Computing Hybridization Networks for Multiple Rooted Binary Phylogenetic Trees by Maximum Acyclic Agreement Forests

Benjamin Albrecht

Institut für Informatik, Ludwig-Maximilians-Universität, Germany

Abstract It is a known fact that given two rooted binary phylogenetic trees the concept of maximum acyclic agreement forests is sufficient to compute hybridization networks with minimum hybridization number. In this work we demonstrate by, first, presenting an algorithm and, second, showing its correctness that this concept is also sufficient in the case of multiple input trees. In detail, we show that for computing hybridization networks with minimum hybridization number for multiple rooted binary phylogenetic trees, it is sufficient to consider only maximum acyclic agreement forests instead of acyclic agreement forests of arbitrary size.

Keywords Directed Acyclic Graphs · Hybridization · Maximum Acyclic Agreement Forests · Bounded Search · Phylogenetics

* Benjamin Albrecht
Institut für Informatik, Ludwig-Maximilians-Universität, Germany
Tel.: +49-89-2180-4069
E-mail: albrecht@bio.ifi.lmu.de
1 Introduction

The evolution of certain species is often described by a single phylogenetic tree outlining particular speciation events. Because of reticulation events, in some cases, however, a tree is insufficient, since different genetic sequences give rise to different phylogenetic trees each containing different speciation events. An important reticulation event, which could be prevalently discovered in plants but also in animals, is hybridization [12]. For outlining hybridization events different phylogenetic trees that have been constructed for certain species are reconciled into a single hybridization network. In contrast to phylogenetic trees, hybridization networks do not only contain internal nodes of indegree one, outlining speciation events, but, additionally, can contain nodes of indegree larger than one each outlining putative hybridization events.

The problem of computing hybridization networks is known to be NP-hard [4] even for the simplest case when only two binary phylogenetic input trees are given. In the general case, however, if the input consists of more than two rooted binary trees, the problem still remains fixed-parameter tractable as recently shown by van Iersel and Linz [10]. Nevertheless, this means when developing an algorithm solving this problem, the challenge is not only to guarantee its correctness, but, especially, to guarantee a good practical running time, since, due to its complexity, the theoretical running time grows exponentially with the input size. Thus, the algorithm, on the one hand, has to be quite sophisticated, and, on the other hand, has to be executed efficiently which can be achieved, for example, by running exhaustive parts of the algorithms in parallel.

A method to compute hybridization networks for two given binary input trees can be divided into the following two major steps: First, maximum acyclic agreement forests are computed by cutting down the input trees, and, second, the components of such an agreement forest are again reattached by certain reticulate edges such that both input trees are embedded in the resulting network. Such agreement forests can be seen as common subparts occurring in both input trees. The term maximum simply denotes that there is no smaller set fulfilling the properties of an agreement forest, and, the acyclic constraint just guarantees that it is possible to reattach its components in a specific way to a hybridization network. There exists, typically, not just one hybridization network but, in general, a really large number. To recognize putative hybridization events, biologists are interested in all possible hybridization networks, since the more frequently a hybridization event is present in a set of possible evolutionary scenarios the higher is its probability that it really affected the evolution of the considered species. Thus, there is a need for two algorithms: one for the computation of all maximum acyclic agreement forests and another one for computing all hybridization networks for each of those agreement forests.

Note that if only two input trees are given, for computing networks with minimum hybridization number it does not matter if the latter algorithm really computes all hybridization networks. However, when adding further input trees to this subset of networks, in a worst case scenario one never receives a network with minimum hybridization number, since those networks that would enable
an embedding of these trees with minimum hybridization number are missing in
the initial set.

While there are some programs providing methods for computing hybridization
networks for two input trees \cite{7,9}, in this work we will introduce an efficient
algorithm computing hybridization networks for an arbitrary number, especially
for more than two input trees whose work flow can be briefly summarized as
follows: Starting with one input tree, all other input trees are embedded se-
quentially into a growing number of networks by adding further reticulate edges
corresponding to certain components of a maximum acyclic agreement forest.
As mentioned above, to guarantee the computation of a hybridization network
with minimum hybridization number, it is important that each input tree is
added to a so far computed network in all different ways. This implies, espe-
cially, that at the beginning, when adding the second tree of the ordering to the
first one, all topologically different hybridization networks have to be computed.
Missing one of those networks could imply that a computational path leading
to a hybridization network is lost, and, in a worst case scenario, the resulting
output only consists of networks whose hybridization number is not minimal.
Thus, to solve the problem of computing hybridization networks for more than
two input trees we have to solve the subproblem of generating all topologically
different hybridization networks for a given maximum acyclic agreement forest.
Note that, as shown in Section 3, a crucial observation of this work is that it
suffices to consider only maximum acyclic agreement forests for the computation
of hybridization networks with minimum hybridization number displaying more
than two input trees.

Please note, that, currently, there is one program PIRN (v2.0) \cite{16,17} that
is also able to compute hybridization networks with minimum hybridization
number for multiple rooted phylogenetic trees. A simulation, however, revealed
that an implementation of our algorithm provides the clearly better practical
running time and, additionally, is able to output a larger set of networks \cite{3}.
Moreover, in contrast to our method, there does not exist a proof of correctness
showing that the underlying algorithm of PIRN guarantees the computation of
networks with minimum hybridization number.

This work is organized as follows: First, in Section 2 and 3 some basic defini-
tions are introduced. In Section 4 we give a detailed description of our algorithm
ALLHNnetworks whose correctness is showed in the following section. Finally,
we end this paper by briefly discussing the theoretical running time of the pre-
sented algorithm in Section 6 and by giving some concluding remarks.
2 Preliminaries

In this section, we give some preliminary definitions concerning phylogenetic trees, hybridization networks, and agreement forests following [8, 13], which will be used in Section 4 for describing the algorithm allHNetworks and in Section 5 for showing its correctness.

**Phylogenetic trees.** A rooted phylogenetic $\mathcal{X}$-tree $T$ is a directed acyclic graph whose edges are directed from the root to the leaves and whose nodes, except for the root, have an degree not equal to two. Its leaves are bijectively labeled by the taxa set $\mathcal{X}$ that usually consists of certain species or genes and is denoted by $L(T)$. Considering a node $v$ of $T$ the label set $L(v)$ contains each taxon which is contained in its subtree. Given a set of trees $\mathcal{F}$ the label set $L(\mathcal{F})$ is simply the union of each label set of a tree $F_i \in \mathcal{F}$.

Now, based on a taxa set $\mathcal{X}' \subseteq \mathcal{X}$, we can define a restricted subtree of a rooted phylogenetic $\mathcal{X}$-tree denoted by $T|_{\mathcal{X}'}$. The restricted tree $T|_{\mathcal{X}'}$ is computed by, first, deleting each leaf whose taxon is not contained in $\mathcal{X}'$ repeatedly, resulting in a subgraph $T'$ that is denoted by $T(\mathcal{X}')$, and, second, by removing each node of degree two. In the following, we will call such nodes of degree 2, except for the root, isolated nodes. Given a tree $T$, $\tilde{T}$ refers to the tree $T$ whose isolated nodes have been removed. Note that the result of such a restriction in respect of a certain taxa set $\mathcal{X}'$ is a rooted phylogenetic $\mathcal{X}'$-tree.

**Hybridization networks.** A hybridization network $N$ is a rooted phylogenetic network whose leaves are bijectively labeled by the taxa set $\mathcal{X}$. A hybridization network can contain undirected but no directed cycles and, thus, there can exist nodes of indegree greater one. In general, those nodes are denoted as hybridization nodes and each of its incoming edges as hybridization. Usually, to increase the readability of $N$ the outdegree of every hybridization node is one.

We say a hybridization network $N$ on $\mathcal{X}$ displays a certain rooted phylogenetic $\mathcal{X}$-tree $T$, with $\mathcal{X}' \subseteq \mathcal{X}$, if we can delete a certain set of hybridization edges followed by removing each isolated node such that the resulting rooted phylogenetic $\mathcal{X}$-tree $T$ contains $T'$ as a restricted subtree on $\mathcal{X}'$. Note that the algorithm allHNetworks (cf. Sec. 4) computes a hybridization network that, one the one hand, displays a certain set of trees and, on the other hand, is optimal in terms of its hybridization number (cf. Def. 1).

Now given a hybridization network $N$ on $\mathcal{X}$, displaying a rooted phylogenetic $\mathcal{X}$-tree $T$, the reduced network $N|_{T, \mathcal{X}'}$, with $\mathcal{X}' \subseteq \mathcal{X}$, is computed as follows: First, all hybridization edges that are not necessary for displaying $T$ are deleted and, second, each leaf that is not labeled by $\mathcal{X}'$ is removed repeatedly. The result is a directed graph similar to $T|_{\mathcal{X}'}$ but still containing isolated nodes and, thus, each node in $N|_{T, \mathcal{X}'}$ can be mapped back to exactly one certain node of the unrestricted network $N$. Moreover, we say the edge set $E(N, T)$ denotes all hybridization edges that are necessary for displaying $T$ in $N$ and the network $N - E(N, T)$ corresponds to a reduced network that is computed by, first, dele-
ing each edge in $E(N,T)$, and, second, by removing each isolated node.

**Agreement forests.** For a formal definition of an acyclic agreement forest $F$, we refer to the work of Scornavacca et al.\cite{13}. Note that before the computation of an agreement forest of two rooted phylogenetic $\mathcal{X}$-trees $T_1$ and $T_2$ both trees are transformed into $T'_1$ and $T'_2$ by attaching an edge to its roots which leaf is labeled by a unique taxon $\rho \notin \mathcal{X}$ such that $\mathcal{L}(T'_1) = \mathcal{L}(T'_2) = \mathcal{X} \cup \{\rho\}$. Thus, an acyclic agreement forest always contains a component $F_\rho$ with $\rho \in \mathcal{L}(F_\rho)$, the so-called root component.

Given an acyclic agreement forest $F = \{F_\rho, F_1, \ldots, F_k\}$ on two trees $T_1$ and $T_2$, we can compute different acyclic orderings by constructing a directed graph $AG(T_1, T_2, F)$ containing one certain node for each $F_i \in F$. An edge of $AG(T_1, T_2, F)$ is directed from a source node corresponding to $F_i$ to a target node corresponding to $F_j$, $i \neq j$, if either the root of $T_1(F_i)$ is an ancestor of $T_1(F_j)$ or the root of $T_2(F_i)$ is an ancestor of $T_2(F_j)$. Based on $AG(T_1, T_2, F)$ we call an ordering of the components $\Pi(F) = (F_\rho, F_1, \ldots, F_k)$ an acyclic ordering if for each two components $F_i$ and $F_j$, $i > j$, there is no direct path in $AG(T_1, T_2, F)$ leading from the node corresponding to $F_j$ to the node corresponding to $F_i$. Note that usually there exist several acyclic orderings of $F$.

Given a hybridization network $N$ on $\mathcal{X}$ displaying a set of rooted phylogenetic $\mathcal{X}$-trees $\mathcal{T}$ and an acyclic agreement forest $F$ of two trees $T'_1$ and $T'_i$ with $T'_1, T'_i \in \mathcal{T}$ we say that $N$ displays this agreement forest, denoted by $N \supset F$, if we can derive $F$ from $N$ by deleting each hybridization edge that is not necessary for displaying both trees, $T'_1$ and $T'_i$, followed by removing all isolated nodes of each resulting subtree.

### 3 Optimization criteria

Since hybridization events occurring during the evolution of certain species are considered as rare phenomena, networks containing a minimum number of hybridization edges outline the most likely scenario in this context. Moreover, hybridization events involving more than two species, in general, are not possible and, thus, as a direct consequence, one could claim that methods should only compute networks having hybridization nodes with exactly two in-edges. The readability of such networks, however, gets worse, since for each hybridization node, that could be drawn with $n$ in-edges, would result in a network containing a stack of $n - 1$ hybridization nodes. Thus, to keep the readability of the network and to guarantee the most likely network, the hybridization number is most commonly defined as follows:

**Definition 1.** The hybridization number $h(N)$ of a hybridization network $N$ is

$$h(N) = \sum_{v \in V \setminus \{\rho\}} (\delta^{-}(v) - 1),$$

where $V$ denotes the set of nodes in $N$ and $\rho$ the root of $N$. 

5
Moreover, we have to clarify in which hybridization networks biologists are interested: Given a hybridization network containing a node \( v \) of indegree of least 3 you can generate further topologically different networks by dragging some of the hybridization edges upwards what, in the biological context, would mean that these events happened right before the event corresponding to the in-edges of \( v \). This can be done, however, for all different combinations of in-edges and, thus, in general, there exists a large number of different orderings of hybridization events all being valid solutions from a mathematical point of view. As a consequence, to keep the number of resulting networks small, we are only interested in those networks with minimum hybridization number providing, additionally, a minimum number of unresolved hybridization nodes. In the following we will refer to such networks as *canonical* networks.

### 4 The Algorithm allHNetworks

Given a set of rooted phylogenetic \( \mathcal{X} \)-trees \( \mathcal{T} = \{ T_1, \ldots, T_n \} \) and a parameter \( k \in \mathbb{N} \), our algorithm allHNetworks follows a branch-and-bound approach by conducting the following major steps: For each ordering of \( \mathcal{T} \), the trees are added sequentially to a set of networks \( \mathcal{N} \). At the beginning, \( \mathcal{N} \) consists only of one element, which is the first input tree of the ordering. By adding the other input trees sequentially, the size of \( \mathcal{N} \) grows rapidly, since each input tree \( T_i \) is added to each network within \( \mathcal{N} \) in several different ways. Each time the hybridization number of an extended network exceeds \( k \), this new network is rejected, since, by adding further input trees, its hybridization number is never decreased.

Depending on two different questions, the algorithm provides two different abort criteria:

1. How many hybridization edges are necessary to display all trees within one network?
   
   Abort criterion: As soon as one hybridization network is computed whose hybridization number is \( k \) and there is no other hybridization network having a hybridization number less than \( k \).

2. What are the hybridization networks containing the minimum number of hybridization edges?
   
   Abort criterion: As soon as all hybridization networks are computed whose hybridization number is \( k \) and there is no other hybridization network having a hybridization number less than \( k \).

To guarantee an efficient computation of a hybridization network, parameter \( k \) is set to an initial value and is increased by one if a network displaying \( \mathcal{T} \) with hybridization number \( k \) could not be computed so far. At the beginning \( k \) can be either simply set to zero or to a lower bound, e.g.

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

In practice, however, the lower bound does not improve the practical running time significantly, since the required steps for those \( k \)'s that can be skipped are usually of rather low computational complexity.
4.1 Adding Trees to Networks

Given a hybridization network $N$, we say that a tree $T$ is displayed in $N$ if you can delete a certain subset of hybridization edges $E(N, T)$ together with all resulting isolated nodes such that the topology of the so reduced network $\bar{N} | T, X$ forms a tree that is isomorphic to $T$ (cf. Sec. 2). This implies, if such a subset does not exist, we have to insert new hybridization edges for displaying $T$ in $N$. Given an embedded tree $T'$ that is already displayed in $N$, those edges can be derived from the components of an acyclic agreement forest of $T'$ and $T$. In general, we can summarize the basic steps for adding an input tree $T$ to a so far computed network $N$ as follows:

1. Choose an embedded tree $T'$ of $N$ by selecting one in-edge of each hybridization node.
2. Compute a maximum acyclic agreement forests $F$ of the two trees $T'$ and $T$.
3. For each component of $F$, except $F_{\rho}$, a source and target node is created such that, by connecting those nodes, $T$ is embedded in the resulting network.

Note that this step is discussed separately in the following Section 4.2.

The resulting network depends on the chosen embedded tree $T'$, since different embedded trees lead to different maximum acyclic agreement forests and different maximum acyclic agreement forests, again, lead to different hybridization edges, which have to be added to $N$ for embedding $T$. Thus, to guarantee the computation of all hybridization networks with minimum hybridization number, all three steps have to be run for all embedded trees, what can be up to $2^r$ for a network containing $r$ hybridization nodes, and all maximum acyclic agreement forests of the chosen embedded tree and the current input tree. Note that the adding of the components of a maximum acyclic agreement forest to a so far computed network is not a trivial step, since, usually depending on other so far added hybridization edges, there exist many ways how this can be achieved. We will discuss this important step in detail in the following section (cf. Sec. 4.2).

4.2 Adding Components to a Network

Given a particular ordering of rooted phylogenetic $X$-trees $(T_1, \ldots, T_n)$, a network $N$ displaying each tree $T_k$ with $k < i$ together with an embedded tree $T'$ of $N$, we can add $T_i$ to $N$ by connecting certain parts of the network corresponding to the components of a maximum acyclic agreement forest $F$ of $T'$ and $T_i$. Hence, for each component a certain target and a certain source node in $N$ has to be computed which are then connected by a new hybridization edge. Different source and target nodes can lead to topologically different networks containing different sets of embedded trees. Thus, to guarantee the computation of a hybridization network with minimum hybridization number at the end of the algorithm, we have to compute all different combinations of possible source and target nodes for each component of $F$ for each possible acyclic ordering of $F$.

In general, we can summarize all important steps for adding the components of $F$ to $N$ as follows:
1. Compute an acyclic ordering \((F_\rho, F_1, \ldots, F_k)\) of \(F\).
2. Add each component of this ordering \((F_\rho, F_1, \ldots, F_k)\), except \(F_\rho\), sequentially to \(N\) by connecting all possible source and target nodes.

Note that the output of these two steps is usually a large number of new networks, since, in general, there exists different ways of how the components can be inserted into \(N\) such that \(T_i\) is displayed in the resulting network. Whereas each acyclic ordering of \(F\) can be simply computed with the help of a directed graph \(AG(T', T_i, F)\) (cf. Sec. 2), the insertion of a certain component is quite more complex. Given a certain acyclic ordering \((F_\rho, F_1, \ldots, F_k)\) of a maximum acyclic agreement forest \(F\), the algorithm adds each component \(F_i \in F\) sequentially to \(N\) before going into details, we give some brief information about the properties of hybridization edges within the network \(N\).

**Properties of Hybridization Edges.** Some hybridization edges of \(N\) are necessary for displaying the embedded tree \(T'\) and some hybridization edges are necessary for displaying the input tree \(T_i\) that is going to be added to \(N\). At the beginning no hybridization edges are necessary for displaying \(T_i\) and all hybridization edges that are necessary for displaying \(T'\) in \(N\), which are all edges in \(E(N, T')\), are mapped to \(T'\). Whereas the mapping of \(T'\) stays unchanged, the mapping of \(T_i\) is updated after each insertion of a component \(F_i \in F\) such that after the insertion of all components all those hybridization edges are mapped to \(T_i\) that are necessary for its embedding in \(N\).

Now, given such a mapping of the hybridization edges, we can describe the way of adding a component \(F_i\) of the acyclic ordering to the so far computed network \(N\) by the following three steps I–III.

**I Computation of target and source nodes.** Let \(F' = \{F_\rho, F_1, \ldots, F_{i-1}\} \subset F = \{F_\rho, F_1, \ldots, F_k\}\) be the set of components that has been added to the network so far and \(F_i\) the component that is added in the current step. Note that at the beginning \(L(F')\) equals \(L(F_\rho)\), since the first component that is added to the network is the second component of the acyclic ordering which is \(F_1\).

**I.I Computation of Target Nodes.** The set of target nodes \(V_t\) is computed such that it contains all nodes \(v\) with \(\tilde{N}_{\mathcal{T'}, \mathcal{L}(F')}(v)\) isomorphic to \(T_i\vert_{\mathcal{L}(F_i)}\). Note that, due to the restriction of the network to \(\mathcal{L}(F')\) and due to other so far added hybridization edges, this set can contain more than one node.

Since we are only interested in canonical networks we omit those target nodes that are source nodes of hybridization edges. Note that this is an optional step just reducing the number of reported hybridization networks (cf. Sec. I).

**I.II Computation of Source Nodes of Type A.** The set of source nodes of \(T_{\text{Type } A}\), denoted by \(V_{\text{s}}\), is computed such that it contains all nodes \(v\) with \(\tilde{N}_{\mathcal{T'}, \mathcal{L}(F')}(v)\) isomorphic to \(T_i|_{\mathcal{L}(F')}|_{\mathcal{L}(F_i)}\), where \(v_{\text{sib}}\) denotes the sibling of the node \(v\) with \(\mathcal{L}(v) = \mathcal{L}(F_i)\) in \(T_i|_{\mathcal{L}(F')}\). Note that again, due to the restriction of the network to \(\mathcal{L}(F')\) and due to other so far added hybridization edges,
this set can contain more than one node.

**I.III Computation of Source Nodes of Type B.** The set of source nodes of Type B, denoted by $V^B_s$, is computed such that it contains each node $v$ of a subtree that, on the one hand, is attached to a node in $V^A_s$ and, on the other hand, does not contain any taxa of $\mathcal{L}(F')$. Moreover, the intersection of each leaf set $\mathcal{L}(F)$ of a component $F \neq F_\rho$ in $\mathcal{F}$ and the leaf set $\mathcal{L}(v)$ of the node $v$ has to be either empty or $\mathcal{L}(F)$ in total. This implies, in detail, that $v$ must not be part of a subtree rooted at a source node corresponding to a component that is added afterwards.

The definitions of target and source nodes is illustrated in Figure 1.

![Fig. 1](image)

**Remark.** Regarding two components $F_i$ and $F_j$ of a fixed ordering of $\mathcal{F}$ with $i < j$ and $\mathcal{F}^* = \{F_\rho, F_1, \ldots, F_j\}$, then it might be the case that the two roots of $T_i|_{\mathcal{L}(\mathcal{F}^*)}(F_i)$ and $T_i|_{\mathcal{L}(\mathcal{F}^*)}(F_j)$ are siblings in $T_i|_{\mathcal{L}(\mathcal{F}^*)}$. In this case $F_i$ could be either added before $F_j$ or vice versa. If $F_i$ is inserted before $F_j$, a node whose leaf set corresponds to $\mathcal{L}(F_i)$ in $N|_{T_i,\mathcal{L}(\mathcal{F}^*)}$ would act as source node when adding $F_j$ to $N$. By adding $F_j$ before $F_i$ this would happen the other way round what, apparently, would make a difference in terms of the topology of the resulting network simply due to the insertion of two different directed edges. As a consequence, to receive all different topologically hybridization networks $\mathcal{N}$ displaying $T_i$, we have to consider all different acyclic orderings of a maximum acyclic agree-
II Adding new hybridization edges. Given a target node $t$ and a source node $s$ of either Type A or Type B, which are computed as described above, a new hybridization edge is inserted as follows:

1. The in-edge of the source node $s$ is split such that there is a new node $s'$ which is linked to $s$ and to the parent of $s$. If the parent of $t$ is of indegree one $t$ is split in the same way. Otherwise, its parent node acts as $t'$ which allows the computation of networks containing nodes of indegree $\geq 3$. Note that this is an optional step which does not influence the computation of minimum hybridization networks.
2. Now, the two nodes, $s'$ and $t'$, are connected through a path $P$ consisting of two edges. Since we do not allow nodes of indegree $>1$ as source nodes this enables an attaching of a target node to a certain hybridization edge.
3. Finally, the hybridization edge of the new added path $P$ is mapped to $T_i$.

Now, for each possible pair of target and source nodes a new network $\hat{N}$ is generated. This is necessary, since each of the final networks contains different sets of embedded trees that are used for inserting the next input tree (see Sec. 4.1). Omitting one of those possible networks could mean that you might miss an embedded tree that is necessary for following a computational path leading to networks with minimum hybridization number. In a worst case scenario, these could be the only hybridization network and, thus, the output would consist only of those networks whose hybridization number is not minimal.

III Updating hybridization edges. After an adding of a certain component the edges of a new computed network $\hat{N}$ are updated by following each path leading from a taxa in $X' \cup L(F_i)$ to the root. Whenever we arrive at a hybridization node, we choose the hybridization edge, which is mapped to $T_i$, or, if such an edge does not exist, the edge which is mapped to $T'$. Note that there always exists a unique path $P$ from a taxon in $X'$ leading to the root and, thus, we can set up a new mapping by assigning $T_i$ to the set of hybridization edges $E_P$ which is contained in $P$. Note that $E_P$ equals $E(\hat{N}, T_i | F_\rho,\ldots,F_i)$ which means that after adding $F_i$ the new network $\hat{N}$ displays $T_i | L(F_\rho,\ldots,F_i)$. 

10
We end this section by giving a brief pseudocode summarizing the important steps of our algorithm allHNetworks. Note that each step denoted by a roman numeral refers to an equally marked part listed in the previous Section 4.2.

Algorithm 1: allHNetworks($T$)

\begin{algorithm}
\begin{algorithmic}[1]
  \Input Set of rooted phylogenetic $X$-trees $T$
  \Output All topologically different hybridization networks $N$ with minimum hybridization number
  \For{$k = 1, \ldots$}
    \State $N = \emptyset$
    \ForEach ordering $\pi$ of $T$
      \State $T_1 = \pi(1)$
      \State $N' = \{T_1\}$
      \For{$i = 2$ to $n$}
        \State $T_i = \pi(i)$
        \ForAll $N \in N'$
          \ForAll $T'$ displayed in $N$
            \ForAll maximum acyclic agreement forest $F$ of $T'$ and $T_i$
              \ForAll acyclic ordering $(F_\rho, F_1, \ldots, F_m)$ of $F$
                \For{$j = 1$ to $m$}
                  \State I. Compute target nodes $V_t$ of $F_j$;
                  \State II. Compute source nodes of Type A $V_a$ of $F_j$;
                  \State III. Compute source nodes of Type B $V_b$ of $F_j$;
                  \State $V_s = V_a \cup V_b$;
                  \ForAll $(s, t) \in V_s \times V_t$
                    \State $N' = N$;
                    \State II. Insert hybridization edge $(s, t)$;
                    \State III. Update all hybridization edges of $N'$;
                \EndFor
              \EndFor
            \EndFor
          \EndFor
        \EndFor
      \EndFor
    \EndFor
    \If{$R(N') < k$}
      \State $N' = N' \cup \{N'\}$
    \EndIf
    \State $N = N \cup N'$
  \EndFor
  \If{$N \neq \emptyset$}
    \State return $N$
  \EndIf
\end{algorithmic}
\end{algorithm}
5 Proof of correctness

The algorithm ALLNETWORKS described in the previous Section guarantees the computation of all topologically different hybridization networks which are, in our context, all canonical networks as defined in Section 3. The proof given in this section is based on three Lemmas that are now discussed in detail:

Lemma 1. Given a certain ordering of rooted phylogenetic $\mathcal{X}$-trees $(T_1, \ldots, T_i)$, a network $N_{i-1}$ displaying each tree in \{ $T_1, \ldots, T_{i-1}$ \}, an embedded tree $T'$ of $N_{i-1}$, and a maximum acyclic agreement forest $F$ of $T'$ and $T_i$. The algorithm ALLNETWORKS computes a set of all hybridization networks $N$ displaying $T_i$ such that there is no other hybridization network $N_i \notin \mathcal{N}$ with $N_i \supset F$ and $N_i - E(N_i, T_i) \cong N_{i-1}$.

Proof. Given a fixed acyclic ordering $\Pi(F) = (F_p, F_1, \ldots, F_k)$ of a maximum acyclic agreement forest $F$ of two trees $T'$ and $T_i$, for each component $F_i \neq F_p$, all possible target and source nodes in $N_{i-1}$ are computed: In the following let $F' = \{F_p, F_1, \ldots, F_{i-1}\}$, $\mathcal{X}' = \mathcal{L}(F')$, and $v_{\text{sib}}$ be the sibling of a node $v$ with $\mathcal{L}(v) = \mathcal{L}(F_i)$ in $T_i|_{\mathcal{L}(F') \cup \mathcal{L}(F_i)}$:

- Since for each target node $t \in \mathcal{V}_i$ the two trees $\tilde{N}_{i-1}|_{T'_i \cup \mathcal{L}(F')} (t)$ and $T_i|_{\mathcal{L}(F_i)}$ are isomorphic, each node not in $\mathcal{V}_i$ does not fulfill this property and, thus, the resulting network $N_i$ would not display $F_i$ and, as a direct consequence, $N_i$ would not display $T_i$.
- For each source node $s \in \mathcal{V}_s^A \cup \mathcal{V}_s^B$ either the two trees $\tilde{N}_{i-1}|_{T'_i \cup \mathcal{L}(F')} (s)$ and $T_i|_{\mathcal{L}(F')} (v_{\text{sib}})$ are isomorphic (if $s \in \mathcal{V}_s^A$) or, after the insertion of all components in $F$, there exists a certain path leading to such a node whose edges are all necessary for displaying $T_i$ (if $s \in \mathcal{V}_s^B$). Choosing a node $s'$ not in $\mathcal{V}_s^A$ nor in $\mathcal{V}_s^B$ as source node, the hybridization edge $e$ inserted for $s'$ and $t \in \mathcal{V}_i$, could not be used for displaying $T_i$ in $N$, since $T_i|_{\mathcal{X}' \cup \mathcal{L}(F_i)}$ does not contain a node $w'$ whose subtree $\tilde{T}_i|_{\mathcal{X}' \cup \mathcal{L}(F_i)} (s')$ is isomorphic to $\tilde{N}_{i}|_{T'_i \cup \mathcal{L}(F')} (s')$.

Thus, for a fixed ordering $\Pi(F)$ the algorithm ALLNETWORKS guarantees the computation of all possible source and target nodes that can be used for adding $F_i$ correctly to the so far computed network $N_{i-1}$.

Now, for finishing the proof of Lemma 1, we still have to show that for each hybridization network $N_i$ with $N_i \supset F$ and $N_i - E(N_i, T_i) \cong N_{i-1}$ we can derive an acyclic agreement forest $F$ that can be used for its computation. This can be done by applying the following three simple steps to $N_i$:

- Delete all hybridization edges that are not necessary for embedding $T_i$. The result is a binary tree $\tilde{T}_i$ still containing isolated nodes.
- For each remaining hybridization edge assign its distance to the root of $\tilde{T}_i$.
- Delete all hybridization edges from $\tilde{T}_i$, remove each isolated node, and sort the resulting set of subtrees increasingly to the distance-values of its corresponding deleted in-edge that has been assigned in the previous step.
As a direct consequence, for each $N_i$ with $N_i \supset F$ and $N_i - E(N_i, T_i) \cong N_{i-1}$, there exists an acyclic ordering $\Pi_i(F)$ such that the algorithm guarantees the computation of all hybridization networks $N_i$ displaying $T_i$ what finishes the proof of Lemma 1.

We have shown so far, that, given an ordering of input trees $\Pi^* = (T_1, \ldots, T_n)$, each input tree $T_i$ is added to the so far computed network by inserting each maximum acyclic agreement forest $F$ of each embedded tree $T'$ and $T_i$ in all different ways such that there is no other network of different topology displaying $T_i$ and $F$. Thus, by performing these insertion steps sequentially for each input tree in $\Pi^*$, all hybridization networks, each displaying a certain set of maximum acyclic agreement forests, are computed. We still have to answer the question, however, why acyclic agreement forests that are not maximal and are leading to the computation of a hybridization networks with minimum hybridization number, do not have to be considered. For example, as depicted in Figure 2, it could be the case that more hybridization edges have to be investigated when inserting a leading input tree of $\Pi^*$ in order to receive an embedded tree that is necessary to obtain a particular hybridization network with minimum hybridization number at the end. In the following proof, however, we will show that if such a network exists, this network is also contained in the set of hybridization networks computed for a different ordering of the input trees $\Pi$.

**Lemma 2.** Given a set of input trees $\mathcal{T} = \{T_1, \ldots, T_n\}$ the algorithm allH-Networks computes the set of all hybridization networks $\mathcal{N}'$ by just considering maximum acyclic agreement forests of an input tree $T_i$ and an embedded tree $T'$ of a network in $\mathcal{N}$ for all possible orderings of $\mathcal{T}$.

**Proof.** To establish the proof we will discuss why the algorithm allH-Networks has not to consider non maximum agreement forests leading to hybridization networks with minimum hybridization number.

We will first show by an induction over $n = |\mathcal{T}|$, the number of input trees, that, if for a specific ordering of the input trees $\Pi$ a hybridization network $N^*$ can only be computed by applying a non maximum acyclic agreement forest, then, in this case, there exists another different ordering $\Pi^*$ computing $N^*$ by only taking components of maximum acyclic agreement forests for each input tree into account.

**Start.** The assumption, obviously, holds for $n = 1$. For $n = 2$ an agreement forest that is not maximal cannot lead to a hybridization network, since each maximum acyclic agreement forest $F$ would cause a network of lower hybridization number. This is the case, since the algorithm inserts a hybridization edge for exactly each component of an agreement forest, except $F_\rho$, and, thus, in this simple case, the hybridization number simply equals $|F| - 1$. Note that, due to Lemma 1 in this case, all hybridization networks are computed.

**Step.** Now, let $\Pi^* = (T_1, \ldots, T_i, \ldots, T_n)$, with $n > 2$, be an ordering of input trees. Assuming the algorithm allH-Networks computes all hybridization networks for the ordering $(T_1, \ldots, T_i, \ldots, T_{n-1})$ and there exists a specific,
so-called hidden, hybridization network \( N^* \) for \( \Pi^* \) that could only be computed by inserting hybridization edges for a non maximum acyclic agreement forest corresponding to a certain input tree \( T_i \) (1 \( \leq i < n \)) and a certain embedded tree \( T_i^* \) of the network \( N_{i-1}^* \) displaying \( T_1, \ldots, T_i-1 \). In this case, however, you can show that this network \( N^* \) is also computed by applying the algorithm to the ordering \( \Pi = (T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n, T_i) \) where \( T_i \) is added in a final step.

We will show that, by applying the algorithm \textsc{allHNnetworks} to \( \Pi \), there exists an embedded tree \( T'_n \) that enables an insertion of \( T_n \) into a certain network \( N_{n-2} \) such that each tree, except the last tree \( T_i \), is displayed in the resulting network \( N_{n-1} \) by the same set of edges as in \( N^* \). Then, due to Lemma 4, it is possible to insert \( T_i \) to \( N_{n-1} \) such that the resulting network \( N \) equals \( N^* \).

Now, let \( N_{n-2} \) be a hybridization network displaying each tree except \( T_i \) and \( T_n \) in the same topological way as it is the case for \( N^* \). In detail, \( N_{n-2} \) is isomorphic to \( N^* \) where, first, each edge that is not necessary for displaying \( T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n-1 \) is deleted and, second, all resulting isolated nodes are removed. Note that such a network \( N_{n-2} \) has to exist, since, by assumption, the algorithm applied to \( \Pi^* \) is able to compute all hybridization networks embedding \( T_1, \ldots, T_{n-1} \). Now, given \( N_{n-2} \), we can derive \( T'_n \) as follows.

Let \( T'_n \) be the embedded tree that is chosen for the insertion of \( T_n \) into \( N_{n-1}^* \) such that we receive the hidden hybridization network \( N^* \). This tree \( T'_n \) is partially displayed in \( N_{n-2} \) what means that we can transfer some hybridization edges from \( N^* \) to \( N_{n-2} \) that are necessary for displaying \( T'_n \) as follows. First, we mark these hybridization edges in \( N^* \) that are necessary for displaying \( T'_n \) and, second, we delete those hybridization edges that are not necessary for displaying the trees \( T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n-1, T_n \), which are exactly those hybridization edges that are only necessary for displaying \( T_i \) and \( T_n \) and not for any other input trees. Finally, after removing all isolated nodes we receive a network topologically isomorphic to \( N_{n-2} \) — simply due to its definition. Now, the remaining set of marked edges in this reduced network simply corresponds to a certain edge set which is denoted by \( E_+ \) in the following. Since those edges that are not in \( E_+ \) could be added by choosing certain edges corresponding to the embedded tree \( T'_n \), the tree \( T'_n \), we are looking for, is displayed in \( N_{n-2} \) by two different kind of edges that are, either, part of \( E_+ \) (denoted by \textit{Type 1}) or necessary for displaying \( T'_n \) (denoted by \textit{Type 2}). We can mark the embedding of \( T'_n \) in \( N_{n-2} \) as follows: Walking up from each leaf to the root we either arrive at nodes of indegree one or at nodes with larger indegree. In the first case we simply mark its only in-edge. Otherwise, in the latter case, we mark the in-edge of \textit{Type 1} if it exists. Otherwise, if such an edge does not exist, we mark the in-edge that is of \textit{Type 2}. Note that one of those edge-types always has to exist due to the following observation.

If the last selected hybridization edge corresponds to \textit{Type 2} the next hybridization node, again, must provide an edge of \textit{Type 2}, since \( T'_n \) is displayed in \( N_{n-1}^* \) because, just by assumption, the algorithm computes the whole set of networks displaying \( T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_n-1 \). Otherwise, if the last selected
hybridization edge is of Type 1 the next hybridization node must not necessarily again provide such an edge. If this is the case, however, there must exist an edge of Type 2, since the only reason why there does not exist an edge of Type 1 is because this missing edge is also necessary for displaying \( T_i \) in \( N^* \) and, thus, this node must be necessary for displaying \( T_i^* \) implying that it has to provide an in-edge of Type 2.

Generally speaking, each path, as described above, tries to follow those edges that are necessary for displaying \( T_i^* \). If a path gets stuck, since some parts for displaying \( T_i^* \) in \( N_{n-2} \) are missing because those parts are necessary for only displaying \( T_i \) and not for any other so far added input trees, it follows an indirection consisting of those edges necessary for displaying \( T_i^* \). As a consequence of these indirections, those hybridization edges are added that could only be inserted to \( N_{n-1}^* \) for a non maximum acyclic agreement forest of \( T_i \) and \( T_i^* \) and that are necessary for enabling an insertion of \( T_n \) such that \( N^* \) is computed.

Now, by inserting \( T_n \) to \( N_{n-2} \) by choosing \( T_n^* \) as embedded tree, one of the resulting networks equals \( N_{n-1} \) displaying \( T_n \) with the same set of hybridization edges as in \( N^* \). In detail, the network \( N_{n-1} \) is isomorphic to \( N^* \) where, first, each hybridization edge that is necessary for displaying \( T_i \) is removed and, second, each isolated node is deleted. Note that the acyclic agreement forest of \( T_n \) and \( T_n^* \), has to be maximal, since the insertion of \( T_i \) into \( N_{n-1}^* \) for receiving \( N^* \) is independent from those hybridization edges necessary for displaying \( T_n \). This implies that if we would add a non maximum acyclic agreement forest in this case, we would receive a network not isomorphic to \( N^* \). Moreover, due to this independence and due to Lemma 1, \( T_i \) can be added to \( N_{n-1} \) such that we finally receive \( N^* \).

As a consequence of the induction above, if for a certain ordering of the input trees there exists a certain tree \( T_i \) that has to be added by a non maximum acyclic agreement forest in order to enable a specific insertion of another input tree \( T_j \) (\( i < j \)) which is necessary for the computation of a particular hybridization network with minimum hybridization number, then, in this case, we can compute this network by simply applying the algorithm \textsc{AllHNetworks} to an ordering where \( T_i \) is added after \( T_j \). Thus, for each hybridization network of minimum hybridization number that could be computed by our algorithm only by applying non maximum acyclic agreement forests, there exists a certain ordering of the input trees such that our algorithm is able to compute this network. Finally, as a direct consequence, since our algorithm takes all possible orderings of input trees into account, our algorithm obviously guarantees the computation of all such hidden hybridization networks with minimum hybridization number without taking non maximum acyclic agreement forests into account. Thus, together with the following Lemma the correctness of Lemma 2 is established. \( \square \)

Due to Lemma 2 our algorithm computes all hybridization networks containing an acyclic agreement forest \( F \) of each input tree \( T \in \mathcal{T} \) and a specific embedded tree \( T' \). Now, to finish the proof, it remains to show that this set of hybridization networks is not just a subset of all hybridization networks which
Fig. 2. A short example demonstrating the need for two different input tree orderings: By running the algorithm \textsc{AllHNetworks} for the ordering $\Pi_1 = (\text{Tree 0, Tree 1, Tree 2})$ only those networks with hybridization number two, as the one denoted by \texttt{Hybridization Network 2}, are computed providing a hybridization node whose subtree consists of taxon 4. This is the case, since the only maximum acyclic agreement forest of \text{Tree 0} and \text{Tree 1} is of size two containing the component consisting of the single taxon 4. To compute the network denoted as \texttt{Hybridization Network 4} at bottom right, you have to apply the algorithm to the ordering $\Pi_2 = (\text{Tree 0, Tree 2, Tree 1})$, since now, in a first step, by adding \text{Tree 2} to \text{Tree 0} the network denoted as \texttt{Hybridization Network 0} at bottom left is computed. Based on this network you can select an embedded tree $T'$ by choosing its blue in-edge. As a direct consequence, the only maximum acyclic agreement forest of $T'$ and \text{Tree 2} is of size two containing the component consisting of the single taxon 3 and, thus, by adding this component to the network \texttt{Hybridization Network 0}, finally, the network \texttt{Hybridization Network 4} is computed. Note that, regarding the first mentioned ordering $\Pi_1$, this network could only be computed by our algorithm by taking the \textit{non} maximum acyclic agreement forest of \text{Tree 0} and \text{Tree 1} of size three, containing the two components consisting of the single taxa 3 and 6, into account.
means that for each hybridization network $N$, displaying a set of trees $\mathcal{T}$, we can find an embedded tree $T'$ of $N$ such that the acyclic agreement forest of $T'$ and $T \in \mathcal{T}$ is contained in $N$. Note that, if this is the case, with respect to Lemma 2, there always exists a certain ordering of $\mathcal{T}$ such that $N$ can be computed by a sequence of maximum acyclic agreement forests of an input tree and a particular embedded tree in $N$.

**Lemma 3.** Given a hybridization network $N$ displaying a set of trees $\mathcal{T}$, for each $T \in \mathcal{T}$ there exists an embedded tree $T'$ of $N$ such that an acyclic agreement forest of $T$ and $T'$ is contained in $N$.

**Proof.** To prove this Lemma, we show how such an embedded tree can be simply constructed from any given hybridization network $N$ and a given input tree $T$ in $\mathcal{T}$. For each leaf, we compute a certain path up to the root. Each time we arrive at a hybridization node, we either have already marked an incoming edge or we mark the incoming edge which is not necessary for displaying $T$ in $N$. Now, by deleting each edge from $N$ that is either not marked or not necessary for displaying $T$, we receive a bicontracting network $N'$. Finally, the acyclic agreement forest of $T$ and $T'$ can be simply computed by deleting all remaining hybridization edges in $N'$. Note that the ordering of $\mathcal{T}$ does not influence the existence of such an acyclic agreement forest but only its minimum size that has to be considered on the computational path leading to $N$. \(\square\)

## 6 Running Time of allHNetworks

To analyze the theoretical running time of the presented algorithm allHNetworks we have to discuss the complexity of three major steps involving the computation of embedded trees, the computation of all maximum acyclic agreement forests of size $k$, and the computation of all possible hybridization edges which can be added for a given maximum acyclic agreement forest. Each of those major steps has to be applied sequentially for each insertion of an input tree within the given ordering for a certain set of so far computed networks. At the beginning, when adding the second input tree, this set of networks only consists of the first tree of the ordering. However, as shown in the upcoming part, the number of the so far computed networks grows exponentially in the number of input trees. Thus, to cope with this huge computational effort, we have worked out some speed-ups concerning the algorithm itself as well as its execution on the provided system (cf. Sec. 6.2).
6.1 Theoretical Running Time

**Theorem 1.** The theoretical running time of the algorithm ALLHNetworks for computing all hybridization networks with hybridization number $k$ is

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

where $E$ denotes the edge set, $V$ the node set, and $X$ the taxon set of a resulting network.

**Proof.** To show the correctness of Theorem 1 we divide the estimation of its theoretical running time into the following four parts and discuss each of those parts separately:

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

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

- **Part C.** The number of all maximum acyclic agreement forests of size $k$ for two input trees, $T_1$ and $T_2$, can be estimated by $O\left(\binom{|E|}{k}\right)$, where $E'$, with $|E'| < |E|$, simply denotes the edge set of either $T_1$ or $T_2$. Nevertheless, in practice, this number is clearly lower since, in general, less than $k$ hybridization events, say $r$, are necessary for the insertion of one of the input trees and, usually, only a few number of all these possible $\binom{|E|}{r}$ set of components fulfills the definition of an acyclic agreement forest.

  Given an agreement forest of size $k$, there exist at most $k!$ acyclic orderings. Note that, as for the number of all maximum acyclic agreement forests, there exist, in general, clearly less orderings. This number, however, can be large if there are a lot of components consisting just of a single taxon. The running time for the computation of those maximum acyclic agreement forests is indicated in [13, Theorem 3] by $O(3^{|X|})$, where $X$ denotes the taxon set of an input tree.

- **Part D.** As already mentioned during the presentation of the algorithm ALLHNetworks, there is usually more than one possible hybridization edge that
can be added for a component of a maximum acyclic agreement forest of size less or equal $k$ to a so far computed network $N$. This number is, obviously, bounded by $\binom{|V|}{2}$ where $V$ denotes its set of nodes corresponding to $N$. In practice, however, this number is, in general, clearly lower, since, obviously, only a small fraction of all possible node pairs enables a correct embedding of the input tree. Given source and target nodes, an edge can, apparently, be simply added by only performing a constant number of basic tree operations. \hfill \square

\section{6.2 Speeding Up the Algorithm allHNetworks}

To handle the huge computational effort that is indicated in Theorem 1, it is very important to implement the algorithm in an efficient way. In this section we demonstrate how this can be done by, on the one hand, improving its execution on the provided system and, on the other hand, reducing the number of computational steps by modifying the algorithm as described in Section 4.

**Parallelization.** To improve the practical running time of our algorithm the search for hybridization networks of size $k$ can be run in parallel as follows: As described in Section 4, the adding of a tree $T_i$ of a specific ordering of the input trees to a so far computed network results in several new networks, which are then applied to the next input tree $T_{i+1}$. Since such adding steps run independently from each other, these steps can be parallelized in an easy way. Note that, based on the number of hybridization edges of a so far computed network, each of those steps is more or less likely to result in a hybridization network. Thus, you can, additionally, set up a priority queue to process the most promising networks first which, on the one hand, depends on the number of embedded input trees and, on the other hand, of its current hybridization number. Note that such a priority queue is only speeding up the computation of the hybridization number, since in this case the search can be aborted if the first hybridization network could be computed successfully. For the computation of all hybridization networks, however, each network has to be processed anyway until its hybridization number exceeds $k$ or all input trees could be added successfully.

**Applying reductions to the input trees.** To reduce the size of the input trees, we can apply two well known reduction steps before entering the algorithm, which in detail is the subtree reduction, following the work of Bordewich and Semple [6], and the cluster reduction, following the work of Baroni and Semple [5]. Whereas the subtree reduction just replaces all maximum common subtrees of all input trees, what is often only a small number of small subtrees, the cluster reduction cuts down the set of input trees into all minimum common clusters whose hybridization networks can be computed independently by running the presented algorithm for each of those clusters separately. Thus, a cluster reduction, usually, provokes a huge speed-up for the practical running time, since it can divide a big problem into several small problems that can be solved efficiently on its own.

**Reducing the number of acyclic orderings.** Applying the algorithm to two different acyclic orderings $\Pi_1$ and $\Pi_2$, each based on the same acyclic agreement forest of an embedded tree $T'$ and an input tree $T_i$, can result in two
identical set of networks. Otherwise, the two acyclic orderings must contain two components, $F_i$ and $F_j$, that are neighbors in $T_i$ and fulfill a certain property in terms of their position within the two acyclic orderings $\Pi_1$ and $\Pi_2$: Let $i_1$ and $i_2$ be the position of $F_i$ and $j_1$ and $j_2$ be the position of $F_j$ in $\Pi_1$ and $\Pi_2$, respectively. Then, in this case, either $i_1 > j_1$ and $i_2 < j_2$ or $i_1 < j_1$ and $i_2 > j_2$ must hold. Now, based on this observation, one can reject those acyclic orderings generating identical set of hybridization networks by simply taking the positions of the neighboring components in terms of the input tree $T_1$ into account.

7 Discussion

To analyze hybridization events it is of high interest to compute all topologically different hybridization networks with minimum hybridization number, since the more often an event occurs in all those networks the more likely it may be part of the true underlying evolutionary scenario. In this paper, we have, first, presented the algorithmic framework of the algorithm allHNetworks, which is able to compute all those networks for multiple rooted binary input trees, and, second, proved its correctness. Its theoretical running time, however, reveals that the number of possible networks growths in a strong exponential manner in terms of the number and the size of the input trees. As a consequence, it is very important to implement the algorithm in an efficient way what is demonstrated in the forthcoming paper of Albrecht [3]. A major problem, however, that cannot be solved neither by algorithmic nor technical speed-ups, is the exponential size of the solution space. Thus, we absolutely agree with the publication of Kelk et al. [11] suggesting to introduce a general method that enables the generation of certain constraints limiting the set of possible solutions.
Bibliography

[1] Albrecht, B., Scornavacca, C., Cenci, A., Huson, D. H. (2011) Fast computation of minimum hybridization networks. Bioinformatics, 28(2), 191–197.

[2] Albrecht, B. (2011) Fast Computation of Hybridization Networks. Master’s thesis, University of Tübingen, Mathematisch-Naturwissenschaftliche Fakultät, Tübingen, Germany.

[3] Albrecht, B. (2014) Computing all hybridization networks for multiple binary phylogenetic input trees. Submitted to Bioinformatics.

[4] Bordewich, M., Semple, C. (2007) Computing the minimum number of hybridization events for a consistent evolutionary history. Discrete Appl. Math., 155, 914-928.

[5] Baroni, M., Semple, C. (2006) Hybrids in real time. Syst. Biol., 55, 46–56.

[6] Bordewich, M., Semple, C. (2005) On the computational complexity of the rooted subtree prune and regraft distance. Ann. Combinator., 8, 409–423.

[7] Chen, Z.-Z. and Wang, L. (2010) HybridNet: a tool for constructing hybridization networks. Bioinformatics, 26, 2912–1913.

[8] Huson, D. H., Rupp, R., Scornavacca, C. (2011) Phylogenetic networks: Concepts, Algorithm and Applications. Cambridge University Press.

[9] Huson, D. H., Scornavacca, C. (2012) Dendroscope 3: An interactive tool for rooted phylogenetic trees and networks. Syst. Biol., accepted, doi:10.1093/sysbio/sys062.

[10] van Iersel, L., Linz, S. (2013) A quadratic kernel for computing the hybridization number of multiple trees. Inform. Process. Lett. 113, 318-323.

[11] Kelk, S., Linz, S., Morrison, D.A. (2013) Fighting network space; it is time for an SQL-type language to filter phylogenetic networks. arXiv:1310.6844

[12] Mallet, J. (2007) Hybrid speciation. Nature, 446, 279-283.

[13] Scornavacca, C., Linz, S., Albrecht, B. (2012) A first step towards computing all hybridization networks for two rooted binary phylogenetic trees. J. of Comput. Biol., 19(11), 1227–1242.

[14] Whidden, C., Zeh, N. (2010) Fast FPT algorithms for computing rooted agreement forests: theory and experiments. In Festa, P. (ed.) Proceedings of the Symposium on Experiment Algorithms, vol. 6049 of Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 141–153.

[15] Whidden, C., Beiko, R.G., Zeh, N. (2011) Fixed-Parameter and Approximation Algorithms for Maximum Agreement Forests. [arXiv:1108.2664]

[16] Wu, Y. (2010) Close Lower and Upper Bounds for the Minimum Reticulate Network of Multiple Phylogenetic Trees. Bioinformatics, 26(12), i140–i148.

[17] Wu, Y. (2013) An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. J. Comput. Biol., 20(10), 792–804.