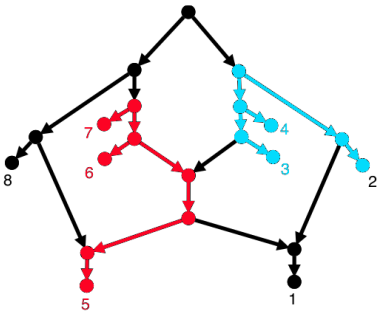


Figure 1.3: Fig. 9a from [15] is repeated here. This network is simple level-3 network representing the clusters $Cl(\mathcal{T})$ of the trees $\mathcal{T}$ which are shown in the previous figure. This network is optimal, as will be shown later. The coloured subgraphs represent different clusters, which correspond to those in the previous figure.

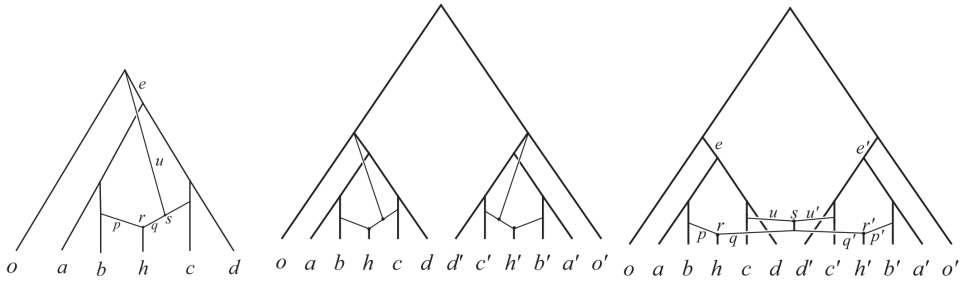
Regarding complexity of this problem, Kelk and Scornavacca showed that constructing a phylogenetic network from (softwired) clusters is fixed parameter tractable (FPT) in 2011[16]. In our case, this means that if we regard the level as fixed parameter and the degree of the polynomial does not depend on the level, the running time is polynomial in the number of taxa and clusters. However, the authors remarked that the running time of their algorithm "is too high to be of any practical interest". In particular, given that the level of an optimal network is k, the complexity of their algorithm is $f(k) \cdot poly(n)$, where "the $f(k)$ that is encountered (...) can be extremely exponential in k".

A related problem is the *hybridization number* problem, i.e. given multiple input trees, what is the smallest number of hybridization events that are required in a phylogenetic network which displays the input trees? This problem is NP-hard too. If the input trees are three binary trees, then the problem can be solved in time $O(c^k \cdot poly(n))$ (EPT)[11], where k is the level of the output network.

Note that if a phylogenetic network displays the input trees, then the network represents all clusters in the input trees. The converse however, is not (always) true. To show this, consider the network in Fig. 1.3 again. This network represents all clusters $Cl(\mathcal{T})$ from the four input trees $\mathcal{T}$ from Fig. 1.2, however, it does not display the first tree, nor the bottom left tree. As a consequence, the number of reticulations in an optimal network which represents the clusters, is a lower bound on the hybridization number. We do not study the hybridization number, however, it can be used as upper bound on the reticulation number in an optimal network.

1.1. Related Work

In this section, we address two questions. Firstly, what has been done before on the field of computing lower bounds on the *hybridization number*? Secondly, what



(a) A phylogenetic network (b) A minimum level network (c) A minimum reticulation network

Figure 1.4: (a) An optimal rooted network N that represents the clusters $\mathcal{C} = \{ab, abh, bh, cd, cdh, abcdh, abcd\}$, using two reticulations r and s. Note that the role of edge u is to switch off the taxon h so that the cluster $abcd$ can be represented by the tree edge e. (b) Two copies of N embedded into a minimum level (decomposable) network requiring four reticulations to represent all clusters in $\mathcal{C}$ and a second set $\mathcal{C}'$ of corresponding ones on $\{o', a', b', c', d', h'\}$. (c) This network represents the described clusters in (b) too, using only three reticulations. However, this is gained at an undesirable price: decomposability is abandoned (all clusters in $\mathcal{C}$ are compatible with those in $\mathcal{C}'$, yet they are all represented in the same biconnected component of the network) and so two completely unrelated parts of the phylogeny are linked together via reticulation edges.

algorithms have been developed in the past, that build a phylogenetic network N based on several input trees or clusters, such that N represents all clusters in the trees? In particular, to what extent is the level and/or the number of reticulations minimized?

Firstly, note that multiple works are devoted to computing lower bounds on the hybridization number. See for example [6], [18] and [22]. These bounds were settled with chromosomes in mind. Thereby, the order of the genes in the chromosomes matter. However, this does not matter when bounding the number of hybridizations. This is the reason why the bound of Hudson and Kaplan cannot be used to bound the number of hybridizations (see [5]). In the same paper, the authors propose several theorems (Theorems 3.2 and 3.1), which may be "applied to phylogenetic networks with hybridization as well." Thm. 3.1 gives a (trivial) lower bound on the number of hybridization events in a phylogenetic network. Thm. 3.2 is closely related to this. In 2009, HybridInterleave was introduced[3]. This algorithm computes the hybridization number for two binary trees. Another algorithm called PIRN[26] has the same objective as HybridInterleave. Its advantage over HybridInterleave is that it can accept more than two binary trees. On the other hand, HybridInterleave has the advantage that it is guaranteed to find an optimal solution.

Again, the hybridization number can be used as upper bound on the reticulation number in an optimal network. However, we wonder: what is the minimum reticulation number in a network which represents the *clusters* that are represented

in the input trees? Besides, what algorithms have been developed in the past, that build a phylogenetic network based on several input trees or clusters? In 2010, the algorithm called SplitsTree, was the most widely-used software for the construction of phylogenetic (split) networks, according to Huson et al.[7]. However, this algorithm was designed for inferring *unrooted* networks. The same holds for the algorithm called NeighborNet[2].

1.1.1. The Cass Algorithm and Derivatives

In 2010, the Cass algorithm[12] was introduced. This algorithm aims to create minimum level networks that represent a given set of clusters. When the input consists of exactly two binary trees T_1 and T_2, then Cass correctly minimizes level and even constructs a network of minimum level, which has been proven in [15].

In general, the Cass algorithm constructs networks with fewer reticulations than any other method. However, it has several drawbacks. Firstly, Cass is not suitable for constructing networks with a large number of reticulations, since its running time does not scale well with the level. Secondly, the constructed network highly depends on the order of the input data. Furthermore, the Cass algorithm does not always minimize the level, as shown in [15] too.

In regard of these issues, in 2013, an improved Cass algorithm was proposed by Wang *et al.*, called Lnetwork[25]. "Lnetwork is significantly faster than Cass and effectively weakens the influence of input data order. Moreover, Lnetwork can construct a much simpler network than most of the other available methods." However, it is not said that this algorithm is optimal, i.e. it does not necessarily return an optimal network.

BIMLR is another algorithm based on Cass. "BIMLR is faster than Cass and less dependent on the input data order. Moreover, BIMLR is able to construct much simpler networks than almost all other methods."[24]

The refrain in all testimonies of these Cass-based algorithms is that they are "able to construct much simpler networks than almost all other methods." Furthermore, these algorithms work for very general input trees, namely for multiple non-binary trees, which one gets in practice. These facts form a good motivation to use it as basis for our algorithm.

1.2. Other Applications

Phylogenetic networks are not only useful to represent genetic evolution over time and their dependencies. They could teach us *general principles* of how they evolved in the past and consequently, how they may evolve in the future. They are used in *linguistic phylogenetics* too, in order to illustrate the chronological development of and dependencies between languages. Secondly, phylogenetics are used in *forensics*, to asses evidence based on DNA in court cases. Thirdly, the theory is applied in the Linnaean *classification* of organisms. Similarly, it may help in diagnosing (categorising) diseases. Fourthly, phylogenetics is used to study the spread of pathogens (like the recent SARS-CoV-2 virus, see e.g. [17]), as well as their origin. Besides,

it can be used to detect which DNA protein may cause the "potential to increase transmissibility or virulence of the virus"[23]. On top of this, it can help in studying the similarities among different diseases and viruses. Fifthly, the study can help to form a conservation policy when decisions have to be made on which species should be especially protected from extinction. Sixthly, imagine that some manuscript has been copied by hand multiple times, and some copies have been copied again. Then phylogenetics can be used to determine a model for the dependencies between the manuscripts and their historical order in time. Seventhly, based on the assumption that "chemical abundances are inherited between generations of stars", phylogenetics is used "to trace the evolutionary history of our Galaxy"[13]. Let us leave it here. Note that the commonality in all these applications is that phylogenetics is used to study processes in nature.

1.3. Contribution

One of our main results is an algorithm called (M)STCass, which is guaranteed to find an optimal network for a given set of input clusters. This algorithm starts removing certain taxa one by one (or per set of taxa), until a tree can be built on the remained taxa, which represents the input clusters restricted those taxa. We call this phase of the algorithm the *removal phase*. Then, the taxa are inserted in reversed order in it, in a certain way. We call this second phase the *building phase* of (M)STCass. A set of taxa that is removed in the removal phase, is called an *ST-set*. An example of an ST-set is a pendant subtree. Further details on this are given in the next chapter, where some definitions and introductory terms are given as preparation and detailed introduction to the rest of this work. Besides, our problem and research questions are defined.

As stated earlier, the Cass algorithm does not always give an optimal solution. In Chapter 3, the Cass algorithm is analysed, (possible) solutions are proposed to make this algorithm optimal and one of these is handled in detail. Based on this, an optimal algorithm is presented, called STCass. In addition, Section 3.7 elaborates on the question if it is required to consider all ST-sets in the removal phase of STCass, or if considering only the maximal ST-sets would be sufficient. The last seems to be the case and based on this, an improved algorithm called MSTCass is presented (see Section 3.8 and see Chapter 7 for details on its implementation). This algorithm considers only maximal ST-sets (MST-sets), is optimal too, and much faster than STCass. The complexity of both algorithms is analysed in Section 3.9. The running time of MSTCass is polynomial for any fixed level k. Besides, some (heuristic) optimizations are given in Section 3.10 and Section 3.11.

Details on (M)STCass will be handled in Chapter 4. In particular, it will be discussed how to ensure that in any iteration in the network building phase of STCass, the resulting network N' will represent the required clusters (see Section 4.1 - Section 4.2). Section 4.3 addresses the question how to minimize the reticulation number $r(N')$ in network N', being the result of extending some network N (in the building phase of STCass) by inserting an ST-set. Furthermore, based on the regular occurrence of having to know the lowest common ancestors (LCAs) per

cluster, we show how to find the LCAs per cluster C, restricted to those which represent cluster C, in Section 4.4. The chapter ends with some notes on (M)STCass, since it appears that certain possibilities to build the network may be excluded from further exploration (see Section 4.5 - Section 4.6).

At this point, we switch the topic to finding bounds on the reticulation number $r(N)$ per biconnected component N in an optimal network. This is motivated by the fact that, if a lower bound on $r(N)$ per biconnected component N in an optimal network, is known a priori, this knowledge can be used to speed up any Cass-based algorithm. Therefore, we analyse substructures of the so-called incompatibility graph based on the input clusters. From this, we derive more than five (conjectured) lower bounds on $r(N)$ in Chapter 5. As a tool to do so, we introduce a certain type of network called a collnet (in Section 5.1).

After that, we study upper bounds on $r(N)$ in Chapter 6. A new upper bound is conjectured, which is easy to verify algorithmically. Due to its simplicity, its value is mostly higher than the hybrid number for two trees, however, it can give an upper bound on $r(N)$ when (clusters from) three or more trees are given, and often in less than a second.

In the third part of this work, we present some details of our implementation of the two proposed algorithms STCass and MSTCass in Chapter 7. In particular, Section 7.1 handles the techniques that have been used. In Section 7.2, the differences in the implementations of STCass and MSTCass are listed. For comparison and since the original implementation in Dendroscope does not work any more in the latest version of Dendroscope, Cass has been implemented too.

The results are presented in Chapter 8. In particular, the performance of both algorithms is addressed in Section 8.1, both in terms of the runtime and optimality, with optimality regarding the level and reticulation number of the resulting networks. All results are compared with those of Cass. In addition, the lower bounds on $r(N)$, which are determined by the formulas given in Chapter 5, are presented. Similarly, our new upper bound on $r(N)$ has been evaluated on the test data, whose results are shown.

Lastly, the results are discussed and conclusions are drawn in Chapter 9. Besides, some possibilities for improvements are summed.

2

Preliminaries

Let us consider some definitions as preparation for and introduction to the rest of this work. Firstly, we define what a phylogenetic network is and give some of its most important properties. Then, clusters are explained. After that, the corresponding incompatibility graph is introduced and defined. Lastly, the problem is stated for which a solution is proposed in this work.

2.1. Network Definition

Consider a set $\mathcal{X}$ of taxa. A *rooted phylogenetic network* thereon is defined as follows.

Definition 2.1. *A rooted phylogenetic network, on a set of taxa $\mathcal{X}$, is an acyclic digraph $N = (V, E)$, having the following properties.*

1. *It contains exactly one root, being a node in V having in-degree zero;*
2. *Each vertex $v \in V$ has either in-degree 1 or out-degree 1 (not both), except for the root;*
3. *The leaves (having out-degree 0) are bijectively labelled by $\mathcal{X}$.*

Each node therein is called either a root, split node, reticulation (node) or a leaf. The (unique) node having in-degree zero is the *root*. A node is called a *split node*

if its in-degree is 1. Each node having in-degree at least 2 and out-degree 1 is a *reticulation*. Finally, the other nodes have in-degree one and out-degree zero, which are called *leaves*. In general, nodes which are not leaves are called *internal nodes*.

2.2. Network Properties

Important properties of such a network, say network N, are the *reticulation number* $r(N)$ and the *level* $\ell(N)$. Note that the reticulation number $r(N)$ equals the total number of reticulations, if and only if the network is binary. For example, a network may contain one reticulation having in-degree three. Then, we say that the number of reticulations is one, while the reticulation number $r(N)$ is two (two binary reticulations would be needed if the network would be binary).

The reticulation number can be expressed as follows in Eq. 2.1. Note that $\delta^-(v)$ stands for the in-degree of node $v \in V$.

$$r(N) = \sum_{v \in V : \delta(v)^- > 0} \left(\delta^-(v) - 1\right) = |E| - |V| + 1 \tag{2.1}$$

A *biconnected component* of a rooted phylogenetic network N is a maximal subgraph that cannot be disconnected by removing a single node. An example of a network having multiple biconnected components, is illustrated in Fig. 2.1.

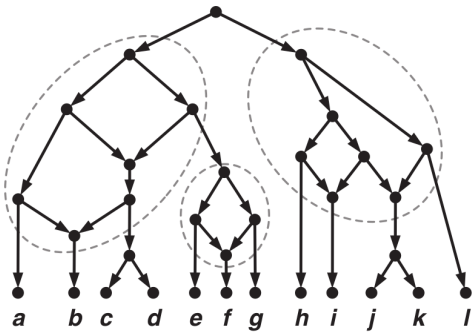


Figure 2.1: This figure is a copy of Fig. 1 in [12]. A phylogenetic network is shown, having exactly five reticulations. The encircled subgraphs are its three biconnected components. The level of this network is two, since each biconnected component contains at most two reticulations.

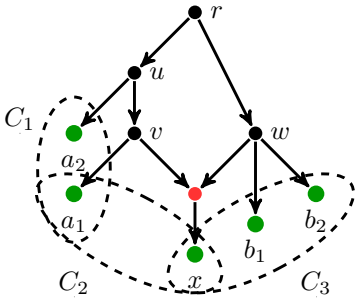
Observe that in a tree, there is no biconnected component. Furthermore, no leaf (taxon), nor the root is contained in any biconnected component. Now, a network N has *level* $\ell(N) = k$ if the reticulation number per biconnected component is at most k.

2.3. CLUSTERS

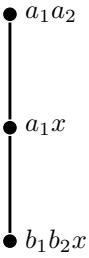
Regarding the clusters in a network, let (u, v) be any edge in network N. Then, the leaf descendants of node v form a *cluster*, being a proper subset of the taxa $\mathcal{X}$. A cluster C is said to be a *singleton* if $|C| = 1$. Moreover, it is said that the edge (u, v) *represents* cluster C. Any network N represents a cluster if it contains an edge which represents that cluster. If the removal of that edge would subdivide the network into two disconnected components, then the edge is called a *cut-edge*. Now, consider a set $\mathcal{C}$ of clusters on the taxa $\mathcal{X}$. A network is said to *represent* the clusters $\mathcal{C}$ if the network represents each cluster in $\mathcal{C}$. Note that in this work, we neglect singletons in $\mathcal{C}$, since they will always be represented in a network on $\mathcal{X}$.

A network can represent a cluster in the hardwired sense and in the softwired sense. A cluster is *represented in the hardwired sense* if the cluster is exactly the set of leaf descendants of some internal node. Given a network N, a *switching* T_N of N is obtained by preserving one incoming edge per reticulation and by removing the other incoming reticulation edges. A network N is said to *represent* some cluster C *in the softwired sense* if there is a switching T_N of N which represents cluster C in the hardwired sense. In this thesis, when it is written that N represents some cluster C, this is meant in the softwired sense.

Example 2.2. Below, the difference between hardwired and softwired clusters is illustrated. Consider Fig. 2.2a. The hardwired cluster represented by edge (r, w) is b_1b_2x. That represented by edge (u, v) is a_1x. The softwired clusters represented by edge (r, u) are a_1a_2 and a_1a_2x.



(a) An optimal network representing $\mathcal{C}$.



(b) The corresponding incompatibility graph $IG(\mathcal{C})$.

Figure 2.2: (a) An optimal network is shown that represents the clusters in $\mathcal{C} = \{a_1a_2, a_1x, b_1b_2x\}$, containing exactly one reticulation. (b) shows the corresponding incompatibility graph $IG(\mathcal{C})$.

Given a set of clusters $\mathcal{C}$ and a subset $S \subset \mathcal{X}$ of the taxa, $\mathcal{C}|S$ denotes the set of clusters restricted to these taxa S, i.e. $\mathcal{C}|S = \mathcal{C}\backslash(\mathcal{X}\backslash S)$. For example, consider Fig. 2.2a again. $\mathcal{C}|\{a_1, a_2, x\} = \{a_1a_2, a_1x\}$ there.

Given two clusters C_1 and C_2, we say that they are *compatible* if either $C_1 \cap C_2 = \emptyset$ or $C_1 \subseteq C_2$ or $C_2 \subseteq C_1$. Otherwise, they are *incompatible*. For an example, consider Fig. 2.2a again. Clusters C_1 and C_2 are incompatible, since they overlap in a_1 and both clusters have at least one taxon that is not contained in the other. Clusters C_1 and C_3 are compatible, since they are disjoint.

A set of taxa $S \subset \mathcal{X}$ is called *separated* (by $\mathcal{C}$) if there is a cluster $C \in \mathcal{C}$ that is incompatible with S. E.g. in Fig. 2.2a, if $S = b_1b_2x$, then cluster a_1x is incompatible with S and hence, S is separated.

A set of taxa $S \subset \mathcal{X}$ is an *ST-set* (strict tree set) w.r.t. $\mathcal{C}$, if S is not separated by $\mathcal{C}$ and any two clusters $C_1, C_2 \in \mathcal{C}|S$ are compatible. For example, if $S = b_1b_2$ in Fig. 2.2a, then S is not separated and it is an ST-set. Moreover, it is a maximal ST-set; an *ST-set* S is *maximal* (abbreviated as MST-set) if there is no larger ST-set T with $S \subsetneq T$.

The Cass algorithm (which is analysed in the next chapter) includes a collapse and decollapse step. Say that MST-set $S \subset \mathcal{X}$ is *collapsed*, then this means that the leaves S are replaced by e.g. label l and the clusters and leaves are updated accordingly. In the collapse step of Cass, all MST-sets are collapsed. We will see that this sometimes prevents Cass from producing an optimal network. The inverse process of collapsing is called *decollapsing* of such a meta-taxon l. Then firstly, a tree is built on the leaves S, representing the clusters in $\mathcal{C}|S$, and then its root replaces meta-taxon l. See Fig. 3.3a for an example, where meta-taxon 34 is decollapsed to a 'cherry' on the taxa 3 and 4, see Fig. 3.3d.

2.4. Incompatibility Graph

Given a set $\mathcal{C}$ of clusters, the *incompatibility graph* $IG(\mathcal{C})$ is the undirected graph (V, E) whose vertices are the clusters ($V = \mathcal{C}$) and edge (C_1, C_2) is present if and only if the clusters C_1 and C_2 are incompatible. For an example, see Fig. 2.2b.

In order to build a network that represents a set $\mathcal{C}$ of clusters, it is good to consider the incompatibility graph. In particular, it has been shown (see Theorem 1 in [12]), that there is always an optimal network, i.e. lowest level network (not necessarily having the lowest reticulation number), whose biconnected components correspond to the connected components of the incompatibility graph. This last part intuitively states what is formally described as a *decomposable network* with respect to $\mathcal{C}$, as described in [8] and which is built on [4]. The definition is repeated below.

Let ϵ be an *edge assignment*, being a mapping from each cluster $C \in \mathcal{C}$ to one of the edges $\epsilon(C)$ that represents it. A network N is *decomposable network with respect to* $\mathcal{C}$ (or simply: N is decomposable), if there exists an edge assignment ϵ such that for all pairs of clusters C_1 and C_2 in $\mathcal{C}$, the edges $\epsilon(C_1)$ and $\epsilon(C_2)$ lie in the same biconnected component of N if and only if C_1 and C_2 lie in the same connected component of the incompatibility graph $IG(\mathcal{C})$. For an example of a decomposable and non-decomposable network, see Fig. 1.4b and Fig. 1.4c in the previous section, respectively.

This decomposition property is a main ingredient for the Cass algorithm, which

we use as a basis to build our optimal algorithm. On top of this, there is a biologically inspired motivation, namely that using this approach, chances are low that totally unrelated parts of the network are related to each other (via a reticulation). Again, see e.g. Fig. 1.4 in the previous chapter for an example.

2.5. Problem Definition

Consider a set $\mathcal{X}$ of taxa and a set $\mathcal{C}$ of clusters on it. Our goal is to build a decomposable, rooted phylogenetic network N that represents all the clusters in $\mathcal{C}$.

The main question we address is: how to build a network N on the taxa $\mathcal{X}$, that represents the clusters $\mathcal{C}$, such that the reticulation number per biconnected component is as small as possible and thereby, the level $\ell(\mathcal{C})$ is minimized? Related to this, given the rules and guide lines which we have found as answer to this question, we wonder: are they all necessary, can they be restricted or can certain cases that are considered be excluded?

Let us subdivide these questions. Firstly, under what condition(s) is the reticulation number per biconnected component minimized? Secondly, how does this translate to the decisions that should be made (by an optimal algorithm) when building an optimal network? Thirdly, what lower bounds can be derived on the number of reticulations $r(N)$ per biconnected component N in a network that represents the input clusters? Fourthly, we address a question which might have more theoretical than practical interest: what upper bounds can be posed on $r(N)$? Lastly, how can an optimal algorithm be optimized, e.g. by using heuristic optimizations?

3

Transforming Cass to an Optimal Algorithm

In this chapter, we address the following questions: Why does the Cass algorithm not always give optimal results[15]? Secondly, how can we solve it? Thirdly, under what conditions will the reticulation number per biconnected component be minimized? Lastly, how to make an optimal algorithm faster?

3.1. The Problem of the Non-Optimality of Cass

As mentioned in the introduction, the Cass algorithm does not always give an optimal solution. "The problem with Cass seems to be that while the step of always collapsing at every iteration all maximal ST-sets and treating them as meta-taxa (in the sense of Corollary 11) is a locally optimal move; it can force us to use too many reticulation edges when hanging (trees corresponding to) maximal ST-sets below a reticulation in the outward phase."[15]

More concretely, this problem becomes visible when a meta-taxon is decollapsed, while there is a reticulation reachable from the tail of its cut-edge. Consider the following example.

Example 3.1. An example of a case in which Cass does not give an optimal solution is illustrated in Section 7.2 of [15]. The goal is to find an optimal network

representing the clusters in the four trees, which trees are repeated in Fig. 3.1 below. Fig. 3.2a shows an optimal solution of level 3. The Cass algorithm however, does not find it. Its best result is a non-optimal level-4 network and one of them is shown in Fig. 3.2b.

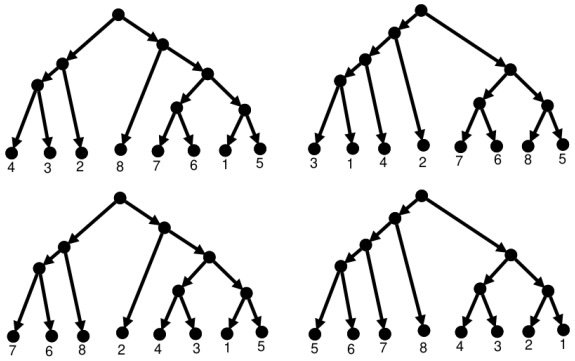
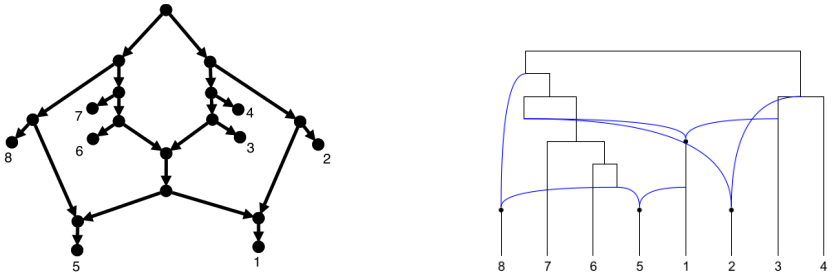


Figure 3.1: This is a copy of Fig. 8 in [15]. Let $\mathcal{T}$ be the set of four trees shown here. The Cass algorithm returns a network N that represents $Cl(\mathcal{T})$ where $r(N) = \ell(N) = 4$. However, the next figure shows that the true value of $r(Cl(\mathcal{T})) = \ell(Cl(\mathcal{T}))$ is at most 3.



(a) (b)

Figure 3.2: Fig. 9 in [15] is repeated here. (a) A simple level-3 network and (b) a simple level-4 network, both representing $Cl(\mathcal{T})$, where $\mathcal{T}$ is defined as described in the previous figure. The level-4 network was produced by Cass.

Now, we focus on one iteration of Cass in the building phase, in order to illustrate why Cass does not find an optimal solution and furthermore, how Cass can be improved such that it can find one. At some point in the outward building phase of the algorithm, a network may have been built comprising the induced subnetwork

as shown in Fig. 3.3a. Suppose that there is a meta-taxon, say M, having leaves $\mathcal{X}_M$ (3 and 4 in our example), hanging below a split node v, subdividing edge (u, t), where node t is a reticulation and furthermore, assume that the only leaf which is reachable from reticulation t, is leaf x (leaf 5 in the figure).

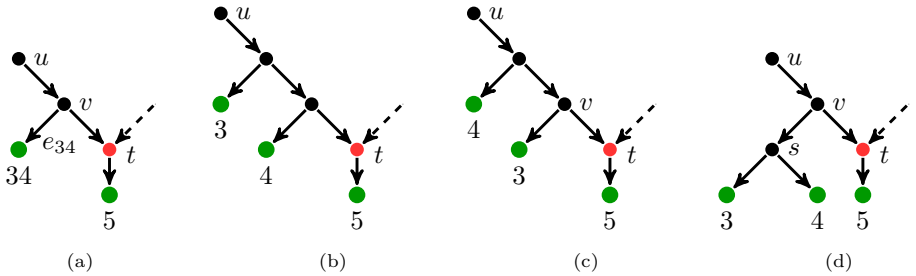


Figure 3.3: The left-most topology might be part of a network in some outer iteration of a Cass-based algorithm. Meta-taxon $M = 34$ is to be decollapsed to a subtree on the leaves 3 and 4 ($\mathcal{X}_M = \{3, 4\}$). Edge $e_M = e_{34}$ is its cut-edge. Cass decollapses the meta-taxon as illustrated in figure (d). However, when using a brute-force solving strategy like the Cass algorithm, one should decollapse the meta-taxon in all possible -three in this case- ways and investigate the resulting networks topologies. (These ways of decollapsing are described in the Decollapse method as presented in the next section.)

Assume that the meta-taxon M should be decollapsed now. Cass will only consider case (d) in Fig. 3.3, while actually there are -two in this case- more possibilities to decollapse this meta-taxon, as illustrated in the cases (b) and (c) in the same figure. Note that then, the definition of 'decollapsing' such a meta-taxon, as given in the preliminaries, should be changed. More details on this will be given in the next section.

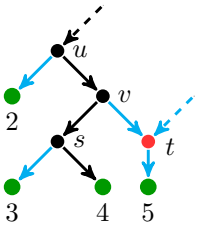
N.B. The same network topologies will be obtained if firstly, the meta-taxon M is inserted and decollapsed directly after it, and secondly, leaf x is hung back thereafter. Furthermore, note that in general, if some meta-taxon hangs below a split node, there are $(2|\mathcal{X}_M| - 1)$ ways of decollapsing it.

In order to illustrate why neglecting the cases (b) and (c) might lead to a non-optimal network, suppose that leaf $y = 1$ is to be inserted after decollapsing meta-node $M = 34$ as in Fig. 3.3d. Furthermore, assume that the clusters $\{1, 2\}$, $\{1, 3\}$

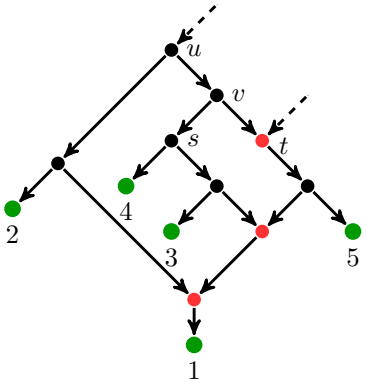
and $\{1, 5\}$ are to be represented, where leaf 2 is a child of node u, see Fig. 3.4a.

Cass would insert the leaf below two new binary reticulations, below the edges $(u, 2)$ and $(s, 3)$ (see Fig. 3.4a), and below one of the edges $(t, 5), (v, t)$ or the right inward arc of reticulation t or above it. (You may check this for yourself, or see the appendix of [15] for a proof.) This leads to a new network, e.g. as depicted in Fig. 3.4b.

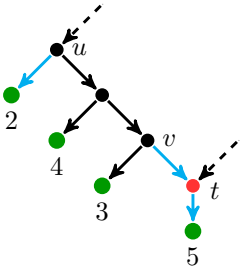
However, when we would decollapse meta-taxon $M = 34$ as illustrated in Fig. 3.4c instead of Fig. 3.4a, then it is sufficient to add only one binary reticulation and taxon $y = 1$ below it, namely below the edges $(u, 2)$ and one of (v, t) and $(t, 5)$ (see Fig. 3.4d). Again, this example illustrates that in a Cass-based algorithm, all $(2|\mathcal{X}_M| - 1)$ possible network topologies should be considered (when using a brute-force approach) after decollapsing some meta-taxon M.



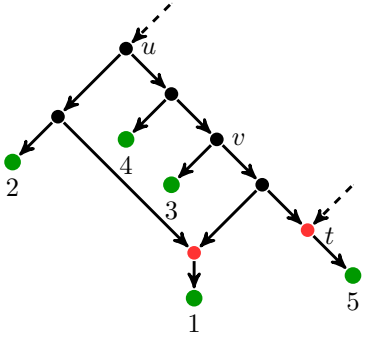
(a) In this way, Cass decollapses meta-taxon $M = 34$ to the leaves 3 and 4 after inserting leaf 5. The cyan coloured edges are candidate hang-edges for the taxon labelled '1'.



(b) Then, given that taxon '1' will be inserted and the clusters 12, 13 and 15 should be represented next, Cass will add two reticulations, e.g. as illustrated here.



(c) This way of decollapsing meta-taxon $M = 34$ should be considered (too). The cyan coloured edges are candidate hang-edges for the taxon labelled '1'.



(d) In case (c), after inserting taxon 1 below a reticulation below the edges $(u, 2)$ and (v, t), this figure shows a part of the optimal solution (see Fig. 3.2a).

Figure 3.4: Figure (a) represents the single way decollapsing meta-taxon 34 to the taxa 3 and 4 that Cass considers. When adding a taxon labelled '1', in order to let the network represent the clusters 12, 13 and 15, Cass will add two reticulations (see the appendix of [15] for a proof), e.g. as shown in figure (b). Figure (c) shows another way of decollapsing meta-taxon 34 that an optimal algorithm should consider (too). Based on this topology, figure (d) shows that adding only one reticulation and taxon 1 below it, is sufficient.

3.2. Solution 1: Redefine the Decollapsing Step

We propose two solutions for the problem stated in the previous section. The first one is as follows. Let de decollapsing operation in the Cass algorithm be redefined as presented below in Alg. 1.

Algorithm 1: Decollapse$(N, M, \mathcal{X}_M)$: Decollapse the meta-taxon M to a subtree on the set $\mathcal{X}_M$ of taxa. Return a list of networks comprising all possible ways of decollapsing the meta-taxon.

Input: (network N, meta-taxon label M, meta-taxon $\mathcal{X}_M \subset \mathcal{X}$)

Output: Decollapse$(N, M, \mathcal{X}_M)$

$p_M :=$ GetParent(N, M) // Get the parent of the meta-taxon

if p_M *is a reticulation* **then**

  decollapse: decollapse M in N as in the original Cass algorithm

  return $\{N\}$

/* p_M is a split node */

$\mathcal{N} := \emptyset$ // Initialize

$n :=$ GetNeighbour(N, M) // Get the neighbour of the meta-taxon

remove outgoing edges: in N, remove the two outgoing edges of p

decollapse: decollapse meta-taxon M in N, to a pendant subtree on $\mathcal{X}_M$

$E := E(subtree)$ // List the edges in the subtree

add the incoming edge of p_M to E

merge root: merge the root of the new subtree with node p_M in N

for *each edge* e *in* E **do**

  let N' be a copy of N

  reconnect neighbour: place a new node v on edge e and add edge (v, n) to N'

  add N' to $\mathcal{N}$

return $\mathcal{N}$

It is not obvious that this will always lead to an optimal network. In fact, this method implies that in each iteration in the inward phase of the Cass algorithm, a maximal ST-set (MST-set) is removed. Towards the end of this chapter, we will see

that this will always lead to an optimal solution (see Thm. 3.10).

3.3. Solution 2: Remove One ST-set per Iteration without Collapsing the Others

Another solution is the following. Instead of collapsing all MST-sets in each iteration in the inward ST-set removal phase of Cass and decollapsing them later, (collapse and) remove exactly one ST-set each time and try this for all ST-sets. Furthermore, when hanging (a tree corresponding to) an ST-set below a reticulation in some network N, instead of always considering only two hang-edges as Cass does, one should select the hang-edges based on the clusters $\mathcal{C}|\mathcal{X}(N')$ to be represented in the resulting network N'. Details on selecting suitable hang-edges will be handled in Chapter 4. This procedure will lead to an optimal algorithm, which will be shown in Section 3.6.

3.4. The Algorithm STCass

The lastly proposed solution will lead to an optimal algorithm, which we call STCass, see Alg. 2 below. STCass(k) will always return a k-reticulation network, if it exists (see Section 3.6). It is assumed that the set of input clusters $\mathcal{C}$ is such that $IG(\mathcal{C})$ comprises at most one connected component of size ≥ 2, i.e. $\mathcal{C}$ is unseparated.

In STCass, the function GetHangEdgeCombinations (see line 11) determines below which edges the inserted (reticulation and) leaf will be hung. This function may be regarded as the core of STCass. Requirements on it will be presented in the next section.

Lastly, note that in practice, there should be made a top-level algorithm which calls STCass(k) per connected component in $IG(\mathcal{C})$. For an example of such a top-level procedure, see Alg. 6 in App. A, and replace Cass(k) by STCass(k) on line 7 of it.

Algorithm 2: STCass$(\mathcal{C}, \mathcal{X}, k)$: construct a simple level-k network representing the clusters in $\mathcal{C}$ (inspired by the Cass algorithm, see Alg. 1 in [12] or Alg. 5 in App. A). In contrast with Cass, STCass will always return a level-k network which represents the clusters $\mathcal{C}$, if it exists. See the text for details.

Input: $(\mathcal{C}, \mathcal{X}, k, k')$

Output: STCass$(\mathcal{C}, \mathcal{X}, k, k')$

/* Initially, $k' = k$ */

if *there exists a tree representing the clusters in* $\mathcal{C}$ **then**
  return the unique tree representing the clusters in $\mathcal{C}$
if $k' = 0$ **then**
  return $\emptyset$
$\mathcal{N} := \emptyset$
$\mathcal{M} :=$ GetSTSets$(\mathcal{C})$ // Get all ST-sets in $\mathcal{C}$

```
7  for M ∈ 𝓜 do
8      (collapse and) remove the ST-set: 𝒞′ := 𝒞\M
9      recurse: 𝒩′ = STCass(𝒞′, 𝒳(𝒞′), k, k′ − 1)
10     for each network N′ in 𝒩′ do
11         ℰ = GetHangEdgeCombinations(𝒞, N′, k′, M)
12         for each combination of hang-edges E in ℰ do
13             if |E| = 1 then
14                 let N″ be a copy of N′
15                 add the ST-set below the edge: in N″, place a node v on
                    edge e ∈ E; create a leaf l labeled M and an edge from v to
                    l; finally, decollapse meta-taxon M to a pendant subtree on
                    the leaves 𝒳_M, such that l is its root
                   /* r(N″) ≤ k′ still holds, since no reticulation is added  */
16                 save the network: 𝒩 := 𝒩 ∪ {N″}
17             else
18                 let N″ be a copy of N′
19                 add the ST-set below a reticulation: create in N″
                    reticulation t, a leaf l labeled M and an edge from t to l;
                    then, for each edge e_i in E, insert in N″ a node v_i into edge
                    e_i and add an edge from v_i to t; finally, decollapse
                    meta-taxon M to a pendant subtree on the leaves 𝒳_M, such
                    that l is its root
20                 create all binary refinements of N″:
                    𝒩″ := GetBinaryRefinements(N″)
21                 for each binary network N‴ in 𝒩″ do
22                     if r(N‴) ≤ k′ then
23                         save the network: 𝒩 := 𝒩 ∪ {N‴}
24 return 𝒩
```

3.5. The Function GetHangEdgeCombinations

When STCass builds a network, it uses a function called GetHangEdgeCombinations (see line 11 in Alg. 2). In this section, we shortly address requirements on this function. Besides, a framework is proposed for it.

Requirement 1: The function GetHangEdgeCombinations should return a list of edge combinations in the given network N at that moment. Therein, each combination represents a set of edges below which leaf x will be hung (see the lines 12 - 15 in Alg. 2). We call the edges in such an edge combination *hang-edges*.

Requirement 2: For each edge combination E that is returned, the edges in E should be such, that the resulting network N' will represent all clusters in $\mathcal{C}|\mathcal{X}(N')$.

Requirement 3: For each edge combination E that is returned, $|E| \leq k' + 1$ should hold. Otherwise, N' would contain too many reticulations.

Requirement 4: In order to be sure that STCass returns an optimal level-k network, if it exists, GetHangEdgeCombinations should return *all* possible combinations of edges to hang x below, given that the other requirements 1 - 3 are satisfied. However, as we will see in Conj. 4.19, it might be sufficient to consider only a subset of them.

Note that from now on, we will frequently consider the *minimal clusters in* $\mathcal{C}$, which we denote as $minimal(\mathcal{C}) := \{C \in \mathcal{C} : \nexists C' \in \mathcal{C} \text{ such that } C' \subsetneq C\}$. Recall (as stated in Chapter 2) that we do not regard any singleton as cluster.

Example 3.2. Given the set of clusters $\mathcal{C} = \{12, 123, 234, 235\}$, the minimal clusters therein are $minimal(\mathcal{C}) = \{12, 234, 235\}$.

An example of the function GetHangEdgeCombinations is proposed in Alg. 3. For the background on selecting suitable hang-edges, see Chapter 4. The following lemma gives the worst-case runtime of the function.

Lemma 3.3. *Let $n = |\mathcal{X}|$ and $m = |\mathcal{C}|$. Let τ be as follows.*

$$\tau := \max_{x \in \mathcal{X}} \{|minimal(\mathcal{C}_x)|\} \tag{3.1}$$

(Observe that if the clusters in $\mathcal{C}$ stem from a set $\mathcal{T}$ of multiple trees, i.e. if $\mathcal{C} = Cl(\mathcal{T})$, then τ is at most $|\mathcal{T}|$.) Then, the time required by the function GetHangEdgeCombinations is as follows.

$$\mathcal{O}\left(m^2 n + \tau n^\tau\right) \tag{3.2}$$

Besides, the function returns $\mathcal{O}(n^\tau)$ edge combinations.

Proof. Let $n = |\mathcal{X}|$ and $m = |\mathcal{C}|$ and let τ be as required. Determining the set $\mathcal{C}_x$ costs time $\mathcal{O}(mn)$. Determining the minimal clusters in it, costs time $\mathcal{O}(m^2 n)$. The

cardinality of $\mathcal{C}_x$ and $\mathcal{C}_{\bar{x}}$ is at most τ. The function GetHangEdges (see the lines 6 and 8 in Alg. 3) needs time $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$ hang-edges are returned. Filtering the sets of cluster-hang-edges $E_C \in \mathcal{E}'$ costs time $\mathcal{O}(m^2 n)$. Next, listing all edge combinations $\mathcal{E}$ such that $E \cap \mathcal{E}'(C) \neq \emptyset$ for all $E \in \mathcal{E}$, requires time $\mathcal{O}(n^\tau)$ resulting in $\mathcal{O}(n^\tau)$ combinations. Lastly, filtering $\mathcal{E}$ costs time $\mathcal{O}(\tau n^\tau)$. Summing the time requirements gives the following total runtime t.

$$t := \mathcal{O}(mn + m^2 n + t(n + n^2) + \tau n^2 + m^2 n + n^\tau + \tau n^\tau) = \mathcal{O}\left(m^2 n + \tau n^\tau\right) \quad (3.3)$$

□

Algorithm 3: GetHangEdgeCombinations$(\mathcal{C}, N, k', x)$: Given a set of clusters $\mathcal{C}$ to be represented in the resulting network after hanging leaf (or ST-set) x in network N, return a list of edge combinations in N, where each combination is a set of edges below which leaf (or ST-set) x should be hung, satisfying the requirements which are mentioned in the text.

Input: (Cluster set $\mathcal{C}$, network N, maximum number k' of reticulations to add, leaf x to add)

Output: GetHangEdgeCombinations$(\mathcal{C}, N, k', x)$

/* Initialize */

$\mathcal{E}' := \emptyset$ // An indexed list of hang-edges per certain cluster

/* Ensure that all clusters in $\mathcal{C}_x$ will be represented in the resulting network N' after hanging x in N */

$\mathcal{C}_x := \{C \in \mathcal{C} : x \in C\}$

$\mathcal{C}_x := minimal(\mathcal{C}_x)$ // Use this to implement Conj. 4.6

for *each* $C' \in \mathcal{C}_x$ **do**

  $C := C' \backslash \{x\}$

  $\mathcal{E}'(C) :=$ GetHangEdges(C, N)

/* Ensure that the clusters in $\mathcal{C}_{\bar{x}}$ will be represented and by this, all clusters in $\mathcal{C}|\mathcal{X}(N')$ which do not contain x */

for *each* $C \in \mathcal{C}_{\bar{x}}$ **do**

  $\mathcal{E}'(C) :=$ GetHangEdges(C, N)

/* The following line implements Thm. 4.12 */

filter the cluster-hang-edges sets: $\mathcal{E}' :=$ GetMinimalSets$(\mathcal{E}')$

list all edge combinations: for each indexed cluster C in $\mathcal{E}'$, pick one edge from $\mathcal{E}'(C)$; do this for all combinations and store them in a list $\mathcal{E}$

/* Ensure that N' will represent all clusters that do not contain x */

filter the list: $\mathcal{E} :=$ filter each edge combination $E \in \mathcal{E}$ and preserve those and only those satisfying $|E| \leq k' + 1$

return $\mathcal{E}$

3.6. The Optimality of STCass

Why will STCass(k) return a level-k network for the input clusters, if it exists? In other words, why is STCass always able to return an optimal solution? This is answered below.

Theorem 3.4. *STCass(k) (Alg. 2) will always give a level-k network representing the given clusters, if it exists.*

Proof. Let N be an optimal level-k (thus a k-reticulation) network which represents some cluster set $\mathcal{C}$ on $\mathcal{X}$. Assume that N is binary, since each network can be decomposed into a binary network. By Lemma 7 in [15], by repeatedly removing one of the lowest reticulations in N and the pendant subtree below it on taxa $S_i \subset \mathcal{X}$ in the i-th iteration, until a tree is left, one will need exactly $r(N)$ iterations. Note that any S_i might be empty, except the first set. Furthermore, observe that each S_i is an ST-set (w.r.t. $\mathcal{C}$).

Let us apply this to STCass. Let S_i and one reticulation just above it be removed in the i-th iteration in the inward removal phase of the algorithm. Then, in the outward building phase of STCass, all these operations are reversed, resulting in optimal network N.

In detail, consider one iteration in the outward phase of STCass. Let N' be the network at that moment and suppose that some non-empty ST-set S_i is to be hung in N'. Note that S_i will be among the ST-sets $\mathcal{M}$ obtained on line 6 in Alg. 2. Network N' will be one of those that are obtained on line 9 and which are handled in the for-loop thereafter. Given that N'' should be the target network after placing S below a new reticulation in N', the required combination of hang-edges (i.e. edges to subdivide, below which a new reticulation is hung, see line 19 in Alg. 2) will be among those that are obtained by the function GetHangEdgeCombinations (on line 11), by requirement 3 (and 2) in the previous section. Since $r(N'') \leq k$, N'' will be used to hang the next ST-set in. Note that, if there are only empty ST-sets between two non-empty ST-sets S_i and S_j ($i < j$), then this implies that S_i will be hung below a $(j - i + 1)$-input reticulation in N'.

Since the above mentioned facts hold for any iteration in the outward (building)

phase of STCass, optimal network N will be obtained finally. The result follows.

□

Note that a sequence $\mathcal{S}$ of ST-sets as described in the proof above, is called a *(maximal) ST-set tree sequence* in other literature, like [15]. Furthermore, observe that such a sequence $\mathcal{S}$ can be found by letting ST-set $S_i \subset \mathcal{X}$ be the leaves in one of the lowest pendant subtrees and by removing this and the reticulation above it from network N. This should be repeated, until a tree is left. Meanwhile, add each ST-set S_i to $\mathcal{S}$.

3.7. On Considering ST-sets or MST-sets Only

In STCass, all ST-sets are considered. However, we will see that it is sufficient to consider MST-sets only (Thm. 3.10). For level-1 and level-2 networks, this has been shown to be true in [12].

Below, some network substructures are studied from which can be derived that the overall network is not optimal. A start is made by considering a level-1 network and it will be shown that, if the pendant subtree below its reticulation is not an MST-set, then this network is not optimal. Based on the observations in that proof, one can state that if STCass hangs a non-maximal ST-set S below a new reticulation t in the network that is built at some moment, then later, there should be placed an ST-set below at least one of the incoming edges of reticulation t, in order to obtain an optimal network.

After that, given a set of clusters $\mathcal{C}$, assume that STCass obtained an optimal network N which represents the clusters in $\mathcal{C}$, using some ST-set sequence $\mathcal{S}$. Additionally, suppose that S is the last ST-set which is hung back, after which only MST-sets are hung back, m in number. We conjecture that in this case, there is another optimal network $N' \neq N$ that represents the clusters $\mathcal{C}$, which is obtained via another sequence of ST-sets $\mathcal{S}'$, satisfying $|\mathcal{S}'| \leq |\mathcal{S}|$, in which the last $m+1$ sets are all MST-sets. By induction, this implies that for any set $\mathcal{C}$ of clusters, STCass can find an optimal network which represents $\mathcal{C}$ when considering MST-sets only. The last seems to be the case (see Thm. 3.10).

Lemma 3.5. *Let N be a (sub)network containing exactly 1 reticulation, in which the pendant subtree below the reticulation is not a maximal ST-set. Then this (sub)network is not optimal.*

Proof. Consider the fundamental structure of a 1-reticulation network N, as depicted in Fig. 3.5. Regard the leaves as ST-sets. Suppose that x is not a maximal ST-set. Let the MST-set which is a superset of x be denoted by S. Let the set of clusters to be represented in the (sub)network be denoted by $\mathcal{C}$. W.l.o.g., w.m.a. that a_1 (see the figure) is a subset of S. This implies that there is no cluster in

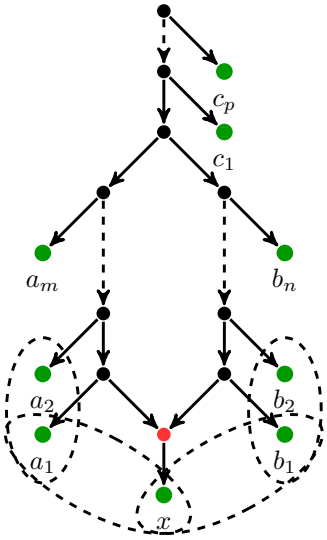


Figure 3.5: This is an illustration of the fundamental structure of a network that contains exactly one reticulation. The leaves may be considered as MST-sets. The dashed ellipses indicate the minimal clusters. Here, m and n are at least 1 and $p \geq 0$.

$\mathcal{C}$ which is a superset of both x and $b_1, \ldots, b_l$, for any $1 \leq l \leq n$, which is not a superset of any a_i, with $1 \leq i \leq m$, otherwise S would be separated (*).

Now, we claim that each cluster in $\mathcal{C}$ which is a superset of both x and a_1, is a superset of all the sets $a_i (i = 1, \ldots, m)$ (**). Assume that the reticulation is not redundant (otherwise, the network would not be optimal). This implies firstly, that there is a cluster C_1 in $\mathcal{C}$ which is a superset of both x and a_1. Secondly, w.m.a. that there is another cluster which is a superset of both a_1 and a_2, but not of x. Besides, there is a cluster which is a superset of a_m and hence, it is a superset of all sets $a_i, i = 1, \ldots, m$. If this cluster would not be a superset of x, then any cluster in $\mathcal{C}$ which is a superset of x, should contain all sets $a_i (i = 1, \ldots, m)$ too, otherwise it would be incompatible with cluster C_1. Hence, any cluster in $\mathcal{C}$ which is a strict superset of x, is a superset of all sets $a_i (i = 1, \ldots, m)$ too, which proves the claim.

From (*) and (**), one can conclude that there is a (sub)tree which represents all clusters in $\mathcal{C}$, as shown in Fig. 3.6. Hence, no reticulation is needed, so the original network N was not optimal. □

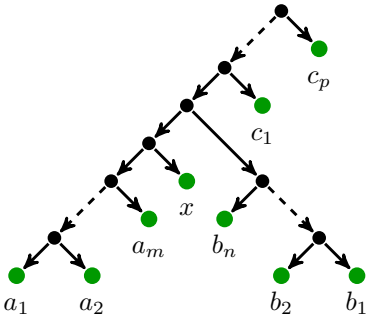


Figure 3.6: Given a (sub)network comprising exactly one reticulation (as shown in Fig. 3.5), in which the pendant subtree (on x) below the reticulation is not an MST-set, and given that the MST-set which is a superset of x, is a superset of a_1 too, the (sub)tree in this figure represents the clusters in $\mathcal{C}$.

Theorem 3.6. *Consider a level-k network. Let N be one of its biconnected components comprising k reticulations. Pick one of its lowest reticulations. Suppose that the pendant subtree below it is not an MST-set. Then subnetwork N is not optimal.*

Proof. Let N be as required and let t be one of the lowest reticulations. Let the pendant subtree below the reticulation be denoted as x. Then there is an induced subnetwork of N which can be represented as illustrated in Fig. 3.7.

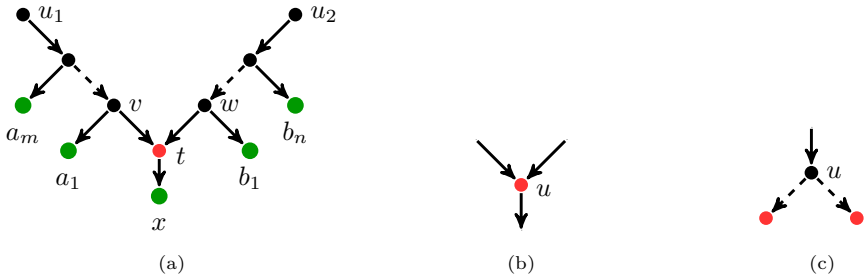


Figure 3.7: Given that t is one of the lowest reticulations in a network, having a pendant subtree below it, say ST-set x, there is a subnetwork which can be represented as shown in (a), with $0 \leq m, n$. The nodes u_1 and u_2 each represent either a reticulation (see (b) or a binary split node of which both outgoing edges have a reticulation among their descendants (see (c).

Assume that the pendant subtree x is not an MST-set. Then, there is an MST-set, say S, which is a strict superset of x. Consider the case that a_1 is a subset of

that MST-set S. Now, similar to the proof of Lemma 3.5, the statements (*) and (**) in that proof hold here too. Therefore, we conclude that S is a superset of x and all sets $a_1, \ldots, a_m$, reticulation t is redundant and furthermore, the network can be reduced to that in Fig. 3.8. Therefore, the original network N is not optimal.

N.B. If b_1 would be a subset of MST-set S, then x should be placed just above b_n instead of a_m in Fig. 3.8. □

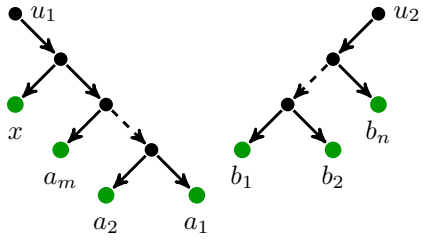


Figure 3.8: Consider Fig. 3.7. Given that pendant subtree x is not an MST-set and given that the MST-set which is a superset of x is a superset of a_1 too, we derive that reticulation t is redundant; the induced subnetwork as illustrated here, represents all clusters in $\mathcal{C}$ on the leaves that are shown.

Corollary 3.7. *In the first iteration of a* Cass*-based algorithm, an MST-set should be removed.*

Proof. Removing a non-maximal ST-set in the first inner iteration, implies that this set is hung in the network in the last outer iteration. This implies that the resulting network is not optimal, by Thm. 3.6. Hence, the firstly removed ST-set should be maximal. □

Corollary 3.8. *Consider* STCass *(Alg. 2). Suppose that in the outward building phase, each ST-set that is added, is added below at least one binary reticulation. Let t be the binary reticulation below which the last non-maximal ST-set is hung. If later, no MST-set is hung below a new reticulation, below at least one of the incoming edges of reticulation t, then the final network is not optimal.*

Proof. Let t be as required. Suppose that no MST-set is hung below at least one of the incoming edges of this reticulation. Then the final network consists of either a subnetwork as illustrated in Fig. 3.7, or it has an induced subnetwork as depicted

in Fig. 3.9. In the first case, analogous to the proof of Thm. 3.6, we conclude that the final network is not optimal.

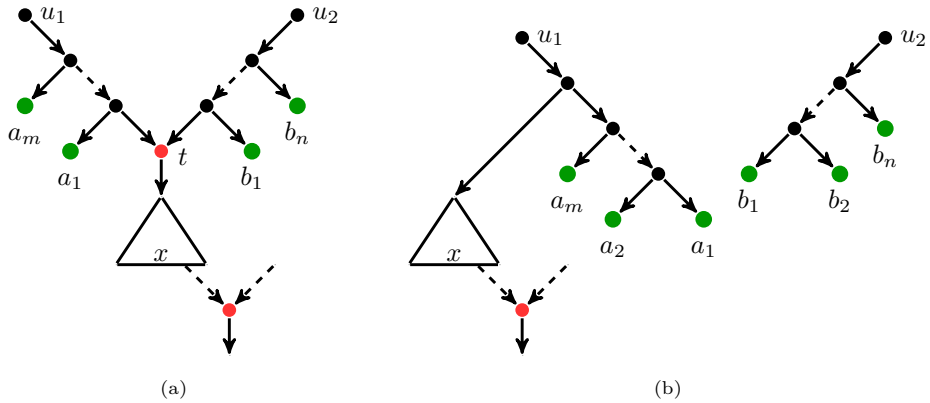


Figure 3.9: Given that t is the last reticulation below which a non-maximal ST-set is hung, say ST-set x, there is a subnetwork which can be represented as shown in Fig. 3.7 or as shown here in subfigure (a), in which one or more edges in the tree on x are subdivided and go to a reticulation. The nodes u_1 and u_2 each represent either a reticulation (see Fig. 3.7b) or a split node of which both outgoing edges have a reticulation among their descendants (see Fig. 3.7c).

In the second case, where one or multiple reticulations have been hung below the subtree on x, the network can be transformed from a situation as illustrated in Fig. 3.9a to one like in Fig. 3.9b for the same reasons (*) and (**) as given in the proof of Lemma 3.5. Hence, reticulation t is redundant. □

Conjecture 3.9. *Consider STCass (Alg. 2). Assume that in the outward building phase, each ST-set that is added, is added below at least one binary reticulation. Let t be the binary reticulation below which the last non-maximal ST-set is hung. Suppose that from then on, m MST-sets are hung back, such that the final network N is optimal. Let $\mathcal{S}$ be the sequence of ST-sets that have been removed in the inward phase of the algorithm (in which the first m are thus MST-sets and the $(m+1)-th$ set is a non-maximal ST-set). Then there is a network $N' \neq N$ which can be found by STCass using another sequence $\mathcal{S}'$ in which the first $m+1$ sets are all MST-sets now and furthermore, $|\mathcal{S}'| \leq |\mathcal{S}|$.*

Given some set of clusters $\mathcal{C}$, by induction, Conj. 3.9 implies that there exists an

MST-set sequence to be removed/added by STCass, such that an optimal network representing $\mathcal{C}$ will be obtained. The last is the case.

Theorem 3.10. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Then, in the inward phase of STCass, it is sufficient to consider MST-sets only, in order to obtain an optimal network which represents the clusters in $\mathcal{C}$.*

Proof. Consider Lemma 9 from [15], which is repeated here: "Given a set of clusters $\mathcal{C}$ on $\mathcal{X}$, there exists an MST-set tree sequence $(S_1, S_2, \ldots, S_p)$ such that $p \leq r(\mathcal{C})$." Such a sequence can be found in a similar way as shown in the proof of Thm. 3.4. □

3.8. The Algorithm MSTCass

Thm. 3.10 implies that in STCass, the function GetSTSets may be replaced by GetMSTSets, in order to obtain the MST-sets only. We call the resulting algorithm MSTCass. Furthermore, the theorem implies that this algorithm is optimal too.

Corollary 3.11. *MSTCass will always find a network having the lowest level possible for a given set of input clusters.*

Proof. This directly follows from our definition of MSTCass and Thm. 3.10. □

An advantage of MSTCass is that it is much more efficient than STCass. In particular, it has a polynomial runtime for a fixed level k, as we will see in the next section.

3.9. The Time Complexity of STCass and MSTCass

Let us address (upper bounds on) the time complexity of the proposed algorithms STCass and MSTCass and compare it with that of Cass. Firstly, consider the time complexity of STCass.

Lemma 3.12. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$, such that $r(\mathcal{C}) \geq 1$. Then, there are at most $2^{|\mathcal{X}|-2}+2$ different ST-sets. Furthermore, these can be obtained in time $\mathcal{O}(2^{|\mathcal{X}|-2})$.*

Proof. Let $\mathcal{C}$ be a set of m clusters on n taxa $\mathcal{X}$. The set $\mathcal{M}$ of MST-sets can be obtained in time $\mathcal{O}(n^3 m)$. Its cardinality is at most n, by Cor. 4 in [15].

Let M be an MST-set from $\mathcal{M}$ of cardinality at least two. Let T_M be the tree on M representing the clusters $\mathcal{C}|M$ (which may be non-binary). Given any split node v in $V(T_M)$, let $\delta^+(v)$ be its out-degree. Now, suppose that this node would be split into two split nodes v_1 and v_2, having edge (v_1, v_2) in between, such that $2 \leq i \leq \delta^+(v) - 1$ of the out-going edges of v are those of v_2 now, and the other out-going edges of v are those of v_1 now. Then, the hardwired cluster below node v_2 is an ST-set. Note that the hardwired cluster below v_1 is an ST-set too. In this way, $\sum_{i=2}^{\delta^+(v)} \binom{\delta^+(v)}{i} = 2^{\delta^+(v)} - \delta^+(v) - 1$ ST-sets can be defined per split node v in T_M. Hence, the number c_M of non-trivial ST-sets in T_M, including M itself, is as follows.

$$c_M = \sum_{v \in V(T_M): \delta^+(v) \geq 2} \left(2^{\delta^+(v)} - \delta^+(v) - 1\right) \tag{3.4}$$

c_M is maximized if there is only one split node v in T_M. It follows that $c_M \leq 2^{|M|} - |M| - 1 \leq 2^{|M|}$. Since this holds for every MST-set M, we get a total number c of ST-sets as given below.

$$c := \sum_{M \in \mathcal{M}} c_M \leq \sum_{M \in \mathcal{M}} 2^{|M|} \overset{(*)}{\leq} 2^{n-2} + 2 \tag{3.5}$$

(*) Note that $|\mathcal{M}| \geq 3$, since $r(\mathcal{C}) \geq 1$. Furthermore, c is maximal if as many MST-sets as possible are singletons and only one comprises the other $n - |\mathcal{M}|$ taxa. Finally, all ST-sets can be obtained in time $\mathcal{O}(2^{n-2})$. □

Theorem 3.13. *Let $\mathcal{C}$ be a set of m clusters on n taxa $\mathcal{X}$, such that $r(\mathcal{C}) \geq 1$. Let τ be as follows.*

$$\tau := \max_{x \in \mathcal{X}} \{|minimal(\mathcal{C}_x)|\} \tag{3.6}$$

Then, STCass takes time

$$\mathcal{O}\left(2^{(\tau+1)(k-1)(n-2)} \left[m^2 n + n^{2\tau} \left(mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.7}$$

which is upper bounded by $\mathcal{O}\left(2^{(k+2)(k-1)(n-2)} \left[m^2 n + n^{2(k+1)} \left(mn + (n/2)^k\right)\right]\right)$. Besides, the number of constructed networks is of order $\mathcal{O}\left(2^{n-\tau-1} \cdot n^{k(\tau+1)-2}\right)$, which is upper bounded by $\mathcal{O}\left(2^{n-2} \cdot n^{k'(k+2)-2}\right)$.

Proof. Let $n = |\mathcal{X}|$ and $m = |\mathcal{C}|$. By induction on k', we will show that STCass$(\mathcal{C}, \mathcal{X}, k, k')$ takes time $\mathcal{O}\left(2^{(\tau+1)(k'-1)(n-2)}\left[m^2n + n^{2\tau}\left(mn + (n/2)^{\tau-1}\right)\right]\right)$ and returns at most $\mathcal{O}\left(2^{n-\tau-1} \cdot n^{k'(\tau+1)-2}\right)$ networks if $k' \geq 1$.

Building a tree (lines 1-2) can be done in $\mathcal{O}(n^2m)$ time. If $k' = 0$, then at most one network (i.e. tree) is returned. Consider a fixed $k' \geq 1$ now. Determining the ST-sets costs time $\mathcal{O}(2^{n-2})$ by Lemma 3.12. If $k' \geq 1$, then STCass loops over at most $2^{n-2} + 2$ ST-sets (by Lemma 3.12) and at most $\mathcal{O}(n^\nu)$ recursively created networks, where a rough upper bound on ν is $\mathcal{O}((\tau+1)(k'-1))$. Let ϵ be the time complexity of the function GetHangEdgeCombinations, for example $\epsilon = m^2n + \tau n^\tau$ (see Lemma 3.3). The function returns $\mathcal{O}(n^\tau)$ edge combinations by Lemma 3.3. Adding a leaf (or ST-set) costs time $\mathcal{O}(\tau + mn)$, since hanging a meta-taxon in a network costs time $\mathcal{O}(\tau)$ and decollapsing it costs time $\mathcal{O}(nm)$. Let the resulting network be called N''. Observe that the number of hang-edges $|E|$ is at most $\tau+1$ (τ for representing $\mathcal{C}_x$ and at most one for representing the other clusters). If the number of hang-edges is two or more, then creating all binary refinements (on line 20 of Alg. 2) of such network N'' costs time $\mathcal{O}\left(\binom{\tau+1}{2}\binom{\tau}{2}\cdots\binom{3}{2}\right) = \mathcal{O}\left((n/2)^{\tau-1}\right)$.

Summing the running times gives a total runtime of order:

$$\mathcal{O}\left(mn^2 + 2^{n-2} + 2^{\nu(n-2)}\left[\epsilon + n^\tau \cdot n^\tau\left(\tau + mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.8}$$

Using $\nu = (\tau+1)(k'-1)$ and $\epsilon = m^2n + \tau n^\tau$, one gets the following.

$$= \mathcal{O}\left(mn^2 + 2^{n-2} + 2^{(\tau+1)(n-2)(k'-1)}\left[m^2n + \tau n^\tau + n^{2\tau}\left(\tau + mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.9}$$

$$= \mathcal{O}\left(2^{(\tau+1)(n-2)(k'-1)}\left[m^2n + n^{2\tau}\left(mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.10}$$

Hence, this is the runtime for STCass, which is the same for fixed k'. Observe that $1 \leq \tau \leq k+1$ always holds. Therefore, one may say that STCass runs in time $\mathcal{O}\left(2^{(k+2)(k-1)(n-2)}\left[m^2n + n^{2(k+1)}\left(mn + (n/2)^k\right)\right]\right)$.

The number of constructed networks is at most $\mathcal{O}\left(2^{n-2} \cdot n^{(\tau+1)(k'-1)} \cdot (n/2)^{\tau-1}\right) = \mathcal{O}\left(2^{n-\tau-1} \cdot n^{k'(\tau+1)-2}\right)$. (Observe that $\tau \geq 2$ and recall that $k' \geq 1$ and $n \geq 3$.) Hence, the number of constructed networks by STCass(k) upper bounded by $\mathcal{O}\left(2^{n-2} \cdot n^{k(k+2)-2}\right)$, since $1 \leq \tau \leq k+1$. □

Note that this is a very rough bound for the running time of STCass(k). The bottleneck is formed by the determination of all ST-sets ($\mathcal{O}(2^{n-2})$). Therefore, if this step can be eliminated, the runtime would be polynomial in n and m, for a fixed level k. This is shown below.

Lemma 3.14. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$, such that $r(\mathcal{C}) \geq 1$. Then, there are at most $|\mathcal{X}|$ different MST-sets, which can be determined in time $\mathcal{O}(|\mathcal{C}||\mathcal{X}|^4)$.*

Proof. Let $\mathcal{C}$ and $\mathcal{X}$ be as required. Define $n := |\mathcal{X}|$ and $m := |\mathcal{C}|$. The fact that there are at most n MST-sets has been proven in Cor. 4 in [15].

Now, a valid procedure to obtain the MST-sets, as proposed in the proof of Lemma 5 in [15], is repeated. "Start with a set $\mathcal{S}$ of n singleton ST-sets. If there are two distinct ST-sets S_1, $S_2 \in \mathcal{S}$ such that $S_1 \cup S_2$ is an ST-set, then remove S_1 and S_2 from $\mathcal{S}$ and add $S_1 \cup S_2$ to $\mathcal{S}$. Repeat this until it is no longer possible."

Its runtime is analysed below. Looping through at most each pair of ST-sets costs time $\mathcal{O}\left(\binom{n}{2}\right)$. In each of these iterations, it should be determined if the two selected ST-sets S_1 and S_2 are such that $S_1 \cup S_2$ is an ST-set. This requires looping through $\mathcal{C}$ and checking the compatibility with each cluster ($\mathcal{O}(mn)$). If this check succeeds, it should be validated that $\mathcal{C}|(S_1 \cup S_2)$ is separating. This requires time $\mathcal{O}(m \cdot |S_1 \cup S_2|^2) = \mathcal{O}(mn^2)$, since $|S_1 \cup S_2| \leq n$. In total, this gives the following runtime.

$$\mathcal{O}\left(\binom{n}{2}\left(mn + mn^2\right)\right) = \mathcal{O}(mn^4) \tag{3.11}$$

□

Theorem 3.15. *Let $\mathcal{C}$ be a set of m clusters on n taxa $\mathcal{X}$, such that $r(\mathcal{C}) \geq 1$. Let τ be as follows.*

$$\tau := \max_{x \in \mathcal{X}} \{|minimal(\mathcal{C}_x)|\} \tag{3.12}$$

Then, MSTCass takes time $\mathcal{O}\left(n2^{(\tau+1)(k'-1)}\left[m^2n + n^{2\tau}\left(mn + (n/2)^{\tau-1}\right)\right]\right)$, which is upper bounded by $\mathcal{O}\left(n2^{(k+2)(k-1)}\left[m^2n + n^{2(k+1)}\left(mn + (n/2)^k\right)\right]\right)$. Besides, the number of constructed networks is of order $\mathcal{O}\left(2^{1-\tau} \cdot n^{k(\tau+1)-1}\right)$.

Proof. Let $n = |\mathcal{X}|$ and $m = |\mathcal{C}|$. Following the proof of Thm. 3.13 and using

Lemma 3.14, the total runtime is as follows.

$$\mathcal{O}\left(mn^2 + mn^4 + n2^{\nu}\left[\epsilon + n^{\tau} \cdot n^{\tau}\left(\tau + mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.13}$$

Using $\nu = (\tau+1)(k'-1)$ and $\epsilon = m^2n + \tau n^{\tau}$, we get the following.

$$= \mathcal{O}\left(mn^2 + mn^4 + n2^{(\tau+1)(k'-1)}\left[m^2n + \tau n^{\tau} + n^{2\tau}\left(\tau + mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.14}$$

$$= \mathcal{O}\left(n2^{(\tau+1)(k'-1)}\left[m^2n + n^{2\tau}\left(mn + (n/2)^{\tau-1}\right)\right]\right) \tag{3.15}$$

This is the runtime for MSTCass, which is the same for fixed k'. Observe that $1 \leq \tau \leq k+1$ holds. Therefore, one may state that the runtime of MSTCass(k) is $\mathcal{O}\left(n2^{(k+2)(k-1)}\left[m^2n + n^{2(k+1)}\left(mn + (n/2)^{k}\right)\right]\right)$.

The number of constructed networks is at most $\mathcal{O}\left(n \cdot n^{(\tau+1)(k'-1)} \cdot (n/2)^{\tau-1}\right) = \mathcal{O}\left(2^{1-\tau} \cdot n^{k'(\tau+1)-1}\right)$. (Observe that $\tau \geq 2$ and recall that $k' \geq 1$ and $n \geq 3$.) Furthermore, observe that $1 \leq \tau \leq k+1$ holds. Therefore, one may state that the number of constructed networks by MSTCass is $\mathcal{O}(n^{k(k+2)-1})$. □

The time complexity and number of constructed networks of the algorithms STCass and MSTCass are summarized in Table 3.1. For comparison, the corresponding values for Cass are listed too (which originate from [12]).

	Runtime	UB #Networks
Cass	$\mathcal{O}\left(mn^{3k+2}\right)$	$\mathcal{O}\left(n^{3k}\right)$
STCass	$\mathcal{O}\left(2^{(k+2)(k-1)(n-2)}\left[m^2n + n^{2(k+1)}\left(mn + (n/2)^{k}\right)\right]\right)$	$\mathcal{O}\left(2^{n-2} \cdot n^{k(k+2)-2}\right)$
MSTCass	$\mathcal{O}\left(n2^{(k+2)(k-1)}\left[m^2n + n^{2(k+1)}\left(mn + (n/2)^{k}\right)\right]\right)$	$\mathcal{O}\left(n^{k(k+2)-1}\right)$

Table 3.1: (Upper bounds on) the time complexity of the algorithms STCass and MSTCass are listed in the second column and an upper bound (UB) on the number of constructed networks is given in the last column. Note that $m = |\mathcal{C}|$ and $n = |\mathcal{X}|$. For comparison, these values are given for Cass too (which originate from [12]). Note that it is assumed that $k \geq 1$ for STCass and MSTCass.

3.10. Heuristic Optimizations

The algorithms STCass and MSTCass can be optimized in different ways. One of them is using heuristic optimizations. In particular, we present a heuristic by which

low priority is given to the selection of certain (M)ST-sets in the inward ST-set removal phase of the algorithm.

Heuristic (ST-set priorities). Let S and S' be two (M)ST-sets. By the notation $S \to S'$, we denote that each cluster which is a superset of S, is that of S' too (which is inspired by the notation in Section 4 in [19]). Suppose that STCass is in the i-th iteration in the inward phase, i.e. the removal phase. Let $\mathcal{C}|\mathcal{X}_i$ be the set of clusters on taxa $\mathcal{X}_i$ at the moment. Let $\mathcal{M}_i$ be all (M)ST-sets therein. Suppose that there is an (M)ST-set $S \in \mathcal{M}_i$, such that $S \to S'$ for some (M)ST-set $S' \in \mathcal{M}_i$, and $S'' \not\to S$, for all $S'' \in \mathcal{M}$. Then give removing S low priority. (End of the heuristic.)

What is the motivation for it? Note that in [19], the authors define a *terminal* to be a taxon $x \in \mathcal{X}$ for which there is no another taxon $x' \in \mathcal{X}$ such that $x' \neq x$ and $x \to x'$. In our case however, S may be regarded as the opposite of a terminal, say a *source*. By Observation 3 in the same paper, we deduct that if S is such a source, then there does not exist two incompatible clusters $C_1, C_2 \in \mathcal{C}$ such that $C_1 \cap C_2 = S$. Therefore, we presume that S is not located in one of the lowest parts of any optimal network. The reason is that, if S would be below a reticulation, then S would not be an MST-set, which would imply that the network is not optimal by Thm. 3.6. On top of this, in the inward ST-set removal phase of STCass, ideally, in each iteration some ST-set S is removed from $\mathcal{C}$, such that $r(\mathcal{C} \backslash S) \leq r(\mathcal{C}) - 1$. These facts combined give us the presumption that it is not wise to remove S from $\mathcal{C}$ if there are other ST-sets to remove. Therefore, we give removing S low priority.

3.11. Further Optimizations

Several other optimization approaches will be presented in the coming chapters, like the following. Firstly, since (M)STCass(k) is called repeatedly for increasing values of k, finding a (high) lower bound on the level k would speed up the overall execution time of (M)STCass (see Chapter 5). Secondly, the function GetHangEdgeCombinations should be designed to select hang-edges and combinations as efficiently as possible (see Chapter 4).

4

Details on Building an Optimal Network

In this chapter, we give (greedy) guide lines on how to build a phylogenetic network from a set of clusters $\mathcal{C}$. The procedure is as follows, in short. Firstly, we remove leaf by leaf (or a collapsed ST-set) from $\mathcal{X}(\mathcal{C})$ and check if a tree can be built on them, say on leaf set $\mathcal{X}_0$, such that it represents the clusters in $\mathcal{C}|\mathcal{X}_0$. As soon as a tree can be built, we add the removed leaves to this tree in reversed order, leaf by leaf (or an ST-set).

Say that we want to hang leaf x in some network N, resulting in some network N', such that N' represents the clusters in $\mathcal{C}|\mathcal{X}(N')$. We will propose conditions which will ensure that N' represents the clusters in $\mathcal{C}|\mathcal{X}(N')$ in Section 4.1. Based on this, it will be discussed how to choose suitable hang-edges, in the section thereafter. On top of this, guide lines are proposed to minimize the number of reticulations in the resulting network N'. Then, requirements are given on the function GetHangEdgeCombinations in STCass (see line 9 in Alg. 2). Besides, it will turn out that it is useful to store and use a list of lowest common ancestors (LCAs) per cluster, which is addressed in Section 4.4.

4.1. Ensuring that N' Represents $\mathcal{C}|\mathcal{X}(N')$

Consider STCass as presented in Alg. 2. Suppose that $\mathcal{C}$ is a set of clusters on taxa $\mathcal{X}$. In each iteration in the outward phase of STCass, some leaf $x \in \mathcal{X}\backslash\mathcal{X}(N)$ (which

may be a collapsed ST-set) is hung in some network N, which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$. Again, this should be done in such a way, that the resulting network N' represents all clusters in $\mathcal{C}|\mathcal{X}(N')$. In this section, we address the question how to ensure this.

Definition 4.1. *Given a set of clusters $\mathcal{C}$ on the taxa $\mathcal{X}$, some network N which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$ and leaf $x \in \mathcal{X}\backslash\mathcal{X}(N)$ to hang therein, resulting in some network N', we define the set $\mathcal{C}_x^{N'}$ as given below.*

$$\mathcal{C}_x^{N'} = \{C \in \mathcal{C}|\mathcal{X}(N') : x \in C\} \tag{4.1}$$

Observe that it contains exactly all clusters on $\mathcal{X}(N')$ which contain leaf x. In the sequel of this work, we will abbreviate this set as $\mathcal{C}_x$.

Example 4.2. Let $\mathcal{C} = \{ab, abh, bh, ch, abch, abc\}$. Suppose that N is the tree on taxa a and c. Then $\mathcal{C}_h = \{ah, ch, ach\}$.

Opposite to the minimal clusters, let the *maximal clusters* in some set of clusters $\mathcal{C}$ be defined as $maximal(\mathcal{C}) = \{C \in \mathcal{C} : \nexists C' \in \mathcal{C}$ such that $C \subset C'\}$. These are the clusters in $\mathcal{C}$ for which there is no strict supercluster in $\mathcal{C}$.

Definition 4.3. *Let $\mathcal{C}$ be a set of clusters on the taxa $\mathcal{X}$. Let N be a network representing the clusters in $\mathcal{C}|\mathcal{X}(N)$ and let $x \subset \mathcal{X}$ be a leaf to be hung therein, resulting in some network N'. Then, the set of clusters $C_{\bar{x}}^{N'}$ is defined as follows.*

$$\mathcal{C}_{\bar{x}}^{N'} = maximal\left(\{C \in \mathcal{C}|\mathcal{X}(N') : x \notin C, \mathcal{X}(\mathcal{C}_x)\backslash\{x\} \subseteq C\}\right) \tag{4.2}$$

Observe that it contains the maximal clusters on $\mathcal{X}(N')$ that contain all leaves in $\mathcal{X}(\mathcal{C}_x)$, except x. Often, we denote this set shortly as $\mathcal{C}_{\bar{x}}$.

Example 4.2 (Continued)**.** Again, assume that N is the tree on taxa a and c and $\mathcal{C}$ is as before. Suppose that taxon h is to be hung in N. Then $\mathcal{C}_{\bar{h}} = \{ac\}$ (since $\mathcal{X}(\mathcal{C}_x) = ach$).

Now, let us repeat the research question of this section: How can it be ensured that hanging taxon x in network N is done in such a way, that the resulting network N' represents all clusters in $\mathcal{C}(N')$? Say that one ensures that N' will represent a certain subset of clusters $\mathcal{D} \subseteq \mathcal{C}|\mathcal{X}(N')$. Furthermore, assume that $\mathcal{D} \neq \mathcal{C}|\mathcal{X}(N')$. How should $\mathcal{D}$ be chosen, such that the property $\mathcal{D} \subseteq \mathcal{C}(N') \Rightarrow \mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}(N)$ holds?

Lemma 4.4. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$ and let N be a network on a subset of taxa $\mathcal{X}(N) \subset \mathcal{X}(\mathcal{C})$, which represents all clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that leaf $x \in \mathcal{X}\backslash\mathcal{X}(N)$ is to be hung in network N, such that all clusters in $\mathcal{C}|(\mathcal{X}(N)\cup\{x\}) = \mathcal{C}|\mathcal{X}(N')$ will be represented in the resulting network N'. Let $\mathcal{D} = \mathcal{C}_x \cup \{\mathcal{X}(N)\}$. Then, $\mathcal{D} \subseteq \mathcal{C}(N')$ implies that $\mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}(N')$.*

Proof. Let $\mathcal{C}, \mathcal{X}, x, N$ and N' be as required. The set of clusters to be represented in network N' is $\mathcal{C}|\mathcal{X}(N)$. Therein, the clusters containing x are exactly those in $\mathcal{C}_x$ (*). The other clusters were represented in network N. Assume that the cluster $\mathcal{X}(N)$ is represented in N'. Then, this implies that x is hung (a.o.) below the incoming edge of the root of N. It follows that $\mathcal{C}(N) \subset \mathcal{C}(N')$. This means that $\mathcal{X}(N) \in \mathcal{C}(N') \Rightarrow \mathcal{C}(N) \cup \{\mathcal{X}(N)\} \subseteq \mathcal{C}(N')$. Together with (*), it can be concluded that $\mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}_x \cup \mathcal{C}(N) \cup \{\mathcal{X}(N)\} \subseteq \mathcal{C}(N')$. The result follows. □

Observation 4.5. *Observe that Lemma 4.4 does not hold if $\mathcal{C}_x$ would only contain the minimal clusters comprising x, i.e. if $\mathcal{C}_x$ would be as follows.*

$$\mathcal{C}_x = \mathit{minimal}\,(\{C \in \mathcal{C}|\mathcal{X}(N')|x \in C\}) \tag{4.3}$$

To show this, consider the following example. Let $\mathcal{C}$ be as defined in Ex. 4.2. Let network N on taxa a, b, c, h, o be as depicted in Fig. 4.1. You may check yourself that $\mathcal{C}|\mathcal{X}(N) = \{bc, bh, bch, ch\}$ and that N represents all these clusters and furthermore, no reticulation is redundant. Suppose that leaf a should be hung in network N. Now, $\mathcal{C}_a = \{ab\}$ and $\mathcal{C}_{\bar{a}} = \{bch\}$. Based on this, we may decide to hang a back below edge (u, t). However, N' does not represent the clusters abc and abh now!

Conjecture 4.6 (Minimal $\mathcal{C}_x$ Sufficient)**.** *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. At some point in the outward phase of STCass, let N be a network on $\mathcal{X}(N) \subset \mathcal{X}$ representing the clusters in $\mathcal{C}|\mathcal{X}(N)$. Let x be a leaf in $\mathcal{X}\backslash\mathcal{X}(N)$ to be hung back therein. Suppose that the set $\mathcal{C}_x$ is defined as a set of minimal clusters, namely $\mathcal{C}_x = \mathit{minimal}\,(\{C \in \mathcal{C}|\mathcal{X}(N')|x \in C\})$. Besides, let $\mathcal{D} = \mathcal{C}_x \cup \{\mathcal{X}(N)\}$. Let (W) denote the implication $\mathcal{D} \subseteq \mathcal{C}(N') \Rightarrow \mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}(N')$. If (W) does not hold when hanging taxon x in N, then there is another sequence of removing leaves (or ST-*

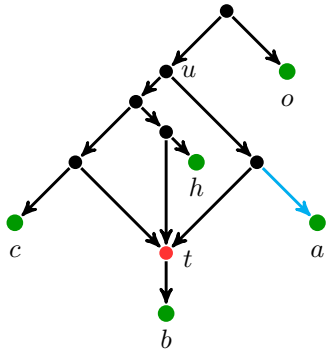


Figure 4.1: The black edges form a network N on the taxa b, c, h and o, which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$, where $\mathcal{C}$ is as given in Ex. 4.2. Leaf a with its cyan cut-edge is hung below edge (u, t), which is an illegal way of hanging leaf a in N, since the clusters abh and abc are not represented now.

sets) in the inward phase of STCass, such that (W) is true for each newly inserted leaf in the outward phase.

Example 4.2 (Continued)**.** In order to get some insights in the conjecture above, consider the following example. Suppose that (M)STCass has removed taxon h. Then the tree N on abc, representing the corresponding clusters in $\mathcal{C}|abc$, is as illustrated in Fig. 4.2a. Note that $\mathcal{C}|abc = \{ab, abc\}$. Taxon h is to be hung in

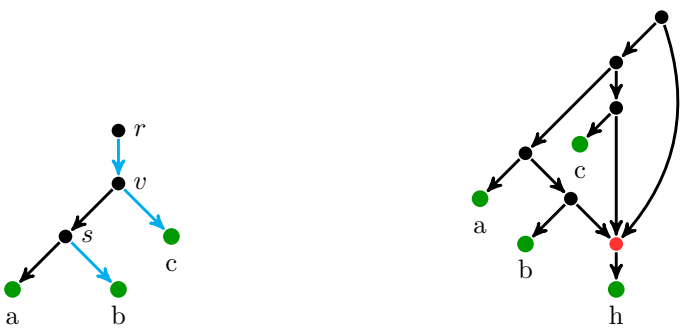


Figure 4.2: (a) represents the unique tree on abc, representing the clusters $\mathcal{C}|abc = \{ab, abc\}$. Taxon h is to be inserted below the cyan coloured edges. (b) shows the resulting network.

this tree. Note that $\mathcal{C}_h = \{bh, ch\}$ and $\mathcal{C}_{\bar{h}} = \{abc\}$. Therefore, the hang-edges

should be those as indicated by the cyan colour in Fig. 4.2a. After hanging taxon h below these edges, the resulting network will be as depicted in Fig. 4.2b (where the root-edge is removed). Observe that it has two reticulations. Furthermore, this is optimal, since there is a triangle in the corresponding incompatibility graph (see Thm. 5.5 in Section 5.3). This shows that there is another ST-set tree sequence than $\mathcal{S} = \{a, b\}$, which does give an optimal solution.

Conjecture 4.7. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$ and let N be a network on a subset of taxa $\mathcal{X}(N) \subset \mathcal{X}(\mathcal{C})$, which represents all clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that leaf $x \in \mathcal{X}\backslash\mathcal{X}(N)$ is to be hung in network N, such that all clusters in $\mathcal{C}|(\mathcal{X}(N) \cup \{x\}) = \mathcal{C}|\mathcal{X}(N')$ will be represented in the resulting network N'. Let $\mathcal{D}$ be defined as follows.*

$$C^* := \textit{any cluster in } \mathcal{C}_{\bar{x}} \tag{4.4}$$

$$\mathcal{D} := \mathcal{C}_x \cup \{C^*\} \tag{4.5}$$

Then, $\mathcal{D} \subseteq \mathcal{C}(N')$ implies $\mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}(N')$.

Lemma 4.8. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a network on $\mathcal{X}(N) \subset \mathcal{X}$ which represents all clusters in $\mathcal{C}|\mathcal{X}(N)$. Let $x \in \mathcal{X}\backslash\mathcal{X}(N)$ be a taxon to hang in the network. Then, the resulting network N' represents some cluster $C \in \mathcal{C}_{\bar{x}}$ if for some $v \in LCA(C) \subset V(N)$, there is be a predecessor p of v in N', and a path P from p to x, and a subtree T_C on C having root v in some switching of N', such that P does not intersect with T_C in N'.*

Proof. Let $\mathcal{C}, \mathcal{X}, N, N', C$ and x be as required. Suppose that N' represents C. Then, there is a subtree T_C on C in some switching T of N' whose root is v (note that $v \in LCA(C)$). Observe that x is connected to the root of $T\backslash T_C$, otherwise, N' would not represent cluster C (which does not contain x). Hence, there is a path P from a predecessor of v to x which does not intersect with T_C in N'. □

Now, given that we hang taxon x in some network N, we know (guide lines) how to ensure that the clusters $\mathcal{C}|\mathcal{X}(N')$ will be represented in the resulting network N'. This can be used to determine suitable hang-edges to hang x below, which will be handled in the next sections.

4.2. Determining valid Hang-Edges

In the building phase of (M)STCass, a function is used called GetHangEdgeCombinations. This function returns a list of edge combinations in the network N at that moment. Each combination in it represents a set of edges below which leaf x will be hung, such that the resulting network N' will represent all clusters in $\mathcal{C}|\mathcal{X}(N')$. This section addresses the question: below which edge(s) should taxon x be hung, such that after this, N' represents all clusters in $\mathcal{C}|\mathcal{X}(N')$?

Before proposing the next theorem, consider the definition of 0-edges. An edge in a network is called a *0-edge* if it represents $\emptyset$ as (softwired) cluster. For example, each incoming edge of a reticulation is a *0-edge*. Observe that there is no path from a 0-edge to any leaf, without any reticulation on it. Furthermore, define $LCA(C)$ as the set of lowest common ancestors (LCAs) of the leaves in cluster C.

Theorem 4.9 (Hang-edges)**.** *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a binary network on $\mathcal{X}(N) \subset \mathcal{X}$ which represents all clusters in $\mathcal{C}|\mathcal{X}(N)$. Let $x \in \mathcal{X}\backslash\mathcal{X}(N)$ be a leaf to hang in the network, such that the resulting network N' will represent all clusters in $\mathcal{C}|\mathcal{X}(N')$. Consider some cluster $C' \in \mathcal{C}_x$. Define the cluster $C := C'\backslash\{x\} \in \mathcal{C}(N)$. Restrict hanging x in N to hanging it below edges (i.e. not below any node). Then for some vertex $v \in LCA(C) \subset V(N)$ which represents C, leaf x should be hung...*

1. *...below incoming edge (u, v) of v; or*
2. *below any edge in any subtree T_C on C in N, except if N' should represent cluster C and if besides, there would be no path connecting x with $N'\backslash T'_C$ in N', where T'_C is the subtree in N' corresponding to T_C in N; or*
3. *below any 0-edge that is reachable via a path consisting of only (zero or more) 0-edges from u; or*
4. *below any 0-edge that is reachable via a path consisting of only (zero or more) 0-edges, from any vertex in a subtree T_C on C as described above, except if $C \in \mathcal{C}|\mathcal{X}(N')$ and furthermore, if there would be no path in N' connecting x with $N'\backslash T_C$; or*
5. *below any edge from which u is reachable and which represents cluster C.*

Proof. Let $\mathcal{C}, x, C', C, N$ and v be as required for the lemma. Observe that v cannot be a reticulation and therefore, it has in-degree 1, so (u, v) is uniquely determined

(given node v). Now, for each of the four cases listed in the theorem, we have to show that if x is hung below (a.o.) an edge as defined there, the resulting network N' will represent cluster C'. Furthermore, we have to give the conditions on which cluster C would be represented too.

1. The incoming edge of v represents cluster C. Therefore, hanging leaf x below it would imply that the resulting network represents a.o. the clusters C and $C \cup \{x\}$, where the last equals C'.
2. Similar to the first case, N' will represent C' in this case. If N' should represent C too, then there should be a path in N' connecting x with $N' \backslash T'_C$, otherwise x would always be included in any supercluster $C' \supseteq C$.
3. Again, similar as in point 1, N' will represent C'. Edge (u, v) (which is not a 0-edge) still represents cluster C.
4. Assume that there is a 0-edge e adjacent to u or a node in a subtree T' on $\mathcal{X}(C)$, of any switching T_N of N. Then, hanging x below it would imply that $C \cup \{x\}$ is represented in N', i.e. cluster C'. Now, assume there is another 0-edge adjacent to e. Then analogously, the same conclusion can be drawn. This can be repeated, which shows that N' will represent C'. Analogous to point 2, N' will represent C too, if and only if there is a path in N' connecting x with $N' \backslash T_C$.
5. Suppose that (p, q) is an edge (from which u is reachable and) which represents C. By placing a new split node s on it and by hanging x below it, N' will represent both cluster C (edge (s, q) will represent it) and cluster $C \cup \{x\}$, i.e. cluster C' (edge (p, s) will represent it).

These cases show that leaf x *may* be hung below one of the edges as described, since N' represents cluster C' then, and if required, C too. Observe that valid edges to hang x below, other than those in the cases 1 - 4, are exactly the edges described in point 5. In other words, if x is not hung below one of these edges, then N" will not represent cluster C'. This means that x *should* be hung below one of the edges as described in these points. □

Observe that, given some $v \in LCA(C)$ in a binary network and its parent node u, each incoming edge of u is an edge as described in point 5 in Thm. 4.9. Note that all our executions of STCass pointed out that it may be sufficient to consider only a subset of the edges that are described in Thm. 4.9, see Conj. 4.10.

Conjecture 4.10 (Restricted Hang-edges)**.** *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let G be some connected component in $IG(\mathcal{C})$. Suppose that N is a binary network which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$, in which leaf $x \in \mathcal{X}\backslash\mathcal{X}(N)$ should be hung. Consider some cluster $C' \in \mathcal{C}_x$. Define the cluster $C := C'\backslash\{x\} \in \mathcal{C}(N)$. Restrict hanging x in N to hanging it below edges (i.e. not below any node). Then for some vertex $v \in LCA(C) \subset V(N)$, leaf x should be hung...*

1. *...below incoming edge (u, v) of v; or*
2. *below the incoming edge of any taxon $y \in C$; or*
3. *below any 0-edge that is reachable via a path consisting of only (zero or more) 0-edges from u; or*
4. *below any 0-edge that is reachable via a path consisting of only (zero or more) 0-edges, from any vertex in a subtree T_C on C as described above, except if $C \in \mathcal{C}|\mathcal{X}(N')$ and furthermore, if there would be no path in N' connecting x with $N'\backslash T_C$; or*
5. *below any edge from which u is reachable and which represents cluster C.*

So far, only hang-edges are considered which will ensure that each cluster $C \in \mathcal{C}_x$ will be represented in N'. Now, hang-edges are determined to ensure that the clusters in $\mathcal{C}_{\bar{x}}$ will be represented too.

Theorem 4.11. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a network on $\mathcal{X}(N) \subset \mathcal{X}$ which represents all clusters in $\mathcal{C}|\mathcal{X}(N)$. Let $x \in \mathcal{X}\backslash\mathcal{X}(N)$ be a taxon to hang in the network. Then, the resulting network N' represents some cluster $C \in \mathcal{C}_{\bar{x}}$, if at least one hang-edge $(u, p) \in E(N)$ is chosen such that there is some $v \in LCA(C) \subset V(N)$ reachable from p and such that v represents C.*

Proof. Let $\mathcal{C}, \mathcal{X}, N, x$ be as required. Let $C \in \mathcal{C}_{\bar{x}}$. Let edge $(u, p) \in E(N)$ be such that there is some $v \in LCA(C) \subset V(N)$ reachable from p and such that v represents C. Let w be the split node placed on edge (u, p), below which x is hung (possibly

with a reticulation t in between). Then, edge (w, x) or the path (w, t, x) is disjoint from any subtree T_C on C in some switching of N'. Therefore, N' will represent cluster C, by Lemma 4.8. □

4.3. Minimizing $r(N')$

In the previous section, it has been shown how to choose a set of hang-edges, such that network N' will represent all clusters in $\mathcal{C}|\mathcal{X}(N')$ after hanging leaf (or ST-set) x in N. Some of the hang-edges -possibly all- ensure that N' will represent the clusters in $\mathcal{C}|\mathcal{X}(N')$ which contain x. If there are other hang-edges, these are intended to let N' represent the clusters in $\mathcal{C}|\mathcal{X}(N')$ which do not contain x. For each cluster $C \in \mathcal{C}|\mathcal{X}(N')$, we know how to determine a set of hang-edges E_C (as defined in Thm. 4.9 for any cluster $C \in \mathcal{C}_x$, and see Lemma 4.8 for requirements for any cluster $C \in \mathcal{C}_{\bar{x}}$), such that, if x is hung below one of these edges, cluster C will be represented in network N'. Now, we address the following question: given these hang-edges E_C per cluster $C \in \mathcal{C}|\mathcal{X}(N')$, how to determine the set of hang-edges E below which to hang taxon x actually, such that in the end, $r(N)$ is minimized?

In order to minimize $r(N')$, we should solve the optimization problem (D) as given below. Note that $r(N') = r(N)+|E|-1$ and therefore, minimizing $r(N')$ given $r(N)$ equals minimizing $|E|$. Moreover, note that for each cluster $C \in \mathcal{C}|\mathcal{X}(N')$, there should be a suitable hang-edge, which is modelled by Eq. 4.7.

$$(D) \qquad \min |E| \tag{4.6}$$

$$\text{subject to: } E_C \cap E \neq \emptyset, \qquad \forall C \in \mathcal{C}|\mathcal{X}(N') \tag{4.7}$$

Theorem 4.12 (Minimal Cluster Hang-Edge Sets)**.** *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a network representing the clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that x is a taxon in $\mathcal{X}\backslash\mathcal{X}(N)$, which is to be hung in N, in order to get some network N' which should represent $\mathcal{C}|\mathcal{X}(N')$. Consider any cluster $C \in \mathcal{C}_x$. Let E_C be the set of hang-edges corresponding to those as defined in Thm. 4.9. Then in optimization problem (D), i.e. in order to minimize $r(N')$, it is sufficient to consider only the minimal hang-edge sets E_C, i.e. the following sets.*

$$\widehat{\mathcal{E}} := \mathit{minimal}\,(\{E_C | \forall C \in \mathcal{C}|\mathcal{X}(N')\}) \tag{4.8}$$

Proof. Let E_C be any set in $\widehat{\mathcal{E}}$. Let E^* be any optimal solution for set E in optimization problem (D). If $E_C \cap E^* \neq \emptyset$, then $E_{C'} \cap E^* \neq \emptyset$ for all $E_{C'} \supset E_C$. Hence, $E_C \cap E^* \neq \emptyset$ for all $C \in \mathcal{C}|\mathcal{X}(N')$. Therefore, it is sufficient to consider only the minimal hang-edge sets, i.e. those in $\widehat{\mathcal{E}}$. □

Conjecture 4.13 (Reduced Minimal Hang-Edge Sets)**.** *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a network representing the clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that x is a leaf in $\mathcal{X}\backslash\mathcal{X}(N)$, which is to be hung in N, in order to get some network N' which should represent $\mathcal{C}|\mathcal{X}(N')$. Consider any cluster $C \in \mathcal{C}_x$. Let E_C be the set of hang-edges corresponding to those as defined in Conj. 4.10. Then in optimization problem (D), i.e. in order to minimize $r(N')$, it is sufficient to consider only the minimal hang-edge sets E_C, i.e. the following sets.*

$$\widehat{\mathcal{E}} := \textit{minimal}\,(\{E_C | \forall C \in \mathcal{C}|\mathcal{X}(N')\}) \tag{4.9}$$

Conjecture 4.14. *Eq. 4.6 may be decomposed into the following optimization problem (E), where $E = E_x \cup E_{\bar{x}}$, where E_x and $E_{\bar{x}}$ are disjoint and besides, its objective value equals that of (D).*

$$(E) \qquad \min\ |E_x| + |E_{\bar{x}}| \tag{4.10}$$

$$\textit{subject to: } E_C \cap E_x \neq \emptyset, \qquad \forall C \in \mathcal{C}_x \tag{4.11}$$

$$E_C \cap E_{\bar{x}} \neq \emptyset, \qquad \forall C \in \mathcal{C}_{\bar{x}} \tag{4.12}$$

Theorem 4.15. *Let $(E_x^*, E_{\bar{x}}^*)$ be an optimal solution for $(E_x, E_{\bar{x}})$ in (E). Then $|E_{\bar{x}}^*| = 1$ if and only if there is a supercluster C of $\mathcal{X}(\mathcal{C}_x)$ in $\mathcal{C}$. Otherwise, $|E_{\bar{x}}^*| = 0$.*

Proof. Let $(E_x^*, E_{\bar{x}}^*)$ be an optimal solution for $(E_x, E_{\bar{x}})$ in (E). Observe from Lemma 4.4, that by adding an edge from the incoming edge of the root of N to the added reticulation (i.e. the parent of x), $\mathcal{C}(N) \subset \mathcal{C}(N')$ holds. Therefore, in any case, $|E_{\bar{x}}^*| \leq 1$. Besides, observe that $E_{\bar{x}}^* \neq \emptyset$ if and only if there is a supercluster C of $\mathcal{X}(\mathcal{C}_x)$ in $\mathcal{C}$, i.e. $\mathcal{C}_{\hat{x}} \neq \emptyset$ (compare lines 7 - 11 in Alg. 3). The result follows. □

Observation 4.16. *Note that for an optimal solution $(E_x^*, E_{\bar{x}}^*)$ of (E), $|E_x^*| \leq \tau$ holds, where τ is the number of trees from which the clusters in $\mathcal{C}$ originate. Since $|E_{\bar{x}}^*| \leq 1$ by Thm. 4.15, it follows that at most τ binary reticulations will be added to N when hanging taxon x therein.*

Corollary 4.17. *By Thm. 4.15, (E) can be rewritten as follows.*

$$(F) \qquad \min\ |E_x| + \mathbf{1}_{\mathcal{C}_{\bar{x}} \neq \emptyset} \tag{4.13}$$

$$\text{subject to: } E_C \cap E_x \neq \emptyset, \qquad \forall C \in \mathcal{C}_x \tag{4.14}$$

Proof. This directly follows from (the proof of) Thm. 4.15. □

Corollary 4.18. *Let $\mathcal{C}$ be a set of clusters on $\mathcal{X}$. Let N be a network representing the clusters in $\mathcal{C}|\mathcal{X}(N)$. Let $x \in \mathcal{X}\backslash\mathcal{X}(N)$ be a taxon to hang in it, such that the resulting network N' will represent $\mathcal{C}|\mathcal{X}(N')$. In order to let N' represent all the clusters in $\mathcal{C}|\mathcal{X}(N')$ which do not contain x, x should be hung (a.o.) below one of the edges in $\bigcap_{C \in \mathcal{C}_{\bar{x}}} E_C$.*

Proof. Let $(E_x^*, E_{\bar{x}}^*)$ be an optimal solution for $(E_x, E_{\bar{x}})$ in (E). Suppose that $\mathcal{C}_{\bar{x}} \neq \emptyset$. Then by Thm. 4.15, $|E_{\bar{x}}| = 1$, i.e. one hang-edge should be chosen such that hanging x below it, N' will represent $\mathcal{C}$. Observe that $\bigcap_{C \in \mathcal{C}_{\bar{x}}} E_C = \bigcap_{C \in \mathcal{C}: x \notin C} E_C$. The result follows. □

Conjecture 4.19 (Smallest Hang-Edge Combinations)**.** *Consider STCass(k) (see Alg. 2). Let $\mathcal{E}$ be all hang-edge combinations (which are in principle determined by the function GetHangEdgeCombinations on line 11). It is sufficient to use only those hang-edge combinations $E \in \mathcal{E}$ whose size is minimal, i.e. the following set $\mathcal{E}'$, in order to obtain an optimal network finally.*

$$\mathcal{E}' := \left\{ E' \in \mathcal{E} : |E'| = \min_{E \in \mathcal{E}} |E| \right\} \tag{4.15}$$

In order to make this section complete, it remains to show how to minimize $|E_x|$ in (E). Note that the solution to (D) is the so-called vertex cover number in a hypergraph. The problem is called the hypergraph vertex cover problem or equivalently, the set cover problem, which is one of Karp's 21 NP-complete problems. Despite its complexity in general, we suspect that in this case, the problem can be solved in polynomial time.

Conjecture 4.20. *Optimization problem (F) can be solved in polynomial time.*

4.4. Lowest Common Ancestors

In Section 4.2, where it has been presented how to choose valid hang-edges, an LCA of some cluster was needed regularly. This suggests that in practice, it might be useful to store an indexed list of LCAs per cluster in the current network. Therefore, let $LCA(N;C)$ represent the set of the lowest common ancestors of cluster C in network N.

Lemma 4.21. *Let $\mathcal{C}$ be a set of clusters on the set of taxa $\mathcal{X}$. Let N be a network representing the clusters in $\mathcal{C}|\mathcal{X}(N)$. Let x be a leaf that is added to network N and call the resulting network N'. Let C' be a cluster from $\mathcal{C}_x$ and define $C := C'\backslash\{x\}$. Then, the LCAs of C' in N' are as follows.*

$$LCA(N';C') = \bigcup_{v \in LCA(N;C)} LCA(N';\{v,x\}) \tag{4.16}$$

Proof. Let $\mathcal{C}, x, N, N', C$ and C' be as required. Observe that any LCA of $C' = C \cup \{x\}$ in N', is an LCA of some $v \in LCA(N;C)$ and x. □

Using Alg. 4, the LCAs per cluster in $C|\mathcal{X}(N)$ can be stored and updated efficiently after hanging a leaf in network N. Therefore, when executing (M)STCass, it is recommended to save a list of the clusters and their LCAs -under this condition that for each LCA v of some cluster C that is stored, edge (u,v) represents cluster C-, for the sake of computational efficiency.

Lemma 4.22. *Let $\mathcal{X}$ be a set of taxa. Let $\mathcal{C}$ be a set of clusters thereon. Let N be a network which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that taxon $x \in \mathcal{X}$ has been hung in this network below certain edges, such that the resulting network N' represents the clusters in $\mathcal{C}|\mathcal{X}(N')$. Let $\mathcal{L}$ be an indexed list, such that for each cluster C in $\mathcal{C}|\mathcal{X}(N)$, $\mathcal{L}(C)$ contains the lowest common ancestors of C, with one restriction: each LCA $v \in \mathcal{L}(C)$ is such that v represents cluster C. Then algorithm UpdateLCA (see Alg. 4) updates the LCAs $\mathcal{L}$ such that after execution, $\mathcal{L}(C')$ contains all LCAs v' which represent C' in N', for each cluster C' in $\mathcal{C}|\mathcal{X}(N')$.*

Proof. Let $\mathcal{C}, \mathcal{X}, N$ and N' be as required. Note that the lines 1 - 2 in Alg. 4 ensure that in the first call of UpdateLCA, the LCAs of the clusters in $\mathcal{C}|\mathcal{X}(N)$ are known. Let C' be any cluster in $\mathcal{C}|\mathcal{X}(N')$ which contains x. Let C be cluster C' without x, i.e. $C := C'\backslash\{x\}$. By Lemma 4.21, these facts imply that Eq. 4.16

holds. This is determined on the lines 7 - 15 in Alg. 4, with the restriction that the LCAs v on the right-hand side of Eq. 4.16, all represent cluster C. If some LCA $v \in LCA(N;C)$ would not represent cluster C, then $LCA(N';\{v,x\})$ will not represent cluster C' neither. Hence, all LCAs of C' which represent C' in N', will be contained in $\mathcal{L}(C')$. □

Note that this lemma leaves room for $\mathcal{L}(C')$ to contain LCAs of cluster C' in $\mathcal{C}|\mathcal{X}(N')$, which do not represent cluster C'. Below, we will see that the function UpdateLCA in Alg. 4 will correctly update the stored LCAs $\mathcal{L}$, such that fore each C', $\mathcal{L}(C')$ exactly comprises the LCAs v which represent C'.

Theorem 4.23. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let N be a network which represents the clusters in $\mathcal{C}|\mathcal{X}(N)$. Suppose that leaf x has been hung in this network below certain edges, such that the resulting network N' represents the clusters in $\mathcal{C}|\mathcal{X}(N')$. Let $\mathcal{L}$ be an indexed list, such that for each cluster C in $\mathcal{C}|\mathcal{X}(N)$, $\mathcal{L}(C)$ contains the LCAs of C, with one restriction: each LCA $v \in \mathcal{L}(C)$ is such that v represents cluster C. Then algorithm UpdateLCA (see Alg. 4) updates the LCAs $\mathcal{L}$ such that after execution, $\mathcal{L}(C')$ contains exactly the LCAs of each cluster C' in $\mathcal{C}|\mathcal{X}(N')$, with the restriction that for each $v' \in \mathcal{L}(C')$, v' represents C'.*

Proof. Let $\mathcal{C}, \mathcal{X}, N$ and N' be as required. By Lemma 4.22, for each cluster C' in $\mathcal{C}|\mathcal{X}(N')$, $\mathcal{L}(C')$ contains all LCAs v' which represent C' in N'. Let C' be any cluster in $\mathcal{C}|\mathcal{X}(N')$ which contains x. Let C be cluster C' without x, i.e. $C := C' \backslash \{x\}$.

It should be shown that for each $v' \in \mathcal{L}(C')$, v' represents C'. Let v be any LCA from $\mathcal{L}(C)$. Let (u', v) be its incoming edge in N'. If $|LCA(N;C)| = 1$, then let us denote the unique LCA as $lca(N;C)$. Suppose that x is hung below an edge of types 1, ..., 5 in Thm. 4.9. Let us consider each case separately.

1. In this case, node $u' = lca(N';\{v,x\})$ represents cluster C'. v still represents C.
2. Node $v = lca(N';\{v,x\})$ represents C'. If C would be in $\mathcal{C}|\mathcal{X}(N')$, then the statement ensures that there is a path from $N' \backslash T_C$ to x, not intersecting T_C, which ensures that N' actually represents C.
3. Node $u = lca(N';\{v,x\})$ represents C'. C is represented by node v.

4. Node $v = lca(N'; \{v, x\})$ represents C'. If C would be in $\mathcal{C}|\mathcal{X}(N')$, then the statement ensures that there is a path from $N' \backslash T_C$ to x, not intersecting T_C, which ensures that N' actually represents C.

5. Let p be the node dividing any edge as described in this case, such that p is a parent of x or a parent of its parent. Then, p in $LCA(N'; \{v, x\})$ represents C' and furthermore, node v still represents C in N'. Note that there might be multiple of such nodes p.

□

Algorithm 4: Suppose that leaf x is inserted into network N, that represented all clusters $\mathcal{C}|\mathcal{X}(N)$, such that the resulting network N' represents all clusters in $\mathcal{C}|\mathcal{X}(N')$. Given that $\mathcal{L}(C)$ contains exactly the LCAs v for each cluster $C \in \mathcal{C}|\mathcal{X}(N)$, such that v represents C, this algorithm updates $\mathcal{L}$, such that $\mathcal{L}(C')$ will contain exactly those LCAs of C' for each cluster $C' \in \mathcal{C}|\mathcal{X}(N')$, which represent C' (see Thm. 4.23).

Input: A cluster set $\mathcal{C}$, the set of taxa $\mathcal{X}$, network N satisfying $\mathcal{C}|\mathcal{X}(N) \subseteq \mathcal{C}(N)$, network N' satisfying $\mathcal{C}|\mathcal{X}(N') \subseteq \mathcal{C}(N')$, leaf x and an indexed list $\mathcal{L}$. See the text for details.

Output: UpdateLCA$(\mathcal{C}, \mathcal{X}, N, N', x, \mathcal{L})$

```
/* Inititialize ℒ, if needed                                          */
if N is a tree then
    ℒ := GetLCAs(N, 𝒞)          // Determine the LCAs per cluster in 𝒞|𝒳(N)
t := GetParent(x, N')
for cluster C' ∈ 𝒞|𝒳(N') do
    if x ∉ C' then
        continue
    /* Compute L := LCA(C'), such that each LCA v ∈ L represents C', using
       a.o. ℒ(C)                                                      */
    C := C'\{x}
    if |C| = 1 then
        L := C
    else
        L := ℒ(C)
    for v ∈ L do                                          // v represents C
        for v' ∈ LCA(N'; {v, t}) do
            L := L ∪ {v'}
    ℒ(C') := L
/* Clean ℒ                                                            */
for each cluster C ∈ 𝒞|𝒳(N) indexed in ℒ do
    if C ∉ 𝒞' then
        Remove ℒ(C)
return ℒ                        // Return the updated LCAs per cluster in 𝒞|𝒳(N')
```

4.5. Exclude Certain Binary Refinements

In the outward building phase of (M)STCass, after hanging some leaf x below a reticulation in network N'' (see line 19 in Alg. 2), all binary refinements N''' of the resulting network are created and (possibly) saved. This is not always needed.

Observation 4.24. *After hanging a leaf below a reticulation in (M)STCass, it is not always necessary to continue with (store) all binary refinements of the resulting network.*

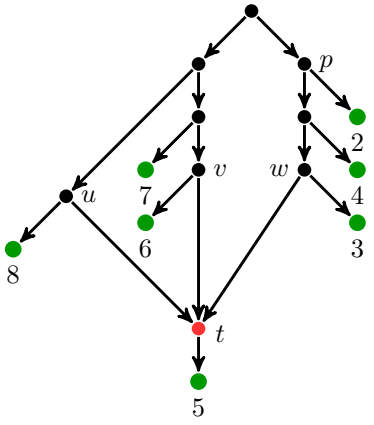
Example 4.25. Consider the four trees $\mathcal{T}$ in Fig. 3.1. Let $\mathcal{C} = Cl(\mathcal{T})$. Observe that the network as shown in Fig. 4.3a, represents all clusters in $\mathcal{C}|2345678$. The lastly added leaf so far, is leaf 5, which has been placed below a 3-input reticulation t. We will show that it is not needed to store and investigate all binary refinements of this network.

Now, leaf 1 is going to be hung in the network. Note that $minimal(\mathcal{C}_1) = \{12, 13, 15\}$ and $\mathcal{C}_{\bar{1}} = \emptyset$. Furthermore, exactly one binary reticulation should be added in order to obtain an optimal network. It is left to the reader to check that in case (b) in Fig. 4.3, leaf 1 cannot be added below exactly one binary reticulation, such that the resulting network will represent both the clusters 12 and 1567. Similarly, in case (c), the clusters 13 and 1567 cannot be represented simultaneously after adding the leaf. In case (d) however, leaf 1 may be hung below a.o. edge (t_1, t_2). Therefore, the cases (b) and (c) can be excluded from further exploration.

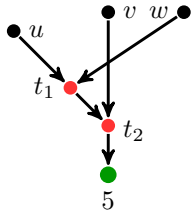
4.6. Exclude Certain Hang-Edges

Besides excluding certain binary network refinements, some hang-edges can be excluded from investigation, from the ones as defined in Thm. 4.9 and Conj. 4.10. This is illustrated below.

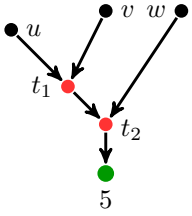
Example 4.25 (Continued)**.** Consider Fig. 4.3a again and suppose that leaf 1 is going to be hung below a new binary reticulation. Since the resulting network should represent cluster 12, leaf 1 should be hung a.o. below edge $(p, 2)$. Besides, the clusters 13 and 1567 should be represented and hence, observe that the reticulation



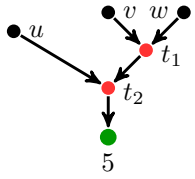
(a) This network represents all clusters in $Cl(\mathcal{T})|2345678$ for the four trees $\mathcal{T}$ in Fig. 3.1.



(b) This binary refinement does not have to be considered, since leaf 1 cannot be added below one binary reticulation such that both clusters 12 and 1567 will be represented.



(c) This binary refinement does not have to be considered neither, since leaf 1 cannot be added below one binary reticulation such that both clusters 13 and 1567 will be represented.



(d) This binary refinement is the only useful one.

Figure 4.3: This is an illustration that not all binary refinements have to be considered for further exploration in the outward phase of (M)STCass. Given the network in (a), only binary refinement (d) can be used in the next iteration of the algorithm, when leaf 1 is hung in the network.

may not be connected to edge (w, t). Furthermore, the leaf may not be hung below edge (v, t), since then, cluster 13 would not be represented. Analogously, the leaf may not be hung below the edges (u, t) and $(t, 5)$.

Now, the knowledge that it is sufficient to add exactly one binary reticulation, forces us to decompose reticulation t into two binary reticulations. This should be done as illustrated in Fig. 4.3d (as shown before). Hence, the only valid hang-edge that is left, is edge (t_1, t_2) (compare Fig. 3.2a).

5

Lower Bounds on the Minimum Reticulation Number

Given some set $\mathcal{X}$ of taxa and a set $\mathcal{C}$ of clusters thereon, now, from the previous chapter, we know how to build an optimal network which represents $\mathcal{C}$, by using the proposed algorithm STCass. Given a connected component of the incompatibility graph, STCass(k) tries to build a corresponding biconnected component comprising at most k reticulations.

Without any knowledge on the level, the original Cass algorithm has been designed with the idea of calling STCass(k) repeatedly with $k = 0, 1, 2$, etc., until a level-k (sub)network is found. The resulting (biconnected) components are combined afterwards, in order to get an optimal network finally (see Alg. 6 for an overview of this top-level procedure).

It follows that the overall execution time of this procedure can be reduced if a high lower bound on k is known. Therefore, in this chapter, we study several substructures of a connected component G in $IG(\mathcal{C})$, and what they impose on the reticulation number $r(G) = r(N)$ for the corresponding biconnected component N that represents the clusters in $V(G)$. As a tool to study this, we introduce a type of network, which we call a collnet, in the next section.

5.1. Collnets

As a tool to study the minimum number of reticulations needed in a network which represents certain clusters, we impose some requirements on a network and call the resulting network a collnet. Recall that if MST-set $S \subset \mathcal{X}$ is *collapsed*, then this means that in each cluster $C \in \mathcal{C}$ which is a strict superset of S, the elements of S are replaced by a single new taxon, e.g. having label l and furthermore, clusters in $\mathcal{C}$ which are a subset of S are neglected. This implies that the elements of S are no longer in $\mathcal{X}$ and $\mathcal{X}$ contains l now.

Definition 5.1. *Let $\mathcal{C}$ be a set of clusters. Let N be an (optimal) network which represents the clusters in $\mathcal{C}$. Furthermore, let N be such that all MST-sets are singletons (which is possible by Cor. 11 in [15]). We call the resulting network an (optimal) collnet (collapsed MST-sets network).*

One **property** of a collnet is the following. Given that N is a network which represents the clusters in $\mathcal{C}$, let N' be the corresponding collnet, obtained by moving all MST-sets below cut-edges and by collapsing all MST-sets. Then, $r(N) = r(N')$, by Cor. 11 in [15].

5.2. $|minimal(\mathcal{C}(N))|$ versus $r(N)$

In order to derive a lower bound on $r(G)$, for some connected component G in $IG(\mathcal{C})$, one might consider the number of minimal clusters represented in a corresponding biconnected component (network) N. Given an upper bound on $|minimal(\mathcal{C}(N))|$ in terms of $r(N)$ and a lower bound on $|minimal(\mathcal{C}(N))|$ in terms of $|minimal(\mathcal{C})|$, one can derive a lower bound on $r(N)$ in terms of $|minimal(\mathcal{C})|$. Note that from now on, $\mathbf{1}_x$ will be used to denote the indicator function, which is one if condition x is true and zero otherwise.

Theorem 5.2. *The number $|minimal(\mathcal{C}(N))|$ of minimal clusters represented in an optimal network N may scale quadratically in the number of reticulations $r(N)$.*

Proof. Suppose that the subnetwork N' as illustrated in Fig. 5.1 is part of a larger optimal network N. Then, the number of minimal clusters in N' is as follows.

$$|minimal(\mathcal{C}(N'))| = \mathbf{1}_{n\geq 2} + q\mathbf{1}_{n\geq 1} + \binom{q}{2} \leq \frac{1}{2}q^2 + \frac{1}{2}q + 1 \tag{5.1}$$

Since $|minimal(\mathcal{C}(N))| \geq |minimal(\mathcal{C}(N'))|$ and $r(N) \geq q$, the result follows.

□

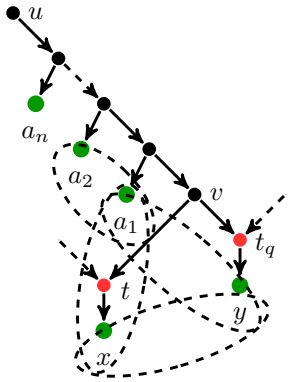


Figure 5.1: Assume there are $q \geq 2$ reticulations below node v and $n \geq 0$. Then, the number of minimal clusters in this subnetwork N' is at most $2r(N')^2 + r(N') + 1$. The clusters $\{a_1, x\}$ and $\{a_1, y\}$ are represented if and only if $n \geq 1$ and cluster $\{a_1, a_2\}$ is represented if and only if $n \geq 2$.

Corollary 5.3. *The number $|minimal(\mathcal{C}(N))|$ of minimal clusters represented in an optimal network N may scale quadratically in the number of taxa $|\mathcal{X}|$.*

Proof. Combine Thm. 5.2 with Thm. 6.1 in Chapter 6. □

Conjecture 5.4. *Let $\mathcal{C}$ be a set of clusters for which an optimal level-k collnet N exists ($k \geq 1$). Then, the following inequalities give an upper and lower bound on the number of minimal clusters that N represents.*

$$r(N) + 1 \leq |\{C \in \mathcal{C}(N) : |C| = 2\}| = |minimal(\mathcal{C}(N))| \leq 2r(N)^2 + r(N) + 1 \quad (5.2)$$

5.3. Triangles in $IG(\mathcal{C})$

From this section onwards, some elementary substructures of $IG(\mathcal{C})$ are studied. Firstly, we consider the presence of a triangle in the incompatibility graph. In practice, this will often be the case (see e.g. Table 8.3), however, this result can be used to derive higher valued lower bounds on $r(G)$ for more advanced substructures of a connected component G in $IG(\mathcal{C})$.

Theorem 5.5. *Let $\mathcal{C}$ be a set of clusters. If there is a triangle in $IG(\mathcal{C})$, then at least two reticulations are needed in a network which represents the clusters in $\mathcal{C}$.*

Proof. Given a set of clusters $\mathcal{C}$, assume that there is a triangle in $IG(\mathcal{C})$. Let the vertices, i.e. the clusters, of the triangle be named C_i, for $i = 1, 2, 3$. Assume there

exists a (sub)network N which contains at most one reticulation and represents the clusters C_1 and C_2. Observe that it should contain at least one reticulation, since (C_1, C_2) is in $IG(\mathcal{C})$. Therefore, assume that the network contains exactly one reticulation. We show that N cannot contain only one reticulation if it should represent cluster C_3 too.

Observe the fundamental structure of a 1-reticulation network as illustrated in Fig. 3.5. Its leaves may be regarded as ST-sets. W.l.o.g., w.m.a. that cluster C_1 is a superset of a_1 and x (see the figure). Say that it is a superset of x and a_1 up to and including a_i, for some $1 \leq i \leq m$. Then cluster C_2 (which is incompatible with C_1) is either a superset of a_i and a_{i+1}, if $i < m$, and not of x (call it: cluster C_2 is on the a-side), or it is a superset of x and b_1 and not of any a_j $(j = 1, \ldots, m)$ (say cluster C_2 is on the b-side).

Firstly, consider the *case that C_2 is on the a-side*, where C_2 is a superset of $a_1, \ldots, a_i$, with $i < m$, but not of x. Observe that cluster C_3 should have a non-empty intersection with both clusters C_1 and C_2, since both edges (C_1, C_3) and (C_2, C_3) are in $IG(\mathcal{C})$. This implies that cluster C_3 is a strict superset of a_i. Observe that any strict superset of a_i is that of all a_j with $j < i$ too and therefore, so is cluster C_3. Besides, it should contain at least one taxon which is not in cluster C_1 and hence, it should be a superset of some a_j, with $j > i$. However, then we get that cluster C_3 is a superset of C_2, which contradicts the fact that (C_2, C_3) is in $IG(\mathcal{C})$. Hence, this case is impossible.

It remains to consider the *case that C_2 is on the b-side* and is a superset of x and b_1 up to and including b_j, for some $1 \leq j \leq n$. Again, cluster C_3 has overlap with both clusters C_1 and C_2 and therefore, it should is a strict superset of x. C_3 is not a superset nor a subset of C_1, so C_3 may not contain any set a_j $(1 \leq j \leq m)$. Similarly, C_3 may not contain any set b_j $(1 \leq j \leq n)$. We conclude that this case is impossible too.

All in all, no network comprising exactly one reticulation can represent three pairwise incompatible clusters. The result follows. □

This result might open a way to analyse other substructures which contain multiple triangles. For example, we conjecture that, if there is an '8'-like substructure in

some connected component G in $IG(\mathcal{C})$, comprising two partly overlapping triangles, then $r(G) \geq 3$.

Conjecture 5.6. *Let $\mathcal{C}$ be a set of clusters. If some connected component G in $IG(\mathcal{C})$ contains an '8' as induced subgraph, i.e. G comprises two triangles, having exactly one overlapping node (as shown in Fig. 5.2), then $r(G) \geq 3$.*

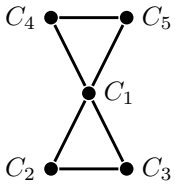


Figure 5.2: This figure illustrates an '8' as induced subgraph in $IG(\mathcal{C})$, comprising two triangles, which have exactly one overlapping node (i.e. cluster).

Example 5.7. This example cannot prove the validity of Conj. 5.6, however, it may give some insights. Consider the four trees $\mathcal{T}$ in Fig. 3.1. The minimal clusters in $Cl(\mathcal{T})$ are $12, 13, 15, 34, 56, 58$ and 67 (where again, e.g. 12 denotes cluster $\{1, 2\}$). Fig. 5.3 shows the corresponding incompatibility graph.

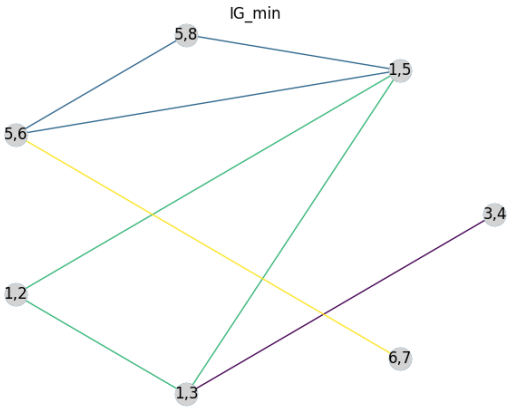


Figure 5.3: This figure shows the incompatibility graph on the minimal clusters in $Cl(\mathcal{T})$ from the trees $\mathcal{T}$ shown in Fig. 3.1. Different edge colours indicate different intersections $C_1 \cap C_2$ of its endpoints, i.e. clusters C_1 and C_2.

Observe that it contains an '8'. Therefore, the conjecture tells that at least 3

reticulations are required in a network which represents the clusters in $Cl(\mathcal{T})$. In this case, this bound is strict, since $r(N) = 3$ for an optimal network N.

5.4. Cycles in $IG(\mathcal{C})$

The second substructure of $IG(\mathcal{C})$ we investigate, is a cycle. The following result only holds for a cycle in $IG(minimal(\mathcal{C}))$; observe that there could be a 4-cycle in some connected component $G \in IG(\mathcal{C})$, while $r(G) = 1$, namely if e.g. $\{a_1a_2x, a_1x, b_1x, b_1b_2x\} \subseteq \mathcal{C}$.

Theorem 5.8. *Let $\mathcal{C}$ be a set of clusters. Let G be a connected component in $IG(minimal(\mathcal{C}))$. If there is a cycle in G, then $r(G) \geq 2$.*

Proof. Let $\mathcal{C}$ and G be as required for the theorem. Consider a general 1-reticulation network as depicted in Fig. 3.5 and regard the leaves as MST-sets. The minimal clusters therein, are $\{a_1, a_2\}, \{a_1, x\}, \{x, b_1\}$ and $\{b_1, b_2\}$. The corresponding incompatibility graph $IG(minimal(V(G)))$ thereon, is shown in Fig. 5.4. Since it does not contain any cycle, we conclude that the presence of a cycle would imply that $r(G)$ cannot be one (and trivially not zero), so it must be at least two then. □

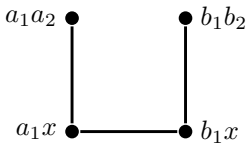


Figure 5.4: Consider a general level-1 network as illustrated in Fig. 3.5 and regard the leaves as (collapsed) MST-sets. This figure shows the corresponding incompatibility graph on the minimal clusters therein. Note that a cluster like $\{a_1, x\}$ is represented as a_1x.

5.5. Star-Like Substructures in $IG(\mathcal{C})$

The following elementary substructure of $IG(\mathcal{C})$ we study, is a star-like substructure. We consider one node and its neighbours in $IG(\mathcal{C})$, whether there are edges between the neighbours or not. For several conditions on such a substructure of some connected component G in $IG(\mathcal{C})$, we study its implication on $r(G)$.

Theorem 5.9. *Let $\mathcal{C}$ be a set of clusters. Let C_0 be a node in $IG(\mathcal{C})$ having $n \geq 2$ neighbours. Call the neighbours $C_1, \ldots, C_n$. Let H be the induced subgraph of $IG(\mathcal{C})$*

on these $n+1$ nodes. Suppose that for any three clusters C_0, C_i and C_j, with $1 \leq i < j \leq n$, it holds that $C_0 \cap C_i \cap C_j = \emptyset$ (i.e. all intersections $C_0 \cap C_i$ and $C_0 \cap C_j$ are disjoint). Then, the following holds.

$$r(H) \geq n - \mathbf{1}_{C_0 = \bigcup_{i=1,\ldots,n}(C_0 \cap C_i)} \tag{5.3}$$

Proof. Given some set of clusters $\mathcal{C}$, let H and $C_0, \ldots, C_n$ be as described in the theorem. Let us call the intersections of two clusters $I_i := C_0 \cap C_i$ for $i = 1, \ldots, n$. Besides, suppose that $C_0 = \bigcup_{i=1,\ldots,n} I_i$ (*) holds.

Firstly, we show that $r(N) \geq n - 1$. Let N be an optimal collnet representing the clusters in $V(H)$. Observe that all sets I_i (for all $i = 1, \ldots, n$) are MST-sets and therefore, each one hangs below a cut-edge. Since N represents cluster C_0, there is an induced subtree on $I_1 \cup \cdots \cup I_n = C_0$ in N. Similarly, there is an induced subtree on C_i $(i = 1, \ldots, n)$ in N. For each but one of the n sets I_i $(i = 1, \ldots, n)$, N should contain a reticulation to let it be a subset of either cluster C_0 or C_i (for your imagination, you may consider Ex. 5.10 below). This requires at least $n - 1$ reticulations.

Now, assume that (*) does not hold. We have to show that $r(N) \geq n$. Again, let N be a collnet representing the clusters in $V(H)$. Define the MST-set $I_{n+1} := C_0 \backslash (I_1 \cup \cdots \cup I_n)$. Similar as before, each but one of the $n+1$ sets I_i should be switched to be a subset of either cluster C_0 or C_i, for all $i = 1, \ldots, n+1$. This requires at least n reticulations. □

Example 5.10. Let $\mathcal{C} = \{abc, ad, be, cf\}$. The corresponding incompatibility graph H is illustrated in Fig. 5.5. Node $C_0 := abc$ has three neighbours: $C_1 := ad$, $C_2 := be$ and $C_3 := cf$. Let $I_i := C_0 \cap C_i$, for $i = 1, 2, 3$. Then, $C_0 = \bigcup_{i=1,2,3} I_i$. Furthermore, all intersections I_i $(i = 1, 2, 3)$ are disjoint. Therefore, $r(H) \geq 3 - 1 = 2$ by Thm. 5.9. An optimal network representing $\mathcal{C}$ is depicted in Fig. 5.5b, and comprises exactly two reticulations. Observe its similarity with the network shown in Fig. 6.1 in Chapter 6. Reticulation t_1 switches taxon a to be either in cluster $C_0 = abc$ or in cluster $C_1 = ad$. Similarly, reticulation t_2 switches taxon b to be either in cluster C_0 or C_2. One taxon in cluster C_0, which is taxon c in this case, does not need a

reticulation to let it be part of either cluster C_0 or C_3.

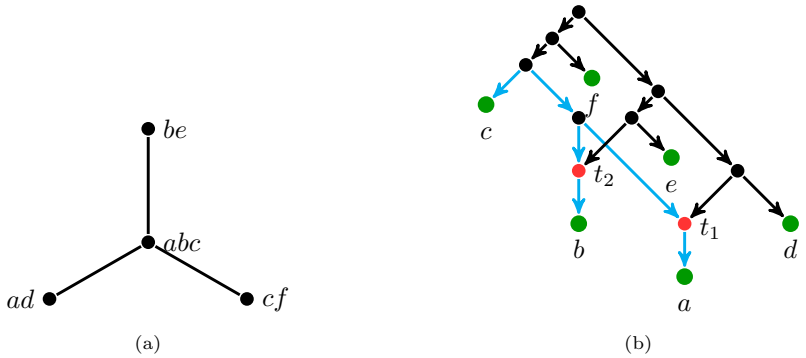


Figure 5.5: The incompatibility graph is shown for the set of clusters $\mathcal{C} = \{abc, ad, be, cf\}$ (a). Any network representing these clusters has reticulation number ≥ 2 by Thm. 5.9. One optimal network that represents $\mathcal{C}$ is shown in subfigure (b). The cyan coloured subnetwork is the induced subtree on $C_0 = abc$.

Based on this result, some conjectures are proposed below. Given that $\mathcal{C}$ is a set of clusters, let $Nb(C)$ denote the set of neighbours of cluster C in $V(IG(\mathcal{C}))$.

Conjecture 5.11 (Intersections LB)**.** *Let $\mathcal{C}$ be a set of clusters. Let G be a connected component in $IG(\mathcal{C})$. Then the following inequality gives a lower bound on the number of reticulations $r(G)$.*

$$\mathcal{I}(C) := \{C \cap C_j | \forall C_j \in Nb(C)\} \tag{5.4}$$

$$\mathcal{D}(C) := \{C' \in Nb(C) | C \cap C' \in \mathcal{I}(C)\} \tag{5.5}$$

$$r(G) \geq \max_{C \in V(G)} \left[|minimal(\mathcal{I}(C))| - \mathbf{1}_{C = \bigcup_{C' \in \mathcal{D}(C)} (C \cap C')} \right] \tag{5.6}$$

Example 5.10 (Continued)**.** Let G be $IG(\mathcal{C})$ as shown in Fig. 5.5a and let cluster $C_0 = abc$ again. Then the sets $\mathcal{I}(C_0)$ and $\mathcal{D}(C_0)$ are as follows.

$$\mathcal{I}(C_0) := \{a, b, c\} \tag{5.7}$$

$$\mathcal{D}(C_0) := \{ad, be, cf\} \tag{5.8}$$

Define $m(C) := |minimal(\mathcal{I}(C))| - \mathbf{1}_{C = \bigcup_{C' \in \mathcal{D}(C)} (C \cap C')}$, for any cluster $C \in V(G)$. It follows that $m(C_0) = 3 - 1 = 2$ and $m(ad) = 1 - 0 = 1 = m(bc) = m(cf)$.

Therefore, a lower bound on $r(G)$ is two by Conj. 5.11:

$$r(G) \geq \max_{C \in V(G)} \left[|minimal(\mathcal{I}(C))| - \mathbf{1}_{C = \bigcup_{C' \in \mathcal{D}(C)} (C \cap C')} \right] = \max(2,1,1,1) = 2 \tag{5.9}$$

Observe that Conj. 5.11 is true if $minimal(\mathcal{D}(C_0))$ is a set of disjoint clusters, for each cluster C_0 in $V(G)$, since Thm. 5.9 can be applied then (where $n = |minimal(\mathcal{D}(C_0))|$). Furthermore, the set of intersections between pairs of clusters can be studied for all pairs in G too. This led us to the following conjecture.

Conjecture 5.12 (Global Intersections LB)**.** *Let $\mathcal{C}$ be a set of clusters. Let G be a connected component in $IG(\mathcal{C})$. The following inequality gives a lower bound on $r(G)$.*

$$r(G) \geq \left\lceil \frac{1}{3} |\{C_i \cap C_j | \forall C_i, C_j \in V(G), i \neq j\}| \right\rceil \tag{5.10}$$

Observe that strict equality holds in the case that each connected component in $IG(\mathcal{C})$ is as illustrated in Fig. 5.4. Now, instead of considering the sets $C_0 \cap C_j$, let us consider the difference sets $C_0 \backslash C_j$. The minimal clusters among these can be used to define a lower bound on $r(g)$.

Conjecture 5.13 (Differences LB)**.** *Let $\mathcal{C}$ be a set of clusters and let G be a connected component in $IG(\mathcal{C})$. Then, a lower bound on the number of reticulations $r(G)$ in the corresponding biconnected component is as follows.*

$$\mathcal{D}(C) := \{C_j \backslash C | \forall C_j \in Nb(C)\} \tag{5.11}$$

$$r(G) \geq \max_{C \in V(G)} |minimal(\mathcal{D}(C))| - 1 \tag{5.12}$$

Example 5.14. Note that this example does not show the validity of Conj. 5.13, however, it may support it and give us some insights. Again, consider the four trees $\mathcal{T}$ in Fig. 3.1. The corresponding incompatibility graph $IG(minimal(Cl(\mathcal{T})))$ on the minimal clusters in $Cl(\mathcal{T})$ is illustrated in Fig. 5.3. Observe that node $C_0 = 15$ is connected to the nodes $12, 13, 56$ and 58. This implies that $\{C \backslash C_0 | \forall C \in Nb(C_0)\} = \{2, 3, 6, 8\}$, which has size four. Therefore, by Conj. 5.13, at least three reticulations are required in a network to represent the clusters in $Cl(\mathcal{T})$. In this example, as we have seen before, this lower bound is strict.

Similar as for the set of intersections, these difference sets $C_0 \backslash C_j$ can be studied for all pairs of clusters in G. This raised the following conjecture.

Conjecture 5.15 (Global Differences LB)**.** *Let $\mathcal{C}$ be a set of clusters. Let G be a connected component in $IG(\mathcal{C})$. The following inequality gives a lower bound on $r(G)$.*

$$r(G) \geq \left\lceil \frac{1}{5} \left| minimal \{C_i \backslash C_j | \forall C_i, C_j \in \mathcal{C}, i \neq j\} \right| \right\rceil \tag{5.13}$$

The factor 1/5 has been chosen based on some theoretical insights. Consider e.g. Fig. 3.5, where $a_1a_2 \backslash a_1x = a_2$, $a_1x \backslash a_1a_2 = x$, $a_1x \backslash b_1x = a_1$, $b_1x \backslash a_1x = b_1$ and $b_1b_2 \backslash b_1x = b_2$. Therefore, $minimal\{C_i \backslash C_j | \forall C_i, C_j \in \mathcal{C}, i \neq j\} = \{a_1, a_2, x, b_1, b_2\}$, whose cardinality is five.

Surely, more (advanced) results can be found involving a star-like substructures of $IG(\mathcal{C})$. However, we continue to address a related substructure now, namely n-cliques.

5.6. n-Cliques in $IG(\mathcal{C})$

Now, imagine that some connected component G in $IG(\mathcal{C})$ contains an n-clique K_n as induced subgraph, for some $n \geq 2$. What does it say about $r(G)$?

Conjecture 5.16 (Clique LB)**.** *Let $\mathcal{C}$ be a set of clusters. If there is an n-clique in some connected component G of $IG(\mathcal{C})$, then $r(G) \geq n - 1$.*

Note that we have shown this conjecture to be true for $n \leq 3$ (see Thm. 5.5 for $n = 3$). Furthermore, observe that each node C_0 in $V(K_n)$ is the centre node of some star-like substructure as discussed in the previous section, comprising $n - 1$ leaves (which may give an indication of the truth of Conj. 5.16). Below, a special case of an n-clique is addressed.

Theorem 5.17. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. Let G be a connected component in $IG(\mathcal{C})$. Assume that it contains an n-clique K_n for some $n \geq 2$, such that for all $C_i, C_j \in V(K_n)$, $C_i \cap C_j = I$, for all $1 \leq i < j \leq n$, where $I \subset \mathcal{X}$. Then $r(G) \geq n - 1$.*

Proof. Let $\mathcal{C}$ and G be as required. Let N be an optimal collnet representing the clusters in $V(G)$. Observe that I is an MST-set and hence, it is below a cut-edge. Furthermore, I should be switched (using reticulations) to be part of either cluster C_1 or C_2 or ... or C_n. Since no two different clusters C_i and C_j (for $1 \leq i < j \leq n$) have other overlapping taxa than those in I, this requires $n - 1$ reticulations. □

Theorem 5.18. *Let $\mathcal{C}$ be a set of clusters. Let G be a connected component*

in $IG(\mathcal{C})$. *Let* G' *be the corresponding connected component in* $IG(minimal(\mathcal{C}))$. *Then,* $r(G') \leq r(G)$.

Proof. This follows from the fact that $IG(minimal(\mathcal{C}))$ is an induced subgraph of $IG(\mathcal{C})$ and hence, G' is that of G. □

Now, given some connected component G in $IG(\mathcal{C})$, we have studied some substructures of it and several of their implications on $r(G)$. As stated earlier, these findings can be used to speed up Alg. 6. In particular, if $\mathcal{C}$ is a set of clusters and $r(G)$ seems to be at least $k_{min} \geq 1$, then STCass(k) does not have to be evaluated for $k < k_{min}$ in order to find an optimal network representing $\mathcal{C}$.

Although the following may not be practically interesting (except for debugging purposes), one might wonder: what would be an upper bound on the reticulation number? This is addressed in the next chapter.

6

Upper Bounds on the Minimum Reticulation Number

In Chapter 5, lower bounds for $r(G)$ were derived, given some connected component G in $IG(\mathcal{C})$. Although it may not have practical interests (except for e.g. debugging purposes), we will study some upper bounds on $r(G)$ in this section.

Theorem 6.1. *In an optimal network N on taxa $\mathcal{X}$, $r(N) \leq |\mathcal{X}| - 1$.*

Proof. Consider the network which is shown in Fig. 6.1, on the taxa $\mathcal{X} = \{x_1, \ldots, x_n\}$ ($n \geq 3$). Observe that it represents every cluster $C \subseteq \mathcal{X}$. The number of taxa is n and the number of reticulations is $n - 1$. In an optimal network, no reticulation is redundant and therefore, no optimal network will contain more than $n - 1$ reticulations.

□

In general, tighter upper bounds can be found on $r(G)$. One example is the

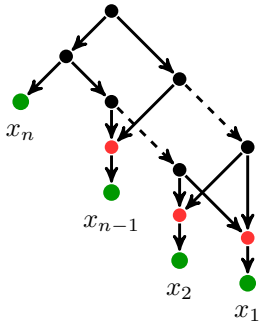


Figure 6.1: A collnet on $\mathcal{X} = \{x_1, \ldots, x_n\}$ $(n \geq 3)$ is shown which represents all clusters $C \subseteq \mathcal{X}$.

following, which we conjecture to be true and can be determined algorithmically easily (mostly within a second).

Conjecture 6.2. *Let $\mathcal{C}$ be a set of clusters on taxa $\mathcal{X}$. For the moment, let $\mathcal{C}_x := minimal(C \in \mathcal{C} | \mathcal{X}(N') : x \in C)$. Then the reticulation number $r(\mathcal{C})$ is at most the following.*

$$r(\mathcal{C}) \leq \sum_{x \in \mathcal{X}} |\mathcal{C}_x| - M_1 - M_2, \tag{6.1}$$

where M_1 and M_2 are defined as follows.

$$M_1 = \max_{x \in \mathcal{X}}(|\mathcal{C}_x|) \tag{6.2}$$

$$x^* = \arg\max_{x \in \mathcal{X}}(|\mathcal{C}_x|) \tag{6.3}$$

$$M_2 = \max_{y \in \mathcal{X} \setminus \{x^*\}}(|\mathcal{C}_y|) \tag{6.4}$$

A motivation for this conjecture to be true, is as follows. Note that for the moment, we use the definition of $\mathcal{C}_x$ as given above, containing only minimal clusters. Let leaf x^* be $\arg\max_{x \in \mathcal{X}}(|\mathcal{C}_x|)$ and let $y^* = \arg\max_{y \in \mathcal{X} \setminus x^*}(|\mathcal{C}_y|)$. Let N be the tree on these leaves (i.e. a *cherry*) and add some dummy root δ and connect it to the root r of the tree by adding edge (δ, r). Let the other leaves, i.e. those in $\mathcal{X} \setminus \{x^*, y^*\}$, be named $x_1, x_2, \ldots, x_{n-2}$. Then, for each leaf x_i $(i = 1, \ldots, n-2)$ we add to the network, at most $|\mathcal{C}_{x_i}| - 1$ binary reticulations are needed to let the resulting network represent all clusters containing leaf x_i, plus at most one additional binary reticulation (e.g. to the root) in order to let the network represent the other clusters (see e.g. Thm. 4.15). In total, this gives at most $|C_{x_i}|$ additional reticulations per leaf x_i we add. Note that this is assumed to be possible due to Conj. 4.6. Summing these values gives Eq. 6.1. Besides, empirical support is given by e.g. the results in the last two columns of Table 8.3.

Lastly, suppose that we are given a set $\mathcal{T}$ of trees. As stated in the introduction, another upper bound on $r(Cl(\mathcal{T}))$ is given by the *hybridization number*.

7

Implementation

STCass has been implemented in Python. It has been done Python, since it is a widely used language, it enables rapid prototyping and besides, it is well documented. Some algorithmic techniques that are used, are discussed below. Then, it will be shown which rules we have implemented in STCass. Besides, we introduce MSTCass, an improved version of STCass, which runs faster. This has been implemented in Python too. For comparison, Cass has been implemented in Java, using the original source code provided by L.L.J. Van Iersel.

7.1. Techniques

(M)STCass has been implemented using several techniques. Firstly, a *stack* (queue) is used to store problem instances in any stage of the algorithm (be it in the inward removal phase or in the outward building phase). Secondly, we tested both a LIFO queue and a FIFO priority queue, where in the last case, the further the stage was of the problem instance, e.g. the more reticulations it has in the building phase, the more priority was given to handle that instance. Both cases result in a depth-first search. The **LIFO queue** performed the best of the two, i.e. this gave the fastest algorithm, and the results in the next chapter are based on this. Thirdly, *multicore processing* has been implemented. Four processes were used. Unfortunately, due to an unknown reason, this led to an increase of the runtime almost always. Hence, a working multiprocessing implementation is still in the future and the results are based on a single process algorithm.

The original Cass algorithm uses a *stack* too in order to store the problem instances in every iteration. It uses a FIFO queue (without priority) and a single process. Note that in general, a Python implementation runs slower than one in Java.

7.2. STCass versus MSTCass

The optimal algorithms STCass (see Section 3.4) and MSTCass (see Section 3.8) have been implemented with several optimizations. Some optimizations are based on theorems and others are based on several conjectures made throughout this work. Table 7.1 gives an overview of which heuristics and conjectures have been applied.

The source code is available on request. The main results are summarized in the following chapter.

Rule	**STCass**	**MSTCass**
MST-sets only	-	V
Heuristic: MST priorities	-	V
Lower bound on $k^{(1)}$	V	V
$\mathcal{C}_x$: minimal clusters only$^{(2)}$	V	V
Restricted Hang-edges$^{(3)}$	V	V
Minimal Cluster Hang-edge Sets$^{(4)}$	V	V
Smallest Hang-Edge Combinations$^{(5)}$	-	V

Table 7.1: Several algorithms have been implemented, each using different solving strategies. *: Conj. 5.11 and Conj. 5.13 are included in determining a lower bound on k. (2): Motivated by Conj. 4.6, only the minimal clusters are considered. (3): Only use hang-edges as defined in Conj. 4.10. (4): See Thm. 4.12. (5): See Conj. 4.19.

8

Results

The performance of both the algorithms STCass and MSTCass have been tested on each combination of two or more trees from those of the *Poaceae* data set. "This dataset consists of six phylogenetic trees of grasses of the *Poaceae* family, originally published by the Grass Phylogeny Working Group (2001) and reanalysed in [20]. The phylogenetic trees are based on sequences from six different gene loci, namely ITS, ndhF, phyB, rbcL, rpoC and waxy, and contain 47, 65, 40, 37, 34 and 19 taxa, respectively."[12] For each combination, the trees were restricted to the taxa they have in common. On top of these 57 instances, the four trees from Fig. 3.1 have been used as input, which instance has been called 'Ellusiveness Fig. 9'.

The performance of the algorithms on these data are presented in the section below. After that, it will be shown how well the proposed lower bounds and upper bound in Conj. 6.2 work in practice (on the given instances).

8.1. The Performance of STCass and MSTCass

The performance of both the algorithms STCass and MSTCass have been tested on 58 problem instances (as described above). For comparison, we executed the Cass algorithm on the same instances. The algorithms have been run until one solution was found. Per algorithm, the runtime t is logged (in seconds), the level k of the network that was found and the number r of reticulations in that network, per input instance.

The execution time has been limited to five minutes. On a timeout, the listed

value of k is the level for which the running algorithm was searching a network at that moment. The results are summarized in Table 8.2.

Some specifications of the computer that was used, are the following: the computer contained an Intel Core i7 4700MQ CPU @2.40 GHz, 16 GB RAM, 120 GB SSD plus some HDD and a Debian-based operating system.

8.1.1. Runtime

Regarding the average runtimes per algorithm, these are listed in Table 8.1, per level k of the output network, for each of the three algorithms Cass, STCass and MSTCass. Note that the results for which any of three algorithms did not find any network, have not been used for computing these averages.

From Table 8.1, it is clear that the average runtime rapidly increases for increasing values of k, especially for Cass. Furthermore, Cass runs the fastest of the three. MSTCass was almost 13 times faster than STCass. Besides, the runtimes of MSTCass is comparable to that of Cass for such small instances as have been used in the tests.

Observe that the runtime of MSTCass is less than that of Cass when the optimal level k^* is 5 or 6. This may give an indication that for $k^* \gg 4$, MSTCass will be much faster than Cass, which is left to investigate in the future.

Level k	**#Instances**	**Avg. Runtime per Algorithm [s]**		
		Cass	**STCass**	**MSTCass**
0	8	0.00	0.01	0.01
2	10	0.02	0.08	0.05
3	5	0.06	0.86	0.18
4	12	0.44	45.53	2.48
5	3	1.94	25.10	1.08
6	1	19.03	26.08	17.15
Weighted Average		**0.8**	**16.7**	**1.3**

Table 8.1: For each of the algorithms Cass, STCass and MSTCass, this table presents the average runtimes in seconds per level k of the output network. Results for instances for which the algorithms could not find any network (within five minutes), are not included in this analysis.

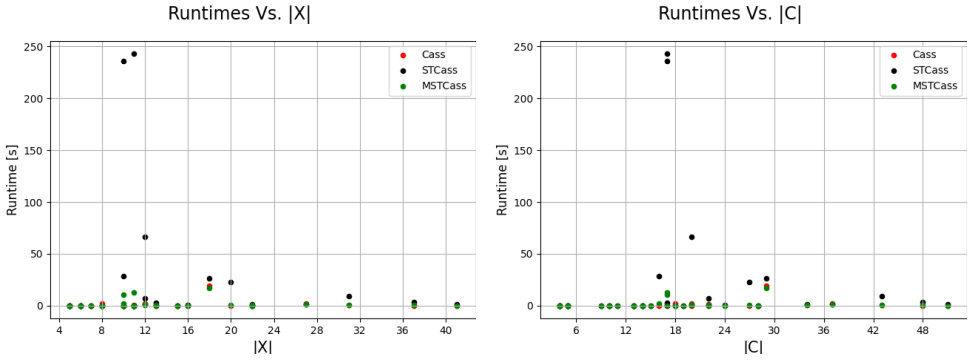
What could cause these differences? Cass is for these relatively small sized instances the fastest algorithm, which at least partly depends on it implementation using the language Java, which is faster than Python. Besides, STCass needs more time than Cass, however, MSTCass is much faster than STCass. It would be interesting to implement both Cass and MSTCass in Java in the future. In general, MSTCass is faster than STCass, which in short, is caused by the differences listed

in Table 7.1. In particular, firstly, the search tree becomes smaller by considering MST-sets only, as MSTCass does, instead of all ST-sets (in STCass). Secondly, a heuristic is applied to determine the order of selecting MST-sets in the inward removal phase in MSTCass, while that order depends on the input definition (order) in STCass. Thirdly, the hang-edge combinations to consider are filtered in MSTCass, while not in STCass.

What other conclusions can be drawn from the results? One may observe in Fig. 8.2 that, when STCass runs longer than 35s, then likely, the level is ≥ 5; and when it runs longer than 70s, then the level is likely ≥ 6. Similarly, when MSTCass runs longer than 3s, the level is likely ≥ 5. Note that many more test data should be used to make stronger evidence for such time bounds. This might be used to develop a heuristic which starts searching for a higher level network as soon as such time bounds are exceeded.

Moreover, there are some remarkable differences in the runtimes for the different algorithms for some tree combinations. STCass seems to need much more time than the other algorithms for the combinations ndhF/rpoC (≥ 300 s versus ≤ 2 s), ndhF/phyB/ITS (≥ 300 s versus ≤ 4 s), ndhF/phyB/rpoC (300 s versus ≤ 12 s), ndhF/rpoC/waxy/ITS (243 s versus ≤ 13 s) and ndhF/rbcL/rpoC/waxy/ITS (236 s versus ≤ 11 s). This may be due to the increased complexity of STCass over Cass and MSTCass and besides, the selection of which ST-set to remove in the inward phase is different for the different algorithms and might need improvement.

On the other hand, STCass is very fast (9 s) in finding a solution for the tree combination ndhF/rbcL/rpoC, while the other algorithms could not even find any solution within five minutes. This can only be caused by the rapid selection of the correct ST-sets to remove in the inward phase and by selecting correct combinations of hang-edges in the outward phase of the algorithm.



(a) (b)

Figure 8.1: **(a)** A scatter plot of the runtimes versus the number of taxa $|\mathcal{X}|$. There seems to be a very week relation between the two variables, even for Cass. **(b)** A similar scatter plot, where the runtimes are plotted versus the number of clusters $|\mathcal{C}|$. Here too, there is a very week relation between the two variables.

Does the runtime depend on the number of taxa $|\mathcal{X}|$ too? Consider Fig. 8.1a. It shows that there is a very weak relation between the runtimes and the number of taxa $|\mathcal{X}|$ of the input trees. This is clear from the (linear) Pearson correlation coefficient of the runtimes versus $|\mathcal{X}|$ too, which are 0.11, -0.06 and 0.04 for Cass, STCass and MSTCass, respectively.

Fig. 8.1b shows a similar scatter plot, where the runtime is shown versus the number of clusters $|\mathcal{C}|$ per instance. Like in the previous case, there is a very weak (linear) correlation between the two variables.

8.1.2. Optimality

As shown before, both the algorithms STCass (see Thm. 3.4) and MSTCass (see Cor. 3.11) are optimal, whereas Cass is not. This is visible in the results (see Table 8.2) for the special instance 'Elusiveness Fig. 9' of the trees $\mathcal{T}$ from Fig. 3.1. Cass produces a network of level four, while the optimal level is three, which STCass and MSTCass do find.

Data			Cass			STCass			MSTCass		
Trees	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	$t[s]$	k	r	$t[s]$	k	r	$t[s]$	k	r
Elusiveness Fig. 9	18	7	1.921	4	4	0.178	3	3	0.090	3	3
ndhF ITS	70	46	300.002	≥ 6	≥ 13	300.101	≥ 6	≥ 6	300.103	≥ 5	≥ 5
ndhF phyB	51	40	0.273	4	8	1.079	4	8	0.284	4	8
ndhF rbcL	48	36	0.101	3	8	3.787	3	8	0.499	3	8
ndhF rpoC	47	34	1.961	5	9	300.103	≥ 5	≥ 5	1.307	5	9
ndhF waxy	27	19	0.262	4	6	22.516	4	6	0.817	4	6
phyB ITS	43	30	0.399	4	8	9.370	4	8	0.589	4	8
phyB rbcL	28	21	0.026	2	4	0.134	2	4	0.104	2	4
phyB rpoC	28	21	0.081	3	4	0.183	3	4	0.110	3	4
phyB waxy	18	14	0.021	2	3	0.043	2	3	0.042	2	3
rbcL ITS	46	29	300.004	≥ 5	≥ 6	300.102	≥ 6	≥ 6	300.103	≥ 6	≥ 6
rbcL rpoC	37	26	1.955	5	7	1.458	5	7	1.300	5	7
rbcL waxy	17	12	0.160	4	4	2.784	4	4	0.290	4	4
rpoC ITS	48	31	300.002	≥ 6	≥ 8	300.103	≥ 5	≥ 5	300.102	≥ 6	≥ 6
rpoC waxy	13	10	0.031	2	2	0.039	2	2	0.028	2	2
waxy ITS	22	15	0.165	4	5	0.874	4	5	0.184	4	5
ndhF phyB ITS	48	30	1.526	5	10	300.100	≥ 5	≥ 5	4.002	5	10
ndhF phyB rbcL	34	21	0.413	4	7	1.063	4	7	0.864	4	7
ndhF phyB rpoC	33	21	11.405	4	6	300.103	≥ 4	≥ 4	5.546	4	6
ndhF phyB waxy	19	14	0.027	2	4	0.175	2	4	0.133	2	4
ndhF rbcL ITS	50	28	300.003	≥ 5	≥ 5	300.102	≥ 6	≥ 6	300.101	≥ 5	≥ 5
ndhF rbcL rpoC	43	26	300.053	≥ 6	≥ 8	9.072	6	8	300.076	≥ 6	≥ 7
ndhF rbcL waxy	20	12	0.169	4	4	0.432	4	4	0.921	4	4
ndhF rpoC ITS	58	31	300.001	≥ 5	≥ 7	300.011	≥ 5	≥ 5	300.103	≥ 6	≥ 6
ndhF rpoC waxy	14	10	0.040	3	3	0.070	3	3	0.086	3	3
ndhF waxy ITS	26	15	300.009	≥ 5	≥ 7	300.103	≥ 4	≥ 4	9.742	5	7
phyB rbcL ITS	29	17	19.034	6	6	26.075	6	6	17.151	6	6
phyB rbcL rpoC	24	15	0.196	4	5	0.567	4	5	0.196	4	5
phyB rbcL waxy	10	7	0.021	2	2	0.030	2	2	0.029	2	2
phyB rpoC ITS	32	19	300.362	≥ 5	≥ 6	300.048	≥ 5	≥ 5	300.102	≥ 5	≥ 5
phyB rpoC waxy	5	5	0.000	0	0	0.008	0	0	0.009	0	0
phyB waxy ITS	14	10	0.022	2	3	0.051	2	3	0.039	2	3
rbcL rpoC ITS	48	24	300.004	≥ 5	≥ 5	300.035	≥ 6	≥ 6	300.065	≥ 6	≥ 6
rbcL rpoC waxy	13	9	0.043	3	3	0.145	3	3	0.128	3	3
rbcL waxy ITS	20	11	2.238	5	5	66.541	5	5	1.196	5	5
rpoC waxy ITS	16	10	0.767	4	4	300.022	≥ 4	≥ 4	9.876	4	4
ndhF phyB rbcL ITS	33	17	300.170	≥ 7	≥ 7	300.100	≥ 7	≥ 7	300.069	≥ 7	≥ 7
ndhF phyB rbcL rpoC	29	15	6.320	5	7	300.035	≥ 4	≥ 6	284.283	5	7
ndhF phyB rbcL waxy	11	7	0.019	2	2	0.059	2	2	0.034	2	2
ndhF phyB rpoC ITS	35	19	300.073	≥ 6	≥ 7	304.092	≥ 5	≥ 5	300.103	≥ 5	≥ 5
ndhF phyB rpoC waxy	5	5	0.000	0	0	0.008	0	0	0.015	0	0
ndhF phyB waxy ITS	15	10	0.030	2	4	0.218	2	4	0.069	2	4
ndhF rbcL rpoC ITS	52	24	300.003	≥ 5	≥ 5	300.100	≥ 6	≥ 6	300.103	≥ 6	≥ 6
ndhF rbcL rpoC waxy	14	9	0.045	3	3	0.121	3	3	0.078	3	3
ndhF rbcL waxy ITS	22	11	1.633	5	5	7.316	5	5	0.742	5	5
ndhF rpoC waxy ITS	17	10	0.750	4	4	243.162	4	4	13.101	4	4
phyB rbcL rpoC ITS	28	14	300.437	≥ 6	≥ 6	300.102	≥ 6	≥ 6	300.103	≥ 6	≥ 6
phyB rbcL rpoC waxy	4	4	0.000	0	0	0.010	0	0	0.008	0	0
phyB rbcL waxy ITS	9	6	0.020	2	2	0.033	2	2	0.027	2	2
phyB rpoC waxy ITS	5	5	0.000	0	0	0.011	0	0	0.008	0	0
rbcL rpoC waxy ITS	16	9	0.282	4	4	28.480	4	4	1.974	4	4
ndhF phyB rbcL rpoC ITS	31	14	300.024	≥ 6	≥ 6	300.104	≥ 6	≥ 6	300.103	≥ 6	≥ 6
ndhF phyB rbcL rpoC waxy	4	4	0.000	0	0	0.008	0	0	0.008	0	0
ndhF phyB rbcL waxy ITS	10	6	0.020	2	2	0.054	2	2	0.034	2	2
ndhF phyB rpoC waxy ITS	5	5	0.000	0	0	0.011	0	0	0.008	0	0
ndhF rbcL rpoC waxy ITS	17	9	0.267	4	4	235.832	4	4	10.492	4	4
phyB rbcL rpoC waxy ITS	4	4	0.000	0	0	0.018	0	0	0.008	0	0
ndhF phyB rbcL rpoC waxy ITS	4	4	0.000	0	0	0.013	0	0	0.008	0	0
Average	**18.3**	**13.6**	**0.8**	**2.7**	**3.5**	**16.7**	**2.6**	**3.4**	**1.3**	**2.6**	**3.4**

Table 8.2: For each combination $\mathcal{T}$ of trees from the *Poaceae* data set, the algorithms Cass, STCass and MSTCass have been applied to build a network which represents the clusters from the input trees $\mathcal{T}$. For each algorithm, the solving time t [s] is given, the level k and the reticulation number r of the output network, per instance $\mathcal{T}$. The sign '$\geq$' indicates that no solution has been found within five minutes (these data points are not included in the averages, which rows are marked dark gray). Each non-optimal level of the output network is marked red. If on a timeout, STCass or MSTCass were busy investigating a higher level network than Cass on a timeout, the corresponding results are marked green.

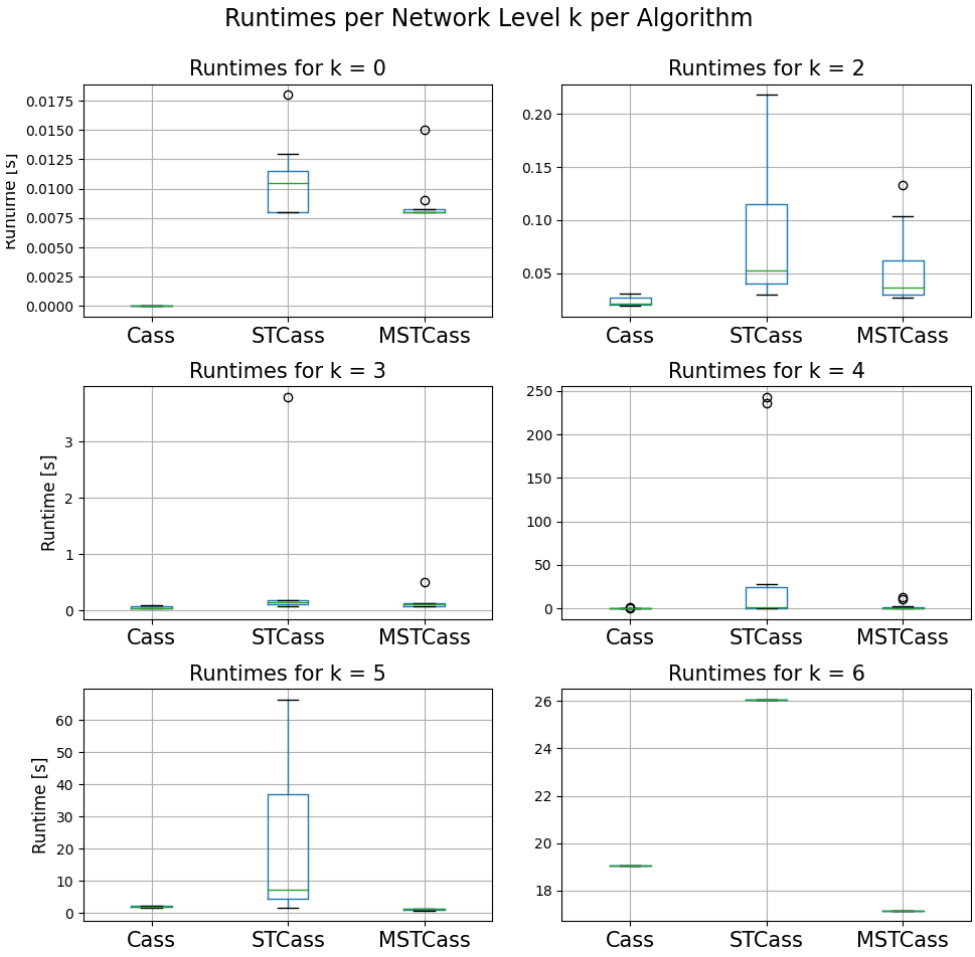


Figure 8.2: These box plots show the runtimes per level k of the output networks, per algorithm. Note that the results for which the algorithms did not find any network, have not been used in this analysis.

8.2. Lower and Upper Bounds on the Level and Reticulation Number

For each combination of trees $\mathcal{T}$, lower bounds (LBs) on the level $k = \ell(Cl(\mathcal{T}))$ and reticulation number $r = r(Cl(\mathcal{T}))$ have been determined using Conj. 5.11 (Intersection LB), Conj. 5.12 (Global Intersection LB), Conj. 5.13 (Differences LB), Conj. 5.15 (Global Differences LB) and Conj. 5.16 (Clique LB). Besides, upper bounds (UBs) on these numbers have been determined by applying Conj. 6.2 (UB). The results are summarized in Table 8.3. Those for the lower bounds are visualized in box plots in Fig. 8.3 and for the upper bound in Fig. 8.4.

Regarding the lower bounds, the first lower bound (Intersection LB) performs the best of the five, in general (see Fig. 8.3). Remarkably, none of the lower bounds give that the optimal level k^* is four or higher, if so. This indicates that there is much room for improvement to find better lower bounds.

Regarding the upper bound, its expected value seems to be approximately 2.5 times the optimal level, see Fig. 8.4. This leaves room for improvement too. On the other hand, the runtime for evaluating this bound has not been recorded, since it can be evaluated within a second (or at most some seconds), due to its simplicity.

Trees	Optimal		Int. LB		Gl. I. LB		Diffs. LB		Gl. D. LB		Clique LB		UB	
	k	r	k	r	k	r	k	r	k	r	k	r	k	r
Elusiveness Fig. 9	3	3	3	3	2	2	3	3	2	2	2	2	7	7
ndhF ITS	≥ 6	≥ 13	4	8	3	6	3	5	3	6	1	3	29	44
ndhF phyB	4	8	2	6	2	6	1	2	2	6	1	5	11	17
ndhF rbcL	3	8	2	5	1	4	1	2	2	5	1	4	9	20
ndhF rpoC	5	9	3	7	2	6	2	2	2	6	1	5	15	21
ndhF waxy	4	6	2	3	2	3	1	1	1	2	1	2	11	14
phyB ITS	4	8	3	6	2	4	2	3	2	4	1	3	13	23
phyB rbcL	2	4	2	3	1	2	1	2	2	3	1	2	7	12
phyB rpoC	3	4	2	3	2	3	1	2	2	3	1	2	9	12
phyB waxy	2	3	1	2	1	2	0	0	1	2	1	2	3	6
rbcL ITS	≥ 6	≥ 6	3	4	4	5	2	2	3	4	1	2	27	28
rbcL rpoC	5	7	3	5	3	5	2	3	2	4	1	3	15	19
rbcL waxy	4	4	2	2	2	2	1	1	1	1	1	1	9	9
rpoC ITS	≥ 6	≥ 8	3	4	3	4	2	2	3	4	1	2	29	32
rpoC waxy	2	2	2	2	1	1	1	1	1	1	1	1	5	5
waxy ITS	4	5	2	3	1	2	1	1	1	2	1	2	11	12
ndhF phyB ITS	5	10	3	7	2	4	2	3	2	4	2	5	17	28
ndhF phyB rbcL	4	7	2	4	2	3	1	2	2	4	2	3	10	19
ndhF phyB rpoC	4	6	3	5	3	4	2	3	2	3	2	4	10	14
ndhF phyB waxy	2	4	1	2	1	2	0	0	1	2	1	2	5	8
ndhF rbcL ITS	≥ 6	≥ 6	4	4	4	4	2	2	3	3	2	2	26	26
ndhF rbcL rpoC	6	8	3	5	4	6	2	4	2	4	2	4	19	25
ndhF rbcL waxy	4	4	3	3	2	2	1	1	2	2	2	2	10	10
ndhF rpoC ITS	≥ 6	≥ 7	5	6	4	5	4	5	3	4	2	3	29	33
ndhF rpoC waxy	3	3	2	2	1	1	1	1	1	1	1	1	6	6
ndhF waxy ITS	5	7	3	4	2	3	1	1	2	3	2	3	13	16
phyB rbcL ITS	6	6	3	3	3	3	2	2	2	2	1	1	15	15
phyB rbcL rpoC	4	5	3	4	2	3	1	2	2	3	2	3	9	12
phyB rbcL waxy	2	2	1	1	1	1	1	1	1	1	1	1	4	4
phyB rpoC ITS	≥ 5	≥ 6	4	5	2	3	2	2	2	3	2	3	17	18
phyB rpoC waxy	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phyB waxy ITS	2	3	1	2	1	2	0	0	1	2	1	2	3	6
rbcL rpoC ITS	≥ 6	≥ 6	5	5	5	5	3	3	3	3	2	2	22	22
rbcL rpoC waxy	3	3	2	2	1	1	1	1	1	1	1	1	6	6
rbcL waxy ITS	5	5	2	2	2	2	1	1	2	2	2	2	10	10
rpoC waxy ITS	4	4	2	2	2	2	1	1	1	1	2	2	9	9
ndhF phyB rbcL ITS	≥ 7	≥ 7	3	3	3	3	2	2	2	2	2	2	15	15
ndhF phyB rbcL rpoC	5	7	3	5	3	4	2	3	2	3	2	4	9	13
ndhF phyB rbcL waxy	2	2	2	2	1	1	1	1	1	1	2	2	5	5
ndhF phyB rpoC ITS	≥ 6	≥ 7	4	5	3	4	3	3	2	3	3	4	17	18
ndhF phyB rpoC waxy	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ndhF phyB waxy ITS	2	4	2	3	1	2	0	0	1	2	2	3	4	7
ndhF rbcL rpoC ITS	≥ 6	≥ 6	5	5	5	5	4	4	3	3	3	3	22	22
ndhF rbcL rpoC waxy	3	3	2	2	1	1	1	1	1	1	2	2	7	7
ndhF rbcL waxy ITS	5	5	3	3	2	2	1	1	2	2	2	2	10	10
ndhF rpoC waxy ITS	4	4	2	2	2	2	1	1	1	1	2	2	9	9
phyB rbcL rpoC ITS	≥ 6	≥ 6	4	4	3	3	2	2	2	2	2	2	12	12
phyB rbcL rpoC waxy	0	0	0	0	0	0	0	0	0	0	0	0	0	0
phyB rbcL waxy ITS	2	2	1	1	1	1	1	1	1	1	1	1	4	4
phyB rpoC waxy ITS	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rbcL rpoC waxy ITS	4	4	2	2	2	2	1	1	1	1	2	2	8	8
ndhF phyB rbcL rpoC ITS	≥ 6	≥ 6	4	4	3	3	3	3	2	2	3	3	12	12
ndhF phyB rbcL rpoC waxy	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ndhF phyB rbcL waxy ITS	2	2	2	2	1	1	1	1	1	1	2	2	5	5
ndhF phyB rpoC waxy ITS	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ndhF rbcL rpoC waxy ITS	4	4	2	2	2	2	1	1	1	1	3	3	8	8
phyB rbcL rpoC waxy ITS	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ndhF phyB rbcL rpoC waxy ITS	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Average	**3.0**	**4.0**	**1.8**	**2.7**	**1.5**	**2.2**	**1.0**	**1.2**	**1.2**	**2.0**	**1.3**	**2.0**	**7.4**	**9.6**

Table 8.3: For each combination of trees from the *Poaceae* data set, the optimal level k and reticulation number r are given, as well as the lower bounds determined by Conj. 5.11 (Int. LB), Conj. 5.12 (Gl. Int. LB), Conj. 5.13 (Differences LB), Conj. 5.15 (Gl. Differences LB) and Conj. 5.16 (Clique LB). The last two columns show upper bounds on these values, which are determined using Conj. 6.2 (UB). Coloured numbers indicate tight bounds. A '$\geq$' sign indicates that no solution has been found within five minutes (these data points are not included in the averages and are marked dark gray).

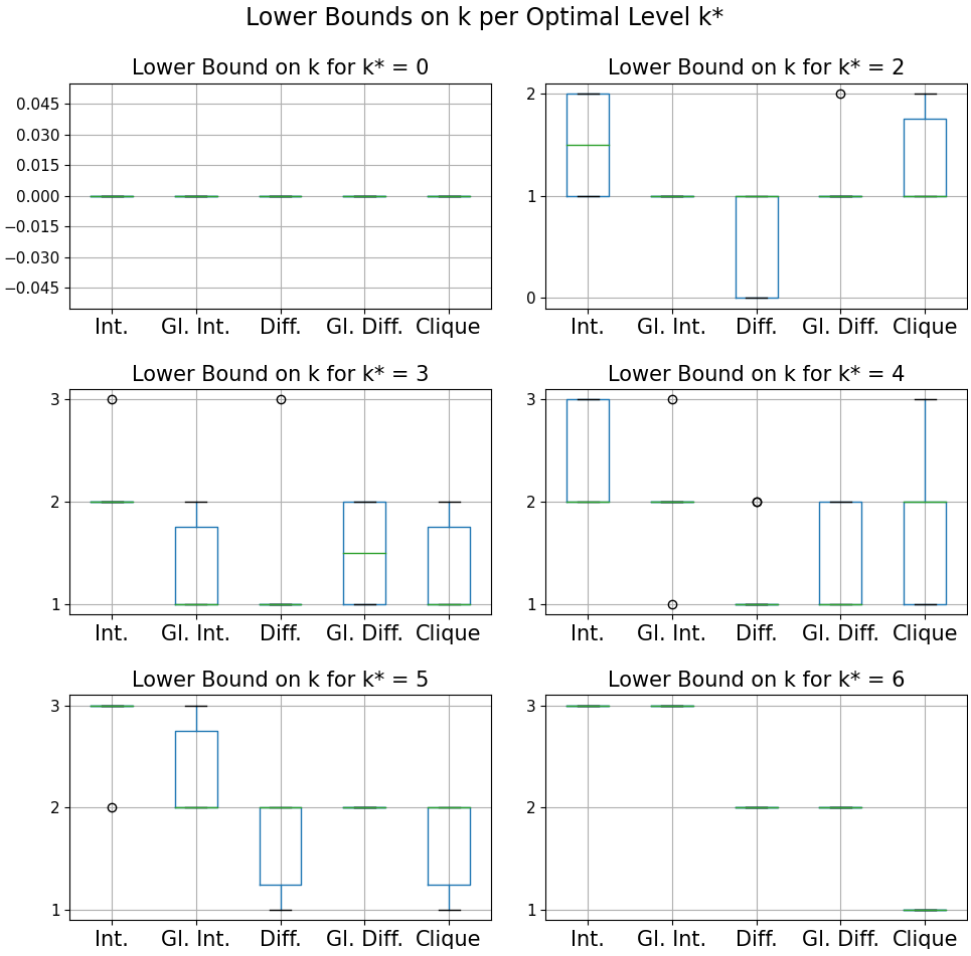


Figure 8.3: These box plots give an indication of the distribution of the five lower bounds which are determined by the proposed inequalities in Chapter 5. The bounds for instances for which all the three algorithms did not find any network (see the dark gray rows in Table 8.3), have not been used in this analysis, since the optimal level and reticulation number could not be derived from our results.

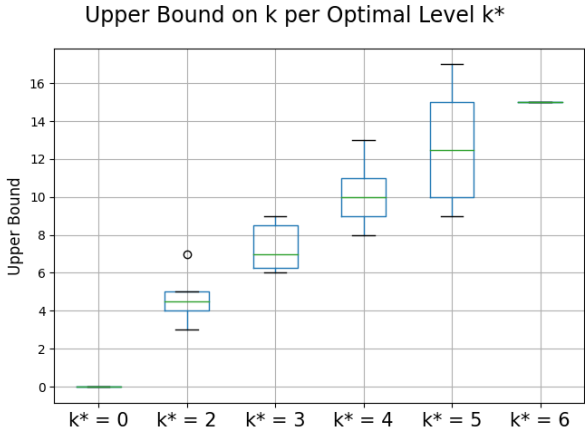


Figure 8.4: These box plots show the upper bound determined by the proposed inequality in Conj. 6.2 per optimal level k^*, for the tested instances. Only results for which the algorithms could find a network have been used in this analysis.

9

Discussion & Conclusions

In this work, two optimal algorithms called STCass and MSTCass have been presented to build an optimal network representing the clusters in e.g. multiple input trees, optimal in the sense that the network level is minimized. Note that there are no restrictions on the input clusters. In particular, they may stem from both binary as well as non-binary trees, or not from any particular tree(s) at all.

As others have observed for Cass-based algorithms, they often produce networks with less reticulations than most other algorithms. This is confirmed by our results of (M)STCass, since (M)STCass is optimal and the level and reticulation number of the output networks of Cass rarely deviate from optimality. The solving speed of MSTCass for many practical data sets, like the *Poaceae* grass data set, makes it a good candidate for any research involving phylogenetics (see Section 1.2 for examples).

Another advantage of (M)STCass is that like the original Cass algorithm, it can be used for any set of clusters. Besides, we have shown that in general, MSTCass runs faster than STCass. Cass runs faster for the tested (small sized) instances, which may be due to the faster Java compiled code than that in Python, which is the language in which (M)STCass has been implemented. Besides, this is caused by the fact that Cass collapses more (and has no optimality guarantee because of this), resulting in less hang-edges to choose from.

At least seven lower bounds have been proposed on the number of reticulations $r(N)$ per biconnected component N in a network that represents the input clusters. Such lower bounds are beneficial to lower the runtime of any Cass-based algorithm. On top of this, one new upper bound on $r(N)$ is conjectured, which can be evaluated

within one second or at most a few. In practice however, PIRN gives better lower bounds (see e.g. the appendix of [12]).

Finally, let us consider some of the many open questions. Firstly, it would be interesting to compare the performance of MSTCass with that of Cass by implementing it in Java too.

Secondly, regarding further optimization of the runtime, when hanging some leaf x in a network N, may we restrict the clusters $\mathcal{C}_x$ to be only the minimal clusters (see Conj. 4.6)? And is considering only one cluster from $\mathcal{C}_{\hat{x}}$ sufficient (if this set is non-empty) in order to obtain an optimal network finally (see Conj. 4.7)? Fourthly, may we restrict the hang-edges to those which are described in Conj. 4.10 (and Conj. 4.13)?

Then, regarding the minimization of $r(N)$, can the valid sets of hang-edges be determined in polynomial time (see Conj. 4.14 and Conj. 4.20)? And how (see a.o. Conj. 4.19)? In this respect, it might be worth the effort implementing a machine learning (ML) based algorithm which determines good MST-sets to remove in the inward removal phase of MSTCass. Say that the algorithm is in the i-th iteration and the set of clusters $\mathcal{C}_i$ remained, then the sets $minimal(\mathcal{C}_x) = minimal(\{C \in \mathcal{C}_i | x \in C\})$ could be used as input, for the current leaf x to add, as well as for the others to be added after it. Besides, the fact whether some MST-set S is a terminal or a source or none of these, may be part of the input, since it may be wise to give e.g. a source low priority (see Section 3.10).

Furthermore, ML may be used to determine good hang-edges in the network N, in each iteration in the outward building phase of MSTCass(k). This has the potential to speed up finding a network of minimum level k^*. As input for an ML-based approach, the locations of the hang-edges in N are important. Moreover, per cluster in $\mathcal{C}|\mathcal{X}(N)$, the locations of the LCAs might be considered. A more advanced algorithm might consider the sets $\mathcal{C}_x$ too for each leaf x to be added in the current iteration or later.

Besides, when one searches a lower bound on $r(N)$, are our conjectured lower bounds on this number valid? Especially the best performing one involving intersections of clusters, see Conj. 5.11? Maybe the work done on finding a lower bound on the hybridization number can be used in establishing lower bounds on $r(N)$, like that in [22].

Furthermore, suppose that the clusters originate from several trees. Then in general, MSTCass might give more representative results if the edge distances in those trees would be considered. It would be surprising if e.g. species A have lived before some species B according to the network model, while in reality, this was not the case. Regarding the time complexity of STCass, we have shown that the algorithm runs in polynomial time if the level k is fixed.

Regarding the outward building phase of (M)STCass, assume that an ST-set is hung in a network which should be hung below multiple reticulations. Then all possible binary networks are created. However, it can be the case that only a one or a few refinements can lead to an optimal network. In order to make the search tree smaller, we suggest to put effort in searching for rules by which certain refinements can be excluded from further exploration, if they cannot lead to an optimal network.

Regarding the runtimes of the three algorithms, it would be interesting to record the runtimes of e.g. MSTCass(k^*), given that $k^* = \ell(Cl(\mathcal{T}))$ for the given input trees $\mathcal{T}$. The relation between the resulting runtime and minimum level k^* can help in creating a heuristic, which jumps to a higher value of k if no network has been found within $B(k)$ seconds, for some runtime upper bound $B(k)$ per level k. For example, if the runtime of MSTCass(4) exceeds 20 seconds, then chances are high that $k^* \geq 5$.

Furthermore, the current algorithm MSTCass searches for a lowest level network which represents the input clusters. In practice, preferably, a network is demanded in which certain (input) trees occur as subnetwork. It would be interesting to create an algorithm which builds such a network.

Acknowledgements

First and foremost, thanks go to my Creator and abundant Source of all blessings by Whom and Whom only I have been able to accomplish this work. We thank Leo van Iersel for his continuous guidance when doing this work, in particular on cross points, and for giving insights, recommendations or attentions when needed. Further thanks go to my family and friends and others for their sympathy regarding this work. Besides, we thank Sam van Poelgeest for his recommendations on implementing multiprocessing in Python.

Bibliography

[1] Benjamin Albrecht. "Computing all hybridization networks for multiple binary phylogenetic input trees". In: *BMC Bioinformatics* 16.1 (2015), p. 236. ISSN: 1471-2105. DOI: 10.1186/s12859-015-0660-7. URL: https://doi.org/10.1186/s12859-015-0660-7.

[2] David Bryant and Vincent Moulton. "Neighbor-Net: An Agglomerative Method for the Construction of Phylogenetic Networks". In: *Molecular Biology and Evolution* 21.2 (Feb. 2004), pp. 255–265. ISSN: 0737-4038. DOI: 10.1093/molbev/msh018. eprint: https://academic.oup.com/mbe/article-pdf/21/2/255/4016168/msh018.pdf. URL: https://doi.org/10.1093/molbev/msh018.

[3] Joshua Collins, Simone Linz, and Charles Semple. "Quantifying hybridization in realistic time". en. In: *J Comput Biol* 18.10 (Jan. 2011), pp. 1305–1318.

[4] Dan Gusfield and Vikas Bansal. "A Fundamental Decomposition Theory for Phylogenetic Networks and Incompatible Characters". In: *Research in Computational Molecular Biology*. Ed. by Satoru Miyano et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 217–232. ISBN: 978-3-540-31950-4.

[5] Dan Gusfield, Dean Hickerson, and Satish Eddhu. "An efficiently computed lower bound on the number of recombinations in phylogenetic networks: Theory and empirical study". In: *Discrete Applied Mathematics* 155.6 (2007). Computational Molecular Biology Series, Issue V, pp. 806–830. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2005.05.044. URL: https://www.sciencedirect.com/science/article/pii/S0166218X06003702.

[6] Richard R Hudson and Norman L Kaplan. "STATISTICAL PROPERTIES OF THE NUMBER OF RECOMBINATION EVENTS IN THE HISTORY OF A SAMPLE OF DNA SEQUENCES". In: *Genetics* 111.1 (Sept. 1985), pp. 147–164. ISSN: 1943-2631. DOI: 10.1093/genetics/111.1.147. eprint: https://academic.oup.com/genetics/article-pdf/111/1/147/34449116/genetics0147.pdf. URL: https://doi.org/10.1093/genetics/111.1.147.

[7] Daniel H. Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511974076.

[8] Daniel H. Huson et al. "Computing galled networks from real data". In: *Bioinformatics* 25.12 (May 2009), pp. i85–i93. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp217. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/12/i85/472218/btp217.pdf. URL: https://doi.org/10.1093/bioinformatics/btp217.

[9] Leo van Iersel, Charles Semple, and Mike Steel. *Locating a tree in a phylogenetic network*. 2010. DOI: 10.48550/ARXIV.1006.3122. URL: https://arxiv.org/abs/1006.3122.

[10] Leo van Iersel et al. *A Practical Fixed-Parameter Algorithm for Constructing Tree-Child Networks from Multiple Binary Trees*. 2019. DOI: 10.48550/ARXIV.1907.08474. URL: https://arxiv.org/abs/1907.08474.

[11] Leo van Iersel et al. *Hybridization Number on Three Rooted Binary Trees is EPT*. 2014. DOI: 10.48550/ARXIV.1402.2136. URL: https://arxiv.org/abs/1402.2136.

[12] Leo van Iersel et al. "Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters". In: *Bioinformatics* 26.12 (June 2010), pp. i124–i131. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq202. eprint: https://academic.oup.com/bioinformatics/article-

pdf/26/12/i124/16894356/btq202.pdf. URL: https://doi.org/10.1093/bioinformatics/btq202.

[13] Holly Jackson et al. "Using heritability of stellar chemistry to reveal the history of the Milky Way". In: *Monthly Notices of the Royal Astronomical Society* 502.1 (2021), pp. 32–47. DOI: 10.1093/mnras/staa4028. URL: https://doi.org/10.1093%2Fmnras%2Fstaa4028.

[14] Iyad A. Kanj et al. "Seeing the trees and their branches in the network is hard". In: *Theoretical Computer Science* 401.1 (2008), pp. 153–164. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2008.04.019. URL: https://www.sciencedirect.com/science/article/pii/S0304397508002879.

[15] Steven Kelk, Celine Scornavacca, and Leo van Iersel. "On the Elusiveness of Clusters". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 9.2 (2012), pp. 517–534. DOI: 10.1109/TCBB.2011.128.

[16] Steven M. Kelk and Céline Scornavacca. "Constructing Minimal Phylogenetic Networks from Softwired Clusters is Fixed Parameter Tractable". In: *Algorithmica* 68 (2011), pp. 886–915. URL: https://doi.org/10.1007/s00453-012-9708-5.

[17] Jacob E. Lemieux et al. "Phylogenetic analysis of SARS-CoV-2 in Boston highlights the impact of superspreading events". In: *Science* 371.6529 (2021), eabe3261. DOI: 10.1126/science.abe3261. eprint: https://www.science.org/doi/pdf/10.1126/science.abe3261. URL: https://www.science.org/doi/abs/10.1126/science.abe3261.

[18] Simon R Myers and Robert C Griffiths. "Bounds on the Minimum Number of Recombination Events in a Sample History". In: *Genetics* 163.1 (Jan. 2003), pp. 375–394. ISSN: 1943-2631. DOI: 10.1093/genetics/163.1.375. eprint: https://academic.oup.com/genetics/article-pdf/163/1/375/42051257/genetics0375.pdf. URL: https://doi.org/10.1093/genetics/163.1.375.

[19] Teresa Piovesan and Steven Kelk. *A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees*. 2012. DOI: 10.48550/ARXIV.1207.6090. URL: https://arxiv.org/abs/1207.6090.

[20] Heiko A. Schmidt. "Phylogenetic trees from large datasets". PhD thesis. Universität Düsseldorf, 2003.

[21] Charles Semple and Mike Steel. "Phylogenetics". In: (2003). URL: http://scholar.google.com/scholar.bib?q=info:r68BHLfaPAwJ:scholar.google.com/&output=citation&scisig=AAGBfm0AAAAAVUxSkiaGFww2jlsBnkftePbvG42&scisf=4&hl=en&scfhb=1.

[22] Hein-J. Song Y. "On the minimum number of recombination events in the evolutionary history of DNA sequences". In: *Journal of Mathematical Biology* 48.2 (Feb. 2004), pp. 160–186. DOI: 10.1007/s00285-003-0227-5. URL: https://doi.org/10.1007/s00285-003-0227-5.

[23] Yatish Turakhia et al. "Pandemic-scale phylogenomics reveals the SARS-CoV-2 recombination landscape". In: *Nature* 609.7929 (2022), pp. 994–997. ISSN: 1476-4687. DOI: 10.1038/s41586-022-05189-9. URL: https://doi.org/10.1038/s41586-022-05189-9.

[24] Juan Wang et al. "BIMLR: A method for constructing rooted phylogenetic networks from rooted phylogenetic trees". In: *Gene* 527.1 (2013), pp. 344–351. ISSN: 0378-1119. DOI: https://doi.org/10.1016/j.gene.2013.06.036. URL: https://www.sciencedirect.com/science/article/pii/S0378111913007932.

[25] Juan Wang et al. "Lnetwork: an efficient and effective method for constructing phylogenetic networks". In: *Bioinformatics* 29.18 (June 2013), pp. 2269–2276. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btt378. eprint: https://academic.oup.com/bioinformatics/article-pdf/29/18/2269/17127291/btt378.pdf. URL: https://doi.org/10.1093/bioinformatics/btt378.

[26] Yufeng Wu. “Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees”. In: *Bioinformatics* 26.12 (June 2010), pp. i140–i148. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq198. eprint: https://academic.oup.com/bioinformatics/article-pdf/26/12/i140/16894064/btq198.pdf. URL: https://doi.org/10.1093/bioinformatics/btq198.

A

The Original Cass Algorithm

The original Cass algorithm[12] is repeated below in Alg. 5. In order to solve an instance in practice, a higher level routine is required which determines the incompatibility graph, its connected components and calls Cass(k) repeatedly, for increasing values of level k. This is scetched in Alg. 6.

Observe the function GetRetNumLowerBound on line 5 in Alg. 6. The original Cass algorithm is designed with the idea that $k = 0$ in the first iteration. However, the rules and guide lines (conjectures) we have presented in this paper may be used to derive a lower bound on the level k of an optimal network. Obviously, this speeds up computations.

Algorithm 5: Cass(k): Construct a simple level−k network from clusters[12], if it exists.

Input: $(\mathcal{C}, \mathcal{X}, k, k')$

Output: Cass$(\mathcal{C}, \mathcal{X}, k, k')$

/* Initially, $k' = k$ */

if $k' = 0$ **then**
  return the unique tree representing exactly the clusters in $\mathcal{C}$ or return $\emptyset$ if no such tree exists
$\mathcal{N} := \emptyset$
/* δ is a dummy taxon not in $\mathcal{X}$ */
for $x \in \mathcal{X} \cup \{\delta\}$ **do**
  remove leaf: $\mathcal{C}' := \mathcal{C} \backslash \{x\}$
  collapse: $\mathcal{C}'' :=$ Collapse$(\mathcal{C}')$
  recurse: $\mathcal{N}' =$ Cass$(\mathcal{C}'', \mathcal{X}(\mathcal{C}''), k, k' - 1)$
  for *each network* N' *in* $\mathcal{N}'$ **do**
    decollapse: replace each leaf of N' labeled by a maximal ST-set S w.r.t. $\mathcal{C}'$ by the tree on S repreeenting exactly those clusters in $\mathcal{C}'|S$
    for *each pair of edges* e_1, e_2 *(not necessarily distinct)* **do**
      let N'' be a copy of N'
      add leaf below reticulation: create in N'' a reticulation t, a leaf l labeled x and an edge from t to l; then, for $i = 1, 2$, insert into N'' a node v_i into edge e_i and add an edge from v_i to t
      if N'' *represents* $\mathcal{C}$ **then**
        save network: $\mathcal{N} := \mathcal{N} \cup \{N''\}$
if $k' = k$ **then**
  return any simple level-k network in $\mathcal{N}$, after removing each leaf labeled δ and contracting each edge connecting two reticulations
return $\mathcal{N}$

Algorithm 6: CASS: construct a simple, optimal network from clusters. It uses CASS(k).

Input: $(\mathcal{C}, \mathcal{X})$

Output: CASS$(\mathcal{C}, \mathcal{X})$

```
1  IG := GetIncompatibilityGraph(C, X)
2  CC := GetConnectedComponents(IG)
3  N := empty network                          // This will become an optimal network
4  for C in CC do
       /* Handle connected component C                                          */
       /* Get a lower bound on the number of reticulations                       */
5      k := GetRetNumLowerBound(C)
6      while N is empty do
           /* Get a corresponding optimal subnetwork.  Cass may be replaced by
              STCass                                                            */
7          N := Cass(k)
8          k := k + 1
       /* Merge one of the subnetworks that are found with the final network    */
9      N := Merge(N, N[0])
10 return N
```